Entwicklung und Erprobung neuer Messgeräte und Methoden für die rationelle Optimierung von neuen Elektrolyten für Lithium - Ionen-Batterien

Dissertation
zur Erlangung des Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)
der
Naturwissenschaftlichen Fakultät IV
Chemie und Pharmazie
der Universität Regensburg

vorgelegt von
Hans-Georg Schweiger
Regensburg 2004

Promotionsgesuch eingereicht am:	5. 11. 2004
Tag des Kolloquiums:	15. 12. 2004
Diese Arbeit wurde angeleitet von:	PD Dr. H. J. Gores
Prüfungsausschuss:	Prof. Dr. Dr. h.c. J. Barthel, Vorsitzender
	PD Dr. H. J. Gores
	Prof. Dr. W. Kunz
	Prof. Dr. N. Korber

Meinen Eltern

und

Geschwistern

Danksagung

Die vorliegende Arbeit entstand in der Zeit von September 2002 bis November 2004 am Lehrstuhl für Chemie IV – Physikalische Chemie (Solution Chemistry) – der naturwissenschaftlichen Fakultät IV – Chemie und Pharmazie – der Universität Regensburg.

Mein ganz besonderer Dank gilt meinem Dokorvater Herrn Privatdozent Dr. Heiner Jakob Gores für die Bereitstellung des interessanten Themas. Er war für mich in dieser Zeit nicht nur ein kompetenter Ratgeber, sondern der Doktor-„Vater" schlechthin. Seine ausgezeichnete Betreuung sowie die gemeinschaftlichen und sportlichen Aktivitäten und seine Gastfreundschaft trugen zu einem besonders harmonischen Klima bei. Sein ständiger und unermüdlicher Einsatz für die Balange „seiner Doktoranden", seine fachliche Unterstützung in allen theoretischen und praktischen Fragen sowie bei der Mittelbeschaffung waren ein bedeutender Faktor für den Erfolg und das Gelingen dieser Arbeit.

Herrn Prof. Dr. Werner Kunz und allen Mitarbeitern des Lehrstuhls danke ich für die angenehme und konstruktive Zusammenarbeit sowie Frau Dr. E. Schnell für ihre Unterstützung.

Dem Projektpartner dieser Arbeit, der Gaia Akkumulatorenwerke GmbH, Nordhausen, danke ich für die Finanzierung dieses Projekts. Der Firma Chemetall (Frankfurt am Main) möchte ich für die Finanzierung der Entwicklung der neuen Methode zur Bestimmung von Wasserspuren danken.

Meinen Laborkollegen, Herrn Dr. H. Bruglachner, Herrn Dr. M. Eberwein, Herrn Dipl. Chem. T. Herzig, Herrn Dipl. Chem. J. Lodermeyer, Herrn Dr. S. Jordan, Herrn Dipl. Chem. (FH) E. Riedl, Dr. J. de Roche und besonders Herrn Dipl. Chem. M. Multerer danke ich für die gute und freundschaftliche Atmosphäre in der Arbeitsgruppe und ihre stete Hilfsbereitschaft.

Besonders möchte ich mich auch bei meinen beiden Schwerpunktspraktikanten Herrn P. Wachter und Herrn F. Wudy für die geduldige und sorgfältige Durchführung der Messungen mit der Phasendiagrammanlage und für die Auswertung dieser Messungen danken.

Herrn Dipl. Ing. (F.H.) A. Engelhardt danke ich für die Durchführung von Konstruktionsaufgaben. Weiterhin gilt mein Dank den Mitarbeitern der Elektronik-, Glasbläser- und der Mechanikwerkstätten der Fakultät für Chemie und Pharmazie für die zügige Ausführung der Aufträge und für ihre Hilfestellungen.

Nicht vergessen möchte ich unseren unersetzlichen Lageristen Herrn W. Simon für seine stete Hilfsbereitschaft.

Herrn Dipl. Ing. (FH) M. Spannbauer möchte ich für die Elektronikkenntnisse danken, die er mir im Rahmen früherer Projekte vermittelt hat, ohne die die neuentwickelten Geräte nicht verwirklicht hätten werden können.

Meinen Eltern möchte ich für die Unterstützung während meines Studiums danken.

Inhaltsverzeichnis

1 Motivation

Werden Lithium-Ionen-Batterien in der Luft- und Raumfahrt oder im Automobilbereich, wie zum Beispiel beim Hybridantrieb oder im 42V Bordnetz eingesetzt, so werden an sie wesentlich höhere Anforderungen gestellt, als dies bei Akkumulatoren aus dem Consumerbereich, zum Beispiel Batterien für Handys, der Fall ist. Der weite Temperaturbereich, in dem diese Batterien verwendet werden sollen, erhöht die ohnehin großen Herausforderungen noch weiter.

1.1 Stand der Technik

In Abbildung 1 ist der typische Aufbau einer Lithium-Ionen-Batterie dargestellt. Als Anodenmaterial wird Kohlenstoff eingesetzt. Dieser wird mit einem Binder, zum Beispiel PVdF[1], auf den Stromableiter aus Kupferfolie aufgebracht. Für das Kathodenmaterial werden Lithium-Übergangsmetalloxide oder Spinelle verwendet, die ihrerseits auf einem Stromableiter aus Aluminium fixiert sind. Die beiden Aktivmaterialien sind durch einen Separator voneinander getrennt, um einen elektrischen Kurzschluss zu vermeiden. Dieser Separator ist mit einem Elektrolyten getränkt, der für die Ionenleitung verantwortlich ist. Nach dem Stand der Technik [1] besteht dieser Elektrolyt aus einem Leitsalz wie $LiPF_6$, das in Mischungen aus organischen Carbonaten gelöst wird.

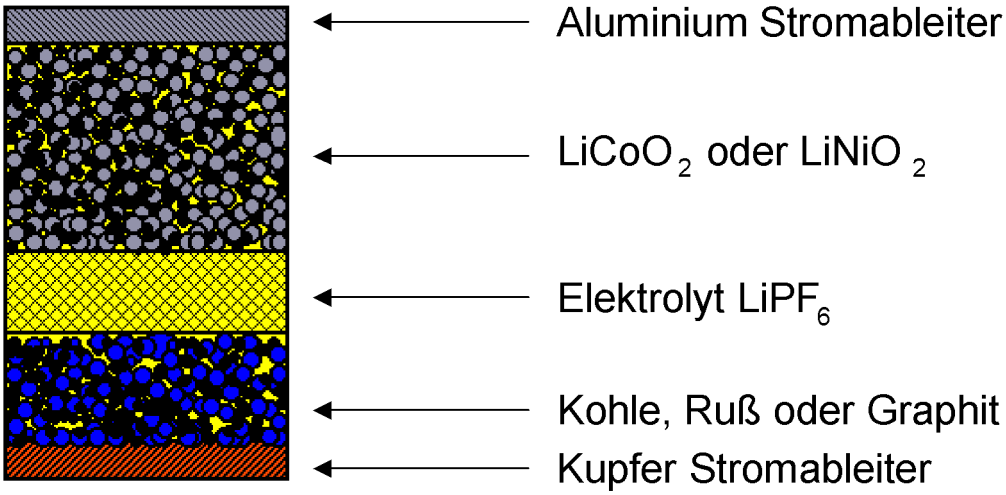


Abbildung 1 Aufbau einer Lithium-Ionen-Batterie [2]

Beim Entladevorgang werden Lithium-Ionen aus dem Wirtsgitter des Anodenmaterials in das Kathodematerial hineintransportiert, wobei gleichzeitig Elektronen von der Anode zur Kathode fließen. Beim Ladevorgang läuft der Prozess umgekehrt ab.

$$Li_{x+dx}M_yO_{2y} + Li_{dx}C_6 \rightleftharpoons Li_{dx}M_yO_{2y} + Li_{x+dx}C_6 \quad (1)$$

[1] Polyvinylidenfluorid

In Gleichung (1) steht M für das Metall Co, Ni oder Mn und es gilt: x+dx =1, dx<<1, y = 1 (Co, Ni), bei Mn 2. Aufgrund der unterschiedlichen chemischen Potenziale der Lithium-Ionen in den Elektrodenmaterialen besteht eine Potenzialdifferenz zwischen den Elektroden, die als Spannung an den Ableitern abgegriffen werden kann. Diese Spannung erreicht rund 4 V, wenn das Metall in Gl. (1) Co oder Ni ist, da das Potenzial der Anode dann im Bereich von (0,1 - 0,5 V vs. Li/Li^+) [3] und das der Kathode im Bereich von (3,0 – 4,3 V vs. Li/Li^+ liegt) [4]. Wird statt Li_xCoO_2 oder Li_xNiO_2 und deren Mischungen und Li_xC_6 eine andere Kombination für die aktiven Elektrodenmaterialien verwendet, kann diese Spannung auch niedriger sein, z. B. erhält man bei Li_xTiS_2 statt Li_xC_6 mit den genannten Kathodenmaterialien nur rund 2 V.
Die dem Stand der Technik entsprechenden Batterien auf der Basis von Elektrolyten mit fluorhaltigen Lithiumsalzen weisen eine Reihe von Nachteilen auf:

- Hydrolyse des $LiPF_6$ zu PF_5, und HF sowie ihren Folgeprodukten [5],[6],[7] durch Wasserspuren,
- Reaktion der Hydrolyseprodukte mit eingesetzten Komponenten [5],[6] und den Stromableitern (Cu, Al),
- geringe thermische Stabilität [7],
- hohe Kosten des Leitsalzes,
- Toxizität der Hydrolyseprodukte[7],
- $LiPF_6$ und $LiBF_4$ können nicht mit Manganspinell [7] eingesetzt werden, der ökologisch unbedenklicher und wesentlich wirtschaftlicher wäre (Mn 0,55$/lb [8]),
- hohe Kosten der $LiCoO_2$- oder $LiNiO_2$-Kathodenmaterialien,
  Co 22 US$/lb, Ni 6,2 $/lb [9],
- geringe Umweltverträglichkeit der $LiCoO_2$- oder $LiNiO_2$-Elektroden und des Leitsalzes.

1.2 Zielsetzung

Um diese Probleme zu vermeiden, wurde von Barthel et al. die Klasse der Lithiumborate eingeführt [10] bis [13]. Diese Arbeitsgruppe hat gezeigt, dass man Bor tetraedrisch mit einer Vielzahl von zweizähnigen Liganden verknüpfen kann, die –OH, --CO(OH) oder $-SO_3H$ und deren Kombinationen enthalten.
Ein Vertreter dieser Salze, das Borat mit Oxalsäure, das mit zweimal zwei CO(OH)-Gruppen koordiniert ist, kann einfach und preiswert hergestellt werden; so ist Lithium-bis[oxalato(2-)]borat(1-) (LiBOB), siehe Abbildung 2 fast zeitgleich von Wietelmann [1], Angell [14] und einem Mitglied unserer Arbeitsgruppe, Eberwein [15], als Leitsalz für Lithium-Ionen-Batterien vorgeschlagen worden.

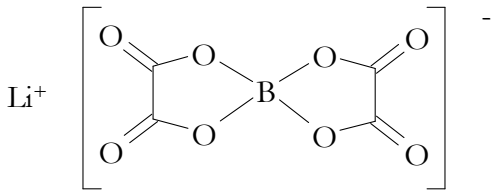


Abbildung 2 Lithium-bis[oxalato(2-)]borat(1-)

Dieses Salz zeichnet sich gegenüber den bisher verwendeten Leitsalzen durch eine Reihe von Eigenschaften aus, durch die es sich für den technischen Einsatz sehr gut eignet:

- höhere Sicherheit aufgrund großer thermischer Stabilität [16],
- enthält kein Fluor, daher höhere Sicherheit [5], [17] und Umweltverträglichkeit,
- geringe Korrosion an Aluminiumstromableitern [18],
- sehr gute SEI-Bildung am Anodenmaterial [19],
- Eignung für den Einsatz bei hohen Temperaturen (>40°C) [16], [17],
- schwach koordinierend [20], daher relativ hohe Leitfähigkeit in Standardlösungsmitteln[18],
- geringe Hydrolyseneigung [5],
- geringe Molmasse, daher hohe Energie- und Leistungsdichte [14],
- geringe Herstellungskosten,
- Kompatibilität mit Manganspinell als Kathodenmaterial [16].

Durch die Kombination dieser positiven Eigenschaften [20] stellt dieses Salz einen sehr guten Ersatz für das bisher eingesetzte $LiPF_6$ dar. Da aber die Zusammensetzungen der bisherigen Elektrolytlösungen für Salze wie $LiPF_6$ ausgelegt worden sind, muss man die Elektrolytzusammensetzungen für LiBOB erneut optimieren, damit es in Lithium-Ionen-Batterien erfolgreich eingesetzt werden kann, wie Brodd et al. dieses Jahr in ihrem Review-Artikel feststellten:

„*This salt requires reformulation of the solvent composition for good performance*" [21]

Das Hauptziel der hier vorgelegten Arbeit war, Elektrolytlösungen, die LiBOB enthalten, für den technischen Einsatz in Lithium-Ionen-Zellen zu optimieren. Da für diese Optimierungen eine große Anzahl verschiedenster Experimente durchzuführen waren, war ein weiteres Ziel, Geräte und Methoden zu entwickeln, mit denen es möglich ist, eine große Anzahl an Versuchen schneller durchzuführen und Verfahren zu finden, die es erlauben, die Anzahl der notwendigen Versuche zu reduzieren. Zusätzlich mussten einige weitere Aufgaben erledigt werden, darunter die effektive Reinigung des Salzes, die keineswegs trivial ist, und die Bestimmung seines Wassergehalts. Zur Reinigung des Salzes sei betont, dass die Reinigung von Batterieelektrolyten wie $LiBF_4$ und $LiPF_6$, bei der viele Firmen beteiligt waren, jeweils rund ein Jahrzehnt in Anspruch nahm, wie auch die vielen hierzu vorliegenden Patente zeigen.
Die folgende Zusammenstellung kann als Wegweiser durch diese umfangreiche Arbeit benutzt werden. Außerdem stehen dafür die Gliederung der Arbeit, das fein unterteilte Inhaltsverzeichnis sowie die Zusammenfassung, in der die Höhepunkte der Arbeit herausgestellt werden, zur Verfügung. Alle nicht unmittelbar für das Verständnis der Arbeit notwendigen Teile wurden als 400 seitiger Anhang beigefügt.

In **Kapitel 2** wird eine neuentwickelte Anlage beschrieben, mit der es möglich ist, eine große Anzahl an Elektrolytlösungen simultan zu untersuchen. Die Ergebnisse, die man mit dieser Anlage erhält, werden dabei weit weniger durch eine Unterkühlung verfälscht als dies bei DTA- oder DSC-Messungen der Fall ist. Neben der Untersuchung der Abhängigkeit der Leitfähigkeit von der Temperatur, kann diese Anlage auch zur Bestimmung der Flüssigkeitsbereiche von Elektrolytlösungen und zur Erstellung von Phasendiagrammen eingesetzt werden. Ein weiteres Anwendungsgebiet dieser Anlage ist die Untersuchung von Hydrolysekinetiken. In diesem eher technisch ausgerichte-

ten Kapitel wird die benötigte Theorie, der elektronische Aufbau der Messanlage und ihre Steuersoftware beschrieben.

Kapitel 3 beschäftigt sich mit den Messungen der Schmelzpunkte von Reinstoffen und mit der Bestimmung der Phasendiagramme von Lösungsmittelmischungen, die für Lithium-Ionen-Batterien technisch relevant sind. Für diese Messungen wurde die in Kapitel 3 beschriebene Anlage eingesetzt, da mit ihr ein wesentlich höherer Probendurchsatz als beispielweise bei DTA-Messungen erzielt werden kann. Kern der Arbeiten sollte die Reduktion der Unterkühlung sein, welche die Auswertung der Daten erheblich stört; dazu sollten verschiedene Verfahren eingesetzt und untersucht werden. Auch darüber wird in diesem Kapitel berichtet.

Kapitel 4 ist eine Zusammenfassung der Ergebnisse zur Optimierung von Elektrolytlösungen, mit denen es möglich ist, den nutzbaren Temperaturbereich für Li-Ionen-Zellen deutlich zu erweitern. Da der Innenwiderstand einer Batterie die Hochstromfähigkeit und damit die Leistungsabgabe limitiert, ist es von besonderer Bedeutung, die Leitfähigkeit der Elektrolytlösung zu optimieren, insbesondere da diese mit fallender Temperatur exponentiell abnimmt. Um die Leitfähigkeit zu verbessern, sollten Multikomponenten-Lösungsmittelmischungen eingesetzt werden, für deren Optimierung eine große Anzahl an Messungen nötig gewesen wäre. Für diese Optimierungen sollen auch neuartige Lösungsmittelmischungen eingesetzt werden. Um dieses Vorhaben schnell und systematisch zu realisieren, sollte erstmals in diesem Zusammenhang die Simplexmethode eingesetzt werden, mit der eine drastische Reduzierung der Anzahl der Versuche erreicht werden sollte. Diese geometrische Methode wurde zwar bisher für eine große Zahl von Optimierungen, insbesondere in Wirtschaft und Technik eingesetzt, nicht aber in diesem Zusammenhang, in dem bisher trial-and–error, empirische sowie semiempirische Methoden vorherrschten [22].

In **Kapitel 5** wird ein neuentwickeltes Batterietestsystem beschrieben, mit dem Batterien galvanostatisch zyklisiert werden können. Diese Entwicklung wurde notwendig, weil zugesagte Mittel nicht rechtzeitig zur Verfügung standen. Das Batterietestsystem zeichnet sich durch einen modularen Aufbau aus, so dass es leicht mit verschiedenen Sensoren erweitert werden kann.

In **Kapitel 6** werden Untersuchungen über die Stabilität von Lithium-Ionen-Batterien im Zusammenhang mit dem Einsatz bei hohen Temperaturen und hohen Belastungen wiedergegeben. Bei diesen Versuchen sollte erstmalig eine Elektrolytlösung, die LiBOB als Leitsalz enthält, durch Zusatz von Additiven verbessert werden. Durch den Zusatz dieser Additive sollte eine geeignetere „soild-elektrolyte-interface“ (SEI) aufgebaut werden, mit dem Ziel, Lebensdauer und Ladungsausbeuten der zyklisierten Zellen zu steigern.

In **Kapitel 7** werden mehrere Ergebnisse der Untersuchungen von Verunreinigungen an LiBOB-Chargen zusammengefasst, wobei ein breites Spektrum an analytischen Methoden eingesetzt wurde. Wasser, das als Hauptverunreinigung in diesen Chargen gefunden wird, kann aufgrund von Nebenreaktionen mittels Karl-Fischer-Titration nicht quantitativ bestimmt werden. Daher sollte eine neue Methode entwickelt werden, die diese Nachteile nicht aufweist. Um den Wassergehalt der LiBOB-Chargen zu reduzieren, wurden mehrere Verfahren entwickelt, die in diesem Kapitel beschrieben werden.

In **Kapitel 8** werden alle Ergebnisse zusammengestellt, die bezogen auf den Stand der Literatur, als „neu“ anzusehen sind. Dieses kleine Abschlusskapitel gewährt einen schnellen Zugang zu allen wesentlichen Neuentwicklungen. Außerdem wird darin über bereits realisierte, in Arbeit befindliche und geplante Veröffentlichungen berichtet. Was Patente und Gebrauchsmuster angeht, beschränkt sich der Autor in diesem Zusammenhang naturgemäß auf bereits eingereichte Arbeiten.

Im **Anhang** werden die durchgeführten Versuche, die erhaltenen Ergebnisse sowie die eingesetzten Geräte und Verfahren detailliert beschrieben. Die Genauigkeit der neuentwickelten Geräte und Verfahren wird ebenfalls im Anhang anhand von Berechnungen und Versuchen dargestellt. Die Schaltpläne der entwickelten Geräte werden ebenso wie die Quelltexte der Steuersoftware aufgeführt, um eine zukünftige Weiterentwicklung und Verbreitung zu gewährleisten.

1.3 Literaturverzeichnis

[1] J-i Yamaki in W. A. van Schalkwijk und B. Scrosati (Hrsg.) *Advances in Lithium-Ion Batteries*, Kluwer/Plenum, New York, (2002)
[2] GAIA Akkumulatorenwerke GmbH. [Online]. http://www.gaia-akku-online.de
[3] M. Winter, J. O. Besenhard, in J. O. Besenhard (Hrsg.), *Handbook of Battery Materials,* Wiley-VCH, Berlin (1999)
[4] J. M. Tarascon, G. Amatucci, L. Klein, C. Schmutz, F. Shokoohi, *Prog. Batteries Battery Mater.*, **15**, 168-180 (1996)
[5] J.-C. Panitz, U. Wietelmann and M. Scholl, *Nonaqueous battery electrolytes containing organic liquids, lithium salts, and lithium bis(oxalato)borate*, DE 10111410, (2002).
[6] K. Xu, S. Zhang, T. R. Jow, W. u. Xu and C. A. Angell, *Electrochem. Solid-State Lett.*, **5**, A26 (2002)
[7] U. Lischka, U. Wietelmann and M. Wegner, *Lithium bis(oxalato)borate, method for its production and application*, DE 19829030, (1999)
[8] U.S. Geological Survey [Online]. http://minerals.usgs.gov/
[9] Kitco base metals [Online]. http://www.kitcometals.com
[10] J. Barthel, M. Wühr, R. Buestrich, und H. J. Gores, *J. Electrochem. Soc.,* **142,** 2527 (1995)
[11] J. Barthel, R. Buestrich, E. Carl und H. J. Gores, *J. Electrochem. Soc.,* **143,** 3565 (1996)
[12] J. Barthel, R. Buestrich, E. Carl und H. J. Gores, *J. Electrochem. Soc.,* **143,** 3572 (1996)
[13] J. Barthel, A. Schmid und H. J. Gores, *J. Electrochem. Soc.,* **147,** 21 (2000)
[14] W. Xu und C. A. Angell, *Electrochem. Solid-State Lett.*, **4**, E1 (2001)
[15] M. Eberwein, *Darstellung und elektrochemische Charakterisierung von oxidationsstabilen Lithiumboraten und Lithiumphosphaten für den Einsatz in Lithium-Ionen-Zellen, Diplomarbeit*, Regensburg (2000)
[16] J. C. Panitz, K. Schade, B. Schwanze, U. Wietelmann, S. Stenger, *Electrochemical cell for a lithium ion battery with improved high-temperature stability*, WO 2003075371, (2003)
[17] J. Jiang, und J. R. Dahn, *Electrochem. Solid-State Lett.,* **6,** A180 (2003)
[18] T. R. Jow, K. Xu, M. S. Ding, S. S. Zhang, J. L. Allen, und K. Amine, *J. Electrochem. Soc.,* **151,** A1702 (2004)
[19] K. Xu, S. Zhang, B. A. Poese und T. R. Jow, *Electrochem. Solid-State Lett.,* **5,** A259 (2002)
[20] W. Xu, A. J. Shusterman, R. M. Marzke und C. A. Angell, *Electrochem. Solid-State Lett.*, **151**, A632 (2004)
[21] R. J. Brodd, et al, *J. Electrochem. Soc.*, **151**, K1-K11 (2004)
[22] J. Barthel in J. O. Besenhard (Hrsg.), *Handbook of Battery Materials,* Wiley-VCH, Berlin (1999)

2 Entwicklung einer neuen Anlage zur Messung thermischer Eigenschaften von Elektrolytlösungen

2.1 Zielsetzung

Die in diesem Kapitel beschriebene Anlage wird für drei verschiedene Anwendungsbereiche eingesetzt. Die erste Anwendung stellt die Bestimmung der Temperaturabhängigkeit der Leitfähigkeit von Elektrolytlösungen dar. Beispiele hierfür sind in Kapitel 4.3 und 6.5 aufgeführt. Bisher wurden diese Messungen mit der in Kapitel 4.2.2 dargestellten Anlage durchgeführt. Dabei wurden die Leitfähigkeiten bei verschiedenen konstanten Temperaturen gemessen. Aus diesen Messwerten wurde dann durch Interpolation die Abhängigkeit der Leitfähigkeit von der Temperatur bestimmt. Da bei dieser Methode die Temperatureinstellung sehr viel Zeit in Anspruch nimmt und nur acht Messzellen gleichzeitig untersucht werden können, war es sinnvoll, eine Methode zu entwickeln, mit welcher derartige Untersuchungen deutlich schneller und anhand einer größeren Anzahl an Proben durchgeführt werden können.
Ein weiteres Anwendungsgebiet, bei dem dieses Anlage eingesetzt wird, ist die Untersuchung von Kinetiken mittels Leitfähigkeitsmessungen. Beispiele für solche Messungen sind Hydrolyseuntersuchungen, wie sie von Multerer [1] durchgeführt werden. Bis jetzt wurde die Messung dieser Kinetiken mit der in Kapitel 4.2.2 beschriebenen Anlage, oder ähnlichen auf Kohlrauschbrücken basierenden Anlagen durchgeführt. Da bei diesen Anlagen die Messung der Leitfähigkeiten durch manuellen Abgleich der Messbrücke erfolgt, kann daher immer nur eine Messung gleichzeitig erfolgen. Auch deshalb war es sinnvoll, die vorliegende Apparatur zu bauen, mit der es möglich ist, die Untersuchung von Leitfähigkeiten an einer großen Anzahl von Proben gleichzeitig und automatisch durchführen zu können. Durch die Automatisierung der Messmethode können auch sehr langsame Kinetiken untersucht werden. Das ist ein weiterer wichtiger Vorteil dieses Verfahrens.
Der dritte Einsatzbereich dieser neuen Anlage ist die Bestimmung von Phasendiagrammen. Diese und die Kenntnis des Flüssigkeitsbereichs von Elektrolytslösungen ist in der Entwicklung von Batterien, Superkondensatoren und anderen technischen elektrochemischen Zellen von großer Bedeutung. Normalerweise werden Phasendiagramme meist durch Abkühl- oder Aufheizexperimente bestimmt. Diese Untersuchungen wurden nach dem Stand der Technik mittels Differenz-Thermo-Analyse (DTA) oder Differential Scanning Calorimetry (DSC) durchgeführt. Diese Messungen sind stark durch Unterkühlungseffekte benachteiligt. Weitere Methoden und ihre Nachteile sind in Kapitel 2.2 beschrieben. Um diese Nachteile umgehen zu können und um einen im Gegensatz zu den in Kapitel 2.2 aufgeführten Methoden deutlich höheren Datendurchsatz zu erreichen, wird eine Anlage benötigt, mit der es möglich ist, an vielen Proben gleichzeitig Abkühl- und Aufheizkurven aufzunehmen. Die Änderung der Leitfähigkeit kann dabei bei leitfähigen Proben als zusätzliche Detektionsmöglichkeit für einen Phasenübergang verwendet werden. Ziel bei der Entwicklung dieser Anlage war es, ein Phasendiagramm einer binären Mischung im Laufe einer Woche zu bestimmen. Dabei soll eine statistische Absicherung der Ergebnisse durch eine große Anzahl an verschiedenen Zusammensetzungen und durch mehrmalige Wiederholung der Messung erreicht werden.

2.2 Grundlagen der Thermischen Analyse

Die Bestimmung invarianter (f=0) Punkte in Einstoffsystemen (Tripelpunkte) ist relativ einfach und kann mit großer Genauigkeit (Fehler < 1 mK) durchgeführt werden. Daher gründen sich auch viele Eichstandards auf Tripelpunkte, wie den des Wassers bei 273,16 K. Auch für univariante Punkte (f=1) gelingt dies bei Einstoffsystemen noch mit relativ hoher Genauigkeit. Für nicht zur Unterkühlung neigende reine Substanzen können mit der von Rossini und Mitarbeitern entwickelten Zelle (NBS) [5], die von Mathieu [6] und Hammer [7] zur Bestimmung des Schmelzpunktes von Acetonitril benutzt worden war, in günstigen Fällen Genauigkeiten von 1-3 mK erreicht werden. Die Mitarbeiter des National Bureau of Standards um Rossini bestimmten eine Vielzahl von Schmelzpunkten mit einem Fehler von ±10 mK [5].

In der letzten Zeit werden zunehmend DTA und DSC zur Untersuchung von Phasendiagrammen eingesetzt [8], [9] und [10]. Die Methoden sind zwar schnell und benötigen nur ein kleines Probenvolumen, sind aber von erheblich geringerer Genauigkeit [5] und wegen des wenig umfangreichen Probevolumens auch wesentlich anfälliger gegen systematische Fehler. Die reduzierte Genauigkeit ergibt sich schon allein durch die fehlende Durchmischung, der Stofftransport kann hier nur durch die langsame Diffusion erfolgen. Insbesondere durch Unterkühlungseffekte können die Messergebnisse, die mit diesen Methoden erhalten werden, stark verfälscht sein. Eine genaue Bestimmung des Phasenübergangs ist somit nur schwer möglich, da die Unterkühlungen bei Lösungsmitteln mehr als 10°C betragen können. Noch stärker ausgeprägt sind Unterkühlungen bei ionischen Fluiden die bei DTA/DSC Messungen bis zu 200 °C unterkühlen können [11]. Ein weiterer Nachteil dieser Methoden ist in der Verwendung von üblichen Probenmassen im Bereich von 1 bis 100 mg zu sehen. Bei diesen geringen Mengen machen sich Effekte, die durch die Gefäßwände verursacht werden, stark bemerkbar. Bei kleinen Mengen spielen schon geringste Mengen an Verunreinigungen eine große Rolle, die durch eine Erniedrigung des Schmelzpunkts oder durch eine Erhöhung des Siedepunkts die wahre Umwandlungstemperatur verfälschen. Sollen Phasenübergänge in Mischungen untersucht werden, so ist für eine gute Durchmischung zu sorgen, so dass die Zusammensetzung der Lösung konstant bleibt und damit der Gleichgewichtszusammensetzung entspricht. Herkömmliche DTA/DSC Geräte verfügen über keine solche Durchmischungsvorrichtung. Somit eignen sich diese Geräte nur bedingt für die Untersuchung von Mehrkomponentensystemen.

Eine Alternative zu den obigen Methoden stellt die von Andrews, Kohman and Johnston entwickelte Methode der Untersuchung von Abkühlkurven dar [12]. In der Literatur finden sich noch weitere Beispiele für diese Methode, wie zum Beispiel in [13] und [5]. Bei diesen Methoden wird eine Probe in ein Gefäß eingebracht und die Änderung der Temperatur innerhalb der Probe beobachtet, während die Probe langsam abgekühlt oder aufgeheizt wird. Mit diesem Verfahren ist eine merkliche Steigerung der Genauigkeit verbunden, da hier deutlich größere Probenmengen verwendet werden können. Auch können bei dieser Methode wesentlich langsamere Abkühl- bzw. Aufheizraten eingesetzt werden, da hier keine Reduktion der Empfindlichkeit mit abnehmenden Abkühl- bzw. Aufheizraten auftritt. Dadurch können Unterkühlungseffekte vermieden werden. Sollen aber Phasenübergänge beobachtet werden, bei denen nur geringe Energiemengen umgesetzt werden, so stößt diese Methode an ihre Grenzen, da sie im Vergleich zur DTA relativ unempfindlich ist. Eine Beobachtung von Schmelzvorgängen ist mit dieser Methode, ebenso bei der DTA/DSC, nur schwer möglich, da keine ausreichende Durchmischung einer festen Probe gegeben ist [23]. So schmilzt zum Beispiel ein Feststoff, der sich im Probengefäß befindet, erst vom Rand her, wohingegen der Temperaturfühler, der meist in der Mitte des Probengefäßes angebracht ist, noch vom

Feststoff umgeben ist. Daher sind die mit dieser Methode bestimmten Schmelzpunkte meist stark mit Fehlern behaftet.
Bei allen diesen Arbeiten werden die zu untersuchenden Substanzen einem zeitabhängigen Temperaturprofil unterworfen, die Veränderung der Temperatur in Abhängigkeit mit der Zeit wird aufgezeichnet. Tritt ein Phasenübergang auf, so erfolgt aufgrund der Umwandlungsenthalpie bei Reinstoffen ein Haltepunkt und bei Mehrkomponentensystemen ein Knickpunkt in den Temperaturzeitfunktionen, da sich bei diesen Systemen die Zusammensetzung laufend ändert, siehe auch Abbildung 3.

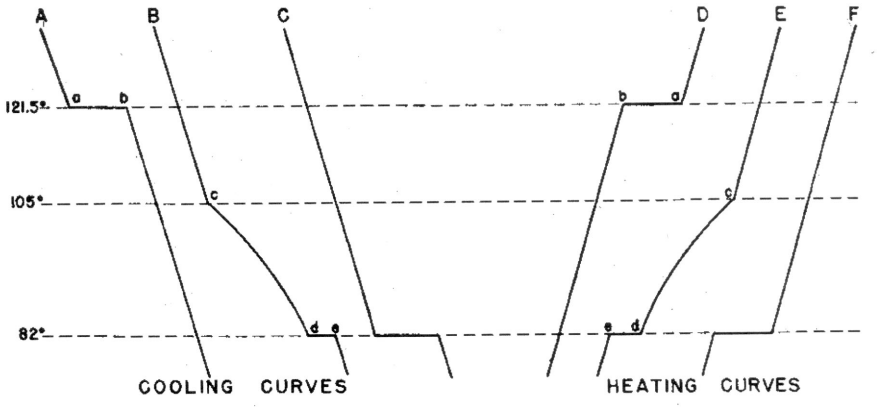


Abbildung 3 Beispiele von Abkühlungskurven bzw. Erwärmungskurven des Systems Benzoesäure/Zimtsäure [5]

Aus den mit diesen Methoden erhaltenen Daten können Schmelzdiagramme konstruiert werden. (siehe Kapitel 3.2). Für eine vorgegebene Mischung wird von der Schmelze ausgehend, die Temperatur-Zeit-Funktion der Probe aufgezeichnet. Hierbei treten drei typische Kurven auf:

a) Reine Substanz A (Kurve A in Abbildung 3.)
Die Temperatur bleibt nach Erreichen des Schmelzpunktes $T_{fus,A}$ konstant (invarianter Punkt, Haltepunkt), bis jegliche Substanz erstarrt ist. Die auftretende Erstarrungswärme kompensiert die Wärmeabgabe nach außen. Danach fällt die Temperatur kontinuierlich.

b) Beliebiges Gemisch x_A (Kurve B in Abbildung 3)
T fällt bis zum Punkt C kontinuierlich. Hier beginnt reines A auszufallen, so dass die Erstarrungswärme die Wärmeabgabe teilweise kompensiert und damit die Kurve flacher macht. Die Folge der geänderten Abkühlungsgeschwindigkeit ist ein Knickpunkt. Mit Erreichen der eutektischen Zusammensetzung kristallisiert die Schmelze vollständig aus (Haltepunkt).

c) Eutektisches Gemisch (Kurve C in Abbildung 3)
Mit Erreichen der Temperatur T_E kristallisiert die Mischung unter Ausbildung eines langandauernden Haltepunktes vollständig aus.
In allen anderen Fällen ist die Abkühlungsfunktion durch das exponentielle Newtonsche Abkühlungsgesetz gegeben; dessen Differentialform lautet:

$$\frac{dT}{dt} = k\ (T - T_{außen}) \tag{2}$$

Da bei dem Experiment leicht eine durch verzögerte Kristallisation verursachte Unterkühlung der Schmelze eintreten kann, ist es erforderlich, die gesuchten Punkte T_{A-B}, T_C bzw. T_{D-E} durch Extrapolation der waagerechten oder nahezu waagerechten Kurventeile zu erhalten. Insbesondere muss D als Schnittpunkt der beiden Kurvenzüge festgelegt werden.
Diese Unterkühlungen treten bei Aufheizkurven, die in Abbildung 3 auf der rechten Seite dargestellt sind, (Kurven D, E, F) nicht auf. Bei diesen Messungen stellt aber die Durchmischung der Proben ein schwierig zu lösendes Problem dar, weil Systeme gerührt werden müssen, die einen großen Anteil an Feststoff aufweisen. Erschwerend kommt hinzu, dass die Zusammensetzung des Feststoffes nicht homogen ist, und daher die Zusammensetzung der Flüssigkeit nicht mit der Gleichgewichtszusammensetzung übereinstimmt. In Kapitel 3.3 ist die praktische Ausführung der Auswertung beschrieben.
Für Präzisionsbestimmungen dieser Abkühl- und Aufheizkurven ist ein großer Aufwand erforderlich, wobei alle systematischen Fehler, wie Wärmeleitung über das Pt-Thermometer, Joulesche Wärme vom Pt-Thermometer, ungenügende Äquilibrierung und Druckeffekte ebenso eliminiert werden müssen und außerdem für eine hohe Reinheit der Substanzen zu sorgen ist. Dabei geht es nicht nur um die üblichen Verunreinigungen der Probe, auch durch gelöste Gase können sich beachtliche systematische Fehler ergeben. So beobachteten z. B. Richards et al [14] eine Gefrierpunktserniedrigung von 31 mK durch gelöste Luft bei der Bestimmung des Schmelzpunktes von Benzol.
Bei den für die neuentwickelte Apparatur angestrebten Untersuchungen hat man noch mit erheblich größeren Fehlern zu rechnen, da die zu bestimmende Liquiduslinie bivariant (f=2) oder bei konstantem Druck univariant (f=1) ist. Fällt, z. B. bei einem binären eutektischen System bei Abkühlung (siehe 3.2) ein reiner Stoff aus, ergeben sich neben den unvermeidlichen Temperaturgradienten noch starke Konzentrationsgradienten, die durch geeignete Durchmischungsmethoden möglichst schnell ausgeglichen werden müssen. Das wesentliche Problem besteht darin, dass man durch Abkühlungs- und auch Aufheizkurven (also im thermodynamischen Nichtgleichgewicht) versucht, eine Gleichgewichtsgröße, einen Punkt auf der Liquiduskurve zu bestimmen. Wählt man die Abkühlungsraten zu hoch, erhält man zwar gut auswertbare T(t)-Kurven, die aber stark fehlerbehaftet sind; wählt man die Abkühlungsraten zu niedrig, ergeben sich nicht auswertbare T(t)-Kurven. Die Konsequenz sind erheblich gesteigerte Fehlergrenzen, die in günstigen Fällen bei 0,1 K [5] bis 1 K [15], [16], [17] liegen, aber leicht bis zu 10 K erreichen können [5].
Eine weitere Alternative ist in [18] beschrieben. Bei dieser Methode werden die Proben in abgeschlossenen Glasgefäßen in ein Ölbad eingebracht, dessen Temperatur kontinuierlich verändert wird. Phasenübergänge werden nach dieser Methode optisch mittels Photosensoren detektiert. Als Temperatur des Phasenübergangs wird die Temperatur des Ölbads zum Zeitpunkt der Änderung der optischen Durchlässigkeit des Systems angenommen. Die Nachteile dieser Methode sind zum einen darin zu sehen, dass sich diese Methode nur für optisch durchlässige Proben eignet. Ein Einsatz bei stark gefärbten oder auch durch Schwebstoffe getrübten Proben, wie sie in der Technik häufig vorkommen, ist aber nicht möglich, da hier die optische Detektion versagt. Ein weiterer Nachteil dieser Methode besteht darin, dass nur die Temperatur im Ölbad und nicht aber innerhalb der Probe bestimmt wird. Es ergibt sich dabei die Gefahr, dass trotz geringer Abkühlraten zwischen der Probe und dem Ölbad eine Temperaturdifferenz besteht, und daher der Phasenübergang einer falschen Temperatur zugeordnet wird. Die Beobachtung von Schmelzvorgängen durch Aufheizen einer Probe ist mit dieser Apparatur nur schwer möglich, da die Lichtdurchlässigkeit amorpher Festkörper sehr gering ist. Eine sichere Detektion des Phasenübergangs ist also erst bei einer nahezu vollständig geschmolzenen Probe möglich. Dieser Zustand wird aber auch bei geringen Aufheiz-

raten erst deutlich über dem Schmelzpunkt erreicht. Auch die Beobachtung von Erstarrungsvorgängen ist mir dieser Apparatur nur schwer möglich, wenn eine Unterkühlung auftritt. Eine Unterkühlung kann im Gegensatz zu den obigen Methoden nicht erkannt werden, da die Temperatur im Ölbad und nicht innerhalb der Probe gemessen wird. Ein Temperaturanstieg, hervorgerufen durch die Erstarrungswärme, wird somit nicht detektiert. Es ist daher keine sichere Aussage über Erstarrungspunkte möglich.

2.3 Grundlagen der Leitfähigkeitsmessung

Die Grundlagen der Theorie der Leitfähigkeit von Elektrolytlösungen ist in Kapitel 4.2.1.1 beschrieben. Daher wird hier nur die Messmethode erläutert.
Die Leitfähigkeit G von Elektrolytlösungen kann man mit Hilfe von Widerstandsmessungen auf der Basis von Gleichspannungs- oder Wechselspannungsmethoden bestimmen. Für diesen Zweck werden geeignete Messzellen eingesetzt, deren Widerstand bei eingefülltem Elektrolyt bestimmt wird. Sind die geometrischen Dimensionen des flüssigen Leiters, Querschnitt A und Elektrodenabstand l genau bekannt, kann die spezifische Leitfähigkeit der Elektrolytlösung κ berechnet werden.

$$\kappa = \frac{1}{R} \cdot \frac{l}{A} \tag{3}$$

Da es in der Praxis jedoch nicht einfach ist, die geometrischen Dimensionen in einer elektrochemischen Zelle genau zu bestimmen, werden quaderförmige, genau vermessene Zellen nur benutzt, um einige Eichwerte für $\varkappa$ zu erhalten [19], [20]. Um mit beliebig geformten Zellen spezifische Leitfähigkeiten von Elektrolytlösungen bestimmen zu können, werden diese dann mit Eichlösungen bekannter spezifischer Leitfähigkeit geeicht. Gleichung (3) lässt sich für diesen Zweck so umschreiben:

$$\kappa = C \cdot G \tag{4}$$

Die Zellkonstante C einer beliebig geformten Leitfähigkeitszelle kann näherungsweise als eine Summe von Einzelzellen i, die zur Zellkonstante C gemäß $C = \Sigma Ci = \Sigma(li/Ai)$ beitragen, aufgefasst werden.
Aufgrund von Polarisationseffekten können Gleichspannungsmethoden zur Bestimmung von Elektrolytleitfähigkeiten nicht angewandt werden. Außerdem können sie nur an reversibel arbeitenden Elektroden zum Einsatz kommen. Wechselspannungsmethoden weisen diese Nachteile nicht auf. Üblicherweise werden für die Präzisionsbestimmung von Leitfähigkeiten Messanlagen, wie zum Beispiel die in Kapitel 4.2.2 beschriebene Anlage, eingesetzt.

2.4 Überblick über die Anlage

In Abbildung 4 ist die Anlage zur Bestimmung von Phasendiagrammen gezeigt. Die Messzellen (1) tauchen dabei in den Thermostaten (6) ein. Dieser ist über das Ventil (2) mit dem Kryostaten, der sich im Nebenraum befindet, verbunden. Mit diesem Kryostaten, der von einem Programmgeber gesteuert wird, kann das Thermostatenbad linear abgekühlt und aufgeheizt werden. Die Messzellen

sind an das 30-Kanalthermometer (5) und das Leitfähigkeitsmessgerät (4) angeschlossen. Diese Geräte werden mit einem Steuerrechner, der in Abbildung 4 nicht abgebildet ist, angesteuert. Das Referenzthermometer (3) dient zur Kalibrierung der Temperaturfühler. Für die Kalibrierung wird die Temperatur im Bad des Thermostaten mit der PID-Regeleinrichtung (9) konstant gehalten.

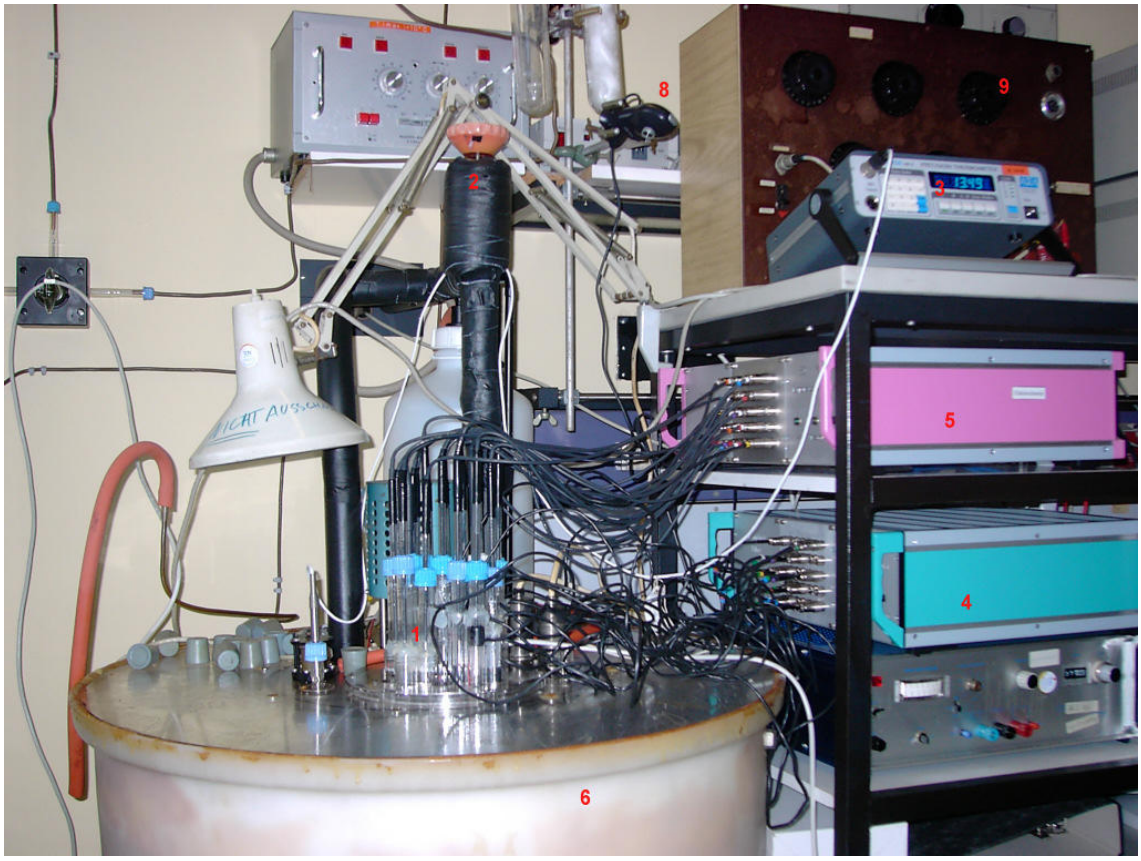


Abbildung 4 Anlage für die Messung von Phasendiagrammen, temperaturabhängigen Leitfähigkeiten und Leitfähigkeitskinetiken

2.5 Mechanischer Aufbau, Messzellen und Thermostat

2.5.1 Messzellen

Die verwendeten Messzellen stellen eine Weiterentwicklung der in [21] beschriebenen Zellen dar. Im Gegensatz zu den dort dargestellten Messzellen ist die in Abbildung 5 gezeigte Messzelle mit einem luftgefüllten Isolationsmantel ausgestattet. Dieser Mantel erfüllt zwei Aufgaben. Zum einen werden Temperaturschwankungen des Ölbads noch weiter gedämpft, zum anderen wird durch die Isolation eine Verbesserung der Auflösung der Knick- und Haltepunkte erwartet, da die freigesetzte Wärmemenge nicht sofort an das Ölbad abgegeben wird und dadurch die Einstellung der Gleichgewichtstemperatur erleichtert wird.

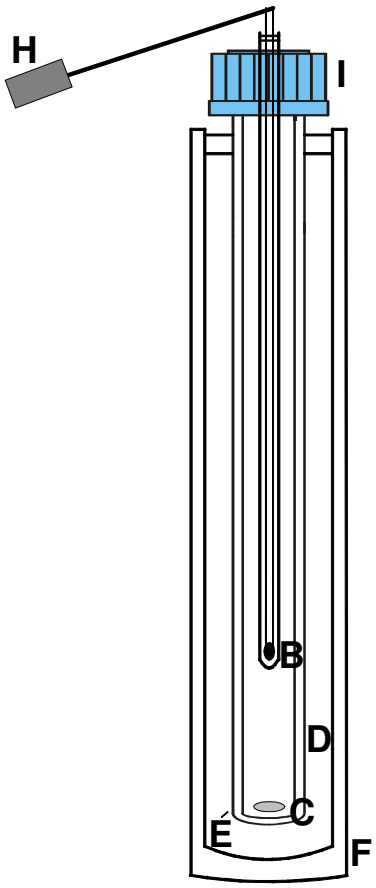


Abbildung 5 Messzelle ohne Elektroden

Die Gefäße bestehen aus einem 34 cm langen, unten abgeschmolzenen Glasrohr D mit einem Außendurchmesser von 15 mm. Das obere Ende dieser Gefäße ist mit einem SQ 18 Glasgewindeanschluss versehen, der über eine durchbohrte, mit Teflon beschichtete Gummidichtung den Temperaturmessfühler B aufnimmt. Ein sicherer Verschluss wird durch eine verschraubbare Kappe I gewährleistet. Der Isolationsmantel E wird durch ein weiteres, unten abgeschmolzenes Glasrohr F hergestellt, das über das innere Glasrohr geschoben wird. Das innere Glasrohr wird durch O-Ringe im Mantelrohr und durch eine Verklebung mit Heißkleber zentriert. Die Lösung wird mit einem Magnetrührstab C gerührt.
Die für die Leitfähigkeitsmessungen eingesetzten Messzellen, siehe Abbildung 6, weisen einen analogen Aufbau auf. Sie unterscheiden sich durch zwei Platinringelektroden, die in den Messzellen anbracht sind.

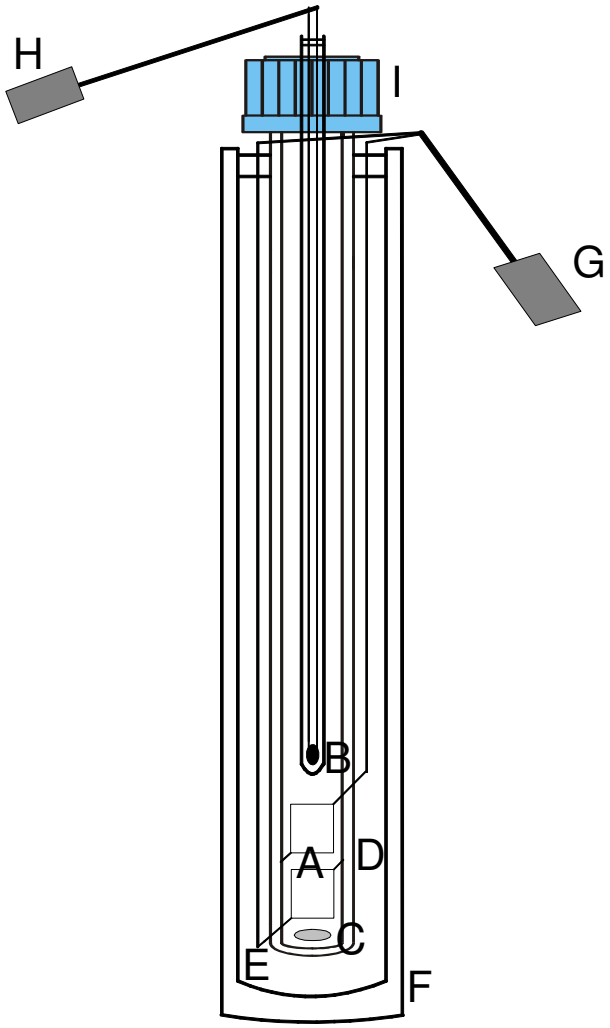


Abbildung 6 Messzelle mit Elektroden

Die Elektroden A werden dabei durch Platindrähte, die in der Glaswand eingeschmolzen sind, fixiert. Die Zellkonstante dieser Zellen liegt im Bereich von 0,4 bis 0,5 cm^{-1}. Über diese Drähte erfolgt auch die Kontaktierung mit dem Stecker G. Um eine gute Abschirmung zu erreichen, sind die Anschlussleitungen innerhalb der Messzelle als verdrillte Paarleitung ausgeführt. Außerhalb der Messzelle wird eine doppelt abgeschirmte Messleitung verwendet, die auch bei den Thermistoren zum Einsatz kommt, siehe Abschnitt 2.7.2.1. Da die Zellkonstante dieser Messzellen unter anderem auch von ihrem Füllstand abhängt, ist es bei jeder Messung nötig, die Zellkonstanten neu zu bestimmen, siehe Abschnitt 2.8.3.

2.5.2 Rührwerke

Bei der Aufnahme von Abkühlkurven wie auch von Erwärmungskurven ist eine gute Durchmischung der zu untersuchenden Substanz unumgänglich, um Temperaturinhomogenitäten und somit Messfehler zu vermeiden. Die Durchmischung der Lösung bei der von Hammer entwickelten Apparatur erfolgt durch eine Blasenrührung mit Inertgas [22]. Der Nachteil dieser Methode besteht darin, dass von Mischungen Komponenten ausgetrieben werden, die einen hohen Dampfdruck besitzen und damit die Messungen verfälschen. Diesen Nachteil weist die in der Diplomarbeit des Autors [21] gezeigte Hubrührapparatur nicht auf. Bei dieser Apparatur erfolgt die Durchmischung durch Glasstäbe, welche in Magnete eingebettet sind, die durch ein Magnetfeld vertikal bewegt

werden. Da aber diese Glasstäbe ungebremst auf den Boden der Messzelle aufprallen, ist die Haltbarkeit dieser Rührstäbe sehr gering. Bei Messzeiten von mehreren Tagen beträgt die Ausfallquote ca. 50 %.

Aus diesem Grund wurde eine neue Apparatur entwickelt, welche die Lösung mit einem drehenden Magnetstab durchmischt. Die hier beschriebene Apparatur wurde von Engelhardt nach Vorgaben von Gores konstruiert. Bei dieser Apparatur, die in Abbildung 7 als Schnittbild dargestellt ist, werden die Magnetrührstäbchen A mit Magneten B angetrieben, die auf den Zahnrädern C eines Planetengetriebes angebracht sind. Das Planetengetriebe wird über einen Gleichstrommotor D, der über eine Welle E mit dem Getriebe verbunden ist, angetrieben.

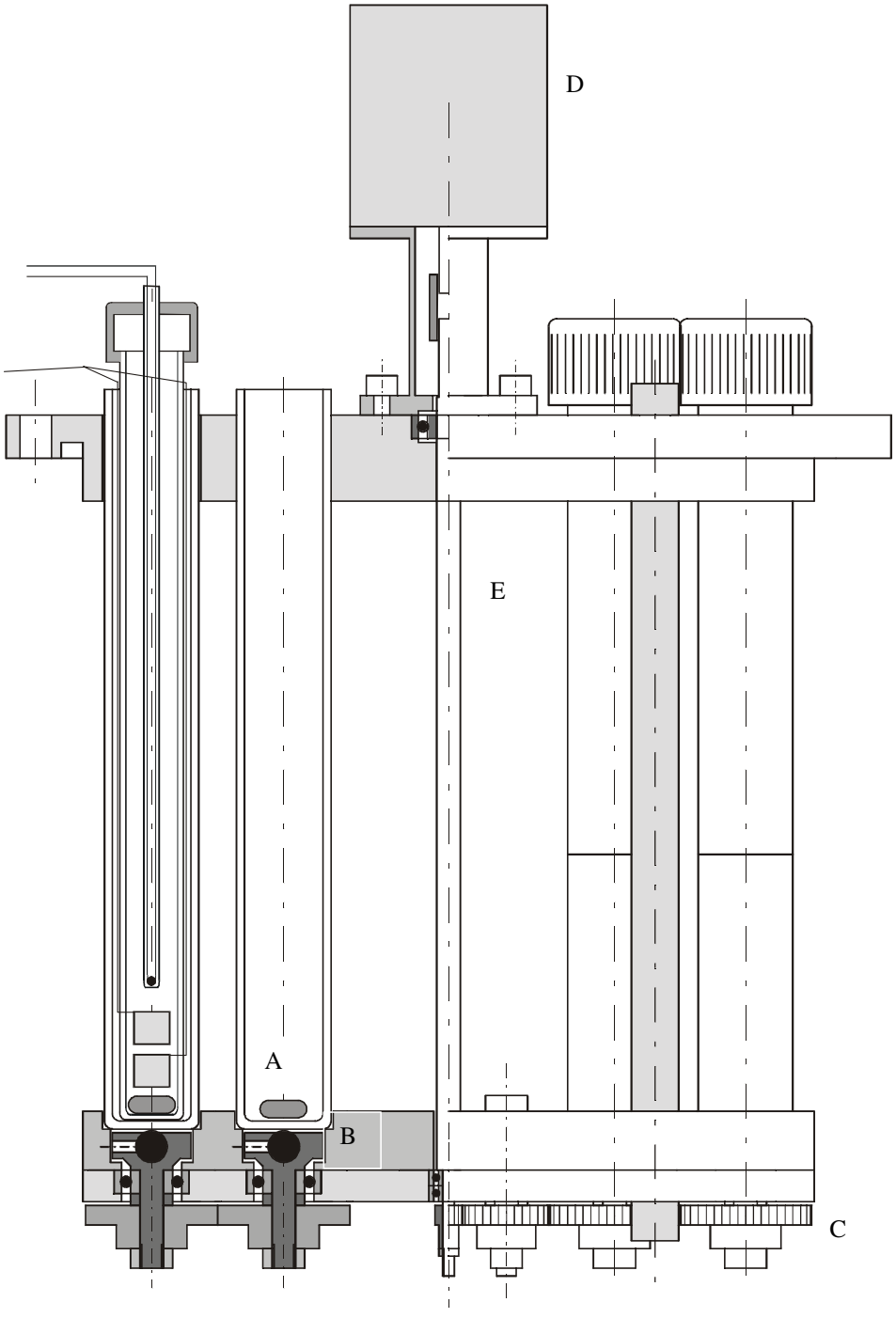


Abbildung 7 Zahnradrührapparatur

Die Zahnradrührapparatur nimmt maximal 21 von den in Kapitel 2.5.1 beschriebenen Messgefäßen auf, die durch den Deckel F und durch Aussparungen in der Bodenplatte G des Rührwerks gehalten werden. Aufgrund des großen Temperaturbereichs (-50 °C bis +40 °C), bei dem dieses Rührwerk eingesetzt wird, sind die Anforderungen an die Auslegung des Planetengetriebes sehr hoch. Insbesondere das Zahnrad, das die Bewegung der Motorwelle auf das Planetengetriebe überträgt, weist einen hohen Verschleiß auf. Weitere Problemstellen dieser Apparatur sind in den Kugellagern, welche die Zahnräder in der Bodenplatte fixieren, zu sehen. Werden Kugellager eingesetzt, die nicht gekapselt sind, so besteht die Gefahr, dass sich der Abrieb von den Zahnrädern in den Kugellagern ablagert und diese dann blockieren. Eine Verbesserung der Haltbarkeit der Anlage wird durch gekapselte Kugellager erreicht. Da aber bei diesem Lagertyp die Kugeln durch eine Fettpackung geschmiert werden, die durch das Silikonöl langsam heraus gelöst wird, besteht auch hier die Gefahr des Versagens dieser Lager bei längerer Betriebszeit. Für die Ansteuerung des Rührmotors wird eine Motorsteuerung der Firma Dunker vom Typ RS-200 verwendet, bei der die Drehzahl des Motors über eine Steuerspannung eingestellt wird, die vom Digital/Analog-Wandler des Thermometers erzeugt wird (siehe auch Abschnitt 2.7.2.6). Die Motorsteuerung wird mit einem vom Autor dieser Arbeit entwickelten Leistungsnetzteil versorgt.
Aufgrund der sich ergebenden Probleme wurde ein weiteres Rührwerk entwickelt, das ohne bewegte mechanische Bauteile auskommt. Das Prinzip für diese Durchmischungsvorrichtung besteht darin, dass durch vier quadratisch angeordnete Spulen Magnetfelder erzeugt werden. Wird in geeigneter Weise Magnetfeld variiert, so kann ein Magnetrührstab, der senkrecht zu den Spulen liegt, in eine Drehbewegung versetzt werden.
In Abbildung 8 ist ein Schnittbild eines derartigen Rührwerks gezeigt. Wie auch beim Zahnradrührwerk werden die Messzellen durch die Deckplatte und durch Aufnahmen in der Bodenplatte gehalten. In der Bodenplatte dieses Rührwerks befinden sich Magnetspulen, mit denen das Wandelfeld erzeugt wird. Diese Spulenplatte wurde aus einem Synthesenblock für die Kombinatorische Chemie vom Typ SAS der Firma Multisyntech (Witten) entnommen[2] und für diese Halter angepasst. Die Ansteuerung dieser Platte, die im Gehäuse des Konduktometers untergebracht ist, wurde von der Elektronikwerkstatt der Fakultät für Chemie aufgebaut.

[2] Herrn Dr. von Nussbaum und der Bayer AG (Leverkusen) sei an dieser Stelle für die Überlassung des Syntheseblocks gedankt.

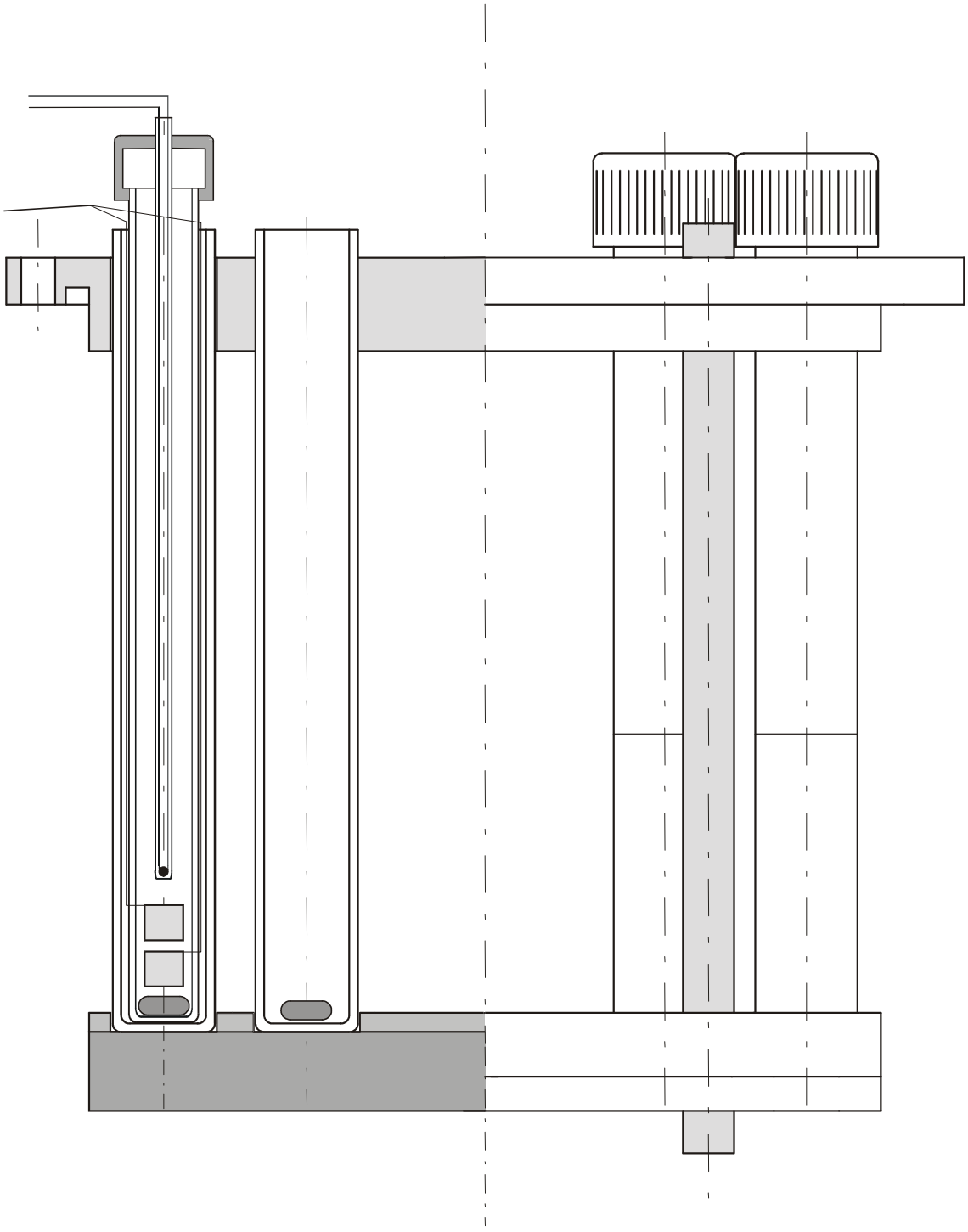


Abbildung 8 Wandelfeldrührwerk

2.5.3 Thermostatenanlage

Zur Bestimmung von Schmelzpunkten durch Abkühl- bzw. Erwärmungskurven ist es erforderlich, die zu untersuchende Substanz einem Temperaturgradienten auszusetzen. Bei der in [22] beschriebenen Methode wird das Gefäß, das die zu untersuchende Substanz enthält, in einen Thermostaten getaucht, der auf eine Temperatur eingestellt ist, die tiefer als die kleinste zu erwartende Schmelztemperatur ist. Durch Variation dieser Endtemperatur lassen sich verschiedene Abkühlraten realisieren [23]. Um eine hinreichend kleine Abkühlrate zu erzielen, hat das in [22] beschriebene Gefäß einen Vakuumisolationsmantel. Da der Isoliermantel der in 2.5.1 beschriebenen Messzellen aus

Platzgründen nicht so stark wie erforderlich ausgeführt werden konnte, ist es nötig, die gewünschten Kühl- bzw. Heizraten durch Veränderung der Thermostattemperatur zu erzielen. Für diesen Zweck wurde eine Thermostatenanlage verwendet, deren Aufbau in [24] beschrieben und in Abbildung 9 dargestellt ist.

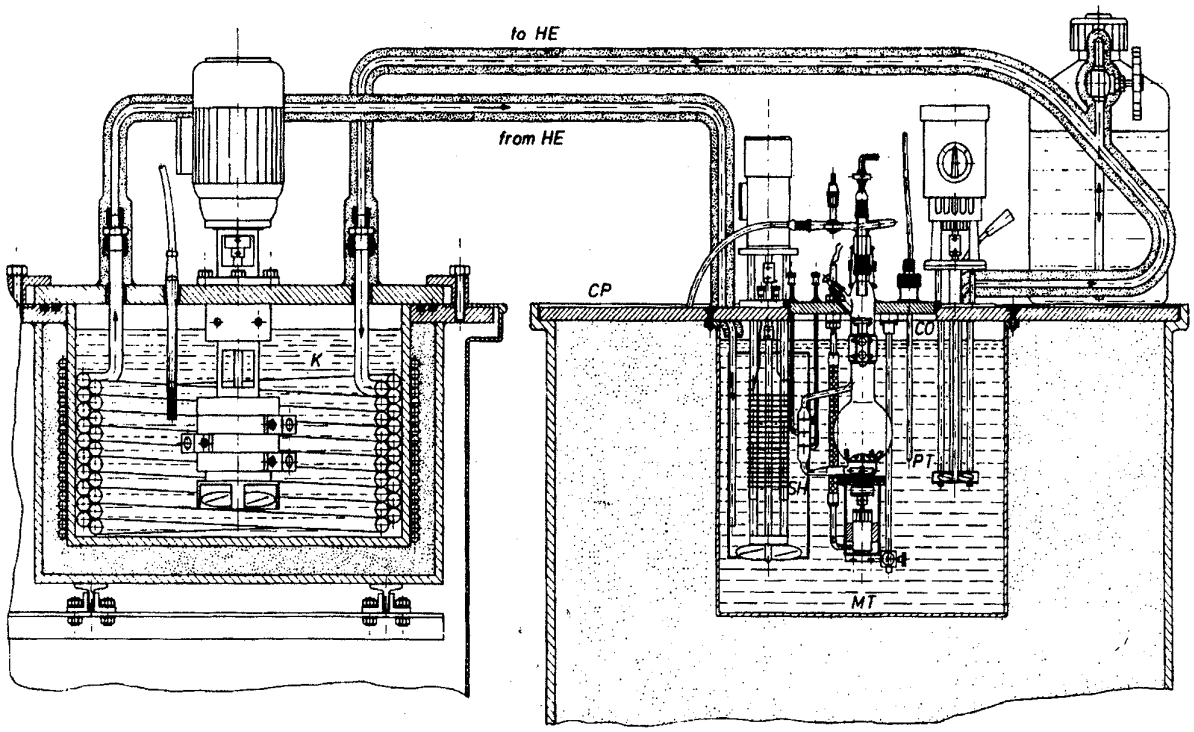


Abbildung 9 Thermostatenanlage [24]

Für den in Abbildung 9 links dargestellten Kryostaten, der als Wärme-Senke dient, stehen zwei verschiedene Anlagen zur Verfügung: ein Kryostat der Firma Lauda (Lauda-Königshofen) K90SW und ein Kryostat vom Typ RUK 40 S. Die Kältemaschine K90SW lässt sich auf eine konstante Temperatur im Bereich von –80°C bis +10 °C einstellen. Die Abkühlrate wird durch die Kälteleistung und durch den Durchfluss an Kältemittel eingestellt. Verschiedene Aufheizarten werden durch einen Zusatzheizer realisiert. Wie in [21] beschrieben, ist es mit dieser Anlage nicht möglich, lineare Temperaturgradienten einzustellen. Sollen lineare Temperaturgradienten eingesetzt werden, wodurch die Auswertung der Messung deutlich vereinfacht wird, so kann ein Kryostat vom Typ RUK 40 S der Firma Lauda an die Thermostatenanlage angeschlossen werden. Durch einen Programmgeber ist es mit diesem Gerät möglich, die Temperatur im Thermostaten mit der Zeit linear zu verändern. Aufgrund der geringeren Kälteleistung dieses Geräts beträgt die niedrigste Temperatur im Bad des Thermostaten, die erreicht werden kann, -35 °C. Diese Temperatur beschränkt derzeit noch die untersuchbaren Systeme. Ein bereits bestellter Kryostat, Firma Huber, Typ unistat 390w wird den Temperaturbereich zu tiefen Temperaturen hin bis –90°C erweitern.

Als kälteübertragendes Medium zwischen dem Kryostaten und dem Thermostaten dient bei beiden Anlagen denaturiertes Ethanol, dessen Durchflussmenge mit dem in Abbildung 9 oben rechts dargestellten Ventil eingestellt werden kann.

Wird die Thermostatenanlage bei konstanter Temperatur betrieben, so wird die bereits oben angeführte von Barthel et. al. [24] beschriebene Regelanlage eingesetzt. Als Wärmequelle fungiert dabei

ein Widerstandsheizer, der durch einen PID-Regler über einen Leistungsverstärker angesteuert wird. Das Regelsignal dieses Reglers wird mit einer Kohlrauschbrücke erzeugt, die auch zur Temperatureinstellung dient, an die als Temperaturfühler ein Pt-500 Widerstandsthermometer angeschlossen ist. Für eine homogene Temperaturverteilung des mit dem Silikonöl Baysilon® M5 gefüllten Thermostatenbades, in das in Abbildung 9 eine Leitfähigkeitsmesszelle eintaucht, sorgt ein Rührwerk. Mit zwei Tauchsiedern ist es möglich, die Temperatur des Ölbades in kurzer Zeit auf eine gewünschte Zieltemperatur aufzuheizen. Die Abkühlung des Thermostaten wie auch das Einsetzen von großvolumigen Apparaturen erfordern eine Niveauregulierung des Flüssigkeitsspiegels im Ölbad, die durch das im oberen rechten Rand dargestellte Niveaureguliergefäß erreicht wird.

2.5.4 Sicherheitseinrichtungen

Da die Thermostatenanlage im Gegensatz zu den früheren Anwendungen, bei denen sie nur tagsüber unter Aufsicht benutzt wurde, bei den Abkühlungs- und Aufheizexperimenten nun auch über mehrere Tage ununterbrochen betrieben wird, sind eine Reihe von Sicherheitsmaßnahmen notwendig. Kern der Sicherheitseinrichtungen ist eine neuentwickelte Abschaltvorrichtung, mit der die komplette Thermostatenanlage außer Betrieb gesetzt werden kann. An diese Vorrichtung ist eine Übertemperatursicherung angeschlossen, die anspricht, wenn die Temperatur innerhalb des Thermostaten auf über 80 °C ansteigt. Da es selbst bei Temperaturen unterhalb 80 °C denkbar sein kann, dass der Thermostat aufgrund der Ausdehnung des Silikonöls überläuft, wurde noch ein Schwimmer angebracht, der ebenfalls über die Abschaltvorrichtung die Anlage ausschalten kann. Die dritte Sicherheitseinrichtung ist eine Abschaltung der Anlage, die vom Steuerrechner aus erfolgen kann. Für diese Abschaltmöglichkeit wird die Abschaltvorrichtung mit dem Thermometer verbunden. Wird als Betriebssystem für den Steuerrechner Microsoft Windows XP eingesetzt, so kann über die Remote Desktop Verbindung dieses Betriebssystems auf den Steuerrechner von jedem Rechner mit Internetzugang aus zugegriffen und dieser ferngesteuert werden. Zusätzlich verfügt dieser Rechner noch über eine WEB-CAM, mit welcher der Thermostat zusätzlich überwacht wird.
Der Kryostat wurde mit einem Wasserwächter der Firma Greisinger Electronic GmbH (Regenstauf) ausgestattet. Mit diesem Wasserwächter wird der Kryostat vom Kühlwasserkreislauf getrennt und über eine neuentwickelte Abschaltvorrichtung von der Stromversorgung abgetrennt, wenn ein Leck im Kühlwasserkreislauf des Kryostaten auftritt.

2.6 Einleitende Bemerkungen (Modularität, Rechnerkopplung, Programme)

Ein Ziel bei der Entwicklung der Messgeräte und der Steuersoftware war, diese soweit wie möglich modular zu gestalten. Durch diese Modularisierung konnte ein erheblicher Teil des Entwicklungsaufwands eingespart werden, da einzelne Module für verschiedene Geräte eingesetzt werden können. Ein gutes Beispiel[3] hierfür ist die Digital/Analog-Wandlerstufe. Diese wird beim Thermometer sowohl für die Ansteuerung des Heizers wie auch für das Zahnradrührwerk verwendet. Beim Leitfähigkeitsmessgerät dient sie für die Ansteuerung des Frequenzgenerators und beim Batterietestsys-

[3] Eine Reihe von Modulen der Geräte, die in dieser Arbeit beschrieben werden, wurden auch für eine Vielzahl von weiteren Geräten, welche der Autor entwickelt hat, eingesetzt (Polarograph, Handthermometer, Datenerfassungssysteme, ...).

tem wird sie zum Einstellen des Konstantstroms benutzt. Durch diese Modularisierung konnte auch sehr leicht das Leitfähigkeitsmessgerät entwickelt werden, das im Prinzip dem Thermometer entspricht, bei dem anstelle von Thermistoren Leitfähigkeitsmesszellen angeschlossen sind, die mit Wechselspannung betrieben werden.
Eine deutliche Flexibilisierung der Messgeräte wird durch den Einsatz von Mikrokontrollern zur Steuerung der einzelnen Bauteile erreicht. Durch eine Einführung einer Steuersprache, mit der alle wichtigen Grundfunktionen der einzelnen Geräte kontrolliert werden können, muss für eine Erweiterung der Möglichkeiten des Messgeräts meist nicht die Software des Mikrokontrollers verändert werden, sondern es reicht aus, die Steuersoftware, die auf einem PC ausgeführt wird, anzupassen. Durch die Steuersprache ist auch ein Austausch der einzelnen Komponenten eines Geräts möglich, ohne dass die Steuersoftware verändert werden muss. Dadurch können Geräte entwickelt werden, die zwar intern anders aufgebaut sind, aber nach außen hin keine Unterschiede aufweisen.
Auch die Steuersoftware wurde modular gestaltet, so dass viele Teile des Quelltextes für verschiedene Geräte eingesetzt werden können. Im Prinzip folgen alle entwickelten Messgeräte folgendem Schema:

- Sensoren und Umformung beziehungsweise Aktuatoren und Ansteuerung
- Analog/Digital-Wandler beziehungsweise Digital/Analog-Wandler
- Mikrokontroller
- Interface
- Steuerrechner

Innerhalb dieser Ebenen können dann einzelne Module ausgetauscht werden, ohne dass große Änderungen der anderen Ebenen erfolgen müssen, wobei für ein neues Gerät meist nur eine neue Kombination von Sensoren und Aktuatoren verwendet wird, die über eine passende Messwertumformung beziehungsweise Ansteuerung mit den Wandler angeschlossen wird. Durch die Mikrokontroller erfolgt dann die Ansteuerung der Wandler und eine erste Aufbereitung der Messdaten, die auf den jeweiligen Einsatzbereich abgestimmt sind. Mit der Software des Steuerrechners können dann über eine Steuersprache die Messungen durchgeführt und die erfassten Messdaten gespeichert und ausgewertet werden.

2.7 Das Multikanalthermometer

Insbesondere für die Bestimmung von Phasendiagrammen durch Abkühl- beziehungsweise Aufheizexperimente werden hohe Anforderungen hinsichtlich der Genauigkeit und Auflösung der Temperaturmessung gestellt. Da die Anlage vor allem für die Untersuchung von Eigenschaften von Elektrolyten bei tiefen Temperaturen konzipiert ist, beträgt ihr Messbereich –50 °C bis +40 °C. Die Messgenauigkeit sollte, vor allem bei tiefen Temperaturen im Bereich von Platinwiderstandsthermometern, also bei ± 25 mK liegen. Um Unterkühlungseffekte untersuchen zu können, sollte dabei die Auflösung des Messgeräts bei mindestens 1 mK liegen. Die Geschwindigkeit, mit der die Messdaten erfasst werden, sollte im Bereich der Größenordnung von einer Messung pro Sekunde liegen, damit auch bei großen Abkühl- bzw. Aufheizraten eine genaue Bestimmung des Temperaturverlaufs möglich ist. Um eine ausreichende Anzahl an Versuchen gleichzeitig durchführen zu können, sollte das Thermometer über 30 Messstellen verfügen.

Das hier beschriebene Messgerät erfüllt diese Anforderungen. Es ist eine Weiterentwicklung des in der Diplomarbeit des Autors [21] beschriebenen 8-Kanal Prototypen. Da dieses Gerät ein sehr großes Eigenrauschen aufweist, war es ein weiteres Ziel dieser Arbeit, dieses zu reduzieren. Darüber hinaus wurde eine Vielzahl an Verbesserungen bei der Auswahl der einzelnen Komponenten, des mechanischen Aufbaus und vor allem bei der Störfestigkeit durchgeführt.
In Abbildung 10 ist die Vorderansicht des Thermometers abgebildet, in der die drei Funktionsbereiche des Geräts deutlich sichtbar werden. Im Einschub, der sich auf der rechten Seite befindet, ist die Stromversorgung und der Digitalteil des Thermometers untergebracht. Der mittlere Einschub enthält die eigentliche Messelektronik, mit den Spannungsteilern, den Operationsverstärkern und den Analog/Digitalwandlern. An der Vorderseite dieses Einschubs befinden sich auch die 30 Anschlüsse für die Temperaturfühler. Durch die Trennung der beiden Funktionsbereiche wird eine sehr gute Abschirmung erreicht. Auf der linken Seite ist der Einschub für die Leistungselektronik für das Zahnradrührwerk untergebracht.

Abbildung 10 30-Kanalthermometer, Vorderansicht

2.7.1 Funktionsprinzip

Normalerweise werden Platinthermometer für hochgenaue Temperaturmessungen verwendet, da sie über eine sehr gute Langzeitstabilität und Reproduzierbarkeit verfügen. Aus diesem Grund werden diese Fühler für die Interpolation zwischen den Fixpunkten der internationalen Temperaturskala ITS-90 verwendet [25],[26]. Da aber diese Sensoren nur einen kleinen Temperaturkoeffizienten von typischerweise $3{,}9\times10^{-3}$ 1/K aufweisen [27], benötigen diese Sensoren eine aufwändige Elektronik für die Messdatenerfassung. Normalerweise wird der Widerstand dieser Bauteile mit Brückenmethoden gemessen, wie zum Beispiel mit einer klassischen Kohlrauschbrücke [28] oder Transformatorbrücken [29], [30]. Da aber diese Schaltungen zu langsam sind, wenn jede Sekunde Temperaturen an 30 verschiedenen Messzellen gemessen werden, stellte sich die Frage nach einer Alternative. Erschwerend kommt hinzu, dass diese Temperatursensoren, welche die Anforderungen für auf nationale Standards rückführbare Messungen erfüllen, aus gewickeltem Platindraht beste-

hen. Diese Sensoren weisen aufgrund ihrer Bauart relativ große Abmessungen auf, wodurch sie nicht mehr in kleinen Messzellen eingesetzt werden können. Als viel kritischer ist aber die große Zeitkonstante dieser Fühler zu bewerten, die Messungen mit hoher Zeit- und Temperaturauflösung unmöglich macht. Platinwiderstandssensoren, die mit der Dünnschicht-Technik hergestellt wurden, könnten dieses Problem lösen. Da diese Sensoren nicht den Anforderungen bezüglich der internationalen Temperaturskala ITS-90 genügen [31],[32], wird der Hauptvorteil der Platinwiderstände, nämlich der Kompatibilität zum Standard, verloren. Ein weiterer gewichtiger Nachteil von Platinwiderstandssensoren ist ihr hoher Anschaffungspreis, der im Bereich von 150 € für einen unkalibrierten Sensor beginnt und bis in die Größenordnung von 750 € für einen kalibrierten Sensor ansteigt. Soll eine Anlage mit 30 Temperaturmessstellen ausgerüstet werden, wäre daher eine sehr große Investition für die Sensoren nötig.

Thermistoren hingegen zeichnen sich nicht nur durch ihren großen Temperaturkoeffizienten aus; typische Werte liegen im Bereich von –0,03 1/K bis –0,06 1/K [33], was im Vergleich zu Platinwiderstandsthermometern ca. um den Faktor 10 größer ist. Aufgrund des großen Temperaturkoeffizienten wird der Aufbau der Auswertungsschaltung deutlich erleichtert, da für sie ein einfacher Spannungsteiler an Stelle der Brückenschaltungen eingesetzt werden kann. Neben der Kostenreduzierung, die durch den einfachen Schaltungsaufbau erreicht wird, stellt vor allem der günstige Preis eines Thermistors von unter 10 € das wichtigste Argument dar, das für die Anwendung von Thermistoren bei vielkanaligen Messgeräten spricht. Durch die Spannungsteilerschaltung wird auch eine im Vergleich zu Brückenschaltungen deutlich einfachere Messdatenerfassung möglich, da diese Schaltung eine leicht auswertbare Spannung liefert, und im Gegensatz zu Brücken keine komplizierten Abgleichvorrichtungen aufgebaut werden müssen.

Das Kunstwort Thermistor stammt aus dem Englischen; es leitet sich ab aus *thermally sensitive resistor*. Thermistoren werden nach ihrem Temperaturverhalten in zwei Klassen eingeteilt. Wenn ihr Widerstand mit der Temperatur zunimmt, dann ist ihr Temperaturkoeffizient positiv und man spricht von einem PTC (*positive temperature coefficient*). Dieses Verhalten ist typisch für metallische Leiter, wie bei den oben beschriebenen Platinwiderständen. Fällt der Widerstand hingegen bei zunehmender Temperatur, wie bei Halbleitern beobachtet wird, so erhält man einen NTC (*negative temperature coefficient*) Widerstand. Diese Sensoren werden meist aus Übergangsmetalloxiden wie Mangan-, Kobalt-, Nickel-, Eisen- oder Titanoxid, hergestellt [34].

Thermistoren haben einen schlechten Ruf als Temperatursensoren, da ihnen eine unzureichende Langzeitstabilität und Reproduzierbarkeit nachgesagt wird, wofür sich allerdings in der Literatur so gut wie keine Beweise finden [35]. Im Gegensatz zu diesem Vorurteil, das vermutlich auf Erfahrungen basiert, die sehr lange zurückliegen, können Thermistoren eine sehr hohe Langzeitstabilität aufweisen [36]. Beispielsweise beträgt die Drift eines Thermistors ±200 µK/1000 h [35]. In der Literatur wird von Hans [37], [38] von einem Thermometer berichtet, das einen Thermistor als Temperatursensor einsetzt, und das eine Genauigkeit von 10 mK besitzt. Die verminderte Langzeitstabilität zeigt sich weniger in der Veränderung des Temperaturkoeffizienten als in plötzlichen Veränderungen des Nennwiderstandes. Die Sprünge werden vor allem durch Gefügeveränderungen des Sensormaterials hervorgerufen [39]. Durch geschickte Voralterung bei höheren Temperaturen [40] und durch Optimierung der Zusammensetzung der Metalloxide, insbesondere deren Gefüge, gelingt es, diese Fehler zu minimieren [39]. Durch die Einführung der Sägetechnik, bei der die Sensoren aus einem großen homogenen Sinterkörper herausgesägt werden, wurde eine starke Reduzierung der Exemplarstreuung gegenüber der Sintertechnologie, bei der jeder einzelne Sensor für sich gesintert wurde, erreicht [39]. Dank dieser Verfahren kann man mittlerweile Thermistoren herstellen, die eine so geringe Exemplarstreuung und eine hohe Langzeitstabilität aufweisen, dass es mög-

lich ist, mit ihnen Temperaturen mit einer Genauigkeit von ±0,1K zu messen, vorausgesetzt man verwendet eine Standardeichkurve [40]. Durch diese Verbesserungen konnte auch die Toleranz der Thermistoren auf einen Bereich von ± 0,1 K angehoben werden, wie die Thermistoren der Firmen Rhopoint Components Ltd. (Surrey) oder der Firma BetaTHERM Corporation (Shrewsbury) zeigen [40]. Im Anhang 11.3.1.3 wird nachgewiesen, dass die in diesem Gerät eingesetzten Thermistoren selbst über lange Zeiträume und trotz starker Belastung über eine sehr gute Langzeitstabilität verfügen.
Das in dieser Arbeit beschriebene Thermometer basiert auf einem Spannungsteiler, der aus einem Thermistor und einem Präzisionswiderstand aufgebaut ist. Dieser Spannungsteiler wird dazu eingesetzt, den temperaturabhängigen Widerstand des Thermistors in eine temperaturabhängige Spannung umzuformen. Diese Spannung wird dann mit einem Analog/Digital-Wandler digitalisiert, der von einem Mikrokontroller oder von einem PC angesteuert wird. Die Temperatur lässt sich dann mit der Steinhart-Hart Gleichung (5) berechnen. Um eine Genauigkeit zu erzielen, die höher ist als 0,1 °C, und die nur erreicht wird, wenn die vom Hersteller der Thermistoren angegebenen Standardkalibrierparameter verwendet werden, ist eine individuelle Kalibrierung der Messfühler nötig. Das Thermometer verfügt über vier Analog/Digital-Wandler mit jeweils acht Eingangskanälen, an die jeweils ein Spannungsteiler über einen Impedanzwandler angeschlossen ist. Da zwei Kanäle zur Bestimmung der Temperatur innerhalb des Messgeräts verwendet werden, verfügt das Gerät über 30 Messstellen.

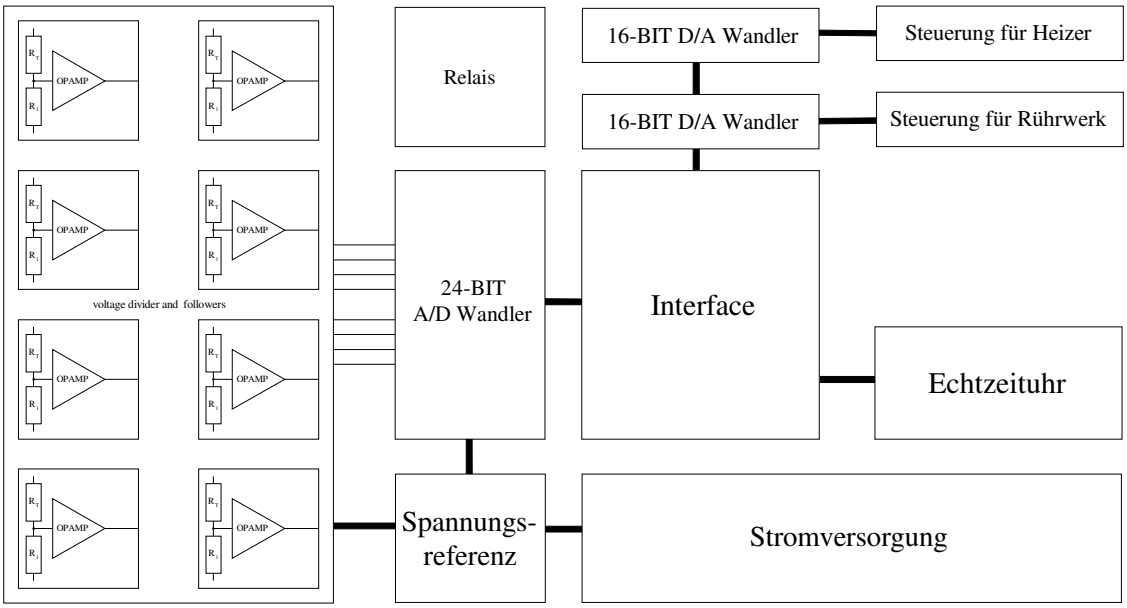


Abbildung 11 Blockschaltbild des Thermometers

In Abbildung 11 ist das Blockschaltbild des Thermometers dargestellt. Bei dieser Abbildung wird nur einer der vier Analog/Digitalwandler mit den angeschlossenen Spannungsteilern gezeigt.
Die vom Analog/Digital-Wandler gemessenen Spannungen und die mit der Echtzeituhr erfassten Zeiten werden vom Mikrokontroller beziehungsweise vom Steuerrechner über eine Interfaceschaltung übertragen. Über die beiden Digital/Analog-Wandler wird eine Steuerspannung zur Einstellung der Rührgeschwindigkeit des Zahnradrührwerks und eine Steuerspannung für den Heizer des Thermostaten erzeugt.

2.7.2 Beschreibung des Systems

2.7.2.1 Sensor

Als Temperatursensoren werden Thermistoren der Firma BetaTHERM Corporation (Shrewsbury) vom Typ BetaCurve eingesetzt. Diese Sensoren können laut Hersteller im Temperaturbereich von 0 °C bis 70 °C austauschbar eingesetzt werden, wobei die Genauigkeit 0,1 K beträgt, wenn die Standardkalibrierdaten des Herstellers verwendet werden [40]. Aufgrund ihrer außerordentlichen Langzeitstabilität, die im Bereich von –0,097 K innerhalb von neun Jahren liegt, sofern der Sensor bei 75 °C gelagert wird [41[], sind diese Sensoren für den Ersatz von Platinwiderständen geeignet. Wie in Abschnitt 2.7.3 gezeigt, kann die Genauigkeit dieser Sensoren durch eine individuelle Kalibrierung noch deutlich verbessert werden. In Abschnitt 2.7.4 wird die Stabilität dieser Sensoren beschrieben, wobei eine Abweichung von 0,02 K beobachtet wird, wenn der Sensor einigen Zyklen zwischen +40 °C und –50 °C unterworfen wird. Laut Hersteller sind sie im Temperaturbereich von 0 °C bis 70 °C austauschbar einsetzbar Die Abweichungen betragen dann 0,1 °C, vorausgesetzt man verwendet eine Standardeichkurve.
Die Temperaturabhängigkeit dieser Sensoren wird mit der Gleichung von Steinhart und Hart (5) beschrieben.

$$\frac{1}{T} = A + B \cdot \ln\frac{R_T}{R_0} + C \cdot (\ln\frac{R_T}{R_0})^3 \text{ [42]} \qquad (5)$$

wobei R_0 = 1 Ω, R_T der Widerstand bei der Temperatur T und A ,B ,C Fitparameter sind.

Um Fehler zu vermeiden, die durch Joulesche Wärme (I^2R) verursachte Selbsterwärmung der Thermistoren entstehen können sollte der Stromfluss durch die Thermistoren so gering wie möglich sein, wie in Abschnitt 2.7.2.2 gezeigt wird.
Die Firma BetaTHERM bietet die Sensoren mit nominalen Widerständen bei 25 °C im Bereich von 2,2 KΩ bis 1 MΩ an. Aufgrund der exponentiellen Zunahme der Widerstände bei tiefen Temperaturen, ist besonders bei Thermistoren mit hohen Widerständen die Vermeidung von parasitären Widerständen ein schwieriges Unterfangen. Diese Fehler werden beispielsweise durch den Isolationswiderstand der Kabel und durch die sehr geringe Leitfähigkeit der Wärmeleitpaste verursacht. Beträgt beispielsweise der Widerstand eines Thermistors bei 25°C 1 MΩ, so steigt dieser bei –60 °C auf 0,6 GΩ an. Werden hingegen Thermistoren eingesetzt, die nur einen niedrigen Widerstand aufweisen, so wird aufgrund des dadurch resultierenden hohen Messstroms das Messergebnis durch die Selbsterwärmung stark verfälscht, siehe auch Abschnitt 2.7.2.2. Aus diesem Grund wurde ein Thermistor vom Typ 30K6A1 ausgewählt, der bei 25°C einen Widerstand von 30 KΩ aufweist und bei –60 °C nur auf 5,4 MΩ ansteigt. Dieser Widerstand stellt einen guten Kompromiss hinsichtlich der Selbsterwärmung und den parasitären Widerständen dar. Für diesen Thermistor gibt der Hersteller die Steinhart-Hart-Parameter wie folgt an: A=1,068981×10-3 1/K; B×2,120700x10-4 1/K; C×9,019537x10-8 1/K. Aufgrund seiner geringen Abmessungen besitzt dieser Sensor eine sehr kleine Ansprechzeit von 1 s in gerührten Lösungen [40], wodurch eine zeitlich hochaufgelöste Temperaturmessung möglich wird. Die Spitze des Glasrohres wurde mit Wärmeleitpaste gefüllt, um eine möglichst schnelle Wärmeübertragung zu gewährleisten.

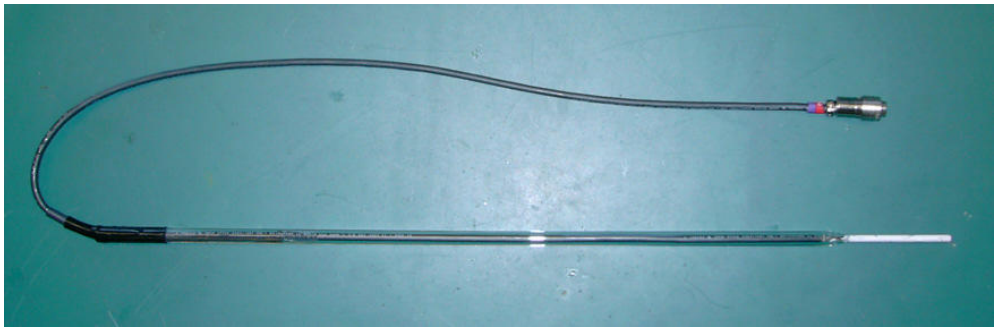


Abbildung 12 Messfühler zur Temperaturmessung

Um die Thermistoren vor den Messlösungen zu schützen, sind sie in ein 47cm langes Glasrohr eingebettet, welches mit Gießharz verschlossen ist, siehe Abbildung 12. Über eine Leitung vom TYP AWM 2464/TU61 der Firma Metrofunk (Berlin), bestehend aus einer paarverdrillten Kupferlitze, welche doppelt abgeschirmt ist, sind die Thermistoren über einen PRC05 Stecker mit dem Messgerät verbunden. Diese aufwändigen Kabel sind notwendig, um die Messleitungen vor hochfrequenten Störungen abzuschirmen.

2.7.2.2 Spannungsteiler

Für Präzisionswiderstandsmessungen werden normalerweise Brückenmethoden wie zum Beispiel Wheatstone-, Kohlrausch- oder Transformatorbrücken eingesetzt. Sollen die Messdaten mit einem Computer erfasst werden, so muss auf motorisierte Brücken zurückgegriffen werden. Da aber diese Brücken mit einer Abtastrate von 0,5 Hz oder schlechter für die in dieser Arbeit beschriebenen Anwendungsbereiche viel zu langsam sind, können sie nicht verwendet werden. Als Alternative bieten sich selbstabgleichende Brücken an. Beispiele für solche Systeme werden von Caleb et al. [43] und von White et al. [44] beschrieben. Dank des großen Temperaturkoeffizienten der Thermistoren kann aber eine viel einfachere Methode zur Widerstandsmessung eingesetzt werden, als dies bei Platinwiderständen der Fall ist [21]. Dieser Spannungsteiler, der sich aus dem Thermistor zur Temperaturmessung R_T und dem Referenzwiderstand R_1 zusammensetzt, ist in Abbildung 13 auf der linken Seite abgebildet.

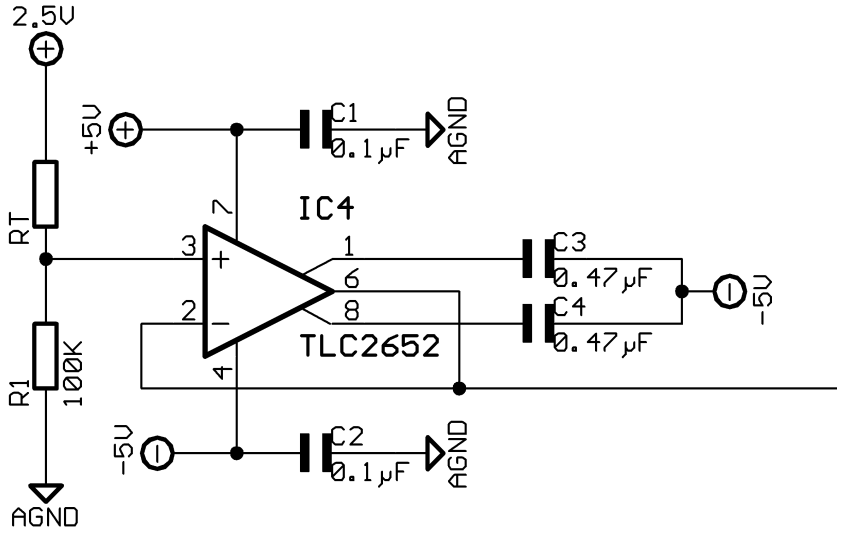


Abbildung 13 Spannungsteilerschaltung zur Temperaturmessung mit Impedanzwandler

Durch diesen Spannungsteiler wird eine temperaturabhängige Spannung erzeugt, die mittels eines Analog/Digital-Wandlers leicht gemessen werden kann. Diese Spannung U, welche am Referenzwiderstand abfällt, kann nach folgender Gleichung, die sich aus dem Ohmschen Gesetz und den Kirchhoffschen Maschenregeln ergibt berechnet werden.

$$U = \frac{R_1}{R_1 + R_T} U_{REF} \tag{6}$$

Die Temperaturabhängigkeit des Spannungsteilers ergibt sich, wenn die Steinhart-Hart-Gleichung (5) nach R_T aufgelöst wird und das Ergebnis in Gleichung (6) eingesetzt wird, woraus folgender Zusammenhang resultiert:

$$U = \frac{R_1}{\exp\left[\sqrt[3]{-\left[\frac{\left(A-T^{-1}\right)^2}{\frac{C}{2}} + \sqrt{\left[\frac{\left(A-T^{-1}\right)^2}{C}\right]^2 + \left[\left(\frac{B}{C}\right)^3\right]}\right]} + \sqrt[3]{-\left[\frac{\left(A-T^{-1}\right)^2}{\frac{C}{2}} - \sqrt{\left[\frac{\left(A-T^{-1}\right)^2}{C}\right]^2 + \left[\left(\frac{B}{C}\right)^3\right]}\right]} + R_1\right]} \cdot U_{ref} \tag{7}$$

Durch die Umwandlung der Widerstandsinformation wird auch noch eine grobe Linearisierung erreicht [45], [46] und [47]. Da hier die Korrelation zwischen der Temperatur und der gemessenen Spannung mit einem Computer durchgeführt wird, stellt diese Linearisierung auf den ersten Blick keine wichtige Verbesserung dar. Bei genauerer Betrachtung ergibt sich aber dank dieser Eigenschaft eine sehr gute Ausnutzung des dynamischen Bereichs des nachgeschalteten Analog/Digital-Wandlers.
Die Temperatur T des Thermistors lässt sich aus der gemessenen Spannung U berechnen, wenn Gleichung (7) nach T aufgelöst wird:

$$T = \frac{1}{A + B \cdot \ln \frac{\frac{R_1 \cdot U_{REF}}{U} - R_1}{R_0} + C \cdot \left(\ln \frac{\frac{R_1 \cdot U_{REF}}{U} - R_1}{R_0} \right)^3} \tag{8}$$

In Abbildung 14 ist die Abhängigkeit der Ausgangsspannung des Spannungsteilers von der Temperatur und dem Referenzwiderstand dargestellt. Wie dieser Abbildung entnommen werden kann, nimmt die Steigung der Spannung U(θ) mit zunehmender Temperatur ab, wodurch auch die Auflösung der Temperaturmessung reduziert wird. Sollte das Thermometer im Gegensatz zu der hier gezeigten Anwendung für hohe Temperaturen eingesetzt werden, so sollten der Thermistor R_T und der Referenzwiderstand R_1 im Spannungsteiler die Plätze tauschen, wodurch sich für diesen Anwendungszweck eine deutlich geeignetere Kennlinie ergibt [21].

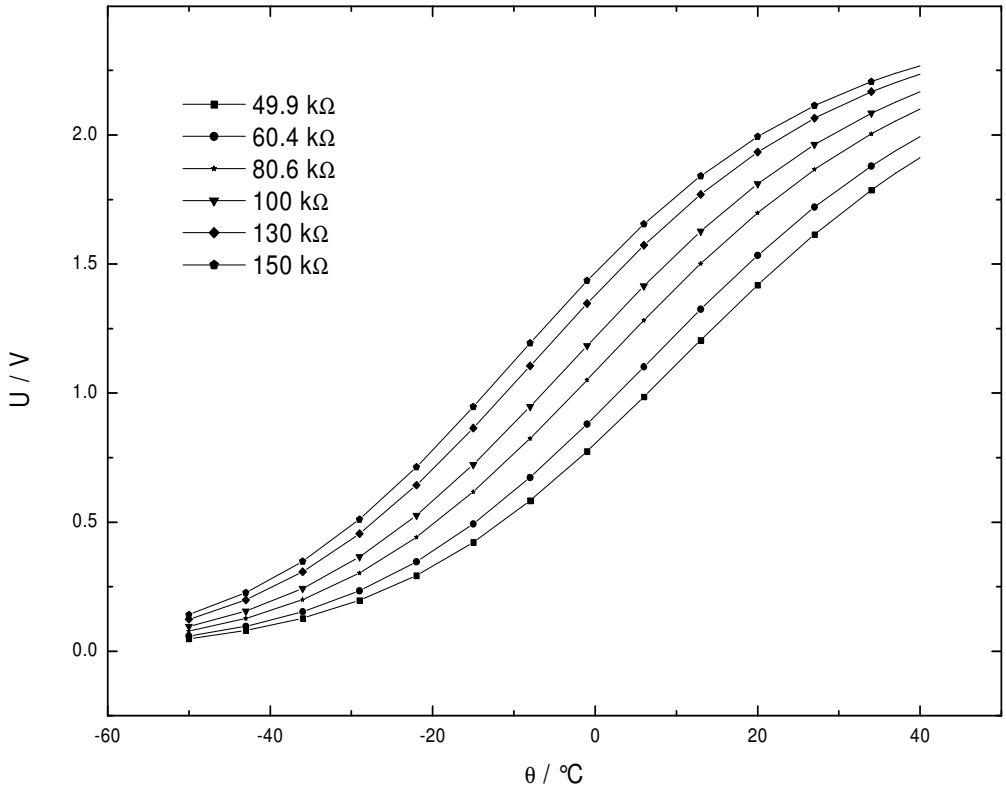


Abbildung 14 Temperaturabhängigkeit der Ausgangsspannung des Spannungsteiler in Abhängigkeit des Referenzwiderstandes

Der Spannungsteiler sollte so ausgelegt sein, dass der dynamische Bereich des Analog/Digital-Wandlers, der durch die Spannungsreferenz vorgeben wird, optimal genützt wird. Da, wie in Abschnitt 2.7.2.4 beschrieben, sowohl der Analog/Digital-Wandler wie auch die Spannungsteiler aus der gleichen Spannungsreferenz versorgt werden, wird der dynamische Bereich im besten Fall nur durch den Spannungsteiler bestimmt. Die höchste Temperaturauflösung wird erreicht, wenn die Steigung des in Abbildung 14 gezeigten Graphen maximal ist. Wie diesem Graph entnommen werden kann, ist die Steigung über den gesamten Bereich bei mittleren Werten von R_1 am größten, wodurch sich auch eine optimale Ausnutzung des dynamischen Bereichs des Analog/Digital-Wandlers ergibt.

Der Widerstand R_1 erzeugt nicht nur einen geeigneten Spannungsabfall, sondern er begrenzt auch den Stromfluss durch den Thermistor und somit auch seine Eigenerwärmung. In der Schaltung von O´Grady [48] wird ein Spannungsteiler verwendet, bei dem ein dritter Thermistor für die Strombegrenzung eingesetzt wird. Dieser Widerstand erweist sich nicht nur aufgrund der Reduktion der Ausgangsspannung als nachteilig. Ebenso störend sind die zusätzlichen Fehler, die durch diesen Widerstand hervorgerufen werden.

Um Fehler, die durch Selbsterwärmung der Thermistoren verursacht werden, so klein wie möglich zu halten, muss die Leistung, die den Thermistor erwärmt, weitgehendst reduziert werden. Diese Leistung lässt sich mit Gleichung (9) berechnen.

$$P = R_T \cdot \left(\frac{U_{REF}}{R_T + R_1} \right)^2 \tag{9}$$

Abbildung 15 zeigt die Leistungsabgabe der Thermistoren im Temperaturbereich von –50 °C bis +40 °C bei verschiedenen Werten für R_1. Wie erwartet, wird eine niedrige Leistungsabgabe mit großen Widerständen erreicht, die auch bei mittleren Widerstandswerten noch hinreichend gering ist. Die Leistungsabgabe der Thermistoren weist hier ein Maximum auf. Dieser Höchstwert wird durch zwei gegenläufige Effekte verursacht. Zum einen ist bei konstantem Strom die Leistung an einem Widerstand umso größer, je größer der Widerstand ist. Da aber hier der Stromfluss durch den Widerstand begrenzt wird, ergibt sich der in Abbildung 15 gezeigte Verlauf.

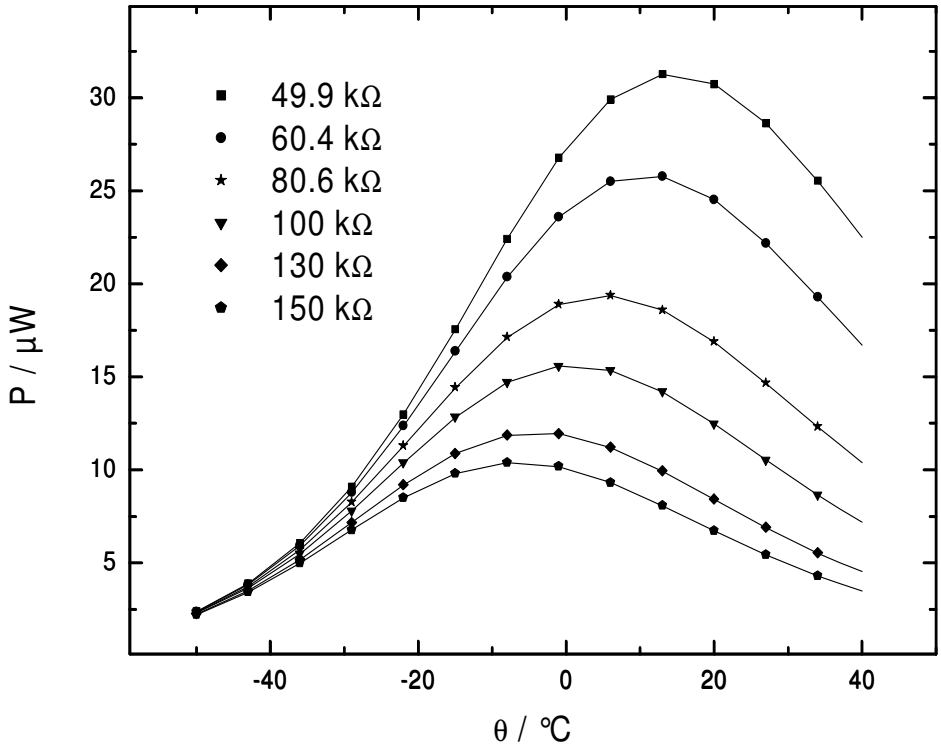


Abbildung 15 Joulesche Leistung der Thermistoren bei verschiedenen Temperaturen und verschiedenen Werten für R_1

Ein weiterer wichtiger Parameter, der die Genauigkeit des Messgeräts bestimmt, ist sein Rauschen, das auch durch die Wahl des Referenzwiderstandes beeinflusst wird. Um diesen Einfluss zu testen, wurden 12 Präzisionswiderstände von Typ Econistor der Firma Rhopoint mit Werten im Bereich von 50 kΩ bis 150 kΩ für R_1 eingesetzt. Als Thermistor wurde für diese Messungen der 30K6A1 der Firma BetaTHERM verwendet und mit der in Kapitel 2.5.3 beschriebenen Thermostatenanlage auf –50,00 °C, +5,00 °C und +40,00 °C thermostatisiert. Bei diesen Temperaturen wurde dann für

jeweils 2000 s das Rauschen bestimmt[4]. Die Ergebnisse dieser Messungen sind in Abbildung 16 dargestellt, in der die statistische Streuung der gemessenen Temperatur gegen den Widerstand aufgetragen ist.

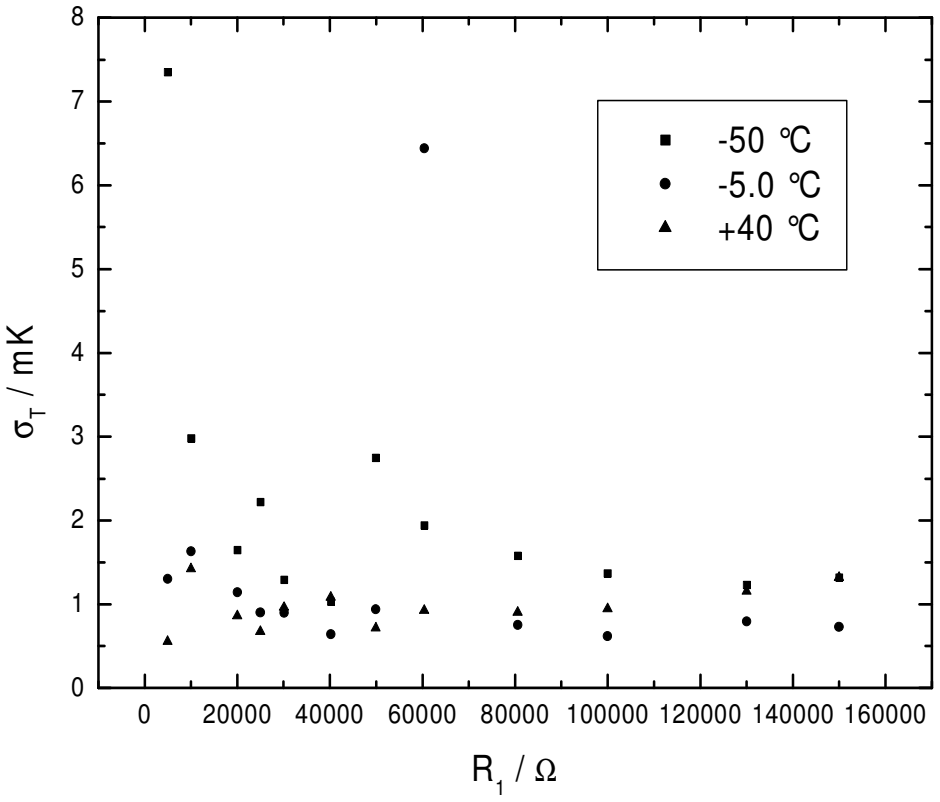


Abbildung 16 Einfluss des Referenzwiderstandes auf das Rauschen des Thermometers

Mit zunehmenden Widerstandswerten wird eine Abnahme des Rauschens beobachtet. Daher eignet sich ein Wert von 100 kΩ als sehr gut für den Referenzwiderstand, da auch bei diesem Wert noch eine sehr gute Auflösung mit einer geringen Selbsterwärmung verknüpft ist.
Damit die Temperaturmessung nicht durch Schwankungen der Luft, die das Messgerät umgibt, verfälscht wird, muss der Festwiderstand R_1 einen möglichst kleinen Temperaturkoeffizienten aufweisen. Darüber hinaus sollte sein Langzeitdrift sehr klein sein, so dass sich möglichst lange Kalibrierintervalle ergeben. Eine sehr geringe Toleranz ist nicht erforderlich, da anfängliche Abweichungen vom Sollwert in der Kalibrierung verschwinden. Als Festwiderstand R_1 wird daher ein 100 kΩ Präzisionswiderstand von Rhopoint, ein Econistor, eingesetzt. Dieses Bauteil weist dank seines internen Aufbaus, bei dem die beiden Übergänge zwischen den Kupferanschlussleitungen und den

[4] Für diese Messung wurde eigens ein 2-Kanalprototyp entwickelt, der mit dem 30-Kanalthermometer, bis auf die fehlenden Messkanäle, identisch ist. Bei diesem Gerät sind Referenzwiderstände steckbar ausgeführt und können dadurch leicht ausgetauscht werden. Dieser Prototyp kann anstelle des Analogteils des 30-Kanalthermometers an den Digitalteil dieses Thermometers angeschlossen werden und simuliert vollständig die Funktionen des 30-Kanalthermometers.

Widerstandsdrähten maximal 2 mm von einander entfernt sind, wodurch eine Minimierung der Temperaturunterschiede zwischen den beiden Übergängen und somit ihrer Thermospannung erreicht wird, einen sehr kleinen Temperaturkoeffizienten von 3ppm/K auf [49].
Ein positiver Nebeneffekt dieser Spannungsteilerschaltung ergibt sich dadurch, dass der entsprechende Eingang des A/D-Wandlers durch den Widerstand R_1 auf das Massepotenzial gezogen wird, so dass dieser Eingang auf einem definierten Potenzial liegt, und dadurch Störungen der anderen Messkanäle vermieden werden, welche durch Kanalübersprecher hervorgerufen werden, wenn der Thermistor nicht mit dem Gerät verbunden ist.

2.7.2.3 Impedanzwandler

Würde der Analog/Digital-Wandler direkt an den Spannungsteiler angeschlossen, so würde seine Ausgangspannung durch den kleinen Innenwiderstand des Analog/Digital-Wandlers, der 39 kΩ beträgt [50], belastet und damit das Messergebnis verfälscht. Um dieses Problem zu beheben, wird zwischen dem Spannungsteiler und dem Analog/Digital-Wandler ein Spannungsfolger, auch Impedanzwandler, geschaltet. Für die Schaltung, die in Abbildung 13 auf der rechten Seite gezeigt ist, wurde aufgrund seines kleinen Offsetfehlers ein chopperstabiliserter Operationsverstärker vom Typ TLC2652A der Firma Texas Instruments Corp. (Austin) eingesetzt. Dieses Bauteil verfügt über eine hohe Eingangs- und über eine niedrige Ausgangsimpedanz, so dass der Spannungsteiler nicht belastet wird. Da dieser Baustein mit einer Taktfrequenz von 450 Hz [51] betrieben wird, welche als ganzes Vielfaches der Filterfrequenz des Eingangsfilters des Analog/Digital-Wandlers von diesem sehr stark unterdrückt wird, stellt die Verwendung eines getakteten Operationsverstärkers, der aufgrund der hohen Anforderungen hinsichtlich seiner Offsetspannung nötig ist, kein Problem dar.

2.7.2.4 Spannungsreferenz

Die Spannungsteiler werden mit 2,5 V versorgt, die von einer Maxim MAX6325CPA Spannungsreferenz erzeugt werden. Diese Spannungsquelle stellt eine Ausgangsspannung von (2,500 ±0,001) V mit einem Temperaturkoeffizienten von 1 ppm und einem sehr geringen Rauschen von 1,5 μVp-p (0,1 bis 10 Hz) zur Verfügung [52]. Die Beschaltung dieses Bausteins entspricht der Schaltung der Spannungsreferenz des Digital/Analog-Wandlers des Zyklisiergeräts, siehe Abbildung 39 rechts unten. Fehler, die durch die Spannungsreferenz erzeugt werden, wie zum Beispiel eine Drift, wirken sich nicht auf die Messung aus, da dieselbe Referenz auch die Analog/Digital-Wandler mit Spannung versorgt. Daher werden durch den Analog/Digital-Wandler keine Spannungen, sondern das Verhältnis zwischen dem Widerstand des Thermistors und dem Referenzwiderstand bestimmt, welches nicht durch die Spannung der Referenzspannung beeinflusst wird [53].

2.7.2.5 Analog/Digital-Wandler

Das Herz der Schaltung stellen die vier Analog/Digital-Wandler dar, an die jeweils acht Messfühler angeschlossen sind. Die charakteristische Kenngröße dieser Wandler stellt seine Auflösung dar. So wie in Abschnitt 2.7 beschrieben, beträgt die geforderte Temperaturauflösung 1 mK. Soll diese mit der in Abschnitt 2.7.2.2 gezeigten Schaltung erreicht werden, so ist mindestens eine Auflösung von 21-BIT für den Analog/Digital-Wandler nötig [21]. Typischerweise werden Sigma-Delta Analog/Digital-Wandler in Anwendungen eingesetzt, wenn eine sehr hohe Auflösung benötigt wird.

Aufgrund ihres Funktionsprinzips, das auf der Integration der Messegröße basiert, weisen diese Bausteine eine exzellente Störunterdrückung für hochfrequente Störungen auf. Ihre dadurch bedingte langsame Messrate stellt bei der in dieser Arbeit beschriebenen Anlage keinen Nachteil dar, da hier die Datenrate bei einer Messung pro Sekunde liegt. Um die Anzahl der nötigen Bauteile zu reduzieren ist es günstig, einen Analog/Digital-Wandler mit mehreren Messkanälen einzusetzen, da dadurch externe Multiplexer eingespart werden können.
Bei dem hier beschriebenen Thermometer wird ein Analog/Digital-Wandler ADS1241E der Firma Texas Instruments Inc. (Dallas) eingesetzt. Dieser 24-BIT Wandler verfügt über acht Eingangskanäle und ist mit einem integrierten programmierbaren Verstärker (PGA) ausgestattet. Mit diesem Verstärker können die Ausgangsspannungen der Spannungsteiler um den Faktor 1 bis 128 verstärkt werden. Im Gegensatz zum AD77, der in der von O'Grady [48] beschriebenen Schaltung eingesetzt wird, hat dieser Wandler einen deutlich geringeren Offsetfehler und eine deutlich kleinere Drift des Offsets. Da diese beiden Abweichungen den Hauptanteil der Ungenauigkeiten des Analog/Digital-Wandlers ausmachen (siehe Fehlerrechnung, Kapitel 11.3.1.1), stellt dies eine deutliche Verbesserung gegenüber der Schaltung von O'Grady dar. Die Datenrate des Wandlers beträgt maximal 15 Messungen pro Sekunde, kann aber zugunsten der Messgenauigkeit reduziert werden [50].
Da das Umschalten zwischen den einzelnen Messkanälen und der damit verbundenen Neukalibrierung des Wandlers viel Zeit benötigt, wird bei dieser Anwendung die schnellst mögliche Datenrate eingesetzt. Alle vier Analog/Digital-Wandler sind über ein SPI Interface[5] mit dem Mikrokontroller verbunden. Um einen synchronen Betrieb der vier Analog/Digital-Wandler sicherzustellen, wurde dieses Interface so modifiziert, dass sämtliche Leitungen die Daten vom Mikrokontroller zum Analog/Digital-Wandler übertragen, die wie SCLK, CS, DIN und DSYNC, je nur einmal vorhanden sind und gleichzeitig an alle vier Wandler angeschlossen sind. Im Gegensatz dazu ist jede Datenleitung, die Daten vom Analog/Digital-Wandler zum Mikrokontroller überträgt, separat für jeden Analog/Digital-Wandler vorhanden. Durch diese Ansteuerung können alle vier Wandler simultan ausgelesen werden.

2.7.2.6 Zusätzliche Schaltkreise

Echtzeituhr

Um eine stabile Zeitbasis zu erhalten, die unabhängig von der Rechnerzeit ist, enthält die Schaltung eine eigene Echtzeituhr. Für diese Aufgabe wird ein DS1305 von Dallas Semiconductor eingesetzt. Bei diesem Baustein handelt es sich um eine Echtzeituhr mit einem Takt von einer Sekunde [54]. Eine detaillierte Beschreibung dieses Bausteins und seiner Ansteuerung findet sich in der Diplomarbeit des Autors [21].

Interface

Über die Interfaceschaltung wird das Thermometer entweder direkt an den Steuerrechner oder an das Leitfähigkeitsmessgerät angeschlossen. Dieses Interface besteht aus Pufferbausteinen vom Typ 74LS245, welche die Signalflanken aufbereiten und somit eine sichere Kommunikation gewährleisten. Zusätzlich dazu sind die Datenleitungen, über welche die Kommunikation mit den Analog/Digital-Wandlern abgewickelt wird, über kapazitive Koppelbausteine vom Typ IOS508 galvanisch vom Digitalteil getrennt, um aus diesem resultierende Störungen abzutrennen.

[5] Bei dem von der Firma Motorola definierten SPI Interface werden die Sende- und Empfangsdaten auf getrennten Datenleitungen übertragen. Für das Taktsignal und die Masse gibt es je eine separate Leitung.

Digital/Analog-Wandler

Für die Ansteuerung des Zahnradrührwerkes und für einen Heizer, mit dem die Temperatur des Thermostaten geregelt werden kann, sind zwei Digital/Analog-Wandler vorgesehen. Der Aufbau dieser Wandler ist identisch mit dem des Digital/Analog-Wandlers der Batterietestsysteme, die in Abschnitt 5.2.2 beschrieben sind.

Netzteil

Da bei dem 8-Kanalprotoytpen der Hauptanteil des Rauschens des Messgeräts durch das Netzteil verursacht wird [21], war es ein wichtiges Ziel, dieses Rauschen soweit wie möglich zu reduzieren. Bei diesem Prototypen wurde ein aktiver Schaltregler für die Erzeugung der bipolaren Spannungsversorgung eingesetzt. Da diesem Bauteil aufgrund seiner Betriebsart der Ausgangspannung immer eine hochfrequente Störspannung überlagert ist, wurden bei dem 30-Kanalthermometer diese Bauteile nicht mehr verwendet. Stattdessen wurde für dieses Gerät ein aufwändig stabilisiertes und gefiltertes Netzteil aufgebaut, in dem klassische Transformatoren und Linearregler eingesetzt wurden. Mit diesem Netzteil, dessen Schaltplan in Anhang 11.5.1.1 dargestellt ist, konnte eine Reduzierung des Rauschens des Netzteils um den Faktor 25 erreicht werden.

2.7.3 Kalibrierung

Bei der Kalibrierung werden jeweils die einzelnen Messfühler mit den zugehörigen Messkanälen verbunden; jeder Fühler wird zusammen mit seinem Messkanal kalibriert. Für die Kalibrierung wird die in Abschnitt 2.5.3 beschriebene Thermostatenanlage eingesetzt. Die Kontrolle der Temperatur des Ölbads erfolgt dabei mit einem Thermometer vom Typ F-250 MkII der Firma Automatic Systems Labs. (Milton Keynes). Dieses Thermometer und sein Messfühler wurden von WYCO (Romsay) nach den Standards des United Kingdom Accreditation Service kalibriert. Dank dieser Kalibrierung wird eine Messunsicherheit von 25 mK erreicht [55].

Die Kalibrierung wurde im Bereich von –60,00 °C bis +40,00 °C in 10 °C durchgeführt, wobei bei jeder Temperatur die Spannungen, die an den Spannungsteilern anliegen, für 1000 s mit der Kalibrierfunktionalität der Steuersoftware gemessen wurden. Zur Bestimmung der Fitparameter A,B,C und U_{REF} wird Gleichung (8) mittels nichtlinearer Ausgleichsrechung an die erhaltenen U(T) Funktionen angepasst. Die Anpassung des Widerstands R_1 versagt mit dieser Methode, da keine stabile Abbruchsbedingung gefunden wird, wenn auch dieser Parameter frei gewählt wird. Daher wird dieser Wert auf seinen nominalen Wert von 100,0 kΩ gesetzt.

Die nach dieser Methode erhaltenen Kalibrierparameter der einzelnen Messfühler sind im Anhang aufgeführt.

2.7.4 Abschätzung der Messgenauigkeit

Die Messgenauigkeit des Thermometers wurde mit zwei Methoden untersucht. Zum einen wurde eine Fehlerrechnung nach der Methode der Größtfehlerabschätzung durchgeführt, um die Beträge der einzelnen Bauteile des Thermometers abschätzen zu können. Die Berechnungen wurden vor dem Bau des Messgerätes auch dazu verwendet, um geeignete Bauteile auszuwählen. Bei diesen Berechnungen wurden zunächst die Fehler der Thermistoren vernachlässigt, so dass nur die Eigenschaften der Schaltung untersucht wurden. Für den ungünstigsten Fall beträgt die maximale berechnete Abweichung 6 mK (siehe auch Anhang 11.3.1.1). In einem zweiten Schritt wurde noch die Messgenauigkeit und die Alterungsstabilität der Messfühler in Kombination mit dem Messgerät experimentell bestimmt. Im Bereich von –50 °C bis +10 °C liegt die maximale Abweichung bei nur

30 mK (siehe auch Anhang 11.3.1.3). In diesem Messbereich ist also das hier beschriebene 30-Kanalthermometer ebenso genau wie das ASL Referenzthermometer. Besonders hervorzuheben ist, dass diese Präzision auch noch nach sechs Monaten Benutzung erreicht wird. Da bei der Fehlerrechnung das Eigenrauschen der Bauteile nicht berücksichtigt werden konnte, wurde dieses ebenfalls experimentell bestimmt. Im schlechtesten Fall liegt das Rauschen bei 2,5 mK, wobei meist Werte von deutlich kleiner als 1 mK erreicht werden, siehe auch Anhang 11.3.1.2.

2.7.5 Eingesetzte Methoden zur Steigerung der Messgenauigkeit

Die Verbesserung der Messgenauigkeit des 30-Kanalmessgeräts gegenüber dem 8-Kanalprototypen setzte an zwei Punkten an. Zum einen werden bei dem 30-Kanalthermometer Thermistoren mit einer anderen Kennlinie und andere Referenzwiderstände eingesetzt, wodurch neben der Selbsterwärmung auch das Eigenrauschen reduziert werden konnte. Durch diese neuen Komponenten konnte darüber hinaus auch noch die Auflösung des Systems verbessert werden. Um diese Bausteine auswählen zu können, wurden umfangreiche Berechnungen und Messungen durchgeführt (siehe Anhang 11.3.1).
Die Hauptursache für die Messfehler des 8-Kanalprotypen lag im Rauschen dieses Geräts, das in der Größenordnung von bis zu 30 mK liegt. Um dieses Rauschen zu vermindern, wurde eine Vielzahl von Maßnahmen durchgeführt. Als am effektivsten erwies sich dabei der Einsatz eines sehr gut stabilisierten Netzteils, welches für dieses Gerät eigens entwickelt wurde. Maßnahmen wie der Einsatz von Netzfiltern und abgeschirmten Gehäusen und Messleitungen sorgen für eine Abschirmung vor elektromagnetischen Interferenzen. Insbesondere wurden im Gegensatz zu den beim 8-Kanalprototypen eingesetzten Koaxialkabeln [21], bei denen das Messsignal auch über den ungeschirmten Außenleiter geführt wird, Messleitungen benutzt, bei denen die paarverdrillten Innenleiter von einer doppelten Abschirmung umgeben sind. Diese Abschirmung liegt auf dem Massepotenzial des Messgeräts, wodurch eine sehr gute Abschirmung erreicht wird. Durch die Trennung des Analog- und Digitalteils der Schaltung wird eine Störung der Messung, die durch den Digitalteil verursacht werden könnte, vermieden. Besonders hervorzuheben ist dabei die komplette Trennung der Potenziale beider Module, die durch kapazitive Koppelbausteine und durch zwei separate Netzteile erreicht wird. Einen nicht zu unterschätzenden Anteil an den Verbesserungen hat auch das mit Hilfe von Eagle der Firma Cadsoft (Pleiskirchen) optimierte Platinenlayout, wodurch eine weitere deutliche Reduzierung von Störungen erreicht wird. Durch die Kombination dieser Methoden ist es gelungen, das Rauschen des Messgeräts um den Faktor 12 zu reduzieren.

2.8 Das Multikanalleitfähigkeitsmessgerät

Für die Bestimmung der Abhängigkeit der Leitfähigkeit von der Temperatur, zur Untersuchung von Hydrolysekinetiken und als zusätzliche Detektionsmöglichkeit für die Bestimmung von Phasenübergängen wurde ein Leitfähigkeitsmessgerät mit vielen Messkanälen benötigt, das mit dem Thermometer gekoppelt werden kann. Der Leitfähigkeitsbereich, den dieses Gerät abdecken soll, liegt in der Größenordnung von 0,1 mS/cm bis 15 mS/cm. Der Messfehler soll dabei kleiner als 1% sein. An die Auflösung des Messgeräts werden deutlich höhere Anforderungen gestellt, da es als zusätzliche Detektionsmöglichkeit für die Bestimmung von Phasenübergängen eingesetzt wird. Auch bei der Untersuchung von Kinetiken ist eine hohe Auflösung von großer Wichtigkeit. Daher

ist eine Auflösung von 0,1 µS/cm notwendig. Bei dem hier beschriebenen Messgerät handelt es sich um eine Weiterentwicklung des 30-Kanalthermometers, bei dem statt Thermistoren Leitfähigkeitsmesszellen als Sensoren eingesetzt werden. Aus den in Abschnitt 2.3 aufgeführten Gründen muss im Gegensatz zur Temperaturmessung die Messung mit Wechselspannung durchgeführt werden. Daher wurde dieses Messgerät mit Sinusgeneratoren für die Erzeugung der Wechselspannung ausgestattet. Nach der Gleichrichtung mit Präzisionsgleichrichtern erfolgt dann die Messdatenerfassung in Analogie zum Thermometer, siehe Abschnitt 2.7.
Wie das Thermometer auch ist das Konduktometer in drei verschiedene Funktionsbereiche unterteilt, was auch aus Abbildung 10 ersichtlich ist. Im Einschub, der sich auf der rechten Seite befindet, ist die Stromversorgung und der Digitalteil des Leitfähigkeitsmessgeräts untergebracht, der mit dem Thermometer identisch ist. Im mittleren Einschub, an dessen Vorderseite die Messfühler angeschlossen werden, befindet sich die eigentliche Messelektronik mit den Spannungsteilern, den Gleichrichtern und den Analog/Digital-Wandlern. Die Ansteuerung des Wandlerfeldrührwerks ist auf der linken Seite des Geräts untergebracht.

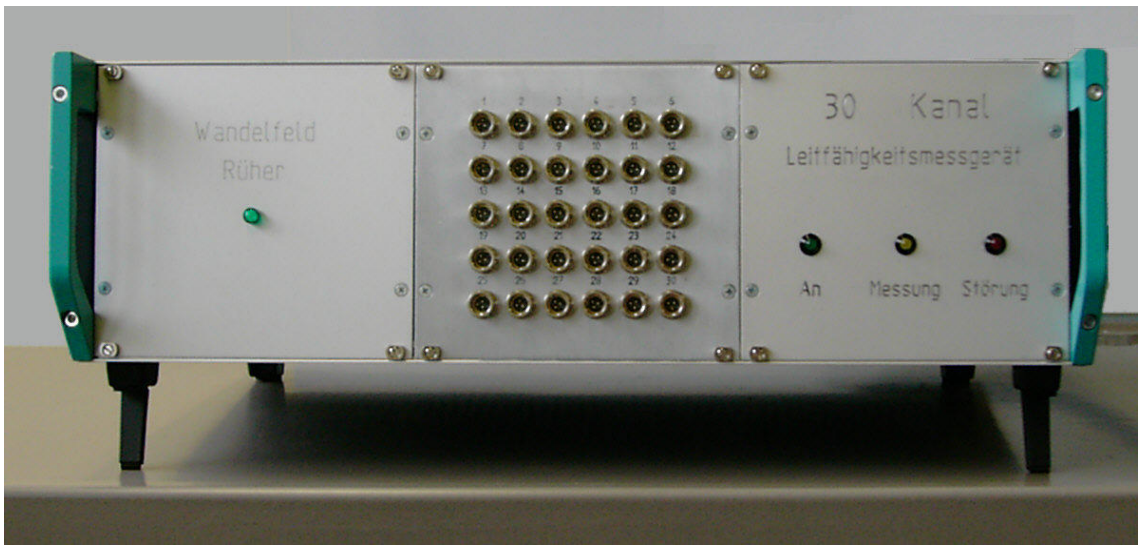


Abbildung 17 30-Kanalkonduktometer, Vorderansicht

Die Mikrokontrollerplatine, die für die Kopplung des Konduktometers und des Thermometers eingesetzt wird, ist ebenfalls in diesem Gerät untergebracht. Das Thermometer wird an der Rückseite des Leitfähigkeitsmessgerätes angeschlossen.

2.8.1 Funktionsprinzip

Für die Messung von Leitfähigkeiten werden normalerweise Messbrücken, wie sie zum Beispiel in Kapitel 4.2.2 beschrieben sind, eingesetzt. Sollen an 30-Messkanälen Leitfähigkeiten gemessen werden, so können, wie auch bei der Temperaturmessung, diese Brückenmethoden aus den gleichen Gründen nicht verwendet werden, siehe Abschnitt 2.7.1. Daher wird auch beim Leitfähigkeitsmessgerät auf eine Potentiometerschaltung zurückgegriffen. Da die Leitfähigkeitsmessungen aufgrund der in Abschnitt 2.3 aufgeführten Gründe mit Wechselspannung durchgeführt werden müssen, wird die Spannungsteilerschaltung im Gegensatz zum Thermometer mit Wechselspannung gespeist. Diese wird mit einem Sinusgenerator erzeugt, dessen Frequenz einstellbar ist. Die Ausgangsspannung der Potentiometerschaltungen wird mit Präzisionsgleichrichtern in Gleichspannungen umgeformt. Diese werden dann wie beim Thermometer auch digitalisiert und ausgewertet. Mit dieser

Methode wird die Leitfähigkeit der Messzelle festgelegt. Durch eine Kalibrierung, siehe Abschnitt 2.8.3, kann die spezifische Leitfähigkeit der Lösung bestimmt werden.

2.8.2 Beschreibung des Systems

Der einzige Unterschied zum Thermometer besteht beim Konduktometer darin, dass die Messung mit Wechselspannung anstatt mit Gleichspannung durchgeführt wird. Da beide Geräte bis auf einige Komponenten identisch sind, werden in diesem Kapitel nur die Module beschrieben, bei denen sich das Leitfähigkeitsmessgerät vom Thermometer unterscheidet. Für die Beschreibung der anderen Module, wie zum Beispiel Analog/Digital-Wandler, Spannungsreferenz und die zusätzlichen Hilfsschaltkreise sei hier auf das Kapitel 2.7.2 verwiesen.

2.8.2.1 Sinus-Generator

Zur Erzeugung dieser Wechselspannung U_{GEN}, mit der die Spannungsteiler des Leitfähigkeitsmessgeräts gespeist werden, wird ein Frequenzgenerator der Firma Maxim vom Typ MAX038 eingesetzt. Mit diesem Baustein können Sinus-, Dreieck-, Sägezahn- und Rechteckwechselspannungen im Bereich von 0,1 Hz bis 20 MHz erzeugt werden [56]. Die Wellenform dieses Generators wird durch die Beschaltung der Pins A_0 und A_1 bestimmt.

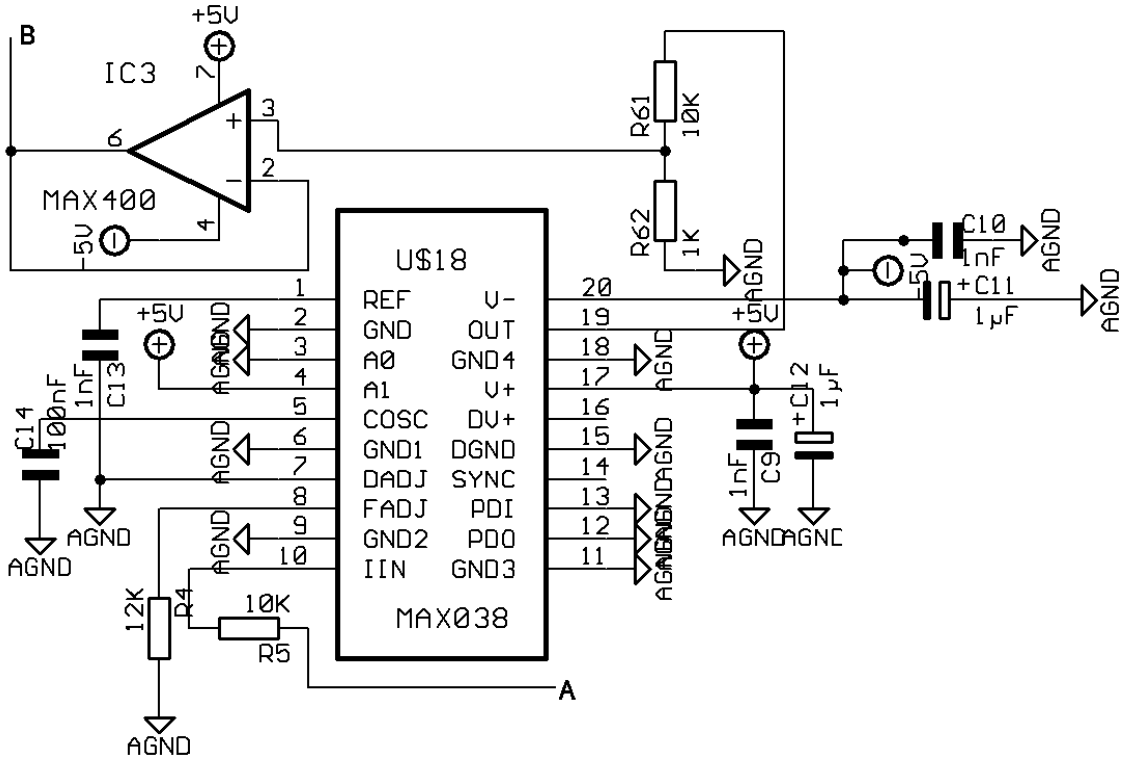


Abbildung 18 Schaltplan des Frequenzgenerators

Durch den Strom, der am Pin I_{IN} angelegt wird, durch die Spannung an Pin F_{ADJ} und durch die Kapazität des Kondensators C_{14} wird die Ausgangsfrequenz dieses Bausteins bestimmt [56]. Der Strom, der über den Pin I_{IN} eingespeist wird, kann durch die Steuerspannung, die mit dem Digital/Analog-Wandler des Messgeräts erzeugt wird, variiert werden, so dass beliebige Frequenzen im Bereich von 300 Hz bis 6 kHz eingestellt werden können. Die Amplitude dieser Wechselspannung

beträgt 1,0 V [56]. Da diese Spannung für eine Messung an Elektrolytsystemen zu groß ist, wird diese durch den Abschwächer, der aus den Widerständen R_{61}, R_{62} und dem Operationsverstärker IC_3 besteht, um den Faktor 11 reduziert (siehe auch Abschnitt 5.2.3). Über die Leitung B wird dann diese Wechselspannung in die Spannungsteilerschaltungen eingespeist, wobei immer ein Frequenzgenerator, acht Spannungsteiler versorgt, die an einen Analog/Digital-Wandler angeschlossen sind.

2.8.2.2 Messgleichrichter

Da es mit Analog/Digital-Wandlern normalerweise nicht möglich ist, Wechselspannungen zu messen, ist es nötig, die Wechselspannung in eine Gleichspannung umzuwandeln. In Gleichspannungsnetzteilen werden für diesen Zweck normalerweise Diodengleichrichter eingesetzt. Da aber die Durchbruchspannung von Siliziumdioden bei 0,6 V liegt, ist es mit diesen Gleichrichtern unmöglich, Spannungen gleichzurichten, die unterhalb dieser Spannungen liegen. Bei Spannungen über dieser Grenze tritt diese Durchbruchspannung immer als Verlust auf. Im günstigsten Fall beträgt die Amplitude der Ausgangsspannung des Spannungsteilers 90,9 mV, sie liegt also deutlich unter der Durchbruchspannung von Siliziumdioden.
Sollen also so kleine Spannungen gleichgerichtet werden, ist es erforderlich, spezielle Präzisionsgleichrichterschaltungen einzusetzen, bei denen Operationsverstärker verwendet werden. Durch die hohe Schleifenverstärkung dieser Bausteine lässt sich der Spannungsabfall an der Diode kompensieren, so dass eine Gleichrichtung auch von kleinen Spannungen möglich wird [57]. Beispiele für diese Schaltungen werden in der Literatur zum Beispiel von Horowitz und Hill [58] und Friese [57] beschrieben.
Der große Nachteil dieser Schaltungen besteht darin, dass sie sehr leicht zum Schwingen angeregt werden können, da sie sehr instabile Verstärkerschaltungen darstellen. Besonders wenn viele dieser Schaltungen zusammen auf einer Platine aufgebaut werden, stellt dieses ein großes Problem dar, weil sich dann diese Schwingkreise gegenseitig anregen können. Um diese Schwierigkeit zu vermeiden, werden in dieser Schaltung statt den Präzisionsgleichrichtern Echteffektivwert-Konverter der Firma Analog Devices Inc. (Norwood) vom Typ AD736 eingesetzt. Mit diesen Bausteinen können Wechselspannungen U_{AC} mit beliebigem Signalverlauf gleichgerichtet werden [59], wobei der Effektivwert U_{RMS} der Wechselspannung als Ergebnis geliefert wird. Dieser berechnet sich nach Gleichung (10). Die Amplitude einer sinusförmigen Wechselspannung lässt sich dann durch Multiplizieren des Effektivwerts mit der Wurzel aus 2 berechnen.

$$U_{RMS} = \sqrt{Avg(U_{AC}^2)} \tag{10}$$

Der maximale Eingangspegel dieser Bausteine liegt bei 200 mV [59] und ist somit ideal für das Leitfähigkeitsmessgerät geeignet, da bei diesem Gerät der maximal zu erwartende Pegel bei 180 mV liegt. In Abbildung 19 sind die Spannungsteilerschaltung, die aus der Messzelle R_{LF} und dem Referenzwiderstand R_{19} besteht sowie der Gleichrichter abgebildet. An Punkt A wird der Spannungsteiler mit der Wechselspannung U_{GEN} versorgt.

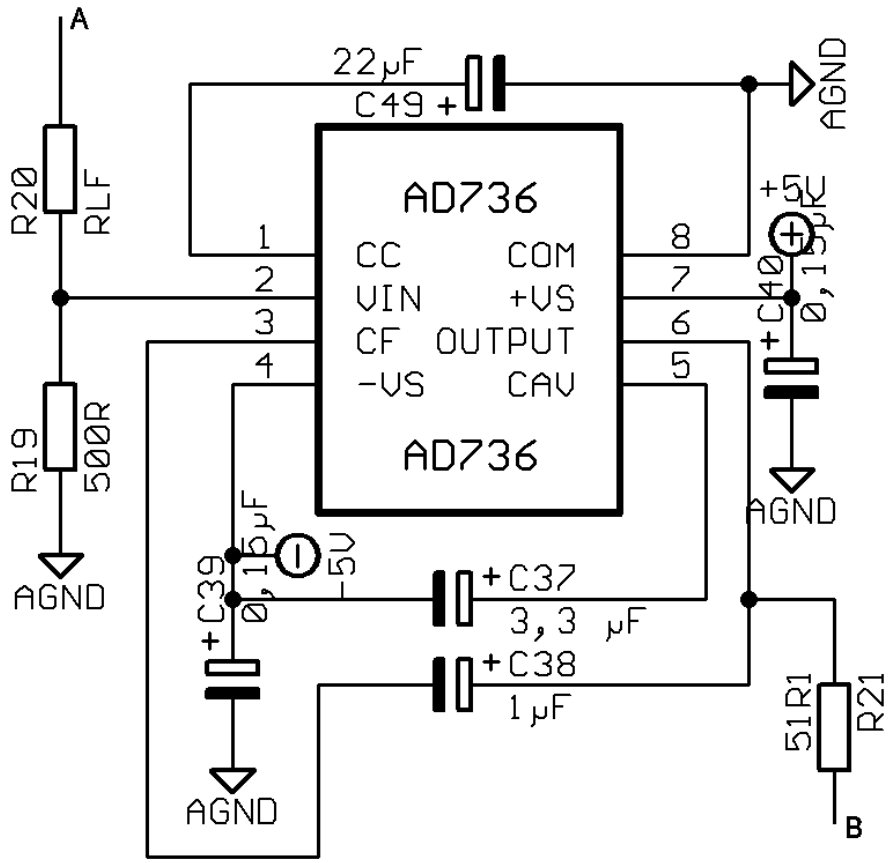


Abbildung 19 Spannungsteiler für Leitfähigkeitsmessungen und Gleichrichter für kleine Wechselspannungen

In der Mitte der Potentiometerschaltung wird die Spannung U vom Gleichrichter abgegriffen. Am Ausgang (PIN 6) steht dann die gleichgerichtete Ausgangsspannung zur Verfügung, die über einen Widerstand (R_{21}), der als Tiefpassfilter dient, zum Analog/Digital-Wandler geleitet wird. Der Kondensator C_{37} dient zur Bildung des Mittelwerts der Wechselspannung, womit der Effektivwert bestimmt wird. Das Ausgangssignal wird mit dem Filterkondensator C_{38} geglättet. Mit dem Koppelkondensator C_{49} wird ein Gleichspannungsanteil vom Eingang der Schaltung abgetrennt. Die Versorgungsspannungen dieses Bausteins werden mit den Kondensatoren C_{39} und C_{40} stabilisiert. Der Widerstand der Leitfähigkeitsmesszelle R_{LF} lässt sich anhand von Gleichung (6) und der Effektivspannung U_{RMS} bestimmen, wenn diese umgestellt wird.

$$R_{LF} = \frac{R_{19} \cdot U_{GEN}}{\sqrt{2} \cdot U_{RMS}} \tag{11}$$

Die spezifische Leitfähigkeit lässt sich bei bekannter Zellkonstante C nach Gleichung (22) berechnen, wodurch sich dann folgender Zusammenhang ergibt:

$$\kappa = \frac{1}{\frac{R_{19} \cdot U_{GEN}}{\sqrt{2} \cdot U_{RMS}}} \cdot C \tag{12}$$

2.8.3 Kalibrierung

Beim Thermometer werden die verwendeten Messfühler vor der Messung kalibriert, so dass sofort auswertbare Messwerte zur Verfügung stehen. Die Zellkonstante der Leitfähigkeitsmesszellen sind sehr stark vom Füllstand und von der geometrischen Anordnung der Elektroden zueinander abhängig. Da bei der Reinigung dieser Zellen der Rückstab entfernt wird, wobei dieser zwangsweise an die Elektroden stößt, ist es wenig sinnvoll, die in Abschnitt 2.5.1 beschriebenen Messzellen schon im voraus zu kalibrieren.
Daher werden bei diesem Messgerät anstatt der spezifischen Leitfähigkeiten von Elektrolytlösungen nur die Leitfähigkeiten der Messzellen erfasst und gespeichert. Um aus diesen Werten spezifische Leitfähigkeiten berechnen zu können, wird das in Kapitel 4.2.2 beschriebene Verfahren eingesetzt. Es wird mit den in Kapitel 4.2.2 gezeigten Messzellen die spezifische Leitfähigkeit der Elektrolytlösung bei einer bekannten Temperatur bestimmt. Anhand dieser Messung kann nun die Zellkonstante der Messzelle und damit die spezifische Leitfähigkeit über den gesamten Temperaturverlauf ermittelt werden.

2.9 Kopplung von Thermometrie und Konduktometrie

Sowohl das 30-Kanalthermometer, wie auch das 30-Konduktometer verfügen über eine 25-polige parallele Schnittstelle[6], über welche die einzelnen Bausteine dieser Geräte angesteuert werden. Über diese Schnittstelle können diese auch mit dem Steuerrechner verbunden werden. Sollen aber beide Geräte gleichzeitig eingesetzt werden, so ist es aus den in Kapitel 5.2.4 aufgeführten Gründen sinnvoll, beide Geräte über einen Mikrokontroller zu steuern und die erfassten Messdaten über eine serielle Schnittstelle an den Steuerrechner zu übertragen. Für diesen Zweck wurde eine Mikrokontrollerschaltung entworfen, deren Schaltplan in Anhang 11.5.3 aufgeführt ist. Wie auch beim Batterietestsystem, siehe Kapitel 5.2.4, wird auch hier ein ATMEGA16 der Firma Atmel (San Jose) eingesetzt.
Die Steuersoftware dieses Mikrokontrollers ATMEGA16 wurde mit Hilfe der WinAVR (GNU GCC) entwickelt. Der Quelltext dieser Software ist in Anhang 11.7.1 aufgeführt. Die Hauptroutine des Steuerprogramms besteht aus einer Endlosschleife, in der die Steuerkommandos, die der PC an das Batterietestsystem schickt, ausgewertet werden. Zur Ansteuerung des Messgeräts wird die gleiche Protokollsprache wie beim Zyklisiergerät verwendet, siehe 5.2.5. Die Übertragungsparameter sind dieselben.

[6] Diese Schnittstelle ist beim Leitfähigkeitsmessgerät nur intern vorhanden und wird nicht über das Gehäuse herausgeführt.

In Tabelle 1 sind die verfügbaren Steuerkommandos für den Mikrokontroller aufgelistet:

Kommando	Beschreibung
hglm1	LED Messen Leitfähigkeitsmessgerät an
hglm0	LED Messen Leitfähigkeitsmessgerät aus
hgtm1	LED Messen Thermometer an
hgtm0	LED Messen Thermometer aus
hgls1	LED Störung Leitfähigkeitsmessgerät an
hgls0	LED Störung Leitfähigkeitsmessgerät aus
hgts1	LED Störung Thermometer an, Thermostat aus
hgts0	LED Störung Thermometer aus, Thermostat an
hglr1	Wandelfeldrührer an
hglr0	Wandelfeldrührer aus
hgttrX	Zahnradrührwerk X=0-100
hglfX	Frequenz für Leitfähigkeitsmessung X=370 – 5000 Hz
hgtu	Steuerspannung für Heizer in µV (0 bis 2,5V)
hgta1	Uhr an
hgta0	Uhr aus
httz	Liest die Uhrzeit
hgrr	Reset der A/D-Wandler
hg32	Alle 32 Kanäle werden gemessen
hgmmX	Kanal X der Wandler wird gemessen (1 bis 8)

Tabelle 1 Steuerkommandos des Zyklisiergerätes

Das Mikrokontrollerprogramm verzweigt je nach ausgewähltem Programmpunkt und ruft die entsprechenden Routinen auf, mit denen zum Beispiel der Analog/Digital-Wandler ausgelesen wird. Die Durchführung der Messung selbst erfolgt dann vom Steuerprogramm des PCs aus, siehe Abschnitt 2.10.

2.10 Messsoftware

Für die Durchführung der Experimente wurde eine graphische Steuersoftware entwickelt, die auf den aktuellen 32-Bit Windowsbetriebssystemen wie Windows 2000 und Windows XP einsetzbar ist. Der Quelltext dieser Software, die mit Microsoft Visual Studio 6.0 entwickelt wurde, ist in Anhang 11.7.2 aufgeführt. Diese Software kann sowohl mit dem 8-Kanalprototypen, dem 30-Kanalthermometer und mit der Kombination aus 30-Kanalleitfähigkeitsmessgerät und 30-Kanalthermometer eingesetzt werden.

Neben den Funktionen für das Aufnehmen von Abkühl- und Aufheizkurven mit beiden Thermometern, gibt es auch Funktionen zur Anzeige der Temperaturen in den einzelnen Messzellen. Für Untersuchungen von Kinetiken mittels Leitfähigkeitsmessungen stehen noch weitere Funktionen zur Verfügung. Zur Kalibrierung der einzelnen Messfühler wurden Funktionen für die Kalibrierung

implementiert, mit denen die Thermistoren aller Messgeräte gegen das ASL-Thermometer kalibriert werden können. Die Auswertung kann sowohl anhand der Zeitverläufe der Temperatur wie auch anhand der Leitfähigkeit erfolgen. Zusätzlich ist es möglich, noch die Abhängigkeit der Leitfähigkeit von der Temperatur auszuwerten. Über die Steuerfunktionen können die Thermostatenanlage und die Rührwerke gesteuert werden. Über Testfunktionen besteht die Möglichkeit eine Reihe von Tests an der Anlage, wie zum Beispiel Aufnahmen von U(t) Funktionen oder auch die Ansteuerung des 2-Kanalprototypen durchzuführen.
Die erfassten Messdaten können am Ende der Messung im ASCII Format abgespeichert werden, wobei zwei verschiedene Arten von Dateien erzeugt werden. Die erste Datei enthält die gesamten Messdaten der Messung für alle Kanäle. Außerdem wird noch für jeden einzelnen Messkanal eine Messdatei erstellt, welche die Abhängigkeit von der Temperatur mit der Zeit und die der Leitfähigkeit mit der Zeit enthält. Das Dateiformat ist in [60] beschrieben. Eine Bedienungsanleitung für die Software und die Anlage wurde von Wudy [60] erstellt.

2.11 Zusammenfassung

Mit der in diesem Kapitel vorgestellten Anlage ist es möglich, eine Vielzahl von verschieden Fragestellungen zu bearbeiten. Neben der Untersuchung von temperaturabhängigen Leitfähigkeiten und der Bestimmung von Schmelz- und Erstarrungsdiagrammen, können mit ihr auch Untersuchungen von Kinetiken, wie zum Beispiel Hydrolysekinetiken durchgeführt werden.
Dank der großen Anzahl von Messstellen ist es möglich, eine große Anzahl von Systemen in kurzer Zeit zu untersuchen, wodurch eine systematische Untersuchung von Mehrkomponenten-Systemen erst ermöglicht wird. Mit dieser Anlage ist eine Zeitersparnis um den Faktor 30 gegenüber den alten Messmethoden erreichbar. Daher konnten mit ihr eine sehr große Anzahl an Messungen durchgeführt werden, siehe Kapitel 3, Kapitel 4.3 und Kapitel 6.5. Wären die alten Methoden eingesetzt worden, so hätte man für diese Messungen circa neun Mannjahre benötigt. Schon alleine diese Messungen decken ein Vielfaches des Entwicklungsaufwandes ab.
Ein Ziel bei der Entwicklung dieser Anlage war es, diese so zu gestalten, dass sie leicht in einer Kleinserie nachgebaut und an spezifische Probleme angepasst werden kann. Da sich die Anlage nicht aus gekauften Messgeräten zusammensetzt, sondern die wesentlichen Komponenten auf Eigenentwicklungen basieren, kann dies auch leicht durchgeführt werden. Bei der Entwicklung der einzelnen Messgeräte musste aufgrund der großen Anzahl von Messstellen sehr auf den Preis des einzelnen Messkanals geachtet werden. Daher konnten etablierte Techniken wie Brückenschaltungen und Platinthermometer nicht eingesetzt werden, sondern mussten durch kostengünstigere Methoden ersetzt werden, wobei aber keine Abstriche hinsichtlich der Genauigkeit gemacht werden durften. Ein Beispiel dafür stellt die Verwendung von preiswerten Thermistoren als Messfühler dar, so dass das 30-Kanalthermometer eine sehr gute Alternative zu herkömmlichen Platinwiderstandsthermometern darstellt, wobei eine Reihe von gewichtigen Nachteilen dieser Thermometer vermieden werden kann. Durch den Einsatz von Thermistoren ergibt sich nicht nur eine Kostenreduzierung um einen Faktor von fast 100, wodurch der Aufbau von Thermometern mit vielen Messkanälen erst erschwinglich wird. Im Gegensatz zu Messgeräten, bei denen Multiplexer zur Vervielfältigung der Messstellenzahl eingesetzt werden, weist dieses Messgerät aber eine deutlich höhere Messgeschwindigkeit auf, so dass die Bestimmung von Abkühl- und Aufheizkurven sowie Kinetiken bei vielen Messstellen erst sinnvoll möglich wird. Wenn Thermistoren sorgfältig auswählt und gegen ein Referenzthermometer kalibriert werden, ist es, wie gezeigt, möglich, die gleiche Stabilität und Genauigkeit wie bei Platinthermometern zu erreichen. Dabei konnte auch

lität und Genauigkeit wie bei Platinthermometern zu erreichen. Dabei konnte auch nachgewiesen werden, dass selbst nach einem halben Jahr intensiven Gebrauchs im Bereich zwischen –50 °C und +10 °C die Abweichungen bei höchstens 30 mK liegen. Aufgrund dieser Beobachtung können Thermistoren Platinthermometer für gewisse Anwendungen ersetzen, wobei sie diesen hinsichtlich Kosten, Abmessungen und Ansprechgeschwindigkeit deutlich überlegen sind. Schließlich konnte aufgrund des deutlich höheren Temperaturkoeffizienten der NTCs eine wesentlich einfachere Auswerteschaltung eingesetzt werden, wodurch sich weitere deutliche Kostenersparnisse ergeben.
Auch beim Leitfähigkeitsmessgerät konnte durch den Ersatz der Brückenschaltungen durch Potentiometerschaltungen der Kostenaufwand bedeutend reduziert werden, wobei noch eine wesentliche Zunahme der Messgeschwindigkeit erreicht wurde.
Dank des flexiblen und modularen Aufbaus der Anlage ist es leicht möglich, diese mit anderen Messmethoden zu erweitern. Ebenso kann man die Anlage für andere Einsatzbereiche einzusetzen. Ein Beispiel dafür ist die Steuerung von chemischen Prozessen. Diese Prozesse sind mit Änderungen der Temperatur und bei ionischen Bestandteilen mit Änderungen der Leitfähigkeit verknüpft. Diese Signale können benutzt werden, um die Prozesse zu steuern. Hierbei ist die hohe Zeitauflösung der im Rahmen dieser Arbeit entwickelten, intensiv getesteten und vielfach eingesetzten Anlage von Vorteil.
Gegenüber dem in der Diplomarbeit entwickelten Prototypen, mit dem es nur möglich ist, Temperaturen zu messen, und somit nicht für das oben genannte breite Einsatzspektrum verwendet werden kann, beinhaltet diese Anlage die folgenden Verbesserungen bei der Temperaturmessung:

- 30 Messkanäle
- Reduktion des Rauschens um den Faktor 12 auf unter 2,5mK
- Reduktion der Fehler durch Eigenerwärmung
- Ansteuerung der Komponenten mittels Mikrokontroller, daher einfache Erweiterungsmöglichkeit

Der Einsatz der Fehlerrechnung half bereits in der Planungsphase, angemessene Lösungen zu finden und die Anlage im Vergleich mit der in der Diplomarbeit [21] entwickelten 8-Kanal-Anlage erheblich zu verbessern. Außerdem bestätigt die Fehlerrechnung die Relevanz der gemessenen Daten.

2.12 Literaturverzeichnis

[1] M. Multerer, *persönliche Mitteilung* (2004)
[2] M. S. Ding, K. Xu, S. S. Zhang, T. R. Jow, K. Amine and G. L. Henriksen, *J. Electrochem. Soc*, **146**, 3974 (1999)
[3] M. S. Ding, K. Xu and T. R. Jow, *J. Electrochem. Soc*, **147**, 1688 (2000)
[4] R. C. Mackenzie, *Differential Thermal Analysis*, London (1970)
[5] A. Weissberger, B. W. Rossiter, *Methods of Physical Chemistry, Vol. 1, Part V*, New York (1971)
[6] M. P. Mathieu, *Bul. Soc. Chim. Belges*, **63**, 333 (1954)
[7] H. Hammer, *Eichung eines Platinwiderstandsthermometers zur Präzisionsmessung im Bereich von -60 bis 50 °C nach der Internationalen Praktischen Temperaturskala 1968*, *Diplomarbeit*, Saarbrücken (1970)
[8] M. S. Ding, K. Xu and T. R. Jow, *Journal of Thermal Analysis and Calorimetry*, **62**, 177 (2000)
[9] M. S. Ding, K. Xu and T. R. Jow, *J. Electrochem. Soc.*, **147**, 1688 (2000)
[10] M. S. Ding, K. Xu, S. Zhang and T. R. Jow, *J. Electrochem. Soc.*,, **148**, A299 (2001)
[11]H. L. Ngo, K. LeCompte, L. Hargens and A. B. McEwen, *Thermochimica Acta*, **357-358**, 97 (2000)
[12] D. H. Andrews, G. T. Kohman and J. Johnston, *J. Phys. Chem.*, **29**, 914 (1925)
[13] A. R. Glasgow, Jr., G. S. Ross, A. T. Horton, D. Enagonio, H. D. Dixon, C. P. Saylor, G. T. Furukawa, M. L. Reilly and J. M. Henning, *Analytica Chimica Acta*, **17**, 54 (1957)
[14] T. W. Richards, E. K. Carver and W. C. Schumb, *J. Am. Chem. Soc.*, **41**, 2019 (1919)
[15] M.-J. Lee, Y.-K. Chang, H. m. Lin and C.-H. Chen, *J. Chem. Eng. Data*, **42**, 349 (1997)
[16] J. R. Goates, J. B. Ott, J. F. Moellmer and D. W. Farrell, *J. Chem. Thermodyn.*, **11**, 709 (1979)
[17] J. R. Goates, J. B. Ott and A. H. Budge, *J. Phys. Chem.*, **65**, 2162 (1961)
[18] S. Schrödle, R. Buchner and W. Kunz, *Fluid Phase Equilibria,* **216**, 175 (2004)
[19] G. Jones und B. C. Bradshaw, *J. Amer. Chem. Soc.* **55**, 1780 (1933)
[20] J. Barthel, F. Feuerlein, R. Neueder und R. Wachter, *J. Sol. Chem.* **9**, 209 (1980)
[21] H.-G. Schweiger, *Entwicklung einer Präzisionstemperaturmessanlage zur schnellen Messung von Phasendiagrammen und chemische und elektrochemische Charakterisierung von Lithium-bis[oxalato(2-)]borat(1-)*, *Diplomarbeit*, Regensburg (2002)
[22] H. Hammer, *Eichung eines Platinwiderstandsthermometers zur Präzisionsmessung im Bereich von -60 bis 50°C nach der Internationalen Praktischen Temperaturskala 1968*, *Diplomarbeit*, Saarbrücken (1970).
[23] E. S. Carl, *Neue Elektrolyte in organischen Carbonatlösungen zur Anwendung in sekundären Lithium-Ionen Batterien*, *Dissertation*, Regensburg (1998)
[24] J. Barthel, R. Wachter, *Ber. Bunsenges. Phys. Chem.*, **83**, 634 (1979)
[25] M.L. McGlashan, *J. Chem. Thermodyn.* **22**, 7, (1990)
[26] *The international temperature scale of 1990,* Bureau International Des Poids Et Mesures, 1990.
[27] G.C.M Meijer, A.W. Herwaarden, *Sensor Series,* (1994)
[28] Wachter, *Eine Anordnung zur Präzisionsmessung der Temperaturabhängigkeit der Leitfähigkeit von Elektrolytlösungen*, *Dissertation*, Saarbrücken (1968)
[29] P. Sturm, *Transformator-Brücke und Platin-Normalwiderstand zur Temperaturkalibrierung*, *Diplomarbeit*, Regensburg (1996)
[30] D. R. White, *J. Phys. E: Sci. Instrum.*, **753** (1982)
[31] BIMP, *Supplementary information of the International Temperature Scale of 1990*, Bureau International Des Poids Et Mesures, (1990)

[32] BIMP, *Techniques for Approximating the International Temperature Scale of 1990,* Bureau International Des Poids Et Mesures, (1990).
[33] J.V. Nicholas and D.R. White, *Traceable Temperatures*, Chichester, 2001.
[34] J. Barthel and R. Wachter, *Thermometric Titrations*, Regensburg (1975)
[35] P. Sydenham and G. Collins, *J. Phys. E (Scientific Instruments)* **8**, 311, (1975)
[36] A. Sloman, P. Buggs, J. Molloy, and D. Stewart, *Measurement Science & Technology* **7**, 1653 (1996)
[37] V. Hans and H. Kollmeier, *Elektronik* **41**, 43 (1992)
[38] V. Hans, *Proceedings of the IEEE International Symposium on Industrial Electronics,* **1,** 29 (1992)
[39] H. Schaumburg, *Sensoren*, Stuttgart (1992)
[40] BetaTHERM Corporation Shrewsbury, *BetaCurve Interchangeable Thermistor Series I*, http://www.betatherm.com/datasheets/PC301.php?p_id=28, (2002)
[40] BetaTHERM Corporation Shrewsbury, *BetaCurve Interchangeable Thermistor Series I,* [Online] Available: http://www.betatherm.com/betacurbetachip.htm
[41] BetaTHERM Corporation, *Stability and Reliability of Thermistors,* Shrewsbury, [Online]. Available: http://www.betatherm.com/stability
[42] S. R. Hart, J. S. Steinhart, *Deep Sea Research*, **15**, 497, 1968
[43] P. Caleb, F. Wolfendale, *Impedance apparatus having lead wire resistancecompensation means*, U.S. Patent 3 584 296, June 8, 1971
[44] D.R. White, *J. Phys. E: Sci. Instrum.* 753-754 (1982)
[45] A. Khan, *International Journal of Electronics,* **59** 129 (1985)
[46] C. Molee and P. Vitale. *Electronic Design*, **26**, 90, (1978)
[47] W.R. Beakley, *Journal of Scientific Instruments,* **28**, 176 (1951)
[48] A. O'Grady, *Analog Devices Technical Note*, Analog Devices, Norwood
[49] Rhopoint Components Ltd., *Econistor 8E16&8E24,* Oxted, UK, 2002
[50] Texas Instruments Inc., *ADS1240/ADS1241 24-Bit analog-to-digital converter,* Dallas, TX, 2000.
[51] Texas Instruments Inc., TLC2652, TLC2652A, TLC2652Y, Advanced LinCMOS Precision Chopper-Stabilized Operational Amplifiers, Dallas, TX, 2001
[52] Maxim Integrated Products Inc., MAX6325/MAX6341/MAX6350 1ppm/°C, Low-Noise, +2,5V/+4,096V/+5V Voltage References, Sunnyvale,CA, 2001
[53] H. Trietley, *Radio-Electronics* vol. 56, no. 3, pp. 67, 1985
[54] Dallas Semiconductor Corp., *Datenblatt zu DS1305 Serial Alarm Real Time Clock* (1995)
[55] Automatic Systems Laboratories LTD,.*F250 Mk II Manual*, Milton Keynes
[56] Maxim Integrated Products Inc., MAX038 *High-Frequency Waveform Generator*, Sunnyvale,CA, 2001
[57] W. Friese, *Funkamateur*, **52**, 796 (2003)
[58] P. Horowitz and W. Hill, *The Art of Electronics*, Cambridge, Cambridge (1989)
[59] AD736 *Low Cost, Low Power True RMS-to-DC Converter,* Analog Devices Inc., Norwood, MA, 2000
[60] F. Wudy, *Bedienungsanleitung Phasendiagrammmessanlage,* Regensburg 2004

3 Phasendiagramme und Flüssigkeitsbereiche von Elektrolytlösungen

3.1 Zielsetzung

Der Flüssigkeitsbereich von Elektrolytlösungen ist von entscheidender Bedeutung für den Einsatz in Lithium-Ionen-Batterien. Beispielsweise wird für den Einsatz im Automobilsektor eine Tieftemperaturstabilität von bis zu –30 °C gefordert. Da nach dem „mixed solvent approach", siehe Kapitel 4.2.1.2, Mischungen verschiedener Lösungsmittel für die Herstellung geeigneter Elektrolyte eingesetzt werden, benötigt man die Kenntnis der Phasendiagramme dieser Mischungen. Im Rahmen von Schwerpunktpraktika[7] wurde von Wachter und Wudy die in Kapitel 2 beschriebene Anlage eingesetzt, um eine Vielzahl von Systemen zu untersuchen und daraus Informationen über die Stärken und Schwächen der Methode zu erhalten. Ein Ziel dieser Arbeiten war es, mit dieser Anlage Messungen sowohl an salzfreien Lösungsmittelmischungen als auch an Elektrolytmischungen durchzuführen, um den Einfluss des Leitsalzes zu untersuchen. Zusätzlich wurden noch Messungen an hochreinen Lösungsmitteln durchgeführt. Ionische Fluide stellen aufgrund ihrer hohen Eigenleitfähigkeit und ihres geringen Dampfdrucks einen vielversprechenden Ersatz für Lösungsmittel [1] dar. Daher wurden auch einige dieser Substanzen untersucht.
Obwohl von Jow et al. [2] bis [5] bereits einige binäre Lösungsmittelgemische aus dem Bereich der organischen Carbonate vermessen wurden, war es ein Ziel, diese Messungen zu wiederholen, um einen Vergleich zwischen beiden Methoden zu erhalten. Zusätzlich dazu wurde noch die Zusammensetzung der eutektischen Mischung mittels ^{1}H-NMR Spektroskopie untersucht, um eine unabhängige Aussage über die Zusammensetzung des Eutektikums zu erhalten.
Da auch die Viskosität einen großen Einfluss auf die Unterkühlung besitzt, wurde gezielt eine Reihe von hochreinen Lösungsmitteln verschiedener Viskositäten auf ihre Unterkühlungsneigung hin untersucht.
Aufheizexperimente haben im Gegensatz dazu nicht den Nachteil einer Unterkühlung. Carl [6] konnte jedoch mit der von Hammer [7] beschriebenen Apparatur keine auswertbaren Aufheizkurven erzielen. Da dies aber mit der in Abschnitt 2 beschriebenen Apparatur möglich ist, kann ein Vergleich zwischen den Daten, die mittels Abkühlung und mittels Aufheizung erhalten werden, durchgeführt werden.

3.2 Grundlagen der Methode und Theorie

Zur Theorie der Bestimmung von Fest-Flüssig-Übergängen sei hier auf Kapitel 2.2 verwiesen. Daher werden in diesem Kapitel nur Schmelzdiagramme binärer, eutektischer Mischungen beschrieben, die dem einfachsten Fall entsprechen. Ein verallgemeinertes Beispiel ist als Temperatur-Molenbruch-Diagramm in Abbildung 20 dargestellt. Für dieses System (Komponenten A und B) gelten folgende Bedingungen:

[7] Diese Praktika wurden von H. J. Gores und dem Verfasser der Arbeit angeleitet. Den beiden Praktikanten P. Wachter und F. Wudy sei an dieser Stelle für ihr großes Engagement bei der Durchführung und Auswertung der Messungen gedankt.

- A und B sind in flüssiger Phase (=Phase α) in jedem Gewichtsverhältnis vollständig mischbar.
- A und B sind in fester Phase (=Phase β) völlig unmischbar[8].
- Der Druck ist konstant.

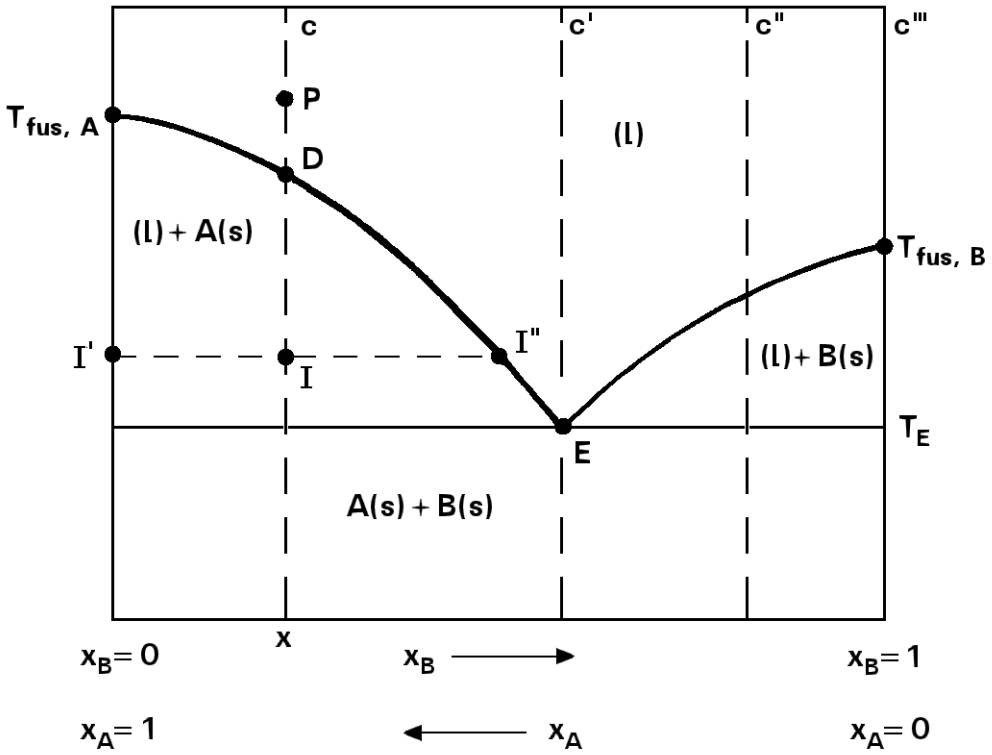


Abbildung 20 Phasendiagramm eines binären Gemisches mit Eutektikum

Das Zustandsdiagramm enthält verschiedene Gebiete mit unterschiedlicher Zusammensetzung:

- (L) : homogene flüssige Mischung der Komponenten A und B;
- (L) + A(s): Koexistenzgebiet von (l) und fester Komponente A;
- (L) + B(s): Koexistenzgebiet von (l) und fester Komponente B;
- A(s) + B(s): Existenzgebiet von A(s) neben B(s).

Die Linien $T_{fus,A}E$ (Erstarrungskurve von A = Schmelzkurve von A = Löslichkeitskurve von A) und $T_{fus,B}E$ (Erstarrungskurve von B = Schmelzkurve von B = Löslichkeitskurve von B) werden aus den Knickpunkten der Abkühl- beziehungsweise Aufheizkurven bestimmt (siehe Kapitel 2.2). Aus den Haltepunkten dieser Messungen werden neben der Isotherme durch den Punkt E (Eutektische Isotherme) auch die Festpunkte der Reinstoffe A und B, $T_{fus,A}$ und $T_{fus,B}$ (bei $x_A = 1$ bzw. $x_B = 1$) ermittelt (siehe Kapitel 2.2 und 3.3). Diese Kurven begrenzen die verschiedenen Phasenbereiche. Als ausgezeichneten Punkt enthält das Diagramm den eutektischen Punkt E. Soll eine Elektrolytmischung in einem elektrochemischen System eingesetzt werden, so ist es zwingend erforderlich, dass der Elektrolyt komplett flüssig bleibt, also im Gebiet (L) liegt. Ein teilweise gefrorener Elektrolyt weist nicht nur eine deutlich geringere Leitfähigkeit auf. Viel schwerer wiegt die Gefahr, dass das Elektrodenmaterial oder der Separator irreparabel durch den ausfallenden Festkörper be-

[8] Es bilden sich keine Mischkristalle, sondern es fällt ein Gemisch aus (beliebig kleinen) Reinkristallen aus.

schädigt werden. Durch die Wahl der Lösungsmittelzusammensetzung und durch den Salzgehalt kann der Flüssigkeitsbereich des Elektrolyten eingestellt werden (siehe Abschnitt 4.2.1.2).

3.3 Methoden zur Auswertung der Messdaten

Wie in Kapitel 2.2 beschrieben, entsprechen die Knick- und Haltepunkte den Schmelz- oder Erstarrungskurven. Da bei den Abkühlungsmessungen meist eine starke Unterkühlung auftritt, ist ein Interpolationsverfahren zur Bestimmung der Knick- und Haltepunkte notwendig. Wie in Abbildung 21 gezeigt[9], wird zur Bestimmung des Haltepunkts sowohl auf den Abkühlungsast (blau) als auch den Ast der Halteline (rot) eine Gerade mittels linearer Regression angepasst. Der Haltepunkt ist dabei durch den Schnittpunkt der beiden Geraden definiert.

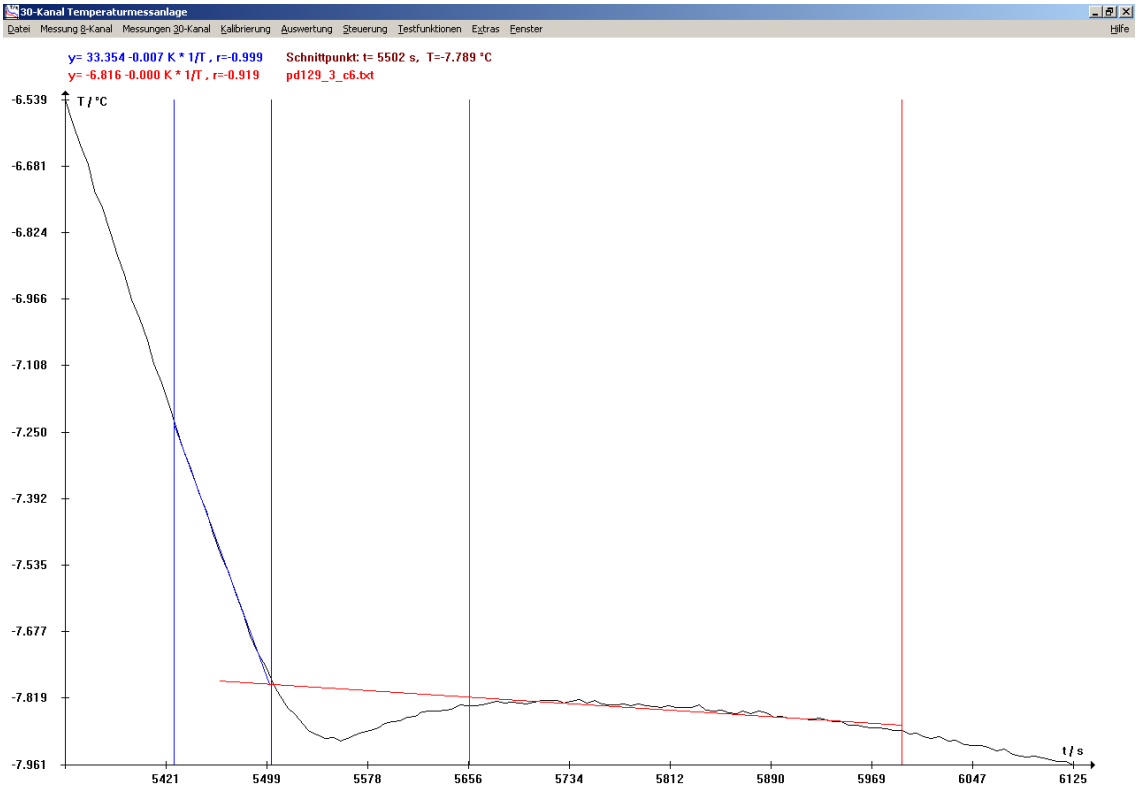


Abbildung 21 Auswertung von Haltepunkten am Beispiel des Eutektikums von EC/DMC, Abkühlexperiment

Die Unterkühlung der Lösung entspricht dabei der Differenz zwischen dem Haltepunkt und dem tiefsten Punkt der Unterkühlung.
Bei der Auswertung der Knickpunkte wird wie in Abbildung 22 gezeigt ein analoges Verfahren angewendet:

[9] Die in diesem Kapitel gezeigten Abbildungen sind Bildschirmkopien der Steuer- und Auswertesoftware, die vom Autor dieser Arbeit entwickelt wurde. Mit dieser Software können die hier beschriebenen Regressionen leicht durchgeführt werden.

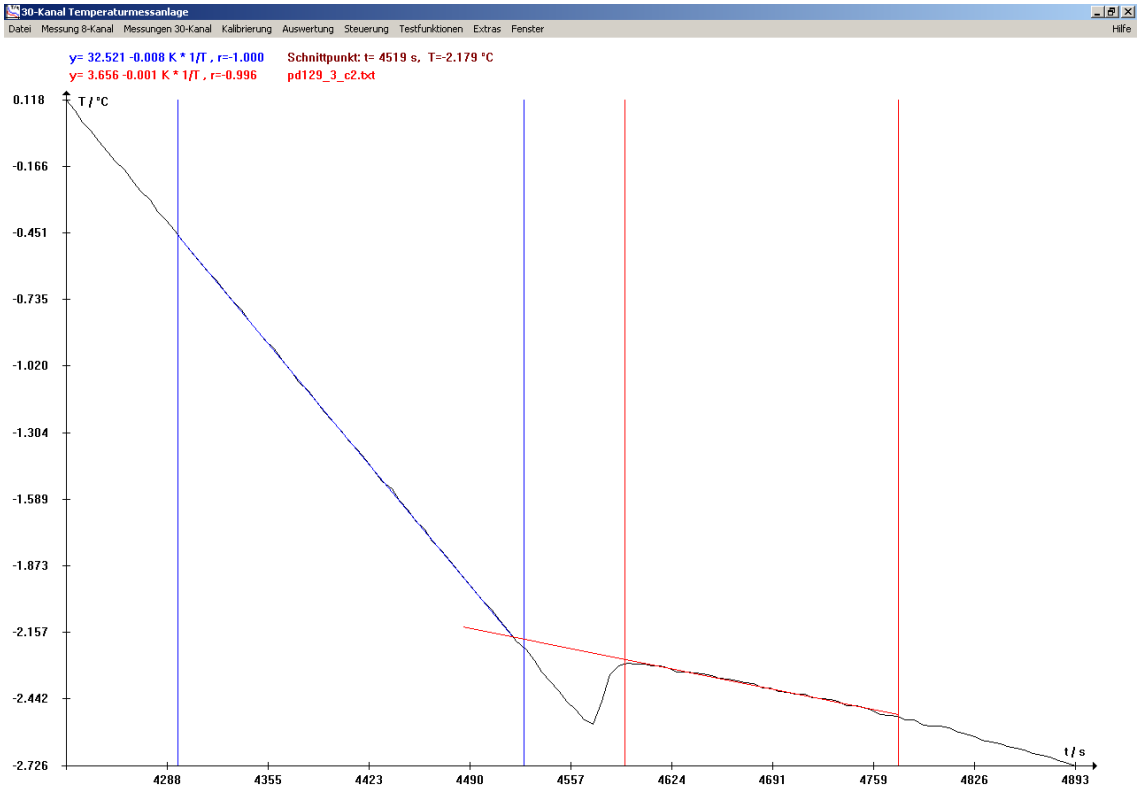


Abbildung 22 Auswertung von Knickpunkten am Beispiel der binären Mischung EC/DMC, Molenbruch x_{DMC}=0,174, Abkühlexperiment

Auch in diesem Beispiel wird der Knickpunkt durch den Schnittpunkt der beiden Geraden bestimmt, die mittels linearer Regression ermittelt wurden. Die Unterkühlung wird analog zu den Haltepunkten bestimmt. Ist die Unterkühlung sehr stark ausgeprägt, so kann dieses Verfahren nicht angewendet werden, da dann die Steigung der in Abbildung 22 rot dargestellten Geraden größer ist als die der anderen und somit ein physikalisch unsinniger Schnittpunkt erhalten würde. In diesem Fall wird als Knickpunkt der höchste Punkt nach der Unterkühlung angenommen.
Bei Aufheizexperimenten wird, wie in Abbildung 23, im Prinzip ein analoges Verfahren angewendet, um den Knickpunkt als Schnittpunkt zweier durch lineare Regression ermittelter Geraden zu bestimmen.

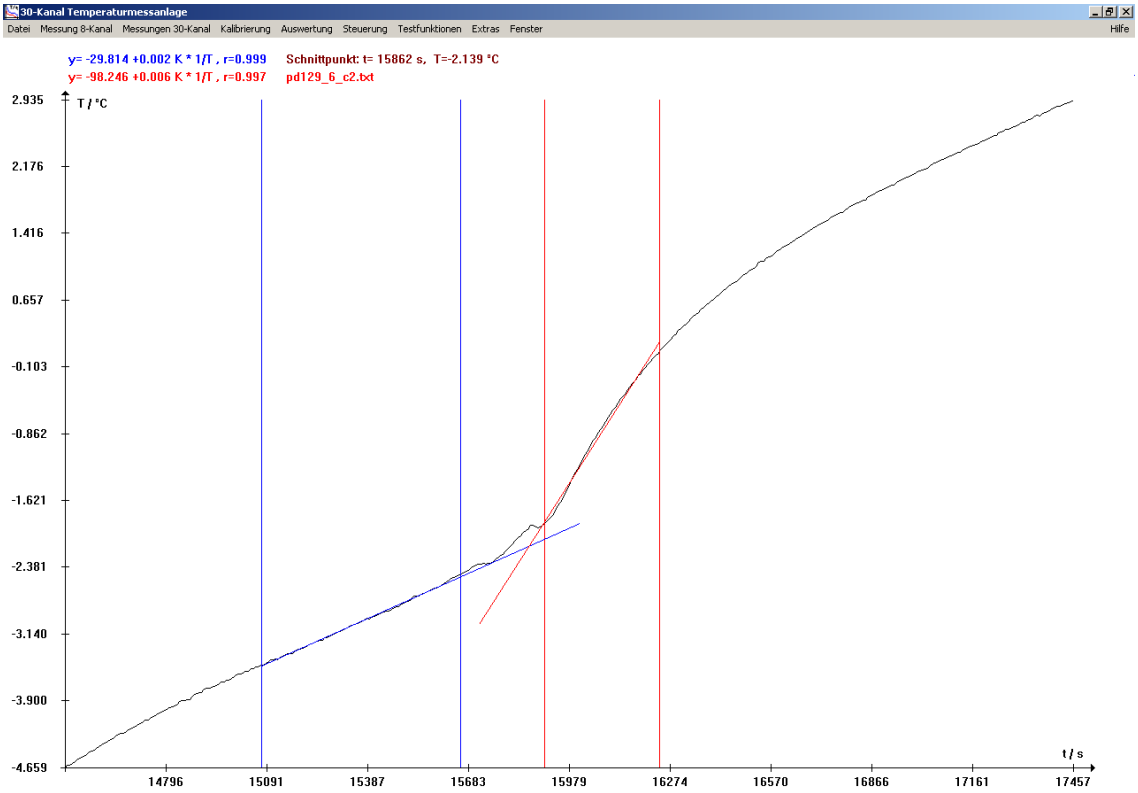


Abbildung 23 Auswertung von Knickpunkten am Beispiel der binären Mischung EC/DMC, Aufheizexperiment

Wie in Kapitel 4.2.1.1 beschrieben, wird der Logarithmus der Leitfähigkeit gegen den Kehrbruch der Temperatur aufgetragen. So wird nach Arrhenius und Eyring 4.2.1.1 ein linearer Zusammenhang gefunden. Die Arrhenius-Eyring-Gleichung ist zwar prinzipiell für Flüssigkeiten ungeeignet, da sie auf einem Transportmodell beruht, welches das Entstehen von Löchern, in die Teilchen hinein wandern, voraussetzt. Eher wären für die Temperaturabhängigkeit der Leitfähigkeit die Vogel-Fulcher-Tamman-Gleichung und deren Varianten geeignet, die u.a. durch kooperative Umordnungen von Domänen in der Flüssigkeit beschrieben werden können. Da aber bei diesen Messungen nur kleine Temperaturbereiche betrachtet werden, reicht der hier benutzte Arrhenius-Eyring-Ansatz völlig aus.
Wird die Löslichkeitstemperatur einer Komponente der Mischung unterschritten und fällt diese als Feststoff aus, so wird ein starker Abfall der Leitfähigkeit beobachtet. Die Temperatur, bei der dieser Knickpunkt auftritt, wie in Abbildung 24 gezeigt, kann entweder direkt abgelesen oder mittels des oben beschriebenen Interpolationsverfahrens bestimmt werden.

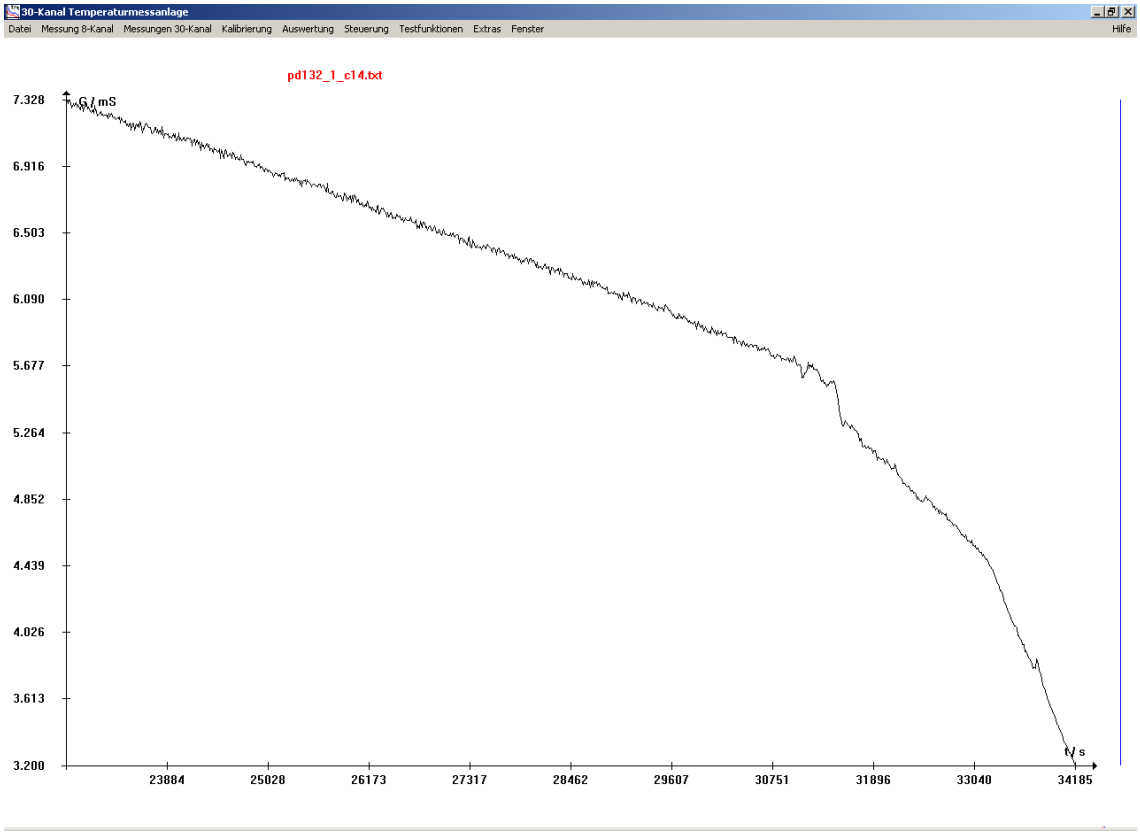


Abbildung 24 Auswertung von Knickpunkten bei leitfähigen Elektrolyten: Bei dem hier vermessenen System handelt es sich um ein Beispiel der Elektrolytlösungen aus Kapitel 4.2.3, mit einer Zusammensetzung von 0,7 mol/kg LiBOB in einer Mischung aus 66,7% EC, 23,3 % PC und 10% DMC

3.4 Beispielhafte Messungen

In diesem Kapitel werden aufgrund der Vielzahl der erhaltenen Messergebnisse nur exemplarische Ergebnisse gezeigt. Da die Messungen an leitfähigen Systemen in Kapitel 4.3 und Kapitel 6.5 beschrieben werden, werden hier nur Messungen an Einzelsubstanzen und Lösungsmittelmischungen aufgeführt.

In Tabelle 2 sind die von Wachter [21] bestimmten Gefrierpunkte θ_{fus}, Unterkühlungen ΔT_{UK} und Schmelzpunkte θmelt von Reinstoffen und ihre Standardabweichungen $\sigma\theta_{fus}$ und $\sigma\theta_{melt}$ aufgeführt. Zusätzlich dazu sind noch die Literaturwerte $\theta_{fus,lit}$ aufgeführt.

	θ_{fus} / °C	$\sigma\theta_{fus}$ / °C	ΔT_{UK} / K	θ_{melt} / °C	$\sigma\theta_{melt}$ / °C	$\theta_{fus,lit}$ / °C
Dimethylcarbonat	4,39	0,06	0,44	4,06	0,31	2,85[8]; 4,6[9] 4,61[6] 4,9[3],[4]
1-Octanol	-16,22	0,21	1,64	-18,86	0,80	-15[10]; -18[11]; -14,83[12]; -15,05[13]
1-Dekanol	5,91	0,13	1,1	5,47	0,14	6,54[12]; 6,88[14]
Ethylencarbonat	35,77	0,15	4,1	34,49	1,12	36,4[15]
Ethylmethylimidazolium-Trifluoracetat-	-12,99	-1,2	4,1	-10,66	0,33	
Trioctylamonium trifluoracetat	--	--		12,52	0,31	
Dimethylsulfoxid	18,2	18,12		17,17	0,77	18,53[16]; 18,54[17]; 18,55[18]; 18,52[19]
Dioxan						11,79[20]

Tabelle 2 Gefrierpunkte θ_{fus}, Unterkühlungen ΔTUK und Schmelzpunkte θmelt von Reinstoffen und ihre Standardabweichungen $\sigma\theta$fus und $\sigma\theta$melt [21]

Die mit dieser Anlage gefundenen Werte für die Schmelz- und Gefriertemperaturen liegen innerhalb der Streuung der Literaturwerte. Die Unterkühlungen liegen im Bereich von 0,3 bis 4,1 K. Besonders am Beispiel des Ethylmethylimidazoliumtrifluoracetats wird die Überlegenheit der hier verwendeten Anlage gegenüber den DTA Messungen bei Ionischen Fluiden deutlich, da die Unterkühlung hier nur 4 K beträgt. Die Abweichungen, die zwischen den Umwandlungstemperaturen bestehen, die mit Abkühlungskurven oder Aufheizkurven bestimmt wurden, betragen sogar nur 2,3 K. Werden diese Temperaturen mittels DTA oder DSC bestimmt, so liegen die Unterkühlungen im Bereich von 10 bis 200 K mit einer mittleren Unterkühlung von 30 K [22]. Umwandlungstemperaturen, welche aus Abkühlkurven erhalten wurden, liegen tendenziell über den Temperaturen welche durch Aufheizexperimente bestimmt wurden [21].

Da Wachter an allen Systemen Messungen sowohl mit Zusatz von Kohlefasern, wie von Jow [2] beschrieben, wie auch ohne diesen Zusatz durchgeführt hat, konnte dann die Auswirkung dieses Zusatzes auf die Unterkühlung bestimmt werden. Es zeigt sich dabei, dass durch den Zusatz von Kohlefasern eine deutliche Reduzierung der Unterkühlung erreicht wird, wie das Beispiel von Dimethylcarbonat und Ethylmethylimidazoliumtrifluoracetat (EMITFAC) zeigt, das in Abbildung 25 dargestellt ist

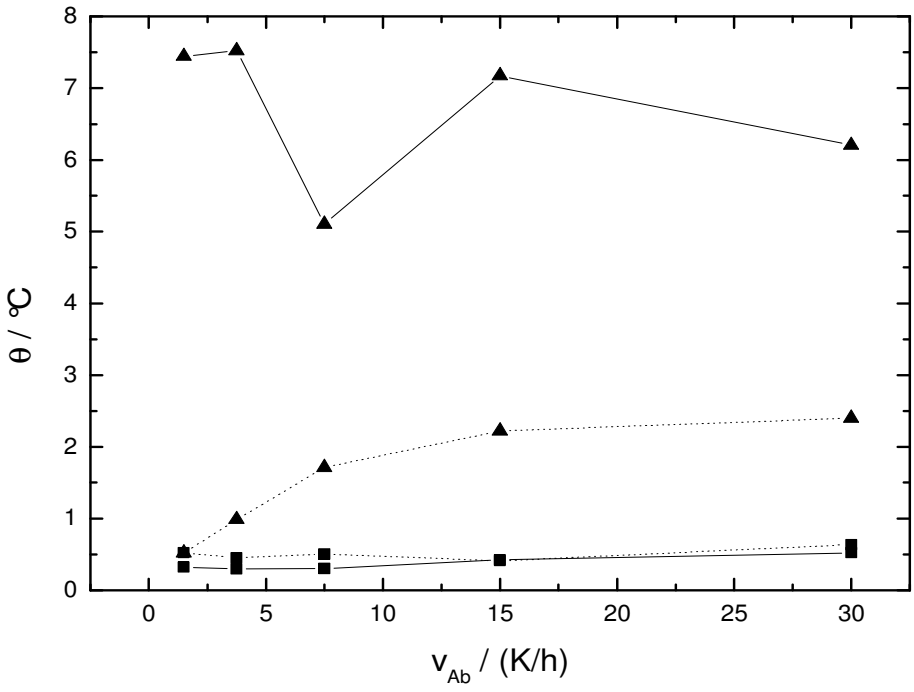


Abbildung 25 : Auswirkung der Zugabe von Kohlefasern auf die Unterkühlung bei DMC und EMI; —■— DMC, ···■·· DMC mit Kohlefasern, —▲— EMITFAC, ··▲·· EMITFAC mit Kohlefasern [21]

Weitere Experimente zur Klärung der Frage, inwieweit die Unterkühlung von der Abkühlungrate und von der Viskosität abhängt, wurden von Wudy [23] und Wachter [21] durchgeführt. Da aber bis jetzt noch nicht alle Daten ausgewertet worden sind kann darüber noch keine klare Aussage getroffen werden.

Das Schmelzdiagramm der binären Mischung aus Ethylencarbonat (EC) und Dimethylcarbonat (DMC) wurde bereits von Ding [3] und Carl [6] beschrieben. Dieses System wurde mit Abkühlbeziehungsweise Aufheizraten im Bereich von 36 K/h bis 3 K/h in Schritten von 3 K/h vermessen. Da bei diesen Messungen keine Abhängigkeit der Messergebnisse von der Temperaturänderungsrate erkennbar war, wurden für die Konstruktion des Phasendiagramms, das in Abbildung 26 gezeigt ist, die erhaltenen Knick- und Haltepunkte gemittelt.

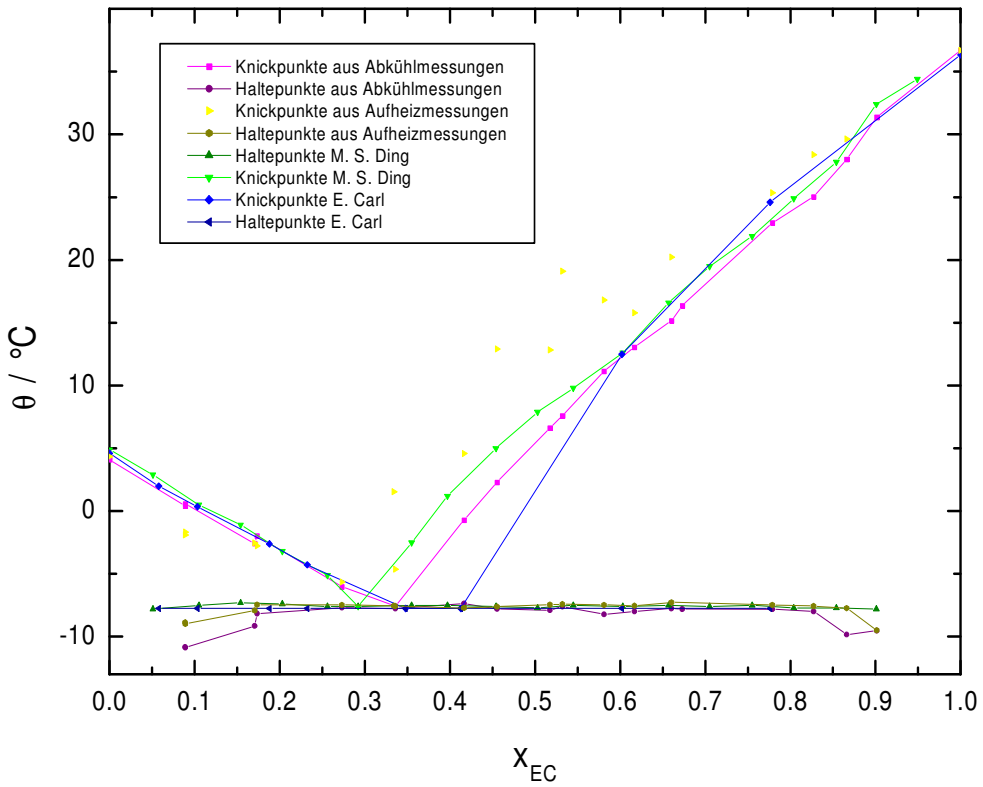


Abbildung 26 Binäres Schmelzdiagramm der Mischung Dimethylcarbonat /Ethylencarbonat

Im dimethylcarbonatreichen Ast des Phasendiagramms weichen die gefundenen Werte nur geringfügig von den Literaturwerten ab. Diese Abweichungen sind im ethylencarbonatreichen Ast, wohl aufgrund der höheren Viskosität, deutlicher ausgeprägt. In diesem Bereich werden auch starke Abweichungen der Literaturwerte untereinander beobachtet. Die größten Abweichungen der Literaturwerte voneinander treten aber im Bereich des Eutektikums auf. Ding gibt für die eutektische Zusammensetzung einen Molenbruch von x_{EC}=0,292 [3] an, bei einer Eutektikumstemperatur von -7,6 °C. Für diese Gleichgewichtstemperatur wird von Carl ein Wert von –7,76 °C berichtet, der Molenbruch für die eutektische Mischung liegt bei x_{EC}=0,348.

Die Daten, die von Carl bestimmt wurden, weisen vom Eutektikum aus in Richtung des ethylencarbonatreichen Astes, im Vergleich zu den anderen Daten, eine sehr große Abweichung von bis zu 15 °C auf. Ob diese Abweichungen durch Unterkühlungseffekte oder andere Fehler verursacht wurden, ist nicht bekannt. Die mit der in Kapitel 2 beschriebenen Anlage bestimmten Werte zeigen bei den Abkühlungsmessungen untereinander etwa die gleiche Streuung wie im Vergleich zu den anderen Messwerten. Für die eutektische Zusammensetzung wird ein Molenbruch von x_{EC}=0,34 erhalten, der im Bereich des von Carl (0,348) angegebenen Wertes liegt.

Die Temperatur des Eutektikums beträgt –7,53 °C. Im Gegensatz zu den Messungen von Jow und Ding wurden bei der Konstruktion des Phasediagramms Mittelwerte aus einer großen Anzahl von Messpunkten verwendet. Die Streuung der einzelnen Messungen untereinander ist dabei deutlich kleiner als die Abweichungen, die zwischen den Ergebnissen von Ding von Carl bestehen.

Im Gegensatz zu den Messungen von Ding[10], bei denen die eutektische Isotherme nahezu waagerecht verläuft, wird hier eine deutliche Abweichung zu tiefen Temperaturen hin an den Rändern des Phasendiagramms beobachtet. Diese Absenkung ist ein Indiz dafür, dass die beiden Substanzen Mischkristalle bilden können, wenn eine Komponente nur in einer sehr kleinen Menge vorhanden ist. Ob eine solche Randlöslichkeit wirklich auftritt, oder ob diese Absenkung ein Effekt ist, der durch starke Unterkühlungen hervorgerufen wird, ließe sich durch Vermessen mehrerer Proben mit einem Molenbruch an EC im Bereich von 0 bis 0,1 und von 0,9 bis 1 klären.
Zusätzlich zu den Abkühlkurven wurde noch die eutektische Zusammensetzung mittels NMR-Spektroskopie bestimmt. Für dieses Experiment wurde eine Mischung aus EC und DMC auf eine Temperatur von –7,50 °C abgekühlt und um das Gleichgewicht einzustellen bei dieser Temperatur für 2 h gerührt. Die überstehende Lösung, deren Zusammensetzung der eutektischen Mischung entspricht, wurde abdekantiert. Die Zusammensetzung dieser Mischung wurde in Analogie zur Wasserbestimmung, siehe Kapitel 7.3, mittels ^{1}H-NMR Spektroskopie untersucht. Bei diesem Experiment wurde eine Zusammensetzung von x_{EC}=0,35 für das Eutektikum gefunden. Da dieser Wert sehr gut dem Wert, der mit den Abkühlkurven bestimmt wird (x_{EC}=0,35), und darüber hinaus auch sehr gut mit dem Wert von Carl (x_{EC}=0,348) übereinstimmt, liegt es sehr nahe, dass es aufgrund von Unterkühlungseffekten mittels DTA/DSC nur mit großen Fehlern möglich ist, binäre Schmelzdiagramme von diesen Lösungsmitteln zu bestimmen, da mit diesen Methoden eine Zusammensetzung von x_{EC}=0,292 [3] erhalten wird.
Im Gegensatz zu den Haltepunkten, die bei den Abkühl- und Aufheizexperimenten eine gute Übereinstimmung zeigen, weisen die Knickpunkte der Aufheizkurven eine deutliche Abweichung von den Literaturwerten auf. Ebenso ist die Streuung dieser Messwerte ungleich höher. Die Messung einer Aufheizkurve erfolgte stets im Anschluss an eine Abkühlung. Da immer unter die eutektische Temperatur hinweg abgekühlt wurde, muss diese Umwandlungstemperatur auch beim Aufheizen überwunden werden. Aufgrund der großen Zeitspanne, die benötigt wird bis die Lösung komplett geschmolzen ist, besteht daher immer die Gefahr, dass der Schmelzvorgang des eutektischen Gemisches den Schmelzvorgang der anderen Phasen überlagert. Um dieses Problem zu umgehen, wäre es nötig nur bis knapp unter die eutektische Temperatur abzukühlen und dann wieder aufzuheizen. Dies ist aber mit der vorhandenen Thermostatenanlage noch nicht durchführbar.
Das in Abbildung 27 gezeigte Phasendiagramm des Systems Dioxan/Dimethylcarbonat wurde von Wudy [23] unter Anleitung des Autors dieser Arbeit vermessen. Die Abkühl- bzw. Aufheizraten dieser Messungen lagen dabei im Bereich von 30 K/h bis 1,5 K/h. Bei diesem System wird wie beim System EC/DMC auch eine Zunahme der Messfehler bei höheren Gehalten an EC beobachtet. Es liegt nahe, dass diese Fehler durch eine zunehmende Unterkühlung, die durch die größer werdende Viskosität begünstigt wird, verursacht werden. Das Eutektikum dieses Systems liegt bei dem Molenbruch x_{DIOXAN} = 0,42. Die eutektische Temperatur liegt bei –21,7 °C. Interessanterweise zeigen hier die Messwerte, die aus den Aufheizexperimenten erhalten wurden, eine deutlich geringere Streuung.

[10] Carl gibt nur eine Temperatur für das Eutektikum an. Diese Temperatur wurde in Abbildung 26 bei allen Zusammensetzungen als Temperatur des Haltepunktes eingetragen. Daher kann aus diesen Messungen keine Aussage über Randlöslichkeiten getroffen werden.

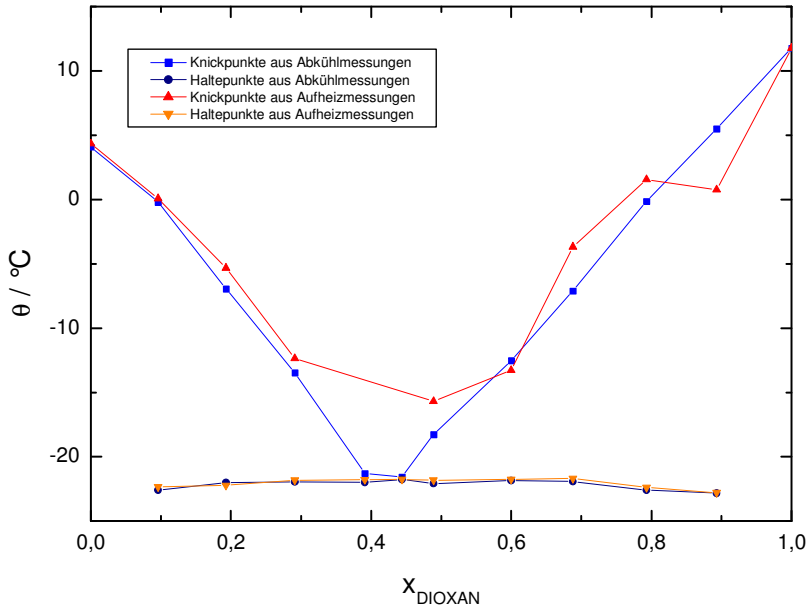


Abbildung 27 Binäres Schmelzdiagramm der Mischung Dioxan/Dimethylcarbonat

Im Schmelzdiagramm des Systems Dimethylsulfoxid/Ethylencarbonat werden die Probleme, die bei dieser Messmethode auftreten, besonders deutlich. Dieses Diagramm, das in Abbildung 28 dargestellt ist, wurde von Wachter [21] unter Anleitung des Autors bestimmt.

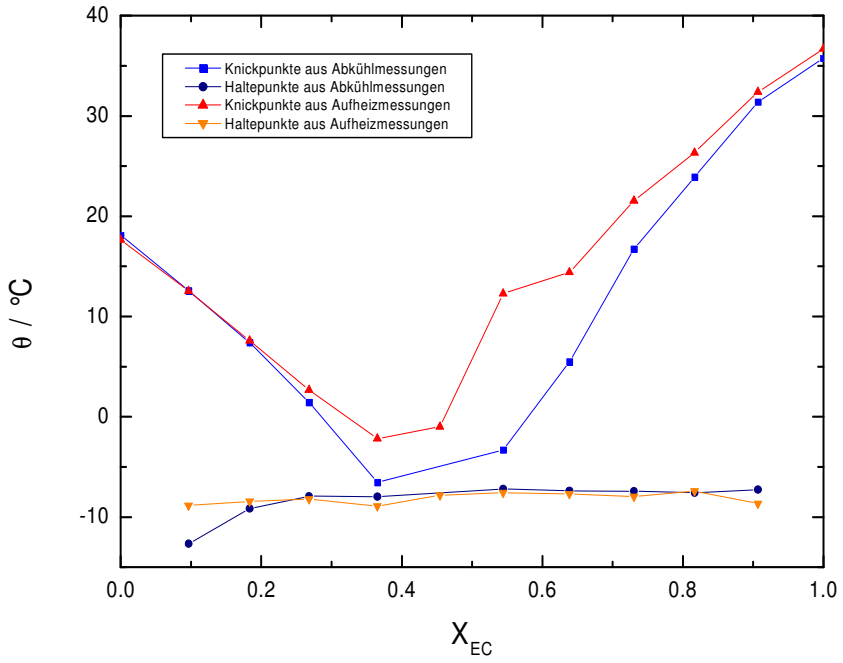


Abbildung 28 Binäres Schmelzdiagramm der Mischung Dimethylsulfoxid/Ethylencarbonat

Auch bei diesem Phasendiagramm wird, insbesondere im DMSO-reichen Ast, eine bessere Übereinstimmung der Daten aus den Aufheizversuchen mit den Daten aus den Abkühlexperimenten erreicht. Die Abweichung der Messwerte wird umso größer, je näher die Zusammensetzung der Mischung der eutektischen Mischung kommt. Es liegt auch hier nahe, dass diese Abweichungen durch eine Überlagerung des Schmelzens des eutektischen Gemisches mit dem totalen Schmelzen der Lösung verursacht wird. Diese Überlagerung ließe sich, wie oben beschrieben, vermeiden, wenn bei den Abkühlexperimenten nur bis knapp über das Eutektikum abgekühlt würde. Stünde ein vom Rechner ansteuerbarer Kyrostat zur Verfügung, wären auch solche Versuche leicht durchführbar, wodurch sich eine deutliche Verbesserung der Messgenauigkeit im Bereich des Eutektikums erreichen ließe.
Durch eine extrem ausgeprägte Unterkühlung, wie es bei dem System Dioxan/Ethylencarbonat, das sich aus zwei hochviskosen Komponenten zusammensetzt, der Fall ist, kann es unmöglich werden, das Phasendiagramm zu konstruieren, wie Abbildung 29 zeigt.

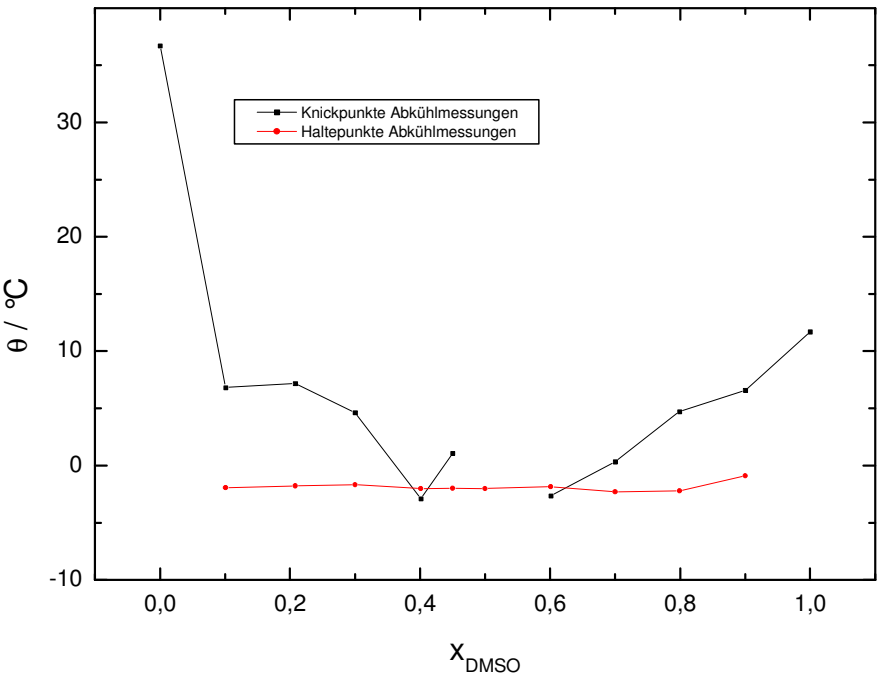


Abbildung 29 Binäres Schmelzdiagramm der Mischung Dioxan/Ethylencarbonat, Versagen der Methode aufgrund sehr starker Unterkühlungen

Aus diesen Messungen lässt sich die eutektische Temperatur bestimmen, der dioxanreiche Ast des Diagramms kann auch noch mit einiger Sicherheit ausgewertet werden, jedoch beim ethylencarbonatreichen Ast versagt die Methode vollständig.

3.5 Probleme und ihre Lösungen

Das Hauptproblem bei der Untersuchung der binären Phasendiagramme ist in der Unterkühlung der Lösungen zu sehen. Diese Unterkühlung tritt bei der Untersuchung hochviskoser Lösungsmittel verstärkt auf. Durch Zugabe von Kristallisationskeimen, wie zum Beispiel Kohlefasern, lässt sich dieser Effekt reduzieren [2]. Auch durch Verwendung eines sogenannten Rückwerks, bei dem die Lösung nicht kontinuierlich, sondern schockartig gerührt wird, lässt sich die Unterkühlung verringern. Ein Beispiel für ein solches Rückwerk ist in der Diplomarbeit des Autors [24] beschrieben, wurde aber aufgrund der geringen Standzeit der Glasrührer nicht mehr eingesetzt. Durch Austausch dieser Rührstäbe durch Kunststoffstäbe ließe sich dieses Problem vermeiden.
Für die Entwicklung von Batterieelektrolyten werden auch Informationen über den Flüssigkeitsbereich von Elektrolytmischungen benötigt, siehe Kapitel 4, die Lösungsmittelkomponenten enthalten, welche einen niedrigen Schmelzpunkt kleiner als –40 °C, aufweisen. Soll das Phasendiagramm solcher Mischungen bestimmt werden, ist ein anderer Kryostat nötig, mit dem lineare Abkühlungen unter –35°C hinaus möglich sind. Soll auch das Eutektikum dieser Mischungen bestimmt werden, so sollte der Temperaturbereich dieses Thermostaten bis –90 °C reichen. Da diese Lösungen eine deutlich geringere Viskosität aufweisen als die hier vermessenen Mischungen, dürfte bei diesen technisch hoch interessanten Systemen die Unterkühlung eine deutlich geringere Rolle spielt als bei den in diesem Kapitel gezeigten Phasendiagrammen, für die Systeme ausgewählt worden sind, die aufgrund ihres Temperaturbereichs komplett mit dem in Kapitel 2.5.3 vorgestellten Kryostaten untersucht werden konnten.
Wird bei den Abkühlungsversuchen unter die Temperatur des Eutektikums abgekühlt, so wird bei den darauffolgenden Aufheizexperimenten die Bestimmung der Knickpunkte deutlich erschwert, da das Schmelzen des eutektischen Gemisches viel Zeit in Anspruch nimmt. Eine Möglichkeit, dieses Problem zu lösen ist, zuerst die Temperatur des Eutektikums zu bestimmen, anschließend bei den Abkühlungen nur bis knapp unter die Temperatur des Eutektikums abzukühlen und daraufhin wieder aufzuheizen. Eine Alternative zu dieser Vorgehensweise wäre, beim Aufheizen knapp oberhalb der Eutektikumstemperatur den Aufheizvorgang für einige Zeit zu unterbrechen und ihn erst dann fortzusetzen, wenn das eutektische Gemisch vollständig geschmolzen ist. Für diese Methode ist es aber nötig, dass die Temperatur des Kryostaten von der Temperatur des Ölbades des Thermostaten aus gesteuert wird. Hierfür bietet es sich an, die Temperatur des Ölbades mit einem weiteren Temperaturfühler zu messen, der an den Kryostaten anschlossen wird. Diese Temperaturinformation wird dann zur Regelung der Temperaturrampen eingesetzt. Wird der Kryostat über eine geeignete Schnittstelle vom Steuerrechner angesteuert, so kann die Aufheizung zum passenden Zeitpunkt angehalten werden.
Ein Ziel der Entwicklung dieser Messanlage war es, schnell und mit hoher Auflösung der Temperatur und der Zusammensetzung Schmelzdiagramme zu bestimmen. Aufgrund der großen Anzahl von Messkanälen und der vollautomatischen Messdatenerfassung, die es ermöglicht, rund um die Uhr zu messen, fallen große Datenmengen an. Zur Auswertung dieser Datenmengen wäre eine Anbindung der Phasendiagrammmessanlage an ein Datenbanksystem wünschenswert, das die Auswertung, Verwaltung und Archivierung der Messdaten vereinfacht.

3.6 Zusammenfassung und Ausblick

Wie die hier aufgeführten Beispiele zeigen, ermöglicht die vorgestellte Methode, Schmelzdiagramme binärer Mischungen zu bestimmen. Die Resultate sind von höherer Genauigkeit als die in der Literatur gezeigten DTA/DSC-Messungen, da bei diesen Methoden eine deutlich größere Verfälschung der Messergebnisse durch Unterkühlung auftritt. Diese Abweichung wird besonders bei der Bestimmung des Eutektikums deutlich. Wird die Zusammensetzung des Eutektikums mittels NMR-Spektroskopie unabhängig von den Abkühlkurven bestimmt, wird eine sehr gute Übereinstimmung mit dem Ergebnis der Abkühlkurven erhalten. Das Ergebnis aus den DTA/DSC Messungen weicht davon deutlich ab.

Durch die große Anzahl an Messkanälen und die automatische Durchführung der Messungen ist es mit dieser Anlage möglich, die Phasendiagramme in sehr kurzer Zeit zu ermitteln. Beispielsweise wurden die Phasendiagramme Dimethylsulfoxid/Ethylencarbonat und Dioxan/Dimethylcarbonat jeweils innerhalb einer Woche vermessen, wobei bei jeder Zusammensetzung fünf Abkühl- und Aufheizversuche durchgeführt wurden. Durch den hohen Probendurchsatz dieser Anlage kann auch erstmals eine statistische Fehlerauswertung der Ergebnisse erfolgen. Diese Auswertung wäre bei den anderen Methoden aufgrund der geringen Anzahl an in einer vergleichbaren Messzeit durchführbaren Experimenten nur sehr schwer möglich gewesen. Wäre zum Beispiel das System Dioxan/Dimethylcarbonat mit der Methode von Carl vermessen worden, wäre bei der gleichen Probenanzahl und der gleichen Anzahl an unterschiedlichen Heiz- und Abkühlraten mindestens ein halbes Jahr Arbeitszeit benötigt worden. Dank des hohen Probendurchsatzes können mit dieser Anlage auch Phasendiagramme ternärer oder gar quartärer Mischungen in endlicher Zeit komplett untersucht werden. Sobald ein Kryostat mit größerer Kälteleistung zur Verfügung steht, mit dem auch Temperaturen bis –90 °C erreicht werden können, ist es auch möglich, eine Vielzahl von Phasendiagrammen von Lösungsmittelmischungen zu untersuchen, die für den Einsatz als Batterieelektrolyte von großer Bedeutung sind.

Eine interessante Erweiterungsmöglichkeit wäre eine Messmethode, die der Differenzthermoanalyse sehr ähnlich ist. So wäre es zum Beispiel möglich, die Differenz der Temperatur innerhalb der Messzellen gegen die Badtemperatur oder gegen die Temperatur einer Messzelle, die mit einer inerten Substanz gefüllt ist, aufzutragen. Mit dieser Methode könnte es möglich sein, die Empfindlichkeit der Anlage nochmals zu steigern. Neben der Optimierung der Rührwerke, bietet auch die Isolation der Messzelle einen Ansatzpunkt für weitere Verbesserungen der Messmethode. Mit der in Abschnitt 3.5 beschriebenen Methode zur Vermeidung der Verfälschung der Aufheizkurven dürfte es möglich sein, die Genauigkeit und Zuverlässigkeit der Aufheizkurven deutlich zu steigern.

Neben der Bestimmung von Phasendiagrammen eignet sich die Anlage auch zur Untersuchung folgender Fragestellungen, welche sich aufgrund des hohen Probendurchsatzes leicht bearbeiten lassen:

- Wie hängt die Unterkühlung von der Abkühlgeschwindigkeit ab, und wie lässt sie sich vermeiden?
- Wie groß ist der Einfluss der Viskosität auf die Unterkühlung?
- Wie unterscheiden sich Haltepunkte, die durch Abkühlung bestimmt werden, von denen, die durch Aufheizung gewonnen werden?
- Wie unterscheiden sich Phasendiagramme, die mittels DTA bestimmt worden sind, von denen aus Abkühl- bzw. Aufheizkurven?
- Wie unterscheiden sich Phasenübergänge, die durch Temperaturänderung detektiert wurden, von denen aus Leitfähigkeitsmessungen?

Erste Arbeiten auf diesem Gebiet wurden bereits von Wachter [21] und Wudy [23] unter Anleitung des Autors begonnen. Weitere werden nach dem Eintreffen des bereits bestellten steuerbaren Kryostaten, der alle hier genannten Anforderungen erfüllt, durchgeführt werden. Es ist zu erwarten, dass mit diesen Ergebnissen wesentliche Fortschritte bei der thermischen und konduktometrischen Bestimmung von Phasendiagrammen erzielt werden können.

3.7 Literaturverzeichnis

[1] P. Wasserscheid, *Ionic Liquids in Synthesis*, Wiley New York (2003)
[2] M. S. Ding, K. Xu, S. S. Zhang, T. R. Jow, K. Amine and G. L. Henriksen, *J. Electrochem. Soc.*, **146**, 3974 (1999)
[3] M. S. Ding, K. Xu and T. R. Jow, *Journal of Thermal Analysis and Calorimetry*, **62**, 177 (2000)
[4] M. S. Ding, K. Xu and T. R. Jow, *J. Electrochem. Soc.*, **147**, 1688 (2000)
[5] M. S. Ding, K. Xu, S. Zhang and T. R. Jow, *J. Electrochem. Soc.*,, **148**, A299 (2001)
[6] E. S. Carl, *Neue Elektrolyte in organischen Carbonatlösungen zur Anwendung in sekundären Lithium-Ionen Batterien*, *Dissertation*, Regensburg (1998)
[7] H. Hammer, *Eichung eines Platinwiderstandsthermometers zur Präzisionsmessung im Bereich von -60 bis 50°C nach der Internationalen Praktischen Temperaturskala 1968*, *Diplomarbeit*, Saarbrücken (1970)
[8] C. Mialkowski, A.Chagnes, B.Carré, D. Lemordant and P. Willmann, *J. Chem. Thermodynamics*, **34**, 1848 (2002)
[9] S. Hossian, D. Linden*, Handbook of Batteries* 2nd ed., New York (1995)
[10] G. L. Dorough, H. B. Glass, T. L. Gresham, G. B. Malone and E. E. Reid, *J. Am. Chem. Soc.*, **63**, 3102 (1941)
[11] M. v. Ardenne, K. Steinfelder und R. Tümmler, *Z. Phys. Chem.*, **220**, 106 (1962)
[12] G. H. Findenegg, *Faraday Transactions*, 1800 (1972)
[13] Z. Plesnar, P. Gierycz and A. Bylicki, *J. Chem. Thermodynamics*, **22**, 394 (1990)
[14] C. W. Hoerr, H. J. Harwood and A.W. Ralston, *J. of organic Chem.*, **9**, 268 (1944)
[15] D. R. Lide, *Handbook of Organic Solvents*, Boca Raton (1995)
[16] H. Schott, *J. Pharm. Sciences*, **58**, 946 (1969)
[17] R. Garnsey and J. E. Prue, *Trans. Faraday Soc.*, **64**, 1210 (1968)
[18] H. L. Schläfer und W. Schaffernicht, *Angew. Chem.*, **72**, 619 (1960)
[19] H. L. Clever and E. F. Westrum, *J. Phys. Chem.*, **74**, 1309 (1970)
[20] G. Jancso, H. Illy und D. Staschewski, *Journal of the Chemical Society, Faraday Transactions 1: Physical Chemistry in Condensed Phases*, **72**, 2203 (1976)
[21] P. Wachter, *Schwerpunktspraktikum*, Regensburg (2004)
[22] H. L. Ngo, K. LeCompte, L. Hargens and A. B. McEwen, *Thermochimica Acta*, **357-358**, 97 (2000)
[23] F. Wudy, *Schwerpunktspraktikum*, Regensburg (2004)
[24] H.-G. Schweiger, *Entwicklung einer Präzisionstemperaturmessanlage zur schnellen Messung von Phasendiagrammen und chemische und elektrochemische Charakterisierung von Lithium-bis[oxalato(2-)]borat(1-)*, *Diplomarbeit*, Regensburg (2002)

4 Optimierung neuer, für niedrige Temperaturen geeigneter Elektrolytlösungen

4.1 Zielsetzung

Der größte Nachteil von Lithium-Ionen-Batterien gegenüber Batterien mit wässrigen Elektrolyten ist ihre geringere Hochstromfähigkeit [1]. Diese wird maßgeblich durch den Innenwiderstand der Batterie beeinflusst, der wiederum direkt von der Leitfähigkeit des eingesetzten Elektrolyten abhängt. Ein hoher Innenwiderstand reduziert darüber hinaus die maximale Leistungsabgabe einer Batterie [2]. Um diese Nachteile der Lithium-Ionen-Batterien zu reduzieren, ist es daher ein Ziel, die spezifische Leitfähigkeit des Elektrolyten zu maximieren.

Um auch den Einsatz des Elektrolyten für tiefe Temperaturen sicherzustellen, ist es sinnvoll, die spezifische Leitfähigkeit des Elektrolyten bei diesen Temperaturen zu optimieren, da sich bei einer hohen spezifischen Leitfähigkeit bei tiefen Temperaturen auch eine hohe spezifische Leitfähigkeit bei hohen Temperaturen ergibt. Daher diente als Zielgröße für die Optimierung die spezifische Leitfähigkeit bei –25,00 °C. Die Kenntnis der Abhängigkeit der spezifischen Leitfähigkeit von der Temperatur ist eine in diesem Zusammenhang gewinnbare wichtige Kenngröße für den jeweiligen Elektrolyten.

Da sich nach Vorgaben der Gaia-Akkumulatorenwerke der Einsatzbereich des Elektrolyten bis auf –25 °C erstrecken soll, eine typische Temperaturuntergrenze für den Anwendungsbereich im Automobilsektor, muss der Flüssigkeitsbereich des Elektrolyten bis zu dieser Temperatur sichergestellt werden. Weitere wichtige Kriterien die an einen Elektrolyten gestellt werden, sind noch ein geringer Dampfdruck, um den Hochtemperatureinsatzbereich nicht zu stark zu limitieren. Für den kommerziellen Einsatz sind auch noch der Preis des Elektrolyten und seine Toxizität von entscheidender Bedeutung.

Da für die Leitfähigkeitsoptimierungen neue Lösungsmittelmischungen eingesetzt werden, deren Komponenten zum Teil noch nie in Lithium-Ionen-Batterien verwendet wurden, ist es nötig die elektrochemische Stabilität dieser neuartigen Elektrolyte sowohl an Modellsystemen wie auch an kompletten Batterien zu untersuchen.

Bei der Leitfähigkeitsoptimierung handelt es sich um ein Optimierungsproblem mit einer großen Anzahl an Einflussgrößen, so dass sich eine rasterartige Untersuchung dieses Problems aufgrund der großen Probenzahl in der Regel verbietet. Für die schnelle und wirtschaftliche Lösung dieses Problems müssen daher Verfahren der Versuchsplanung angewendet werden, mit denen die Anzahl der nötigen Versuche reduziert wird, und Optima schnell gefunden werden können.

4.2 Optimierungen von Leitfähigkeiten mithilfe der Simplex-Methode

4.2.1 Grundlagen der Methode und Theorie

Die Leitfähigkeit einer Elektrolytlösung hängt von folgenden Parametern ab [4]:

- der dynamischen Viskosität
- den Radien der Ionen des Salzes oder den Radien der solvatisierten Ionen
- der Assoszionskonstante K_A des Salzes
- der selektiven Solvatation in Mischlösungsmitteln und der
- Konkurrenz von Solvatation und Assoziation.

Da sich die Parameter gegenseitig beeinflussen und daher meist nicht unabhängig von einander variieren lassen, ist die Leitfähigkeitsoptimierung einer Elektrolytlösung meist ein nicht leicht lösbares Problem, da bis jetzt eine Theorie fehlt, die diese Zusammenhänge umfassend beschreibt und mit der es möglich ist, schon im Voraus Aussagen zu treffen, welches Leitsalz in welchem Lösungsmittel oder welcher Lösungsmittelmischung eine optimale Leitfähigkeit aufweist.
Wird beispielsweise die Viskosität einer binären Lösungsmittelmischung durch Änderung der Zusammensetzung variiert, so wird dadurch auch die Dielektrizitätszahl und damit auch die Ion-Ion Wechselwirkung beeinflusst. Weist darüber hinaus eines der Lösungsmittel noch eine stärkere Neigung zur Ionensolvatation auf, so werden auch noch die Stokes-Radien verändert.
Zur Optimierung der Leitfähigkeiten von Elektrolytlösungen lassen sich drei verschiedene Strategien anwenden [4]:

- Optimierung des Lösungsmittels oder Lösungsmittelgemisches, „mixed solvent approach“
- Selektive Solvatation von Kationen oder Anionen durch Zugabe von Liganden
- Die Veränderungen der Ladungsverteilungen der Kationen durch elektronenschiebende Gruppen (+I-Effekt) oder der Anionen durch elektronenziehende Gruppen (-I-Effekt), das heist Verschmierung der Ladung

Die Variation der Kationen fällt bei der Optimierung von Lithium-Ionen-Batterien von vorne herein aus. Das in dieser Arbeit untersuchte LiBOB ist ein typischer Vertreter der Chelatoborate, die von Barthel und Gores [4] als Möglichkeit für die Optimierung der Leitfähigkeit durch Ladungsverschmierung vorgeschlagen wurden. Die Zugabe von Liganden wie Kronenethern für die selektive Solvatation des Lithiumions stellt eine Möglichkeit dar [1], sie ist aber für den Einsatzbereich in kommerziellen Lithium-Ionen-Batterien aufgrund ihres hohen Preises nur in Nischenbereichen denkbar und hat zwei weitere Nachteile. Erstens ist der Effekt nur bei Lösungen mit Lösungsmitteln sehr niedriger Dielektrizitätszahl bei kleinen Konzentrationen beachtlich, zweitens führt eine zu starke Solvatation des Lithium Ions oft zu Co-intercalation und damit zur Zerstörung des Graphits.

4.2.1.1 Leitfähigkeit der Elektrolytlösung

Die Leitfähigkeit von Elektrolyten wird stark von der Konzentration des Salzes, der Temperatur und dem Lösungsmittel oder der Lösungsmittelmischung beeinflusst.
Bei den in Lithium-Ionen-Batterien eingesetzten Elektrolyten handelt es sich in der Regel um konzentrierte Elektrolytlösungen. Daher wird hier für die Theorie der Leitfähigkeit der verdünnten Elektrolytlösungen auf die Literatur [2] bis [6] verwiesen.
Die spezifische Leitfähigkeit κ eines Elektrolyten hängt von seiner Äquivalentleitfähigkeit Λ ab (13). Diese wiederum wird durch die Ionenbeweglichkeit des Ions λ_i bestimmt (14).

$$\kappa = \lambda n_e c \tag{13}$$

$$\Lambda = \lambda_+ + \lambda_- = F\left(u_+ + u_-\right) \tag{14}$$

wobei F die Faradaykonstante und n_e die elektrochemische Valenz darstellt, für die bei einem binären Salz folgender Zusammenhang gilt:

$$n_e = v_+ z_+ = \left|v_- z_-\right| \tag{15}$$

mit z_+ und z_- als kationische und anionische Ladung und stöchiometrischen Koeffizienten v_+ und v_-.
Bei allen Elektrolyten weist die Abhängigkeit der spezifischen Leitfähigkeit von der Konzentration ein Maximum auf, wenn die Löslichkeit hinreichend groß ist [4]. Dieses Maximum kann bei Elektrolyten, die keine starke Assoziationen der Ionen zeigen, mit Gleichung (16) beschrieben werden [7].

$$d\kappa = n_e\left(\Lambda dc + c d\Lambda\right) = 0 \tag{16}$$

Dieses Maximum wird durch zwei gegenläufige Effekte hervorgerufen [4]. Zum einen wird die Ionenbeweglichkeit mit zunehmender Salzkonzentration herabgesetzt. Bei steigender Salzkonzentration hingegen tritt eine Erhöhung der Ladungsdichten auf, wodurch die Leitfähigkeit erhöht wird. Experimente zeigen, dass $d\Lambda < 0$ ist, wenn $dc > 0$ ist und die Bildung von Tripelionen ausgeschlossen werden kann. Dann ergibt sich für die Lage des Maximums folgender Zusammenhang.

$$\Lambda dc = \left|c d\Lambda\right| \tag{17}$$

Bei gegebener Temperatur lässt sich die Konzentrationsabhängigkeit der Leitfähigkeit mit der Casteel-Amis-Gleichung (18) beschreiben [8],[11], die sich auch sehr gut zur Datenreduktion eignet, da sie durch Parameter beschrieben wird, die für das Leitfähigkeitsmaximum charakteristisch sind, wie maximale spezifische Leitfähigkeit κ_{max} und die molale Konzentration μ, bei der diese erreicht wird.

$$\kappa = \kappa_{max}\left(\frac{m}{\mu}\right)^{a} \exp\left[b(m-\mu)^2\right] - \frac{a}{\mu}(m-\mu) \tag{18}$$

Die Fitparameter a und b[(mol/kg)$^{-2}$] haben dabei keine physikalische Bedeutung. Die vier Parameter κ_{max} , μ, a und b werden mittels der Methode der kleinsten Fehlerquadrate bestimmt. Details für das Fitverfahren und seine Grenzen werden von Barthel und Gores beschrieben [9]. Der Fitprozess kann auf mehrere Variablen ausgeweitet werden, wodurch eine noch bessere Datenreduktion erreicht wird. Beispielsweise haben Ding et al. [10] Polynome dritten Grades eingesetzt. Diese dienen für die Beschreibung der Abhängigkeit der Leitfähigkeit von der Lösungsmittelzusammensetzung[11] und der Salzkonzentration.

Im Allgemeinen weisen die Auftragungen von κ gegen m ein breites Maximum auf, wenn die Löslichkeit hinreichend groß ist [11]. Dieser Zusammenhang trifft sowohl für wässrige, nichtwässrige und gemischte Elektrolyte zu. Die Position des Maximums wird bei allen Lösungen mit zunehmender Temperatur zu höheren Konzentrationen hin verschoben [11].

In binären Lösungsmittelmischungen, die sich aus einem Lösungsmittel hoher Viskosität und hoher Dielektrizitätszahl wie beispielsweise PC oder DMSO und einem Lösungsmittel mit niedriger Viskosität und niedriger Dielektrizitätszahl wie DME oder EA zusammensetzen, verschiebt sich das Maximum der Leitfähigkeit in den Bereich hoher Konzentrationen, wenn der Anteil an Lösungsmittel mit niedriger DK zunimmt [2].

Bei Lösungsmitteln mit hoher Dielektrizitätszahl und damit vernachlässigbarer Ionenassoziation hängt die maximale Leitfähigkeit κ_{max} stark vom Ionenradius r_i ab. Bei unsolvatisierten Ionen können für diese Radien die kristallographischen Radien eingesetzt werden. Für solvatisierte Ionen, wie beispielsweise Li^+ hingegen wird der Stockes-Radius verwendet.

Für die Ionenbeweglichkeit λ_i gilt dann folgender Ansatz:

$$\lambda_i = \frac{F z_i e}{6\pi\eta r_i} \tag{19}$$

Anhand dieser Gleichung, die auf der Reibung von Kugeln in einem zähen Medium fußt, wird auch die Zunahme der Leitfähigkeit mit abnehmender Viskosität η deutlich.

Für die Temperaturabhängigkeit von Transporteigenschaften geschmolzener Salze und glasbildender Flüssigkeiten gilt die Vogel-Fulcher-Tammann (VFT) Gleichung (20):

$$\kappa = A\exp\left(\frac{B}{R(T-T_0)}\right) \tag{20}$$

Bei dieser Gleichung stellen die Parameter A und B charakteristische Funktionen für die jeweilige Transporteigenschaft wie Leitfähigkeit, Diffusionskoeffizient oder den Kehrwert der Viskosität dar. T_0 entspricht dabei der idealen Glasübergangstemperatur.

Eine detailliertere Diskussion der Leitfähigkeit, insbesondere die Rolle der Assoziation findet sich in der Literatur von Barthel und Gores [2], [4].

[11] Molenbruch einer Komponente eines Zweikomponentensystems

4.2.1.2 Mixed solvent approach

Ziel des „mixed solvent approach" ist es, durch Mischen geeigneter Lösungsmittel eine Lösungsmittelmischung herzustellen, die eine optimale Leitfähigkeit für ein bestimmtes Leitsalz aufweist. Acetonitril ist ein Lösungsmittel, welches sich aufgrund seiner relativ hohen Dielektrizitätszahl (ε=35,95 bei 25 °C) [2] und gleichzeitiger niedriger Viskosität (η=0,342 cP bei 25 °C) für diesen Anwendungsbereich in idealer Weise eignen würde. Darüber hinaus wird es wegen seiner elektrochemischen Stabilität in Doppelschichtkondensatoren als Lösungsmittel eingesetzt.

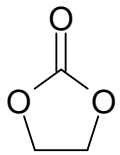
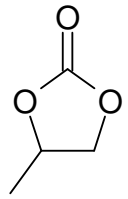
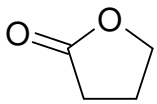
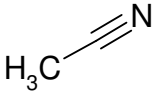
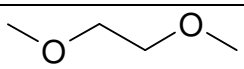
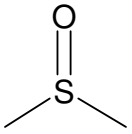
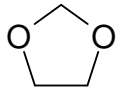
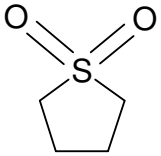
Lösungsmittel	Abk.	Strukturformel	θ_m/°C	θ_b/°C	ε	η / cP
Ethylencarbonat [1,3]Dioxolan-2-on	EC		36,5	238	90,36	1,9
Propylencarbonat 4-Methyl-[1,3]-Dioxolan-2-on	PC		-54,53	242	64,95	2,512
γ-Butyrolacton Dihydrofuran-2-on	γ-GBL		-43,53	204	39,1	1,7315
Acetonitril	AN		-48,835	81,60	35,95	0,341
1,2-Dimethoxyethan	DME		-58	84,50	7,075	0,407
Dimethylsulfoxid	DMSO		18,54	189	46,5	1,992
1,3-Dioxolan	DIOX		-97,22	76,5		0,6
Sulfolan Tetrahydrothiophen-1,1-dioxid	SL		28,45	287,3	43,30	10,287
Tetrahydrofuran	THF		-108,5	65,965	7,43	0,459

Tabelle 3 Lösungsmittel mit hoher Dielektrizitätszahl und hoher Viskosität

Da aber Acetonitril nicht kinetisch stabil gegenüber Lithium und den in Lithium-Ionen-Batterien eingesetzten Kathodenmaterialien ist, und darüber hinaus Cyanwasserstoff als Zersetzungsprodukt im Falle eines Versagens in der Batterie liefern kann, ist dieses Lösungsmittel ungeeignet für den

Einsatz in Lithium-Ionen-Batterien. Es hat sich gezeigt, dass sich die Lösungsmittel, die sich aufgrund ihrer Stabilität für Lithium-Ionen-Batterien eigenen, in zwei Klassen eingeteilt werden können. Zum einen gibt es Lösungsmittel mit hoher Viskosität und Dielektrizitätszahl (siehe Tabelle 3), zum anderen solche, die eine niedrige Viskosität und Dielektrizitätszahl (siehe Tabelle 4) aufweisen [2].

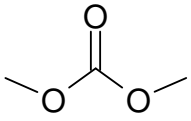
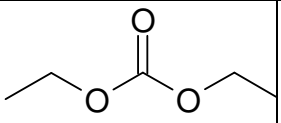
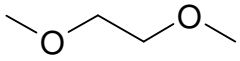
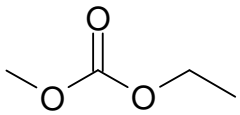
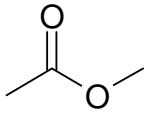
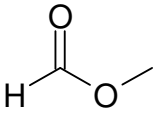
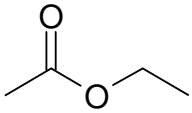
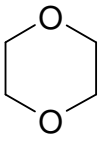
Lösungsmittel	Abk.	Strukturformel	θ_m/°C	θ_b/°C	ε	η / cP
Dimethylcarbonat Kohlensäure-dimethylester	DMC		4,6	90	3,1075	0,5902
Diethylcarbonat Kohlensäure-Diethylester	DEC		-43,0	126,8	2,8059	0,7529
1,2-Dimethoxyethan	DME		-58	84,50	7,075	0,407
Ethylmethylcarbonat Kohlensäure-ethylmethylester	EMC				2,4	0,65
Tetrahydrofuran	THF		-108,5	65,965	7,43	0,459
Methylacetat Essigsäure-methylester	MA		-98,05	56,868	6,68	0,364
Methylformiat Ameisensäure-methylester	MF		-99,0	31,75	8,5	0,328
Ethylacetat Essigsäure-ethylester	EA		-87,75 [12]	75,5 [13]	6,03 [14]	0,44 [15]
1,4-Dioxan	DION		11,79 [16]	100,8 [17]	3,20 [18]	1,19 [19]

Tabelle 4 Lösungsmittel mit kleiner Dielektrizitätszahl und niedriger Viskosität

Die in diesen Tabellen dargestellten Werte, wurden soweit nicht anders angegeben aus [4] entnommen.

Ziel der Leitfähigkeitsoptimierung ist es, die hohe Viskosität von Lösungsmitteln mit hoher Dielektrizitätszahl wie zum Beispiel Propylencarbonat durch Zugabe eines Lösungsmittels mit niedriger Viskosität wie beispielsweise Dimethylcarbonat herabzusetzen. Als Konsequenz daraus wird die Dielektrizitätszahl ebenfalls reduziert [2]. Gelingt es, die Mischung so einzustellen, dass die Assozia-

tion noch keine Rolle spielt, kann eine Leitfähigkeitssteigerung um mehr als 100 % bei Raumtemperatur und ungefähr 1000 % bei tiefen Temperaturen erreicht werden [20].
Eine Möglichkeit, optimale Lösungen zu entwickeln, besteht darin, die Leitfähigkeit bei optimaler Salzkonzentration κ_{max} gegen die Lösungsmittelzusammensetzung aufzutragen. Mit zunehmendem Anteil an Lösungsmittel mit niedriger Viskosität wird eine Zunahme der Leitfähigkeit beobachtet, bis ein Maximum κ^*_{max} erreicht wird. Danach fällt die Leitfähigkeit aufgrund von zunehmender Assoziation des Elektrolyten [20]. Diese Methode findet in der Literatur breite Anwendung [2].
Ein weiterer Vorteil in der Verwendung von Mischungen von Elektrolytlösungen besteht darin, dass der Flüssigkeitsbereich des Elektrolyten für den jeweiligen Einsatzbereich bestmöglich eingestellt werden kann. Lösungsmittel mit niedriger Viskosität besitzen meist auch einen niedrigen Schmelzpunkt. Soll der Elektrolyt bei tiefen Temperaturen eingesetzt werden, wird daher ein großer Anteil dieser Lösungsmittel verwendet. Bei hohen Temperaturen weisen diese Lösungsmittel aber einen hohen Dampfdruck auf. Für diesen Einsatzbereich empfiehlt sich daher der Einsatz von Lösungsmitteln mit hoher Dielektrizitätszahl, die meist einen hohen Siedepunkt aufweisen.
Ein weiterer ganz wesentlicher Vorteil in der Verwendung von Mischungen wird durch eine wesentlich größere Gefrierpunktserniedrigung und Siedepunktserhöhung erreicht, als dies durch das Salz in einem reinen Lösungsmittel alleine der Fall wäre, da hier noch die Gefrierpunktserniedrigung der weiteren Lösungsmittelkomponenten zum tragen kommt. Durch diese günstigen Effekte wird der Flüssigkeitsbereich des Elektrolyten noch zusätzlich erweitert.
Die Optimierung der Leitfähigkeit von Mischungen, die sich aus mehreren Lösungsmitteln zusammensetzen, stellt eine umfangreiche Aufgabe dar, da eine Vielzahl von Systemen untersucht werden muss, obwohl es einige Daumenregeln für die Optimierung gibt [2], [4], [21] bis [25]. Es muss für jede Mischung und für jede Konzentration an Salz die Leitfähigkeit bei verschiedenen Temperaturen gemessen werden. Soll eine Lösung optimiert werden, die sich aus vier Lösungsmittelkomponenten und einem Salz zusammensetzt, so handelt es sich dabei um ein fünfdimensionales Optimierungsproblem, bei dem die Leitfähigkeit maximiert wird. Bei den hier untersuchten Systemen wurde nur die Leitfähigkeit einer konstanten Temperatur, –25 °C optimiert, welche normalerweise als untere Grenze für den Einsatzbereich von Lithium-Ionen-Batterien gilt [25]. Es stellt sich also die Frage nach einer Methode, die es ermöglicht, das Optimierungsproblem möglichst schnell und effektiv zu lösen.

4.2.1.3 Simplexmethode

Der Simplexalgorithmus ist eine mathematische Methode, die für die Lösung von vielfältigen Optimierungsproblemen in Wirtschaft, Wissenschaft und Technik eingesetzt wird. Sie ist besonders geeignet, wenn eine oder mehrere Zielgrößen optimiert werden sollen, die über viele Einflussgrößen kontrolliert werden, wobei der Zusammenhang zwischen den Einflussgrößen und den Zielgrößen nicht bekannt ist.
Die Simplexmethode wurde 1947 von G. B. Dantzig für die Lösung von Optimierungsproblemen der US Air Force entwickelt [26] und von Spendley et al. als „basic simplex-method“ weiterentwickelt [27]. In einer Reihe von Arbeiten wurden Modifikationen dieser Methode vorgestellt, [28] bis [30], wie beispielsweise die Erweiterung auf mehrere Zielgrößen von L. A. Zadeh [31].
Die Methode ist nach der geometrischen Figur Simplex benannt, die eine Verallgemeinerung des Tetraeders aus dem dreidimensionalen Raum in den n-dimensionalen Raum darstellt. Jeder Simplex besitzt n+1 Ecken, wenn n Kontrollgrößen variiert werden sollen. Für jeden Optimierungschritt

wird bei der „basic simplex-method“ der Simplex durch Reflexion an der Seite gespiegelt, die der Ecke gegenüberliegt, bei der die Zielvariable bei einer Maximierung den kleinsten Wert ausweist[12]. Dadurch entsteht ein neuer Simplex, bei dem alle Ecken bis auf die Ecke des Simplexes mit dem schlechtesten Wert für die Zielgröße übereinstimmen. Diese Ecke wird durch diejenige ersetzt, die durch die Reflexion erzeugt wird. Die Koordinaten der neuen Ecke stellen die Parameter für den nächsten Versuchsschritt dar, mit denen der Wert für die Zielgröße an dieser Stelle ermittelt wird. Nun wird wieder an der Stelle gespiegelt, die der Ecke gegenüberliegt, welche den schlechtesten Wert für die Zielgröße aufweist, wodurch wieder neue Bedingungen für den nächsten Versuch gewonnen werden. Durch Wiederholung dieser Schritte wird eine stufenweise Annäherung an ein lokales Maximum erreicht. Durch die Reflexionen werden in der Regel alle Einflussgrößen variiert, wodurch sich eine starke Reduktion der Anzahl der Versuche ergibt.
Bei der herkömmlichen Methode wird ein Simplex mit fester Größe eingesetzt, der nur reflektiert wird. Wird dieser Simplex zu groß gewählt, so ergibt sich nur eine grobe Analyse des Systems, und lokale Minima werden nur ungenau bestimmt, wenn nicht sogar verfehlt. Wird hingegen mit einem zu kleinen Simplex gearbeitet, so wird vor allem zu Anfang der Optimierung eine zu hohe Anzahl an Optimierungsschritten benötigt.
Eine Lösung dieser Probleme stellt die Simplex-Methode mit variabler Simplexgröße dar. Bei dieser Methode werden neben Reflektionen noch Expansionen und Kontraktionen des Simplexes zur Bestimmung von neuen Werten für die Kontrollvariablen eingesetzt.
In Abbildung 30 wird die Expansion mit E bezeichnet, die eine Reflektion (R) mit einem konstanten Faktor darstellt, der im voraus gewählt wird. Die Kontraktion C_+ entspricht ebenfalls einer Reflektion, wobei im Gegensatz zur Expansion der Faktor kleiner als eins ist.

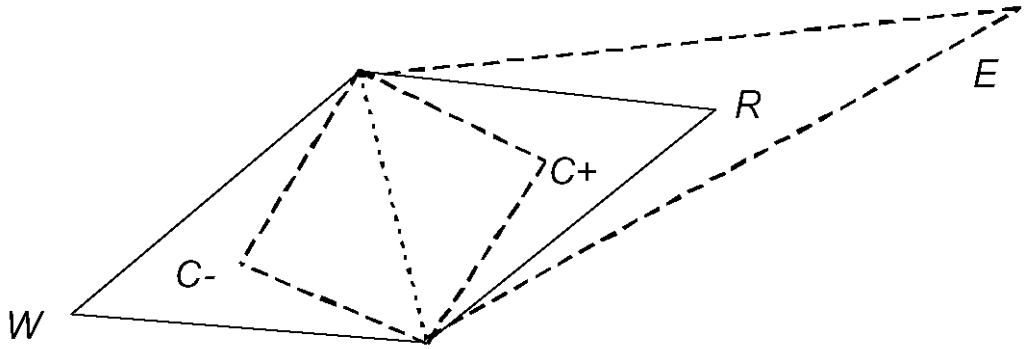


Abbildung 30 Simplexmethode mit variabler Schrittgröße bei zwei Kontrollvariablen [30]

Die Kontraktion kann auch ohne Reflexion C_- durchgeführt werden, zum Beispiel, wenn die neuen Kontrollvariablen näher an der Menge der vorherigen Kontrollvariablen liegen sollen. Für weitere Details zu dieser Methode und über weitere Verbesserungen sei hier auf die Literatur verwiesen [30] bis [34].

[12] Der Klarheit wegen wird hier nur die Suche eines Maximums beschrieben. Für eine Minimierung, für die sich die Simplexmethode genauso eignet, wird natürlich an der Seite gespiegelt, die der Ecke gegenüberliegt, bei der die Zielvariable den größten Wert aufweist. Aus diesem Grund wird auch immer bei der Beschreibung der Optimierung von der Suche des Maximums ausgegangen, obwohl die Überlegungen sinngemäß auch für Minima gelten.

Die Simplexmethode weist zwei Vorteile auf. Der erste positive Aspekt besteht darin, dass keine Funktion bekannt sein muss, die den Zusammenhang zwischen den Kontrollvariablen und den Zielvariablen beschreibt, da es sich bei dieser Methode um ein rein geometrisches Verfahren handelt, bei dem nur gemessene Größen betrachtet werden. Die Startpunkte können beliebig gewählt werden. Mit der Simplexmethode wird immer ein lokales Maximum gefunden, sofern eines existiert. Der zweite Vorteil liegt in der immensen Ersparnis an Zeit und Mitteln durch eine starke Reduktion der Anzahl der durchgeführten Versuche, besonders dann wenn viele Kontrollvariablen berücksichtigt werden sollen. Sollte beispielsweise die Leitfähigkeit das System EC/PC/DMC/EMC/EA und LiBOB optimiert werden, so müssten mehr als 1000 Lösungen hergestellt und vermessen werden, wenn die Zusammensetzung in 10 % Schritten variiert werden soll. Durch den Einsatz von Interpolationsmethoden ließe sich die Zahl auf einige Hundert reduzieren. Wie in Kapitel 4.2.3 gezeigt wird, benötigt man mit der Simplexmethode nur 20 Messungen, um das Ziel zu erreichen.
Der einzige Nachteil dieser Methode liegt darin, dass keine Information über weitere lokale Minima, sofern vorhanden, gewonnen wird.
Die Optimierung der Leitfähigkeit von Elektrolytlösungen ist ein Problem, für dessen Lösung sich die Simplexoptimierung ideal einsetzen lässt. Als Kontrollgrößen dienen hier die Zusammensetzung des Lösungsmittelgemisches und die Konzentration des Elektrolyten. In den hier gezeigten Beispielen sind die Kontrollgrößen m_s, ξ_1, ξ_2, ξ_3 und ξ_4 bei einem System aus einem Salz und fünf Lösungsmittelkomponenten. Die Zielgröße stellt die Leitfähigkeit bei –25,00 °C $\kappa_{-25°C}$ dar, deren Maximum gesucht werden soll. Die molale Konzentration an Salz wird dabei mit m_s bezeichnet (mol/kg Lösungsmittelmischung). ξ_1, ξ_2, ξ_3 und ξ_4 sind die Massenanteile der Lösung in Massenprozent. Der Anteil der fünften Lösungsmittelkomponente muss dabei nicht berücksichtigt werden, da er aufgrund der Mischungsbedingung $\Sigma\, \xi_i$=100%, durch die Summe der anderen drei Anteile bestimmt ist. Das hier gezeigte Beispiel stellt ein sechsdimensionales Optimierungsproblem dar. Da aber mehrdimensionale Funktionen mehrere lokale Maxima aufweisen können, ist mit der Simplexoptimierung nicht gesichert, dass das globale Maximum, nämlich die maximale Leitfähigkeit bei -25,00 °C $\kappa_{-25°C}$, gefunden wird. Daher ist es sinnvoll, die Optimierung mit deutlich unterschiedlichen Startsimplices durchzuführen und zu testen, ob das gleiche Maximum gefunden wird.
Für die Simplexoptimierung wird hier die Software „Multisimplex“ in der Version 2.1 der Firma Grabitech Solutions AB (Sundsvall) eingesetzt. Als Methode wurde das Simplexverfahren mit variabler Simplexgröße verwendet, um möglichst schnell genaue Ergebnisse zu erhalten. Der Reflexionskoeffizient R wurde auf 1 gesetzt, die Kontraktionskoeffizienten C_+ und C_- betrugen jeweils ½ R. Für den Expansionskoeffizienten E wurde ein Wert von 2 R angenommen.

Test der Simplexmethode

Vor dem Einsatz des Multisimplexprogramms wurde eine Testreihe mit den Daten von Ding und Jow [35] durchgeführt. Diese Daten beschrieben die Leitfähigkeiten von $LiPF_6$ in EC/PC und PC/DEC Mischungen bei verschiedenen Temperaturen. Aus den erhaltenen Daten konnten die Autoren für jede der Temperaturen den Zusammenhang zwischen der Molalität m des Leitsalzes $LiPF_6$, dem Massenanteil w von EC und der spezifischen Leitfähigkeit κ ermitteln. Mit der Interpolationsfunktion (21) können die gefundenen Messwerte interpoliert werden [35]

$$\ln(\kappa) = (a_0 + a_1 w + a_2 w^2)\ln(m) + (b_0 + b_1 w + b_2 w^2) + (c_0 + c_1 w + c_2 w^2)m + (d_0 + d_1 w + d_2 w^2)m^2 \quad (21)$$

wobei κ die spezifische Leitfähigkeit in µS/cm, w den Massenanteil an EC, und m die Molalität in mol/kg an $LiPF_6$ in der Lösungsmittelmischung darstellt. Die Parameter a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, und d_2 sind temperaturabhängige Fitparamter. Die Zahlenwerte dieser Parameter sind in [35] angegeben. Dabei ist zu beachten, dass die Anpassung mit Daten im Bereich von m = 0,4 bis 2,3 mol/kg und w = 0 bis 0,6 durchgeführt wurde. Außerhalb dieses Bereiches sind keine verlässlichen Aussagen über die Leitfähigkeit möglich, was jedoch für den Test des Multisimplexprogramms ignoriert wurde. Die Berechnung der Leitfähigkeiten erfolgte aus der angegebenen Gleichung, wozu das Mathematikexpertensystem Maple 7 eingesetzt wurde. Der Test wurde mit den Parametern für 59 °C durchgeführt, weil bei dieser Temperatur Messwerte auf beiden Seiten des Leitfähigkeitsmaximums vorhanden sind. In Abbildung 31 ist die Abhängigkeit der Leitfähigkeit von der molalen $LiPF_6$- Konzentration und von der Zusammensetzung des Lösungsmittelgemisches für diese Temperatur dargestellt.

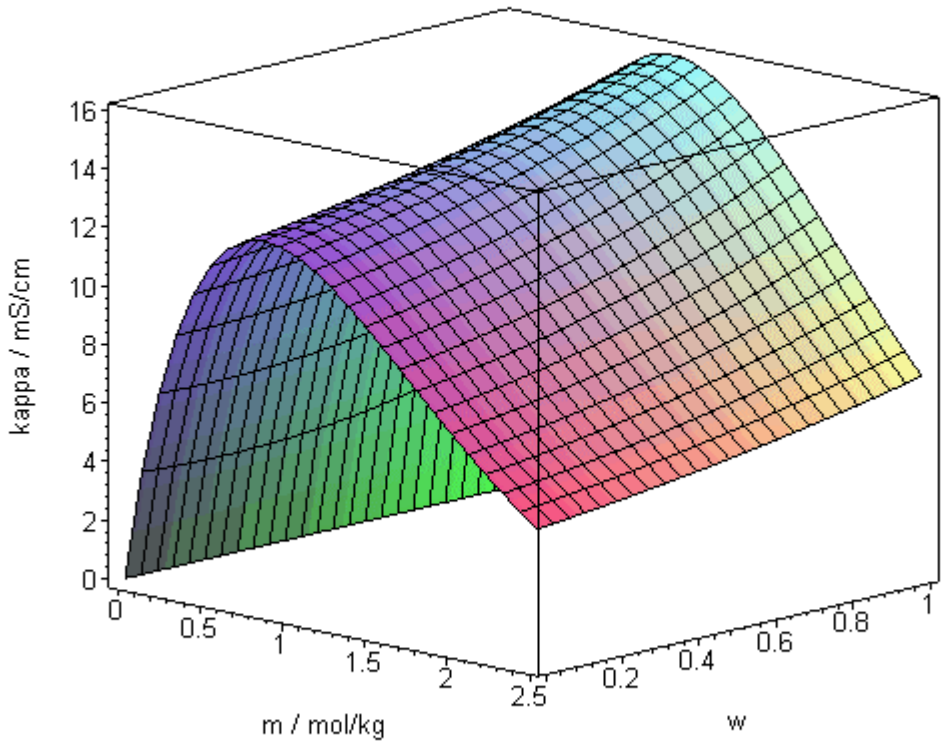


Abbildung 31 Berechnete Leitfähigkeit in Abhängigkeit von LiPF6-Molalität und EC-Gehalt

Das Maximum dieser Kurve wurde durch Nullstellensuche in der ersten Ableitung bestimmt und liegt bei m = 0,985 mol/kg $LiPF_6$ und w = 1. Dabei wird eine Leitfähigkeit von 15,92 mS/cm erreicht.

Um die Eignung der Simplexoptimierung für dieses Problem zu testen, wurde nun versucht, dieses Maximum auch mit dem Multisimplexprogramm zu finden. Als Variablen wurden die beiden Lösungsmittelkomponenten als Mischungskomponenten in Gewichtsprozent der Lösungsmittelmischung und das $LiPF_6$ in mol/kg. eingeführt. Als Startwerte zur Berechnung des ersten Simplexes

wurde eine Mischung von 0,5 mol/kg LM $LiPF_6$ in einer 1:1 Mischung aus EC/PC gewählt, da diese Mischung, wie aus obigem Abschnitt hervorgeht, weit weg vom Leitfähigkeitsmaximum liegt. Als Zielgröße für die Optimierung diente die spezifische Leitfähigkeit, wobei für die Optimierung die modifizierte Simplexmethode ausgewählt wurde. Mit dem Multisimplexprogramm wurde zuerst ein Startsimplex berechnet, der in Tabelle 5 dargestellt ist. Für die jeweilige Mischung wurde dann die spezifische Leitfähigkeit κ nach Gleichung (21) bestimmt.

m_{LiPF6} / (mol/kg)	ξ_{PC} / %	ξ_{EC} / %	κ / (mS/cm)
0,25	25	75	8,48
0,733	37,941	62,059	13,92
0,379	73,296	26,704	9,92

Tabelle 5 Startsimplex

Ausgehend von diesem Startsimplex und den berechneten Leitfähigkeiten wurde dann mittels des Simplexalgorithmus eine neue Zusammensetzung bestimmt, für die dann die Leitfähigkeiten erneut nach der Fitfunktion (21) bestimmt wurden. Mit dieser Methode wurden nun zwanzig Optimierungsschritte durchgeführt, die in Tabelle 6 dargestellt sind.

m_{LiPF6} / (mol/kg)	ξ_{PC} / %	ξ_{EC} / %	κ / (mS/cm)
0,250	25,000	75,000	8,48
0,733	37,941	62,059	13,92
0,379	73,296	26,704	9,92
0,862	86,237	13,763	12,80
1,216	50,882	49,118	13,01
1,087	2,586	97,414	15,68
1,063	35,573	64,427	14,16
1,417	0,218	99,782	14,62
1,157	18,488	81,512	14,67
0,827	20,856	79,144	14,81
0,757	4,954	95,046	15,22
0,874	12,313	87,687	15,25
1,204	9,945	90,055	14,96
1,087	2,586	97,414	15,69
0,869	6,202	93,798	15,51
0,926	8,354	91,646	15,49
1,144	4,738	95,262	15,44
0,938	5,836	94,164	15,61
1,099	0,068	99,932	15,81
1,015	3,582	96,418	15,72
1,027	1,064	98,936	15,85
0,997	0,303	99,697	15,90

Tabelle 6 Optimierung der Leitfähigkeit

Die Zunahme der Leitfähigkeit durch die Simplexoptimierung wird in Abbildung 32 deutlich, in der die Leitfähigkeit gegen die Zahl der Experimente aufgetragen ist.

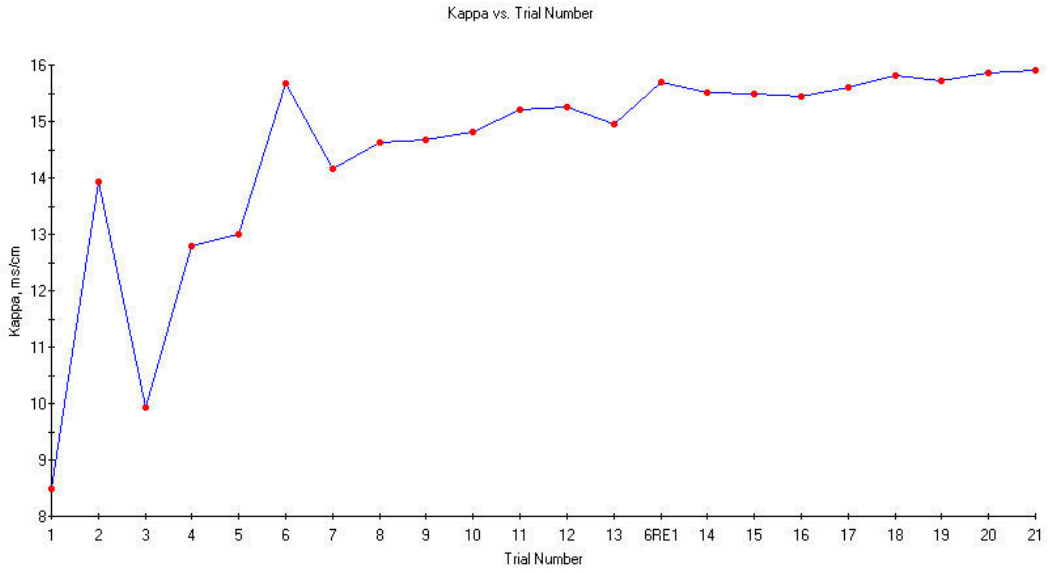


Abbildung 32 Test der Optimierung der Leitfähigkeit mit der Simplexmethode

Mit dem Multisimplexprogramm wird nach 20 Schritten ein Leitfähigkeitsmaximum von 15,90 mS/cm bei einer Zusammensetzung von 0,997 mol/kg $LiPF_6$ in einer Lösung aus 0,30 % PC und 99,70 % EC erhalten. Wie anhand Kurvendiskussion ermittelt, beträgt das Leitfähigkeitsmaximum 15,92 mS/cm bei einer Zusammensetzung aus 0,985 mol/kg $LiPF_6$ in 100% EC.
Da durch die Simplexoptimierung sehr schnell eine sehr gute Annäherung an das Maximum erreicht wurde, könnte diese Methode auch für die Optimierung von aktuell untersuchten Elektrolytlösungen eingesetzt werden, die LiBOB als Leitsalz enthalten.

4.2.2 Versuchsdurchführung

Durch Hydrolyse des eingesetzten Salzes und durch Wasserspuren in den Lösungsmitteln kann eine deutliche Verfälschung der gemessenen Leitfähigkeiten in der Größenordnung von einigen hundert Prozent verursacht werden [4]. Aus diesem Grund ist vor allem bei der Messung von Leitfähigkeiten der Ausschluss einer Kontamination durch Feuchtigkeit von äußerster Wichtigkeit. Daher wurden sämtliche durchgeführten Arbeitschritte, bis auf die Leitfähigkeitsmessung selbst, im Handschuhkasten unter Argon als Schutzgas durchgeführt (Siehe Kapitel 11.1.1.1). Für die Lösungsmittelmischungen wurden Ethylencarbonat (EC), Propylencarbonat (PC), Dimethylcarbonat (DMC), Ethylmethylcarbonat (EMC) und Ethylacetat (EA) eingesetzt. Um Fehler durch Verunreinigungen durch die Lösungsmittel ausschließen zu können, wurden ausschließlich Lösungsmittel der Qualität Selectipur® der Firma Merck (Darmstadt) eingesetzt. Der Wassergehalt dieser Lösungsmittel wurde mittels Karl-Fischer-Titration, siehe Kapitel 11.1.6, bestimmt. Er lag bei allen Lösungsmitteln unter 30 ppm. Als Salz wurde LiBOB der Firma Chemetall (Frankfurt) verwendet. Da zu diesem Zeitpunkt die Charge SCHK02/09 die reinste zur Verfügung stehende Charge war, wurde diese Charge für die Leitfähigkeitsoptimierung eingesetzt.
Zur Herstellung der Lösungen im Handschuhkasten wurde eine Analysenwaage von Mettler Toledo (Greifensee) vom Typ AB 204 verwendet. Aufgrund von Druckschwankungen innerhalb des

Handschuhkastens wird eine Genauigkeit von ±1 mg für die Massenbestimmung angenommen. Der Hersteller gibt einen Fehler von ±0,2 mg an.
Die Leitfähigkeiten wurden mit einer Leitfähigkeitsmessbrücke bestimmt, die von Barthel et. al. beschrieben wurde [36]. Sie setzt sich aus einer symmetrischen Kohlrauschbrücke mit Wagnerschem Hilfszweig, einem Frequenzgenerator und einer Präzisionswiderstandsdekade der Firma Tettex (Zürich) zusammen, mit welcher der Widerstand R der Elektrolytlösung in der Messzelle bestimmt wird. Durch frequenzabhängige Messung der Leitfähigkeit mit anschließender Extrapolation auf unendlich hohe Frequenzen ist es möglich, mit dieser Anlage eine Genauigkeit im Bereich von 0,01 % zu erreichen [36]. Da für die Optimierungsmessungen keine so große Präzision erforderlich ist, wurden alle Messungen bei einer konstanten Frequenz von 5 kHz durchgeführt, wodurch die Genauigkeit der Messung auf weniger als 0,2 % reduziert wird.
Für die Messungen wurde eine miniaturisierte Version der von Barthel et al. [36] beschriebenen Messzellen eingesetzt, um den Verbrauch an Lösungsmitteln und Salz so gering wie möglich zu halten.

Abbildung 33 Leitfähigkeitsmesszelle

Diese Messzellen, die in Abbildung 33 dargestellt sind, benötigen nur noch 3 cm^3 an Elektrolytlösung. Die Zellkonstanten der Messzellen wurden mittels Kreuzeichung bestimmt und sind in Tabelle 7 aufgeführt.

Zelle	B/cm^{-1}
1A	27,829
2	38,247
3	47,783
4	57,509
5	65,175
6	89,623
7	47,404
8	52,303
9	50,818
10	50,986

Tabelle 7 Zellkonstanten der Messzellen

Anhand dieser Zellkonstanten wird die spezifische Leitfähigkeit der Elektrolytlösung aus dem gemessenen Widerstand R mithilfe folgender Gleichung bestimmt:

$$\kappa = B/R \quad (22)$$

Die Messzellen wurden mit einer von Barthel und Wachter beschriebenen [37] Thermostatenanlage thermostatisiert, mit der eine Stabilität von ± 3 mK erreicht wird. Die Temperatur im Thermostatenbad wurde mittels eines ALS F-250 MkII Thermometers der Firma Automatic Systems Laboratories (Milton Keynes) bestimmt. Dieses Thermometer wurde von der Firma WYCO (ROMSAY) gemäss dem UKAS Standard kalibriert, wodurch eine rückführbare Temperaturbestimmung mit einer Genauigkeit von 20 mK erreicht wird.
Für die Optimierung der Leitfähigkeiten der einzelnen Lösungen wurden die Leitfähigkeiten bei -25,00 °C gemessen. Zusätzlich wurden noch die Leitfähigkeiten bei +25,00 °C bestimmt.

4.2.3 Ergebnisse

Im Gegensatz zu dem in Kapitel 4.2.1.3 beschriebenen Test werden hier die Ergebnisse der Optimierungen gemessener Leitfähigkeiten realer Systeme präsentiert. Die Werte für die Kontrollvariablen, die Zusammensetzung der Elektrolytmischung und ihr Gehalt an Leitsalz, wurden mit dem Programm Multisimplex ermittelt. Grundlage für die Optimierung war dabei die spezifische Leitfähigkeit bei –25,00°C, die nach den im Kapitel 4.2.2 beschriebenen Verfahren ermittelt wurde. Neben der Optimierung der Leitfähigkeit von LiBOB in beiden Carbonatmischungen bestehend aus EC/PC/DMC oder EC/PC/DMC/EMC, wurde noch das Optimum der Leitfähigkeit der ethylacetathaltigen LiBOB Elektrolytlösungen mit den Komponenten EC/PC/DMC/EA oder EC/PC/DMC/EMC/EA bestimmt.

4.2.3.1 Optimierung der Leitfähigkeit von LiBOB in EC/PC/DMC Mischungen

In Tabelle 8 sind die Ergebnisse der Optimierung der Leitfähigkeit der Elektrolytlösung LiBOB in EC/PC/DMC aufgeführt. Der Startpunkt für diese Optimierung war eine Lösung von 0,6 mol/kg LiBOB in einer Lösungsmittelmischung mit gleichen Anteilen an Carbonaten. In den ersten vier Zeilen der Tabelle 8 ist der Startsimplex (F) angegeben. In Spalte 8 dieser Tabelle ist die Art des Optimierungsschritts angegeben, der in Abschnitt 4.2.1.3 näher beschrieben wird. Die Temperaturen, die dem ersten beobachteten Knickpunkt zugeordnet werden, der zugleich die tiefstmögliche Temperatur darstellt, bei welcher der Elektrolyt in einer Batterie sicher eingesetzt werden kann, sind in Spalte 9 dieser Tabelle angegeben. Die Bestimmung dieser Temperatur ist in Abschnitt 4.3 beschrieben.

i	m_{LiBOB} / (mol/kg)	ξ_{EC} / %	ξ_{PC} / %	ξ_{DMC} / %	$\kappa_{-25°C}$ / (mS/cm)/	$\kappa_{+25°C}$ / (mS/cm)	Schritt	θ_{KP} / °C
1	0,50	43,30	23,30	33,40	1,06	5,95	F	-9,1
2	0,50	23,30	43,30	33,40	1,09	5,86	F	<-30
3	0,70	23,30	23,30	53,40	0,63	6,99	F	-25,5
4	0,70	43,30	43,30	13,40	--[13]	4,63	F	-25,0
5	0,43	16,63	16,63	66,74	--[13]	6,37	R	-2,2
6	0,63	36,63	36,63	26,74	0,78	5,61	C-	<-30
7	0,39	45,52	45,52	8,96	0,64	4,23	R	-16,2
8	0,47	39,96	39,96	20,08	0,82	5,09	C+	<-30
9	0,35	34,41	34,41	31,18	1,08	5,15	R	-27,6
10	0,43	27,38	27,38	45,24	1,39	6,19	R	-21,9
11	0,41	21,09	21,09	57,82	0,75	6,41	E	-10,6
12	**0,35**	**13,43**	**46,76**	**39,81**	**1,27**	**5,45**	**R**	**-25,5**
13	0,51	8,33	43,88	47,79	1,11	6,52	R	-18,4
14	0,36	9,46	35,38	55,16	0,80	5,90	R	-15,4
15	0,50	23,30	43,30	33,40	1,08	5,85	F	<-30
16	**0,47**	**19,84**	**41,30**	**38,80**	**1,42**	**6,59**	**C-**	**-22,5**
17	0,32	32,10	33,09	10,44	1,15	5,18	R	-28,8
18	0,37	26,16	35,79	38,05	1,24	5,59	C+	-20,6bis-24,6
19	0,49	35,49	22,90	41,61	1,26	6,33	R	-16,0
20	0,56	28,98	25,28	45,74	1,04	6,54	R	-27,8
21	0,42	26,87	33,16	39,97	1,26	5,91	C-	-23,3
22	0,43	27,38	27,38	45,24	1,05	5,93	R	<-30
23	0,47	30,09	28,43	41,48	1,26	6,15	C-	-21,3
24	0,49	24,67	31,59	43,79	1,31	6,29	R	-21,09
25	0,52	22,35	40,18	37,47	1,16	6,10	R	-25,9
26	0,50	26,61	36,98	39,41	1,15	6,03	C+	-26,5
27	0,50	15,32	44,83	39,85	1,23	6,17	R	-24,7
28	0,48	16,28	41,51	42,21	1,25	8,40	R	<-30
29	0,45	25,21	31,45	43,34	1,12	6,32	R	-20,3bis-25,9
30	0,49	17,79	41,48	40,73	1,25	6,19	C-	-21,5

Tabelle 8 Ergebnisse der Optimierung der Leitfähigkeit des Systems LiBOB in EC/PC/DMC

Für die Variation des Startsimplexes wurde eine Variation von 20% für die Komponenten des Lösungsmittels und 0,2 mol/kg für das Salz verwendet. Diese große Variation zu Beginn der Optimierung sollte ein möglichst schnelles Fortschreiten der Optimierung zu Beginn gewährleisten. Aufgrund der variablen Simplexgröße ist aber eine hohe Auflösung am Ende der Optimierung gewährleistet, wodurch eine genaue Bestimmung des Maximums sichergestellt ist.

[13] Komplett gefroren, keine Leitfähigkeit messbar.

Das Leitfähigkeitsmaximum für –25,00 °C $\kappa_{opt,-25\ °C}$ =1,42 mS/cm wird für eine Lösungsmittelmischung der Zusammensetzung 19,84 % EC, 41,30 % PC und 38,80 % DMC mit einer LiBOB Konzentration von 0,47 mol/kg erreicht. Interessanterweise zeigen die Abkühl- und Aufheizexperimente, dass diese Lösung schon einen Knickpunkt bei –22,5 °C aufweist. Bei –25,00 °C, bei der die Leitfähigkeit gemessen wurde, ist also bereits ein Teil der Mischung als Feststoff ausgefallen. Die Mischung mit einer Zusammensetzung von 13,43 % EC, 46,76 % PC und 39,81 % DMC mit einem LiBOB Gehalt von 0,35 mol/kg hingegen zeigt erst einen Knickpunkt bei –25,0 °C und stellt damit bezogen auf die Leitfähigkeit, die optimale Zusammensetzung für einen Elektrolyten dar, der sich aus LiBOB, EC, PC, und DMC zusammensetzt.
Bei höheren Salzgehalten werden wieder geringere Leitfähigkeiten beobachtet. Bei höheren Gehalten an Lösungsmitteln mit geringer dynamischer Viskosität wie DMC, 0,0585 kg/(m*s) [38], steigt die Leitfähigkeit deutlich an. Es ist wahrscheinlich, dass die Zunahme der Leitfähigkeit auf die Herabsetzung der Viskosität zurückzuführen ist, wodurch die Stokesche Reibung reduziert wird. Dies ist ein typisches Verhalten von Salzen mit schwach koordinierenden Anionen wie LiBOB, bei denen die niedrige Viskosität des Lösungsmittels der Garant für eine hohe Leitfähigkeit ist. Im Gegensatz dazu ist bei nichtdelokalisierten Anionen die Ionenassiozation, die maßgeblich durch die dielektrische Permittivität beeinflusst wird, ausschlaggebend.

4.2.3.2 Optimierung der Leitfähigkeit von LiBOB in EC/PC/DMC/EMC Mischungen

Um den Einsatzbereich des Elektrolyten im Tieftemperaturbereich weiter ausdehnen zu können, wird EMC als weiteres Carbonat mit niedriger Viskosität zu den drei Komponenten der genannten Lösungsmittelmischung hinzugefügt. Der Startpunkt dieser Optimierung war der gleiche wie bei der vorherigen Mischung (0,6 mol/kg Salz und gleiche Mengen an Lösungsmitteln). Die Schrittweite betrug hier 0,3 mol/kg für das Salz und 12 % für die Lösungsmittel.

i	m_{LiBOB} / (mol/kg)	ξ_{EC} / %	ξ_{PC} / %	ξ_{DMC} / %	ξ_{EMC} / %	$\kappa_{-25°C}$ / (mS/cm)	$\kappa_{+25°C}$ / (mS/cm)	Schritt	θ_{KP} / °C
1	0,45	31,0	19,0	19,0	31,0	1,29	5,65	F	<-30°C
2	0,75	31,0	31,0	19,0	19,0	0,85	5,87	F	<-30°C
3	0,75	31,0	19,0	31,0	19,0	1,10	6,45	F	<-30°C
4	0,45	19,0	31,0	31,0	19,0	1,42	6,08	F	<-30°C
5	0,75	19,0	19,0	19,0	43,0	1,25	6,28	F	<-30°C
6	0,45	19,0	13,0	31,0	37,0	1,61	5,92	R	<-30°C
7	0,30	13,0	4,0	37,0	46,0	1,17	3,60	E	<-30°C
8	0,30	13,0	22,0	19,0	46,0	1,41	4,76	R	<-30°C
9	0,08	22,0	23,5	31,0	23,5	0,67	2,10	R	<-30°C
10	0,58	19,75	20,1	22,0	38,1	1,42	6,29	C-	<-30°C
11	**0,44**	**4,38**	**24,1**	**32,5**	**39,1**	**1,70**	**5,91**	**R**	**<-30°C**
12	0,66	18,06	22,1	39,3	20,6	1,43	6,76	R	<-30°C
13	0,62	11,59	8,64	31,4	14,5	1,44	5,42	R	<-30°C
14	0,50	6,77	13,77	45,06	34,40	1,56	5,31	R	-25,1
15	0,34	2,81	7,65	30,72	58,83	--[14]	--	R	>+25,0
16	0,58	14,25	18,48	31,12	30,15	1,57	6,38	C-	<-30°C
17	0,37	10,60	26,02	41,47	21,92	1,52	6,02	R	<-30°C
18	0,43	10,85	21,67	38,94	28,53	1,62	5,84	C+	<-30°C
19	0,45	17,47	24,84	24,72	32,98	1,48	5,97	R	<-30°C
20	0,49	9,44	16,54	39,98	34,04	1,57	5,59	C-	-28,9
21	0,32	7,59	19,16	34,09	39,17	1,46	4,68	R	<-30°C
22	0,45	19,00	13,00	31,00	37,00	1,55	5,47	R	<-30°C
23	0,19	4,25	19,49	32,58	43,67	1,02	2,91	E	<-30°C
24	0,33	11,46	22,41	28,29	37,84	1,50	5,12	R	-28,6

Tabelle 9 Ergebnisse der Optimierung der Leitfähigkeit des Systems LiBOB in EC/PC/DMC/EMC

Als Optimum für die Leitfähigkeit bei –25,00 °C wird mit diesem System 1,70 mS/cm erreicht. Die Mischung setzt sich dabei aus 4,38 % EC, 24,1 % PC, 32,5 % DMC, 39,1 % EMC und 0,44 mol/kg LiBOB zusammen. Wie erwartet, zeigen diese Mischungen im Vergleich zur Carbonatmischung mit drei Carbonaten eine deutlich höhere Leitfähigkeit und einen wesentlich größeren Flüssigkeitsbe-

[14] Das Salz war bei Raumtemperatur nicht komplett löslich

reich. Damit sind fast alle hier vermessenen Elektrolyte bis –30°C einsetzbar. Interessanterweise tritt bei dieser Elektrolytmischung eine, wenn auch im Vergleich zu Ethylacetat (siehe Abschnitt 4.2.3.3), geringe Erhöhung der Leitfähigkeit durch Zugabe von EMC auf, obwohl dieses Lösungsmittel mit 0,065 kg/(m*s) [39] eine geringfügig höhere Viskosität als DMC mit 0,0585 kg/(m*s) [38] aufweist.

4.2.3.3 Optimierung der Leitfähigkeit von LiBOB in EC/PC/DMC/EA Mischungen

Um die Viskosität der Elektrolytlösungen noch weiter zu reduzieren und damit die Leitfähigkeit noch weiter zu erhöhen, bietet es sich an, Essigsäureethylester mit einer Viskosität von 0,0439 kg/(m*s) [15] der Carbonatmischung hinzuzufügen. Neben seiner geringen Viskosität zeichnet sich dieses Lösungsmittel vor allem durch seine elektrochemische Stabilität aus, die in Abschnitt 4.4 untersucht wird. Auch aufgrund seines tiefen Schmelzpunktes von –84,0 °C [40] eignete sich dieses Lösungsmittel sehr gut für den Einsatz in Tieftemperaturbatterien.
Die Lösungsmittelmischung für diese Optimierung setzt sich neben Essigsäureethylester noch aus den Carbonaten EC, PC und DMC zusammen.

	m_{LiBOB} / (mol/kg)	ξ_{EC} / %	ξ_{PC} / %	ξ_{DMC} / %	ξ_{EA} / %	$\kappa_{-25°C}$ / (mS/cm)	$\kappa_{+25°C}$ / (mS/cm)	Schritt	θ_{KP} / °C
1	0,70	17,50	17,50	32,50	32,50	2,50	9,20	F	<-30°C
2	0,70	32,50	32,50	32,50	2,50	0,93	6,10	F	<-30°C
3	0,70	32,50	17,50	17,50	32,50	2,05	8,55	F	<-30°C
4	0,50	32,50	17,50	32,50	17,50	1,93	--	F	<-30°C
5	0,50	17,50	32,50	17,50	32,50	1,96	7,22	F	<-30°C
6	0,50	17,50	10,00	17,50	55,00	3,39	9,47	R	<-30°C
7	0,70	10,00	21,30	10,00	58,80	3,53	10,40	R	<-30°C
8	0,80	21,25	0,63	21,25	56,87	3,49	10,65	R	<-30°C
9	0,65	0,63	7,19	23,13	69,05	3,70	8,66	R	<-30°C
10	0,63	7,19	2,04	3,44	87,33	4,31	9,64	R	<-30°C
11	0,89	2,04	5,55	11,41	81,00	4,55	11,21	R	<-30°C
12	0,76	13,11	4,82	16,62	65,45	3,87	10,70	C-	<-30°C
13	0,72	7,87	13,08	11,82	67,23	3,76	10,44	C-	<-30°C
14	0,70	4,09	6,78	16,98	72,15	4,04	9,74	C-	<-30°C
15	0,73	7,24	8,94	11,97	71,85	4,15	10,60	C-	<-30°C
16	0,75	9,13	5,32	13,79	71,76	4,10	10,50	C-	<-30°C
17	0,79	8,71	4,15	3,32	83,82	4,62	11,35	R	<-30°C
18	0,77	3,46	5,02	1,28	90,24	4,83	11,16	R	<-30°C
19	0,75	6,29	6,56	8,42	78,73	4,39	10,78	C-	<-30°C
20	0,98	3,06	8,60	8,78	79,56	4,51	11,78	R	<-30°C
21	0,97	2,34	5,10	3,97	88,59	4,91	12,00	R	<-30°C
22	**1,08**	**0,37**	**4,37**	**1,75**	**93,51**	**5,09**	**12,31**	**E**	**<-30°C**
23	0,79	4,23	0,95	0,1	94,72	4,83	8,07	R	<-30°C
24	0,87	3,12	4,59	6,51	85,78	4,72	11,41	C-	<-30°C
25	0,89	2,04	5,55	11,41	81,00	4,47	10,96	R	<-30°C
26	0,84	5,75	3,94	2,87	87,44	4,78	11,60	C-	<-30°C
27	0,87	3,29	4,08	4,00	88,63	4,81	11,52	C-	<-30°C

Tabelle 10 Ergebnisse der Optimierung der Leitfähigkeit des Systems LiBOB in EC/PC/DMC/EA

Als Startzusammensetzung für die Optimierung wurde wieder mit einer Lösungsmittelmischung begonnen, die aus den gleichen Massenanteilen an Lösungsmitteln besteht. Die Konzentration an LiBOB betrug ebenfalls 0,6 mol/kg. Als Startvariation wurde hier 0,2 mol/kg für das Salz und 15 % für die Lösungsmittel eingestellt. Die Ergebnisse dieser Optimierung sind in Tabelle 10 und dargestellt.

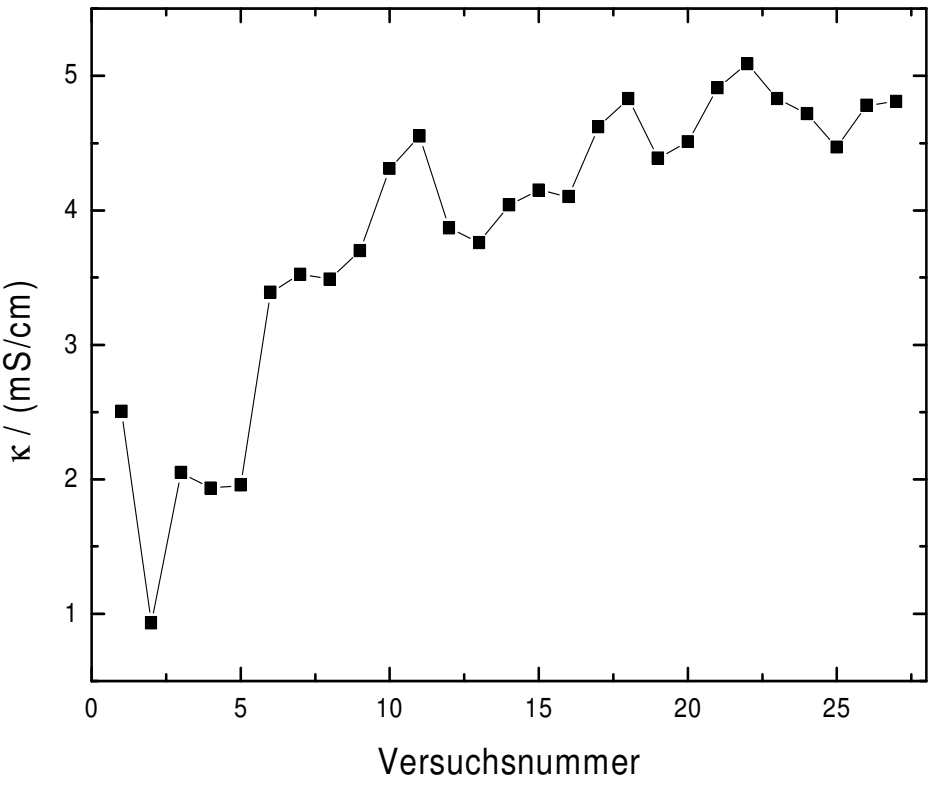


Abbildung 34 Optimierung der Leitfähigkeit von LiBOB in EC/PC/DMC/EA Elektrolytlösungen

Als Leitfähigkeitsoptimum bei –25,00 °C,. $\kappa_{opt,-25\ °C}$, wird für diese Elektrolytlösung ein Wert von 5,09 mS/cm mit einer Elektrolytlösung gefunden, die sich aus 0,37 % EC, 4,37 % PC, 1,75 % DMC und 93,51 % EA und 1,08 mol/kg LiBOB zusammensetzt. Diese Lösung verfügt mit 12,31 mS/cm auch über die höchste gemessene Leitfähigkeit bei +25,00 °C für dieses System. Durch die Zugabe von Essigsäureethylester wird eine enorme Steigerung des Leitfähigkeit erreicht, die durch eine starke Abnahme der Viskosität der Elektrolytlösung aufgrund hoher Gehalte an Essigsäureethylester bedingt ist.

Um sicher zu gehen, dass das gefundene Maximum tatsächlich das globale Maximum ist, wurde noch eine zweite Optimierungsreihe durchgeführt, bei welcher die Startlösung einen hohen PC-Gehalt aufwies (0,6 mol/kg LiBOB, 12,5 % EC, 67 % PC, 12,5 % DMC und 8 % EA). Die Schrittgröße wurde auf 0,2 mol/kg für das Salz und auf 5 % für die Carbonate gesetzt. Die anfängliche Variation für das EA betrug 1 %. In der Tabelle 11 sind die Ergebnisse der einzelnen Optimierungsschritte aufgeführt.

i	m_{LiBOB} / (mol/kg)	ξ_{EC} / %	ξ_{PC} / %	ξ_{DMC} / %	ξ_{EA} / %	$\kappa_{-25°C}$ / (mS/cm)	$\kappa_{+25°C}$ / (mS/cm)	Schritt	θ_{KP} / °C
1	0,70	10,00	64,50	10,00	15,50	1,04	6,23	F	<-30°C
2	0,70	10,00	69,50	15,00	5,50	0,74	5,34	F	<-30°C
3	0,50	15,00	64,50	15,00	5,50	0,90	5,28	F	<-30°C
4	0,50	10,00	69,50	10,00	10,50	1,02	5,29	F	<-30°C
5	0,70	15,00	69,50	10,00	5,50	0,65	5,15	F	<-30°C
6	0,50	7,50	64,50	15,00	13,00	1,15	5,83	R	<-30°C
7	0,40	3,80	62,00	17,50	16,80	1,49	6,13	E	<-30°C
8	0,35	9,38	60,75	11,25	18,62	1,37	5,57	R	<-30°C
9	0,48	1,57	63,88	9,38	25,17	1,73	6,95	R	<-30°C
10	0,46	2,35	55,06	14,07	27,52	1,95	7,25	R	<-30°C
11	0,56	7,13	62,59	11,53	18,75	1,37	6,56	C-	-4,50
12	0,41	6,54	60,94	12,19	20,33	1,51	6,15	C-	<-30°C
13	0,499	5,34	61,66	12,41	20,59	1,53	6,63	C-	<-30°C
14	0,52	4,15	59,27	6,52	30,06	1,86	7,36	R	<-30°C
15	0,57	0,17	59,50	9,00	31,33	1,97	7,78	R	<-30°C
16	0,50	3,70	60,67	11,08	24,55	1,69	6,94	C-	<-30°C
17	0,56	6,29	52,36	10,66	30,69	1,99	7,68	R	<-30°C
18	0,59	4,64	48,89	11,74	34,76	2,24	8,27	E	<-30°C
19	0,57	1,96	51,18	9,58	37,28	2,32	8,34	R	<-30°C
20	**0,61**	**1,08**	**46,43**	**8,84**	**43,65**	**2,73**	**9,20**	**E**	**<-30°C**

Tabelle 11 Optimierung der Leitfähigkeit von LiBOB in EC/PC/DMC/EA Elektrolytlösungen, Start mit PC reicher Lösung

Da auch bei dieser Optimierung eine Zunahme der Leitfähigkeit mit anwachsendem Essigsäureethylestergehalt beobachtet wird und die hergestellten Lösungen zunehmend in dem Bereich liegen, der durch die vorherige Optimierung abgedeckt wird, ist anzunehmen, dass das globale Maximum mit dem Maximum übereinstimmt, das bei der vorangegangenen Optimierung gefunden wurde. Bei beiden Messreihen wurde der Flüssigkeitsbereich des Elektrolyten mit der in Abschnitt 2 gezeigten Methode bis –30°C abgesichert.
Ein Vergleich mit den reinen Carbonatlösungen zeigt, dass hier das Leitfähigkeitsmaximum bei höheren Salzkonzentrationen erreicht wird. Dies ist, wie in Abschnitt 4.2.1.1 beschrieben auf den höheren Anteil an niedrigviskosen Lösungsmitteln zurückzuführen.

4.2.3.4 Optimierung der Leitfähigkeit von LiBOB in EC/PC/DMC/EMC/EA Mischungen

Für die letzte Mischung, die optimiert wurde, wurden alle fünf Lösungsmittel, also EC, PC, DMC, EMC und EA eingesetzt. Zur Festlegung des Startpunktes wurde die Beobachtung hinzugezogen, dass alle bisher gefundenen Leitfähigkeitsmaxima bei Lösungsmittelmischungen gefunden wurden, die einen hohen Anteil an Lösungsmitteln mit niedriger Viskosität aufweisen. Deshalb wurde als Startpunkt eine Zusammensetzung von 46 % EA und 46 % EMC ausgewählt. Der EC Gehalt der Startlösung betrug 3 %, der an PC 4 % und der Gehalt an DMC 1 %. Diese Werte stellen typische

Beispiele für Lösungen mit hohen Leitfähigkeiten dar. Die Schrittweite für das Salz betrug 0,3 mol/kg, die für EC 3 %. Der Anteil an PC wurde mit 4 % variiert, DMC hingegen mit 1 %. Für EMC und EA belief sich die Startvariation auf 30 %. Die Ergebnisse dieser Optimierung sind in Tabelle 12 dargestellt.

	m_{LiBOB} / (mol/kg)	ξ_{EC} / %	ξ_{PC} / %	ξ_{DMC} / %	ξ_{EMC} / %	ξ_{EA} / %	$\kappa_{-25°C}$ / (mS/cm)	$\kappa_{+25°C}$ / (mS/cm)	Schitt	θ_{KP} / °C
1	0,62	1,96	3,02	1,78	55,24	38,00	--[15]	--	F	--
2	0,92	4,96	3,02	1,78	55,24	35,00	--[15]	--	F	--
3	0,62	4,96	3,02	1,78	25,24	65,00	3,27	7,84	F	<-30°C
4	0,92	1,96	3,02	0,78	25,24	69,00	3,66	9,32	F	<-30°C
5	0,57	4,58	6,49	0,71	50,97	37,30	2,45	6,79	F	<-30°C
6	0,92	1,96	7,02	1,78	25,24	64,00	3,61	9,50	F	<-30°C
7	0,56	1,36	6,22	0,98	19,24	72,20	3,37	7,57	R	<-30°C
8	0,84	4,12	7,50	0,66	4,84	82,88	4,52	11,17	R	<-30°C
9	0,70	3,92	6,19	0,99	37,60	51,31	2,95	5,31	C-	<-30°C
10	**0,95**	**0,37**	**8,96**	**0,30**	**19,62**	**70,76**	**5,20**	**12,27**	**R**	**<-30°C**
11	0,77	2,94	6,37	0,94	28,22	61,54	3,46	8,85	C-	<-30°C
12	1,20	3,18	6,93	0,80	22,03	67,07	--[15]	--	R	--
13	0,72	1,81	6,40	0,94	19,94	70,92	3,73	8,99	C-	<-30°C
14	0,97	1,15	6,79	0,84	9,73	81,49	4,47	11,21	R	<-30°C
15	0,90	1,92	6,78	1,24	20,56	69,50	5,10	13,41	C-	<-30°C
16	0,83	1,79	11,55	0,81	4,63	81,21	4,49	11,24	R	<-30°C
17	1,08	1,93	10,24	0,60	3,82	83,42	4,56	--[16]	R	<-30°C
18	0,94	2,90	11,22	0,61	11,66	73,63	4,47	11,10	R	<-30°C
19	0,946	1,59	7,90	0,78	10,22	79,52	4,39	11,09	C-	<-30°C
20	1,052	2,18	5,00	0,62	18,99	73,21	4,03	10,77	R	<-30°C

Tabelle 12 Optimierung der Leitfähigkeit von LiBOB in EC/PC/DMC/EMC/EA Elektrolytlösungen

Das Leitfähigkeitsoptimum für dieses System beträgt 5,20 mS/cm bei –25,00 °C, und stellt damit den höchsten erhaltenen Wert für diese Temperatur dar. Diese Leitfähigkeit wird mit einem Elektrolyten erreicht, der sich aus 0,95 mol/kg LiBOB in einer Lösungsmittelmischung aus 0,37 % EC, 8,96 % PC, 0,30 % DMC, 19,62 % EMC und 70,76 % EA zusammensetzt. Auch bei dieser Mischung liegt, wie erwartet, das Leitfähigkeitsoptimum bei hohen Gehalten an Lösungsmitteln mit niedriger Viskosität. Die Tatsache, dass mit dieser Mischung die höchsten Leitfähigkeiten erzielt werden, verwundert nicht. Hätte EMC einen negativen Einfluss auf die Leitfähigkeit, würde sein Gehalt im Laufe der Optimierung schrittweise reduziert und schließlich ganz aus der Lösungsmittelmischung entfernt. Wie auch bei dieser Optimierungsreihe wurde der Flüssigkeitsbereich mit der in Abschnitt 4.3 beschriebenen Methode bis –30 °C abgesichert.

[15] Das Salz war bei Raumtemperatur nicht komplett löslich.

[16] Die Messzelle wurde während der Messung zerstört.

4.3 Bestimmung des Flüssigkeitsbereichs der Elektrolytlösungen

Um sicherstellen zu können, dass der Flüssigkeitsbereich der hergestellten Elektrolytlösungen bis zu den eingangs geforderten –25 °C gewährleistet ist, ist es nötig, diesen experimentell zu bestimmen. Anhand der in Kapitel 4.2.3 beschriebenen Leitfähigkeitsmessungen ist es nicht möglich, eine Aussage zu treffen, ob der infrage kommende Elektrolyt noch komplett flüssig ist oder ob nicht schon ein Teil der Mischung als Feststoff aus der Lösung ausgefallen ist. Selbst wenn schon ein Teil des Elektrolyten als Feststoff vorliegt, kann der Rest der Lösung noch über eine genügend hohe Leitfähigkeit verfügen, die wie das Beispiel LiBOB in EC/PC/DMC Mischungen zeigt, siehe Abschnitt 4.2.3.1, höher sein kann, als bei Lösungen, bei denen der Elektrolyt noch komplett flüssig ist. Diese Methode eignet sich nur für die Beobachtung, ob ein Elektrolyt komplett gefroren ist oder nicht, da bei einem festen Elektrolyten eine Leitfähigkeit nahe Null beobachtet wird. Daher ist eine genügend große Leitfähigkeit kein Kriterium für den Flüssigkeitsbereich eines Elektrolyten. Auch eine visuelle Kontrolle der Messzellen stellt keine sichere Methode dar, zum einem, weil die Messzellen, nachdem sie für die visuelle Untersuchung aus dem Bad genommen wurden, sich aufgrund ihrer tiefen Temperatur sofort mit einer Eisschicht, die von der Luftfeuchtigkeit verursacht wird, überziehen, zum anderen lässt sich der Elektrolyt aufgrund seiner hohen Viskosität bei diesen Temperaturen nur schwer durch Schütteln in den Kapillarrohren der Messzellen bewegen, so dass auch so keine sichere Aussage darüber getroffen werden kann, ob der Elektrolyt noch flüssig ist. Zur Bestimmung des Flüssigkeitsbereichs der Elektrolytlösungen ist es daher nötig, alle hergestellten Lösungen mit der in Kapitel 2 beschriebenen Phasendiagrammanlage zu vermessen. Diese Anlage erweist sich als ideal für dieses Aufgabe, da durch die Vielzahl der Messzellen dieses Messproblem in kurzer Zeit gelöst werden kann. Würde für diese Aufgabe die von Carl [41] beschriebene Methode benutzt, so würde hierfür mindestens ein Mannjahr benötigt werden. Durch den Einsatz der Phasendiagrammanlage betrug die Messzeit ca. acht Wochen, in denen aufgrund der vollautomatischen Messdatenerfassung Zeit für eine Reihe von anderen Tätigkeiten blieb. Durch die Kopplung der Thermometrie mit der Conduktometrie wird bei diesen Messungen noch nebenbei die Temperaturabhängigkeit der Leitfähigkeit bestimmt. Sollte diese nach den von Carl [41] beschriebenen Methoden durchgeführt werden, so wäre dafür ebenfalls mindestens ein Mannjahr nötig.

4.3.1 Versuchsdurchführung

Für die Durchführung der Bestimmung des Flüssigkeitsbereichs wurde die in Kapitel 2 beschriebene Anlage verwendet, mit der von allen in Kapitel 4.2.3 aufgeführten Lösungen sowohl Abkühl- wie auch Aufheizkurven aufgenommen wurden. Bei diesen Experimenten wurde ein Temperaturbereich von +20 °C bis -35 °C untersucht[17]. Die Abkühl- wie auch die Aufheizexperimente wurden mit einer Temperaturänderungsgeschwindigkeit von 30 K/h und 6 K/h durchgeführt. Da es sich bei den zu untersuchenden Lösungen um leitfähige Elektrolytlösungen handelt, wurde für diese Messungen eine Kopplung von Thermometrie und Conduktometrie eingesetzt. Leitfähigkeiten

[17] Der Temperaturbereich des Kryostaten betrug +20 °C bis –40 °C; aufgrund von Wärmeeintrag in das System wird somit eine Temperatur von –40 °C im Thermostatenbad nicht erreicht, siehe auch Kapitel 2.5.3.

wurden dabei mit einer konstanten Frequenz von 5 kHz gemessen. Da durch die Optimierungsexperimente, siehe Kapitel 4.2.3 die spezifischen Leitfähigkeiten der Lösungen bei –25,00 °C bestimmt wurden, ist eine Kreuzeichung der Abhängigkeit der Leitfähigkeit von der Temperatur möglich. Zur Reduzierung von Unterkühlungseffekten wurde, wie in Kapitel 3.4 beschrieben, zu allen Lösungen Kohlefasern hinzugefügt. Die Messungen wurden nach der in Kapitel 3.3 dargestellten Methode ausgewertet.

4.3.2 Ergebnisse

Die gefundenen Knickpunkte, welche die Tieftemperaturgrenze des Einsatzbereichs der Elektrolytlösung darstellen, sind in Kapitel 4.2.3 in den Tabelle 8 bis Tabelle 12, aufgeführt. Stellvertretend für Messungen werden hier die Ergebnisse für die Lösungen präsentiert, die nach der Optimierung die maximale Leitfähigkeit bei –25,00 °C aufweisen.
In Abbildung 35 sind die spezifischen Leitfähigkeiten in Abhängigkeit von der Temperatur der Lösung aufgetragen.

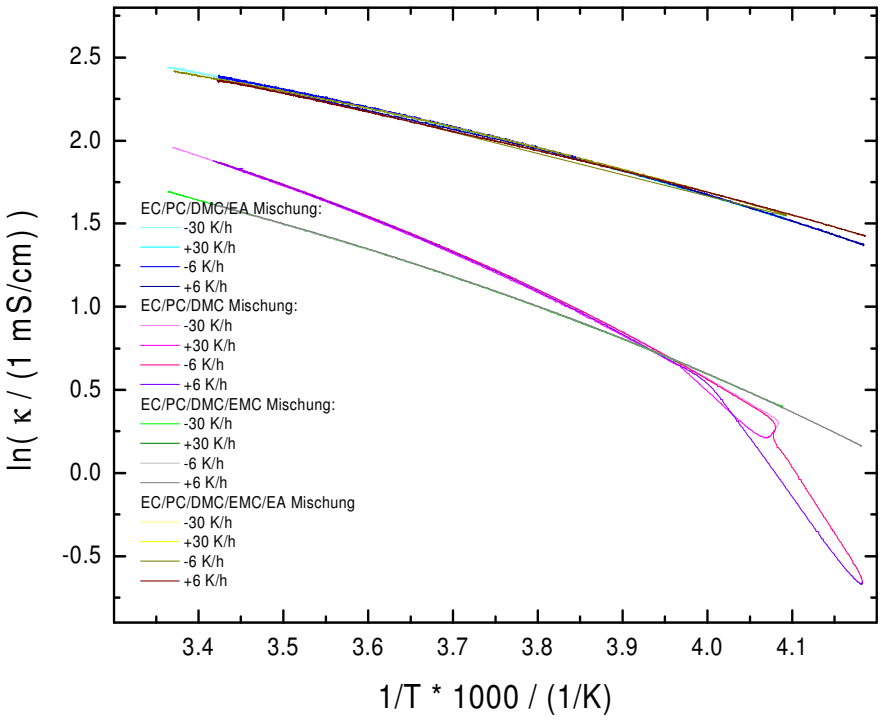


Abbildung 35 Abhängigkeit der Leitfähigkeit von der Temperatur der Elektrolyten mit maximaler Leitfähigkeit(die genauen Zusammensetzungen der Lösungen sind im Text angegeben)

Gerade bei dieser Messung am System EC/PC/DMC (0,47 mol/kg LiBOB, 19,84 % EC, 41,30 % PC und 38,80 % DMC) wird die Gefahr der Unterkühlung der Messlösung deutlich. So wird hier bei den Aufheizexperimenten ein Knickpunkt bei –22,5 °C beobachtet. Wird hingegen die Lösung

abgekühlt, so tritt der Knickpunkt erst bei –28,0 °C auf. Die Unterkühlung beträgt bei diesem System also 5,5 °C, obwohl Kohlefasern als Kristallisationskeime hinzugefügt wurden.
Obwohl die spezifische Leitfähigkeit der Lösung aus EC/PC/DMC/EMC (0,44 mol/kg LiBOB in einer Mischung aus 4,38 % EC, 24,1 % PC, 32,5 % DMC und 39,1 % EMC) bei tiefen Temperaturen höher ist als die der Mischung mit drei Carbonaten, ist die spezifische Leitfähigkeit der Mischung aus vier Carbonaten bei hohen Temperaturen deutlich geringer, was ein Vergleich der in Abbildung 35 dargestellten Messreihen ergibt. Bei dieser wird weder bei den Abkühlkurven, noch bei den Aufheizkurven ein Knickpunkt beobachtet. Dieser Elektrolyt ist daher uneingeschränkt bis -30,00 °C einsetzbar. Ebenfalls keine Knickpunkte werden bei den Abkühl- und Aufheizexperimenten des Elektrolyten mit der Zusammensetzung aus 1,08 mol/kg LiBOB in einer Mischung aus 0,37 % EC, 4,37 % PC, 1,75 % DMC und 93,51 % EA beobachtet, siehe Abbildung 35. Daher ist auch dieser Elektrolyt für den Einsatz bis –30 °C geeignet. Wie dieser Elektrolyt ist auch der Elektrolyt mit einer Zusammensetzung aus 0,95 mol/kg LiBOB in einer Mischung aus 0,37 % EC, 8,96 % PC, 0,30 % DMC, 19,62 % EMC und 70,76 % EA für den gesamten Messbereich geeignet.
Die hier gezeigten Messungen verdeutlichen die Tatsache, dass es nicht möglich ist, nur aufgrund von Leitfähigkeitsmessungen bei einer bestimmten Temperatur auf den Flüssigkeitsbereich eines Elektrolyten zu schließen, da diese, obwohl schon ein Teil der Lösung als Feststoff vorliegt, noch über eine hohe Restleitfähigkeit aufweisen. Mit diesen Messungen konnte nicht nur der Flüssigkeitsbereich des Elektrolyten abgesichert werden, sondern darüber hinaus für alle hergestellten Elektrolytlösungen noch die Abhängigkeit der spezifischen Leitfähigkeit von der Temperatur untersucht werden.

4.4 Untersuchung der elektrochemischen Stabilität

Sollen Lösungsmittel in Batterieelektrolyten eingesetzt werden, so müssen sie über ein hinreichend großes Spannungsfenster verfügen. Auch sollten sie eine ausreichende Stabilität gegenüber den eingesetzten Elektrodenmaterialen aufweisen. Daher wurde vor der Optimierung der Leitfähigkeit zuerst die elektrochemischen Stabilität der eingesetzten Lösungsmittel untersucht. Als Modellelektrolyt wurde eine Lösung von LiBOB in einer reinen Carbonatmischung (EC/PC/DMC) und eine Lösung von LiBOB in ethylacetathaltiger Carbonatmischung (EC/PC/DMC/EA) eingesetzt. Die elektrochemische Stabilität wurde dabei an Elektrodenmaterialien untersucht, die als Modelle für Materialien dienen, die in Lithium-Ionen-Batterien benutzt werden. Für diesen Zweck wurden Platin-, Glassycarbon-[18], Kupfer- und Aluminiumelektroden verwendet. Platin wurde als Elektrodenmaterial ausgewählt, da es sich für die Bestimmung von Spannungsfenstern aufgrund seiner katalytischen Eigenschaften sehr gut eignet. Glassycarbon wurde als Modell für die Kohlenstoffanoden in Lithium-Ionen-Batterien ausgewählt. Kupfer und Aluminium sind typische Materialien, aus denen die Stromableiter in Lithium-Ionen-Batterien gefertigt werden.
Alle Zyklisierungen wurden in Anlehnung an die Methode von Jow [42] durchgeführt. Diese Methode, die inzwischen routinemäßig zur Bestimmung von Spannungsfenstern von Elektrolyten für Doppelschichtkondensatoren eingesetzt wird, entspricht im Prinzip einer normalen Cyclovoltammetrie, bei dem die Zyklengröße der einzelnen Zyklen variiert wird [42]. Die Vorteile dieser Me-

[18] Glaskohlenstoff ist eine Art Kohlenstoff, die durch thermische Zersetzung eines dreidimensional vernetzten Polymers hergestellt wird. Die offene Porosität liegt bei null. Glaskohlenstoff ist sehr beständig gegenüber einer breiten Auswahl an Chemikalien und kann deshalb als Alternative zu den Edelmetallen bei Laboreinrichtungen oder als Elektroden in der Polarographie eingesetzt werden.

thode liegt in der Vermeidung von Verfälschungen der Messergebnisse durch Redoxprodukte an den Elektrodenoberflächen, da die kathodischen und anodischen Äste getrennt untersucht werden und sich somit nicht gegenseitig beeinflussen. Durch die Variation der Zyklengröße können einzelne Prozesse, die in den Cylovoltammogrammen beobachtet werden, miteinander korreliert werden.
In Analogie zu der Methode von Jow wurden ausgehend vom Ruhepotenzial, eine Reihe von Cyclovoltammogrammen mit einen Spannungsvorschub vom 10 mV/s und fünf Zyklen aufgenommen, die sich nur in der Lage des ersten Umkehrpotenzials unterscheiden. Das zweite Umkehrpotenzial war immer das Ruhepotenzial. Für jede Zyklengröße wurde, im Gegensatz zu Jow [42], eine neue Elektrode und eine frische Elektrolytlösung eingesetzt, um eine Verfälschung der Messergebnisse durch Filmbildung auf den Elektroden und durch Kontamination der Elektrolytlösung zu vermeiden. Die weiteren Bedingungen für diese Voltammogramme sind in den Kapiteln 11.1.2 und 11.2.2 aufgeführt. In letzterem Kapitel findet sich auch eine detaillierte Diskussion der Messergebnisse.
In Tabelle 13 sind die wesentlichen Ergebnisse der Messungen an der Elektrolytlösung aus LiBOB in EC/PC/DMC aufgeführt.

Arbeits-elektrode	Richtung	Ruhe-spannung / V	Spannungsgrenze / V		Peaks / V und / mA/cm^2	
			1.Zyklus	5. Zyklus	1.Zyklus	5. Zyklus
Pt	kathodisch	3,10	0,7	0,7	[19]	[19]
Pt	anodisch	3,10	4,5	4,7	--,--	--,--
GC	kathodisch	2,85	0,7	0,6	1,70; -0,043 1,02; 0,07	--,-- 1,06; 0,16
Cu	kathodisch	3,17	1,3	0,7	1,68; -0,22	--,--
Al	anodisch	2,70	5,5	>5,5	--,--	--,--

Tabelle 13 Ergebnisse der Messungen am Modellsystem LiBOB in EC/PC/DMC, Spannungsangaben gegen Li/Li$^+$

In Tabelle 14 sind zum Vergleich die Ergebnisse der Messungen am Testsystem LiBOB in Ethylacetathaltiger Lösung tabelliert.

Arbeits-elektrode	Richtung	Ruhe-spannung / V	Spannungsgrenze / V		Peaks / V und / mA/cm^2	
			1.Zyklus	5. Zyklus	1.Zyklus	5. Zyklus
Pt	kathodisch	3,01	0,8	0,8	[20]	[20]
Pt	anodisch	3,01	4,5	4,7	--,--	--,--
GC	kathodisch	3,33	0,6	0,4	0,95; 0,16	0,88; 0,07
Cu	kathodisch	2,99	1,3	0,7	1,7; -0,14	--,--
Al	anodisch	2,38	3,8	4,5	--,--	--,--

Tabelle 14 Ergebnisse der Messungen am Modellsystem LiBOB in EC/PC/DMC/EA, Spannungsangaben gegen Li/Li$^+$

[19] Siehe Abschnitt 11.2.3.1, Abbildung 65

[20] Siehe Abschnitt 11.2.2.2.1, Abbildung 56

Aus der Bestimmung der elektrochemischen Stabilität ergibt sich, dass beide Modellsysteme ausreichend stabil für den Einsatz in Lithium-Ionen-Batterien sind. An allen Elektroden wird mit zunehmender Zyklenzahl eine steigende Passivierung beobachtet. Durch dieses günstige Verhalten, welches auf die Bildung von Deckschichten zurückzuführen ist, eignen sich diese Elektrolyte besonders für den Einsatz in Batterien. Der Vergleich zwischen den Elektrolyten mit Ethylacetatzusatz und dem ohne, zeigt, dass durch diesen Zusatz eine geringe Erhöhung der Grenzströme und Verkleinerung des Spannungsfensters beobachtet wird.

4.5 Zusammenfassung der Ergebnisse

Die Optimierung der Leitfähigkeit von Elektrolytlösungen ist eine aufwändige Aufgabe, da sowohl die Zusammensetzung der Lösungsmittelmischung, die Konzentration des Salzes und die Temperatur variiert werden kann. Die hohe Anzahl an Experimenten, die nötig wären, verbietet es meist, dieses Problem umfassend durchzurastern. Durch die Anwendung der Simplexmethode auf Literaturdaten konnte gezeigt werden, dass sie sich für dieses Optimierungsproblem eignet.
Mithilfe der Simplexmethode wurden die Leitfähigkeiten von LiBOB in Mischungen aus EC/PC/DMC, EC/PC/DMC/EMC, EC/PC/DMC/EA und EC/PC/DMC/EMC/EA optimiert. Hierfür wurden zwischen 20 und 30 Schritte benötigt, wohingegen für eine umfassende Untersuchung über 3000 Messungen nötig gewesen wären.
Mit den reinen Carbonatmischungen wurde eine deutliche Steigerung gegenüber den Literaturwerten erreicht. Im Vergleich zu $LiPF_6$ in PC/EC (3 mS/cm bei –25 °C [35] [21]) wird durch Einsatz von Essigsäureethylester fast die doppelte spezifische Leitfähigkeit (5,2 mS/cm) erzielt. Vor kurzem wurde von Jow et al. [43] eine Elektrolytmischung vorgestellt, die LiBOB als Leitsalz enthält, bei der auch Ester eingesetzt werden. Die maximale Leitfähigkeit, die mit diesen Mischungen erreicht werden, beträgt 3,8 mS/cm und ist damit den in dieser Arbeit präsentierten Lösungen deutlich unterlegen.
Die Abhängigkeit der Leitfähigkeit von der Temperatur aller hergestellten Lösungen wurde mittels der Phasendiagrammessapparatur untersucht. Durch diese Untersuchungen wurde zusätzlich der Flüssigkeitsbereich der Elektrolyte abgesichert. Die elektrochemische Stabilität der Elektrolytlösungen wurde an Modellssystemen getestet. Mit den optimierten Elektrolyten werden alle in Abschnitt 4.1 geforderten Eigenschaften erfüllt. Daher ist es möglich, mit diesen Elektrolyten Batterien zu produzieren, die sich für den Einsatzbereich bis -25 °C und darunter eignen.
Das Bisoxalatoborat weist, als typischer Vertreter der Chelatoborate, eine hohe Delokalisierung seiner Ladung auf. Die Assoziation dieses Salzes ist daher wenig ausgeprägt. Daher ist für das Erreichen einer hohen Leitfähigkeit keine sonderlich hohe Dielektrizitätszahl des Lösungsmittels notwendig. Wie gezeigt werden konnte, werden hohe Leitfähigkeiten vor allem durch Reduktion der Viskosität der Lösungsmittelmischung und somit durch die Reduktion der Stokeschen Reibung erreicht. Wie die Untersuchungen der elektrochemischen Stabilität gezeigt haben, weist Ethylacetat als typischer Vertreter der Verbindungsklasse der Ester, eine genügend hohe Stabilität für den Einsatz in Lithium-Ionen-Batterien auf. Durch den erstmaligen Einsatz von Essigsäureethylester in Lithium-Ionen-Batterien mit LiBOB als Leitsalz, konnte eine enorme Steigerung der Leitfähigkeiten bei tiefen Temperaturen erzielt werden. Durch den sehr niedrigen Schmelzpunkt des Essigsäureethylesters wird zudem noch der Temperaturbereich, bei dem Lithium-Ionen-Batterien eingesetzt

[21] Elektrolyt mit der größten Leitfähigkeit bei –25 °C der in dieser Arbeit untersuchten Systeme

werden können, enorm in Richtung tiefe Temperaturen hin erweitert. Da sich Ester für den Einsatz in Lithium-Ionen-Batterien bewiesen haben, liegt es nahe, andere Ester auf ihre Eignung hin zu untersuchen. Durch Variation der Säure- beziehungsweise Alkoholreste dieser Verbindungen können sowohl Ester wie Propionsäurepropylester, Buttersäureethylester, Essigsäuremethylester oder Ameisensäureethylester ausgewählt werden. Aufgrund des höheren Siedepunktes eignen sich die längerkettigen Ester besser für den Einsatz bei hohen Temperaturen, wohingegen die kurzkettigen Ester für den Tieftemperatureinsatz prädestiniert sind. Durch die Wahl der geeigneten Ester und der Carbonate ist es somit möglich einen optimalen, für den gewünschten Einsatzbereich angepassten, Elektrolyten maßzuschneidern. Die Simplexoptimierung ist dabei ein sehr leistungsfähiges Hilfsmittel, mit dem diese Aufgabe effizient gelöst werden kann. Der einzige Nachteil von Estern beim Einsatz in Lithium-Ionen-Batterien besteht darin, dass sie im Gegensatz zu Carbonaten, wie Ethylencarbonat nur eine wenig ausgeprägte und geeignete „solid elctrolyte Interface“ (SEI) (siehe Kapitel 6.2.1) ausbilden. Da aber wie in Kapitel 6.2.2 beschrieben, die Ausbildung der SEI sehr gut durch Additive beeinflusst werden kann, ist es daher nicht mehr nötig, dass diese Schicht von Lösungsmitteln ausgebildet werden muss. Vielmehr kann durch Auswahl geeigneter Additive die SEI den jeweiligen Anforderungen angepasst werden, so dass eine größere Freiheit bei der Auswahl der Lösungsmittelkomponenten besteht.

4.6 Literaturverzeichnis

[1] D. Linden, in D. Linden, Hrsg., *Handbook of Batteries*, Belfast (1994)

[2] J. Barthel und H. J. Gores in *Chemistry of Nonaqueous Electrolyte Solutions. Current Progress,* Kapitel 1 (Hrsg.: G. Mamantov, A. I. Popov), VCH, New York (1994)

[3] K. Izutsu, *Electrochemistry in Nonaqueous Solutions,* Wiley-VCH, Weinheim (2002)

[4] J. Barthel, H. J. Gores in *Handbook of Battery Materials,* Kapitel 7 (Hrsg.: J. O. Besenhard), Wiley-VCH, Weinheim (1998)

[5] J. Barthel, H. Krienke und W. Kunz, *The Physical Chemistry of Electrolyte Solutions Solutions* - Modern Aspects, Steinkopff - Springer, Darmstadt, Berlin, (1998)

[6] G. Wedler, *Lehrbuch der Physikalischen Chemie*, Wiley-VCH, Weinheim (1997)

[7] J. Molenat, *J. Chim. Phys. Phys.-Chim. Biol.*, **66**, 825 (1969)

[8] J. F. Casteel und E. A. Amis, *J. Chem. Eng. Data,* **17**, 55 (1972)

[9] H. J. Gores, und J. Barthel, *J. Solution Chem.*, **9**, 939 (1980)

[10] M.S-Ding, K. Xu, S. S. Zang, K. Amine, G. L. Henriksen und T. R. Jow, *J. Electrochem. Soc.,* **148**, A1196 (2001)

[11] J. Barthel, H. J. Gores, und G. Schmeer, *Ber. Bunsenges. Phys. Chem.,* **83**, 911 (1979)

[12] S. S. N. Murthy, Gangasharan und S. K. Nayak, *J. Chem. Soc, Faraday Trans*, **89**, 509 (1993)

[13] T. Okamoto, T. Ohashi, K. Kakuda und I. Koga, *Asymmetric ketones from aldehyde-ester mixtures by catalytic process*, JP 49061111, (1974)

[14] I. J. Krchma und J. W. Williams, *J. Am. Chem. Soc.*, **49**, 2408 (1927)

[15] L. S. Manjeshwar und T. M. Aminabhavi, *J. Chem. Eng. Data*, **33**, 184 (1988)

[16] G. Jancso, H. Illy und D. Staschewski, *J. Chem. Soc, Faraday Trans 1: Phys. Chem. Cond. Phases*, **72**, 2203 (1976)

[17] R. H. Davies, A. Finch und P. J. Gardner, *J. Chem. Thermodynamics*, **12**, 291 (1980) .

[18] F. Corradini, A. Marchetti, M. Tagliazucchi, L. Tassi und G. Tosi, *Australian Journal of Chemistry*, **48**, 1193 (1995)

[19] M. Contreras S., *J. Chem. Eng. Data*, **46**, 1149 (2001)
[20] M. Ue, *J. Electrochem. Soc.*, **143**, L271, (1996)
[21] J. Barthel, H. J. Gores, R. Neueder, und A. Schmid, *Pure Appl. Chem.*, **71**, 1705 (1999).
[22] H. J. Gores, und J. Barthel, *J. Solution Chem.* **9**, 939 (1980)
[23] J.-i. Yamaki in *Advances in Lithium Batteries,* W. A. van Schalkwijk, und B. Scrosati, Hrsg., p. 155, Kluwer Academic/Plenum, New York (2002)
[24] G. E. Blomgren in *Nonaqueous Electrochemistry,* D. Aurbach, und M. Dekker, Hrsg., p. 53, New York (1999)
[25] M. Salomon, H.-P. Lin, E. J. Plichta, und M. Hendickson in *Advances in Lithium Batteries,* W. A. van Schalkwijk, und B. Scrosati, Hrsg., p. 309, Kluwer Academic/Plenum, New York (2002)
[26] E. W. Weisstein, *CRC Concise Encyclopedia of Mathematics*, CRC Press Boca Raton (1999)
[27]W. Spedley, G. R. Hext, und F. R. Himsworth, *Technometrics,* **4**, 441 (1962)
[28] J. A. Nelder und R. Mead, *Computer Journal,* **7**, 308 (1965)
[29] R. A. Aberg, A. G. T. und Gustavson, *Analytica Chimica Acta,* **144**, 39 (1982)
[30] *Users Guide to Multisimplex* ,Version 2.1, Grabitech Solutions AB Sundsvall, Sweden
[31] L. A. Zadeh, *Information and Control*, **8**, 338 (1965)
[32] W. Krabs, *Einführung in die Nichtlineare Optimierung für Ingenieure*, B.G.Teubner, Stuttgart, (1983)
[33] C. Richter, *Optimierungsverfahren und BASIC Programe*, Akademie Verlag, Berlin (1988)
[34] 30. S. S. Rao, *Optimization*, Wiley Eastern Ltd., New Dehli, (1978)
[35] M. S. Ding und T. R. Jow, *J. Electrochem. Soc.* **150**, A620 (2003)
[36] J. Barthel, R. Wachter und H. J. Gores in *Modern Aspects of Electrochemistry*, Vol. 13, Chap. 1, p.1 Hrsg. J. O'.-M. Bockris, Plenum, New York (1979)
[37] J. Barthel, R. Wachter, *Ber. Bunsenges. Phys. Chem.*, **83**, 634 (1979)
[38] A. Rodriguez, J. Canosa, A. Dominguez und J. Tojo, *J. Chem. Eng. Data*, **48**, 146 (2003)
[39] M. Ue, K. Ida, S. Mori, *J. Electrochem. Soc.,* **141,** 2989 (1994)
[40] Y. Marcus, *Ion Solvation*, Wiley Chichester, (1985)
[41] E. S. Carl, *Neue Elektrolyte in organischen Carbonatlösungen zur Anwendung in sekundären Lithium-Ionen Batterien*, *Dissertation*, Regensburg (1998).
[42] K. Xu, S. P. Ding und T. R. Jow, *J. Electrochem. Soc.*, **146**, 4172 (1999)
[43] T. R. Jow, K. Xu, M. S. Ding, S. S. Zhang, J. L. Allen, und K. Amine, *J. Electrochem. Soc.,* **151**, A1702 (2004)

5 Entwicklung und Erprobung einer neuen Anlage zur galvanostatischen Zyklisierung

5.1 Zielsetzung und Anforderungen

Für die in 6 beschriebenen galvanostatischen Zyklisierexperimente wird eine Anlage benötigt, mit der es möglich ist, Batterien mit konstantem Strom zu zyklisieren. Bei diesen Experimenten wird die Batterie mit einem konstanten Strom bis zu einer vorgegebenen Endspannung oder bis zu einem vorgegebenen Bruchteil der maximalen Kapazität aufgeladen. Ist das zu definierende Ladeziel erreicht, wird die Batterie wieder mit einem konstanten Entladestrom bis zu einer vorgegebenen Entladespannung oder Restkapazität entladen. Die Lade- und Entladevorgänge werden einige hundert Male wiederholt.
Für die Durchführung dieser Experimente steht prinzipiell ein Batterietestsystem der Firma BaSyTek (Öllingen) zur Verfügung. Da dieses Gerät aber für große Ströme bis zu 10 A ausgelegt ist, eignet es sich nicht für Messungen im Bereich von einigen Milliampere wie in Kapitel 6 gefordert, da das Eigenrauschen des Geräts in dieser Größenordnung liegt. Galvanostatische Zyklisierungen lassen sich auch mit den elektrochemischen Messplätzen vom Typ Autolab der Firma Metrohm (Filderstadt) und vom Typ IM6 der Firma Zahner (Kronach) durchführen. Da aber diese Geräte für andere Messungen benötigt werden, und mit ihnen nur jeweils eine Batterie gleichzeitig getestet werden kann, war es sinnvoll, einen neuen Messplatz für Zyklisierungen von Lithium-Ionen-Zellen aufzubauen.

Dieses Gerät sollte dabei folgende Grundanforderungen erfüllen:

- Messmodus: galvanostatische Zyklisierung,
- mehrere Messkanäle,
- Strombereich: ±100 μA bis ±400 mA,
- Spannungsbereich: ±1 mV bis ±6 V,
- Ansteuerung via Computer,
- Messgenauigkeit ±1%,
- Unterbrechung der Messung jeweils nach einem Messzyklus möglich,
- Erweiterungsmöglichkeit für Druck-/Temperaturmessung.

Als Alternative zu einem mehrkanaligen Gerät bieten sich auch mehrere Einzelgeräte an, die gemeinsam über einen Steuerrechner angesteuert werden. Dadurch ergibt sich der Vorteil, dass diese Geräte an verschiedenen Orten gleichzeitig eingesetzt werden können. Es wurden daher statt einem Gerät mit fünf Messkanälen, fünf identische Messgeräte gebaut.
Vor der eigentlichen Zyklisierung werden Batterien meist einem Formierungszyklus unterworfen, bei dem ihnen nur ein kleiner Strom aufgeprägt wird. Das Testsystem muss daher zwischen den verschiedenen Zyklenarten unterscheiden können. Bei den in Kapitel 6 beschriebenen Messungen wird im Laufe des Zyklisierungsexperiments die Zyklisierung angehalten und ein Impedanzspektrum aufgenommen. Das Messgerät soll daher in der Lage sein, die Zyklisierung am Ende eines Messzyklus anzuhalten und die Messzelle vom Messgerät galvanisch abzutrennen. Die Messdatenerfassung und -speicherung sollte bei diesem Gerät so erfolgen, dass die Auswertung der Daten

ohne Probleme mit einer Standardsoftware, wie beispielsweise Microcal Origin oder Microsoft Excel erfolgen kann. Um einen Einsatz der Geräte auch in Zukunft gewährleisten zu können, ist es von Vorteil, wenn die Steuersoftware auf einem modernen Betriebssystem wie Microsoft Windows und nicht auf einer veralteten Plattform wie MS-DOS läuft. Ein weiterer wichtiger Punkt für die Zukunftssicherheit des Messgeräts liegt in der Verwendung genormter Schnittstellen für den Anschluss des Messgeräts an den Steuerrechner. Sollten diese Schnittstellen bei zukünftigen Rechnergenerationen nicht mehr verfügbar sein, so können sie aufgrund ihrer Standardisierung leicht nachgerüstet werden.
Neben der Anwendung in der Testung von Batterien stellen galvanostatische Experimente eine in der Elektrochemie universell einsetzbare Methode dar. Sie werden häufig in der Galvanotechnik oder in der Analytik, beispielsweise in der coulometrischen Titration eingesetzt. Es ist daher sinnvoll, das Gerät so zu gestalten, dass es leicht erweitert und für verschiedene Anwendungszwecke angepasst werden kann, wodurch dieses Messsystem über die Batterietestung hinaus universelle Anwendung finden kann. Optimal für eine solche Erweiterungsmöglichkeit ist eine reine Softwarelösung, da damit keine weiteren Eingriffe in das Gerät nötig sind.
Aufgrund der oben genannten Gründe wurde vom Autor dieser Arbeit ein Batterietestsystem mit einem Messkanal entwickelt, das alle oben genannten Anforderungen erfüllt. Um eine ausreichende Anzahl an Messzellen zur Verfügung zu haben, wurde eine Kleinserie von fünf Stück dieser Geräte vom Autor aufgebaut.

5.2 Beschreibung der Schaltung

Das Herz des in Abbildung 36 dargestellten Batterietestsystems ist die Galvanostatenschaltung, an die das zu testende elektrochemische System, zum Beispiel eine Batterie, angeschlossen wird. Durch diese Schaltung ist es möglich, dem zu testenden System einen konstanten Strom aufzuprägen, selbst wenn sich sein Innenwiderstand deutlich ändert.

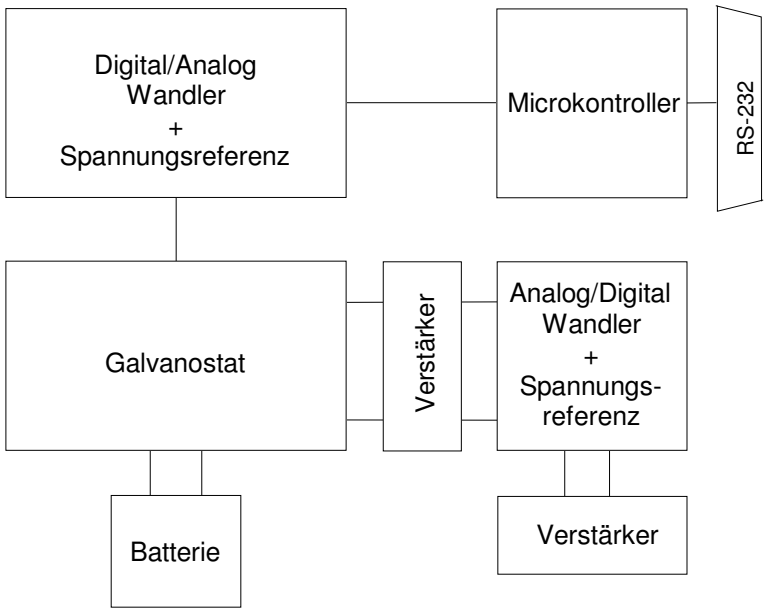


Abbildung 36 Blockschaltbild des Batterietestsystems

Mittels eines Digital/Analog-Wandlers wird eine Steuerspannung erzeugt, mit welcher der Ausgangsstrom des Galvanostaten eingestellt wird. Der Digital/Analog-Wandler wiederum wird über einen Mikrokontroller angesteuert. Mit diesem Mikrokontroller wird ebenfalls der Analog/Digital-Wandler angesprochen. Über eine digital einstellbare Verstärkerstufe ist es mit diesem Wandler möglich, den Strom, der durch die zu testende Batterie fließt, zu messen. Die Spannung der Batterie wird ebenfalls mit diesem Wandler bestimmt, wobei hier die zu messende Spannung wieder über eine digital einstellbare Verstärkerstufe an den Eingangsbereich des Analog/Digital-Wandlers angepasst wird.
Um das Messsystem für zukünftige Einsatzbereiche erweitern zu können, sind an zwei weitere Eingangskanäle des Analog/Digital-Wandlers noch zwei Verstärkerstufen angeschlossen, die es ermöglichen, unabhängig voneinander Spannungen zu messen. Diese Messkanäle können beispielsweise dazu verwendet werden, um Temperatur- oder Drucksensoren anzuschließen, mit denen die zu untersuchenden Batterien zusätzlich überwacht werden können. Die erfassten Messdaten werden vom Mikrokontroller aus über ein RS-232 Interface als ASCII Daten an den Steuerrechner übermittelt. Über dieses Interface erfolgt auch die Datenübertragung zum Steuerrechner. Das Messgerät wird über ein aufwändig stabilisiertes und gesiebtes Netzteil mit Strom versorgt.

5.2.1 Galvanostat

Der Galvanostat, der den konstanten Strom für die Batteriezyklisierung erzeugt, ist das Herzstück des Batterietestsystems. Für die Erzeugung von konstantem Strom gibt es eine Reihe von schaltungstechnischen Varianten. Die einfachste Möglichkeit besteht darin, einen Widerstand vor die elektrochemische Zelle zu schalten, dessen Widerstand um mehrere Größenordungen höher ist, als derjenige der Zelle [1]. Da dann der Stromfluss praktisch nur noch von dem Vorwiderstand abhängt, ist der Stromfluss durch die Messzelle konstant, aufgrund der Tatsache, dass die Widerstandsänderungen innerhalb der Zelle im Vergleich zum Vorwiderstand meist vernachlässigbar sind [1]. Durch Filmbildung auf den Elektroden können aber sehr hohe Widerstände in den Zellen entstehen, selbst wenn diese im ungestörten Zustand nur extrem kleine Widerstände von einigen Ohm aufweisen. Liegt dieser Sachverhalt vor, ist diese Schaltung nicht geeignet. Außerdem ergibt sich das folgende Problem: sollen Ströme, die größer als einige nA sind, durch diese Schaltung fließen, so muss diese Schaltung mit einer Hochspannung versorgt werden. Darüber hinaus wird bei einem Stromfluss durch die Messzelle im Bereich von einigen Hundert mA die Verlustleistung des Vorwiderstandes zum Problem.
Eine Alternative zur Stromkonstanthaltung besteht in der Verwendung von Konstantspannungsreglern, bei denen die Ausgangsspannung über einen Widerstand zum Eingang zurückgekoppelt wird [2],[3]. Da, wie Versuche gezeigt haben, die Langzeitstabilität dieser Schaltungen zu gering ist, ist es notwendig, auf Operationsverstärkerschaltungen zurückzugreifen. In der Literatur [4] werden eine Reihe von Beispielschaltungen für diese Aufgabe aufgeführt.
Die einfachste Möglichkeit für eine solche Schaltung zeigt Abbildung 37:

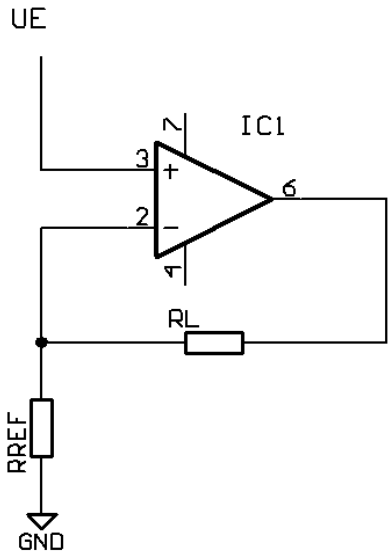


Abbildung 37 Einfachste Galvanostatenschaltung mit Operationsverstärker, aus [4]

Aufgrund der negativen Rückkopplung durch den Referenzwiderstand R_{REF} folgt, dass am invertierenden Eingang des Operationsverstärkers die gleiche Spannung wie an seinem nichtinvertierenden Eingang anliegt. Der Stromfluss I durch R_{REF} lässt sich mit Hilfe des Ohmschen Gesetzes (23) berechnen:

$$I = \frac{U_E}{R_{REF}} \tag{23}$$

Da die Eingangswiderstände des Operationsverstärkers im Vergleich zu den Widerständen R_L und R_{REF} unendlich groß sind, folgt daraus, dass auch durch den Lastwiderstand R_L der Strom I fließt. Da dieser Strom nur von R_{REF} und U_E abhängt, ist der Stromfluss konstant, wenn diese beiden Parameter konstant gehalten werden.
Der Hauptnachteil dieser Schaltung besteht darin, dass der Konstantstrom durch eine Last fließt, bei der kein Anschluss auf Masse liegt, wodurch bei vielen anderen Anwendungsbereichen eine Reihe von Komplikationen verursacht werden [4]. Da dies bei diesem Gerät keine Rolle spielt, ist diese Schaltung anderen überlegen, bei denen ein Anschluss auf Masse liegt, denn bei der hier gezeigten Schaltung ist die Anzahl der verwendeten Bauteile minimal, und daher ist auch die Anzahl der Fehlerquellen erheblich reduziert. Die Genauigkeit des Konstantstroms hängt hier nur von der Präzision und Stabilität des Widerstandes R_{REF} und der Spannungsquelle U_E ab.
In der Abbildung 38 ist die Galvanostatenschaltung des Batterietestsystems gezeigt:

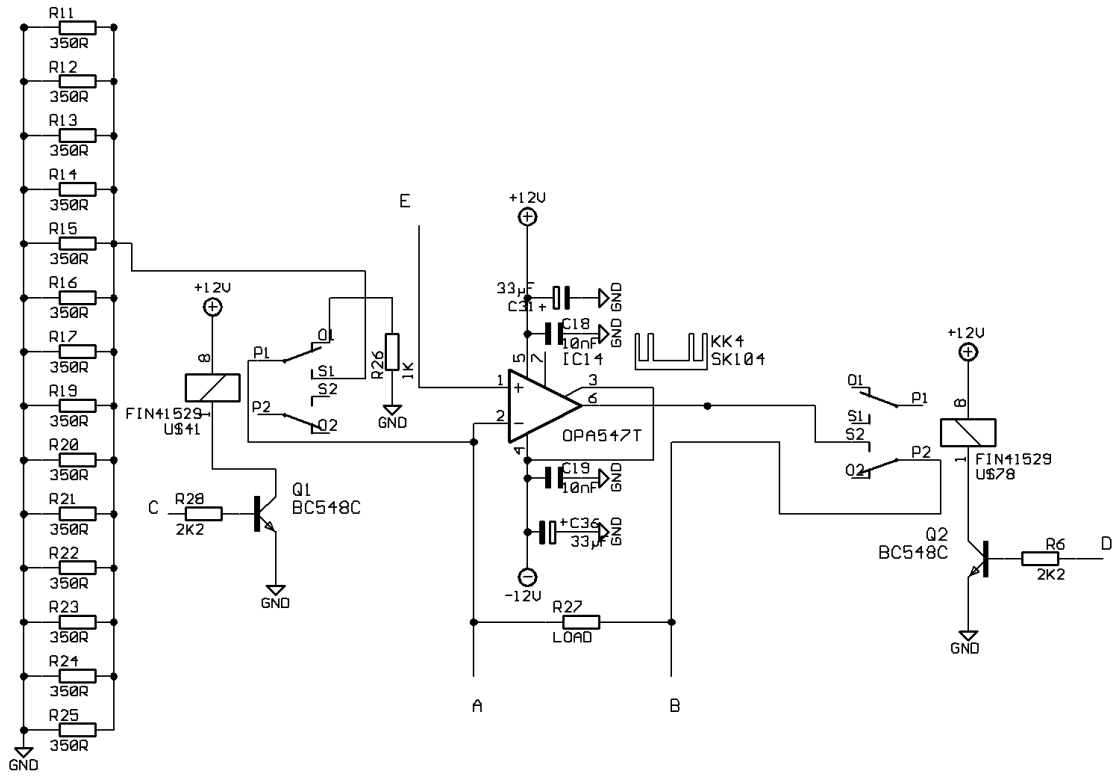


Abbildung 38 Galvanostat des Batterietestsystems

Da dieses Gerät Ströme bis zu 400 mA liefern soll, wurde ein Hochleistungsoperationsverstärker vom Typ OPA547T, IC14 in Abbildung 38, der Firma Texas Instruments eingesetzt. Geeignete Kühlung vorausgesetzt, kann dieses Bauteil Ausgangsströme von bis zu 500 mA liefern [5], so dass auf eine nachgeschaltete Leistungsverstärkerstufe verzichtet werden kann.

Aufgrund des weiten Strombereichs von 100 µA bis 400 mA ist es notwendig, eine Strombereichsumschaltung vorzusehen. Für Ströme bis zu 4 mA wird als Referenzwiderstand ein 1 kΩ Präzisionswiderstand R_{26} vom Typ 8E16D Econistor der Firma Rohpoint eingesetzt. Für den Strombereich bis 400 mA beträgt dieser Widerstand 25 Ω. Da der Konstantstrom auch durch diesen Widerstand fließt, ist die Verlustleistung bei diesen Strömen nicht mehr vernachlässigbar. Daher wurde dieser Widerstand aus 14 Präzisionswiderständen (R_{11} bis R_{25}) mit je 350 Ω aufgebaut, die parallel geschaltet sind. Mit dem Relais FIN41529, das über den Transistor Q1 vom Mikrokontroller angesteuert wird, kann zwischen den beiden Messbereichen umgeschaltet werden.

Über ein zweites Relais, U$78, kann die elektrochemische Zelle, die in Abbildung 38 mit R_{LOAD} bezeichnet wird, mit dem Galvanostaten verbunden werden.

Die Steuerspannung, die der im Abschnitt 5.2.2 beschriebene Digital/Analog-Wandler erzeugt, wird an Punkt E in die Schaltung eingespeist.

Die Bestimmung des Stroms, der durch die Messzelle fließt, erfolgt durch Messung der Spannung an Punkt A in Abbildung 38. Da diese Spannung dem Spannungsabfall am Referenzwiderstand R_{REF} entspricht, kann mit dem Ohmschen Gesetz (23) der Strom berechnet werden. Die Klemmenspannung der elektrochemischen Zelle wird durch Differenzbildung der Spannungen an Punkt A und Punkt B bestimmt. Die Spannungsmessung an beiden Punkten erfolgt durch den in Abschnitt 5.2.3 beschriebenen Analog/Digital-Wandler.

5.2.2 Digital/Analog-Wandler

Die Steuerspannung des Galvanostaten wird mit einem 16-Bit Digital/Analog-Wandler (DAC) vom Typ MAX542 erzeugt, dem eine Verstärkerstufe nachgeschaltet ist, um die nötige hohe Steuerspannung zu erzeugen. In Abbildung 39 ist die Schaltung des Digital/Analog-Wandler und der Verstärkerstufe dargestellt:

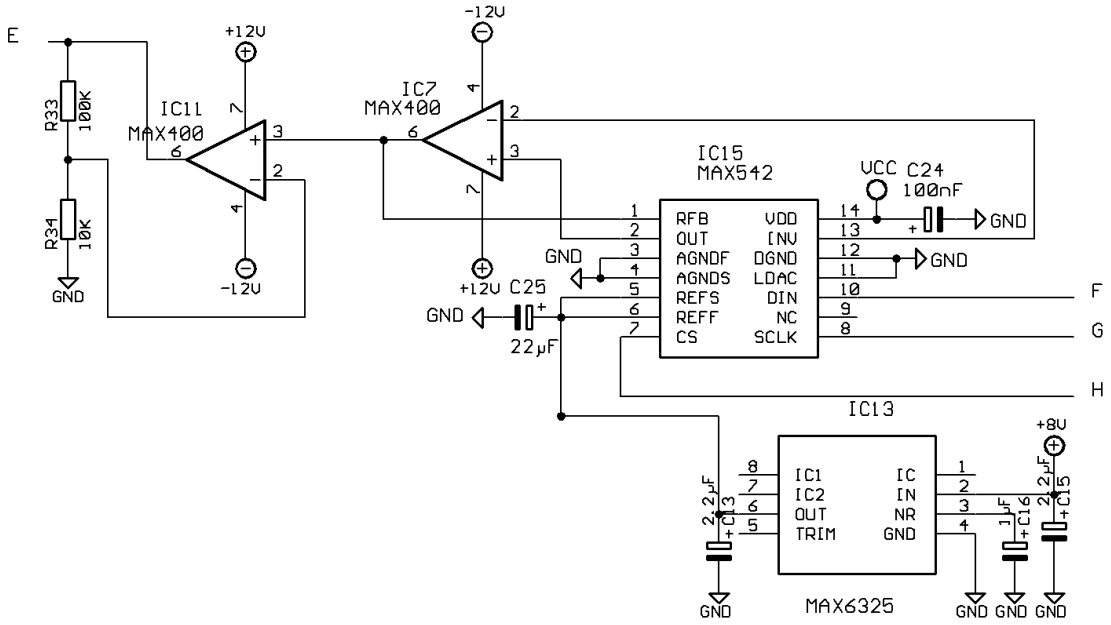


Abbildung 39 Digital/Analog-Wandler mit Verstärkerstufe

Der DAC wird über Pin 14 mit 5V versorgt. Die Kondensatoren C24 und C25 dienen dazu, Störeinflüsse zu minimieren, die durch hohe Taktraten im Digitalteil der Schaltung hervorgerufen werden [6]. Das SPI [22] kompatible Digitalinterface, über das der Wandler an den Mikrokontroller angeschlossen ist, wird über die Leitungen CS (H), SCLK (G), und DIN (F) hergestellt. Das Schnittstellenprotokoll ist im Anhang (11.7.3.4) und in dem Datenblatt [6] beschrieben.
Der Galvanostat wird sowohl als Stromquelle als auch als Stromsenke eingesetzt. Daher ist für diesen Anwendungsbereich eine bipolare Steuerspannung nötig.
Da der D/A-Wandler nur eine Spannung von 0V bis zur Referenzspannung ausgeben kann, wird der Operationsverstärker IC7 eingesetzt. Dieses Bauteil erzeugt aus den 0-2,5V, die der DAC ausgibt, eine bipolare Spannung in einem Bereich von –2,5 bis +2,5V. Die Spannung, die am Ausgang des Operationsverstärkers (PIN6) anliegt, lässt sich nach folgender Formel berechnen:

$$U_{out} = -2,5 + 2 * U_{in} \quad (24)$$

Damit der D/A-Wandler überhaupt eine Spannung generieren kann, benötigt er eine Spannungsreferenz, die an Pin 5 angeschlossen wird. Hierfür wird eine Spannungsreferenz vom Typ MAX6325 eingesetzt, die eine Spannung von +2,5V bereitstellt, aus der alle vom DAC erzeugten Spannungen abgeleitet werden. Für eine detaillierte Beschreibung dieses Bauteils sei hier

[22] Bei der Betriebsart Motorola SPI werden die Sende- und Empfangsdaten auf getrennten Datenleitungen übertragen. Für das Taktsignal und die Masse gibt es je eine separate Leitung.

abgeleitet werden. Für eine detaillierte Beschreibung dieses Bauteils sei hier wieder auf das Datenblatt [7] verwiesen. Die Kondensatoren C13, C14 und C16 dienen zur Stabilisierung der Versorgungsspannung [7].
Über einen nichtinvertierenden Verstärker wird die Ausgangsspannung der DAC-Schaltung um den Faktor 11 verstärkt. Für diesen Verstärker wird wieder aufgrund seiner günstigen Eigenschaften ein Operationsverstärker vom Typ MAX400 eingesetzt. Der Verstärkungsfaktor dieser Schaltung wird durch die Widerstände R_{33} und R_{34} bestimmt. Um die Genauigkeit des Digital/Analog-Wandlers nicht herabzusetzen, wurden für diese Widerstände Präzisionswiderstände vom Typ 8E16D Econistor der Firma Rohpoint eingesetzt, siehe auch Kapitel 2.7.2.2. Die Ausgangsspannung dieses Verstärkers wird dann direkt zur Steuerung des Galvanostaten verwendet.

5.2.3 Analog/Digital-Wandler

Die Spannungen UA und UB, die an der Batterie anliegen, werden über je einen Impedanzwandler, einem Spannungsteiler zugeführt. Durch den hohen Eingangswiderstand des Impedanzwandlers (hier 200 GΩ [10]) wird vermieden, dass hier ein Spannungsabfall im nachgeschalteten Spannungsteiler mit geringem Innenwiderstand auftritt, wodurch das Messergebnis verfälscht würde. Für den Impedanzwandler werden wieder aufgrund ihres sehr geringen Offsetfehlers Operationsverstärker vom Typ MAX400 eingesetzt (IC12 und IC17 in Abbildung 40).

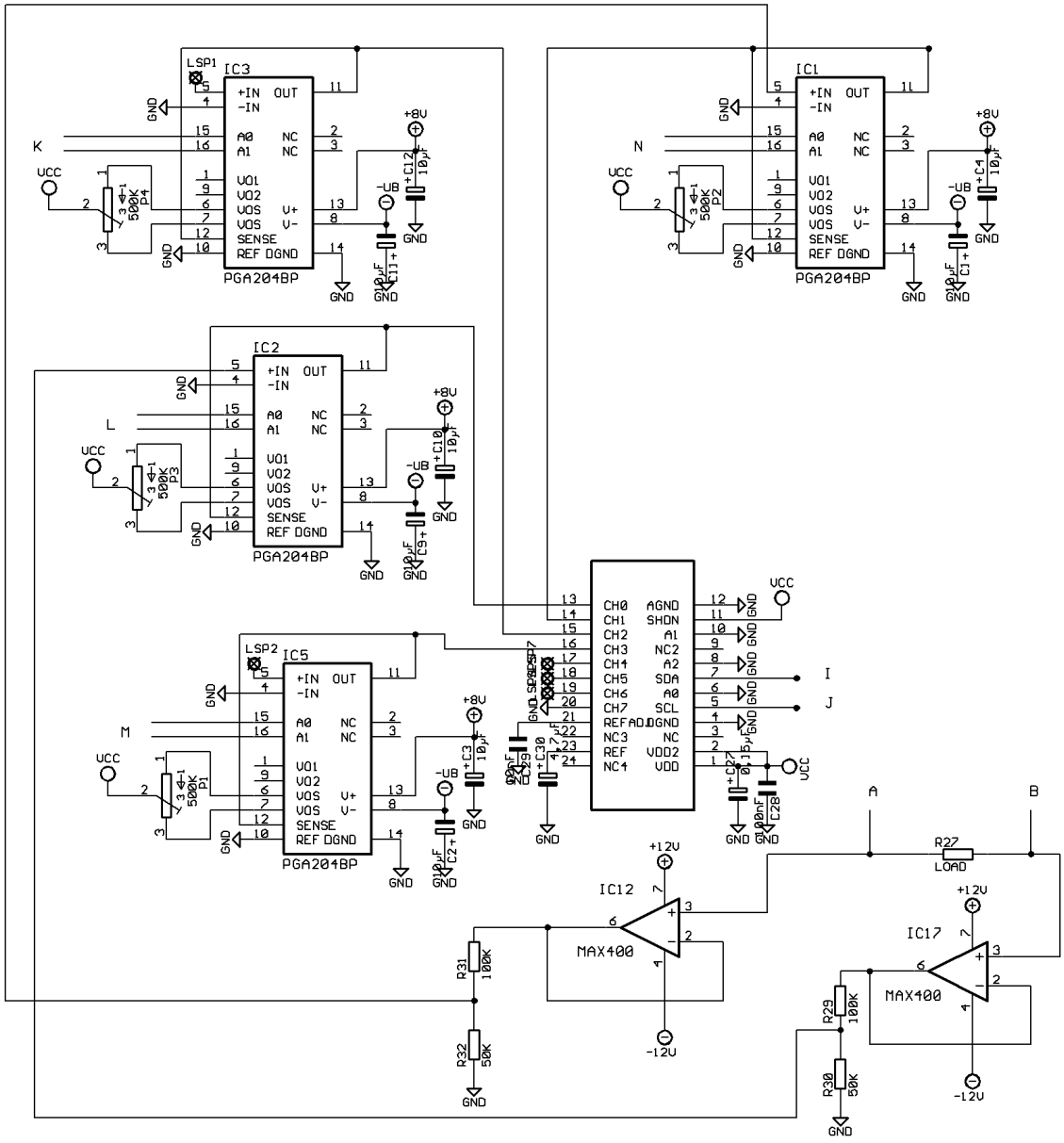


Abbildung 40 Analog/Digital-Wandler mit Verstärkerstufe

Die maximale Spannung, die mit dem Analog/Digital-Wandler gemessen werden kann, ist seine Referenzspannung U_{REF}, die in dieser Schaltung 4,096 V beträgt. Da aber wie in Kapitel 5.2.1 beschrieben, Spannungen bis 8 V auftreten können, ist es notwendig, diese Spannung herabzusetzen. Daher werden die Ausgangsspannungen der Spannungsfolgeschaltung einer Spannungsteilerschal-

tung zugeführt, die aus Präzisionswiderständen vom Typ 8E16D Econistor der Firma Rohpoint realisiert wurde, um Fehler durch Veränderung dieser Widerstände durch Temperaturschwankungen oder Alterung so weit wie möglich zu reduzieren. Die Ausgangsspannung dieser Schaltung lässt sich anhand folgender Gleichung bestimmen:

$$U_{out} = \frac{R_{31}}{R_{32} + R_{31}} U_{in} \tag{25}$$

Nachdem für diese Schaltung Widerstände mit Werten von 100,0 kΩ für R_{31} und 50,0 kΩ für R_{32} eingesetzt wurden, beträgt die Abschwächung dieser Schaltung 0,666. Für den anderen Spannungssteiler gilt das Gleiche. Als Alternative zu dieser Kombination aus Spannungsfolger und Spannungsteiler wäre ein nichtinvertierender Verstärker mit einem Verstärkungsfaktor kleiner als Eins denkbar. Da aber diese Verstärker sehr instabil sind und leicht einen Schwingkreis bilden können, wurde diese Möglichkeit verworfen.
Sind die gemessenen Ströme um ein Vielfaches kleiner als die Referenzspannung des Analog/Digital-Wandlers, ist es nötig diese Spannung zu verstärken, um eine hohe Auflösung und ein gutes Signal-Rausch-Verhältnis zu erreichen. Diese Nachverstärkung erfolgt hier mit einem Verstärkerbaustein vom Typ PGA204 (IC1 und IC2 in Abbildung 40), dessen Spannungsverstärkung in den Stufen 1, 10, 100 und 1000 über eine digitale Schnittstelle eingestellt werden kann. Der Offsetfehler dieser Verstärker kann mit einem Trimmpotentiometer kompensiert werden. Die beiden Kondensatoren, die an Spannungsversorgungsleitungen angeschlossen sind, dienen auch hier der Spannungsstabilisierung.
Zwei weitere digital einstellbare Verstärker (IC3 und IC5 in Abbildung 40) sind für zukünftige Erweiterungen vorgesehen. Sie bieten zum Beispiel Anschlussmöglichkeiten von Sensoren zur Erfassung von Druck und Temperatur innerhalb der Batterie. Je nach eingestelltem Verstärkungsfaktor darf dabei die Ausgangsspannung des Sensors zwischen ±4 mV und ±4 V liegen.
Alle Ausgangsspannungen der Verstärker werden mit dem Analog/Digital-Wandler digitalisiert.
Da die Genauigkeitsanforderungen an die Spannungsmessung bei diesem Gerät deutlich geringer sind als bei der Messapparatur zur Bestimmung von Phasendiagrammen (siehe Abschnitt 2.7.2.5), kann hier zu Gunsten einer höheren Messgeschwindigkeit auf einen hochauflösenden Analog/ Digital-Wandler verzichtet werden.
Zur Digitalisierung der Spannungen wurde auf eine vorhandene Schaltung [8],[9] zurückgegriffen, die auf dem Analog/Digital-Wandler MAX128 von Maxim Integrated Products basiert. So konnte bei der Erstellung der Software durch die Verwendung bereits früher programmierter Funktionen zur Wandleransteuerung viel Zeit gespart werden. Der MAX128 hat eine Auflösung von 12 Bit und acht Eingangskanäle, die über einen integrierten Multiplexer per Software ausgewählt werden. Jeder Kanal kann uni- oder bipolar ausgelesen werden, was Spannungen von 0 bis V_{ref} oder - V_{ref} bis V_{ref} entspricht, wobei sich die Anzahl der Kanäle bei bipolarem Betrieb auf die Hälfte reduziert [10].
V_{ref} ist die Referenzspannung, die bei einer Beschaltung wie in Abbildung 40 von der internen Spannungsreferenz des MAX128 geliefert wird und 4,096 V beträgt [10]. Die Datenübertragung zum Mikrocontroller erfolgt über eine serielle Zweidrahtschnittstelle (Datenleitung SDA (I) und Taktleitung SCL (J)), die zum I²C-Bus kompatibel ist. Das Protokoll für diesen Baustein ist in seinem Datenblatt [10] und im Anhang 11.7.3.3 beschrieben.

5.2.4 Mikrocontroller und Interface

Im Gegensatz zu dem Thermometer, das in der Diplomarbeit des Autors [11] beschrieben wird, erfolgt die Ansteuerung der Analog/Digital- und Digital/Analogwandler nicht über einen PC, der über die IEEE1284 Schnittstelle (Parallelport) angeschlossen wird. Bei dieser Anschlussmöglichkeit wird am Steuerrechner sehr viel Systemlast erzeugt, da dieser das komplette Protokoll aller verwendeten Bauteile abwickeln muss. Aufgrund dessen und der üblichen Begrenzung der Anzahl dieser Schnittstellen auf maximal drei Stück pro PC, kann so nur eine geringe Anzahl an Messgeräten pro Rechner angeschlossen werden. Soll die Steuersoftware auf modernen Betriebsystemen wie Microsoft Windows XP oder Windows 2000 eingesetzt werden, so ist eine direkte Ansteuerung dieser Schnittstelle auch nicht möglich, da hierfür direkte Portzugriffe nötig sind [12], [13].
Eine deutlich günstigere Möglichkeit für die Ansteuerung von Wandler, Verstärker und Relais stellt die Verwendung eines Mikrocontrollers dar. Über ein serielles Interface werden dann diesem Wandler die Einstellungen für die einzelnen Messungen übermittelt, der diese daraufhin autonom ausführt. Die erfassten Messwerte werden in einer einstellbaren Datenrate über dieses Interface an den PC übermittelt.
In diesem Gerät wird wie bei der Messapparatur zur Bestimmung von Phasendiagrammen auch (siehe Abschnitt 2.9) ein ATMEGA16 der Firma ATEML (San Jose) eingesetzt. Dieser 8-BIT RISC Kontroller zeichnet sich vor allem durch seinen großen Flash EEPROM Speicher von 16 kB und seinem großen SRAM von 1 kB aus [14], so dass auch umfangreiche Programme in diesem Kontroller gespeichert werden können, ohne dass ein separater EEPROM Speicher eingesetzt werden muss. Dank der RISC Technologie und seiner hohen Taktfrequenz von bis zu 16 MHz[23] [14] ist auch eine vier Mal schnellere Ansteuerung der Wandler möglich. Die Programmierung dieses Bausteins erfolgt mittels eines ATSTK-500 Starterkits über seine ISP Schnittstelle, die in Abbildung 41 als SV6 bezeichnet ist:

[23] Bei dieser Schaltung wird der interne RC-Oszillator des Mikrocontrollers eingesetzt, der auf seine maximale Frequenz von 8 MHz gestellt ist.

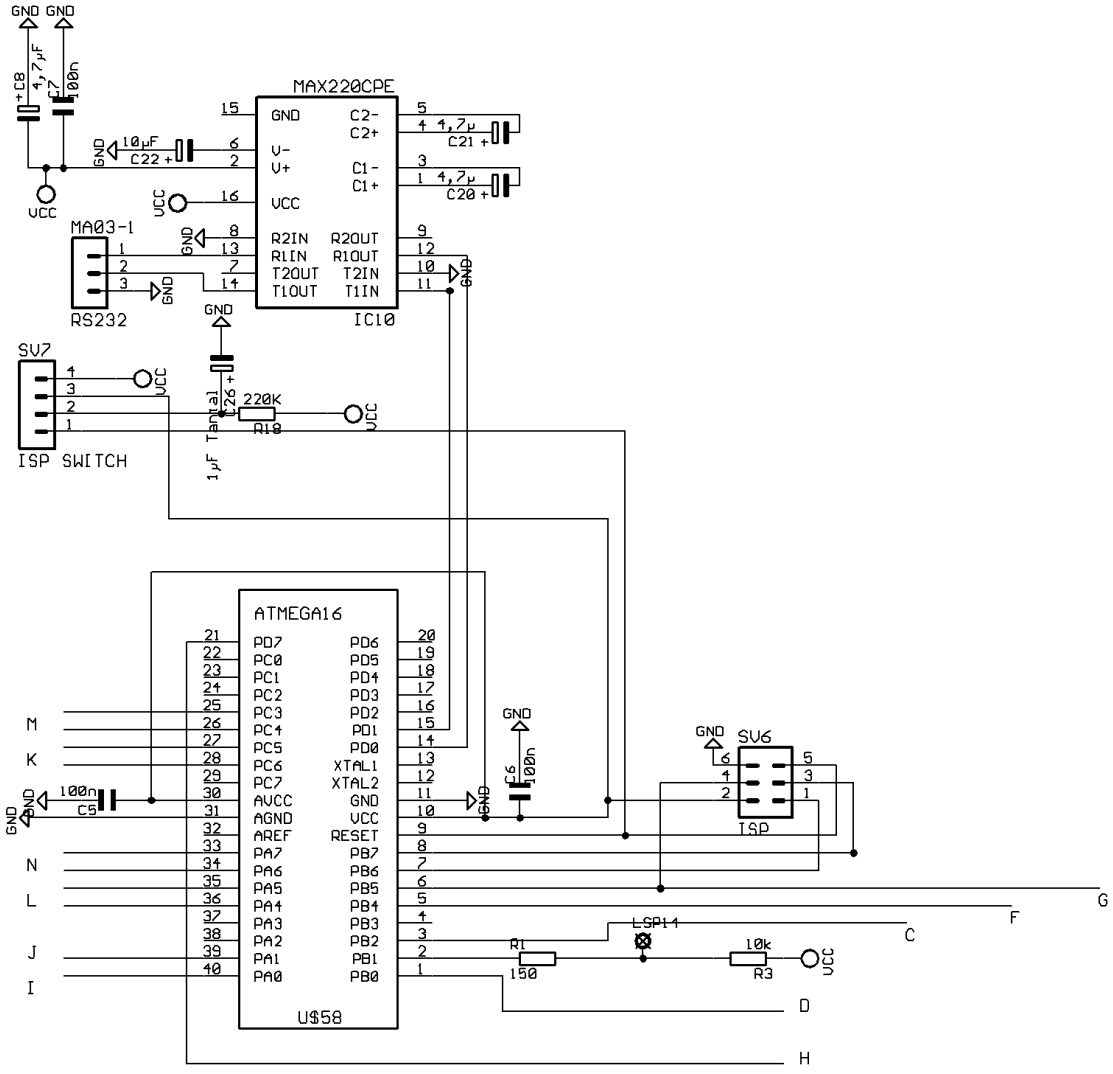


Abbildung 41 Mikrocontroller mit RS-232 Interface

Über den Jumper SV7 kann der Mikrocontroller während des Programmiervorgangs vom übrigen Teil der Schaltung abgetrennt werden. Im Normalbetrieb sind die Pins 1 und 2 und die Pins 3 und 4 dieses Jumpers miteinander verbunden. Über die Verzögerungsschaltung aus R18 und C26 wird der Reset-Impuls des Mikrokontrollers erzeugt. Alle Wandler, Verstärker und Relais sind direkt über die Leitungen C und D sowie F bis N mit dem Mikrocontroller verbunden (siehe Abschnitt 5.2.1 bis 5.2.3).

Für die serielle Datenübertragung wird der integrierte UART des Mikrocontrollers verwendet. Um die Signalpegel von ±12 V der RS-232 Schnittstelle, über die der PC anschlossen wird, auf die TTL Pegel[24] des Mikrocontrollers anzupassen, wird ein Pegelwandler vom Typ MAX220 verwendet. Der Anschluss an den PC erfolgt über eine 9-polige SUB-D Buchse. Als Übertragungsgeschwindigkeit sind 19200 Baud vorgesehen. Dabei werden acht Daten- und zwei Stoppbits und keine Paritätsprüfung und Flusssteuerung verwendet.

[24] 0 V und 5 V

5.2.5 Mikrocontrollersoftware

Die Streusoftware des ATMEGA16 wurde mit Hilfe der WinAVR (GNU GCC) entwickelt. Der Quelltext dieser Software ist in Anhang 11.7.3 aufgeführt.

Die Hauptroutine des Steuerprogramms besteht aus einer Endlosschleife, in der die Steuerkommandos, die der PC an das Batterietestsystem schickt, ausgewertet werden. Zur Ansteuerung des Messgeräts wird eine Protokollsprache verwendet, die an den AT-Standard für Modems angelehnt ist. Da es sich bei diesen Kommandos um reine ASCII Befehle handelt, kann statt der in Abschnitt 5.3 beschriebenen PC-Software auch jedes beliebige ASCII Terminal wie zum Beispiel Hyperterminal der Firma Microsoft verwendet werden, um die grundlegenden Funktionen des Geräts nutzen zu können. Darüber hinaus bietet diese einfache Protokollsprache die Möglichkeit, ohne großen Aufwand PC-Steuerprogramme für andere Messmethoden zu entwickeln, wodurch der Einsatzbereich des Geräts enorm erweitert wird.

In der Tabelle 49 sind die verfügbaren Steuerkommandos für den Mikrocontroller aufgelistet.

Kommando	Beschreibung
hgof	Korrektur des Offset der Messkanäle[25]
hgmm	A/D-Wandler auslesen, alle Kanäle mit Verstärkung > 1 werden gemessen, Messdauer = Integrationszeit
hgms	A/D-Wandler auslesen, alle Kanäle mit Verstärkung > 1 werden gemessen, keine Integration
hgiz	Integrationszeit in Vielfachen von 40 ms, (1 bis 1000000)
hgpaXY	Setzt den Verstärkungsfaktor Y[26] für Kanal X
hgtu	Steuerspannung in µV ausgeben
hgti	Konstantstrom in µA ausgeben
hgts	Schnittstellentest, gibt „test“ zurück
hgsg	Galvanostat für große Ströme einstellen[27]
hgsk	Galvanostat für große Ströme einstellen
hgz1	Zelle an
hgz0	Zelle aus
hilfe	Ausgabe dieser Befehlliste

Tabelle 15 Steuerkommandos des Zyklisiergeräts

Das Mikrocontrollerprogramm ruft je nach ausgewähltem Programmpunkt die entsprechenden Routinen auf, mit denen zum Beispiel der Analog/Digital-Wandler ausgelesen wird.

Für eine Batteriezyklisierung wird dann vom PC-Messprogramm aus eine geeignete Reihenfolge von Steuerbefehlen an das Messgerät geschickt.

[25] Die Methode der Offsetkorrektur wird von Multerer [8] beschrieben.

[26] Zulässige Werte für den Verstärkungsfaktor (0,1,10,100,1000)

[27] Hier wird nur das Relais umgeschaltet. Die Steuerspannung bleibt die gleiche. Wird bei vorgegebener Steuerspannung das Relais umgeschaltet, wird auch der Strom verändert. Soll zwischen beiden Strombereichen gewechselt werden, so ist es sinnvoll, zuerst die Messzelle abzutrennen, dann das Relais umzuschalten und die Steuerspannung einzustellen.

5.2.6 Spannungsversorgung

An die Spannungsversorgung des Batterietestsystems werden hohe Anforderungen hinsichtlich Spannungsstabilität gestellt. Darüber hinaus ist eine starke Unterdrückung von Störsignalen nötig. Das Netzteil, dessen Schaltplan in Abbildung 42 dargestellt ist, wurde daher mit klassischen Linearreglern aufgebaut, da diese gegenüber einem Schaltnetzteil deutlich weniger Störungen produzieren. Eine maximale Störungsunterdrückung wird dadurch erreicht, dass zwei Spannungsregler hintereinander geschaltet werden [15].

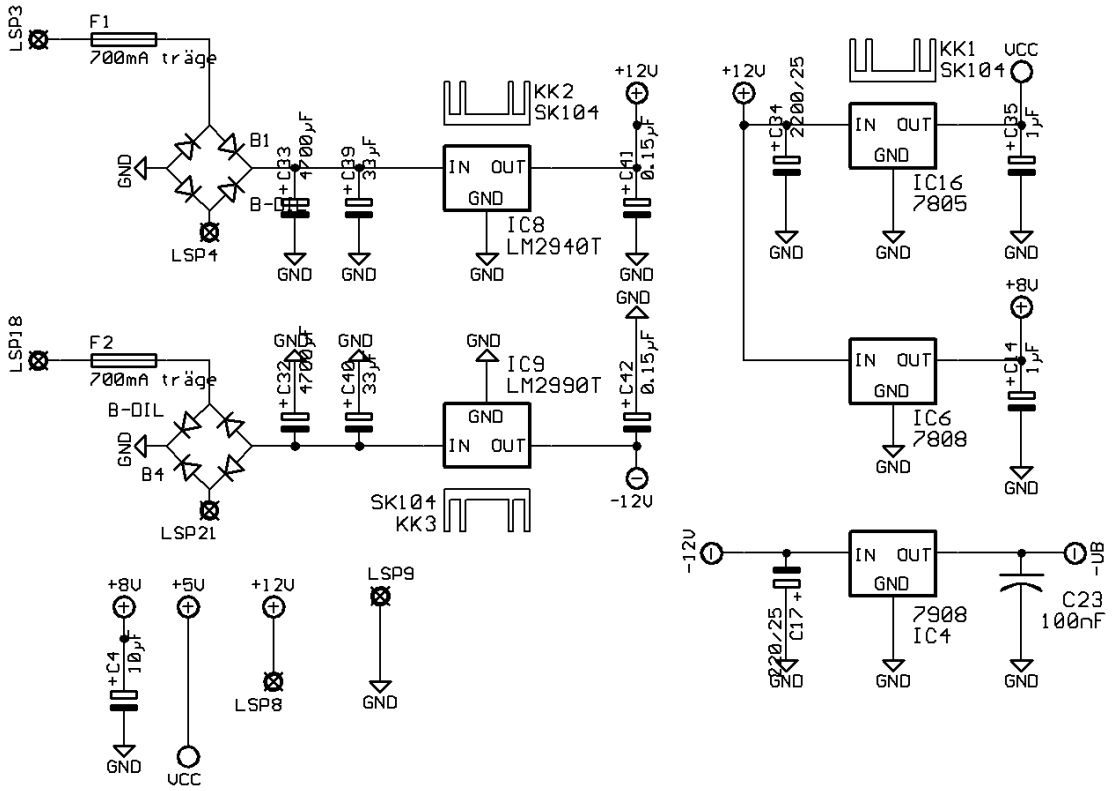


Abbildung 42 Spannungsversorgung des Zyklisiergeräts

Die Kondensatoren dienen zur zusätzlichen Stabilisierung der Ausgangsspannung und zur Schwingungsunterdrückung der Spannungsregler [16], [17]. Durch die beiden großen Elektrolytkondensatoren C_{32} und C_{33} erfolgt eine ausreichende Glättung der pulsierenden Gleichspannung, die durch die Gleichrichter erzeugt wird. Das Netzteil wird über einen Trafo, der nicht in Abbildung 42 gezeigt ist, mit Strom versorgt.

5.3 Steuer- und Messsoftware

Für die Durchführung der Zyklisierungsexperimente wurde eine graphische Steuersoftware entwickelt, die auf den aktuellen 32-Bit Windowsbetriebssystemen wie Windows 2000 und Windows XP einsetzbar ist. Der Quelltext dieser Software, die mit Microsoft Visual Studio 6.0 entworfen wurde, ist in Anhang 11.7.4 aufgeführt. Mit dieser Software ist es möglich, Batterien mit konstantem Strom zu zyklisieren. Eine obere und eine untere Spannungsgrenze dienen dabei als Kriterium für die Umkehrpunkte eines Zyklus. Für die Ausbildung einer geeigneten SEI auf den Elektroden (siehe Anschnitt 6.2.1) und die Formierung der C-Elektrode ist die Lade/Entladerate für den ersten Zyklus meist deutlich geringer als bei den eigentlichen Messzyklen. Daher bietet die Steuersoftware die Möglichkeit, unterschiedliche Ströme für diese Zyklen einzustellen. Darüber hinaus können sowohl bei der Formierung als auch bei den Messzyklen unterschiedliche Lade- und Entladeströme spezifiziert werden. Weitere Parameter für die Messungen sind die Anzahl der Zyklen, die gemessen werden sollen, sowie die maximale Messdauer, nach der die Messung abgebrochen wird. Die Integrationszeit bestimmt die Zeitdauer, über welche die gemessenen Ströme und Spannungen gemittelt werden, bevor sie zum Steuerrechner übertragen werden.
Die erfassten Messdaten können am Ende der Messung im ASCII-Format abgespeichert werden, wobei zwei verschiedene Dateien erzeugt werden. Die erste Datei beinhaltet die rohen Zyklisierungsdaten. Diese umfassen die Abhängigkeit des Stroms, der Zellspannung und der Spannungen der einzelnen A/D-Wandlerkanäle von der Zeit. Das zweite File, das erzeugt wird, enthält die Daten, welche die einzelnen Messzyklen charakterisieren. Diese sind: Nummer des Zyklus, Dauer, aufgeprägter Strom, umgesetzte Ladung und Innenwiderstand der Zelle. Das Format der beiden Dateien ist in Anhang 11.4.2.4 beschrieben.
Der Innenwiderstand der Messzelle wird bei jeder Umschaltung zwischen einem Lade- und Entladezyklus bestimmt. Dazu wird vor und sofort nach der Umschaltung die Klemmenspannung im schnellstmöglichen Messbereich der A-D/Wandler bestimmt. Anhand der Spannungsdifferenz und der Stromänderung kann mit dem Ohmschen Gesetz der Innenwiderstand der elektrochemischen Zelle berechnet werden. Soll während der Zyklisierung am Ende eines Messzyklus ein Impedanzspektrum aufgenommen werden, wie in Kapitel 6.5 beschrieben, kann über die Software die Messung angehalten und die Batterie vom Galvanostaten abgetrennt werden.
Die Bedienung des Zyklisiergeräts und seiner Software ist in Anhang 11.4.2 detailliert beschrieben.

5.4 Zusammenfassung

Mit dem hier gezeigten Gerät zur galvanostatischen Zyklisierung von Batterien wurde ein Batterietestsystem entwickelt, welches das vorhandene Gerät der Firma BaSyTek in idealer Weise im Bereich kleiner Ströme ergänzt. Wie in Kapitel 6 dargestellt, wurde das Gerät für eine Vielzahl von Zyklisierungsexperimenten einsetzt; damit konnte auch seine Langzeitstabilität nachgewiesen werden. Die Genauigkeit des Systems wurde durch eine Größtfehlerabschätzung bestimmt, siehe Anhang 11.3.3. Es erfüllt die eingangs aufgestellten Anforderungen, siehe Abschnitt 5.1. Durch eine umfangreiche Analyse der Beiträge der einzelnen Komponenten zum Gesamtfehler konnten Ansatzpunkte für eine weitere Optimierung des Messgeräts aufgezeigt werden. Die größten Fehler dieser Schaltung werden demnach durch die Analog/Digital- und Digital/Analog-Wandler verursacht. Bei dieser Fehlerrechnung wurde auch die Langzeitstabilität über 10000 h berücksichtigt, so dass auch der Einfluss der Alterung auf das Messgerät bekannt ist.

Aufgrund seiner flexiblen Ansteuerungsmöglichkeit über eine neu entwickelte Steuersprache kann dieses Gerät leicht für andere Fragestellungen, wie zum Beispiel beim galvanischen Beschichten (der Elektroplattierung) eingesetzt werden. Es ist möglich, über weitere Spannungseingänge noch zusätzliche Elektroden und Sensoren, beispielsweise für Druck- und Temperaturmessung, anzuschließen, wodurch sich der Einsatzbereich des Messgeräts nochmals erweitert.
Auch eine eventuelle spätere Serienproduktion wurde bereits bei der Entwicklung des Gerätes in Betracht gezogen.

5.5 Literaturverzeichnis

[1] Abresch, K., Classen, I., *Die coulometrische Analyse,* Weinheim, Verlag Chemie (1961)
[2] STS Thompson Microelectronics, *L200 ADJUSTABLE VOLTAGE AND CURRENT REGULATOR,* Genf (1996)
[3] National Semiconductor, *LM117/LM317A/LM317 3-Terminal Adjustable Regulator,* Santa Clara (2004)
[4] P. Horowitz and W. Hill, *The Art of Electronics*, Cambridge, Cambridge (1989)
[5] Texas Instruments Incorporated, *OPA547 High-Voltage, High-Current OPERATIONAL AMPLIFIER,* Dallas *(2004)*
[6] Maxim Integrated Products Inc., *MAX541, MAX542 +5V, Serial-Input, Voltage-Output 16-Bit DACs,* Sunnyvale (1999)
[7] Maxim Integrated Products Inc., *1ppm/°C, Low-Noise, +2.5V/+4.096V/+5V Voltage References,* Sunnyvale (2001)
[8] M. Multerer, *Entwicklung von Quecksilberfilmelektroden und Untersuchung potentieller Einsatzbereiche in der Abwasseranalyse, Diplomarbeit*, Regensburg (2002)
[9] M. Multerer, H.-G. Schweiger, *Ein selbstgebauter Polarograph, Schwerpunktsarbeit*, Regensburg (2001)
[10] Maxim Integrated Products Inc., *MAX127/128 Multirange, +5V, 12-Bit DAS with 2-Wire Serial Interface*, Sunnyvale (1998)
[11] H.-G. Schweiger, *Entwicklung einer Präzisionstemperaturmessanlage zur schnellen Messung von Phasendiagrammen und chemische und elektrochemische Charakterisierung von Lithium-bis[oxalato(2-)]borat(1-)*, *Diplomarbeit*, Regensburg (2002)
[12] Microsoft, *DOCERR: Port I/O with inp() and outp() Fails on Windows NT, Article ID: Q112298*, Redmond (1996)
[13] H. Custer, *Inside Windows NT*, Redmond (1993)
[14] Atmel Corporation, *ATmega16, 8-bit Microcontroller with 16K Bytes In-System Programmable Flash,* San Jose (2003)
[15] D. Nührmann, *Das große Werkbuch Elektronik*, Franzi´s, Poing (1998)
[16] Fairchild Semiconductor Corp., *Datenblatt zu MC78LXXA/LM78LXXA/MC78L05AA 0.1A Positive Voltage Regulator*, South Portland (2001)
[17] Fairchild Semiconductor Corp., *Datenblatt zu MC79LXXA/LM79LXXA 3-Terminal 0.1A Negative Voltage Regulator*, South Portland (2001)

6 Optimierung neuer Elektrolytlösungen für den Hochtemperatureinsatz in Lithium-Ionen-Zellen

6.1 Zielsetzung

Ein wichtiger Parameter, der die Leistungsfähigkeit einer Lithium-Ionen-Batterie beschreibt, ist ihre Lebensdauer. Hinsichtlich dieser ist im Gegensatz zu einer Primärbatterie nicht die Selbstentladungsrate von entscheidender Bedeutung. Vielmehr stellt bei einer Sekundärbatterie die Zyklenstabilität das entscheidende Kriterium für die Haltbarkeit dar. Darunter versteht man die Anzahl der Zyklen, mit denen die Batterie geladen und entladen werden kann, bis die Kapazität auf einen Teil ihrer nominalen Kapazität, meist 80 % abgefallen ist [1]. Bei höheren Temperaturen und großen Entladeraten tritt verstärkt eine Zersetzung des Elektrolyten oder der Elektrodenmaterialien auf, so dass sich meist eine deutliche Reduzierung der Lebenserwartung einer Batterie ergibt [2].
Sollen Batterien bei hohen Temperaturen eingesetzt werden, stellt die Optimierung der Lebensdauer das größte Problem dar. Ziel dieses Teils der Arbeit ist es, Elektrolyte für Lithium-Ionen-Batterien für den Einsatz bei hohen Temperaturen, hier 60 °C, zu optimieren. Im Gegensatz zur Optimierung für den Tieftemperatureinsatz, siehe Kapitel 4, steht hier nicht die Verbesserung der Leitfähigkeit des Elektrolyten im Vordergrund, sondern die Zyklenstabilität der Batterien.
Außerdem waren auch noch die Endkapazitäten, welche die Kapazität der Batterie nach 500 Zyklen darstellen, zu optimieren.
Ein weiterer Unterschied zur Leitfähigkeitsoptimierung besteht darin, dass bei diesen Versuchen keine Modellsysteme untersucht wurden. Stattdessen wurden für diese Versuche Batteriepacks verwendet, wie sie bereits in der Produktion eingesetzt werden. Da durch Verwendung dieser Zellen die Elektrodenmaterialien schon vorgegeben sind und als Salz LiBOB eingesetzt werden soll, beschränken sich die Parameter für die Optimierung auf die Zusammensetzung der Lösungsmittelmischung und den Zusatz von Additiven. Nach Vorgaben der Gaia Akkumulatorenwerke (Nordhausen) sollten für diese Optimierungen Lösungsmittelmischungen aus Ethylencarbonat, Propylencarbonat und Diethylcarbonat zum Einsatz kommen. Der Flüssigkeitsbereich des Elektrolyten sollte hierbei mindestens den Bereich von 0 °C bis 60 °C abdecken.
LiBOB hat gemäß der Literatur, wie in Abschnitt 1 beschrieben, in diesem Zusammenhang mehrere Vorteile; es ist thermisch stabiler als das Standardsalz $LiPF_6$, ist gegenüber dem billigeren Manganspinell im Gegensatz zu jenem Salz stabil und enthält kein Fluor, was unter Umweltaspekten von Bedeutung ist.

6.2 Grundlagen der Methode und Theorie

Weist sowohl das Salz als auch das Lösungsmittel oder die Lösungsmittelmischung eine ausreichende elektrochemische Stabilität auf, so wird die Zyklenstabilität der Batterie maßgeblich durch das sogenannte „solid electrolyte interface “, kurz SEI [3], beeinflusst [4]. Die Grenzschicht, die sich zwischen der Anode und dem Elektrolyten ausbildet, spielt eine sehr wichtige Rolle in Lithiummetall-, Lithiumionen- und anderen Alkali- und Erdalkalibatterien [4]. Bei Primärbatterien wird durch diese Grenzschicht die Sicherheit, die Selbstentladung, die Hochstromfähigkeit, das

Tieftemperaturverhalten und die Faradaysche Effizienz der Batterie bestimmt. Bei Sekundärbatterien ist sie zusätzlich für die Zyklenausbeute, die Lebensdauer und den irreversiblen Kapazitätsverlust, der beim ersten Zyklus auftritt, verantwortlich. Ziel ist es daher, diese Schicht, die sich auf der Anode ausbildet, für den entsprechenden Anwendungsbereich anzupassen. Da mit zunehmender Temperatur die Effekte, die für die Abnahme der Zyklenstabilität und Zunahme des Innenwiderstandes verantwortlich sind, verstärkt auftreten, ist es sinnvoll, die Optimierung der Zyklenstabilität bei höheren Temperaturen durchzuführen [5].

6.2.1 Die SEI und ihre Bedeutung, Stand des Wissens

Metallisches Lithium hat das negativste Standard-Redoxpotenzial aller Elemente. Da dieses Potenzial negativer als das solvatisierter Elektronen ist, zumindest in Lösungsmitteln wie Ammoniak, Aminen und Ethern, löst sich dieses Metall unter Bildung charakteristischer blauer Lösungen auf [4]. Da die solvatisierten Elektroden sowohl mit dem Elektrolyten als auch mit der Kathode reagieren, ist es notwendig, dass die Elektrolytmischung mindestens eine Komponente enthält, die schnell genug mit dem metallischen Lithium reagiert und eine unlösliche Deckschicht bildet, die das Metall vor weiterer Auflösung schützt [2]. Da in dieser Arbeit keine metallischen Lithiumanoden verwendet wurden, sei an dieser Stelle auf die Literaturübersicht in [2], [5] und [6] verwiesen.
Werden hingegen Kohlenstoffanoden, in die das Lithium intercaliert wird, eingesetzt, so bildet sich diese Schicht durch Reduktion des Elektrolyten beim ersten Ladevorgang bei einem Potenzial von 0,5 bis 1,7 V gegen Li/Li^+. Aufgrund dieser Schichtbildung ist die irreversible Kapazität des ersten Zyklus deutlich größer als die der folgenden Zyklen. Je nach Zusammensetzung des Elektrolyten weist diese Schicht eine unterschiedliche Zusammensetzung auf, da sowohl das Salz, und die Lösungsmittelmischung als auch Verunreinigungen zu ihrem Aufbau beitragen. Schichten aus LiF, Li_2O und Li_2CO_3 werden, wie auch elementares Bor, durch Zersetzung des Salzes und aus Verunreinigungen wie Wasser und Sauerstoff gebildet [4]. Durch die Zersetzung des Lösungsmittels bilden sich meist Polymere im äußeren Teil der SEI [4]. Je nach eingesetztem Salz und der Lösungsmittelmischung wird die SEI durch das Salz oder durch das Lösungsmittel beeinflusst, wobei sich die durch das Lösungsmittel erzeugten Filme durch höhere Temperaturbeständigkeit auszeichnen [13].
Im Gegensatz zu Lithiumanoden dient diese Schicht nicht zur Vermeidung der Auflösung der Anode. Vielmehr schützt sie die Kohlenstoffkathode vor Zerstörung durch Blasenbildung, die durch Cointercalation von solvatisierten Lithiumionen hervorgerufen wird. Diese Cointercalation tritt verstärkt bei Elektrolyten auf, die PC als Hauptbestandteil aufweisen, da dieses bereits bei Potenzialen zersetzt wird, die kleiner sind als das Intercalationspotenzial von Lithium in Graphit [18].
Da aber aufgrund seines großen Flüssigkeitsbereichs, seiner hohen Donorzahl und den günstigen Reaktionsprodukten für die SEI nicht auf dieses Lösungsmittel verzichtet werden kann, stellt gerade dieses Lösungsmittel andererseits besondere Anforderungen an die SEI.

An eine gute SEI werden folgende Anforderungen gestellt [4], [5]:

- Hoher elektronischer Widerstand: reduziert Selbstentladung und verbessert Faradaysche Effizienz,
- Überführungszahl für $t_{Li+} = 1$: verhindert Konzentrationspolarisation und erleichtert Lithium-Intercalation,
- Große ionische Leitfähigkeit: reduziert die Überspannung,
- Homogene Morphologie und chemische Zusammensetzung: fördern gleichmäßige Stromverteilung,
- Gute Haftung auf der Anode,
- Mechanische Stabilität und Flexibilität.

Weitere Aspekte über die Schichtbildung und ihre Charakterisierung finden sich in den Übersichtsarbeiten von Linden [2], Aurbach [5] und Peled [6] und in der darin zitierten Literatur.

6.2.2 Batterieadditive

Methoden zur Verbesserung der SEI und damit zur Optimierung der Batterien lassen sich in zwei Kategorien einteilen. Zum einen gibt es Methoden, bei denen das Anodenmaterial vor dem Einsatz außerhalb der Batterie vorbehandelt wird. Eine dieser Methoden besteht zum Beispiel darin, die Elektrodenoberfläche unter reduktiven Bedingungen vorzubehandeln, wodurch eine starke Reduzierung der irreversiblen Kapazität erreicht wird [5]. Durch gezielte Oxidation der Elektrodenoberfläche lässt sich ebenfalls eine Verbessung der Zyklenstabilität erreichen [4], [7] und [8]. Durch Zugabe von metallischen Nanopartikeln aus Ni [9] oder Ag [10], [11] wird die Anode hinsichtlich ihrer Impedanz, ihrer irreversiblen Kapazität und ihrer Stabilität verbessert. Die anderen Methoden stellen In-Situ-Methoden dar, bei denen die SEI durch Zugabe von Additiven zur Elektrolytlösung optimiert wird. Da die Zusammensetzung und Vorbehandlung der Elektrodenmaterialien vorgegeben war, konnte nur dieser Ansatz für die Optimierung gewählt werden.
Additive, die dem Elektrolyten zugesetzt werden, lassen sich wiederum in drei Kategorien einteilen. Eine Strategie besteht darin, Substanzen einzusetzen, die als Vorläufersubstanzen für eine SEI dienen. Diese Verbindungen weisen dabei eine deutlich größere Reaktionsgeschwindigkeit auf als die anderen Komponenten der Batterie, wodurch die SEI maßgeblich durch diese Additive und nicht mehr durch die Zusammensetzung des Elektrolyten bestimmt wird. Beispiele für diese Additive sind in Tabelle 3 aufgeführt. Die in dieser Tabelle gezeigten Additive entsprechen dabei typischen Batterielösungsmitteln, deren Reaktivität durch Derivatisierung mit funktionellen Gruppen oder durch Austausch mit homologen Atomen vergrößert wurde, um ihre Reaktionsgeschwindigkeit an der Elektrodenoberfläche zu erhöhen. Dimethyldicarbonat zerfällt in Methanol und CO_2, die beide zum Aufbau der SEI als LiOMe und $LiCO_3$ beitragen [19]. Vinylentrithiocarbonat, das bisher noch nicht als Additiv in Lithium-Ionen-Batterien eingesetzt worden ist, wurde aufgrund der großen strukturellen und reaktiven Ähnlichkeiten zu Vinylencarbonat und Ethylensulfit vom Autor dieser Arbeit als weiteres potentielles Additiv angesehen, und wird in dieser Arbeit erstmals getestet.

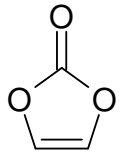
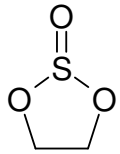
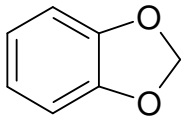
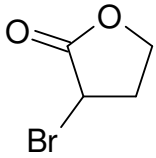
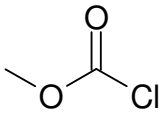
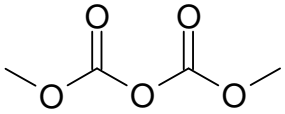
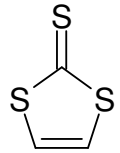
Additiv	Strukturformel	Literatur
Vinylencarbonat		[12],[13],[14], [15], [16]
Ethylensulfit 1,3,2-Dioxathiolane, 2-oxid		[14],[17]
1,3-Benzodioxolan		[13]
α-Brom-γ-butyrolacton		[18]
Methylchlorformiat		[18]
Dimethyldicarbonat		[19]
Vinylentrithiocarbonat 1,3-Dithiol-2-thion		

Tabelle 16 Additive für Lithium- und Lithium-Ionen-Batterien, die als SEI Vorläufersubstanzen dienen

Ein weiterer Ansatz eines Batterieadditivs ist die Zugabe von Metallsalzen [20], bei denen das Metallion nicht in die Elektrodenmaterialien intercalieren kann und einen Film bildet, der für Lithiumionen permeabel ist. Die Wirkungsweise dieser Schicht entspricht dabei der Methode von Honbo [10], nur dass hier die SEI in situ erzeugt wird und nicht vorher im Hochvakuum aufgedampft wird. Tetraethylenglykoldimethylether weist mit seiner Polyglykoletherstruktur eine ähnliche Struktur wie eine SEI auf, die durch Polymerisation von Carbonaten entsteht. Daher wird diese Substanz als Bausstein für eine geeignete SEI angesehen [13]. Die Ethylester, Toluol und Nitromethan dienen ebenfalls der Ausbildung einer SEI [2].

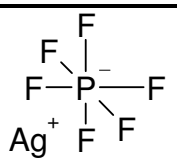
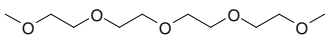
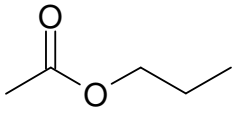
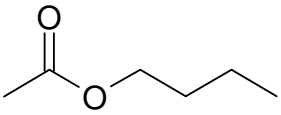
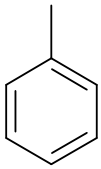
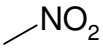
Additiv	Strukturformel	Literatur
Silberhexafluorophosphat		[20]
Tetraethylenglykoldi-methylether 2,5,8,11,14-Pentaoxapentadecan		[13]
Propionsäureethylester		[2]
Buttersäureethylester		[2]
Toluol		[2]
Nitromethan		[2]

Tabelle 17 Additive für Lithium- und Lithium-Ionen-Batterien

Der typische Konzentrationsbereich, in dem die hier gezeigten Additive eingesetzt werden und ihre optimale Leistungsfähigkeit entfalten, liegt im Bereich von 4 bis 5 Massenprozent der Elektrolytlösung [12] -[17] und [20]. Die genannten Additive wurden in Lithium- und Lithium-Ionen-Batterien getestet bei denen $LiClO_4$, $LiPF_6$, $LiBF_4$, $LiAsF_6$ und $LiPF_3(CF_2CF_3)$ als Leitsalze eingesetzt werden. Über den Einsatz von Additiven in Batterien mit Chelatoboraten, insbesondere LiBOB, als Elektrolyt wurde bis jetzt noch nicht berichtet. Daher konnte mithilfe von Tabelle 3 und Tabelle 17 noch keine Vorauswahl getroffen werden. Da sich bei den in der Literatur aufgeführten Beispielen Elektrolytzusammensetzungen, Elektrodenmaterialen und Versuchsbedingungen deutlich unterscheiden, kann auch durch Vergleich dieser Daten untereinander keine Vorauswahl getroffen werden. Soll also ein optimales Additiv für Batterien mit LiBOB als Leitsalz gefunden werden, so ist es sinnvoll, alle in Tabelle 3 und Tabelle 17 aufgeführten Additive unter gleichen Bedingungen auf ihre Eignung hin zu testen.

6.3 Experimentelle Durchführung

Auch bei diesen Experimenten ist die Vermeidung einer Kontamination der eingesetzten Salze, Lösungsmittel und Elektroden aus den in Abschnitt 7.1 genannten Gründen von äußerster Wichtigkeit, um eine Verfälschung der Ergebnisse zu vermeiden. Daher wurden sämtliche durchgeführ-

ten Arbeitsschritte, bis auf die Zyklisierexperimente selbst, im Handschuhkasten unter Argon als Schutzgas durchgeführt (Siehe Kapitel 11.1.1.1). Für die Lösungsmittelmischungen wurden Ethylencarbonat (EC), Propylencarbonat (PC), Diethylcarbonat (DEC), Ethylpropionat (EP) und Ethylbutyrat (EB) eingesetzt. Um Fehler durch Verunreinigungen der Lösungsmittel ausschließen zu können, wurden ausschließlich Lösungsmittel der Qualität Selectipur® der Firma Merck (Darmstadt) verwendet. Der Wassergehalt dieser zertifizierten Lösungsmittel wurde zusätzlich mittels Karl-Fischer-Titration, siehe Kapitel 11.1.6, bestimmt. Er lag bei allen Lösungsmitteln unter 30 ppm. Als Salz wurde LiBOB der Firma Chemetall (Frankfurt) der neuen Charge eingesetzt. Die zur Herstellung der Lösungen verwendeten Geräte sind in Kapitel 7.3.3.1 aufgeführt.

α-Brom-γ-butyrolacton und Methylchlorformiat (82 ppm H_2O) der Firma Aldrich wurden in der Qualität p.a. eingesetzt. Silberhexafluorphosphat und Vinylencarbonat[28] in der Qualität p.a. stammten von der Firma Acros, Dimethyldicarbonat (170 ppm H_2O) Qualität p.a. von Sigma. Der Tetraethylenglykoldimethylether (457 ppm H_2O), p.a., wurde von der Firma VWR geliefert. Das Vinylentrithiocarbonat, 1,3-Benzodioxolan (1133 ppm H_2O), Ethylensulfit (138 ppm H_2O) und der Buttersäureethylester (5 ppm H_2O), alle p.a., stammten von der Firma Fluka. Nitromethan (133 ppm H_2O), Toluol (190 ppm H_2O) und Propionsäureethylester (157 ppm H_2O) wurden in der Qualitätsstufe Selectipur der Firma Merck eingesetzt. Die in Klammern angegeben Wassergehalte wurden miitels Karl-Fischer-Titration bestimmt.

Für die Zyklisierungsexperimente wurden von den Gaia Akkumulatorenwerken zur Verfügung gestellte Batteriepacks verwendet. Bei diesen Packs wird als Kathodenmaterial $Li_xNi_yCo_zO$ eingesetzt, dem Leitruß und ein PVdF-basierter Binder zugesetzt wurde. Als Stromableiter dient für diese Elektrode eine Kupferfolie. Die Anode besteht aus dem „Kohlenstoff" MCMB 10-28, dem wieder Leitruß zugesetzt wurde. Über einen PVdF-basierten Binder ist dieses Material auf einer Aluminiumfolie als Stromableiter fixiert. Die Fläche der Elektroden beträgt 25 cm^2. Bei einer reversiblen Kapazität von 1,63 mAh/cm^2 für die Kathode und 1,79 mAh/cm^2 für die Anode ergibt sich für die Kapazität der Zelle 40 mA/h. Über einen Separator aus mit PVdF beschichtetem Polyolefin sind die beiden Elektroden voneinander getrennt. Diese Zellen wurden mit ca. 0,7 mL des zu testeten Elektrolyten befüllt.

Um Messungen an diesen Batteriepacks durchführen zu können, ist es nötig, diese sicher zu verschließen, um eine Kontamination des Elektrolyten und des Elektrodenmaterials ausschließen zu können. Das Verschweißen der Batteriepacks mit haushaltsüblichen Folienschweißgeräten ist nicht möglich, da durch die eingearbeitete Aluminiumfolie so viel Wärme abgeführt wird, dass die Kunststofffolie nicht verschmilzt. Industriefolienschweißgeräte, wie sie Gaia einsetzt, konnten mangels Trockenraum nicht verwandt werden, da diese Geräte viel zu groß für den Einsatz in Handschuhkästen sind. Um dieses Problem zu umgehen, wurde ein Klemmrahmen entwickelt mit dem es möglich ist, die Batteriepacks durch Druck sicher zu verschließen. In Abbildung 43 ist ein solcher Rahmen gezeigt, der mit 12 Innensechskantschrauben verschraubt wird.

[28] Um einer Polymerisation vorzubeugen, wird Vinylencarbonat mit einem Stabilisator versetzt. Daher wurde es vor der Verwendung unter reduziertem Druck destilliert. Der Wassergehalt konnte aufgrund von Nebenreaktionen, vermutlich Addition von Iod an die Doppelbindung des Vinylencarbonats, nicht bestimmt werden. Die Wasserbestimmung mittels der NMR-Methode, siehe Abschnitt 7.3, kann nicht durchgeführt werden, da Vinylencarbonat Protonen enthält.

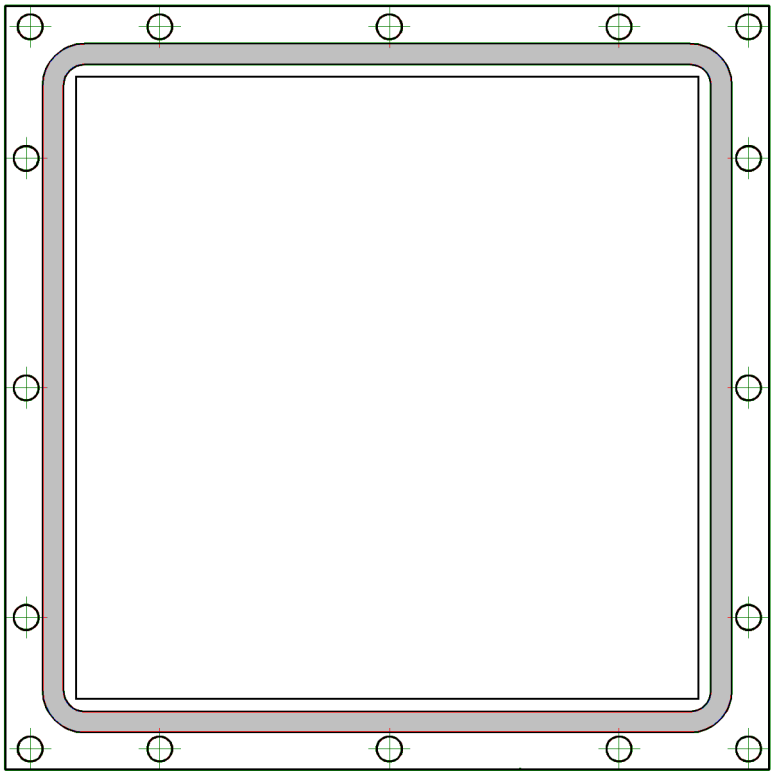


Abbildung 43 Klemmrahmen für Batteriepacks

Um eine sichere Druckübertragung vom Rahmen auf die Folie des Batteriepacks sicherzustellen, wird in die in Abbildung 43 grau dargestellte Nut in beide Hälften des Rahmens eine O-Ring-Dichtung eingelegt.

Die Zellen wurden, nach einem Formierungszyklus mit einer Lade/Entladerate von 1/10 C, für 500 Zyklen mit einer Laderate von 1 C [29]zyklisiert, um eine starke Belastung der Batterie zu simulieren. Die Grenzpotenziale der Zyklen betrugen dabei 4,2 V für den Ladezyklus und 3,0 V für den Entladezyklus, woraus nochmals eine sehr große Belastung für die Batterie resultiert. Diese drastischen Bedingungen wurden ausgewählt, um schlechtest mögliche Einsatzbedingungen zu simulieren. Dadurch ist es möglich, die Versuchsdauer zu verkürzen. Für diese Experimente wurden die vom Autor dieser Arbeit entwickelten Batterietestsysteme eingesetzt, die in Kapitel 5 beschrieben sind.

Vor der Zyklisierung, nach dem Aufladen der Batterie mit dem Formierungsladestrom und nach 500 Zyklen wurde mit dem in Anhang 11.1.3 beschriebenen Zahner IM6 Gerät ein Impedanzspektrum im Frequenzbereich von 100 kHz bis 10 mHz aufgenommen. Die Messung des Impedanzspektrums wurde dabei beim Klemmenpotenzial der Batterie durchgeführt, um diese nicht zu belasten und somit das Messergebnis nicht zu verfälschen.

Wie eingangs gefordert, wurden die Messungen bei 60 °C durchgeführt, indem die Zellen in das Ölbad eines Thermostaten vom Typ RK 20 der Firma Lauda (Lauda-Königshofen) eingetaucht wurden. Der Flüssigkeitsbereich der untersuchten Elektrolytlösungen wurde mit der in Kapitel 2 beschriebenen Apparatur untersucht. Dabei wurden die gleichen Versuchsbedingungen wie bei der Optimierung der Leitfähigkeit bei tiefen Temperaturen, siehe Kapitel 4.3.1, verwendet.

[29] Der Lade- und Entladestrom einer Batterie wird oft als ein Mehrfaches der Nennkapazität C ausgedrückt. Zum Beispiel 0,1 C, hier beträgt der Strom 140 mA, bei einer Zelle mit einer Kapazität von 1,4 Ah.

6.4 Untersuchte Systeme

Die untersuchten Systeme lassen sich in zwei Gruppen einteilen. Zum einen wurden reine Carbonatmischungen untersucht, bei denen ausgehend von einer Lösung aus EC/PC (Massenverhältnis 3/2) DEC als weitere Lösungsmittelkomponente bis zu einem Gehalt von 30 Gew% hinzugefügt wurde. Die Leitsalzkonzentration wurde von 0,5 mol/kg, über 0,7 mol/kg bis zu 0,9 mol/kg variiert. Ausgehend von diesen Vorgaben wurde eine Reihe von Elektrolytmischungen hergestellt, deren Zusammensetzungen und Tieftemperaturgrenzen, die nach der in Abschnitt 6.3 beschriebenen Methode bestimmt wurden, in Tabelle 18 zusammengefasst sind.

Lösung	ξ_{EC} /%	ξ_{PC} / %	ξ_{DEC} / %	m_{LiBOB} / mol/kg	Einsatzbereich / °C
1	66,7	33,3	0	0,50	-0,30
2	66,7	33,3	0	0,70	0,19
3	66,7	33,3	0	0,90	unlöslich bei 25 °C
4	66,7	23,3	10,0	0,50	4,55
5	66,7	23,3	10,0	0,70	-2,23 bis 8,38
6	66,7	23,3	10,0	0,90	unlöslich bei 25 °C
7	66,7	13,3	20,0	0,50	12,89
8	66,7	13,3	20,0	0,70	unlöslich 25 °C
9	66,7	13,3	20,0	0,90	unlöslich 25 °C
10	66,7	0,33	30,0	0,50	14,48
11	66,7	0,33	30,0	0,70	unlöslich 25 °C
12	66,7	0,33	30,0	0,90	unlöslich 25 °C

Tabelle 18 Tieftemperaturstabilität von LiBOB Elektrolytlösungen mit EC, PC und DEC als Lösungsmittelkomponenten

Da der Einsatzbereich der meisten dieser Lösungen, nicht wie in Abschnitt 6.1 gefordert bis 0 °C reicht, wurden noch weitere Lösungen hergestellt, bei denen statt DEC ein Ester als weitere Lösungsmittelkomponente verwendet wird. Im Gegensatz zu Essigsäureethylester, das bei den Tieftemperaturoptimierungen zum Einsatz kam, siehe Kapitel 4, werden hier Propionsäureethylester (EP) und Butansäureethylester (EB) aufgrund ihres höheren Siedepunkts der Vorzug gegeben. Die Zusammensetzungen dieser Lösungen sind in Tabelle 19 aufgeführt.

Lösung	ξ_{EC} /%	ξ_{PC} / %	ξ_{DMC} / %	ξ_{EMC} / %	ξ_{EB} / %	ξ_{EP} / %	m_{LiBOB} / mol/kg	Einsatzbereich / °C
13	20,3	3,50	1,40	--	--	74,8	0,86	<-30
14	20,3	7,16	0,24	15,7	--	56,6	0,76	<-30
15	23,3	10,0	--	--	--	66,7	0,90	<-30
16	33,3	--	--	--	--	66,7	1,00	unlöslich 25 °C
17	28,6	2,98	1,28	--	23,6	--	0,86	< 0
18	33,6	7,17	2,40	--	56,6	--	0,76	unlöslich 25 °C
19	32,3	8,80	--	--	58,9	--	0,90	unlöslich 25 °C

Tabelle 19 Tieftemperaturstabilität von LiBOB Elektrolytlösungen mit EC/PC und Estern als Lösungsmittelkomponenten

Da die Lösungen 1, 2 und 5 aus Tabelle 18 und 13-15 und 17 aus Tabelle 19 eine ausreichende Tieftemperaturstabilität aufweisen, wurden sie auf ihre Zyklenstabilität bei hohen Temperaturen mit der in Abschnitt 6.3 beschriebenen Methode hin untersucht.
Für eine weitere Optimierung der Zyklenstabilität der Elektrolyte können die in Tabelle 16 und Tabelle 17 aufgeführten Additive den Batterieelektrolyten zugesetzt werden. Da die Lösung, die in Tabelle 18 in der ersten Zeile aufgeführt ist, die beste Zyklenstabilität aufweist, wurde diese Lösung für eine weitere Optimierung ausgewählt und mit ihr wurden die einzelnen Additive, die in Tabelle 3 und Tabelle 17 aufgeführt sind, getestet. Für diese Versuche wurden vier Gewichtsprozent an Additiv zugesetzt und in die in Abschnitt 6.3 beschriebenen Batterien eingefüllt und zyklisiert.

6.5 Wesentliche Ergebnisse

Aufgrund der großen Anzahl an einzelnen Messkurven und Spektren werden in diesem Abschnitt nur die wesentlichen Ergebnisse präsentiert. Die einzelnen Messdaten wurden archiviert.
Die umgesetzte Ladung Q berechnet sich aus der Dauer t eines Lade- oder Entladevorgangs und des dabei eingestellten konstanten Stroms I nach Gleichung (26).

$$Q = I \cdot t \tag{26}$$

Ladeströme weisen dabei ein positives Vorzeichen auf, Entladeströme ein negatives. Die relative Kapazität C_{rel} der Batterie bei einem Lade- beziehungsweise Entladevorgang wird auf den Entladestrom des ersten Messzyklus Q_1^- bezogen und auf 100 % normiert.

$$C_{REL} = \frac{Q_i^-}{Q_1^-} \cdot 100\% \tag{27}$$

Die Coulombeffizienz C_{eff}, welche die Effizienz der einzelnen Ladezyklen beschreibt, entspricht dem Ladungsverhältnis der bei den einzelnen Ladevorgängen umgesetzten Ladungen (Q_i^- für die Entladung und Q_i^+ für den Ladungsvorgang).

$$\mathrm{C_{eff}} = \left| \frac{Q_i^-}{Q_i^+} \right| \cdot 100\% \tag{28}$$

Der Innenwiderstand R_i der Zelle wird, wie in Kapitel 5 beschrieben, anhand der Spannungsdifferenz bestimmt, die bei der Umschaltung zwischen dem Lade- und dem Entladestrom und umkehrt, beobachtet wird. Da die hier durchgeführten Experimente aufgrund der hohen Temperaturen, hohen Entladungsraten und einer Entladung von 100%, sehr extreme Bedingungen für eine Lithium–Ionen-Batterie darstellen, wird als Lebensdauer der Batterie die Anzahl der Zyklen angenommen, bei der ihre Kapazität auf 50 % der Anfangskapazität abgefallen ist. In Abbildung 44 und Abbildung 45 sind typische Ergebnisse der Zyklisierexperimente gezeigt. Abbildung 44 stellt die Abnahme der Kapazität C_{rel}, der Ladung Q mit zunehmender Zyklenzahl dar. Zusätzlich dazu ist

noch die Coulombeffizienz dieser Batterie ausgewiesen, die typischerweise bei LiBOB Batterien nahezu 100 % beträgt.

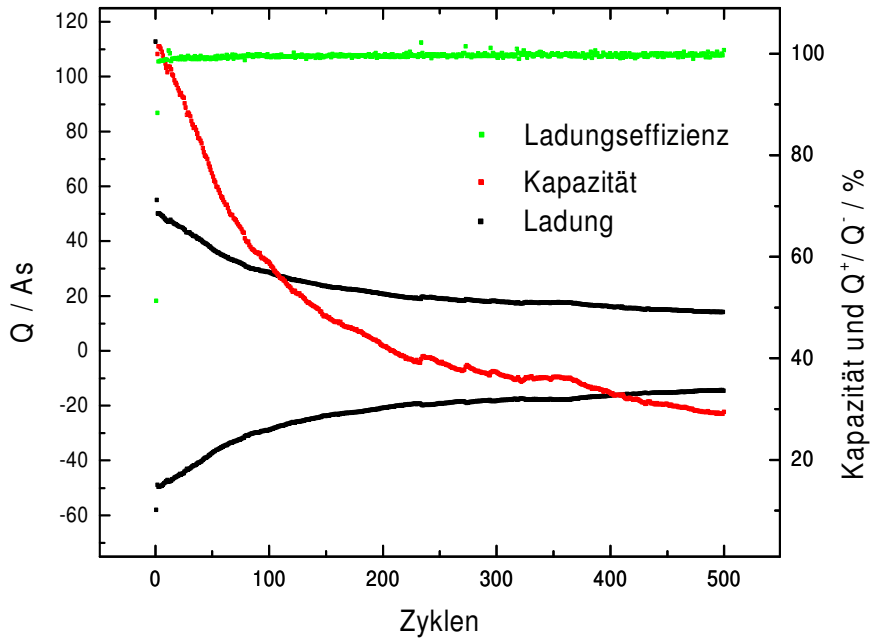


Abbildung 44 Zyklenstabilität der Batterie mit Elektrolyt aus 0,5 mol LiBOB /kg EC/PC 2/1 Mischlösungsmittel + 4 % Toluol

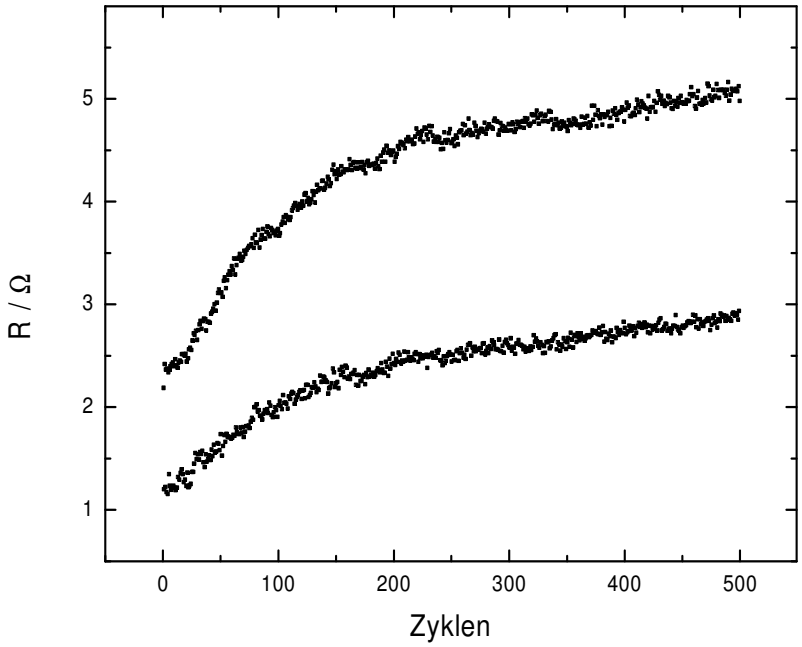


Abbildung 45 Innenwiderstand der Batterie mit Elektrolyt aus 0,5 mol LiBOB /kg EC/PC 2/1 + 4 % Toluol Mischlösungsmittel

Die Veränderung des Innenwiderstandes der Batterie ist in Abbildung 45 gezeigt. Diese Abbildung zeigt zwei Widerstandskurven. Die Kurve mit den kleineren Widerstandswerten entspricht dem Innenwiderstand der Batterie am Ende des Ladevorgangs. Die andere Kurve beschreibt den Widerstand am Ende des Entladevorgangs. Diese Widerstände werden aus den Spannungsänderungen bestimmt, die bei der Umschaltung zwischen Lade- und Entladevorgang und umgekehrt auftreten, siehe auch Kapitel 5.
In Tabelle 20 sind die Ergebnisse der Zyklisierungen der Lösungen ohne Zusatz von Additiven aufgeführt. In dieser Tabelle entspricht $R_{50\%}$ dem Innenwiderstand der Zelle bei 50 % der Anfangskapazität. C_{500} und R_{500} sind die relative Endkapazität und der Innenwiderstand der Batterie am Ende der Zyklisierung.

Lösung	Lebensdauer	$R_{50\%}$ / Ω	C_{500} / %	R_{500} / Ω
1	56	--	19,9	--
2	122	1,7/2,4	15,9	4,3/4,98
5	65	3,6/4,9	--[30]	--[30]
13	31	2,9/3,3	0,5	12,1/13,4
14	69	3,3/4,8	2,0	9,2/10,1
15	41	--[31]	5,4	--[31]

Tabelle 20 Zyklenstabilität von Batterieelektrolyten ohne Zusatz von Additiven

Ein Vergleich der Lösungen untereinander zeigt, dass die Lebensdauer der Elektrolyte, deren Lösungsmittelmischungen sich nur aus Carbonaten zusammensetzen, tendenziell höher ist als bei Elektrolyten, die Ester als zusätzliche Lösungsmittelkomponenten enthalten. Die Ursache für dieses Verhalten dürfte darin liegen, dass die Fähigkeit von Estern, geeignete SEIs auszubilden, deutlich geringer ist, als dies bei Carbonaten der Fall ist. Außerdem könnte sich die Löslichkeit der Reaktionsprodukte der SEI in beiden Lösungsmittelklassen unterscheiden.
Die Carbonatmischungen weisen darüber hinaus einen hohen Anteil an Ethylencarbonat (EC) auf, der mindestens doppelt so groß ist wie bei den anderen Lösungen; so EC ist dafür bekannt, eine sehr gute SEI auszubilden. Daraus kann aber noch nicht sicher geschlossen werden, ob die Zugabe von Estern oder der geringe Gehalt an Ethylencarbonat für die geringere Lebensdauer der Elektrolyte, die Ester enthalten, verantwortlich ist, obwohl es wahrscheinlich ist, dass der Gehalt an EC ausreichen sollte.
Deutlich schlechter schneiden die esterhaltigen Elektrolyte hinsichtlich der Endkapazität nach 500 Zyklen und hinsichtlich des Verlaufs des Innenwiderstands ab, da dieser im Vergleich zu den Carbonat-Mischungen auf mehr als das Doppelte ansteigt.
Obwohl die Lebensdauer der Lösung 1 geringer ist als die der Lösung 2, wurde sie für die weitere Optimierung ausgewählt, da sie über eine höhere Restkapazität verfügt. Die Auswirkung der Additive auf die beiden Lösungen dürfte identisch sein, da sie sich nur im Salzgehalt unterscheiden. In Tabelle 21 sind die Ergebnisse der Zyklisierungsexperimente nach Zugabe von Additiven in der Folge ihrer Qualität aufgelistet.

[30] Dieses Experiment wurde nach 191 Zyklen abgebrochen, daher können keine Aussagen über das Verhalten bis 500 Zyklen getroffen werden.

[31] Zum Zeitpunkt dieser Messung war die Bestimmung des Innenwiderstands in der Steuersoftware der Zyklisiergeräte noch nicht implementiert.

Additiv	Lebensdauer	$R_{50\%}$ / Ω	C_{500} / %	R_{500} / Ω
Tetraethylenglykoldimethylether	264	2,6/3,9	38	2,9/4,4
Propionsäureethylester	246	2,4/4,2	36	2,8/4,5
Methylchlorformiat	204	5,4/7,6	32	6,2/8,2
Buttersäureethylester	187	2,4/4,3	28	3,0/5,1
Toluol	141	2,3/4,2	29,1	2,9/5,1
$AgPF_6$	141	1,8/3,1	21	2,7/4,2
Vinylencarbonat	123	3,2/4,0	18	4,2/5,2
Ethylensulfit	122	3,4/4,5	19	5,2/6,6
ohne Additiv	56	--	20	
Nitromethan	46	3,1/5,0	3,8	8,1/8,9
α-Brom-γ-butyrolacton	33	4,8/7,5	14	7,3/10,4
1,3-Benzodioxolan	32	3,2/5,7	17	6,3/8,8
Dimethyldicarbonat	28	2,8/5,9	15	4,5/9,0
Vinylentrithiocarbonat	11	4,7/6,3	0,2	26/26

Tabelle 21 Auswirkung von Additiven auf die Lebensdauer, Endkapazität und Innenwiderstand von Lithium-Ionen-Batterien

Wie der Tabelle 21 entnommen werden kann, lässt sich die Wirkung der Additive in drei Gruppen einteilen. Zum einen gibt es Additive wie Tetraethylenglykoldimethylether, Propionsäureethylester Methylchlorformiat und Buttersäureethylester, die eine enorme Verbesserung der Lebensdauer und Endkapazität der Batterie bewirken. Durch diese Additive kann die Lebensdauer um den Faktor fünf (!) gesteigert werden. Bei der Endkapazität wurde eine Steigerung auf das Doppelte erreicht. Eine weitere Gruppe von Additiven, wie zum Beispiel Toluol und Ethylensulfit, bringen auch eine sehr gute Verbesserung der Eigenschaften der Batterie, wobei diese aber nicht ganz so ausgeprägt sind. Die dritte Gruppe von Additiven bringt keine Verbesserung. Vielmehr wird die Langzeitstabilität der Batterie deutlich herabgesetzt.
In Abbildung 46 wird anhand der Impedanzspektren, die nach 500 Zyklen aufgenommen wurden, noch einmal der Einfluss von Additiven auf die Zyklenstabilität der Batterien verdeutlicht

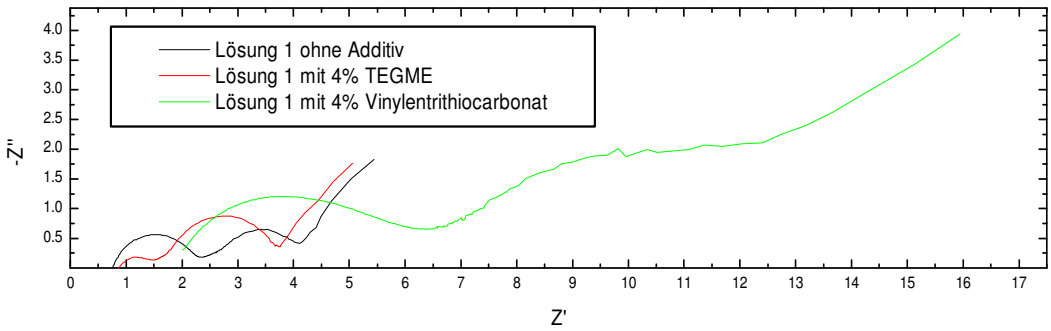


Abbildung 46 Einfluss von Additiven, Impedanzspektrum nach 500 Zyklen

Die schwarze Kurve in dieser Abbildung zeigt das Messergebnis der Batterie, die Lösung 1 als Elektrolyt enthielt. Die grüne Kurve wurde an der Batterie gemessenen, die Tetraethylenglykoldimethylether enthielt, welches sich als das beste Additiv herausgestellt hat. Vinylentrithiocarbonat als das schlechteste Additiv ist in dieser Abbildung rot dargestellt.
In dieser Abbildung wird besonders die Auswirkung der Additive auf die SEI deutlich. Durch den Zusatz von Vinylentrithiocarbonat wird eine sehr stake Deckschicht ausgebildet, welche die Oberfläche der Anode total blockiert.
Die Impedanzspektren zeigen ein Verhalten, wie es bereits von Arbeiten über die SEI an Lithium-Elektroden seit den achtziger Jahren bekannt ist. Demnach kann der erste Halbkreis als Elektrolytwiderstand interpretiert werden, der zweite Halbkreis entspricht einer Kombination eines mit dem Elektrolytwiderstand in Serie geschalteten, parallelgeschalteten R-C-Elements, bestehend aus dem Widerstand der SEI selbst und deren Kapazität sowie dem Durchtrittswiderstand in Serie mit einer Warburgimpedanz und einer parallel geschalteten Kapazität für die große Doppelschicht in den Poren der SEI, selbst wieder in Serie mit dem Elektrolytwiderstand in den Poren, siehe Abbildung 47.

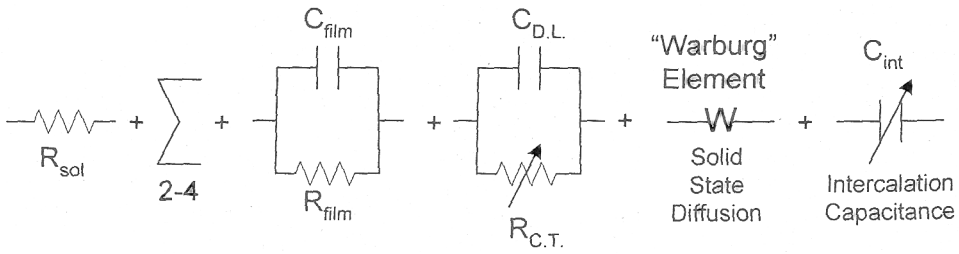


Abbildung 47 Fit-Modell für Impedanzspektren einer einzelnen Batterieelektrode [21]

Bei derartigen Messungen an Einzelelektroden mag es noch sinnvoll sein, die einzelnen Elemente des Ersatzschaltkreises auszuwerten, obwohl dies bereits sieben sind; für komplette Zellen ist dies wohl ein hoffnungsloses Unterfangen, da hier zwei Elektroden und ihre Filme neben Diffusionsphänomenen zu berücksichtigen sind. Aufgrund der sehr ähnlichen Zeitkonstanten können daher die einzelnen Prozesse nicht mehr separiert werden.
Obwohl es nicht sinnvoll ist, die in Abbildung 46 gezeigte Messung detailliert zu interpretieren, könnte in Anlehnung an das genannte Zitat [22] das Folgende angenommen werden: Aus dem ersten Halbkreis kann der Widerstand des Elektrolyten durch Extrapolation auf unendliche Frequenz abgeschätzt werden. Der folgende Halbkreis bei mittleren Frequenzen dürfte weitgehend von der SEI bedeckten Kohlenstoffelektrode bestimmt sein, der Anstieg der Messwerte hin zu niedrigen Frequenzen ist durch die Diffusion bestimmt. Da bei den in diesem Kapitel untersuchten Systeme eine deutliche Veränderung des Hochfrequenzwiderstandes beobachtet wird, siehe Abbildung 48, kann sicher dieser Widerstand nicht alleine durch den Elektrolytwiderstand bestimmt sein. An diesen Beispiel wird die Schwierigkeit der Separation einzelner Prozesse und Parameter deutlich.
In Abbildung 48 und Abbildung 49 ist der Verlauf der Deckschichtbildung mit zunehmender Zyklenzahl dargestellt. Die Spektren, die in Abbildung 48 gezeigt werden, wurden jeweils am Ende von Ladezyklen, die Spektren aus Abbildung 49 am Ende von Entladezyklen, aufgenommen. Aus ihnen kann auf die zunächst stark, dann langsamer wachsenden Deckschichten (insbesondere auf der Anode) geschlossen werden. Die Extrapolation des hochfrequenten Kreises auf unendliche Frequenzen zeigt aber, dass auch hier der Elektrolytwiderstand selbst von der Zahl der Zyklen und den zugegebenen Additiven abhängen würde. Das ist nicht sehr plausibel. Nach Cairns et al. [23] zeigen

$Li_xMn_2O_4$ Kathoden in $LiPF_6$/DMC/EC auch ein ähnliches wie das hier beobachtete Verhalten, insbesondere wenn sie nach Ende eines Ladezyklus aufgenommen wurden. Diese Autoren ordneten den ersten Halbkreis dem Film selbst zu. Auf dieser Basis sind unsere Daten besser zu verstehen, da klar ist, dass der Film und sein Widerstand wachsen, nicht aber, dass der Widerstand des Elektrolyten stark zunimmt. Der zweite Halbkreis und der Warburg-Anteil bei niedrigen Frequenzen wird von Cairns et al. [23] ebenso interpretiert wie das Verhalten an Li-Elektroden von Thevenin et al [22].

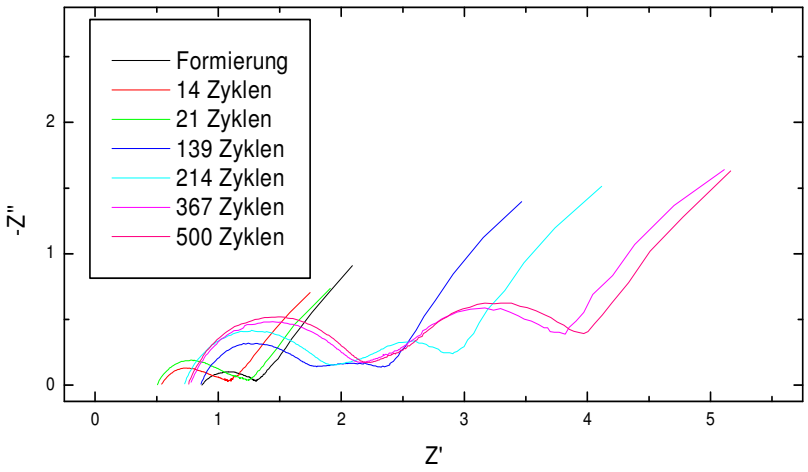


Abbildung 48 Zyklisierung der Batterie mit Lösung 1 ohne Additive, Impedanzspektren wurden am Ende des Ladevorganges aufgenommen

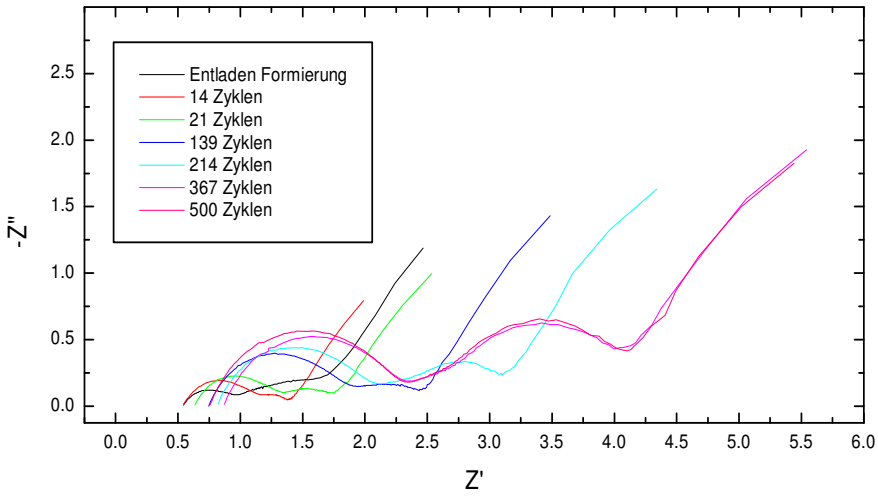


Abbildung 49 Zyklisierung der Batterie mit Lösung 1 ohne Additive, Impedanzspektren wurden am Ende des Entladeladevorganges aufgenommen

Auch in diesen Abbildungen kann sehr gut die Ausbildung einer Deckschicht mit zunehmender Zyklenzahl beobachtet werden [24].

6.6 Zusammenfassung

Das Ziel der hier beschriebenen Versuche war es, einen Elektrolyten mit LiBOB als Leitsalz für Lithium-Ionen-Batterien für den Einsatz bei hohen Temperaturen zu optimieren. Da sich der Einsatzbereich des Elektrolyten nach unten bis 0°C erstreckt, wurde zuerst eine Reihe von möglichen Mischungen getestet, ob sie diese Vorgabe erfüllen. Dabei wurden Mischungen eingesetzt, die sich einerseits nur aus Carbonaten, andererseits aus Carbonaten und Estern zusammensetzen. Da Batterien, die Mischungen enthalten, welche nur aus Carbonaten bestehen, über eine längere Lebensdauer verfügen, wurden diese Mischungen durch den Zusatz von Additiven weiter optimiert.
Mit dem in diesem Kapitel beschriebenen Messungen wurde erstmalig die Auswirkung von Additiven auf Lithium-Ionen-Batterien getestet, die LiBOB als Elektrolyt enthalten. Im Gegensatz zur Literatur [2] und [12] bis [20] wurden alle Additive unter den gleichen Bedingungen (Anode/Kathode, Zyklen, Temperatur) untersucht, so dass die einzelnen Additive untereinander verglichen werden können. Die hier gezeigten Messungen zeichnen sich gegenüber den in der Literatur beschriebenen Messungen dadurch aus, dass hier anstelle von Modellsystemen komplette Batterien untersucht wurden, die aus Materialien bestehen wie sie auch in der Produktion eingesetzt werden und bei denen die Zellgeometrie und das Verhältnis von Elektrodenmaterial und Elektrolyt mit dem von technischen Batterien übereinstimmt.
Einige der in der Literatur verwendeten Messzellen sind einfache Test-Zellen, bei denen die Elektroden in den Elektrolyt tauchen. Diese Zellen entsprechen im Prinzip den Messzellen, die für die cyclovoltammometrischen Messungen verwendet wurden, siehe Anhang 11.1.2. Da aber bei diesen Zellen das Verhältnis von Elektrodenmaterial zu Elektrolyt gänzlich anders ist als in Batterien, können die Messergebnisse beispielsweise aufgrund von Verdünnungseffekten nur schwer auf komplette Batterien übertragen werden.
Zum anderen werden in der Literatur Messungen an Knopfzellen beschrieben. Jene Zellen weisen diesen Nachteil nicht mehr auf, sie können aber wesentlich schwerer thermostatisiert werden. Daher kann mit diesen Zellen die Temperatur innerhalb der Batterien nur schwer kontrolliert werden. Durch den Einsatz der in 6.3 beschriebenen Zellen konnten diese Nachteile vermieden und Messungen durchgeführt werden, deren Ergebnisse auf kommerzielle Batteriesysteme übertragbar sind.
Durch einen Zusatz der gestesteten Additive kann die Lebensdauer der Batterie fast um den Faktor fünf gesteigert werden. Auch bei der Endkapazität wird eine enorme Steigerung von rund 100 % erreicht. Als bestes Additiv erwies sich Tetraethylenglykoldimethylether. Als überraschend erwies sich die Auswirkung des Zusatzes von Propionsäureethylester und Buttersäureethylester. Werden dieses Substanzen in kleinen Mengen (4 %) dem Batterieelektrolyten zugesetzt, so erreicht man signifikante Verbesserungen der Lebensdauer und Endkapazität. Im Gegensatz dazu weisen Elektrolyte, die diese Substanzen als Hauptlösungsmittelkomponente enthalten, deutlich schlechtere Eigenschaften auf als Batterien, die auf reinen Carbonatmischungen basieren.
Ein Ansatz für weitere Optimierungen wäre es daher, die Auswirkung der zugesetzten Additivmenge zu variieren. Es liegt nahe, dass dieser Ansatz auch bei Additiven, bei denen die Verbesserungen etwas geringer waren, noch deutliche Steigerungen bringen könnten, da die Konzentration für diese Additive nicht optimal eingestellt ist. Um diese Optimierung beschleunigen zu können, wäre auch hier der Einsatz der Simplexoptimierung, die in Kapitel 4.2.1.3 beschrieben wurde, hilfreich.

Ein weiterer Ansatz zur Verbesserung der Stabilität der Batterien wäre statt der Zugabe von nur einem Additiv der Zusatz mehrerer Additive. Beispielsweise wäre die Mischung aus Tetraethylenglykoldimethylether und Propionsäureethylester ein interessanter Ansatz. Es wäre ebenfalls denkbar, Mischungen aus drei oder mehr Additiven zuzusetzen. Um das beste Mischungsverhältnis und die optimale Konzentration dieser Additive zu finden, bietet sich wieder die Simplexoptimierung an.
Das beste Additiv, Tetraethylenglykoldimethylether, stammt aus der Klasse der Polyethylenoxide. Ein interessanter Ansatz wäre es auch, in dieser Verbindungsklasse nach Substanzen zu suchen, welche sich noch besser als Additive eigenen. Auch durch die Variation der Endgruppen dieser Substanzen könnten außerdem Verbesserungen erreicht werden. Ebenso wird durch den Zusatz von Estern eine Steigerung der Lebensdauer bewirkt. Bei diesen Verbindungen kann durch Variation des Carbonsäurerestes und der Alkoholgruppe eine große Anzahl an verschiedenen Verbindungen hergestellt werden. Es wäre daher von Interesse, die Auswirkung der Kettenlänge dieser beiden Gruppen auf die Lebensdauer der Batterie zu untersuchen.
Da, wie die Messungen in diesem Kapitel ergeben haben, die Stabilität der Elektrolyte, die Ester enthalten, etwas geringer ist, wäre es sehr interessant, auch diese durch Zusatz von Additiven zu verbessern, da diese Mischungen durch ihre höhere Leitfähigkeit bei tiefen Temperaturen und durch ihren größeren Flüssigkeitsbereich den Mischungen überlegen sind, die nur Carbonate als Lösungsmittelkomponenten enthalten. Insbesondere wäre eine Optimierung der Zyklisierstabilität der Elektrolytlösungen, welche für den Einsatz bei tiefen Temperaturen optimiert wurden, siehe Kapitel 4, von großem technischen Interesse.

6.7 Literaturverzeichnis

[1] G. Hambitzer, K. Pinkwart, C. Ripp, C. Schiller, in *Handbook of Battery Materials,* Chapter 1 (Ed: J. O. Besenhard), Wiley-VCH, Weinheim (1998)
[2] D. Linden, in D. Linden, Ed., *Handbook of Batteries*, Belfast (1994)
[3] E. Peled, *Journal of the Electrochemical Society,* **126,** 2047 (1979)
[4] E. Peled, D. Golodnitsky and J. Pencier, in *Handbook of Battery Materials,* Chapter 6 (Ed: J. O. Besenhard), Wiley-VCH, Weinheim (1998)
[5] D. Aurbach, in *Advances in Lithium-Ion Batteries,* Chapter 1 (Eds. W. A. van Schalkwijk und B. Scrosati), Kluwer Academic, New York (2002)
[6] E. Peled, D. Golodnitsky, in *LITHIUM-ION BATTERIES Solid-Electrolyte Interphase,* Chapter 1 (Eds. P. B. Balbuena und Y. Wang), Imperial College Press, London (2004)
[7] C. Menachem, D. Golodnitsky, E. Peled, *J. Solid State Chem.*, **5**, 81 (2001)
[8] Y. Ein-Eli, V.R Koch, *Journal of the Electrochemical Society,* **144,** 2968 (1997)
[9] J. A. Ritter, R. E. White, und B. N. Popov, *Proc. Electrochem. Soc,* PC 99-25, The Electrochemical Society Inc (2000)
[10] H. Honbo, S. Takeuchi, H. Momose, K. Nishimura, T. Horiba, Y. Muranaka und Y. Kozono, *Denki Kagaku,* **66,** 939 (1998)
[11] J. Suzuki, O. Omae, K. Sekine, und T. Takamura, *Solid State Ionics,* **152-153**, 111 (2002)
[12] D. Aurbach, J. S. Gnanaraj, W. Geissler, and M. Schmidt, *J. Electrochem. Soc.* 151, **A23** (2004)
[13] M. Herstedt, H. Rensmo, H. Siegbahn and K. Edstrom, *Electrochimica Acta*, **49**, 2351 (2004)
[14] R. Mogi, M. Inaba, S.-K. Jeong, Y. Iriyama, T. Abe, and Z. Ogumi, *J. Electrochem. Soc.* **149**, A1578 (2002)

[15] H. Ota, K. Shima, M. Ue and J.-i. Yamaki, *Electrochimica Acta*, **49**, 565 (2004)
[16] D. Aurbach, K. Gamolsky, B. Markovsky, Y. Gofer, M. Schmidt and U. . Heider, *Electrochimica Acta*, **47**, 1423 (2002)
[17] G. H. Wrodnigg, J. O. Besenhard, and M. Winter, *J. Electrochem. Soc.* **146**, 470 (1999)
[18] A. Naji, J. Ghanbaja, P. Willmann and D. . Billaud, *Electrochimica Acta*, **45**, 1893 (2000)
[19] M. D. Levi, E. Markevich, C. Wang, M. Koltypin, and D. Aurbach*, J. Electrochem.* Soc. *151*, A848 (2004)
[20] M.-S. Wu, J.-C. Lin, and P.-C. J. Chiang, *Electrochem. Solid-State Lett.* **7**, A206 (2004)
[21] D. Aurbach, in *Advances in Lithium-Ion Batteries,* (Ed: W. A. van Schalkwijk und B. Scrosati), Kluwer Academic, New York (2002)
[22] M. Garreau, J. Thevenin, B. Milandou, *Study by electrode impedance spectroscopy of the passivating layer of the lithium electrode in aprotic media*, Proceeding of the symposium on lithium batteries, A. N. Dey, Ed., PV 84-1, S. 28, The Electrochemical Society, Pennington (1984)
[23] K. Stribel, E. Sakai, E. Cairns, in *Rechargeable Lithium Batteries*, (Ed. M. Doyle, E. Takeuchi, K. M. Abraham), The Electrochemical Society Inc (2001)
[24] M. Doyle, E. Takeuchi und K. M. Abraham, *PV 2000-21, S. 596,* The Electrochemical Society, (2000)

7 Entwicklung neuer Methoden zur Analyse der Verunreinigungen von Lithium-bis[oxalato(2-)]borat(1-)

7.1 Zielsetzung

Die Leistungsfähigkeit von Batterien kann durch Verunreinigungen des Elektrolyten stark herabgesetzt werden. Insbesondere die Zyklenstabilität, die Stabilität bei hohen Temperaturen und die Langzeitstabilität werden durch Verunreinigungen des Elektrolyten negativ beeinflusst. Eine besonders kritische Verunreinigung ist Wasser. Durch Hydrolyse kann Wasser den Elektrolyten zersetzen. Beispielsweise entseht HF aus PF_6^-. Die dadurch entstehenden Hydrolyseprodukte wiederum können mit den Elektrodenmaterialien oder dem Lösungsmittel reagieren. Auch sind störende elektrochemische Reaktionen der Hydrolyseprodukte an den Elektroden denkbar, welche die Leistungsfähigkeit der Batterie herabsetzen. Darüber hinaus wird Wasser direkt an den Elektroden zu Wasser und Sauerstoff elektrolysiert. Um die Leistungsfähigkeit eines Elektrolyten richtig einschätzen zu können, ist es daher nötig, den Wassergehalt soweit wie möglich zu reinigen, so dass Verunreinigungen keine Rolle mehr spielen.

Umfangreiche Untersuchungen dieser Art wurden für das neuentwickelte Salz Lithium-bis[oxalato(2-)]borat(1-), (LiBOB) durchgeführt, über die in diesem Kapitel berichtet wird. Zuerst wurden die Verunreinigungen der verschiedenen zur Verfügung stehenden Lithiumbisoxalatoborat Chargen bestimmt und ihre Relevanz bezüglich der elektrochemischen Stabilität untersucht, siehe Abschnitt 11.2.3.1 und 4.4.

Da sich Wasser in LiBOB aufgrund von Nebenreaktionen von LiBOB mit den Karl-Fischer-Reagenzien nicht bestimmen lässt, ist es notwendig, eine neue Methode zur Wasserspurenbestimmung in LiBOB zu entwickeln, die diese Probleme nicht aufweist. Darüber wird in Abschnitt 11.2.3.2 berichtet.

Nach der Identifikation der Verunreinigungen ist es natürlich erforderlich, diese soweit zu reduzieren, so dass sie keine Rolle mehr spielen. Für diesen Zweck sollen mehrere Verfahren entwickelt werden und mit den neu entwickelten Analyseverfahren auf ihre Leistungsfähigkeit hin getestet werden, siehe Kapitel 7.4. Mit diesen Verfahren sollte dann noch eine für weitere Versuche ausreichende Menge reinen LiBOBs hergestellt werden, bei dem die elektrochemisch aktiven Verunreinigungen auf ein Mindestmass reduziert sind.

7.2 Bestimmung der Verunreinigungen von LiBOB

Gegenstand dieser Untersuchungen sind Verunreinigungen in verschiedenen LiBOB Chargen, die von den Firmen Chemetall (Frankfurt am Main) und Gaia Akkumulatorenwerke (Nordhausen) erhalten wurden. Die Charge SCHK01/29 ist laut Chemetall die reinste Charge, da es sich dabei um ein umkristallisiertes Produkt handelt. Die Charge SCHK01/42 stellt ein nicht umkristallisiertes Rohprodukt dar. Die Charge SCHK02/09 wurde nach einem neuen Verfahren hergestellt, das dem Autor nicht bekannt ist. Eine Kontamination des Salzes könnte durch die Lagerung und Versand verursacht worden sein. Es ist anzunehmen, dass für diese Art der Verunreinigung das Wasser aus

der Luftfeuchtigkeit verantwortlich ist. Am wahrscheinlichsten ist aber, dass das Salz schon bei der Herstellung verunreinigt wird, da bei der Herstellung Wasser als Lösungsmittel verwendet wird [1]. Zum einen können Nebenprodukte bei der Herstellung des Salzes entstehen oder es können auch nicht umgesetzte Edukte vorhanden sein. Zum anderen könnten Reste von Lösungsmitteln, die beim Trocknungsprozess nicht vollständig entfernt wurden, eine weitere Quelle an Verunreinigungen darstellen, die ihre Ursache in der Produktion des Salzes haben. Als Verunreinigungen, die ihre Ursache in der Herstellung haben, kommen folgende Substanzen in Betracht [2]:

- Monolithiumoxalat
- Dilithiumoxalat
- Lithiumglyoxylat
- Oxalsäure
- Glyoxylsäure
- Glykol
- Dimethylcarbonat
- Wasser
- Ethylacetat
- „PolyLiBOB“

Die Lithiumsalze der Oxalsäure, Glyoxylsäure beziehungsweise die freien Säuren kommen als Nebenprodukte oder im Fall von Oxalsäure als überschüssiges Produkt in Betracht. Glykol kann wie auch die Glyoxylsäure bei der reduktiven Herstellung [1] des LiBOB als mögliches Nebenprodukt entstehen. Dimethylcarbonat, Wasser und Ethylacetat können als Lösungsmittel bei der Produktion von LiBOB eingesetzt werden [1] und in Spuren im Salz zu finden sein. Bei der cyclovoltammometrischen Untersuchung von LiBOB Lösungen an Platin als Arbeitselektrode wurde nach Zusatz dieser Substanzen keine Veränderung beobachtet [2]. Daher wurde versucht, sie mittels NMR Spektroskopie in den LiBOB Chargen nachzuweisen.
In der Diplomarbeit des Autors [2] wird von einem nicht zuordenbaren Peak bei 3,45 ppm im ^{1}H–NMR berichtet, der in allen LiBOB Chargen auftritt, wenn deuteriertes Acetonitril (D-AN) als Lösungsmittel verwendet wird. Ebenso wird in dieser Arbeit von einer Reihe Peaks in Cyclovoltammogrammen an Platin berichtet, die Indizien für Verunreinigungen darstellen. Daher lag der Schwerpunkt der Untersuchungen auf der Klärung der Ursache für diese Verunreinigungen.
Als mögliche Ursachen für diesen Peak kämen Glykol beziehungsweise DMC in Betracht, die im Abschnitt 11.2.3.2.1 ausgeschlossen werden. In den Abschnitten 7.3 und 11.2.3.2.2 wird dieser Peak zweifelsfrei Wasser zugeordnet. Der Nachweis von Wasser mittels ^{1}H NMR Spektroskopie stützt zudem das Verhalten der LiBOB in den cyclovoltammographischen Messungen an Platin, mit denen auch Wasser nachgewiesen worden ist (Abschnitt 11.2.3.1, [2]).
Bei der Untersuchung der Verunreinigungen wurde versucht, diese mit mehreren Methoden zu bestimmen oder sie auszuschließen, um sichere Ergebnisse zu erzielen. Bei der Untersuchung zur Reinheit der verschiedenen LiBOB Chargen wurde hauptsächlich NMR Spektroskopie (^{1}H, ^{11}B und ^{13}C) Spektroskopie, siehe Anhang 11.1.4, eingesetzt. Neben der Charakterisierung der einzelnen Chargen, siehe Anhang 11.2.3.2.2, wurden die in diesem Abschnitt genannten Verunreinigungen durch Additionsexperimente als mögliche Kontamination in LiBOB untersucht, siehe Anhang 11.2.3.2.1. Zusätzlich zu diesen Messungen, wurde noch Anionen ESI Massenspektrometrie eingesetzt, die in Kapitel 11.1.7 beschrieben ist, um weitere Kontaminationen auszuschließen.

Der Autor stellte bereits in seiner Diplomarbeit [2] eine Vielzahl von elektrochemischen Messungen zur Bestimmung der Verunreinigungen dar. Nun wurden nur noch einige Experimente durchgeführt, mit denen bis jetzt nicht untersuchte Kontaminationen ausgeschlossen und bereits bestehende Ergebnisse abgesichert wurden. Eine detaillierte Beschreibung dieser Experimente ist in Anhang 11.2.3.1 aufgeführt.
Eine Reihe der untersuchten LiBOB Chargen weist einen schwer löslichen Rückstand auf. Dieser Feststoff wurde isoliert, siehe Anhang 11.2.3.3, und mittels ^{1}H-, ^{11}B-, und ^{13}C-NMR Spektroskopie und Röntgenpulverdiffraktometrie analysiert. Eine detaillierte Beschreibung der Versuche und der Ergebnisse ist in Anhang 11.2.3 aufgeführt.
Mittels ^{1}H-NMR Spektroskopie konnten die Lithiumsalze der Oxalsäure, Glyoxylsäure beziehungsweise die freien Säuren aus Verunreinigungen der verschiedenen LiBOB Chargen ausgeschlossen werden (Anhang 11.2.3.2.1). Ebenso konnte Dimethylcarbonat und Ethylenglykol als mögliche Ursache für den Peak im ^{1}H-NMR, der bei allen LiBOB Chargen auftritt, ausgeschlossen werden (Anhang 11.2.3.2.1). Dieser Peak konnte zweifelsfrei Wasser zugeordnet werden, wodurch die Hauptverunreinigung aller LiBOB Chargen identifiziert wurde. Die Beobachtung dieses Peaks legte die Grundlagen für eine neue Methode zur Wasserbestimmung in Substanzen, die nicht mittels Karl-Fischer-Titration bestimmt werden können, siehe Abschnitt 7.3. Alle vorhandenen LiBOB Chargen, die noch nicht in der Diplomarbeit des Autors [2] charakterisiert worden sind, wurden mittels ^{1}H-NMR Spektroskopie auf ihre Verunreinigungen hin untersucht, wobei Wasser als einzige wesentliche Verunreinigung identifiziert wurde, die aufgrund ihrer elektrochemischen Eigenschaften eine Rolle für den Einsatz als Leitsalz in Lithium-Ionen-Batterien spielt. Die Zusammensetzung des schwer löslichen Rückstandes konnte mit den zur Verfügung stehenden Methoden nicht geklärt werden.

7.3 Eine neue Methode zur Bestimmung von Wasserspuren bei denen die Karl-Fischer-Titration versagt

7.3.1 Einleitung

Wie in Abschnitt 7.1 geschildert, stellt der Wassergehalt von Elektrolytlösungen einen kritischen Parameter für die Leistungsfähigkeit von Batterieelektrolyten dar. Da LiBOB zumindest bei Normalbedingungen ausreichend hydrolysestabil ist und kein HF bilden kann, wäre dieses Salz anderen Leitsalzen überlegen. Trotzdem ist die Kenntnis des Wassergehalts von LiBOB von großer Bedeutung, da das eingeschleppte Wasser nicht nur mit den Elektrodenmaterialien reagieren kann, sondern auch unerwünschte Produkte mit dem Leitsalz selbst bildet. Wasser ist die Hauptverunreinigung aller LiBOB Chargen, wie in Abschnitt 7.2 gezeigt wird. Daher ist es für einen technischen Einsatz des Elektrolyten von enormer Wichtigkeit, den Wassergehalt zu kontrollieren und möglichst weit zu reduzieren.
Die Standardmethode zur Wassergehaltsbestimmung ist die Titration nach Karl Fischer. Aufgrund von Nebenreaktionen ist es nach Informationen von Chemetall aber ohne Modifikationen unmöglich, damit den Wassergehalt von LiBOB direkt zu bestimmen [8].
Ursachen hierfür könnten zum einen die Esterbildung aus Methanol, das als Lösungsmittel bei der Karl-Fischer-Titration eingesetzt wird, und zum anderen eine Reaktion mit Oxalsäure, welche aus LiBOB abgespalten wurde, sein. Das bei dieser Reaktion entstehende Wasser reagiert natürlich auch bei der Wasserbestimmung, wird mit titriert und verfälscht somit das Messergebnis. Scholz be-

schreibt in [3] genau dieses Verhalten bei der wasserfreien Oxalsäure. Bei Oxalsäure-2-hydrat tritt dieses Verhalten interessanterweise nicht auf [3] und [4].
Die Bildung von Ketalen ist eine weitere störende Nebenreaktion bei der Karl-Fischer-Titration, [5] wenn Aldehyde und Ketone mit dem Alkohol der Karl-Fischer-Lösung reagieren. LiBOB kann auch über einen analogen Mechanismus regieren und somit das Messergebnis verfälschen.
Wird während der Karl-Fischer-Titration der Verlauf des Reagenzverbrauchs, beziehungsweise der Strom bei der coulometrischen Titration der Verlauf des Stroms aufgezeichnet, so ist eine Korrektur des Fehlers, der durch die Nebenreaktion verursacht wird, möglich [3], [5]. Eine weitere Variante besteht darin, die Titration bei Temperaturen von unter 0°C durchzuführen, wodurch die Nebenreaktionen stark verlangsamt werden [3]. Wird das Methanol durch einen Alkohol ersetzt, bei dem eine langsamere Ketal- beziehungsweise Acetatbildung erfolgt, ist auch eine direkte Bestimmung des Wassergehalts von Aldehyden und Ketonen möglich [6]. Auch durch eine Esterbildung durch Borate oder Borsäure kann Wasser erzeugt und somit die Karl-Fischer-Titration gestört werden [7][32]. Auch diese Reaktion kann durch tiefe Temperaturen (-20 °C) so weit verlangsamt werden, dass sie nicht mehr stört [7].
Durch welche dieser Reaktionen die Wasserbestimmung bei LiBOB behindert wird, ist nicht bekannt. Es ist aber wahrscheinlich, dass eine Kombination der oben genannten Reaktionen zur Störung der Karl-Fischer-Titration führt.
Der Firma Chemetall ist es durch Kombination der oben aufgeführten Methoden gelungen, dennoch den Wassergehalt von LiBOB zu bestimmen [8]. Aufgrund des sehr geringen Wassergehalts des LiBOBs werden aber sehr hohe Anforderungen auf die Unterdrückung der Nebenreaktionen gestellt. Typischerweise werden in der Literatur, wenn störende Substanzen anwesend sind, Wassergehalte im Bereich von Prozenten bestimmt [3] bis [6]. Bei den untersuchten LiBOB-Chargen hingegen liegen die Wassergehalte im Bereich von einigen 100 ppm. Daher müssen bei diesen geringen Wassergehalten Nebenreaktionen weit mehr als bei den Literaturbeispielen unterdrückt werden. Sowohl bei der Tieftemperaturmethode, als auch bei der Aufzeichnung der Titrationskurven werden die störenden Nebenreaktionen nur unterdrückt, aber nicht ausgeschlossen; es besteht somit immer die Gefahr, dass zu viel Wasser nachgewiesen wird. Darüber hinaus besteht bei der Bestimmung der Wassergehalte bei tiefen Temperaturen die Möglichkeit der Wasserkontamination durch Kondenswasser aus der Luft, wenn die verwendete Apparatur nicht hermetisch abgeschlossen ist.
Aus diesen Gründen war es erforderlich, nach einer Methode zu suchen, mit der es möglich ist, den Wassergehalt von Salzen wie LiBOB ohne störende Nebenreaktionen zu bestimmen.

7.3.2 Theoretische Grundlagen

Aus einem Vergleich der Integrale der Peaks, die in einem NMR Spektrum vorkommen, lassen sich relative Gehalte bestimmen. Für eine Absolutbestimmung einer Konzentration ist aber die Zugabe einer genau bekannten Menge einer Standardsubstanz erforderlich. Durch einen Vergleich mit dem Integral des Standardpeaks ist es unter Berücksichtigung der Protonenverhältnisse beider Substanzen möglich, den Gehalt zu bestimmen. In der Literatur wird eine ganze Reihe von Beispielen für diese Gehaltsbestimmungen aufgeführt, vgl. z. B. [9], [10] und [11].
Andronov beschreibt in seinem Patent [12][33] eine Methode zur Bestimmung von Wasser in organischen Lösungsmitteln. Als Standard wird bei dieser Methode eine Mischung aus Trimethlychlorsi-

[32] An dieser Stelle sei Dr. S. Fernandez für die Übersetzung des tschechischen Artikels ins Deutsche gedankt.
[33] An dieser Stelle sei Dr. D. Zimin für die Übersetzung des russischen Patents ins Deutsche gedankt.

lan und Hexamethyldisiloxan eingesetzt. In einem Vorversuch wird zuerst mittels ^{1}H-NMR Spektroskopie das Verhältnis der Peaks der beiden Standards ermittelt. Wird zu einer Probe, die diese Mischung enthält, die Substanz hinzugefügt, deren Wassergehalt bestimmt werden soll, so reagiert das Wasser nach folgender Gleichung:

$$2\ (CH_3)_3SiCl + H_2O \rightarrow (CH_3)_3Si\text{-}O\text{-}\ Si\ (CH_3)_3 + 2\ HCl\ [12] \qquad (29)$$

Nach dieser Reaktion wird erneut ein ^{1}H-NMR Spektrum aufgenommen. Der Vorteil der Reaktion des Wassers mit dem Standardreagens liegt darin, dass statt des Wasser-Peaks ein Hexamethyldisiloxan-Peak untersucht wird. Da das Protonenverhältnis von Wasser zu Hexamethyldisiloxan 1:9 beträgt, wird dadurch eine Verstärkung des Signals um den Faktor neun erreicht [12]. Im Gegensatz zum Wasser-Peak, der durch Austauschreaktionen stark verbreitert ist, ergibt sich bei Hexamethyldisiloxan ein gut definierter Peak [12]. Durch die Verwendung von zwei Standards, die sowohl Produkt wie auch Edukt der Reaktion sind, wird eine Verbesserung des Signal-Rauschverhältnisses möglich [12]. In Tetrachlorkohlenstoff weist Andronov mit dieser Methode Wassergehalte im Bereich von 40 ppm nach. Der Nachteil dieser Methode ist vor allem in der Verwendung des Trimethlychlorsilans zu sehen, das nicht eingesetzt werden kann, wenn der Wassergehalt von Substanzen untersucht werden soll, die mit diesem Reagenz reagieren können.
Bei der in dieser Arbeit geschilderten Methode wird eine inerte Substanz als Standard eingesetzt, so dass alle denkbaren störenden Reaktionen von vornherein ausgeschlossen sind.
Diese Standardzugabe ist notwendig, da das LiBOB über keine Protonen verfügt und daher im ^{1}H-NMR Spektrum „unsichtbar“ ist, und somit nicht zur Gehaltsbestimmung herangezogen werden kann. Eine Kalibrierung auf den nicht-deuterierten Anteil des NMR Lösungsmittels, meist Deuteroacetonitril (D-AN), ist nicht nur aufgrund des nicht exakt bekannten Deuterierungsgrades ungünstig, sondern wird durch die Verwendung von Acetonitril als Lösungsmittel bei Umkristallisationen des LiBOBs unmöglich gemacht. Da LiBOB mit Lösungsmittelmolekülen stabile Addukte (Solvate) bildet, die sich nur schwer vollständig entfernen lassen, enthalten die untersuchten Proben immer Lösungsmittelrückstände im Spurenbereich, die eine sichere Kalibrierung mit Hilfe des NMR-Lösungsmittels unmöglich machen. Aus diesem Grund muss ein Standard zugesetzt werden. Es empfiehlt sich, dass dieser Standard nach Möglichkeit nur einen Peak aufweist und dass er eine andere chemische Verschiebung besitzt als die Verunreinigungen. Darüber hinaus sollte dieser Standard keine Reaktionen mit LiBOB beziehungsweise den Verunreinigungen eingehen und leicht dosierbar sein. Als Referenzsubstanz wurde Ethylencarbonat (EC) ausgewählt, das in Acetonitril einen einzigen Peak mit einer chemischen Verschiebung von 4,47 ppm zeigt.
Sollen mittels ^{1}H NMR-Spektroskopie Wassergehalte in Substanzen im Bereich von 100 ppm nachgewiesen werden, so stellt dies enorme Anforderungen an den dynamischen Bereich des NMR-Spektrometers. Durch die begrenzte Löslichkeit der Salze im NMR-Lösungsmittel wird dieses Problem noch verschärft. Bei typischen Messungen wurden hier 100 mg Salz in 1 g Lösungsmittel gelöst. Bei einem Wassergehalt des Salzes von 100 ppm liegt jetzt nur noch ein Wassergehalt von 10 ppm in der Lösung vor, der nachgewiesen werden soll. Werden aber Salze untersucht, die wie das LiBOB oder die meisten Batterieelektrolyten über keine Protonen verfügen, so stellt der dynamische Bereich kein großes Problem mehr dar. Bei dem in diesem Bericht durchgeführten Versuch stammt der größte Peak vom nicht deuterierten Anteil des Deuteroacetonitrils. Dank der Puls-NMR-Spektroskopie und der damit verbundenen Möglichkeit der Integration über lange Messzeit wird der dynamische Bereich des Spektrometers stark erweitert. Aus dem genannten Grund und,

um ein möglichst gutes Signalrauschverhältnis und zu erreichen, ist es nötig, die Spektren mit einer möglichst langen Messzeit (etwa 30 min) aufzunehmen.
Für die molale Konzentration an Wasser, b_{H2O}, gilt folgender Zusammenhang:

$$b_{H_2O} = 2 \cdot INT \cdot b_{EC2} \tag{30}$$

INT ist das Verhältnis des Integrals des Wasserpeaks zum Standardpeak. Der Multiplikator 2 ergibt sich aus dem Verhältnis der Anzahl der Protonen in einem Molekül EC zu der Protonenzahl in einem Molekül Wasser. b_{H2O} ist die molale Konzentration an EC.
Soll die Menge an Verunreinigung an anderen Substanzen bestimmt werden, ist es günstig, das Integral über alle Protonen dieser Substanz zu ermitteln und dann mit dem Verhältnis Protonenzahl in EC zur Protonenzahl der Verunreinigung zu multiplizieren.
Für den Multiplikator X gilt dann

$$X = \frac{x_1}{y_1} \tag{31}$$

wenn das Molekül $H_{x1}A_{x2}B_{x3}$ als Standard eingesetzt wird und das Molekül $H_{y1}A_{y2}B_{y3}$ nachgewiesen wird.
Da die eingesetzte Lösungsmittelmenge, m_{AN}, bekannt ist, kann aus der Molalität leicht die Stoffmenge an Wasser im NMR-Rohr und somit seine Masse bestimmt werden. Da auch die Masse des Salzes, m_{Salz}, bekannt ist, gilt für seinen Wassergehalt als Massenverhältnis:

$$w_{Salz} = \frac{2 \cdot INT \cdot b_{EC2} \cdot m_{AN} \cdot M_{H_2O}}{m_{Salz}} \tag{32}$$

Durch Multiplikation mit 10^6 kann daraus der Wassergehalt in ppm bestimmt werden.

7.3.3 Experimentelle Durchführung

7.3.3.1 Verwendete Geräte und Chemikalien

Für alle Messungen wurde das in Abschnitt 11.1.4 beschriebene NMR-Spektrometer eingesetzt und mit dem in Abschnitt 11.1.4.1 beschriebenen Messprogramm durchgeführt.
In dieser Arbeit wurde Lithium-bis[oxalato(2-)]borat(1-) untersucht, das im Folgenden mit LiBOB abgekürzt wird. Von dieser Substanz wurden vom Hersteller Chemetall (Frankfurt) drei verschiedene Chargen bezogen, siehe auch Abschnitt 7.2.
Um eine Kontamination der Lösungen mit Wasser zu vermeiden, wurden alle Operationen, bis auf die NMR-Messungen selbst, in dem in Abschnitt 11.1.1.1 beschriebenen Handschuhkasten durchgeführt. Massen wurden mit einer Mettler Toledo (Greifensee) AB 204 Analysenwaage bestimmt, die sich im Handschuhkasten befindet. Aufgrund der Druckschwankungen innerhalb des Handschuhkastens wird der Messfehler, der werkseitig mit 0,2 mg angegeben ist, vergrößert. Er wird auf 1 mg abgeschätzt. Die eingesetzten Laborgeräte wurden vor dem Einsatz mindesten für 24 Stunden bei 140°C im Trockenschrank beziehungsweise bei 65 °C im Vakuumtrockenschrank bei 10^{-1} mbar getrocknet.

Als Lösungsmittel wurden Deuteroacetonitril der Firmen Deutero GmbH (Kastellaun) und Eurisо-Top SA (Saint-Aubin) mit einem Deuterierungsgrad von 99,5 %, beziehungsweise von 99,8 % eingesetzt. Der Wassergehalt des Lösungsmittels der Firma Deutero, der mit Karl-Fischer-Titration bestimmt wurde, betrug je nach Charge zwischen 100 und 800 ppm. Der Wassergehalt des Lösungsmittels von Euriso-Top wird vom Hersteller mit kleiner als 500 ppm angegeben. Die verwendete Charge hatte einen Gehalt von 97 ppm, der mittels Karl-Fischer-Titration bestimmt wurde. Da der Wassergehalt der verwendeten deuterierten Acetonitril Chargen eine genaue Bestimmung des Wassergehaltes des LiBOBs unmöglich machen würde, ist es nötig, das Lösungsmittel hinreichend stark zu trocknen. Eine rechnerische Korrektur durch Bestimmung des Wassergehaltes des Lösungsmittels mit Karl-Fischer-Titration ist aufgrund der geringen Wassergehalte und der damit verbunden Messunsicherheit ungünstig. Daher wurde das Lösungsmittel so weit wie möglich getrocknet. Eine einfache Methode hierfür ist das Trocknen über 4Å-Molsieb [13] und [14], welches mit der in Anhang 11.1.8 beschriebenen Hochvakuumanlage bei 300 °C getrocknet wurde. Vor der Verwendung des Lösungsmittels wurde die Lösung vom Molsieb abdekantiert. Die Wassergehalte der Lösungsmittel wurden mit einer Karl-Fischer-Apparatur der Firma Mitsubishi Chemical Ind. Ltd. (Tokio) des Typs Moisturemeter CA-02 bestimmt, siehe Anhang 11.1.6.
Um die Leistungsfähigkeit der NMR Wasserbestimmung zu dokumentieren, wurden Standardadditionen an $LiClO_4$ und $LiPF_6$ als Salz durchgeführt. Beide stammen von der Firma Merck (Darmstadt) und lagen in der Reinheitsstufe *Suprapur®* vor. Das Ethylencarbonat (EC) und das Dimethylcarbonat (DMC) wurden in der Qualität Selectipur der Firma Merck (Darmstadt) eingesetzt. Das verwendete wasserfreie Ethylenglykol stammt ebenfalls von Merck (Darmstadt), war aber nur in der Qualität p.a..
Um durch Vergleich der Peakflächen im NMR Spektrum die tatsächliche Konzentration bestimmen zu können, ist es erforderlich, der zu vermessenden Lösung einen Standard mit bekannter Konzentration hinzuzusetzen. Als Standard wurde Ethylencarbonat (EC) ausgewählt, da dieses nur einen Peak mit einer günstigen chemischen Verschiebung von 4,59 ppm aufweist [15]. Da EC eine typische Lösungsmittelkomponente von Batterieelektrolyten ist, kann davon ausgegangen werden, dass keine Reaktionen mit dem LiBOB stattfinden. Darüber hinaus steht es mit einem Wassergehalt von weniger als 30 ppm in einer ausreichend trockenen Qualität zur Verfügung, so dass Dank der notwendigen hohen Verdünnungen keine Verfälschung des Messergebnisses durch den Wassergehalt des Ethylencarbonats verursacht wird, da dieser dann nur noch verschwindend klein ist.
Um den dynamischen Bereich des NMR Geräts nicht einzuschränken und um eine einfache Auswertung zu gewährleisten, ist es sinnvoll, die molale Konzentration des Standards so zu wählen, dass sie sich in der Größenordung der zu bestimmenden Wasserkonzentration beläuft.
Um eine Einwaage einer sehr kleinen Menge Ethylencarbonat zu vermeiden, wurde zuerst eine Stammlösung von Ethylencarbonat in getrocknetem, deuterierten Acetonitril, hergestellt, deren Molalität b_{EC1} mit folgender Formel berechnet wird:

$$b_{EC1} = \frac{m_{EC}}{M_{EC} \cdot m_{AN}} \qquad (33)^{34}$$

In einem zweiten Verdünnungsschritt wurden m_1 g der Stammlösung mit der Konzentration b_{EC1} zu m_{AN} g getrockneten D-AN gegeben. Die Molalität, b_{EC2}, dieser Lösung lässt sich nicht mit For-

[34] Üblicherweise wird die molale Konzentration mit m bezeichnet. Da sie dann aber leicht mit der Masse m verwechselt wird, ist auch b nach der IUPAC zugelassen.

mel (33) berechnen, da auch der Lösungsmittelanteil der zugegebenen Stammlösung berücksichtigt werden muss. So ergibt sich folgender Zusammenhang für die Molalität der zweiten Lösung:

$$b_{EC2} = \frac{b_{EC1} \cdot m_1}{m_{AN} + (m_1 - b_{EC1} \cdot m_1 / M_{EC})} \tag{34}$$

Für die in den Abschnitten 11.2.5.7 bis 11.2.5.9 beschriebenen Messungen wird eine Lösung mit genau definiertem Wassergehalt benötigt. In Analogie zu der Herstellung des EC Standards wurde auch hier eine Verdünnung über eine Stammlösung verwendet.

7.3.3.2 NMR-Messung und Auswertung

Für die Messung wurden ca. 100 mg des Salzes in ein NMR-Rohr eingefüllt und die Masse des Salzes exakt bestimmt. Anschließend wurde eine genau abgewogene Menge (ca. 1g) der in Abschnitt 7.3.3.1 beschriebenen zweiten Standardlösung hinzugefügt. Da die LiBOB Chargen SCHK01/47 und SCHK02/09 einen in Acetonitril unlöslichen Anteil haben, wurde dieser abgetrennt. Hierfür wurde das verschlossene NMR-Rohr für ca. 1 min geschüttelt, bis sich der lösliche Anteil gelöst hatte. Über Nacht konnte sich dann der nichtlösliche Rückstand absetzen und die klare Lösung wurde mit einer Pasteurpipette abgezogen, in ein weiteres NMR-Röhrchen eingefüllt und vermessen.
Für die Auswertung wurden zuerst mit WIN-NMR die genauen chemischen Verschiebungen des Ethylencarbonatpeaks bei 4,4 ppm und des Wasserpeaks bei 3,4 ppm ermittelt. Zur Bestimmung der Integralflächen wurden die Integrationsfunktionen des Programms eingesetzt. Für beide Peaks wurde eine Korrektur des Bias und der Steigung durchgeführt. Das Integral des Ethylencarbonat-Peaks wurde dann auf 1 normiert, so dass das Integral des Wasserpeaks INT direkt dem Verhältnis der Protonenmolalitäten von Wasser und Ethylencarbonat entspricht. Die Molalität des Wassers lässt sich mit Formel (32) bestimmen:

7.3.4 Ergebnisse

Nach einer Untersuchung des getrockneten Lösungsmittels, siehe Anhang 11.2.5.1, wurde der Wassergehalt von fünf verschiedenen LiBOB Chargen mittels der NMR Methode bestimmt, wobei von jeder dieser Chargen mehrere Proben vermessen wurden. Eine detaillierte Beschreibung dieser Messungen ist in Anhang 11.2.5 aufgeführt. In Tabelle 22 ist der gefundene Wassergehalt w_{Salz}, sein geschätzter Fehler Δw_{Salz}, siehe Anhang 11.3.4.1, und der Wassergehalt w_{KF}, der mittels Karl-Fischer-Titration nach der Methode von Panitz [8] bestimmt wurde aufgeführt.

Charge	w_{Salz} / ppm	Δw_{Salz} / ppm	w_{KF} / ppm
SCHK01/29	120	12	250
SCHK01/47	342	29	2200
SCHK02/09 alt	296	28	650
SCHK02/09 Gaia	440	31	650
SCHK02/09 Chemet.	262	21	650

Tabelle 22 Wassergehalte der einzelnen LiBOB Chargen

Mit der NMR Methode wird im Vergleich zur Karl Fischer Titration deutlich weniger Wasser gefunden. Dies ist ein Hinweis dafür, dass Nebenreaktionen ablaufen, die falsch positive Ergebnisse liefern, obwohl versucht wurde diese Reaktionen zu unterdrücken. Bei der Charge SCHK02/09 Gaia wurde eine starke Streuung der Messergebnisse beobachtet. Die Standardabweichung dieser Messungen liegt bei 160 ppm.
Um eine Aussage über die Genauigkeit und Wasserbestimmung mit NMR treffen zu können, wurde eine Messreihe durchgeführt, bei der genau definierte Wassermengen zugegeben wurden. Da hierfür sehr kleine Wassermengen benötigt werden, wurden die in Abschnitt 7.3.3.1 beschriebenen Wasserstandardlösungen eingesetzt. Ziel war, einen Bereich von 100 bis 3000ppm, der dem typischen Bereich der Wasserkonzentrationen der Salze entspricht, zu überdecken.
Als Salz wurde LiBOB der Charge SCHK01/29 mit einem Wassergehalt von (120 ± 12) ppm, der in Abschnitt 11.2.5.2 bestimmt wurde, eingesetzt.
Der Wassergehalt in der Lösung, die vermessen wird, setzt sich aus der zugegebenen Menge des Wasserstandards m_{H2O} mit seiner molalen Konzentration an Wasser $b_{H2O,S}$ und dem Wasseranteil des Salzes w_{Salz} mit der Masse m_{Salr} des Salzes zusammen. Der Wassergehalt des Lösungsmittels kann vernachlässigt werden, da er durch die in Abschnitt 7.3.3.1 dargestellte Trocknung unter die Nachweisgrenze reduziert wurde. Da mit der hier beschriebenen Methode aber Wassergehalte in Salzen bestimmt werden, ist es sinnvoll, den gesamten Wassergehalt der Lösung auf das Salz zu beziehen:

$$w_{rec} = \frac{\left(b_{H_2O,S} \cdot m_{H_2O} + \frac{m_{Salz} w_{Salz}}{M_{H_2O}} \right) M_{H_2O}}{m_{Salz}} \tag{35}$$

Wird w_{rec} mit 10^6 multipliziert, so ergibt sich der Wassergehalt in ppm. Der Fehler des Sollwassergehaltes des Salzes wird mit Gleichung (64), siehe Anhang 11.3.4.1 bestimmt.
Bei der Durchführung der Standardaddition wurde für jeden Wassergehalt eine eigene Lösung angesetzt. So wurde ausgehend vom EC-Standard mit einer molalen Konzentration von (13,6 ± 0,2) mmol/kg an EC, bei dem das gesamte Wasser aus dem Salz stammt, durch Erhöhung der zugegebenen Menge an Wasser-Standard der Wasseranteil schrittweise erhöht. Im Idealfall sollte der mit NMR ermittelte Wasseranteil mit dem Sollwasseranteil übereinstimmen. Trägt man den mit NMR ermittelten Wassergehalt gegen den Sollwassergehalt auf, so ergibt sich im Idealfall eine Winkelhalbierende durch den ersten Quadranten. In Abbildung 50 sind die mit NMR bestimmten Wassergehalte gegen die vorgegebenen Wassergehalte aufgetragen; die Ergebnisse der einzelnen Messungen sind im Anhang 11.2.5.7 aufgeführt.
.

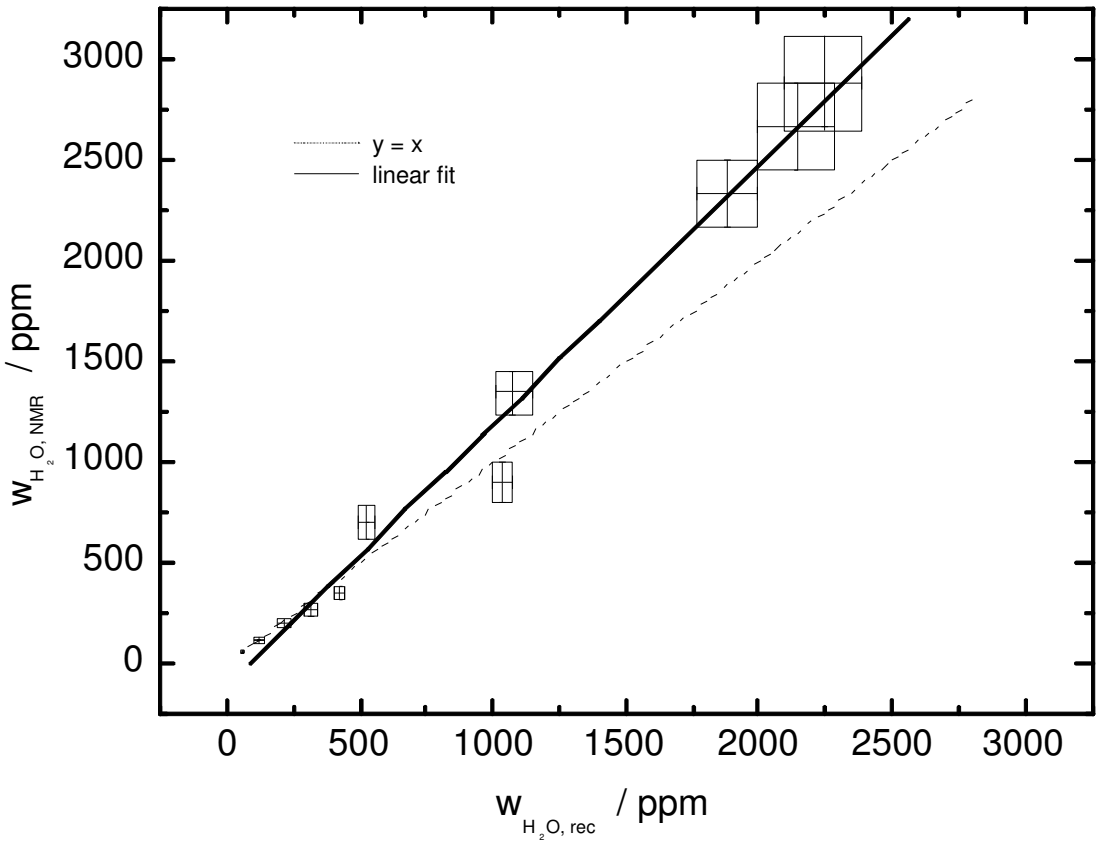


Abbildung 50 Standardaddition LiBOB

Bei der hier durchgeführten Standardaddition wird mehr Wasser gefunden als vorgegeben wurde. Durch lineare Interpolation mit der Methode der kleinsten Fehlerquadrate wird folgender Zusammenhang zwischen dem vorgegebenen, $w_{H20,rec}$, und dem gefundenen Wassergehalt, $w_{H2O,NMR}$, ermittelt:

$$w_{H_2O,\,NMR} = -93\text{ppm} + 1{,}28\; w_{H_2O,\,rec} \tag{36}$$

Für die Abweichung von der Idealgeraden wird somit ein Wert von 28% erhalten. Ursachen für diese Abweichungen könnten in den Kontaminationen der Glasgeräte oder in der Diffusion von Wasser aus der Umgebungsluft durch die Kappen der NMR-Rohre liegen.
Um die Leistungsfähigkeit der Wasserbestimmung auch an anderen Salzen zu untersuchen, wurden Standardadditionen von Wasser mit $LiPF_6$ und $LiClO_4$ als Salz durchgeführt. Die Ergebnisse dieser Messungen sind in den Anhängen 11.2.5.8 und 11.2.5.9 aufgeführt. Bei diesen Messungen für die Wasserbestimmung in $LiClO_4$ wurde eine Abweichung von 6 % gefunden. Bei $LiPF_6$ beträgt sie sogar nur 4 %. Bei der Wasserbestimmung an $LiPF_6$ wurde das Auftreten eines zweiten Peaks beobachtet, der HF als Hydrolyseprodukt des Salzes zugeschrieben wird. Bei der Auswertung dieser Messungen wurde nur dann der Sollwassergehalt gefunden, wenn auch das Integral dieses Peaks bei der Berechung der Gesamtwassermenge berücksichtigt wurde. Das Auftreten dieses Peaks kann für die Untersuchung von Hydrolysekinetiken von $LiPF_6$ und anderen Leitsalzen verwendet werden, wenn zu definierten Zeitpunkten das Verhältnis von Wasser und HF Peak bestimmt wird.

Die statistische Streuung der Messungen wurde an einer Probe gemessen, die zehnmal hintereinander vermessen worden ist Die Abweichung dieser Messungen liegt bei 5 %, siehe 11.3.4.3. Dieser Wert geht als Fehler der Integralbestimmung in den Gesamtfehler ein. Dieser liegt wie die Fehlerabschätzung in Anhang 11.3.4.1 zeigt, in der Größenordung von 10 %.
Die Erfassungsgrenze der Messmethode liegt bei einer Wasserkonzentration von 6 ppm, siehe Abschnitt 11.3.4.2. Diese wird nicht nur durch das NMR-Spektrometer sondern vor allem auch durch die Kontamination der Lösungen durch Wasser, welches während der Messung durch die Verschlusskappen der NMR Rohre diffundiert limitiert. Wie in Anhang 11.3.4.4 gezeigt wird, kann durch spezielle NMR Kappen dieser Fehler soweit reduziert werden, so dass er keine Rolle mehr spielt, wenn die Messungen sofort durchgeführt werden, nachdem die Proben aus dem Handschuhkasten genommen wurden.

7.4 Reinigung von LiBOB

Wasser wurde in Abschnitt 7.2 als Hauptverunreinigung der LiBOB Chargen identifiziert, siehe Abschnitt. Zur Entfernung dieser Verunreinigung, wurde Lithium-bis(oxalatoborat), LiBOB, nach folgenden Methoden getrocknet

- Chromatographie
- Umkristallisation
- Trocknung im Hochvakuum
- Trocknung über P_2O_5
- Wasserextraktion mit Molsieb

Ziel dieser Versuche war nicht nur möglichst trockenes LiBOB zu erhalten, sondern insbesondere die Trocknungsmethoden selbst zu untersuchen. Im Anhang 11.2.4 ist die Durchführung dieser Methoden im Detail beschrieben. Zur Trocknung haben sich dabei die folgenden Methoden als geeignet erwiesen, wie durch NMR-Spektroskopie nachgewiesen wurde:

- Trocknung im Hochvakuum
- Trocknung über P_2O_5
- Wasserextraktion mit Molsieb

Nicht geeignet sind, da der Wassergehalt nicht genügend gesenkt wird:

- Chromatographie
- Umkristallisation

Die Trocknung im Hochvakuum ist die Methode der Wahl, um Wasserspuren aus LiBOB zu entfernen, da hier eine Kontamination durch Trocknungsmittel ausgeschlossen werden kann. Wird bei diesem Verfahren die Temperatur nur langsam gesteigert, und nicht wie in [2] beschrieben, sofort mit maximaler Temperatur getrocknet, so wird auch keine Zersetzung des LiBOBs beobachtet. Die Wasserextraktion mit Molsieb stellt durch die günstige Kombinationsmöglichkeit mit der Umkristallisation ein Verfahren dar, mit dem es zumindest im Labormaßstab möglich ist, sowohl LiBOB mit wenig Aufwand zu trocknen wie auch vom unlöslichen Rückstand zu reinigen.

Die Methode Umkristallisation ist, wie auch die der Chromatographie, aufgrund der starken Neigung von LiBOB Addukte mit Wasser zu bilden, ungeeignet, Wasser aus LiBOB zu entfernen. Die Umkristallisation ist aber eine Methode zur Beseitigung von weiteren Verunreinigungen.
Da die Chargen SCHK01/47 und SCHK02/09, siehe Abschnitt 11.2.3.3, mit einem schwer löslichen Rückstand verunreinigt sind, wurde auch eine Methode entwickelt, mit der dieser Rückstand entfernt und für die in Abschnitt 11.2.3.3 beschriebene Charakterisierung isoliert werden kann. Mit dieser Methode konnte für eine erste Charakterisierung, siehe Abschnitt 11.2.3.3, eine ausreichende Menge Rückstand isoliert und für weitere Experimente ausreichende Mengen reinen LiBOBs hergestellt werden.

7.5 Zusammenfassung

Mittels einer Kombination von NMR Messungen und elektrochemischen Untersuchungsmethoden konnte die Hauptverunreinigung von LiBOB als Wasser identifiziert werden, siehe 7.2. Diese Verunreinigung konnte sowohl in Cyclovoltammetrisch wie auch im ^{1}H-NMR Spektrum beobachtet werden. Durch Additionsexperimente konnte Wasser dann zweifelsfrei bestimmt werden. Mit diesen Messungen konnten zusätzlich eine Reihe von potenziellen Verunreinigungen ausgeschlossen werden.
Aufgrund der Beobachtung des Wasserpeaks im ^{1}H-NMR konnte eine neuartige Methode zur Spurenbestimmung von Wasser in Salzen entwickelt werden, siehe 7.3. Mit dieser Methode[35] ist es nun zum ersten Mal möglich, den Wassergehalt von LiBOB ohne Nebenreaktionen zu bestimmen, die ein fälschlicherweise positives Ergebnis liefern. Durch den Einsatz der NMR Spektroskopie konnte der Wassergehalt von fünf verschiedenen LiBOB Chargen bestimmt werden. Im Gegensatz zu den von Chemetall mit der Karl-Fischer-Titration ermittelten Werten wird hier nur circa die Hälfte an Wasser gefunden. Dieses Ergebnis spricht dafür, dass bei der von Chemetall durchgeführten Karl-Fischer-Titration doch noch Nebenreaktionen stattfanden. Es liegt daher nahe anzunehmen, dass die in der Literatur beschriebenen Methoden zur Unterdrückung dieser Nebenreaktionen, die zur Untersuchung für Wassergehalte im Prozentbereich gut funktionieren, bei der Untersuchung von Wassergehalten im ppm Bereich nicht ausreichend sind.
Im Gegensatz zu der im russischen Patent [12] dargestellten Methode wird hier ein unreaktiver Standard eingesetzt, so dass Nebenreaktionen sicher ausgeschlossen sind. Dadurch wird erstmals eine Wassergehaltsbestimmung von äußerst reaktiven Substanzen möglich.
Anhand von Standardadditionen wurde die Leistungsfähigkeit der Wassergehaltsbestimmung mit ^{1}H-NMR Spektroskopie nachgewiesen. Neben einer Standardadditionsreihe an LiBOB wurden noch Standardadditionsreihen an $LiClO_4$ und $LiPF_6$ durchgeführt, so dass auch die Leistungsfähigkeit der NMR Methode an anderen Salzen demonstriert werden konnte. Alle Messungen wurden einer eingehenden Fehlerrechnung unterzogen, so dass die erhaltenen Ergebnisse auch von dieser Seite her abgesichert wurden.
Bei der Standardaddition an $LiPF_6$ konnte die Hydrolyse des Salzes bei geringen Wassergehalten untersucht werden. Es ergibt sich somit eine elegante Möglichkeit zur Untersuchung der Kinetik der Hydrolyse von Salzen, wenn die NMR Experimente zu definierten Zeitpunkten durchgeführt werden.

[35] Diese Arbeiten wurden freundlicherweise von Chemetall, Frankfurt unterstützt. Für die Informationen, neuen Proben des Lithiumsalzes LiBOB und die finanzielle Unterstützung durch die Firma Chemetall dankt der Verfasser den Herren Dr. U. Wietelmann und Dr. J.-C. Panitz.

Die hier beschriebene Methode eignet sich nicht nur zur Wassergehaltsbestimmung. Es ist natürlich eine quantitative Bestimmung aller protonenhaltiger Substanzen möglich. Dies wurde hier am Beispiel von Verunreinigungen wie Phtalsäureestern und Essigsäureethylester demonstriert.
Anhand dieser Wassergehaltsbestimmungen konnten auch eine Reihe von Trocknungsmethoden auf ihre Leistungsfähigkeit hin untersucht werden, siehe Abschnitt 7.4. Hierbei ist besonders die Trocknung mit Molsieb im Flüssig-Flüssigexstraktor hervorzuheben, mit der in einem Vorgang sowohl der nicht lösliche Anteil abgetrennt, der Wassergehalt reduziert und das Salz umkristallisiert werden kann (Abschnitt 11.2.4.5).

7.6 Literaturverzeichnis

[1] U. Lischka, U. Wietelmann and M. Wegner, *Lithium bis(oxalato)borate, method for its production and application*, DE 19829030, (1999)
[2] H.-G. Schweiger, *Entwicklung einer Präzisionstemperaturmessanlage zur schnellen Messung von Phasendiagrammen und chemische und elektrochemische Charakterisierung von Lithium-bis[oxalato(2-)]borat(1-)*, *Diplomarbeit*, Regensburg (2002)
[3] E. Scholz, *Fresenius' Zeitschrift für Analytische Chemie*, **310**, 423 (1982)
[4] U. Ramin und H. Wollmann, *Pharmazie*, **45**, 840 (1990)
[5] E. Scholz, *Analytical Chemistry*, **57**, 2965 (1985)
[6] Sigma AldrichLaborchemikalien GmbH, *HYDRANAL®-Laboratory Report 385*
[7] D. Prochazkova, *CHEMagazin*, **12**, 29 (2002)
[8] J.-C. Panitz, *persönliche Mitteilung*, (2002)
[9] I. Kosir, M. Kocjancic und J. Kidric, *Analusis*, **26**, 97 (1998)
[10] Z. Xia, L. G. Akim und D. S. Argyropoulos, *Journal of Agricultural and Food Chemistry*, **49**, 3573 (2001)
[11] I. Berregi, J. I. Santos, G. Del Campo, J. I. Miranda und J. M. Aizpurua, *Analytica Chimica Acta*, **486**, 269 (2003)
[12] V. F. Andronov, *Determination of traces of moisture*, SU 834474, (1981)
[13] W. L. Armarego und D. Perrin, *Purification of Laboratory Chemicals*, Oxford (1988)
[14] J. B. Alper, *Analytical Chemistry*, **50**, 381 (1978)

8 Was ist neu in dieser Arbeit?

Diese Dissertation unterscheidet sich deutlich von den bisher von Mitgliedern unserer Arbeitsgruppe vorgelegten Doktorarbeiten. Zum einen handelt es sich im vorliegenden Fall um eine eher technisch orientierte Experimentalarbeit, in deren Rahmen mehrere Messgeräte aufgebaut und getestet wurden. Zum anderen wird hier nicht eine Arbeit vorgelegt, bei der, wie häufig üblich, eine Vielzahl von Messdaten mit bereits existierenden Messplätzen aufgenommen und ausgewertet wurden. Vielmehr wurden in dieser Arbeit neue Geräte und Methoden entwickelt, mit denen es möglich ist, umfangreiche Mess- und Optimierungsaufgaben in kurzer Zeit zu lösen. Durch deren Einsatz wurde eine Reihe von Problemen gelöst, die anfallen, wenn ein neues Leitsalz in Lithium-Ionen-Batterien eingesetzt werden soll.
Die entwickelten Geräte und Methoden wurden für die Bearbeitung von technischen Fragestellungen bei der Entwicklung neuer Elektrolyte für Lithium-Ionen-Batterien eingesetzt. Aufgrund des hohen Durchsatzes dieser Verfahren konnte eine Vielzahl von Problemen der Batterieoptimierung bearbeitet und gelöst werden. Dies wäre mit den bisher angewandten Methoden und Verfahren in derselben Zeit nicht möglich gewesen, so dass sich der Entwicklungsaufwand dieser Anlagen und Verfahren schon während dieser Arbeit mehr als ausgezahlt hat. Schon im Vorfeld der Geräteentwicklung wurden umfangreiche Fehlerrechnungen durchgeführt, um eine optimale Bauteilauswahl treffen zu können. Diese Rechnungen wurden, wenn nötig, noch durch zusätzliche Versuche zur Bestimmung der Genauigkeit ergänzt.
Die neuentwickelten Apparate wurden so gestaltet, dass sie auch für andere Verwendungszwecke eingesetzt oder leicht angepasst werden können. Alle Geräte zeichnen sich durch einen modularen Aufbau aus, wodurch nicht nur die Entwicklung dieser Geräte erleichtert wurde, sondern auch sehr einfach neue Anlagen zur Bearbeitung anderer Probleme mit diesen Modulen aufgebaut werden können. Was an den einzelnen Anlagen, Verfahren und den damit erzielten Ergebnissen neu ist, wird in der folgenden Aufzählung aufgeführt. Diese ermöglicht auch den schnellen Zugang zu den Höhepunkten der Arbeit.

Mit der in **Kapitel 2** beschriebenen Anlage ist es erstmalig möglich, Abkühlungs- und Aufheizkurven an einer großen Anzahl an Proben zu untersuchen. Durch diese große Probenzahl kann das Schmelzdiagramm einer binären Mischung innerhalb einer Woche bestimmt werden, wobei es sogar möglich ist, über eine große Anzahl an Wiederholungen der Messungen, eine statistische Aussage über die Fehlerverteilung zu erhalten. Simultan zur Messung der Temperatur der einzelnen Proben kann mit dieser Anlage auch die Leitfähigkeit der Proben gemessen werden. Diese Leitfähigkeitsmessung eröffnet nicht nur eine zweite Detektionsmöglichkeit für Phasenübergänge, sondern auch einen sehr breiten Anwendungsbereich für die Anlage, der über die Bestimmung von Flüssigkeitsbereichen weit hinaus geht. Beispielsweise können mit dieser Anlage die Temperaturabhängigkeit der Leitfähigkeit von Elektrolytlösungen oder auch Hydrolysekinetiken an einer großen Anzahl an Proben simultan untersucht werden. Um hohe Kosten, die eine große Anzahl an Messkanälen verursacht, wenn klassische Methoden verwendet werden, zu reduzieren und um die Messgeschwindigkeit zu erhöhen, war es erforderlich, neue Techniken einzusetzen. So konnten beispielsweise durch den Einsatz von Thermistoren die Kosten für die Temperaturmessung um den Faktor 100 gegenüber Platinthermometern gesenkt werden, ohne wesentliche Abstriche bei der Genauigkeit zu machen. Durch aufwändige Kalibrierung dieser Thermistoren konnte letztlich eine Genauigkeit erzielt werden, die der von Platinthermometern entspricht. Auch die Langzeitstabilität dieser Fühler unterscheidet sich nur wenig von Platinthermometern, wie an einer großen Anzahl an

Thermistoren nachgewiesen wurde. Mit diesen Arbeiten konnte ein weitverbreitetes Vorurteil, nämlich die mangelhafte Stabilität von Thermistoren, ausgeräumt werden. Auch bei der Leitfähigkeitsmessung konnte durch den Einsatz moderner Bausteine der Nachteil der langsamen Messdatenerfassung der klassischen Brückenschaltung überwunden werden.

In **Kapitel 3** wird anhand mehrerer Phasendiagramme und der Bestimmung der Festpunkte von Einzelsubstanzen die Leistungsfähigkeit der in **Kapitel 2** vorgestellten Anlage demonstriert. Bei diesen Messungen, die nach der herkömmlichen Methode ein Vielfaches der Zeit benötigt hätten, als für eine Doktorarbeit zur Verfügung steht, konnten zum ersten Mal sehr genaue Messungen von Schmelzdiagrammen hochviskoser Flüssigkeiten bestimmt werden.
Ein interessantes, hier störendes irreversibles Phänomen, für das eine Vielzahl von Informationen erhalten wurde, die bei Vorliegen einer noch größeren Zahl von Daten ausgewertet werden könnten, nämlich das Phänomen der Unterkühlung, spielt bei diesen Messungen wunschgemäß eine deutlich geringere Rolle als in Messungen mittels DTA oder DSC. Auch darüber wird in diesem Kapitel berichtet. Zur Reduktion dieser Unterkühlung, welche die Auswertung der Daten erheblich stört, wurden verschiedene Verfahren eingesetzt und untersucht.
Die in diesem Kapitel eingesetzte Methode ist weit weniger anfällig für Fehler, die durch die Unterkühlung der Proben hervorgerufen werden, als dies bei Messungen mit DSC oder DTA der Fall wäre, wie durch eine Bestimmung der eutektischen Zusammensetzung mittels ^{1}H-NMR Spektroskopie gezeigt werden konnte.

In **Kapitel 4** wird der Einsatz der Simplexmethode zur Optimierung der Leitfähigkeit von Elektrolytlösungen beschrieben. Diese geometrische Methode wurde zwar bisher für eine große Zahl von Optimierungen, insbesondere in Wirtschaft und Technik, eingesetzt, nicht aber in diesem Zusammenhang, in dem bisher trial-and–error, empirische sowie semiempirische Methoden vorherrschten. Ziel dieser Optimierung war auch die Untersuchung neuartiger Lösungsmittelmischungen, da die dem Stand der Technik entsprechenden Elektrolytlösungen nur eine geringe Leitfähigkeit aufweisen, sofern ihr Flüssigkeitsbereich überhaupt bis –25 °C reicht. Durch den Einsatz des für Batterieelektrolyte untypischen Lösungsmittels Essigsäureethylester ist es gelungen, sowohl den Flüssigkeitsbereich in Richtung tiefer Temperaturen hin zu erweitern als auch die Leitfähigkeit zu steigern. Die maximale Leitfähigkeit, die bei –25,00 °C erzielt wurde, beträgt 5,2 mS/cm. Im Vergleich zu dem erst kürzlich von Jow et al., siehe Abschnitt 4.5, publizierten Wert von 3,8 mS/cm für eine andere Mischung, stellt dies eine erhebliche Verbesserung dar, wenn die quadratische Abhängigkeit der Ausgangsleistung vom Innenwiderstand einer Batterie in Betracht gezogen wird. Bei den Optimierungen der Leitfähigkeit wurden Lösungsmittelmischungen untersucht, die sich aus bis zu vier organischen Carbonaten und aus bis zu vier organischen Carbonaten und Essigsäureethylester zusammensetzen. Durch den Einsatz der Simplexmethode konnte nicht nur eine deutliche Reduktion der Anzahl der nötigen Versuche, sondern auch eine enorme Steigerung der Leitfähigkeit erreicht werden. Eine hierzu eingereichte Veröffentlichung ist in Druck. Für alle für die Optimierung hergestellten Lösungen, wurde die Leitfähigkeit mit der in **Kapitel 2** beschriebenen Anlage ermittelt. Durch diese Messungen wurde auch der Flüssigkeitsbereich über den gesamten Temperaturbereich sichergestellt. Bei Essigsäureethylester handlet es sich um ein für Batterieelektrolyte untypisches Lösungsmittel, welches bisher noch nicht mit LiBOB als Leitsalz eingesetzt wurde. Daher wurde eine Reihe von Messungen an Modellsystemen durchgeführt. In diesem Zusammenhang wurden sowohl Lösungen von LiBOB in Mischungen untersucht, die sich nur aus organischen

Carbonaten zusammensetzen, als auch Mischungen aus organischen Carbonaten und Essigsäureethylester, um einen aussagekräftigen Vergleich zu erhalten.

In **Kapitel 5** wird ein neuentwickeltes Batterietestsystem vorgestellt. Dieses Gerät ergänzt die bereits vorhandenen Geräte im Bereich kleiner Ströme. Aufgrund seines kostengünstigen Aufbaus stellt dieses eine ernstzunehmende Konkurrenz für die auf dem Markt befindlichen Geräte dar. Da auch dieses Gerät einen modularen Aufbau aufweist, konnten bereits für die Phasendiagrammmessanlage entwickelte Module wiederverwendet werden, so dass bei diesem Gerät eine sehr kurze Entwicklungszeit von nur einer Woche bis zur Fertigstellung des ersten Prototypen benötigt wurde. Aufgrund des kostengünstigen Aufbaus wurde eine Kleinserie, die fünf Geräte umfasst, gebaut, um die in **Kapitel 6** beschriebenen Messungen in kurzer Zeit durchführen zu können. Da diese Geräte über zusätzliche Messeingänge verfügen, können sie leicht für zukünftige Anwendungen ausgerüstet werden. Durch seinen flexiblen Aufbau und seine modular angelegte Steuersoftware kann dieses Gerät auch für viele andere Anwendungsbereiche und Messmethoden eingesetzt werden. Wie auch die Phasendiagrammmessanlage, wird das Batterietestsystem über eine neu entwickelte Steuersprache angesteuert, so dass dieses Gerät für eine große Anzahl von Anwendungsbereichen eingesetzt oder angepasst werden kann, ohne dass es notwendig ist, den elektronischen Aufbau des Gerätes zu verändern.

In **Kapitel 6** wurde die Lebensdauer von Lithium-Ionen-Batterien bei hohen Temperaturen und Belastungen optimiert. Durch Zusatz von Additiven konnte die Lebensdauer bei 60 °C um den Faktor fünf gesteigert werden. Die Endkapazität der Zellen wurde verdoppelt. Im Gegensatz zu den in der Literatur beschriebenen Messungen wurde eine große Anzahl an Additiven unter den gleichen Versuchsbedingungen untersucht. Ebenso wurden, im Gegensatz zu den Messungen in der Literatur, keine Modellsysteme, sondern reale Batterien verwendet, wie sie auch in der Produktion eingesetzt werden. Diese Messungen, bei denen erstmals die Auswirkung von Additiven auf Batterien untersucht wurde, die LiBOB als Salz enthalten, weisen daher eine sehr gute Übertragbarkeit auf die Anwendung in Batterien auf. Da alle Messungen unter den gleichen Bedingungen an realen Zellen durchgeführt wurden, ist es zum ersten Mal möglich, die Auswirkung verschiedener Additive zu vergleichen, da bei den in der Literatur beschriebenen Experimenten unterschiedlichste Elektrodenmaterialien und Elektrolyte eingesetzt wurden.

In **Kapitel 7** werden die Ergebnisse der Untersuchungen verschiedener LiBOB-Chargen bezüglich ihrer Verunreinigungen dargestellt. Reinigungsmethoden wurden auf ihre Wirksamkeit hin untersucht. Um Wasserspuren in LiBOB quantitativ bestimmen zu können, war es erforderlich, ein neues Verfahren zu entwickeln. Die Standardmethode, die Karl-Fischer-Titration, versagt nämlich nicht nur bei diesem Salz, sondern auch bei einer Vielzahl anderer Stoffe, die mit den Reagenzien der Karl-Fischer-Methode reagieren. Mit der hier vorgestellten Methode ist es auch möglich, Wasserbestimmungen durchzuführen, die bei der Karl-Fischer-Titration zu einem falsch positiven Ergebnis führen würden. Für diese Messmethode wird die ^{1}H-NMR-Spektroskopie und eine Bezugssubstanz (Standardaddition) eingesetzt, um einen Wassergehalt im Bereich von ppm quantitativ im Salz nachzuweisen. Eine umfangreiche Fehlerrechnung (Anhang) sichert die Ergebnisse ab. Eine dazu eingereichte Veröffentlichung ist in Druck.

Es versteht sich von selbst, dass diese Arbeit zu weiteren Veröffentlichungen, Patenteinreichungen und Gebrauchsmusterschutzanmeldungen geführt hat. Durch die erforderliche Geheimhaltung der

Arbeiten im Rahmen des industriefinanzierten Projektes sind noch nicht alle Arbeiten eingereicht, sondern warten noch auf die Freigabe durch unseren Kooperationspartner, die Gaia-Akkumulatorenwerke, Nordhausen. Außerdem wurde bei Exist-Seed, einem staatlich geförderten Programm zur Existenzgründung, auf der Basis dieser Dissertation ein Antrag eingereicht, der zwar insoweit angenommen, aber noch nicht positiv beschieden ist.
Eingereichte und geplante Arbeiten sind in der Folge aufgeführt:

Veröffentlichungen:

-H.-G. Schweiger , M. Multerer , M. Schweizer-Berberich, and H. J. Gores, *Finding Conductivity Optima of Battery Electrolytes by Conductivity Measurements Guided by a Simplex Algorithm*, Journal of the Electrochemical Society, **im Druck,**

-H.-G. Schweiger, M. Multerer, U. Wietelmann, J.-C. Panitz, T. Burgemeisterc, and H. J. Gores NMR *Determination of Trace Water in Lithium Salts for Battery Electrolytes*, Journal of the Electrochemical Society, **im Druck,**

-H.-G. Schweiger, M. Multerer, and H. J. Gores, *Fast Multi-channel Precision Thermometer*, IEEE Trans. on Instrumentation and Measurement, **eingereicht am 18.5. 2004, Review process seit 3.8. 2004.**

Patente und Gebrauchsmusterschutzrechte:

- H.-G. Schweiger, M. Multerer, und H. J. Gores, *Patent DE103.59.463.9 Multikanalthermometer*, **angemeldet am 19. 12. 2003, Offenlegung am 23.6.2005**

- M. Schweizer-Berberich, H.-G. Schweiger , M. Multerer, und H. J. Gores, *Patent DE103.59.604.6*, *Elektrolyt zur Verwendung in einer elektrochemischen Zelle und elektrochemische Zelle mit den Elektrolt,* **angemeldet am 30.1.2004, Offenlegung am 23.6.2005**

- H.-G. Schweiger, M. Multerer, und H. J. Gores, *Gebrauchsmuster, Gerät zur Messung von Phasenübergängen, zur schnellen Messung der Temperaturabhängigkeit von Leitfähigkeiten und zur Regelung und Steuerung von chemischen Prozessen*, a**ngemeldet am 20.9.2004**

Veröffentlichungen, die sich in Arbeit befinden und deren Veröffentlichung noch nicht genehmigt ist:

-H.-G. Schweiger , M. Multerer , M. Schweizer-Berberich, and H. J. Gores, *Additives for Electrolytes of Lithium-Ion Batteries based on Lithiumbisoxalatoborate for improved performance at elevated temperatures*, Journal of the Electrochemical Society,

-H.-G. Schweiger , M. Multerer and H. J. Gores, *Multi-channel Apparatus for the Determination of Phase Diagrams* Thermochimica Acta, oder Fluid Phase Equilibria.

-H.-G. Schweiger , M. Multerer and H. J. Gores, *Fast determination of the temperature dependence of the conductivity of electrolyte solutions*, Journal of the Electrochemical Society,

- Hans-Georg Schweiger, Michael Multerer, and H. J. Gores, *Fast Multi-channel Conductivity Measurements*, IEEE Trans. on Instrumentation and Measurement,

-H.-G. Schweiger, M. Multerer, G. Schmeer and H. J. Gores, *Low cost electrochemical measurement System*, Journal of Chemical Education, aufgrund der geplanten Firmengründung/Exist-Seed Förderung noch nicht veröffentlicht.

geplante Patente und Gebrauchsmusterschutzrechte:

- M. Schweizer-Berberich H.-G. Schweiger , M. Multerer, und H. J. Gores, *Additive für Lithium-Ionen-Batterien*,

- weitere Gebrauchsmuster von neu entwickelten Geräten, ca. fünf Neuanmeldungen, aufgrund der geplanten Firmengründung/Exist-Seed Förderung noch nicht eingereicht.

9 Abbildungsverzeichnis

10 Tabellenverzeichnis

11 Anhang

11.1 Grundlegende Arbeitstechniken und Verfahren

11.1.1 Inertgassystem

Um eine Kontamination der eingesetzten Chemikalien mit Sauerstoff und Wasserspuren zu vermeiden wurden, soweit nicht anders aufgeführt, sämtliche Operationen unter Schutzgas ausgeführt. Aus diesem Grund wird daher in der gesamten Arbeit der Einsatz der Schutzgastechnik nicht mehr besonders erwähnt. Neben der klassischen Schlenk-Technik wurden vor allem der Handschuhkasten und speziell konstruierte Messzellen eingesetzt.

11.1.1.1 Argon-Handschuhkasten

Als Handschuhkasten wurde ein Modell der Firma MBraun (Melsungen) vom Typ MB150BG verwendet. Durch die ständige Umwälzung über Molsieb (X-13) zur Entfernung von Wasser und einem BTS-Katalysator zur Bindung von Sauerstoffspuren wird der Wassergehalt unter 0,2 ppm und der Sauerstoffgehalt unter 5 ppm gehalten. Stickstoffspuren, welche Lithiumnitrid bei den in Abschnitt 11.1.2 beschriebenen Lithiumelektroden bilden würde, werden monatlich mit einem 740 °C heißen Titanschwamm vom Typ MB200N entfernt. Lösungsmittelspuren können mittels einer Kühlfalle und flüssigem Stickstoff beseitigt werden. Alle in dieser Arbeit verwendeten Chemikalien wurden in diesem Handschuhkasten aufbewahrt, um Hydrolyse oder die Aufnahme von Wasser zu vermeiden.

11.1.1.2 Stickstoffnachreinigung

Alle Versuche, die nicht im Handschuhkasten durchgeführt werden konnten, wie zum Beispiel das Trocknen und Umkristallisieren des LiBOBs, wurden mittels der Schlenktechnik und Stickstoff oder soweit erforderlich mit Argon als Schutzgas durchgeführt. Das Schutzgas wird über eine Nachreinigungsanlage mit einem BTS-Katalysator von BASF (Ludwigshafen) von Sauerstoffspuren befreit. Wasserspuren werden mit je einer mit Calciumchlorid, Blaugel und einer mit Siccapent® gefüllten Säule entfernt. Da die im Labor vorhandene Stickstoffanlage seit Jahren nicht mehr in Betrieb war, wurde die BTS-Säule mit Formiergas regeneriert und die anderen Säulen neu befüllt.

11.1.2 Cyclovoltammetrie

Messzelle

Die elektrochemischen Messungen der Substanzen wurden alle in einer speziell konstruierten Messzelle durchgeführt, die in [1] beschrieben ist. In dieser Messzelle werden als Referenz und Gegenelektrode ein circa 10 mm breites und circa 7 cm langes Lithiumblech eingesetzt. Das Blech, das als Referenzelektrode dient, wurde wie in [1] beschrieben, präpariert und in die Zelle eingebaut. Da in der vorliegenden Arbeit ausschließlich diese Elektrode als Referenz benutzt wurde, beziehen sich sämtliche Potenzialangaben auf diese Elektrode. Als Arbeitselektroden wurden teflonummantelte Platin-, Glassycarbon-, Aluminium- und Kupferelektroden der Firma Metrohm (Herisau) mit einer Fläche von 0,0707cm² verwendet.

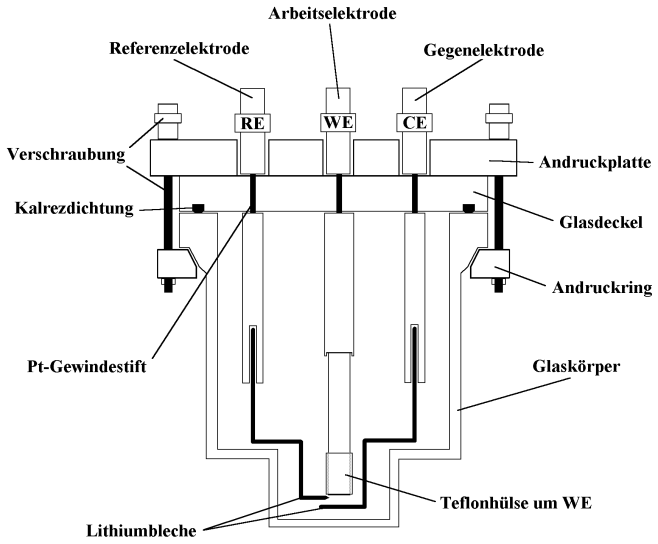


Abbildung 51 Die elektrochemische Messzelle [1]

Um reproduzierbare Ausgangsbedingungen zu schaffen wurden die Elektroden mit einer Poliermaschine der Firma Siemens poliert. Für die Politur wurden Diamantsuspensionen in den Korngrößen 9 μm und 1 μm der Firma Struers (Willich) eingesetzt. Die Polierpads stammen von der Firma Heraeus Kulzer (Wehrheim) und wurden in den Stufen grün, gelb und rot eingesetzt, wobei die Stufe grün mit einer 9 μm Suspension und die beiden anderen Pads mit einer 1 μm Suspension verwendet wurden. Nach der Politur wurden die Elektroden für mindesten 24h bei 60°C im Vakuumtrockenschrank getrocknet.

Autolab

Alle Messungen wurden mit einem Autolab Gerät (Lieferant Metrohm), das mit einem A/D-Wandler ADC164 und mit einem D/A-Wandler DAC164 ausgestattet ist, vorgenommen. Bei allen Messungen wurde die Dreielektrodenanordnung verwendet, wobei als Referenz- und Gegenelektrode ein Lithiumblech dienten. Soweit nicht anders aufgeführt, betrug bei allen Messungen die Vorschubsgeschwindigkeit 10 mV/s. Bei den in den Kapiteln 4.4 und 11.2.3.1 beschriebenen Messungen wurde ausgehend vom ersten Umkehrpotenzial hin polarisiert. Bei allen Messungen wurde als zweites Umkehrpotenzial das Ruhepotenzial verwendet.

11.1.3 Impedanzspektroskopie

Impedanzspektren wurden mit einem elektrochemischen Messplatz der Firma Zahner (Kronach) vom Typ IM6 aufgenommen. Dieses Gerät ist in der Grundausführung vorhanden und besitzt einen Potenzialbereich von ±4 V mit Pufferverstärker ±10 V gegen die Referenzelektrode und ±12 V gegen die Gegenelektrode. Der Strommessbereich reicht von ±100 nA bis ±1 A in 26 Schritten bei einer Auflösung von mindestens 0,1 % des Messbereiches. Impedanzmessungen sind in einem Frequenzbereich von 10 μHz bis 8 MHz möglich.

Alle in dieser Arbeit beschriebenen Impedanzmessungen wurden beim Klemmenpotenzial der elektrochemischen Zelle durchgeführt. Der untersuchte Frequenzbereich betrug 100 kHz bis 10 mHz.

11.1.4 NMR Spektroskopie

11.1.4.1 ^{1}H-NMR und ^{13}C-NMR Spektroskopie

Für alle, bei Raumtemperatur durchgeführten Messungen, wurde ein Bruker AVANCE 300 (300.13 MHz Protonen) mit 5 mm Dualprobenkopf (1H, 13C) und Probenwechsler verwendet. Da kein TMS als interner Standard zugegeben wurde, diente das NMR-Lösungsmittel zur Kalibrierung der chemischen Verschiebung. Die verwendeten NMR-Röhrchen waren vom TYP 507-pp8 von RototecSpintec (Biebesheim). Diese wurden, soweit nicht anders angegeben, mit Naturgummi 5mm „Serum Caps“ (siehe auch Kapitel 11.3.4.4) des gleichen Herstellers verschlossen. Die Spektren wurden mit WIN-NMR 6.2.0.0 der Firma Bruker Daltronik GmbH ausgewertet.
Jedes ^{1}H-NMR Spektrum wurde mit dem zg30 pulse Programm der Firma Bruker aufgenommen. Die Relaxationspause betrug, 2s, das pre-scan delay 6,0µs, die Aufnahmedauer 3,65 s[36]. Bei diesen Messungen wurde über 320 Pulse integriert.

11.1.4.2 ^{11}B-NMR Spektroskopie

Diese Spektren wurden in der NMR-Abteilung der Universität Regensburg unter der Leitung von Herrn Dr. Burgameister an einem Bruker Avance 400 (400,13 MHz Protonen) aufgenommen. Die chemischen Verschiebungen werden in Einheiten der δ-Skala angegeben. Als Standard wurde für ^{11}B externes BF3*Diethylether eingesetzt.

11.1.5 Pulverdiffraktometrie

Die Pulverdiffraktogramme wurden auf einem Stadi-Diffraktometer der Firma STOE & CIE angefertigt. Dazu wurden die Proben fein verrieben und auf Flachbettträgern vermessen, wobei die mikrokristalline Probe auf Mylarfolie mit Bysilone® Paste fixiert wurde. Als Messstrahlung wurde eine, mittels eines Germaniumeinkristalls monochromatisierte CuKα_1 (λ=1,540598 Å) verwendet. Der untersuchte Bereich betrug $8° \leq 2\Theta \leq 90°$C. Als externer Standard wurde Silizium eingesetzt. Die Auswertung erfolgte mit dem Programm WinX^{POW} Ver. 1.08 [37].

11.1.6 Karl-Fischer-Titration

Die Bestimmung des Wassergehaltes von Lösungsmitteln wurde durch eine coulometrische Titration nach der Methode von Karl Fischer durchgeführt, die auf folgendem Prinzip beruht. Die zu untersuchende Substanz wird in eine Lösung eingegeben, die im wesentlichen Methanol, Pyridin

[36] Diese Zeitdauer, zusammen mir der Relaxationspause von 2s ist lange genug um die komplette Relaxation (5 × T1) des Wasserpeaks abzuwarten. T_1 wurde für diesen Peak zu 3,2s Sekunden bestimmt. Da nur mit einem 30° Puls angeregt wird ist beträgt die Relaxationszeit nur 1/3 von 3,2s.

[37] Thomas Bernert vom Lehrstuhl Prof. Pfitzner sei an dieser Stelle für die Durchführung dieser Messungen gedankt.

oder eine andere Hilfsbase B, SO2 und Jodid enthält. Als Titrator dient Jod, das entweder aus einer Stammlösung stammt, oder an einer Platinelektrode elektrochemisch erzeugt wird:

$$2\,I^- \longrightarrow I_2 + 2e^- \quad (37)$$

Das Jod reagiert in der Lösung nach folgender Gleichung (38):

$$I_2 + SO_2 + 3\,B + H_2O + CH_3OH \longrightarrow 2\,[C_5H_5NH]I + [BH][SO_4CH_3] \quad (38)$$

im Verhältnis 1:1 mit dem in der Lösung enthaltenen Wasser. Ist der Äquivalenzpunkt überschritten, so wächst die Konzentration an freiem Jod in der Lösung an. Das Redoxpaar Jod/Jodid kann am Äquivalenzpunkt mit Hilfe einer Platindoppelelektrode amperometrisch oder potentiometrisch detektiert werden. Die Integration des Stromes liefert nach dem Faradayschen Gesetz die Menge des in der Messlösung enthaltenen Wassers.
Die Wassergehaltsbestimmungen wurden mit zwei verschiedenen Geräten durchgeführt. Das Mitsubishi Moisturemeter Model CA-20 basiert auf der coulometrischen Reagenzerzeugung. Die Nachweisgrenze beträgt 10 µg H_2O. Die Genauigkeit beträgt ± 3 ppm im Bereich von 10 ppm bis 1000 ppm. Der Karl-Fischer-Titrator DL 18 der Firma Mettler basiert im Gegensatz dazu auf der volumetrischen Titration. Die Nachweisgrenze dieses Gerätes liegt ebenfalls bei einigen ppm.

11.1.7 Anionen-ESI

Die Massenspektren wurden an der Abteilung für Massenspektrometrie der Universität Regensburg unter der Leitung von Herrn Dr. K. K. Mayer mit einem Fingan Mat 95 aufgenommen. Als Ionisierungsmethode wurde Elektrospray Ionisation (ESI) und als Matrix Acetonitril verwendet.

11.1.8 Hochvakuumstrocknungsanlage

Für das Trocknen im Hochvakuum wurde eine neue Hochvakuumanlage aufgebaut. Das Hochvakuum wird mit einer Öldiffusionspumpe der Firma Leybold Vakuum GmbH (Köln) erzeugt, die einen Unterdruck von 10^{-6} mbar liefert. Als Vorpumpe dient eine Rotationspumpe der Firma Vaccubrand GmbH & Co. KG vom Typ RD4. Über Planflanschbauteile sind zwei Vakuummessgeräte angeschlossen Das Feinvakuum wird mit einem Messgerät der Firma Edwards (West Sussex) vom Typ Penning 8 mit einem Penning 25 Ionisationsmesskopf bestimmt. Das Vorvakuum wird mit einem Penning HKM 10 Messkopf und einem Penning H1005 Messgerät gemessen. Mit einer vom Autor dieser Arbeit entworfenen und von der Glasbläserwerkstatt der Fakultät für Chemie der Universität Regensburg angefertigten Glaskühlfalle werden Lösungsmittelreste ausgefroren. Um eine optimale Dichtigkeit der Anlage zu gewährleisten sind sämtliche Verbindungen mittels Kleinflanschbauteile ausgeführt worden. Sogar die Kühlfalle wird über eine Glasskleinflansche der Aachener Quarzglaswerke und Edelstahlwellschläuche an die Anlage anschlossen.

11.2 Detaillierte Messergebnisse

11.2.1 Kalibrierparameter der Messfühler

Fühler	A 1/K	B 1/K	C 1/K	U_{REF} / V	R_1 / Ω
1	$1{,}048736\times10^{-3}$	$2{,}149957\times10^{-4}$	$8{,}2115\times10^{-8}$	2,502904	100000
2	$1{,}051776\times10^{-3}$	$2{,}150659\times10^{-4}$	$7{,}7936\times10^{-8}$	2,502731	100000
3	$1{,}048329\times10^{-3}$	$2{,}151576\times10^{-4}$	$8{,}1084\times10^{-8}$	2,502487	100000
4	$1{,}049817\times10^{-3}$	$2{,}152516\times10^{-4}$	$7{,}7665\times10^{-8}$	2,502543	100000
5	$1{,}046108\times10^{-3}$	$2{,}155915\times10^{-4}$	$7{,}6677\times10^{-8}$	2,503125	100000
6	$1{,}058231\times10^{-3}$	$2{,}139454\times10^{-4}$	$8{,}1809\times10^{-8}$	2,502623	100000
7	$1{,}046517\times10^{-3}$	$2{,}153733\times10^{-4}$	$8{,}0770\times10^{-8}$	2,502602	100000
8	$1{,}053627\times10^{-3}$	$2{,}145448\times10^{-4}$	$8{,}2264\times10^{-8}$	2,502304	100000
9	$1{,}037914\times10^{-3}$	$2{,}163758\times10^{-4}$	$7{,}9110\times10^{-8}$	2,503801	100000
10	$1{,}046256\times10^{-3}$	$2{,}157059\times10^{-4}$	$7{,}6568\times10^{-8}$	2,503202	100000
11	$1{,}038157\times10^{-3}$	$2{,}164105\times10^{-4}$	$7{,}8190\times10^{-8}$	2,503378	100000
12	$1{,}045069\times10^{-3}$	$2{,}155764\times10^{-4}$	$8{,}0486\times10^{-8}$	2,502799	100000
13	$1{,}026479\times10^{-3}$	$2{,}177988\times10^{-4}$	$7{,}5258\times10^{-8}$	2,505033	100000
14	$1{,}044654\times10^{-3}$	$2{,}158995\times10^{-4}$	$7{,}5904\times10^{-8}$	2,503427	100000
15	$1{,}021098\times10^{-3}$	$2{,}184850\times10^{-4}$	$7{,}3721\times10^{-8}$	2,504787	100000
16	$1{,}044885\times10^{-3}$	$2{,}155899\times10^{-4}$	$8{,}0628\times10^{-8}$	2,502919	100000
17	$1{,}043800\times10^{-3}$	$2{,}152775\times10^{-4}$	$8{,}0673\times10^{-8}$	2,503327	100000
18	$1{,}049933\times10^{-3}$	$2{,}148707\times10^{-4}$	$8{,}2738\times10^{-8}$	2,502753	100000
19	$1{,}045205\times10^{-3}$	$2{,}157517\times10^{-4}$	$7{,}6170\times10^{-8}$	2,502734	100000
20	$1{,}056730\times10^{-3}$	$2{,}141793\times10^{-4}$	$8{,}1300\times10^{-8}$	2,502432	100000
21	$1{,}044674\times10^{-3}$	$2{,}155102\times10^{-4}$	$8{,}1349\times10^{-8}$	2,503175	100000
22	$1{,}045725\times10^{-3}$	$2{,}156988\times10^{-4}$	$7{,}8453\times10^{-8}$	2,502844	100000
23	$1{,}045772\times10^{-3}$	$2{,}154810\times10^{-4}$	$8{,}0426\times10^{-8}$	2,502686	100000
24	$1{,}050169\times10^{-3}$	$2{,}149148\times10^{-4}$	$8{,}1869\times10^{-8}$	2,502599	100000
25	$1{,}050612\times10^{-3}$	$2{,}148288\times10^{-4}$	$7{,}9773\times10^{-8}$	2,503306	100000
26	$1{,}058144\times10^{-3}$	$2{,}140690\times10^{-4}$	$8{,}1138\times10^{-8}$	2,502861	100000
27	$1{,}042137\times10^{-3}$	$2{,}158538\times10^{-4}$	$8{,}0331\times10^{-8}$	2,502822	100000
28	$1{,}052379\times10^{-3}$	$2{,}147856\times10^{-4}$	$8{,}1686\times10^{-8}$	2,502521	100000
29	$1{,}043368\times10^{-3}$	$2{,}157360\times10^{-4}$	$7{,}9955\times10^{-8}$	2,503461	100000
30	$1{,}049903\times10^{-3}$	$2{,}150086\times10^{-4}$	$8{,}1510\times10^{-8}$	2,502910	100000
31	$1{,}068981\times10^{-3}$	$2{,}120700\times10^{-4}$	$9{,}0195\times10^{-8}$	2,500000	100000
32	$1{,}068981\times10^{-3}$	$2{,}120700\times10^{-4}$	$9{,}0195\times10^{-8}$	2,500000	100000

Tabelle 23 Kalibrierparameter der Thermistoren

Die Fühler 31 und 32 werden zur Messung der Innentemperatur des Geräts eingesetzt. Für sie werden die Standardparameter für die Thermistoren eingesetzt. Die restlichen Parameter wurden nach der in Abschnitt 2.7.3 beschriebenen Methode kalibriert.

11.2.2 Untersuchung der elektrochemischen Stabilität von Elektrolytlösungen an Modellsystemen

11.2.2.1 EC/PC/DMC

Als Modellmischung für eine reine Carbonatmischung wurde eine Lösung LiBOB[38] mit Molalität von (0,89 ± 0,03) mol/kg in einer Propylencarbonat/Ethylencarbonat/Dimethylcarbonat Mischung mit einem Gewichtsverhältnis von 1:1:1 eingesetzt. Da die Messungen an der Platinelektrode auch zur Bestimmung der Verunreinigungen der eingesetzten LiBOB Chargen verwendet wurden, werden die Ergebnisse dieser Messungen in dem entsprechenden Kapitel (11.2.3.1) beschrieben.

11.2.2.1.1 Glassycarbonelektrode

In Abbildung 52 sind jeweils die ersten und fünften Zyklen der Cyclovoltammogramme einer Lösung von LiBOB in einer 1:1:1 Mischung (EC/PC/DMC) an einer Glassycarbonelektrode (GC) dargestellt. Das erste Umkehrpotenzial wurde ausgehend von 0,3 V vs. Li/Li+[39] über 0,7 V bis 1,2 V vs. Li/Li+ variiert. Bei allen Messungen entsprach das zweite Umkehrpotenzial dem Ruhepotenzial (OCP), das bei diesem System 2,85 V vs. Li/Li+ beträgt.

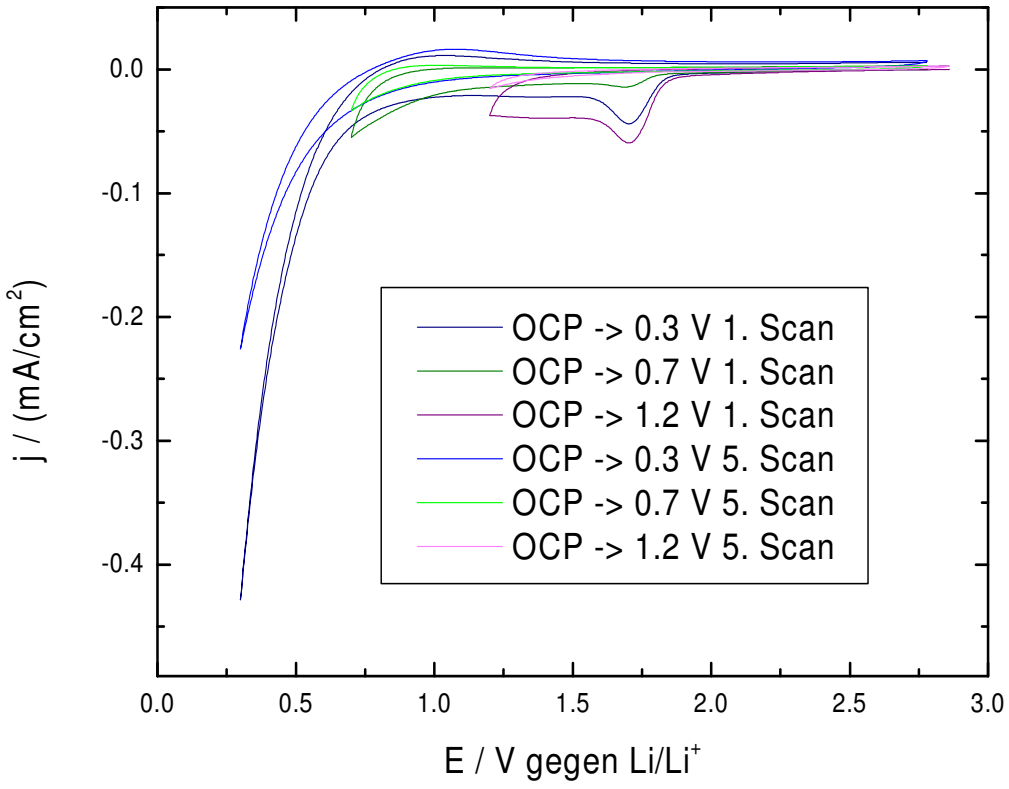


Abbildung 52 kathodische Stabilität von LiBOB in EC/PC/DMC an Glassycarbon

[38] Charge SCHK01/29

[39] Alle Potenzialangaben beziehen sich auf die Pseudoreferenz Li/Li+

In den Cyclovoltammogrammen, die in Abbildung 52 dargestellt sind, ist ein reduktiver Peak bei 1,7 V vs. Li/Li^+ zu erkennen. Es liegt nahe, dass dieser Peak durch eine partiell oxidierte Oberfläche der Kohlenstoffelektrode hervorgerufen wird. Durch die relativ hohen Temperaturen beim Trocknen der Elektroden (65 °C) wird diese Oxidation noch begünstigt. Für diese These spricht, dass in den beiden Zyklen, bei denen das untere Wendepotenzial bei 0,3 V beziehungsweise 0,7 V lag, und somit stärker reduziert wurde und dann in den folgenden Zyklen dieser Peak nicht mehr auftritt, während im CV mit dem höchsten Wendepotenzial (1,2 V), dieser Peak in den nachfolgenden Zyklen noch abgeschwächt auftritt. Die unterschiedliche Peakhöhe bei den einzelnen Messungen ließe sich durch unterschiedlich starke Oxidation der Oberfläche erklären, da für jede Messung eine frische Elektrode verwendet wurde. Aufgrund von leichten Unterschieden beim Trocknungsprozess kann diese verschieden starke Oxidation erklärt werden.
In den Cyclovoltammogrammen wird eine deutliche Abnahme der kathodischen Grenzströme mit der Zyklenzahl beobachtet. Um diese zu untersuchen, wurde ein Experiment durchgeführt, bei dem, statt wie üblich 5 Zyklen, 10 Zyklen aufgenommen worden sind.

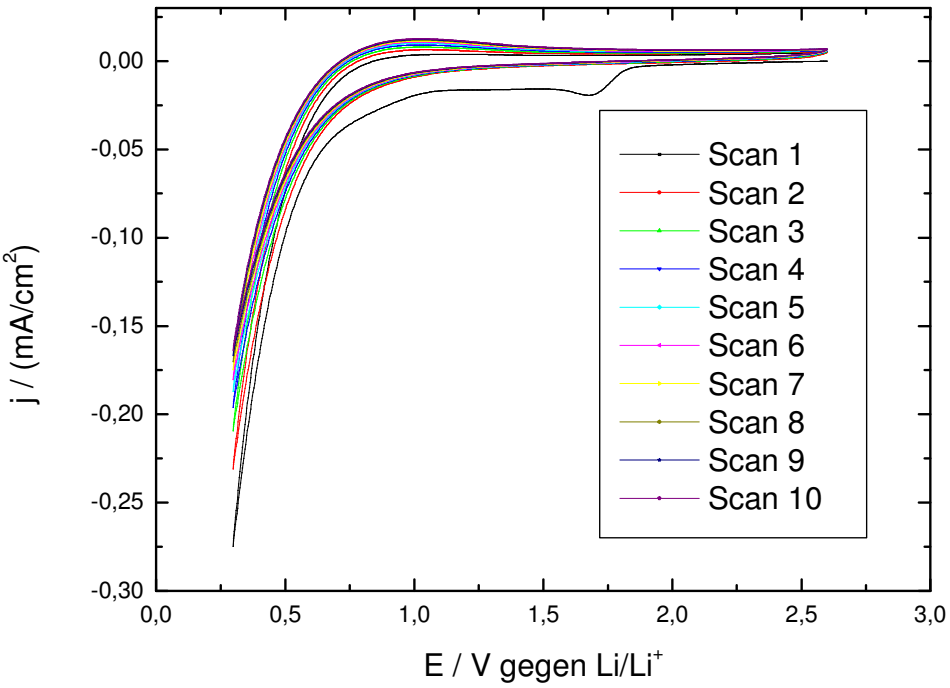


Abbildung 53 Abnahme der kathodischen Grenzströme in zunehmender Zyklenzahl an Glassycarbon

Auch dieses Experiment zeigt eine deutliche Abnahme des kathodischen Grenzstroms mit der Zyklenzahl, wie in Abbildung 53 dargestellt ist. Dieses für Batterien wünschenswerte Verhalten lässt sich durch Ausbildung einer sogenannten „solid-electrolyte interphase“ (SEI) erklären. Siehe auch Kapitel 6.2.1.

11.2.2.1.2 Kupferelektrode

Analog zu den Messungen an Glassycarbon wurde auch die Stabilität von LiBOB in der Carbonatmischung an Kupfer untersucht. Bei diesen Experimenten wurde das erste Umkehrpotenzial zwischen 0,0 V und 1,5 V in 0,5 V Schritten variiert. Das Ruhepotenzial, und somit das zweite Umkehrpotenzial liegt bei diesem System bei 3,17 V. Um eine Oxidation der Kupferelektroden beim Trocken im Vakuumtrockenschrank zu vermeiden, wurden die Elektroden im Handschuhkasten unter Argon mit einem Fine Grit Pad der Firma BAS Instruments Ltd. (Warwickshire) poliert.
In Abbildung 54 sind die ersten beziehungsweise fünften Zyklen der Cyclovoltammogramme an Kupfer gezeigt.

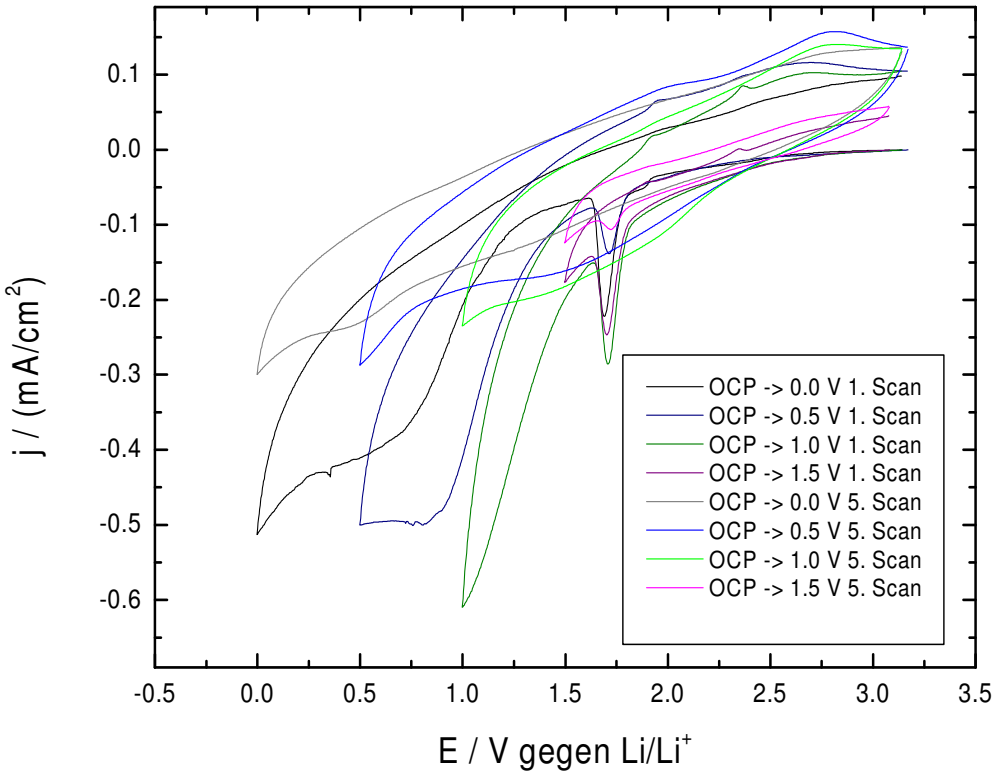


Abbildung 54 kathodische Stabilität von LiBOB in EC/PC/DMC an Kupfer

In dieser Abbildung ist deutlich ein reduktiver Peak bei ca. 1,7 V zu erkennen, der schon beim zweiten Zyklus nicht mehr auftritt. Liegt das Wendepotenzial bei nur 1,5 V, so baut sich dieser Peak nur langsam ab. Da dieser Peak das gleiche günstige Verhalten wie der Peak an Glassycarbon aufweist, dürfte auch hier eine Oxidschicht auf der Oberfläche, die durch den ersten Reduktionszyklus entfernt wurde, die Ursache für diesen Peak sein. Wie beim Glassycarbon tritt auch hier eine deutliche Abnahme der kathodischen Grenzströme mit zunehmender Zyklenzahl auf. Aufgrund dieser Passivierung ist daher Kupfer als Material für Stromableiter in LiBOB haltigen Lithium-Ionen-Batterien geeignet.

11.2.2.1.3 Aluminiumelektrode

Im Gegensatz zu den in den vorigen Kapiteln untersuchten Materialien ist an Aluminium nicht die kathodische Stabilität von Interesse, sondern die anodische, da dieses Material als Stromableiter für das Kathodenmaterial eingesetzt wird.
Wie bei den anderen Materialen auch wurde die Zyklengröße durch Variation des ersten Umkehrpotenzials variiert, wobei hier ein Bereich von 4,5 V bis 5,5 V in 0,5 V Schritten untersucht wurde. Das zweite Wendepotenzial war hier wieder das Ruhepotenzial, welches bei diesem Elektrolyten 2,70 V beträgt.

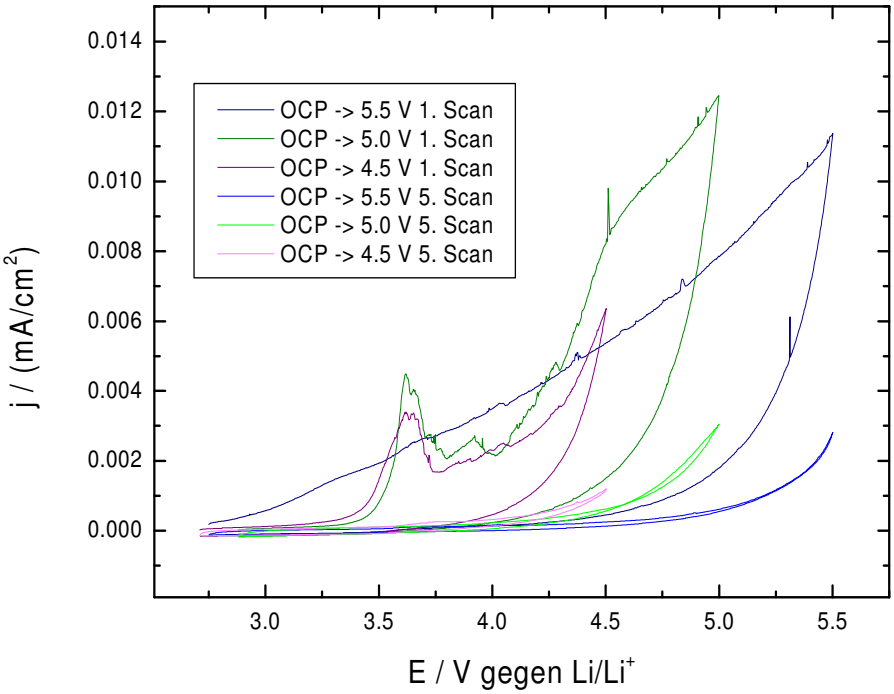


Abbildung 55 anodische Stabilität von LiBOB in EC/PC/DMC an Aluminium

Interessanterweise zeigen die Cyclovoltammogramme, die in Abbildung 55 dargestellt sind, mit einem ersten Wendepotenzial bei 4,5 V beziehungsweise 5,0 V einen Peak bei ca. 3,5V im ersten Zyklus, der bei dem dritten Cyclovoltammogramm nicht beobachtet wird. Auch bei den in der Diplomarbeit des Autors [20] gezeigten Cyclovoltammogrammen tritt dieser Peak nicht auf. Eine mögliche Ursache für diesen Peak dürfte in Verunreinigungen der Elektrodenoberflächen zu sehen sein.
Auch an diesem Elektrodenmaterial wird mit zunehmender Zyklenzahl und abnehmender Zyklengröße eine deutliche Abnahme der Stromdichten beobachtet, wie aus Abbildung 55 ersichtlich ist. Aluminium eignet sich aufgrund der auftretenden Passivierung als Material für Stromableiter in LiBOB Lithium-Ionen-Batterien. Vor allem an diesem Material zeigt sich die Überlegenheit von LiBOB gegenüber halogenidhaltigen Batterieelektrolyten, da sich diese Ionen gegenüber Aluminium als besonders korrosiv erwiesen haben [2], [3].

11.2.2.2 EC/PC/DMC/EA

Nachdem, wie in Kapitel 4.2 gezeigt, Elektrolytlösungen mit hohem Ethylacetatgehalt eine ausgezeichnete Leitfähigkeit bei tiefen Temperaturen aufweisen, sollte deren elektrochemische Stabilität untersucht werden. Als Beispiel für diese Elektrolytlösungen wurde eine Mischung aus Ethylencarbonat/Propylencarbonat/Dimethylcarbonat/Ethylacetat mit einem Gewichtsverhältnis von 1:1:1:1 und einer LiBOB-Molalität von (0,67 ± 0,03) mol/kg untersucht, wobei die gleichen Bedingungen und Elektroden wie bei den reinen Carbonatmischungen, siehe Abschnitt 11.2.2.1, eingesetzt wurden. Da sich das elektrochemische Verhalten dieser Elektrolytlösung gegenüber den eingesetzten Elektroden nicht wesentlich von der reinen Carbonatmischung unterscheidet, wurde hier auf die Variation der Zyklengröße verzichtet.

11.2.2.2.1 Platinelektrode

Das elektrochemische Verhalten der EC/PC/DMC/EA Lösung an Platin wurde sowohl in anodischer wie auch in kathodischer Richtung hin untersucht. In Abbildung 56 ist das Verhalten dieser Lösung bei kathodischen Potenzialen mit zunehmender Zyklenzahl dargestellt. Bei diesem Versuch wurde ausgehend vom Ruhepotenzial von 3,01 V in kathodischer Richtung bis 0,30 V polarisiert.

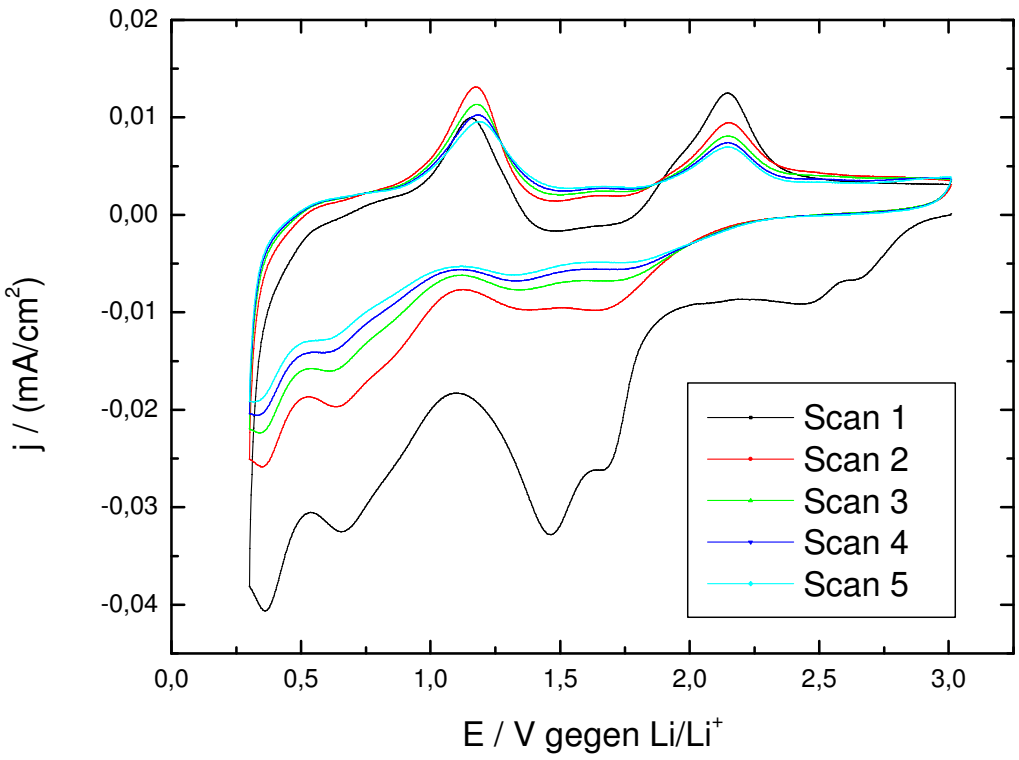


Abbildung 56 kathodische Stabilität von LiBOB in EC/PC/DMC/EA an Platin

Der Vergleich mit dem CV der ethylacetatfreien Lösung aus Abbildung 65 zeigt keine nennenswerten Unterschiede bis auf eine langsamere Abnahme der Wasserreduktionspeaks bei 1,4 V mit zunehmender Zyklenzahl.

Die anodische Stabilität wurde ausgehend vom Ruhepotenzial von 3,03 V bis 5,00 V untersucht. Die Ergebnisse sind in Abbildung 57 dargestellt.

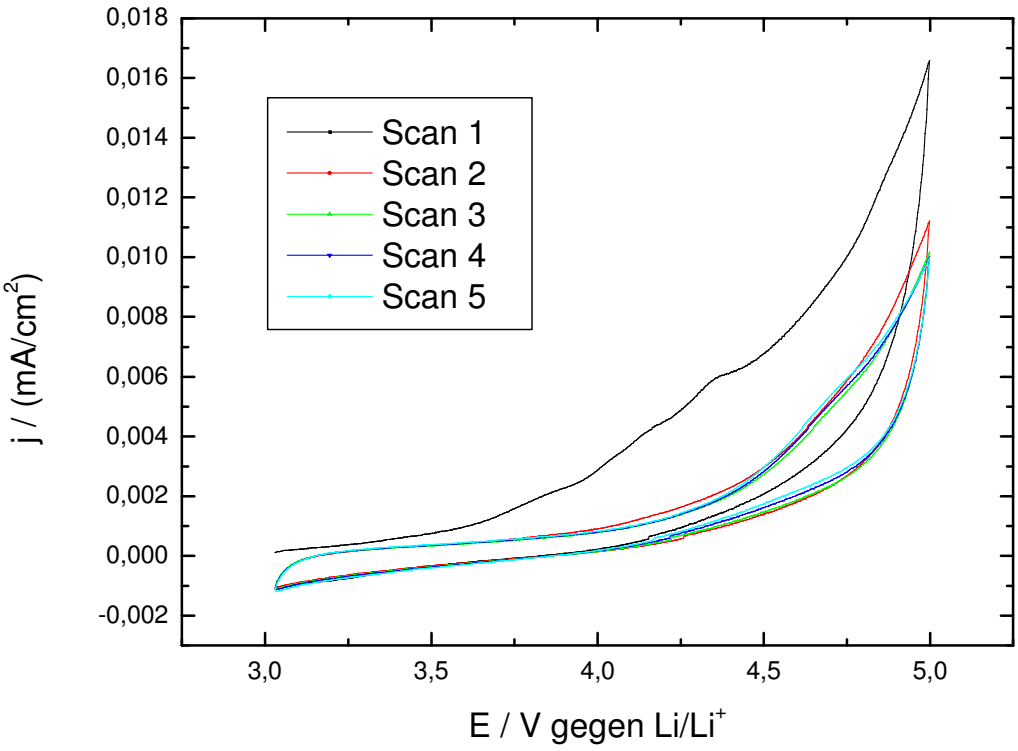


Abbildung 57 anodische Stabilität von LiBOB in EC/PC/DMC/EA an Platin

Bis auf ein minimales Anwachsen der anodischen Grenzströme wird auch hier im Vergleich zur essigsäureethylesterfreien Lösung, siehe Abbildung 65, keine bedeutende Veränderung beobachtet.

11.2.2.2.2 Glassycarbonelektrode

Das kathodische Verhalten von Glassycarbon an dieser Elektrolytmischung wurde im Bereich von 3,33 V (Ruhepotenzial) bis 0,00 V untersucht. Das erhaltene Ergebnis ist in Abbildung 58 dargestellt.

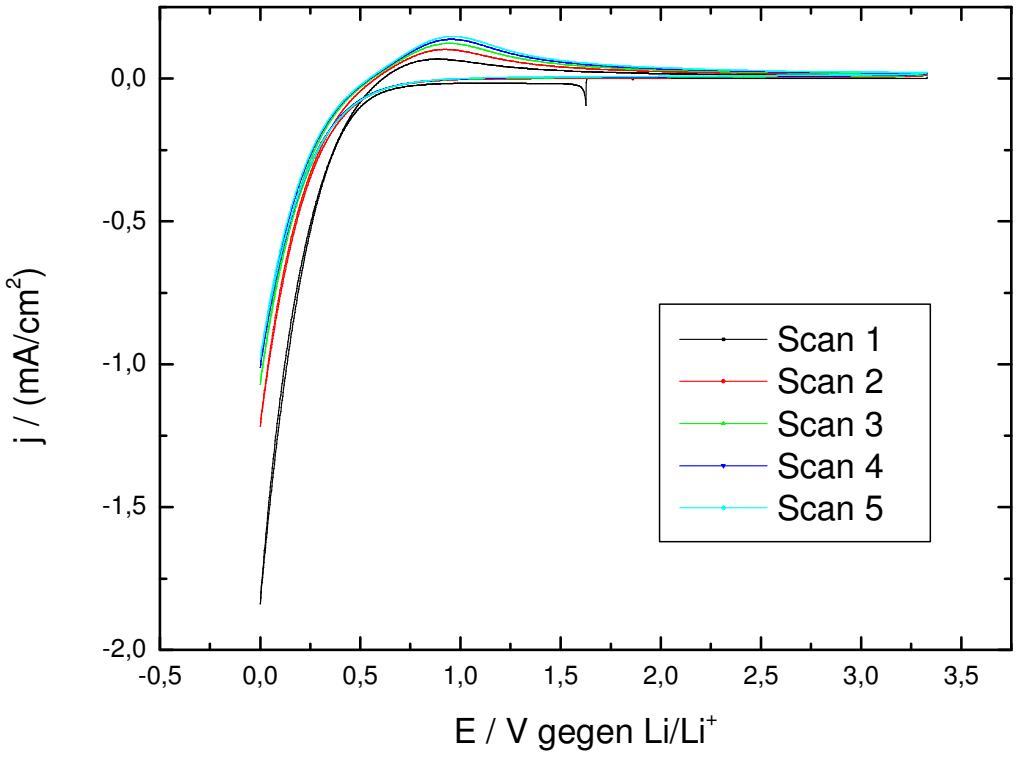


Abbildung 58 kathodische Stabilität von LiBOB in EC/PC/DMC/EA an Glassycarbon

Auch hier zeigt der Vergleich mit der essigsäureethylesterfreien Lösung, Abbildung 52, etwas größere Grenzströme. Darüber hinaus ist der anodische Peak bei 1,5 V gegen Lithium in essigsäureethylesterhaltigen Lösungen nahezu unsichtbar. Die Ursache dafür ist unbekannt.

11.2.2.2.3 Kupferelektrode

Die kathodische Stabilität von Kupfer wurde ausgehend vom Ruhepotenzial bei 2,99 V bis hin zu 0,00 V untersucht. Das erhaltene Cyclovoltammogramm ist in Abbildung 59 dargestellt.

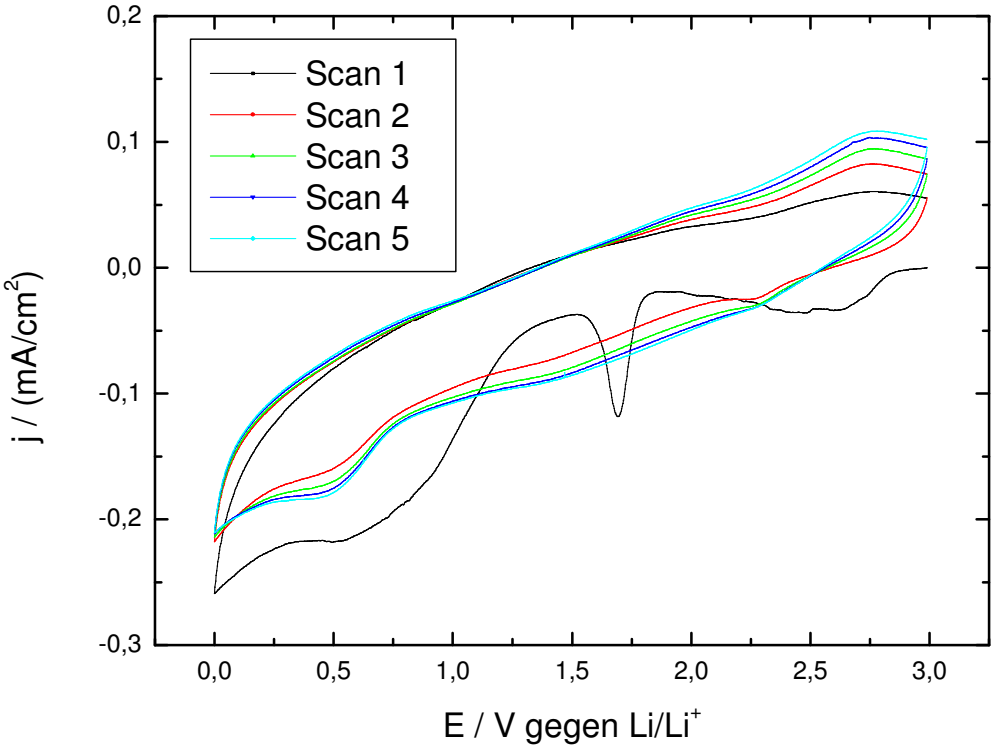


Abbildung 59 kathodische Stabilität von LiBOB in EC/PC/DMC/EA an Kupfer

Im Vergleich zu den Cyclovoltammogrammen der reinen Carbonatmischung, siehe Kapitel 11.2.2.1.2, werden hier Ströme beobachtet, die in etwa um den Faktor 2 größer sind. Darüber hinaus zeigt sich hier im ersten Zyklus ein breiter reduktiver Peak bei 2,5 V. Die Frage, ob diese Differenzen auf eine unterschiedliche Beschaffenheit der Elektrodenoberfläche, die bei Kupferelektroden besonders oxidationsempfindlich ist, oder auf die geringe Reduktionsstabilität des Essigsäureethylesters zurückzuführen ist, kann noch nicht endgültig entschieden werden. Anhand von Messungen an GC für Doppelschichtkondensatoren ist aber davon auszugehen, dass CuO Ursache dieses Verhaltens ist [4].

11.2.2.2.4 Aluminiumelektrode

Auch bei der Untersuchung der anodischen Stabilität der ethylacetathaltigen Elektrolytlösung an Aluminiumelektroden wird eine geringe Zunahme der Stromdichten, vor allem bei niedrigen Potenzialen, festgestellt.

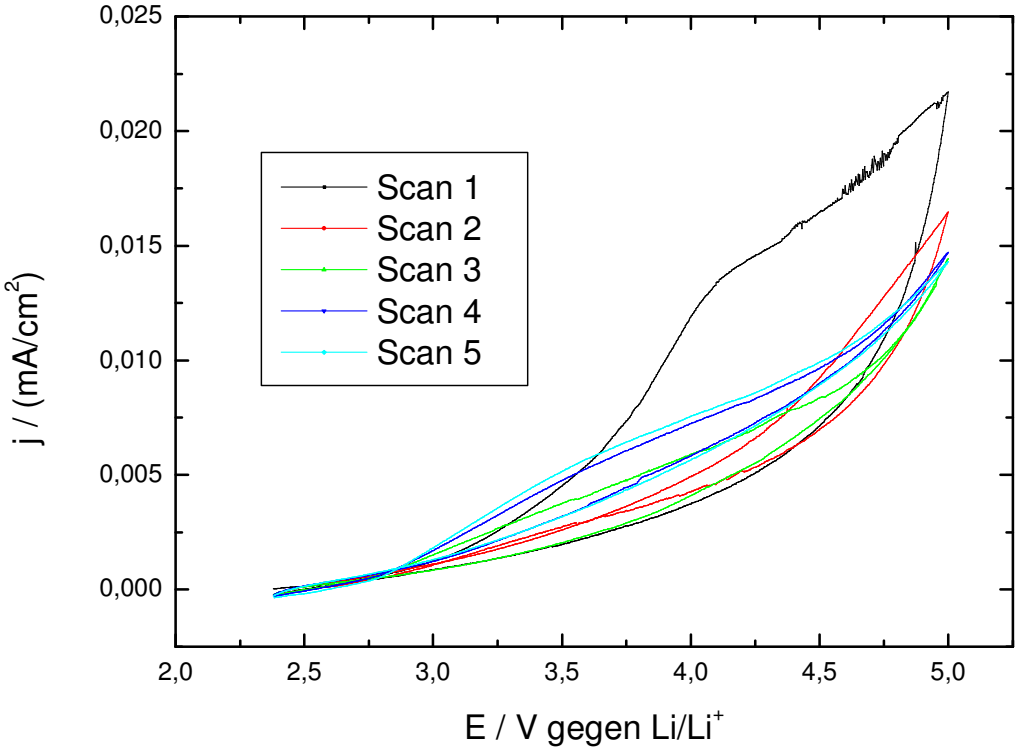


Abbildung 60 anodische Stabilität von LiBOB in EC/PC/DMC/EA an Aluminium

Das Cyclovoltammogramm der ethylacetathaltigen Carbonatmischung an Aluminium wurde im Bereich zwischen dem Ruhepotenzial bei 2,38 V und 5,00 V aufgenommen, und ist in Abbildung 60 dargestellt. Auch hier zeigt sich wie bei der EC/PC/DMC Mischung, siehe Abschnitt 11.2.2.1.3, eine Abnahme der Grenzströme.

11.2.3 Bestimmung der Verunreinigungen der LiBOB Chargen

11.2.3.1 CV-Messungen

In der Diplomarbeit des Autors [20] wurde eine Reihe potenzieller Verunreinigungen als Ursache für die kathodischen Peaks bei ca. 1,3 V und ca. 2,5V, beziehungsweise für die anodischen Peaks bei ca. 1 V, 2,3 V und 3 V vorgestellt, die in Cyclovoltammogrammen an Platinelektroden auftreten. Platin wurde als Elektrodenmaterial ausgewählt, obwohl es kein typisches Material für Batterieelektroden ist, da an Platin aufgrund seiner katalytischen Eigenschaften besonders leicht Zersetzungsreaktionen von etwaigen Verunreinigungen beobachtet werden können. LiBOB wird bei der Reinigung vom Hersteller in Essigsäureethylester bzw. Acetonitril umkristallisiert. Durch geeignete Experimente wurde der Einfluss dieser Lösungsmittel auf die Cyclovoltammogramme an Platin ausgeschlossen [20]. Auch Monolithiumoxalat und Lithiumglyoxylat, die als Verunreinigungen im LiBOB enthalten sein könnten, wurden als Ursache für diese Peaks ausgeschlossen [20]. Da nach der Zugabe von Wasser beziehungsweise Luft eine Veränderung dieser Peaks auftrat, liegt es nahe anzunehmen, dass dadurch diese Peaks verursacht werden.

11.2.3.1.1 Literaturübersicht

Die ersten Untersuchungen über Elektrochemie von Wasserspuren in Elektrolytlösungen für Lithium-Ionen-Batterien wurden von Burrwos und Kirkland [5] durchgeführt. In dieser Arbeit wird die Auswirkung von Wasserspuren in $LiClO_4$/PC Lösungen untersucht. Durch Zugabe von Wasser wurde nachgewiesen, dass sich die Höhe des Peakpaars bei 1,3 V / 2,3 V vergrößert. Das Peakpaar bei 0,4 V und 1,3 V wird einer Lithiumabscheidung auf Platin zugeschrieben [5].
Die meisten Arbeiten zur Charakterisierung der Prozesse, welche die beobachteten Peaks hervorrufen, wurden von Aurbach et al. durchgeführt und in [6] zusammengefasst, wobei von diesem Autor hauptsächlich Gold anstelle von Platin als Arbeitselektrode verwendet wurde. Das Verhalten von $LiAsF_6$ in Dimethoxyethan (DME) an Edelmetallen wird von Aurbach [6] beschrieben und ist in Abbildung 61 dargestellt.

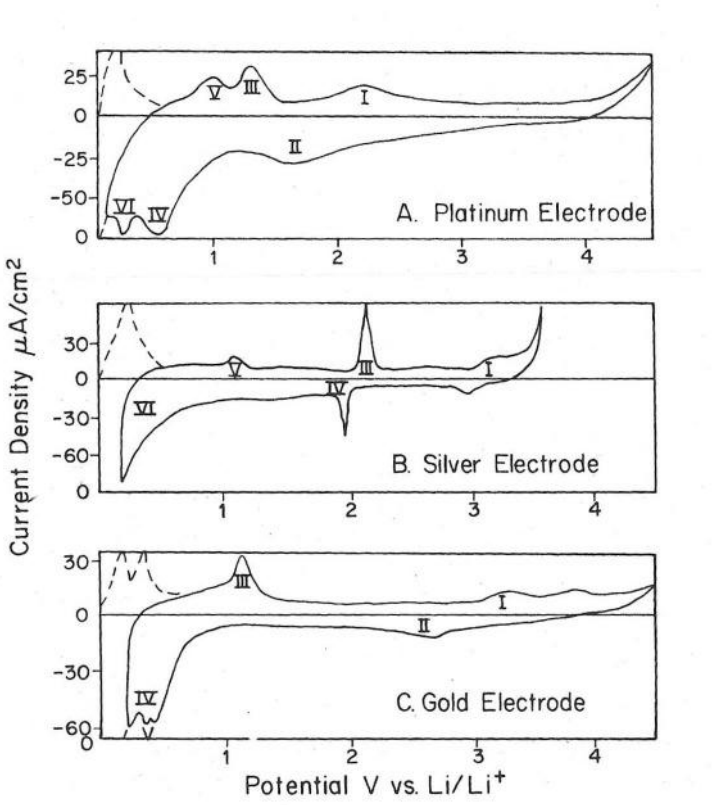


Abbildung 61 Verhalten von Verunreinigungen in CVs von $LiAsF_6$, entnommen aus [5]

Das Verhalten von $LiAsF_6$ in DME an Platin ähnelt stark dem von LiBOB an Platin. Bei Potenzialen unter 0,7V tritt Lithium UDP auf (VI und IV) [6]. Diese beiden Peaks werden auch bei LiBOB in PC/EC/DMC an Platin gefunden, wobei der Peak bei 0,7V nur als Schulter angedeutet ist [20]. Die dazugehörigen Ablösungspeaks (III) und (V) sind beim LiBOB meist zu einem Peak verschmolzen [20]. Laut Aurbach et al. [6], wird an Platin das Peakpaar (I/II) bei 1,6 V und 2,2 V durch Wasserspuren verursacht, die an der Platinoberfläche zu H_2 und LiOH reduziert werden, siehe Gleichung (39), wobei das H_2 beim Rückscan wieder zu Wasser oxidiert wird.

$$Li^+ + H_2O + e^- \rightarrow 1/2H_2 + LiOH \qquad (39)$$

Ein analoges Peakpaar ist auch bei LiBOB in PC/EC/DMC zu beobachten, wie in [20] gezeigt wird. An Gold wird ein ähnliches Peakpaar (I/II) festgestellt, das laut [6] auf die Bildung von $Au(OH)_3$ zurückzuführen ist. Bei Silber hingegen werden diese Peaks (IV/III) durch Intercalation von Lithium in die Oxidschicht der Silberelektrode verursacht [6].
Wie auch bei den Cyclovoltammogrammen von LiBOB-Lösungen an Platin unterscheidet sich der erste CV von $LiAsF_6$ in DME an Goldelektroden von den folgenden Zyklen deutlich [6]. Auch spielen Spuren von Wasser und Sauerstoff eine wesentliche Rolle. Sauerstoff verursacht einen deutlichen Reduktionspeak bei 2V, der verschwindet, wenn die Lösung mit sauerstofffreiem Argon durchgespült wird [6].
Aurbach [7] zeigt anhand der von $LiAsF_6$ und $LiClO_4$ in DME, THF und PC, dass das Verhalten von Sauerstoff kaum durch das Lösungsmittel oder durch das Anion beeinflusst wird. In dieser Veröffentlichung werden auch die Peaks des Wassers an Gold und Silberelektroden beschrieben. Mittels FTIR wird dort nachgewiesen, dass sich die Peaks, die durch das Wasser hervorgerufen werden, durch Filmbildung von LiOH erklären lassen.
Aurbach et. al. [8] beschreiben das elektrochemische Fenster von $LiClO_4$-, $NaClO_4$- und $KClO_4$-Lösungen an Goldelektroden, wobei auch auf die Verunreinigungen durch Wasser eingegangen wird. In dieser Veröffentlichung findet sich auch eine Übersichtsgraphik, die in Abbildung 62 dargestellt ist und in der die einzelnen Prozesse, die an der Goldelektrode stattfinden, aufgeführt sind.

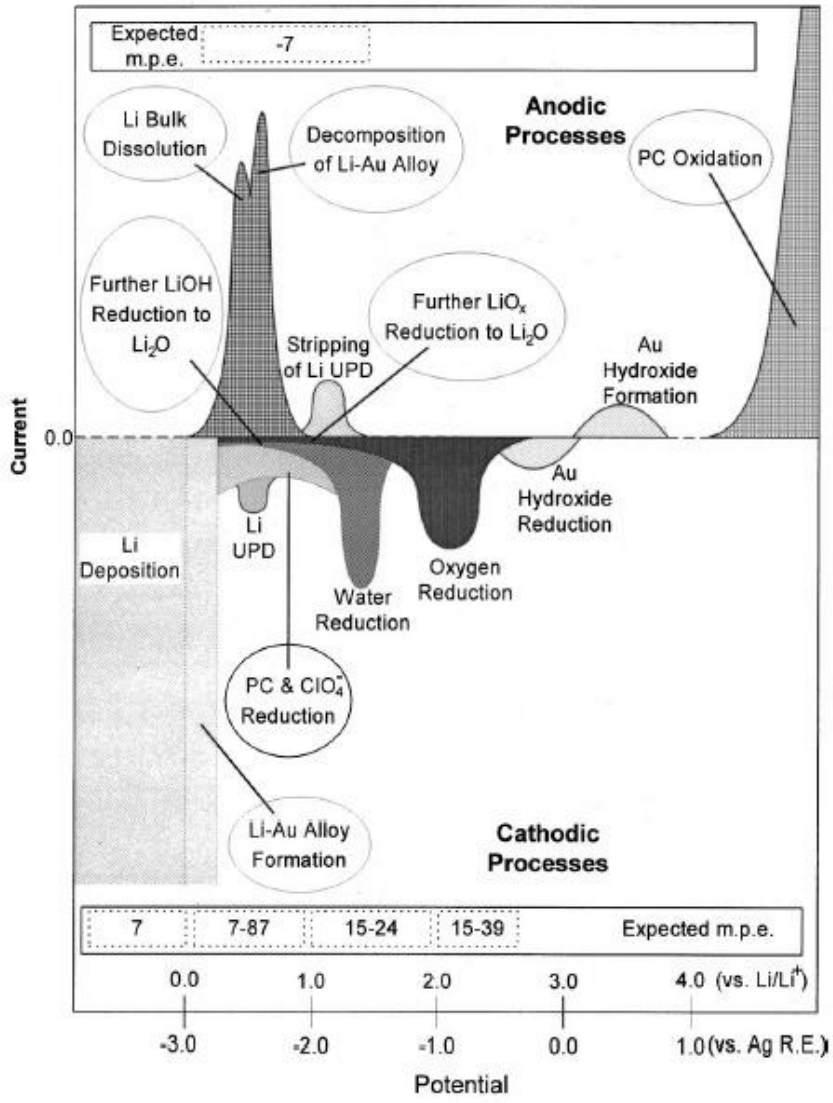


Abbildung 62 Prozesse an Au in $LiClO_4$/PC Lösungen, entnommen aus [8]

In [9] wird noch das Verhalten von Wasserspuren $LiClO_4$/γ-Butyrolacton Lösungen an Gold und Platinelektroden beschrieben, das in weiten Bereichen dem von LiBOB in PC/EC/DMC ähnelt. Aufgrund der großen Unabhängigkeit des Auftretens der Wasser- und Sauerstoffpeaks vom Lösungsmittel, vom Anion und sogar teilweise von der Edelmetallelektrode liegt es sehr nahe, dass auch bei den LiBOB-PC/EC/DMC Lösungen diese Peaks durch diese Verunreinigungen hervorgerufen werden. Verallgemeinernd lässt sich sagen, dass, wenn Wasserspuren in Elektrolytlösungen vorhanden sind, welche Lithiumionen enthalten, die gezeigten und diskutierten Peaks in CVs beobachtet werden.

11.2.3.1.2 Wasser

In der Diplomarbeit des Autors [20] ist ein Cyclovoltammogramm aufgeführt, in dem der Einfluss von Wasser untersucht wurde. Da zu diesem Zeitpunkt die Ursache für das Auftreten der Peaks noch nicht bekannt war, wurde das verwendete Wasser nicht entgast, so dass im ersten Zyklus der Sauerstoffpeak bei 2,5V auftrat.

In einem neuen Versuch wurde Milliporewasser über zwei Stunden lang mit sauerstofffreiem Stickstoff entgast. Vor der Aufnahme des CVs wurden 200µl dieses Wassers zu der üblichen Elektrolytmischung gegeben und anschließend das Cyclovoltammogramm aufgenommen, das in Abbildung 63 dargestellt ist.

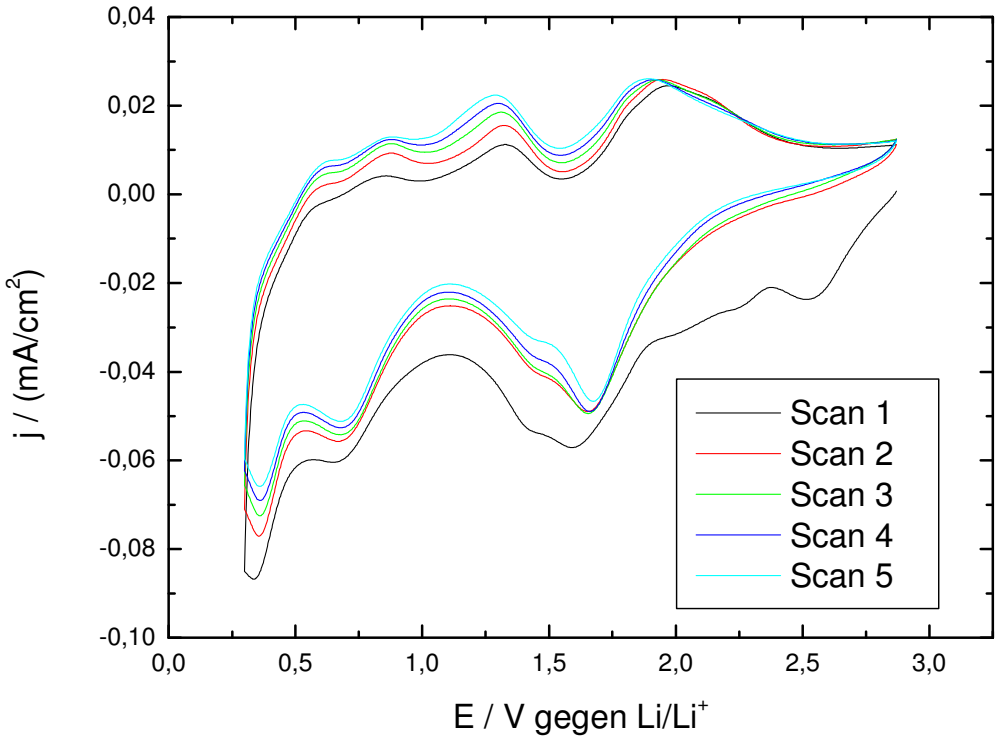


Abbildung 63 CV von LiBOB in PC/EC/DMC an Pt nach Zugabe von entgastem Wasser

Interessanterweise tritt auch hier bei 2,5 V ein kathodischer Peak auf. Diese Beobachtung legt nahe, dass auch Hydrolyseprodukte des LiBOBs einen Peak an dieser Stelle verursachen können.

11.2.3.1.3 Sauerstoff

Um die übliche Elektrolytlösung zu entgasen wurde sie in Analogie zur Literatur [6] für 30 Minuten mit hochreinem Argon gespült. Anschließend wurde mit den üblichen Bedingungen ein CV an einer Platinelektrode aufgenommen.

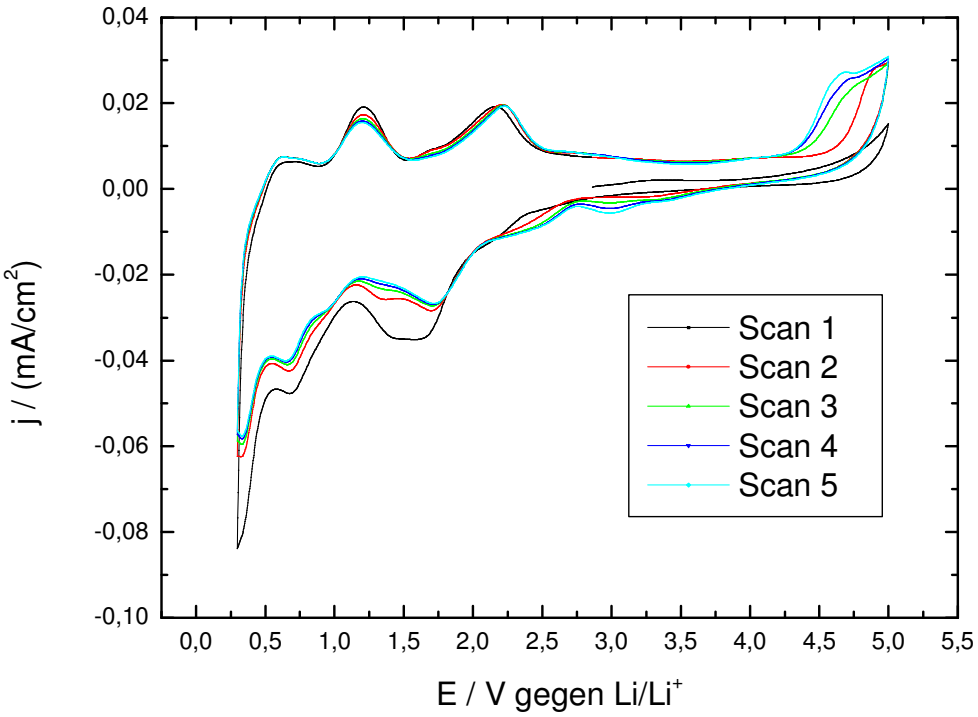


Abbildung 64 Entfernen von Sauerstoffspuren durch Spülen mit Argon

Insbesondere beim ersten Zyklus ist eine deutliche Abnahme des Sauerstoffpeaks bei 2,5 V, im Vergleich zu den in [20] aufgeführten CVs, zu erkennen. Aufgrund dieser Tatsache liegt es sehr nahe, dass der kathodische Peak bei 2,5 V durch Sauerstoff hervorgerufen wird. Mit zunehmender Zyklenzahl ist ein leichtes Anwachsen der Schulter bei 2,5 V zu beobachten, welches auf Undichtigkeiten der verwendeten Messzelle zurückzuführen ist.

11.2.3.1.4 LiBOB an Platin

In Abbildung 65 ist das Verhalten von LiBOB Lösungen an Platin dargestellt. Für diese Abbildung wurden zwei Cyclovoltammogramme, (der kathodische wie auch der anodische Ast) einer LiBOB Lösung (0,89 mol/kg in EC/PC/DMC) zusammengesetzt.

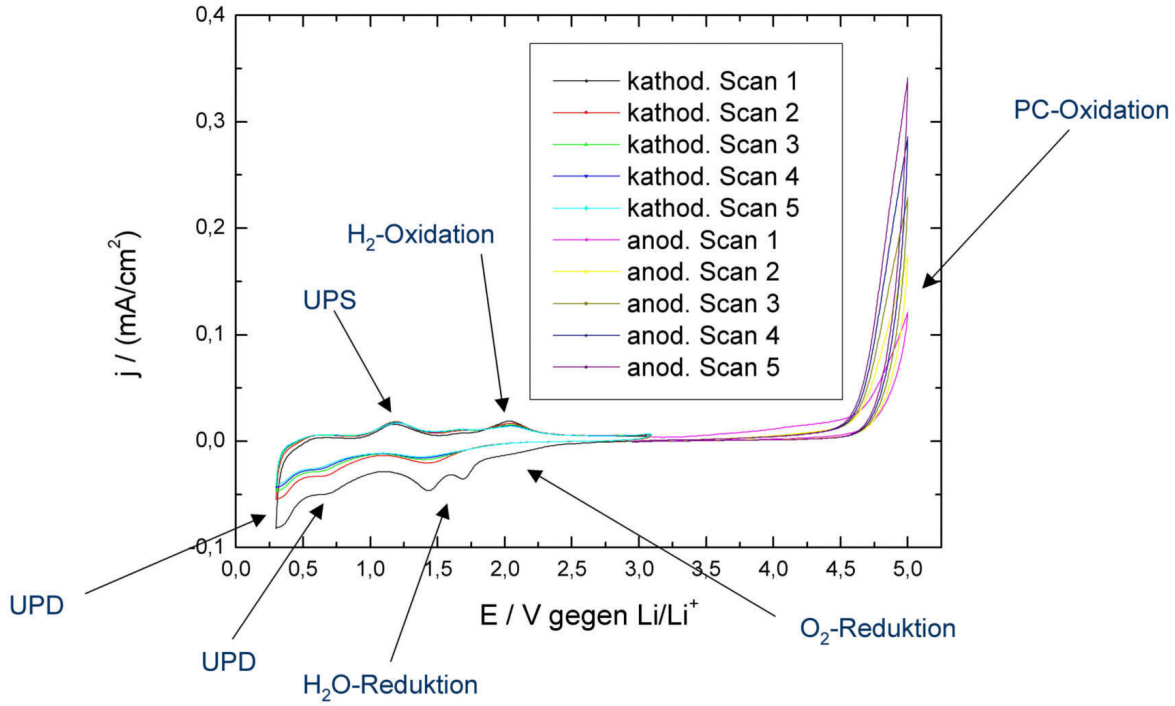


Abbildung 65 Verhalten von LiBOB Lösungen an Platin

LiBOB zeigt wie erwartet das gleiche Verhalten in der Cyclovoltammetrie an Platin wie alle Elektrolytlösungen, die Lithiumionen enthalten, welche mit Wasserspuren verunreinigt sind. Es ist somit möglich, die Ursache sämtlicher Peaks im Grundstrom-Cyclovoltammogramm von LiBOB-Lösungen zu erklären.

11.2.3.1.5 LiBOB an Gold

Um einen Vergleich mit den Ergebnissen von Aurbach zu erhalten, wurde auch von der genannten Elektrolytlösung ein CV an einer Goldelektrode aufgenommen, das in Abbildung 66 dargestellt ist.

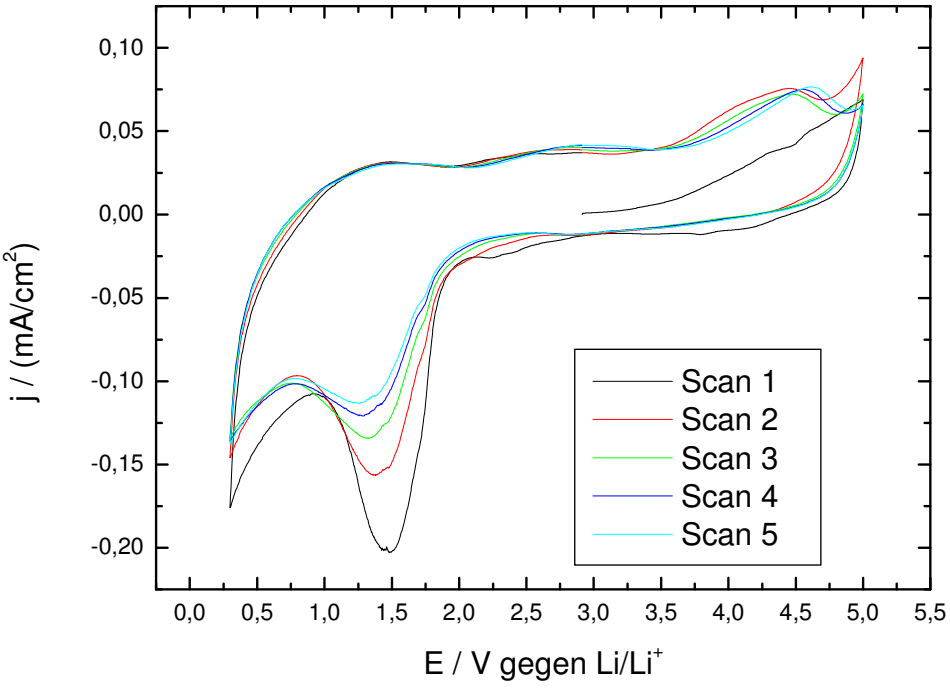


Abbildung 66 CV53_1_1 von LiBOB in PC/EC/DMC an Gold

In diesem CV fehlt, wie auch in den Messungen von Aurbach (siehe Abbildung 62), der anodische Peak des Wassers, der an Platin zu beobachten ist.

11.2.3.1.6 Ausschluss weiterer Kontaminationen

Da ursprünglich die Ursache der Peaks noch nicht bekannt war, wurde, um Verunreinigungen auszuschließen, die durch das Lithiumblech hervorgerufen werden, das sowohl als Referenz wie auch als Gegenelektrode verwendet wird, ein CV von der Standardelektrolytlösung an Platin aufgenommen, das in Abbildung 67 dargestellt ist und bei dem Lithiumblech aus einer anderen Charge verwendet wurde.

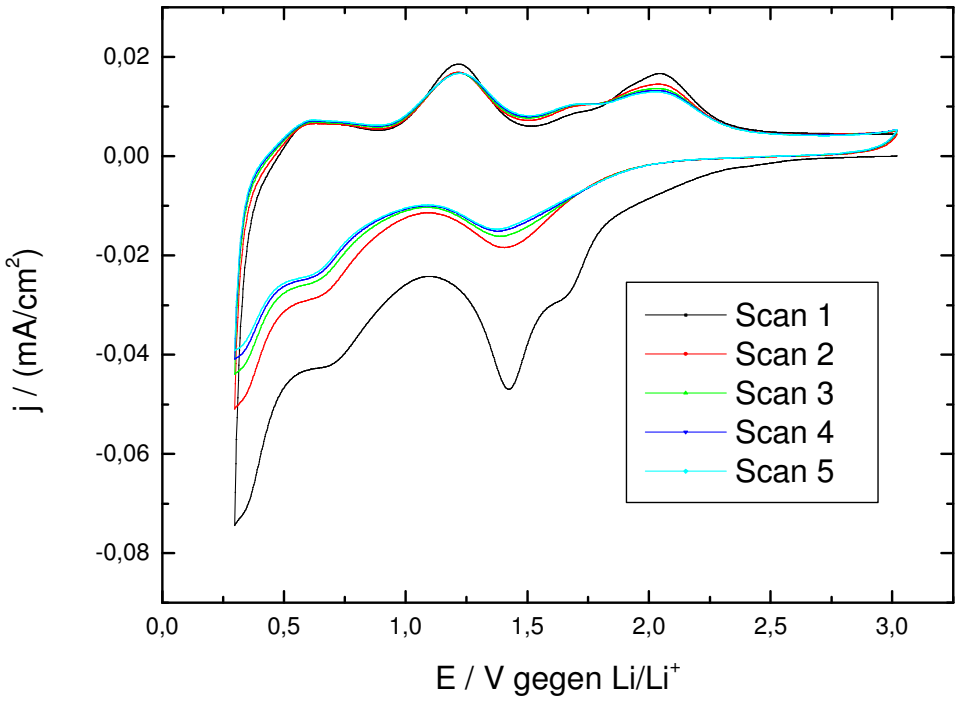


Abbildung 67 LiBOB an PC/EC/DMC, Variation des Platinblechs

Im Vergleich zu den in der Diplomarbeit des Autors [20] aufgeführten CVs tritt hier keine wesentliche Änderung auf, so dass, wie zu erwarten ist, Spurenverunreinigungen aus dem Lithium als Ursache der Peaks ausgeschlossen werden können.

11.2.3.2 NMR Messungen

11.2.3.2.1 Ausschluss potenzieller Verunreinigungen

Oxalsäure, Glyoxylsäure und deren Lithiumsalze

Um die Lithiumsalze der Oxalsäure und Glyoxylsäure in den NMR Spektren zweifelsfrei identifizieren zu können, wurden ^{1}H- und ^{13}C- NMR Spektren von diesen Substanzen in Deutero-Dimethylsulfoxid (D-DMSO) aufgenommen. D-Acetonitril, das bei den anderen NMR Spektren Verwendung fand, schied hier aufgrund der geringen Löslichkeit der Salze in Acetonitril aus. Für die Versuche wurden jeweils 100 mg Substanz in ein NMR-Rohr gegeben und in 1 cm^3 D-DMSO gelöst. Nach wiederholtem Schütteln und Absetzen lassen des nicht gelösten Anteils wurde die überstehende Lösung mit einer Pipette abgesaugt, in ein neues NMR-Rohr überführt und vermessen. Die weiteren Versuchsbedingungen sind in 11.1.4.1 angegeben.

$Li_2C_2O_4$- Dilithiumoxalat

Als Dilithiumoxalat wurde eine Probe von Fluka der Qualität p. a. vermessen.

δ/ ppm	Multiplizität	J / Hz	Integral	Zuordnung
4,00	bs		53,24	H_2O laut [10]
2,72	qi	1,85	1,00	$^{13}C_{sat}$ DMSO
2,53	s		0,27	??
2,50	qi	1,78	136,83	DMSO [10]
2,27	qi	1,85	0,98	$^{13}C_{sat}$ DMSO
2,08	s		1,67	Aceton [10]
1,94	s		0,19	$^{13}C_{sat}$ Aceton
1,23	s		1,21	Schlifffett [10]

Tabelle 24 ^{1}H NMR von $Li_2C_2O_4$

Im ^{13}C-NMR zeigen sich aufgrund der geringen Empfindlichkeit und aufgrund der geringen Konzentration der Verunreinigungen außer DMSO, dem nicht deuterierten Anteil des Lösungsmittels bei 39,4 ppm, (Septett J=53Hz) keine weiteren Peaks.

$LiHC_2O_4$-Monolithiumoxalat

Als Monolithiumoxalat wurde eine Probe des in [20] beschriebenen Salzes vermessen.

δ/ ppm	Multiplizität	J / Hz	Integral	Zuordnung
11,87	bs		1315,10	COO*H*
2,72	qi	1,85	1,00	^{13}C DMSO
2,54	s		0,27	??
2,50	qi	1,78	136,83	DMSO [10]
2,27	qi	1,85	0,98	^{13}C DMSO
2,07	s		0,31	Aceton [10]
1,23	s		0,62	??

Tabelle 25 ^{1}H NMR von $LiHC_2O_4$

Im ^{13}C-NMR werden keine weiteren Peaks, bis auf ein Singulett bei 162,7ppm ($LiHC_2O_4$) gefunden. Außerdem wird noch in D-DMSO bei 39,4 ppm (J=53Hz) ein Septett beobachtet.

$H_2C_2O_4$-Oxalsäure

Als Oxalsäure wurde eine Probe von der Firma Fluka der Qualität p. a. vermessen.

δ/ ppm	Multiplizität	J / Hz	Integral	Zuordnung
13,60	bs		10,55	COO*H*
2,50	qi	1,78	1,00	DMSO [10]
2,07	s		2,07	Aceton [10]

Tabelle 26 ^{1}H NMR von Oxalsäure

$LiHC_2O_3$-Lithiumglyoxylat

Als Glyoxylat wurde eine Probe des in [20] beschriebenen Salzes vermessen.

δ/ ppm	Multiplizität	J / Hz	Integral	Zuordnung
9,37	s		2,47	??
8,27	s		0,91	??
3,37	bs		414,30	H_2O [10]
2,51	qi	1,78	158,16	DMSO [10]
2,10	s		2,97	Aceton [10]

Tabelle 27 ^{1}H NMR Lithiumglyoxylat

Eine Aussage, ob der Peak bei 9,37 ppm oder bei 8,27 ppm vom Aldehydproton der Glyoxylsäure stammt, lässt sich nicht treffen. Im ^{13}C-NMR Spektrum wird ausschließlich der Peak des D-DMSO bei 39,4 ppm (Septett J=53Hz) gefunden.

$H_2C_2O_3$ Glyoxylsäure

Als Glyoxylsäure wurde eine Probe von der Firma Fluka der Qualität p. a. vermessen.

δ/ ppm	Multiplizität	J / Hz	Integral	Zuordnung
12,71	bs		0,80	COO*H*
9,17	s		0,11	*
6,42	bs		1,45	??
5,95	d	27	0,04	*
5,59	s		0,04	*
5,53	s		0,01	*
5,06	s		0,31	*
2,49	qi	1,78	0,24	D-DMSO [10]
2,07	s		0,80	Aceton [10]

Tabelle 28 ^{1}H NMR Glyoxylsäure

* Der Peak wird auch in [11] aufgeführt, aber dort nicht näher spezifiziert

Dimethylcarbonat

Als mögliches Lösungsmittel für die Umkristallisation für LiBOB werden in dem Patent von Chemetall [13] organische Carbonate als mögliche Lösungsmittel für die Umkristallisation angegeben. DMC besitzt neben EC, als einziges unter den für Lithium-Ionen-Batterien üblichen Carbonaten, nur ein chemisch äquivalentes Proton, das darüber hinaus eine chemische Verschiebung von 3,79 ppm aufweist [12]. EC hingegen zeigt eine chemische Verschiebung von 4,18 ppm [12] und scheidet damit als mögliche Ursache aus. Aufgrund dieser Tatsachen liegt die Vermutung nahe, dass der Peak bei 3,45 ppm durch DMC hervorgerufen wird. Um diese Möglichkeit zu untersuchen, wurden 100 mg LiBOB der Charge SCHK01/29 in 1 ml D-AN gelöst und mit 10 µl einer DMC Lösung in D-AN (c=1,11 mol/dm^3) versetzt.

Im ^{1}H-NMR Spektrum zeigt sich nach DMC-Zugabe ein neuer Peak, der im Vergleich zu den reinen LiBOB Lösungen, bei 3,70 ppm liegt. Der Peak bei 3,40 ppm bleibt weiter erhalten. Aufgrund dieser Beobachtung kann auch DMC als Ursache für den Peak bei 3,40 ppm ausgeschlossen werden.
Im ^{13}C-NMR Spektrum zeigt DMC wie erwartet je ein Singulett bei 55,57 ppm und 157,50 ppm.

Ethylenglykol

In der Literatur [10], wird für Ethylenglykol eine chemische Verschiebung von 3,51ppm angegeben. Daher liegt es nahe, dass der in [20] beschriebene Peak bei 3,45 ppm von dieser Substanz herrühren könnte. In [13] werden reduktive Verfahren zur Synthese von LiBOB beschrieben, bei denen das Glykol als Nebenprodukt aus Oxalsäure entstehen könnte. Um zu untersuchen ob dieser Peak tatsächlich von Glykol stammt, wurden 100mg LiBOB der Charge SCHK01/29 in 1 ml D-AN gelöst und mit 10 µl einer Glykol (der Firma Merck, pa. wasserfrei) Lösung in D-AN (1,6 mol/dm^3) versetzt. Zusätzlich wurden noch 10µl einer EC Lösung in D-AN (c=5,7×10^{-3} mol/dm^3) zugesetzt. Das in Abbildung 68 gezeigte NMR-Spektrum weist, wie alle NMR-Spektren der Charge SCHK01/29, eine deutliche Verunreinigung durch Essigsäureethylester auf, siehe auch Abschnitt 11.2.3.2.2.

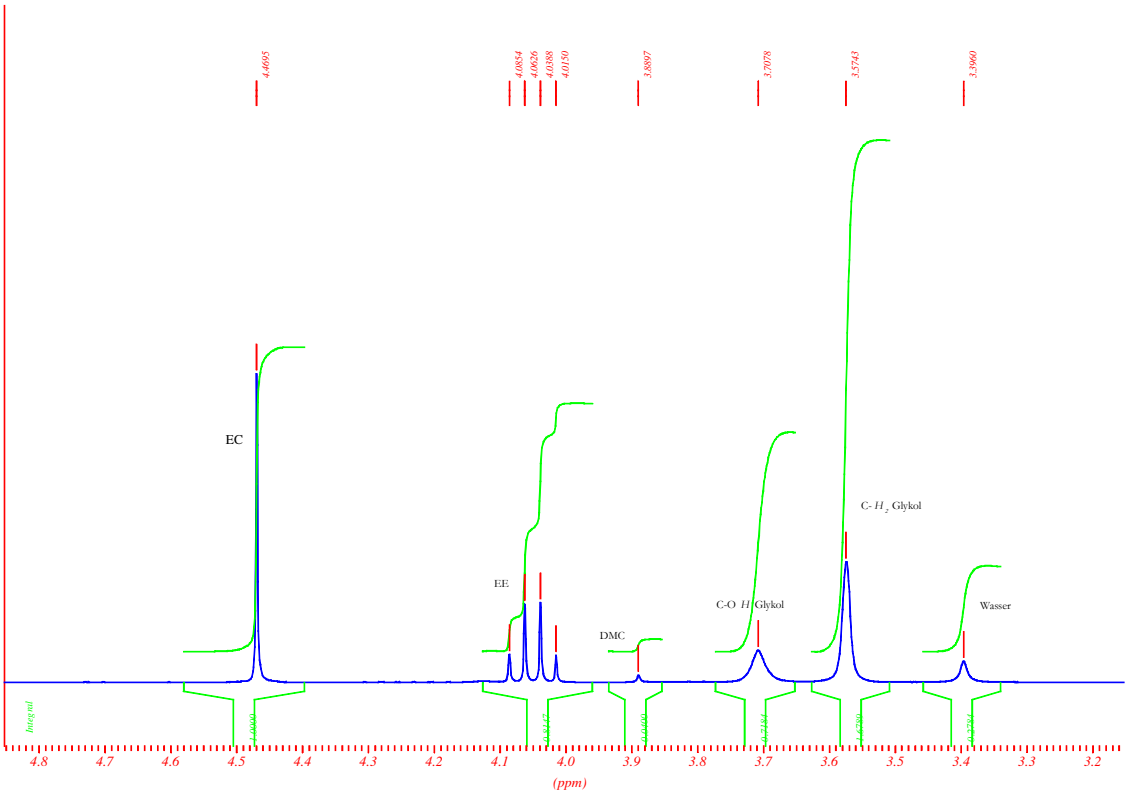


Abbildung 68 NMR von LiBOB mit Glykol, DMC, EA und EC

δ/ ppm	Multiplizität	J / Hz	Integral	Zuordnung
4,45	s		1,00	EC
4,05	q	7,0	0,81	EE

3,89	s		0,04	DMC[40]
3,71	s		0,72	O-*H* Glykol
3,57	s		1,68	C-H_2 Glykol
3,40	S		0,28	H_2O
1,98	S		1,20	EE
1,94	Qi	1,8	2,23	AN
1,19	T	7,1	1,34	EE

Tabelle 29 NMR von LiBOB mit Glykol

Im Gegensatz zur Literatur [10] zeigt das ^{1}H-NMR Spektrum zwei Peaks, bei 3,71 ppm und 3,51 ppm. Das Integralverhältnis von etwa 1:2 spricht dafür, dass es sich bei dem Peak bei 3,71 ppm um die beiden O*H* Protonen des Glykols handelt, der Peak bei 3,57 ppm hingegen durch die vier CH_2 Protonen hervorgerufen wird. Um diese Zuordnung abzusichern, wurde ein ^{1}H-NMR Spektrum einer Lösung von 10 µl Glykol in 1 ml D-AN gemessen, in dem diese beiden Peaks wieder auftraten. Nach Zugabe von 1µl D_2O zu dieser Lösung wurde erneut ein ^{1}H-NMR-Spektrum aufgenommen. In diesem Spektrum ist eine deutliche Abnahme des OH Peaks zu beobachten (Peakverhältnis 0,01 : 1). Darüber hinaus tritt ein breites Singulett bei 1,26 ppm auf, das auf das Wasser zurückzuführen ist. Aufgrund dieser Beobachtung lässt sich sicher nachweisen, dass der Peak bei 3,71ppm von Hydroxylprotonen hervorgerufen wird. Da nach Zugabe von Ethylenglykol zur LiBOB Lösung zwei weitere Peaks auftreten, die eindeutig zugeordnet werden können und der Peak bei 3,40 ppm weiterhin bestehen bleibt, kann Glykol als Verunreinigung des LiBOBs, zumindest innerhalb der Nachweisgrenze der NMR-Spektroskopie, ausgeschlossen werden.

Wasser

Die chemische Verschiebung von Wasser in Deutero-Acetonitril beträgt 2,13 ppm [10]. Die chemische Verschiebung von Wasserprotonen wird aber deutlich durch den Zusatz von Ionen beeinflusst [14], [15]. Durch eine große Konzentration an LiBOB wird dieser Peak beispielsweise zu 3,40 ppm verschoben, so dass leicht eine Fehlinterpretation (Verwechslung mit Glykol) auftreten kann. Beispielsweise wird bei allen untersuchten LiBOB Chargen der Wasserpeak bei ca. 3,4 ppm gefunden. Aus diesem Grund kann Wasser aufgrund seiner chemischen Verschiebung nicht nachweislich identifiziert werden. Eine sichere Identifikation von Wasser lässt sich nur durch ein Standardadditionsexperiment durchführen.

Um nachzuweisen, dass dieser Peak tatsächlich durch Wasser hervorgerufen wird, wurde zuerst 100 mg der Charge SCHK02/09 des LiBOB in 1 cm^3 D-AN gelöst. Die überstehende Lösung wurde vom Niederschlag abgetrennt und mit 10 µl einer EC-Lösung, (in D-AN c=55,7 mol/dm^3) versetzt, um eine Aussage treffen zu können, ob sich das Integral des Wasserpeaks verändert. Von dieser Lösung wurde dann ein NMR-Spektrum aufgenommen, dessen Peaks in Tabelle 30 aufgeführt sind.

[40] Die DMC Spuren wurden durch die Spritze eingeschleppt, mit der das Glykol zugegeben wurde; sie wurde zuvor bei dem im Abschnitt 0 beschriebenen Experiment zur Zugabe der DMC Lösung eingesetzt.

δ/ ppm	Multiplizität	J / Hz	Integral	Zuordnung
4,47	s		1,00	EC
3,44	s		0,65	H_2O
1,94	qi	1,8	7,08	AN

Tabelle 30 ^{1}H-NMR von LiBOB vor der Zugabe von Wasser

Anschließend wurde zu dieser Lösung 10 µl einer Lösung von 100mg H_2O in D-AN hinzugefügt und erneut ein ^{1}H-NMR Spektrum aufgenommen, dessen Peaks in der folgenden Tabelle aufgeführt sind.

δ/ ppm	Multiplizität	J / Hz	Integral	Zuordnung
4,47	S		1,00	EC
3,43	S		5,67	H_2O
1,94	Qi	1,8	9,09	AN

Tabelle 31 ^{1}H-NMR von LiBOB nach der Zugabe von $5{,}56\times10^{-5}$mol H_2O

Aufgrund der Beobachtung, dass kein neuer Peak auftritt, und dass dessen Fläche bei 3,43 ppm deutlich zunimmt, ist nachgewiesen, dass dieser Peak durch Wasserspuren hervorgerufen wird.

11.2.3.2.2 NMR Analysen der einzelnen Chargen

Neben den in der Diplomarbeit des Autors [20] beschriebenen Chargen standen noch vier weitere LiBOB Chargen zur Verfügung. Bei diesen Chargen handelt es sich um je drei Proben der Charge SCHK02/09, von denen je eine Probe 2002 und 2003 von den Gaia Akkumulatorenwerken und eine Probe dieser Charge, 2003 von Chemetall zur Verfügung gestellt wurde. 2004 wurde noch eine weitere Charge hochreinen LiBOBs von Chemetall geliefert.
Um einen Vergleich zwischen den einzelnen Chargen zu bekommen, wurden von jeder ein ^{1}H- und ein ^{13}C-NMR Spektrum aufgenommen. Die genauen Aufnahmebedingungen sind im Abschnitt 11.1.4.1 beschrieben. Hierzu wurden jeweils 100 mg LiBOB in 1 cm^3 D-AN gelöst und mit 10 µl eine EC Standardlösung ($c=5{,}7\times10^{-3}$ mol/dm^3) versetzt und anschließend die Spektren aufgenommen.
Als ergänzende Methode wurde noch Anionen-ESI-Massenspektrometrie eingesetzt, um Verunreinigungen zu identifizieren. Da die Elementaranalyse nicht unter Luft- und Feuchtigkeitsausschluss durchgeführt werden kann und somit die damit erhaltenen Ergebnisse stark streuen [20], wurde auf eine Anwendung dieser Methode für diese Chargen verzichtet.

Charge SCHK02/09 2002, von Gaia geliefert

In folgender Tabelle sind die mittels ^{1}H-NMR Spektroskopie gefundenen Peaks aufgeführt:

δ/ ppm/	Multiplizität	J / Hz	Integral	Zuordnung
5,32	s		0,02	
5,08	s		0,01	
4,48	s		1,00	EC
3,46	s		0,71	H_2O
2,16	qi	2,47	0,04	^{13}C-AN
1,94	qi	2,47	7,62	AN
1,71	qi	2,47	0,05	^{13}C-AN

Tabelle 32 ^{1}H- NMR von LiBOB von Gaia 2002

Welche Substanzen die Peaks bei 5,32 ppm und 5,08 ppm verursachen, ist noch nicht bekannt. Die chemische Verschiebung deutet auf Protonen hin, die sich in Nachbarschaft zu Alkengruppen befinden.

Im ^{1}H-NMR Spektrum deuten sich noch eine Reihe weiterer Peaks an, die aber etwa nur um den Faktor zwei über dem Rauschen liegen.

Aus dem Verhältnis der Integrale des EC und des Wasserpeaks lässt sich der Wassergehalt der Lösung berechnen. Wie in Abschnitt 11.2.3.2.2 angegeben, beträgt die Konzentration an EC in der Lösung, die vermessen wurde, $5{,}7\times10^{-3}$ mol/dm^3. Die Protonenkonzentration von EC $c_{H/EC}$ ist somit wegen der vier Protonen des ECs, gleich $22{,}8\times10^{-3}$ mol/dm^3. Nach dem Messergebnis aus Tabelle 32 ist das Integralverhältnis EC : H_2O = 1 : 0,71. Die Protonenkonzentration von Wasser $c_{H/H2O}$ beträgt somit $16{,}2\times10^{-3}$ mol/dm^3; die Wasserkonzentration damit $8{,}1\times10^{-3}$ mol/dm^3. Der Wassergehalt des reinen NMR Lösungsmittels wurde mit der Karl-Fischer-Titration zu 104 ppm bestimmt. Die Wasserkonzentration des Lösungsmittel beträgt daher $4{,}8\times10^{-3}$ mol/dm^3. Aufgrund der geringen Zugabemenge von EC, das darüber hinaus einen sehr geringen Wassergehalt von 10 ppm aufweist, kann das EC als Wasserquelle vernachlässigt werden. Der restliche Anteil an Wasser, also $3{,}3\times10^{-3}$ mol/dm^3, wird somit durch das LiBOB verursacht. Das Probenvolumen der Messung betrug 1 cm^3, die Stoffmenge vom Wasser somit $3{,}3\times10^{-6}$ mol, also 59 μg. Nimmt man an, dass das Wasser aus dem eingefüllten LiBOB (100mg) stammt, so bekommt man einen Wassergehalt von 590ppm. Aufgrund des hohen Wassergehaltes des NMR-Lösungsmittels ist es bei sehr kleinen Wassergehalten der Salze nicht möglich, Wasser sicher zu bestimmen. Daher kann dieser Wert nur als eine erste Näherung für den wahren Wert angesehen werden. Im Abschnitt 7.3.3.1 wird eine Methode beschrieben, bei der das NMR-Lösungsmittel so weit getrocknet wurde, so dass sein Wassergehalt keine Rolle mehr spielt. Darüber hinaus wird bei dieser Methode keine Volumenzugabe des Standards mehr durchgeführt, die im Handschuhkasten nur schwer zu bewerkstelligen ist und zu große zufällige Fehler zur Folge hat. Mittels ^{13}C NMR wurden folgende Peaks gefunden.

Aufgrund der geringen Empfindlichkeit des ^{13}C-NMR sind hier keine Verunreinigungen nachweisbar.

Das Anionen ESI Massenspektrogramm liefert den LiBOB Anionen Peak bei m/z=186,6, einen Zersetzungspeak, bei dem aus dem Bisoxalatoanion C_3O_2 entwichen ist m/z=114,6. Dianionen-Peaks bei m/z=381,0 von $2A^-+Li^+$, m/z=392,0 von $2A^-+NH_4^+$ und m/z=392,0 von $2A^-+Na^+$ [41].

δ / ppm	Multiplizität	J / Hz	Integral	Zuordung
159,90	s		1,00	LiBOB
118,59	s		14,21	D-AN
66,38	s		0,047	EC
1,49	7	20,7	13,49	D-AN

Tabelle 33 ^{13}C- NMR von LiBOB von Gaia 2002

Charge SCHK02/09 2003, von Gaia geliefert

In folgender Tabelle sind die mittels ^{1}H-NMR Spektroskopie gefundenen Peaks aufgeführt.

δ/ ppm/	Multiplizität	J / Hz	Integral	Zuordnung
5,29	s		0,02	
5,06	s		0,01	
4,47	s		1,00	EC
3,44	s		0,66	H_2O
2,16	qi	2,47	0,04	^{13}C-AN
1,94	qi	2,47	7,57	AN
1,71	qi	2,47	0,05	^{13}C-AN

Tabelle 34 ^{1}H- NMR von LiBOB von Gaia 2003

Der Wassergehalt berechnet sich nach obiger Methode zu 550 ppm. In Abschnitt 7.3 werden Gehalte präsentiert, die mit der genaueren Methode bestimmt wurden.

Auch hier finden sich die oben beschriebenen Peaks, die knapp über der Rauschgrenze liegen. Da sie nahezu das gleiche Muster aufweisen, ist anzunehmen, dass noch weitere Verunreinigungen vorliegen.

Mittels ^{13}C NMR wurden folgende Peaks gefunden.

δ / ppm	Multiplizität	J / Hz	Integral	Zuordnung
159,9	s		1,00	LiBOB
118,59	s		14,13	D-AN
66,38	s		0,05	EC
1,4	7	20,7	13,46	D-AN

Tabelle 35 13C- NMR von LiBOB von Gaia 2003

Im Vergleich zu den anderen Chargen zeigen sich keine Unterschiede.

[41] Die Verunreinigung durch Natrium und Ammonium wurde durch das Spektrometer hervorgerufen.

Das Anionen ESI Massenspektrogramm liefert den LiBOB Anionen-Peak bei m/z=186,6, einen Zersetzungspeak, bei dem aus dem Bisoxalatoanion C_3O_2 entwichen ist m/z=114,6. Dianionen Peaks bei m/z=381,0 von $2A^-+Li^+$, m/z=392,0 von $2A^-+NH_4^+$ und einen Peak bei m/z=575 von $3A^- + 2Li^+$ [41].

Charge SCHK02/09 2003, von Chemetall geliefert

In folgender Tabelle sind die mittels ^{1}H-NMR Spektroskopie gefundenen Peaks aufgeführt.

δ/ ppm/	Multiplizität	J / Hz	Integral	Zuordnung
5,29	s		0,03	
5,05	s		0,01	
4,47	s		1,00	EC
3,44	s		0,65	H_2O
2,16	qi	2,47	0,04	^{13}C-AN
1,94	qi	2,47	7,08	AN
1,71	qi	2,47	0,04	^{13}C-AN

Tabelle 36 ^{1}H- NMR von LiBOB von Chemetall 2003

Für die Peaks, die knapp über dem Rauschlimit liegen, gilt das oben Gesagte. Der Wassergehalt beträgt 540 ppm. Auch für diese Charge werden in Abschnitt 7.3 genauere Werte aufgeführt.
Mittels ^{13}C NMR wurden folgende Peaks gefunden.

δ / ppm	Multiplizität	J / Hz	Integral	Zuordung
159,90	s		1,00	LiBOB
118,59	s		14,13	D-AN
66,38	s		0,05	EC
1,68	7	20,7	13,46	D-AN

Tabelle 37 ^{13}C- NMR von LiBOB von Gaia 2003

Das ^{13}C-Spektrum ist mit den anderen Chargen identisch.
Das Anionen ESI Massenspektrogramm liefert den LiBOB Anionen-Peak bei m/z=186,6, einen Zersetzungspeak, bei dem aus dem Bisoxalatoanion C_3O_2 entwichen ist m/z=114,6, Dianionen-Peaks bei m/z=381,0 von $2A^-+Li$, einen Peak bei m/z=575 von $3A^-+2Li^+$ und einen Peak bei m/z=768 von $4A^-+3Li^+$ [41].

11.2.3.3 Charakterisierung des schwer löslichen Rückstandes

Mögliche Substanzen die im schwer löslichen Rückstand der Chargen SCHK01/47 und SCHK02/09 enthalten sein könnten, wären Glyoxylsäure, Lithiumglyoxylat, Dilithiumoxalat, Monolithiumoxalat, die als Nebenprodukte bei der LiBOB Synthese entstehen können. Als ein weiteres denkbares Nebenprodukt wäre ein Polymer aus Oxalsäure und Borsäure denkbar, ein sogenanntes „Poly-LiBOB“. Mögliche weitere schwer lösliche Verunreinigungen könnten Oxalsäure und Borsäure als nicht umgesetzte Edukte sein

Um den Rückstand der beiden Chargen zu charakterisieren, wurden nach der in Abschnitt 11.2.4.6 beschriebenen Methode jeweils 100mg isoliert. Für die Charakterisierung des Rückstandes wurden ^{1}H-, ^{11}B-, und ^{13}C-NMR Spektroskopie und Röntgenpulverdiffraktometrie eingesetzt. Der Einsatz von Massenspektrometrie scheitert am geringen Dampfdruck der Probe. Die Charakterisierung des Rückstandes erwies sich als problematisch, da er in den gängigen NMR Lösungsmitteln unlöslich ist. Aus diesem Grund wurde versucht, mit D-DMSO, D-AN, und D_2O Teile des Rückstandes zu extrahieren und mittels NMR Spektroskopie zu analysieren. Die Aufnahmebedingungen sind in Abschnitt 11.1.4 beschrieben. Bei der Charge SCHK01/47 wurden sowohl im ^{11}B-NMR Spektrum wie auch im ^{13}C-NMR Spektrum keine Peaks außer dem Lösungsmittelpeak im ^{13}C-NMR Spektrum festgestellt. Bei der Charge SCHK02/09 hingegen wurde im Extrakt mit D-DMSO ein Peak bei 0,57 ppm gefunden, der aber nicht zugeordnet werden konnte. Die Auswertung der ^{1}H-NMR Spektren ergibt keine klare Aussage.
Um weitere Aussagen über die Zusammensetzung des Rückstandes machen zu können, wurde ein Pulverdiffraktogramm der beiden Rückstände vermessen. Die Versuchsbedingungen sind in Abschnitt 11.1.5 aufgeführt. In Abbildung 69 ist das Pulverdiffraktogramm des Rückstands der Charge SCHK02/09 dargestellt.

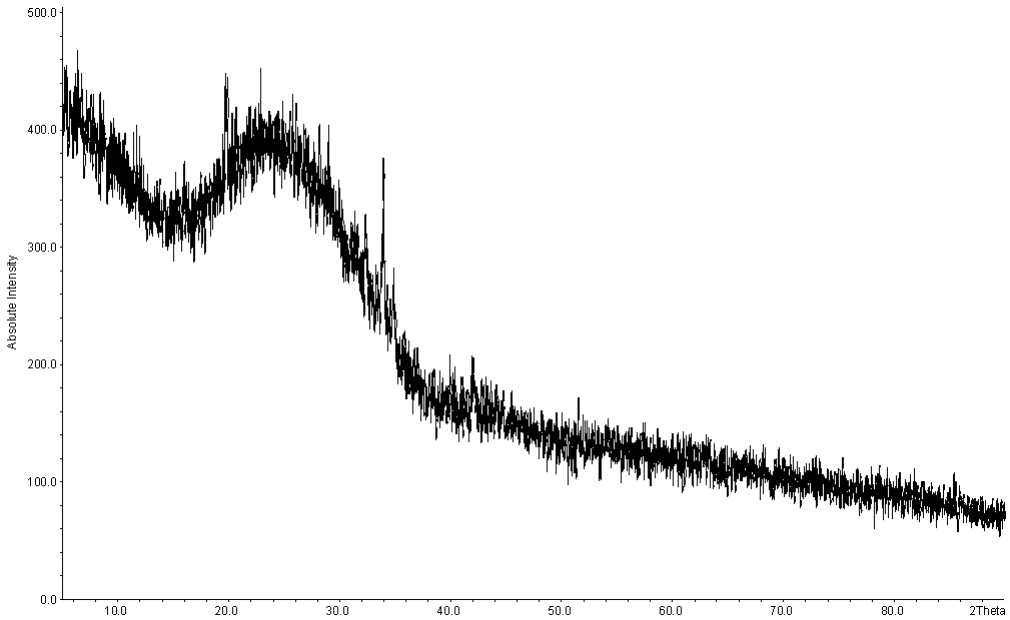


Abbildung 69 Pulverdiffraktogramm des Rückstandes der Charge SCHK02/09

In diesem Diffraktogramm sind nur wenige schwache Reflexe sichtbar, wobei eine amorphe Grundstruktur vorherrscht. Ein Vergleich mit der Strukturdatenbank liefert kein Ergebnis.
In dem in Abbildung 70 dargestellten Pulverdiffraktogramm des Rückstands der Charge SCHK01/47 sind keine Reflexe erkennbar; sie zeigt eine amorphe Struktur.

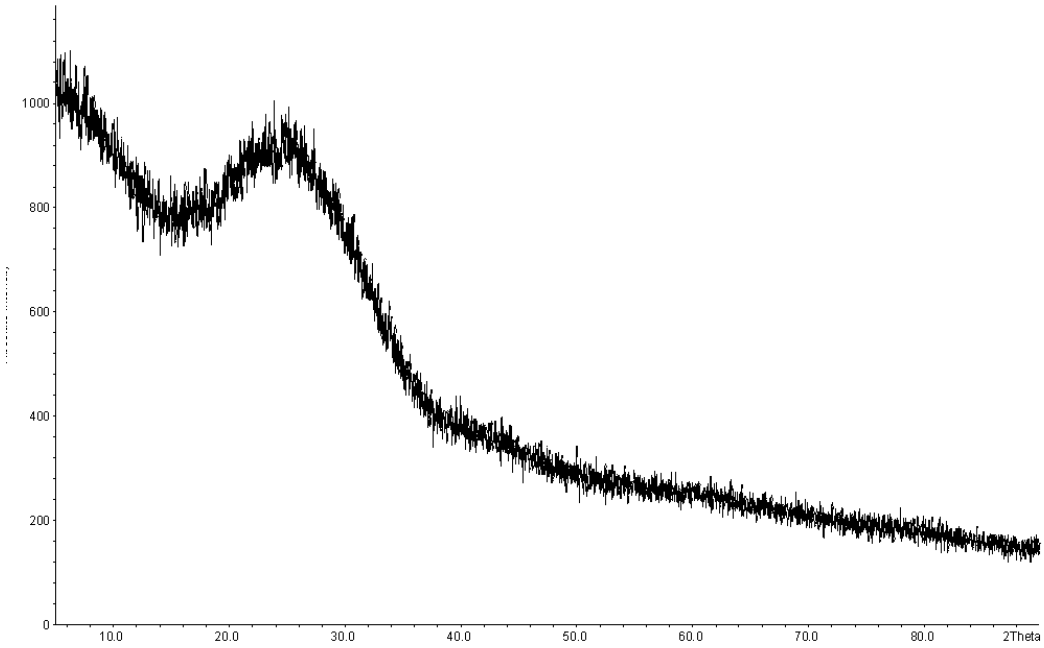


Abbildung 70 Pulverdiffraktogramm des Rückstandes der Charge SCHK01/47

Auch mit diesem Ergebnis ist keine Identifizierung des Rückstandes möglich. Um die Rückstände weiter zu charakterisieren wurde versucht, sie mittels Anion-ESI Massenspektrometrie zu untersuchen. Da sich aber diese Rückstände unter den gegebenen Bedingungen nicht verdampfen lassen, können sie mit dieser Methode auch nicht analysiert werden. Alternative Methoden zur Charakterisierung dieser Rückstände wären Feststoff-NMR Spektroskopie und Ionenchromatographie. Leider stehen diese Methoden nicht zur Verfügung. Da es inzwischen gelungen ist, LiBOB Chargen ohne den Rückstand herzustellen [16], wurde auf eine weitere Untersuchung verzichtet.

11.2.4 Reinigungsmethoden für LiBOB

11.2.4.1 Chromatographie

Die Chromatographie über Aluminiumoxid ist eine Möglichkeit zur Entfernung von Wasserspuren aus Elektrolytlösungen [17] bis [19]. Um diese Methode zu testen, wurde eine Lösung von LiBOB, Charge SCHK01/2 siehe Abschnitt 7.2, mit 0,60 mol/kg_{LM} in einer Mischung aus PC/DMC (1:2) über neutrales Aluminiumoxid der Firma Woelm, im Argon-Handschuhkasten, chromatographiert. Hierzu wurde eine Säule mit einem Durchmesser von 3cm und einer Länge von 10cm eingesetzt. Das Eluat wurde mit einem Leitfähigkeitsdetektor untersucht und nach Leitfähigkeiten in Fraktionen unterteilt. Von der Fraktion mit der höchsten Leitfähigkeit wurde, um die Reinigungswirkung zu untersuchen, mit den in Abschnitt 11.1.2 aufgeführten Bedingungen an einer Platinelektrode ein CV aufgenommen, welches in Abbildung 71 dargestellt ist.

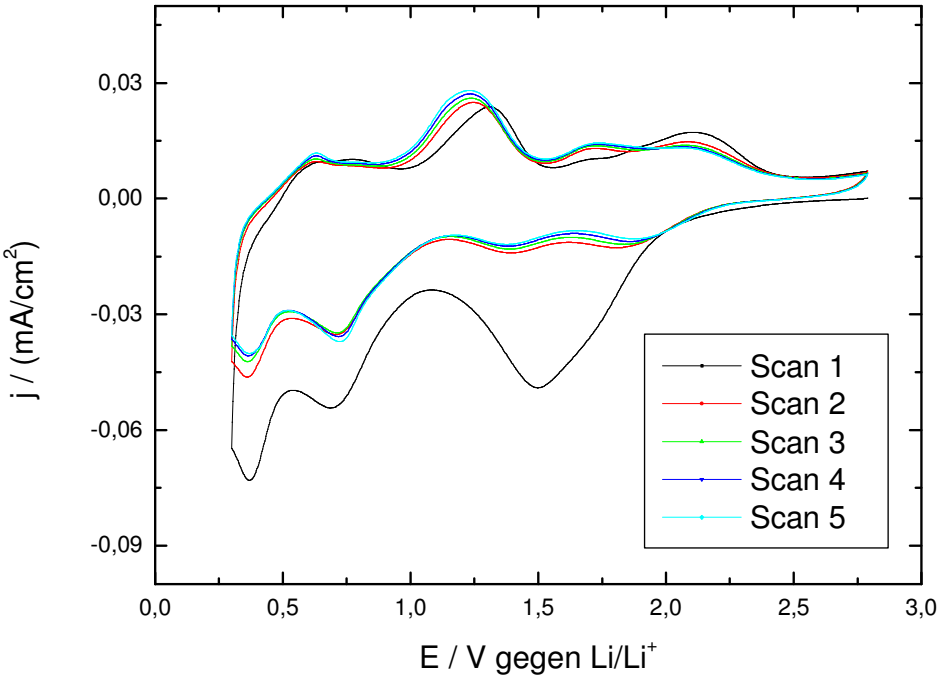


Abbildung 71 LiBOB in PC/DMC an Pt nach Chromatographie

Ein Vergleich mit den in Abschnitt 11.2.3.1 gezeigten Cyclovoltammogrammen zeigt keinen wesentlichen Unterschied. Das für Wasserspuren charakteristische Peakpaar bei 1,5 ppm und 2,0 ppm ist noch immer vorhanden. Aufgrund dieser Beobachtung kann geschlossen werden, dass diese Reinigungsmethode für dieses Problem keine befriedigenden Ergebnisse liefert.

11.2.4.2 Umkristallisation

Um die LiBOB-Charge SCHK01/29, siehe Abschnitt 7.2, noch weiter zu reinigen und die Reinigungswirkung durch Umkristallisation zu testen, wurde die LiBOB Charge SCHK01/29 dreimal umkristallisiert. Laut Information von Chemetall wurde diese bereits vorher mit dieser Methode gereinigt, so dass das Salz insgesamt viermal umkristallisiert worden ist.
Wie in [20] beschrieben, wurde Acetonitril Selectipur Merck als Lösungsmittel eingesetzt, das einen Wassergehalt von 20 ppm aufweist. Der Wassergehalt dieser Charge betrug vor der Umkristallisation 0,3 Gew%. Bei den drei Umkristallisationen wurde wie folgt vorgegangen. Die Umkristallisationen wurden bei einer Badtemperatur von 110°C durchgeführt (Siedpunkt der LiBOB-Lösung in AN). Als optimal erweist sich dabei, für 1 g umzukristallisierendes Salz 5 g Acetonitril einzusetzen. Aufgrund der geringen Lösungsgeschwindigkeit des LiBOB in Acetonitril ist es nötig, für mindestens eine Stunde bei dieser Temperatur zu rühren. Nach dem langsamen Abkühlen auf Raumtemperatur, circa vier Stunden, und drei Tagen bei –18°C fielen klare stäbchenförmige Kristalle aus, die durch Abdekantieren von der überstehenden Lösung abgetrennt wurden. Diese Kristalle wurden anschließend für 24 h im Ölpumpenvakuum bei Raumtemperatur getrocknet. Als Ausbeute wurden jeweils zwischen 65-80% erzielt. Um die Reinigungswirkung der Umkristallisation zu überprüfen, wurde nach jeder Umkristallisation ein ^{1}H-NMR unter mit den in Abschnitt 11.1.4.1 angegebenen Bedingungen gemessen.

Die Charge SCHK01/29 enthält Spuren von Ethylacetat als Verunreinigungen [20]. Schon nach einer Umkristallisation sind diese Spuren mittels ^{1}H-NMR Spektroskopie nicht mehr nachweisbar. Im Gegensatz zum Rohprodukt, werden nach der ersten Umkristallisation Spuren von Phtalsäureester (^{1}H-NMR (300 MHz, D-AN): δ [ppm] = 0,88 (t, 2H,CH_3), 1,29 (m, 2H, CH_2), 4,73 (t, 2H, OCH_2), 7,60 (m, 3H, Aromat), 7,70 (m, 3H, Aromat)), gefunden. Zur Kontamination durch Weichmacher siehe auch Abschnitt 11.2.5.1. Neben dem Wasser-Peak, wird auch wie im Rohprodukt ein nicht zuordenbares Singulett bei 5.27 ppm gefunden. Die Ursache für die Kontamination mit Phtalsäureester, die als Weichmacher eingesetzt werden, ist unbekannt. Das Lösungsmittel, das zur Umkristallisation eingesetzt wurde, scheidet als Ursache aus, wie mittels eines NMR Spektrums nachgewiesen worden ist. Der Wassergehalt des umkristallisierten Produkts blieb im Vergleich zum Rohprodukt konstant.
Durch eine weitere Umkristallisation wurde der Gehalt an Phtalsäureester reduziert und die unbekannte Verunreinigung, die den Peak bei 5,27 ppm hervorruft, entfernt.
Nach der dritten Umkristallisation ist der Weichmacher wieder entfernt worden, der Wassergehalt blieb konstant.
Die Methode der Umkristallisation ist aufgrund der starken Neigung von LiBOB Addukte mit Wasser zu bilden, ungeeignet, Wasser aus LiBOB zu entfernen. Diese Beobachtung ist wegen der geringen Donorzahl des AN auch zu erwarten. Die unbekannte Verunreinigung, die den Peak bei 5,27 ppm verursacht, wird durch wiederholte Umkristallisation entfernt.

11.2.4.3 Trocknung im Hochvakuum

In der Diplomarbeit des Autors [20] wird das Trocknen mit Hochvakuum beschrieben, wobei sich das Salz bei Temperaturen von 140°C und höher zersetzt. Aufgrund der Beobachtung, dass der Wasserpeak bei 3,40 ppm verschwindet, wenn das Salz für mehrere Tage bei 180°C getrocknet wird [20], liegt es nahe, die Trocknungsmethode mit Hochvakuum nochmals zu untersuchen. Da aber bei diesen hohen Temperaturen eine Zersetzung des Salzes auftritt, ist es sinnvoll zu versuchen, das Salz bei tieferen Temperaturen zu trocknen. Aus diesem Grund wurde eine Versuchsreihe durchgeführt, bei dem das LiBOB, Charge SCHK01/29 von Chemetall, siehe Abschnitt 7.2, nach und nach immer höheren Temperaturen ausgesetzt wird. Vor jeder Temperaturerhöhung wurde mittels ^{1}H-NMR der Wassergehalt bestimmt. Für die Trocknungsversuche wurde die in Abschnitt 11.2.4.3 beschriebene Apparatur eingesetzt. Der Druck betrug bei diesen Versuchen 10^{-5} mbar. Um festzustellen, ob schon bei Raumtemperatur Wasser austritt, wurde zuerst das LiBOB für zwei Tage bei Raumtemperatur getrocknet. Der Wassergehalt lag danach schon unter der Nachweisgrenze, davor waren es 590ppm. Anschließend wurde das LiBOB für weitere vier Tage bei Raumtemperatur getrocknet. Der Wassergehalt lag immer noch unter der Nachweisgrenze. Daraufhin wurde die Temperatur auf 40°C gesteigert und das LiBOB für weitere drei Tage im Hochvakuum getrocknet. Auch nach der Temperatursteigerung waren im ^{1}H-NMR keine neuen Peaks entstanden, also tritt bei 40°C noch keine Zersetzung auf. Nun wurde die Temperatur auf 60 °C erhöht und das LiBOB erneut für weitere drei Tage getrocknet. Der Wassergehalt lag, wie zu erwarten, immer noch unter der Nachweisgrenze, im ^{1}H-NMR war noch keine Zersetzung beobachtbar. Nach einer Steigerung auf 80 °C und nach weiteren drei Tagen war immer noch kein Wasser im LiBOB nachweisbar. Auch hier war im NMR noch keine Zersetzung zu sehen. Auch wurde keine Verfärbung beobachtet, die in der Diplomarbeit des Autors [20], bei der Trocknung bei 180 °C beschrieben wird. Die Trocknung im Hochvakuum ist also eine geeignete Methode für die Trocknung von LiBOB. Durch die schrittweise Steigerung der Temperatur im Laufe der Zeit lässt sich auch eine mögliche Zerset-

zung von LiBOB vermeiden, die durch Wasserspuren bei hohen Temperaturen hervorgerufen werden könnte.

11.2.4.4 Trocknung über P_2O_5

Um eine alternative Trocknungsmethode zu der in Abschnitt 11.2.4.4 beschriebenen Hochvakuumstrocknung zu testen, wurden 5,3 g LiBOB der Charge SCHK02/09, siehe Abschnitt 7.2, in einem Exsiccator für sieben Tage im Ölpumpenvakuum bei Raumtemperatur getrocknet. Der Wassergehalt betrug vor dem Trocknen 590 ppm, nach dem Trocknen war er so weit gesunken, dass er mit der NMR-Methode nicht mehr sinnvoll bestimmbar war.

11.2.4.5 Wasserextraktion mit Molsieb

Als eine weitere Methode zum Trocknen von Lösungen kommt der Einsatz von Molsieb in Betracht. Um eine Kontamination des LiBOBs durch Verunreinigungen durch Molsieb zu vermeiden, wurde eine Flüssig-Flüssigextraktionsapparatur eingesetzt, die in Abbildung 72 dargestellt ist.

Abbildung 72 Apparatur zum Trocknen mit Molsieb

In den 500ml Rundkolben wurden 260ml der in Abschnitt 11.2.4.6 beschriebenen Lösung der LiBOB Charge SCHK02/09 (c=0,5 mol/dm³) eingefüllt. Diese Lösung wurde gewählt, da bei ihr schon der schwerlösliche Rückstand abgetrennt war. In den Flüssig-Flüssigexstraktor wurde 4Å-Molsieb gegeben, der zuvor für fünf Tage bei 300°C im Ölpumpenvakuum und für zwei Tage bei derselben Temperatur im Hochvakuum getrocknet worden war. Mittels eines Ölbades wurde die LiBOB/Acetonitrillösung bis zum Rückfluss erhitzt (Badtemperatur 125°C). Der Rückflusskühler war mit einem Blasenzähler und mit einem mit P_2O_5 gefüllten Trockenrohr verschlossen. Durch den Rückfluss am Kühler füllte sich dass Gefäß des Flüssigextraktors mit Acetonitril, das dort seinen Wassergehalt an das Molsieb abgeben konnte. Da durch das Verdampfen des Lösungsmittels das Löslichkeitsprodukt der LiBOB-Lösung im Rundkolben überschritten wurde, wurde über den Rückflusskühler Acetonitril so lange ergänzt, bis sich das Salz wieder gelöst hatte. Durch den Rücklauf des Extraktors konnte das Lösungsmittel wieder zur LiBOB-Lösung zurückfließen. Um eine ausreichende Trocknung zu erhalten, wurde die Apparatur zwei Tage lang betrieben. Anschließend wurde der Rundkolben vom Flüssigextraktors abgetrennt und das Acetonitril vom LiBOB mittels

Ölpumpenvakuum abgezogen. Das Salz wurde daraufhin für 24h bei Raumtemperatur im Ölpumpenvakuum getrocknet**.** Der Wassergehalt war mittels ^{1}H-NMR Spektroskopie nicht mehr bestimmbar.
Da bei der hier beschriebenen Methode eine heißgesättigte Lösung vorliegt, ist es natürlich möglich, diese auch zur Umkristallisation zu nutzen, in dem die Lösung zur Kristallisation abgekühlt wird, anstatt das Lösungsmittel sofort abzusaugen. Davon wurde hier kein Gebrauch gemacht, da hier potenzielle Wasserkontaminationen durch den langandauernden Kristallisationsprozess vermieden werden konnten. Durch die Kombination mit der Kristallisation stellt dieses Verfahren, zumindest im Labormaßstab, eine günstige Kombination zur Reinigung von LiBOB dar. Es wird nicht nur der unlösliche Rückstand abgetrennt sondern auch noch der Wassergehalt deutlich reduziert. Eine weitere Reinigung wird durch die folgende Umkristallisation erzielt.

11.2.4.6 Abtrennung des schwer löslichen Rückstandes

Sowohl die Charge SCHK02/09, wie auch die Charge SCHK01/42, siehe Abschnitt 7.2, ist mit einer in den für Batterieelektrolyten typischen Lösungsmitteln schwer löslichen, weißen Substanz verunreinigt. Ein Ziel war es, diese zu isolieren und zu charakterisieren. Um Unterschiede zwischen den beiden Chargen festzustellen, wurde der schwer lösliche Anteil aus beiden extrahiert.
Da sich der schwer lösliche Anteil nicht in Acetonitril, das LiBOB sich aber sehr gut in Acetonitril löst, wurde dieses aufgrund seines niedrigen Siedepunkts als Extraktionslösungsmittel ausgewählt, wodurch eine leichte nachträgliche Abtrennung des Lösungsmittels vom LiBOB ermöglicht wird.
46 g LiBOB wurden, unter Argon als Schutzgas, in einer 500 ml Glasflasche von Schott mit 400 g Acetonitril *Selectipur*® versetzt und mit einem Magnetrührer drei Stunden gerührt. Durch Stehen lassen über Nacht setzte sich die schwer lösliche Substanz ab, woraufhin 320 ml der überstehenden Lösung mittels einer Spritze vom unlöslichen Rückstand abgezogen wurden. Die restlichen 80 ml, die den Rückstand enthielten, wurden für eine bessere Abtrennung in eine 100 ml Flasche überführt und zwei weitere Tage gewartet, bis sich der Niederschlag absetzte. Nachdem 70 ml der überstehenden Lösung mit einer Spritze entnommen wurden, um das LiBOB komplett aus dem Rückstand zu extrahieren, wurde der Rückstand wieder mit 70 ml frischem Acetonitril versetzt. Nach zwei weitren Tagen wurden wieder die überstehende Lösung entnommen und der Rückstand erneut gelöst. Die Extraktionsschritte wurden noch drei Mal wiederholt. Nach der letzten Abtrennung wurde der feste Rückstand drei Stunden lang im Ölpumpenvakuum bei 10^{-2} mbar und anschließend im Hochvakuum für drei Tage bei 10^{-5} mbar getrocknet. Von beiden Chargen konnten 120 mg Rückstand isoliert werden. Dies entspricht etwa 0,3 %. Da sich im Laufe der Zeit aus den entnommenen Lösungen noch etwas Feststoff absetzte, dürfte der wahre Gehalt an unlöslichem Rückstand in der untersuchten LiBOB-Charge etwas größer sein.
Die Charakterisierung dieses Rückstandes ist Abschnitt 11.2.3.3 beschrieben. Mit diesem Verfahren wurden ca. 100 g reinen LiBOBs aus der Charge SCHK02/09 hergestellt.

11.2.5 Wasserbestimmung mit ^{1}H-NMR Spektroskopie

11.2.5.1 Lösungsmittel ohne Salz

Um die Reinheit der verwendeten Lösungsmittel zu überprüfen, wurde sowohl von dem Lösungsmittel der Firma Deutero wie auch von Euroisotop je ein ^{1}H-NMR Spektrum mit den obigen Bedingungen aufgenommen. Im Spektrum von D-AN von Deutero lässt sich neben dem Quintett des

Acetonitrils bei 1,94 ppm J=2,47 Hz, und dem zu diesem Peak gehörenden ^{13}C-Satelliten bei 2,16 ppm und 1,71 ppm noch ein Singulett bei 2,13 ppm, welches durch Wasserspuren hervorgerufen wird feststellen. Von Gottlieb et al. [10] wird die gleiche chemische Verschiebung von 2,13 ppm für Wasser angegeben.
Das Deuteroacetonitril von Euroisotop zeigt ebenfalls das Quintett des Acetonitrils bei 1,91 ppm mit J=2,47 Hz. Auch werden hier die zugehörigen ^{13}C Satelliten beobachtet (2,13 ppm und 168 ppm). Darüber hinaus finden sich noch ein Triplett bei 2,06 ppm mit J=1,09 Hz und ein Singulett bei 1,07 ppm, die noch nicht zugeordnet werden können.
Um festzustellen, ob Verunreinigungen durch das Trocknen nach der in Abschnitt 7.3.3.1 beschriebenen Methode in das Deuteroacetonitril eingeschleppt werden, wurde ein weiteres NMR ohne Salz aufgenommen. Die hier gefundenen Peaks sind in Tabelle 38 angebeben.

d / ppm	mult	J / Hz	Integral	Zuordnung
8,40	1		1,05	Weichmacher
7,37	2	16,20	1,99	DEP
4,45	1		1,00	EC
4,16	4	7,32	2,00	DEP
4,08	1		0,016	^{13}C Weichmacher
3,81	1		3,08	Weichmacher
3,57	1		0,014	^{13}C Weichmacher
3,16	4	7,32	0,04	
2,16	5	2,47	0,23	^{13}C AN
2,13	1		0,05	H_2O
2,08	4?	2,41	0,06	
1,94	5	2,47	42,59	AN
1,71	5	2,47	0,23	^{13}C AN
1,45	t	7,40	2,99	DEP

Tabelle 38 Verunreinigung durch das Trocknen

Wie aus der Tabelle ersichtlich ist, führt das Trocknen des Lösungsmittels mit Molsieb zu einer deutlichen Verunreinigung.[42] Der Hauptbestandteil der Verunreinigung macht Diethylpthalat (DEP) aus, das die Peaks δ=1,45 (t, 6 H), δ 4,16 (q, 4 H), δ 7,37 (AA´BB´, 4 H) verursacht. Durch den zugesetzten EC Standard kann mit den Gleichungen (31) und (32) der Gehalt an DEP zu 750 ppm bestimmt werden. Die Peaks bei 8,40 ppm und 3,81 ppm dürften auch von Weichmachern stammen. Auch bei der Umkristallisation von LiBOB, siehe Abschnitt 11.2.4.2, werden durch Acetonitril Weichmacher aus den Verschlüssen der verwendeten Flaschen herausgelöst.
Wodurch die Peaks bei 3,16 ppm und bei 2,08 ppm, die nur knapp über der Rauschgrenze liegen, verursacht, kann nicht bestimmt werden, da sie keine eindeutigen Aufspaltungen aufweisen. Die Weichmacher stammen mit Sicherheit aus der Verschlusskappe des Glassgefäßes, in dem das getrocknete D-AN aufbewahrt wurde.

[42] Dies wurde nur bei dem ersten Standard mit (1,69±0,05) mmol/kg EC (siehe Seite 8) beobachtet. Dieser blieb 3-4 Wochen im Handschuhkasten, so dass die Weichmacher in die Lösung transportiert wurden.

11.2.5.2 SCHK01/29

Die Charge SCHK01/29 ist die einzige Charge, die keinen festen Rückstand aufweist und daher direkt vermessen werden konnte. Von dieser Charge wurde je viermal eine Probe von ca. 100mg in ca. 1g Standardlösung, deren molale Konzentration an EC (1,69±0,05) mmol/kg beträgt, vermessen. In Tabelle 39 sind die eingesetzten Mengen und die erhaltenen Ergebnisse aufgeführt.

m_{Salz} / g	m_{AN} / g	INT	m_{H2O} /µg	w_{Salz} / ppm	Δw_{Salz} / ppm
0,071	0,749	0,20	9,1	128	12
0,081	0,739	0,21	9,4	117	11
0,120	0,936	0,25	14,3	119	11
0,113	0,855	0,25	13,0	115	10

Tabelle 39 Wassergehalt der Charge SCHK01/29

Der durchschnittliche Wassergehalt, $w_{Salz,}$ der Charge SCHK01/29 beträgt somit (120 ± 12) ppm. Von Chemetall wird für diese Charge ein Gehalt von 250 ppm angegeben. Der Grund für den wesentlich niedrigeren Wassergehalt, der mit dieser neuen Methode bestimmt wurde, liegt in den eingangs diskutierten Nebenreaktionen, siehe Kapitel 7.3.1.

11.2.5.3 SCHK01/47

Auch von dieser Charge wurde je viermal eine Probe von ca. 100 mg in ca. 1g Standardlösung, die eine molale Konzentration von (13,6 ± 0,2) mmol/kg EC aufweist, vermessen. In Tabelle 40 sind die eingesetzten Mengen und die erhaltenen Ergebnisse aufgeführt:

m_{Salz} / g	m_{AN} / g	INT	m_{H2O} /µg	w_{Salz} / ppm	Δw_{Salz} / ppm
0,108	0,765	0,11	41,3	383	29
0,148	0,877	0,11	45,7	309	22
0,108	0,873	0,09	38,9	360	27
0,140	0,823	0,11	44,1	315	23

Tabelle 40 Wassergehalt der Charge SCHK01/47

Der durchschnittliche Wassergehalt der Charge SCHK01/47 beträgt somit (342 ± 29) ppm. Von Chemetall wird für diese Charge ein wesentlich höherer Wasseranteil von 2200 ppm angegeben.

11.2.5.4 SCHK02/09 alt

Die Charge SCHK02/09 alt wurde Mitte 2002 von den Gaia Akkumulatorenwerken (Nordhausen) erhalten. Von dieser Charge wurde je fünfmal eine Probe von ca. 100 mg in ca. 1g Standardlösung mit einer molalen Konzentration an EC von (1,69±0,05) mmol/kg vermessen. In Tabelle 41 sind die eingesetzten Mengen und die erhaltenen Ergebnisse aufgeführt.

m_{Salz} / g	m_{AN} / g	INT	m_{H2O} /µg	w_{Salz} / ppm	Δw_{Salz} / ppm
0,113	0,775	0,75	35,4	313	28
0,095	1,077	0,42	24,2	289	26
0,094	0,795	0,59	28,6	304	28
0,116	1,090	0,52	34,5	297	27
0,139	0,841	0,75	38,4	276	24

Tabelle 41 Wassergehalt der Charge SCHK02/09 alt

Der durchschnittliche Wassergehalt der Charge SCHK02/09 alt beträgt somit (296 ± 28)ppm. Von Chemetall wird für diese Charge ein Gehalt von 650 ppm angegeben. Auch bei dieser Charge ist der mit der NMR Methode bestimmte Wassergehalt erwartungsgemäß erheblich niedriger.

11.2.5.5 SCHK02/09 Gaia

Die Charge SCHK02/09 Gaia wurde Ende 2002 von den Gaia Akkumulatorenwerken (Nordhausen) erhalten. Von dieser Charge wurde je dreimal eine Probe von ca. 100 mg in ca. 1 g Standardlösung vermessen. Die molale Konzentration an EC der Standardlösung betrug hier (13,6 ± 0,2) mmol/kg. In Tabelle 42 sind die eingesetzten Mengen und die erhaltenen Ergebnisse aufgeführt:

m_{Salz} / g	m_{AN} / g	INT	m_{H2O} /µg	w_{Salz} / ppm	Δw_{Salz} / ppm
0,155	0,825	0,95	43,1	278	20
0,092	0,793	0,98	54,0	587	45
0,117	0,808	0,90	53,7	459	34

Tabelle 42 Wassergehalt der Charge SCHK02/09 Gaia

Hier wird ein Mittelwert von (440 ± 160 ppm) erhalten. Von Chemetall wird auch für diese Charge ein Gehalt von 650 ppm angegeben. Bei dieser Messreihe streuen die Ergebnisse allerdings stark. Die Ursache hierfür könnte in Wasserspuren in den verwendeten Laborgeräten oder auch in Undichtigkeiten der Verschlusskappen der NMR Rohre liegen.

11.2.5.6 SCHK02/09 Chemetall

Die Charge SCHK02/09 Chemetall wurde Ende 2002 von Chemetall (Frankfurt am Main) geliefert. Von dieser Charge wurde je fünfmal eine Probe von ca. 100 mg in ca. 1 g Standardlösung vermessen. Die molale Konzentration an EC der Standardlösung betrug hier (13,6 ± 0,2) mmol/kg. In Tabelle 43 sind die eingesetzten Mengen und die Ergebnisse aufgeführt.

m_{Salz} / g	m_{AN} / g	INT	m_{H2O} /µg	w_{Salz} / ppm	Δw_{Salz} / ppm
0,145	0,822	0,095	38,4	262	14
0,129	0,793	0,083	32,4	251	16
0,137	0,796	0,090	35,1	259	16
0,077	0,661	0,071	22,9	297	21
0,104	0,808	0,064	25,1	241	16

Tabelle 43 Wassergehalt der Charge SCHK02/09 Chemetall

Hier wird ein Mittelwert von (262±21 ppm) bestimmt. Von Chemetall wird für diese Charge auch ein wesentlich höherer Gehalt von 650 ppm angegeben.

11.2.5.7 Standardaddition LiBOB

In Tabelle 44 sind die eingesetzten Mengen und die mittels NMR ermittelten Wassergehalte angegeben:

m_{AN} /g	m_{H2O} /g[43]	m_{LiBOB} / g	INT	w_{rec} / pm	Δw_{rec} /ppm	w_{NMR} /ppm	Δw_{NMR} / ppm
0,885	--	0,065	0,018[44]	120	16	117	9
1,015	--	0,012	0[45,46]	120	16	0	--
0,635	--	0,079	0,635[45]	120	16	116	12
0,608	--	0,100	0,608[45]	120	16	126	13
0,738	0,015	0,051	0,316[45]	217	21	195	22
0,799	0,031	0,053	0,432[45]	314	22	262	30
0,686	0,068	0,075	0,826[45]	420	21	347	37
0,741	0,062	0,051	1,071[45]	523	27	705	80
0,861	0,196	0,071	1,464[45]	1034	32	909	100
0,682	0,385	0,090	0,231[44]	1075	71	1347	103
0,529	0,561	0,062	0,310[44]	2141	147	2675	218
0,290	0,753	0,079	0,445[44]	2249	144	2893	227
--	1,042	0,132	0,601[44]	1883	112	2335	171

Tabelle 44 Standardaddition LiBOB

[43] m_{H2O} bezeichnet die Masse der Wasserstandardlösung, also Wasser und AN. Das gilt auch für die Tabellen 8 und 10.

[44] Die Messung wurde mit einem EC Standard von (13,6 ± 0,2) mmol/kg an EC und einem Wasserstandard von (2,1 ± 0,1) mmol/kg durchgeführt.

[45] Hier wurde ein EC Standard mit (1,16±0,01) mmol/kg EC und ein Wasserstandard mit (18,39±0,05) mmol/kg H_2O eingesetzt.

[46] Das Zustandekommen dieses Ergebnisses wird in Abschnitt 11.3.4.2 diskutiert.

11.2.5.8 Standardaddition $LiPF_6$

Um die Leistungsfähigkeit der Wasserbestimmung auch an anderen Salzen zu untersuchen, wurde auch eine Standardaddition von Wasser mit $LiPF_6$ als Salz durchgeführt. In Tabelle 45 sind die eingesetzte Substanzmenge, die Sollwassergehalte und die mittels NMR ermittelten Integrale des Wasser- und des HF-Peaks angegeben.

	m_{AN} /g	m_{H2O} /g	m_{LiPF6} / g	w_{rec} / pm	Δw_{rec} / ppm	INT_{H2O}	INT_{HF}
1	0,972	--	0,241	0	0	0	0
2	0,623	0,140	0,221	142	9	0	0,890
3	0,796	0,185	0,171	242	14	0,016	0,460
4	0,593	0,578	0,252	513	28	1,393	0,327
5	0,426	0,987	0,157	1405	78	1,904	0,543

Tabelle 45 Standardaddition $LiPF_6$

Im Gegensatz zu LiBOB wird bei $LiPF_6$ kein Wasserpeak gefunden, wenn als NMR-Lösungsmittel nur getrocknetes D-Acetonitril verwendet wird. Ebenso ist auch mit der Karl-Fischer-Titration kein Wasser nachweisbar. Daher ist es für $LiPF_6$ bei der Berechnung der Sollwassermenge nicht erforderlich, den Wassergehalt des Salzes zu berücksichtigen.
Im Gegensatz zu LiBOB hydrolysiert $LiPF_6$ schon bei Anwesenheit geringster Wasserspuren und bildet HF. Dieses HF tritt als ein breites Singulett bei ca. 6,8 ppm im ^{1}H-NMR Spektrum auf. In Abbildung 73, die das Ergebnis der ^{1}H-NMR Messung zeigt, die in Spalte 4 in Tabelle 45 aufgeführt ist, wird dies deutlich. Da bei der Hydrolyse natürlich Wasser verbraucht wird, vermindert sich das Integral des Wasserpeaks. Aus diesem Grund ist es für eine Wasserbestimmung erforderlich, sowohl den Wasserpeak wie auch den HF-Peak zu berücksichtigen. Da zwischen der Zugabe der Wasserstandardlösung und der Durchführung des NMR-Experiments bei den einzelnen Proben eine unterschiedlich lange Zeitdauer lag, ist die Hydrolyse bei den einzelnen Proben schon verschieden weit fortgeschritten, wodurch der in Tabelle 45 erkennbare nichtlineare Zusammenhang zwischen dem Integral des HF Peaks (INT_{HF}) und des Wassergehalts erklärbar ist. Eine Durchführung der ^{1}H-NMR Experimente nach definierten Zeitspannen nach der Zugabe ergibt eine neue Methode zur Untersuchung der Kinetik der $LiPF_6$ Hydrolyse.

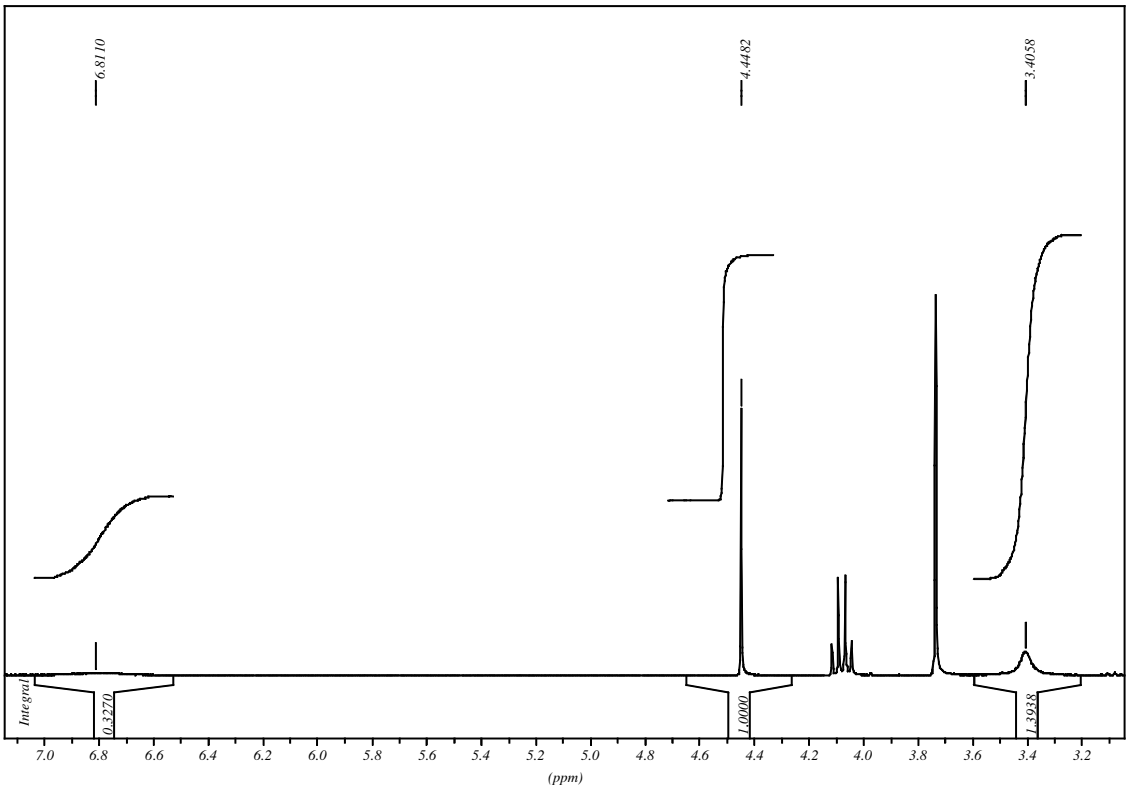


Abbildung 73 NMR $LiPF_6$ Standardaddition 3

In Tabelle 46 sind die für die $LiPF_6$-Standardaddition erhaltenen Ergebnisse dargestellt.

	w_{NMR} / ppm	Δw_{NMR} / ppm	w_{HF} / ppm	Δw_{HF} / ppm	$w_{HF, H2O}$ / ppm	$\Delta w_{HF, H2O}$ / ppm
1	0	0	0	0	0	0
2	0	0	416	35	187	16
3	5,5	0,5	357	31	161	14
4	395	3	206	17	93	8
5	1044	90	662	57	298	26

Tabelle 46 Standardaddition LiPF6

In Abbildung 74 sind in Analogie zu den LiBOB Messungen sowohl die gefundenen wie auch die Sollwassergehalte dargestellt. Aufgrund der Hydrolyse des Salzes wird hier, wenn nur der Wasserpeak berücksichtigt wird, eine zu geringe Menge an Wasser gefunden. Wird aber nun das Wasser, das durch die Hydrolyse zu HF verbraucht wird, hinzuaddiert, so ergibt sich die in Abbildung 74 dargestellte durchgezogene Linie, die im Rahmen der Messgenauigkeit eine gute Näherung des Ideals der Winkelhalbierenden darstellt.

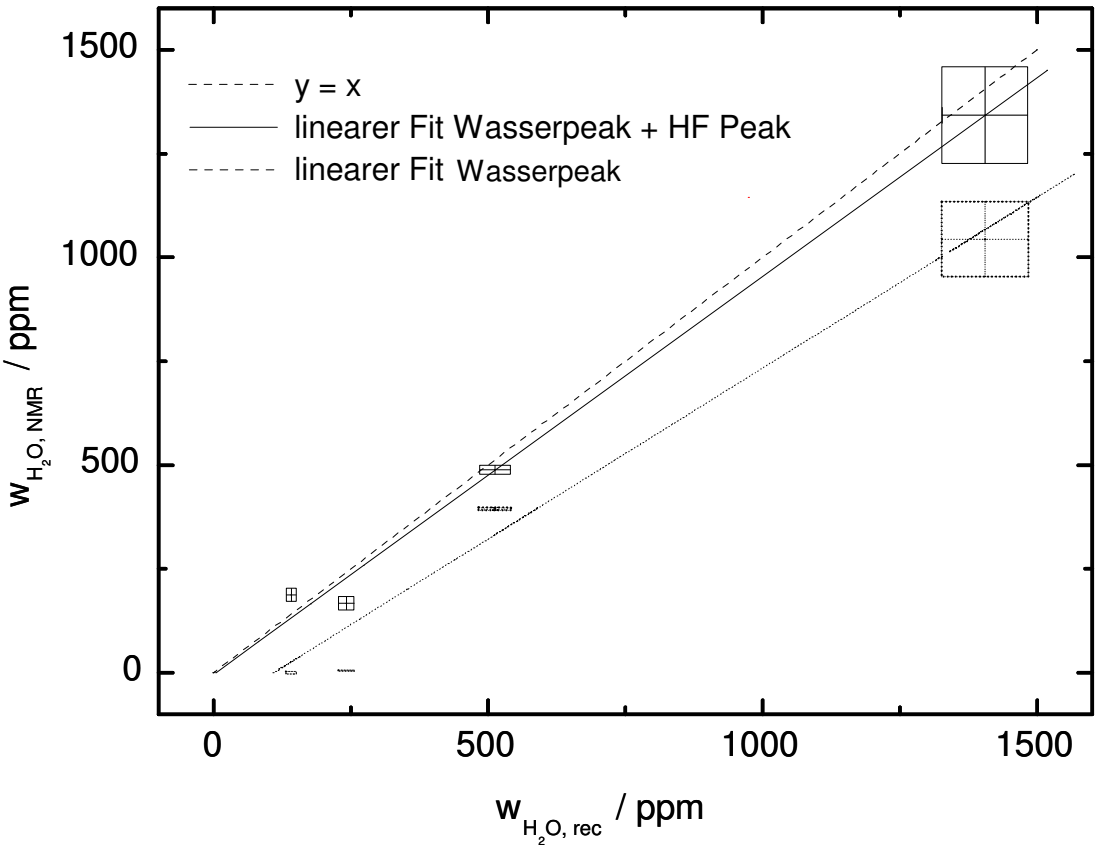


Abbildung 74 Standardaddition LiPF6

Aus der linearen Anpassung für den Gesamtwassergehalt ergibt sich hier:

$$w_{H_2O,\,NMR} = -4{,}6\,\text{ppm} + 0{,}96\; w_{NMR,\,rec} \tag{40}$$

Die Abweichung der Steigung beträgt hier im Gegensatz zum LiBOB nur noch 4%. Auch der Achsenabschnitt weicht nur noch sehr wenig von Null ab. Bei $LiPF_6$, lässt sich also die Wasserbestimmungsmethode mit ^{1}H-NMR Spektroskopie sogar noch besser einsetzen als bei LiBOB.

11.2.5.9 Standardaddition $LiClO_4$

Eine weitere Standardaddition wurde auch an $LiClO_4$ durchgeführt. Die Wahl fiel auf dieses Salz, da es nicht hydrolysiert und ausreichend gut in Acetonitril löslich ist. In Tabelle 47 sind die eingesetzten Mengen und die erhaltenen Ergebnisse dargestellt.

m_{AN} /g	m_{H2O} /g	m_{LiClO4} / g	INT	w_{rec} / pmm	Δw_{rec} / ppm	w_{NMR} / ppm	Δw_{NMR} / ppm
0,953	--	0,135	0,016[47]	66	6	55	4
0,848	--	0,049	0,107[48]	66	6	77	9
0,790	--	0,084	0,168[48]	66	6	66	7
0,708	0,149	0,055	1,370[48]	961	37	899	110
0,771	0,119	0,039	1,042[48]	1076	50	1007	121
0,727	0,440	0,105	0,224[47]	990	60	1218	91
0,689	0,654	0,096	0,269[47]	1575	96	1838	140
0,324	0,873	0,135	0,362[47]	1500	86	1571	115
0,870	0,334	0,042	2,124[48]	2688	90	2632	310
--	1,038	0,071	0,510[47]	3309	210	3645	290

Tabelle 47 Standardaddition an $LiClO_4$

In Abbildung 75 werden auch hier in Analogie zu den LiBOB Messungen die Sollgehalte an Wasser und die gemessenen Wassergehalte dargestellt.

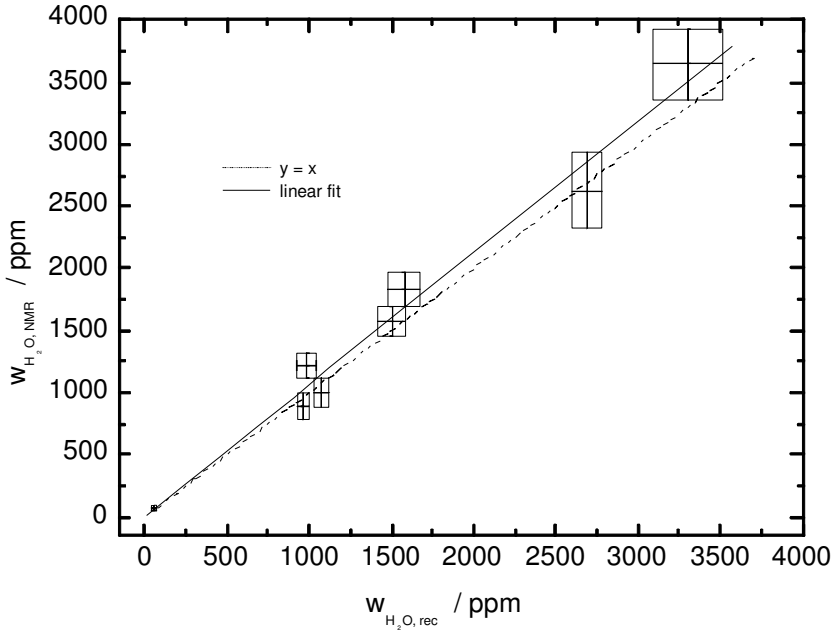


Abbildung 75 Standardaddition $LiClO_4$

[47] Die Messung wurde mit einem EC Standard von (13,6 ± 0,2) mmol/kg an EC und einem Wasserstandard von (2,1 ± 0,1) mmol/kg durchgeführt.

[48] Hier wurde ein EC Standard mit (1,16±0,01) mmol/kg EC und ein Wasserstandard mit (18,39±0,05) mmol/kg H_2O eingesetzt.

Zwischen dem vorgegebenen Wassergehalt und dem mit NMR bestimmten Wassergehalt gilt folgender Zusammenhang, der durch lineare Interpolation erhalten wurde.

$$w_{H_2O,\,NMR} = 0{,}13\,\text{ppm} + 1{,}06\; w_{NMR,rec} \tag{41}$$

Aus der Steigung der Geraden wird eine Abweichung von 6% von der Idealkurve, der Winkelhalbierenden gefunden. Die Abweichung liegt innerhalb des Bereichs, der mit der Fehlerrechnung ermittelt wird.

11.3 Abschätzung der Messfehler

Die Fehlerrechnungen in den folgenden Abschnitten wurden nach der Methode der Größtfehlerabschätzung durchgeführt. Als Fehler wurden abgesehen von Störungen, die durch das Rauschen verursacht werden, sämtliche in den Datenblättern angegebenen Unsicherheiten berücksichtigt. Störungen, die durch Rauschen entstehen, können nur schwer berechnet werden, da in den Datenblättern meist keine Angaben über die spektrale Verteilung des Rauschens gemacht werden. Darüber hinaus müsste der genaue Frequenzgang der Schaltung bekannt sein, um die Auswirkungen des Rauschens korrekt berücksichtigen zu können.
Schwankungen der Raumtemperatur beeinflussen die einzelnen Bauteile aufgrund ihres Temperaturkoeffizienten. Der typische Bereich, in dem diese Temperatur schwankt, liegt erfahrungsgemäß zwischen 20°C und 30°C. Aus diesem Grund wird in den Fehlerrechnungen ein Intervall von ±5°C um 25°C berücksichtigt. Für zeitliche Veränderungen wurde ein Zeitraum von 10.000 Stunden einkalkuliert.

11.3.1 30-Kanalthermometer

11.3.1.1 Fehlerrechnung

Ziel war es ein Thermometer zu entwickeln, das eine Genauigkeit besitzt, die im Bereich von kalibrierten Platinwiderstandsthermometern liegt. Bei der Auswahl der Bauteile wurde darauf geachtet, dass der Gesamtfehler des Messgeräts deutlich unter diesem Wert liegt, um noch genügend Spielraum für die Toleranzen des Thermistoren zu haben. Für die Berechnung der Fehler dieses Geräts, wurde die Methode der Größtfehlerabschätzung eingesetzt, wobei sämtliche Abweichungen der einzelnen Bauteile, die in ihren Datenblättern [21], [22] und [23] aufgeführt sind, in Betracht gezogen wurden. Da hier keine Angaben über die spektrale Verteilung des Rauschens gemacht werden, wurde dieses bei den einzelnen Berechnungen nicht berücksichtigt, sondern stattdessen das Gesamtrauschen des Systems experimentell bestimmt, siehe Anhang 11.3.1.2. Die Toleranzen der einzelnen Bauteile müssen nicht berücksichtigt werden, da sie durch die Kalibrierung, bei der Thermistoren und Messkanäle zusammen kalibriert werden, ausgeglichen werden. Daher müssen bei der Fehlerrechung nur die Veränderungen der Parameter einbezogen werden.
Veränderungen der Raumtemperatur beeinflussen aufgrund der Temperaturkoeffizienten der einzelnen Bausteine das Messergebnis. Für diese Veränderungen wurde ein Bereich von ±5 °C um 25 °C angenommen. Die Langzeitstabilität wurde für einen Zeitraum von 10000 h berücksichtigt. Dies entspricht in etwa eine typischen Zeitraum zwischen zwei Kalibrierungen.

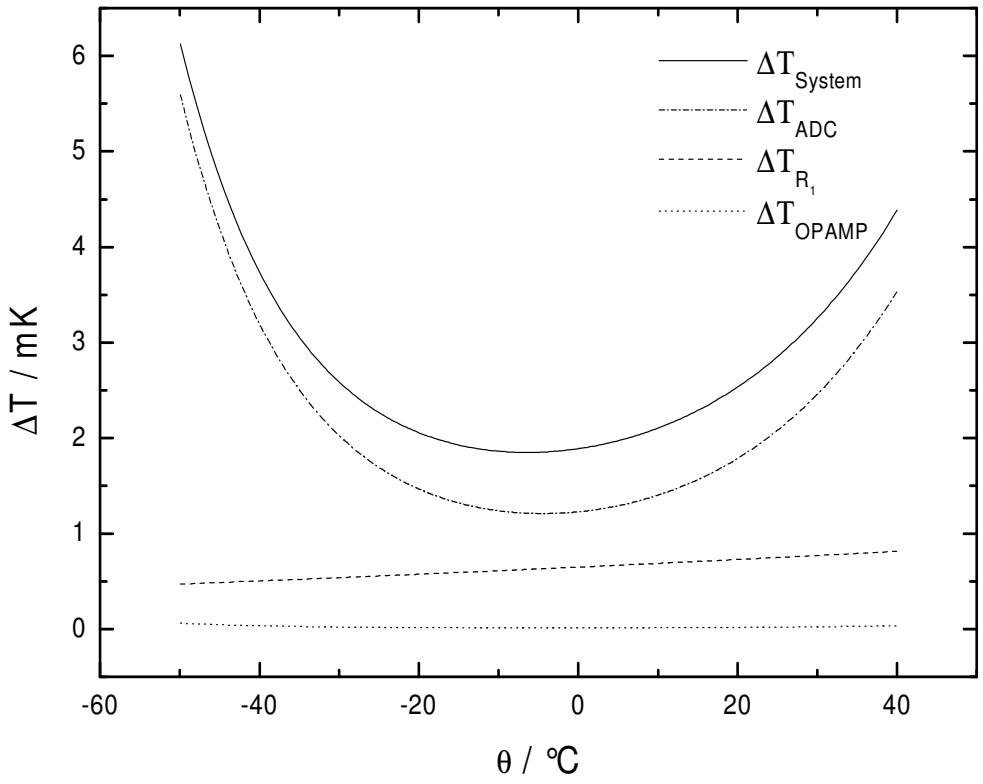


Abbildung 76 Berechnete Fehler des Thermometers und Einfluss seiner Bauteile

Dieser Fehler setzt sich aus den Fehlern der einzelnen Bausteine ΔT_{ADC}, ΔT_{R1} und ΔT_{OPAMP} zusammen. Die Werte wurden mit den in der Diplomarbeit der Autors [20] aufgeführten Methoden berechnet, wobei die veränderten Bedingungen für das 30-Kanalmessgerät berücksichtigt wurden.
Der Gesamtfehler zeigt bei –50 °C ein Maximum von 6 mK, welches deutlich unter den geforderten 25 mK liegt. Den Hauptanteil zum Gesamtfehler trägt dabei der Analog/Digital-Wandler bei. Für eine Verbesserung des Messgerätes ist es daher sinnvoll, Optimierungen zuerst an dieser Stelle anzusetzen. Da die Beiträge des Operationsverstärkers und des Referenzwiderstandes deutlich geringer sind, würde eine Verbesserung dieser Bausteine nur einen kleinen Fortschritt bringen.

11.3.1.2 Bestimmung des Rauschens

Das Rauschen des Messgeräts wurde durch Simulation der Temperatursensoren durch Widerstände bestimmt. Diese Methode ist notwendig, da die Temperaturschwankungen im Thermostaten im Bereich der hier gemessenen Größen liegen, und somit keine klare Trennung zwischen dem Rauschen des Thermostaten und des Thermometers möglich ist. Für diesen Zweck wurden Widerstände im Bereich von 20 KΩ bis 1 MΩ eingesetzt, mit denen ein Temperaturbereich von –37 °C bis +35 °C simuliert wird. Die simulierten Temperaturen wurden für 200.000 s gemessen und anschlie-

ßend die Standardabweichungen der gemessenen Temperaturen berechnet. Die Ergebnisse dieser Messungen sind in Abbildung 77 dargestellt.

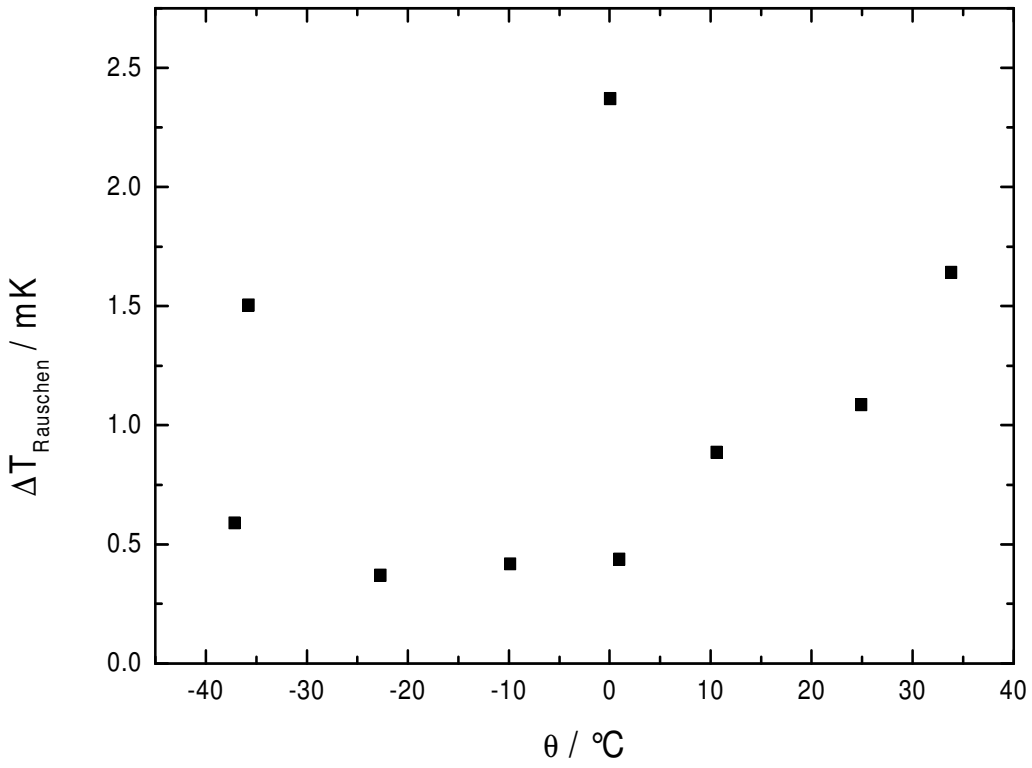


Abbildung 77 Eigenrauschen des 30-Kanalthermometers

Wie diese Messung zeigt, beträgt im gesamten Messbereich das Rauschen der Schaltung unter 2,5 mK, wobei sogar die meisten Werte unter 1 mK liegen.

11.3.1.3 Genauigkeit des Systems und Langzeitstabilität der Thermistoren

Die Langzeitstabilität des Messsystems wurde sechs Monate nach der Kalibrierung überprüft. In der Zwischenzeit wurden das Gerät und die Messfühler im routinemäßigen Messbetrieb eingesetzt, bei dem die Thermistoren ungefähr 100 Zyklen zwischen –50 °C und +40 °C ausgesetzt waren. Der Test der Kalibrierung wurde nach der gleichen Methode wie die Kalibrierung selbst durchgeführt, wobei alle 30 Messkanäle getestet wurden. Dabei wurde im Gegensatz zu dieser alle 5 °C die Temperatur gemessen, um auch Messpunkte zwischen den Stützpunkten für die Kalibrierung zu überprüfen.

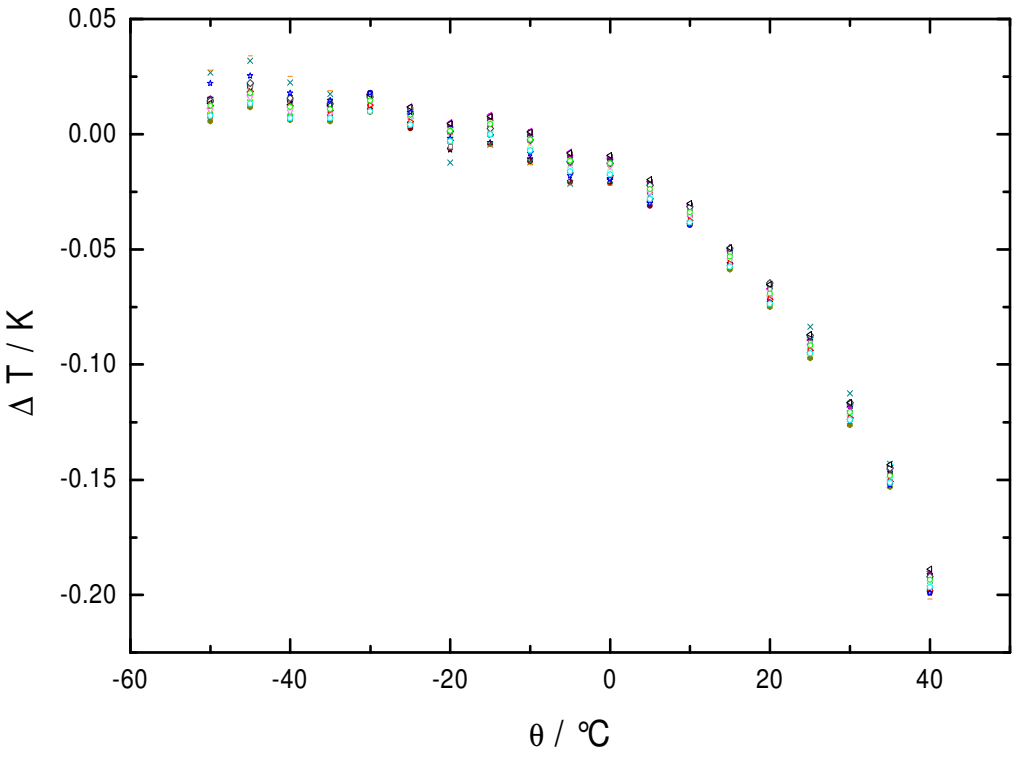


Abbildung 78 Abweichung der Messgenauigkeit nach sechs Monaten

Im Bereich zwischen –50 °C und +10 °C tritt nur eine Abweichung auf, die kleiner als 30 mK ist. In diesem Bereich entspricht die Genauigkeit des 30-Kanalthermometers damit der der Platinthermometer. Bei höheren Temperaturen tritt eine Abweichung bis zu 200 mK auf. Die Verteilung der Fehler ist für die Anwendung, für die dieses Thermometer konzipiert wurde, sehr gut, da bei dieser Anlage Systeme untersucht wurden, bei denen Phasenübergänge im Bereich von –50 °C bis –10 °C liegen.

11.3.2 30-Konduktometer

Ziel des Schaltungsentwurfes war es, mit den Messzellen einen Messbereich von 0,1 mS/cm bis 15 mS/cm abzudecken. Der Messfehler sollte dabei unter 1% liegen. Da durch die Kalibrierung, siehe Abschnitt 2.8.3, alle konstanten Abweichungen eliminiert werden, müssen für die Fehlerrechnung nur Nichtlinearitäten und Schwankungen innerhalb der Messzeit berücksichtigt werden.
Aus der Größtfehlermethode ergibt sich für den Fehler der spezifischen Leitfähigkeit folgender Zusammenhang:

$$\Delta\kappa = \sqrt{2}\cdot\left|\frac{U_{RMS}\cdot C}{R_{19}^2\cdot U_{GEN}}\right|\cdot\left|\Delta R_{19}\right| + \sqrt{2}\cdot\left|\frac{C}{R_{19}\cdot U_{GEN}}\right|\cdot\left|\Delta U_{RMS}\right| + \sqrt{2}\cdot\left|\frac{C\cdot U_{RMS}}{R_{19}\cdot U_{GEN}^2}\right|\cdot\left|\Delta U_{GEN}\right| \quad (42)$$

Der Fehler des Messwiderstandes ΔR_{19} beträgt 7,5 mΩ [49]. Für die Ausgangsspannung U_{GEN} des Sinusgenerators gilt folgender Zusammenhang, wobei Y=1/11 den Abschwächungsfaktor des Abschwächers darstellt. U_{MAX038} ist die Amplitude der Ausgangsspannung des MAX038, deren Fehler ΔU_{MAX038} durch den Temperaturkoeffizienten seiner Spannungsreferenz verursacht wird; er beträgt 0,1 mV [24].

$$U_{GEN} = Y \cdot U_{MAX\,038} \tag{43}$$

Für den Fehler der Ausgangsspannung ΔU_{GEN} der Schaltung gilt dann nach der Größtfehlermethode, wenn für den Fehler des Abschwächers Gleichung (55) eingesetzt wird.

$$\Delta U_{GEN} = |Y| \cdot |\Delta U_{MAX\,038}| + |U_{MAX\,038}| \cdot \left| \frac{R_{62}}{(R_{62}+R_{61})^2} \right| |\Delta R_{31}| + \left| \frac{R_{61}}{(R_{62}+R_{61})^2} \right| |\Delta R_{62}| + |\Delta U_{MAX\,400}| \tag{44}$$

Die Fehler der Widerstände ΔR_{61} und ΔR_{62} betragen 150 mΩ beziehungsweise 15 mΩ [49]. Der Offsetfehler des Operationsverstärkers beträgt 10 µV [25]. Mit den in Kapitel 2.8.2.1 aufgeführten Werten für die Schaltung ergibt sich für den Fehler der Ausgangsspannung ΔU_{GEN} ein Wert von 0,02 mV.

Der Fehler der Spannungsmessung setzt sich aus dem Fehler des Gleichrichters und dem des Analog/Digital-Wandlers zusammen. Der Gleichrichter verursacht durch seine Temperaturkoeffizienten einen Fehler von 0,035 % vom Messwert. Seine Nichtlinearität beträgt 0,25 % des Ergebnisses. Der Fehler des Analog/Digital-Wandlers, in Abhängigkeit des eingelesenen Bitwertes, berechnet sich mit folgender Gleichung:

$$\Delta U_{AD} = \left| \frac{1}{2^{24}} \cdot \frac{U_{ref}}{PGA} \right| \cdot |\Delta BIT| + \left| \frac{1}{2^{24}} \cdot \frac{U_{ref}}{PGA^2} \right| \cdot |\Delta PGA| + |INL \cdot U_{REF}| + |U_{Offset}| \cdot PGA + \left| \frac{1}{2^{24}} \cdot \frac{BIT}{PGA} \right| \cdot |\Delta U_{REF}| \tag{45}$$

Aus dem Datenblatt [22] lassen sich für die Fehler des ADS1241E folgende Werte entnehmen.

Fehler	Wert
Missing Codes/Bits	0
INL/%FSR	0,0015/PGA
Effektive Auflösung Bit	19
Offset drift / ppm/°C	0,02/PGA
Gaindrift ppm/°C	0,5

Tabelle 48 Fehler des ADS1241E [22]

Der Fehler der Spannungsreferenz ΔU_{REF} beträgt maximal 9 µV (siehe Abschnitt 11.3.3.1), da hier nur die kurzfristigen Schwankungen, die innerhalb der Messdauer auftreten, berücksichtigt werden müssen. Die Langzeitstabilität spielt hier keine Rolle, da für die Bestimmung der spezifischen Leitfähigkeit für jede Messung eine individuelle Kalibrierung erfolgt, siehe Abschnitt 2.8.3.

[49] Der Fehler wird nur durch den Temperaturkoeffizienten von 3ppm/°C bei einer Schwankung von 5°C der Raumtemperatur bestimmt. Die Langzeitdrift ist für die Dauer der Messung vernachlässigbar.

In Abbildung 79 sind der berechnete Gesamtfehler des Leitfähigkeitsmessgeräts und die Beiträge der einzelnen Komponenten in Abhängigkeit der gemessenen spezifischen Leitfähigkeit dargestellt. Als Zellkonstante wurde dabei ein Wert von 0,45 cm^{-1} angenommen.

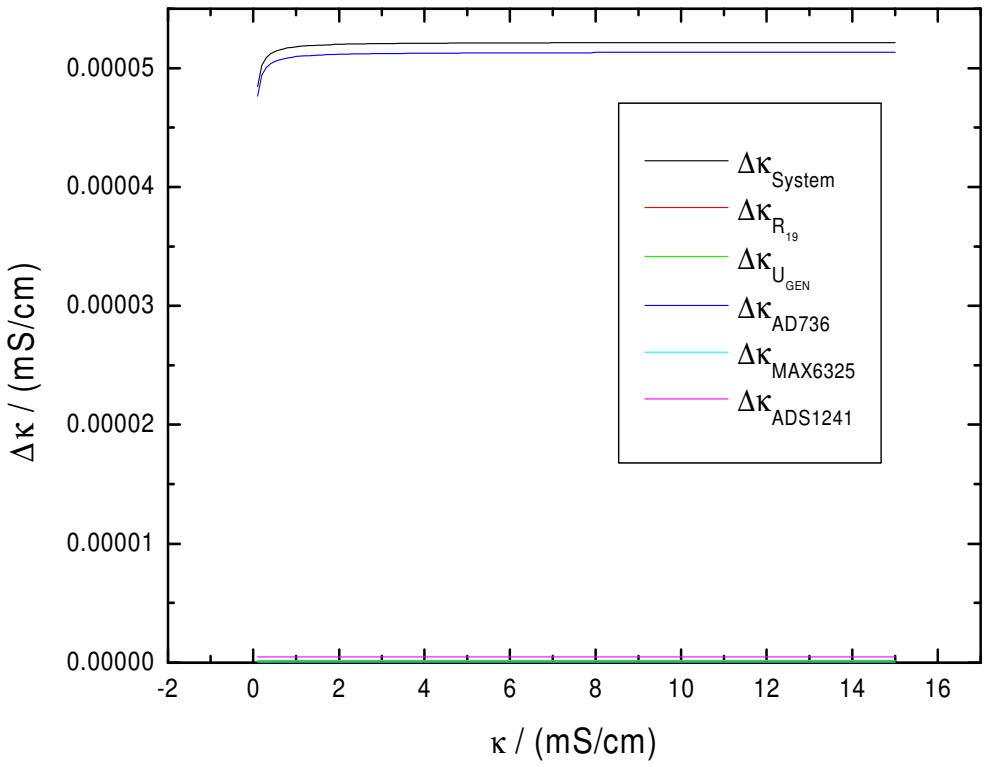


Abbildung 79 Berechneter Fehler des Leitfähigkeitsmessgerätes

Der Hauptanteil des Fehlers wird durch den Gleichrichter AD736 bewirkt. Die anderen Komponenten verursachen einen deutlich kleineren Beitrag. Dies verwundert nicht weiter, da diese Komponenten für das Thermometer ausgewählt wurden, bei dem weit höhere Anforderungen an die Genauigkeit des Messgeräts gestellt werden. Selbst der durch diesen Gleichrichter hervorgerufene Fehler ist deutlich unter der angepeilten Grenze von einem Promille für das Messgerät, so dass keine weiteren Optimierungen mehr nötig sind.

11.3.3 Batterietestsystem

11.3.3.1 Stromerzeugung

Die Größe des Konstantstroms, der durch den Galvanostaten erzeugt wird, wird durch eine Steuerspannung vorgegeben, die durch den Digital/Analog-Wandler, siehe Abschnitt 5.2.2, bereitgestellt wird. Der Strom lässt sich dann aus dieser Steuerspannung nach dem Ohmschen Gesetz (23) berechnen. Nach der Größtfehlermethode gilt für den Fehler des Stroms folgender Zusammenhang:

$$\Delta I = \left|\frac{1}{R_{REF}}\right|\left|\Delta U_e\right| + \left|\frac{U_e}{R_{REF}^{\ 2}}\right|\left|\Delta R_{REF}\right| \tag{46}$$

Der Fehler des Widerstands ΔR_{REF} ist dabei unabhängig vom erzeugten Stromfluss. Für den Bereich kleiner Ströme beträgt der Fehler für diesen Widerstand 1 Ω[50] [26]. Für den Messbereich bis 400 mA werden statt diesem Widerstand 14 parallelgeschaltete 350 Ω Widerstände eingesetzt (Abschnitt 5.2.1), die einen Gesamtwiderstand von 25 Ω ergeben, dessen Fehler 2,8 mΩ [51] beträgt.
Die Steuerspannung U_e des Galvanostaten lässt sich mit folgender Formel berechnen:

$$U_E = \frac{BITWERT \cdot U_{REF}}{32768} \cdot X \tag{47}$$

Dabei entspricht BITWERT dem Datenwort, mit dem der Mikrocontroller den D/A-Wandler ansteuert, der mit einer Referenzspannung U_{REF} versorgt wird. X ist der Verstärkungsfaktor mit dem die Spannung verstärkt wird. Der Fehler der Steuerspannung beträgt damit:

$$\begin{aligned}\Delta U_E = \frac{1}{32768}\Big(&\left|U_{REF} \cdot X\right|\left|\Delta BITWERT\right| + \left|BITWERT \cdot X\right|\left|\Delta U_{REF}\right| + \left|U_{REF} \cdot BITWERT\right|\left|\Delta X\right|\Big) \\ &+\left|U_\theta \cdot X\right| + 2\left|X \cdot \Delta U_{MAX400}\right| + \left|\Delta U_{OPA547}\right|\end{aligned} \tag{48}$$

Die Offsetfehler der Operationsverstärker ΔU_{OPA547} und ΔU_{MAX400} betragen 1,1 mV für den OPA547[52] und 13 µV für die beiden MAX400[53]. Im ungünstigsten Fall beläuft sich der Fehler, der durch den Temperaturkoeffizienten des Digital/Analog-Wandlers hervorgerufen wird, auf 1,3 µV [27]

Die einzelnen Fehler des Bitwertes, ΔBITWERT sind in Tabelle 49 aufgeführt:

Nichtlinearität	max. ±2 LSB
Differentielle Nichtlinearität	max. ±1 LSB
Offset (bipolar)	max. ±1 LSB
Gain Error	max. ±5 LSB

Tabelle 49 Fehler des Digital/Analog-Wandlers MAX542 [27]

[50] Dieser Wert entspricht seiner Toleranz von 0,1 %. Die Fehler, die sich aus dem Temperaturkoeffizienten von 3ppm/°C bei einer Schwankung von 5°C der Raumtemperatur (15 mΩ) und aus der Langzeitstabilität für 10000h (35ppm) (35 mΩ) ergeben, können vernachlässigt werden.

[51] Dieser Wert setzt sich aus dem Temperaturkoeffizienten von 3ppm/°C bei einer Schwankung von 5°C der Raumtemperatur (1,9 mΩ), aus der Langzeitstabilität für 10000h (35ppm) (0,88 mΩ) und der Toleranz von 0,1 % zusammen. Ist die Toleranz der einzelnen Widerstände voneinander statistisch unabhängig, so beträgt der Fehler, der durch die Toleranz verursacht wird 6,7 mΩ.

[52] 1 mV Offset und 0,1 mV Offsetdrift mit der Temperatur

[53] 10 µV Offset und 2,8 µV Drift über 10000h

Der Fehler der Spannungsreferenz ΔU_{REF} beträgt maximal 1,3 mV. Dieser Wert setzt sich aus den in Tabelle 50 angegebenen Fehlern zusammen.

Toleranz	0,02%
Temperaturkoeffizient	0,5 ppm / °C
Spannungsausregelung	10 ppm / V
Stromausregelung	1 ppm / mA
Langzeitdrift	30 ppm / 1000h

Tabelle 50 Fehler der Spannungsreferenz MAX6325 [28]

Der Fehler des Verstärkungsfaktors ΔX wird durch die Toleranzen der beiden Rückkopplungswiderstände R_{33} und R_{34} bestimmt. Nach der Größtfehlerabschätzung gilt für ΔX folgender Zusammenhang:

$$\Delta X = \left|\frac{R_{34}}{R_{33}^2}\right|\left|\Delta R_{33}\right| + \left|\frac{1}{R_{34}}\right|\left|\Delta R_{34}\right| \tag{49}$$

Der R_{34} hat einen Wert von 10 kΩ mit einem Fehler von 10 Ω [54], R_{33} hingegen einen Wert von 100 kΩ mit einem Fehler von 100 Ω [55]. Nach der Größtfehlerabschätzung (49) beträgt somit der Fehler des Verstärkungsfaktor $1{,}1\times10^{-3}$ Ω.

Die Abhängigkeit des Fehlers des Galvanostaten ΔI vom konstantgehaltenen Strom lässt sich mit Gleichung (46) berechen, wenn Gleichung (48) und die Abhängigkeit des Bitwerts vom Strom, Gleichung (50), eingesetzt werden[56],

$$BITWERT = \frac{I \cdot R}{X} \cdot \frac{32768}{U_{REF}} \tag{50}$$

worauf sich Gleichung (51) ergibt:

[54] Dieser Wert entspricht seiner Toleranz von 0,1 %. Die Fehler, die sich aus dem Temperaturkoeffizienten von 3ppm/°C bei einer Schwankung von 5°C der Raumtemperatur (150 mΩ), und aus der Langzeitstabilität für 10000h (35ppm) (350 mΩ) ergeben, können vernachlässigt werden.

[55] Dieser Wert entspricht seiner Toleranz von 0,1 %. Die Fehler, die sich aus der Temperaturkoeffizienten von 3ppm/°C bei einer Schwankung von 5°C der Raumtemperatur (1,5 Ω), und aus der Langzeitstabilität für 10000h (35ppm) (3,5 Ω) ergeben, können vernachlässigt werden.

[56] Der Faktor 32768 entspricht der Auflösung eines bipolaren 16-Bit Wandlers in Richtung positiver oder negativer Spannung.

$$\Delta I = \left|\frac{1}{R_{REF}}\right| \left| \frac{1}{32768} \left(\begin{array}{l} \left|U_{REF} \cdot X\right| \left|\Delta BITWERT\right| + \left|I \cdot R \cdot \frac{32768}{U_{REF}}\right| \left|\Delta U_{REF}\right| + \\ \left|\frac{I \cdot R}{X} \cdot 32768\right| \left|\Delta X\right| \end{array} \right) + \left|U_{\theta} \cdot X\right| + 2\left|X \cdot \Delta U_{MAX\,400}\right| + \left|\Delta U_{OPA547}\right| \right| + \left|\frac{U_e}{R_{REF}^{\,2}}\right| \left|\Delta R_{REF}\right| \quad (51)$$

In den Abbildungen sind die berechneten Fehler des Galvanostaten in Abhängigkeit des Konstantstroms dargestellt.

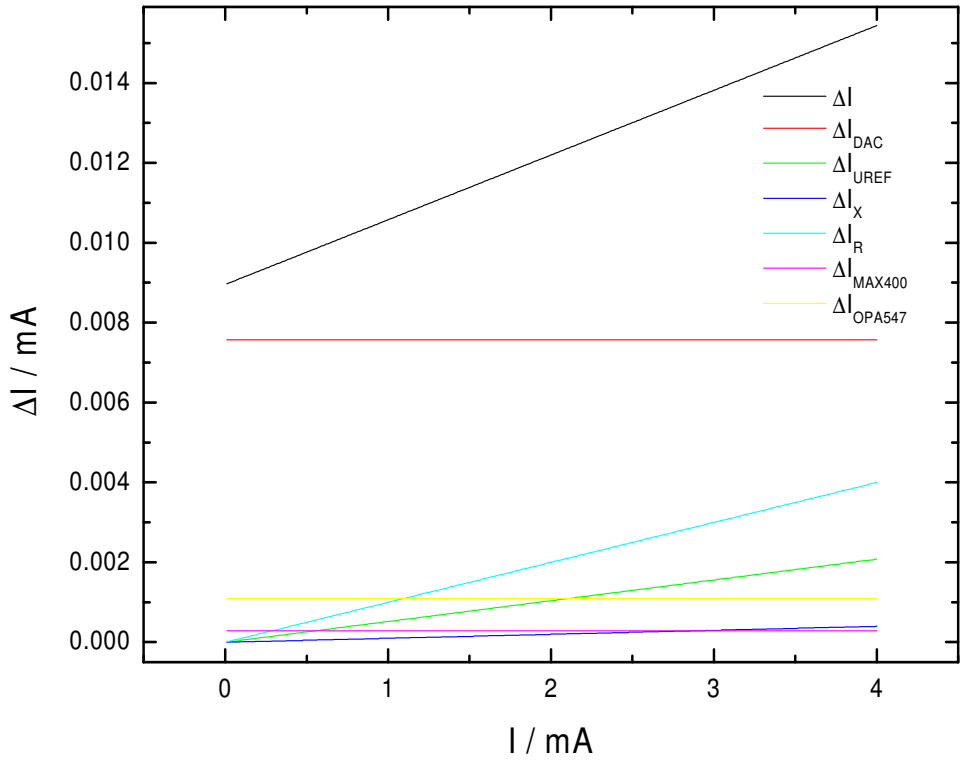


Abbildung 80 Fehler des Konstantstroms für kleine Werte

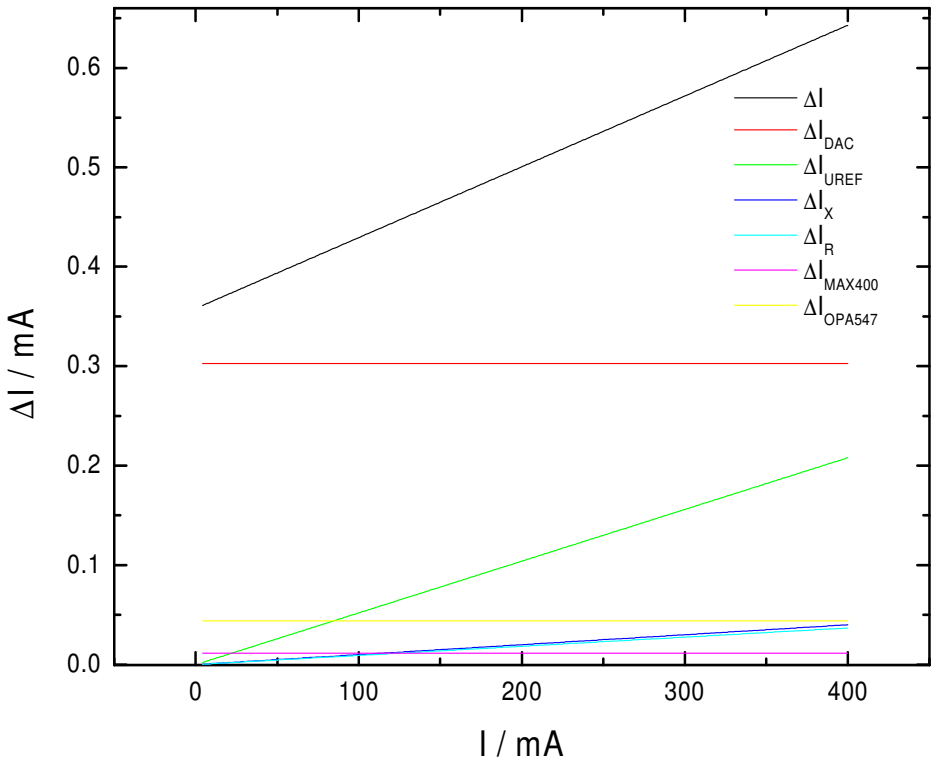


Abbildung 81 Fehler des Konstantstroms im Bereich bis 400 mA

Wie diesen Abbildungen entnommen werden kann, wird der Hauptanteil des Fehlers durch den Digital/Analogwandler verursacht. Die zweitgrößte Fehlerquelle ist die Spannungsreferenz, die diesen Wandler versorgt. Eine Optimierung dieses Messgeräts müsste daher an diesen beiden Bauteilen durchgeführt werden.

11.3.3.2 Strommessung

Für die Strommessung wird die Spannung U_A gemessen, die am Referenzwiderstand R_{REF} abfällt. Aus dieser Spannung kann mit dem Ohmschen Gesetz (23) der Strom berechnet werden. Diese Spannung wird, nachdem sie über zwei Verstärker verstärkt wurde, mit dem Analog/Digitalwandler gemessen:

$$U_A = \frac{BITWERT \cdot U_{REF}}{2048 \cdot X \cdot Y} \tag{52}$$

und somit erhält man für den Strom:

$$I = \frac{BITWERT \cdot U_{REF}}{2048 \cdot X \cdot Y \cdot R_{REF}} \tag{53}$$

Nach der Größtfehlermethode ergibt sich dann für den Fehler der Strommessung, wenn noch der Offsetfehler des Operationsverstärkers, ΔU_{MAX400} berücksichtigt wird [57]:

$$\Delta I = \frac{1}{2048}\left(\begin{array}{l}\left|\frac{U_{REF}}{X \cdot Y \cdot R_{REF}}\right|\left|\Delta BITWERT\right| + \left|\frac{BITWERT}{X \cdot Y \cdot R_{REF}}\right|\left|\Delta U_{REF}\right| + \\ \left|\frac{BITWERT \cdot U_{REF}}{X^2 \cdot Y \cdot R_{REF}}\right|\left|\Delta X\right| + \left|\frac{BITWERT \cdot U_{REF}}{X \cdot Y^2 \cdot R_{REF}}\right|\left|\Delta Y\right| + \left|\frac{BITWERT \cdot U_{REF}}{X \cdot Y \cdot R^2_{REF}}\right|\left|\Delta R_{REF}\right|\end{array}\right) + \left|\frac{X \cdot Y \cdot \Delta U_{MAX\,400}}{R_{REF}}\right| \tag{54}$$

Die Fehler, aus denen sich der Fehler des A/D-Wandlers, ΔBITWERT, zusammensetzt, sind in Tabelle 51 aufgeführt:

Nichtlinearität	max. ±0,5 LSB
Differentielle Nichtlinearität	max. ±1 LSB
Offset (bipolar)	max. ±0,1 LSB[58]
Gain Error	max. ±7 LSB

Tabelle 51 Fehler des A/D Wandlers MAX128 [29]

Der Fehler der Spannungsreferenz ΔU_{REF} beträgt 0,02 V [29], der des PGA-Verstärkers hingegen $\Delta X = 6{,}65 \times 10^{-5}$.

Der Fehler des Vorverstärkers ΔY ergibt sich, wenn die Größtfehlermethode auf Gleichung (25) angewendet wird:

$$\Delta Y = \left|\frac{R_{32}}{\left(R_{32} + R_{31}\right)^2}\right|\left|\Delta R_{31}\right| + \left|\frac{R_{31}}{\left(R_{32} + R_{31}\right)^2}\right|\left|\Delta R_{32}\right| \tag{55}$$

Wird für R_{31} ein Widerstand von (100,0 ± 0,1) kΩ [59] und für R_{32} ein Wert von (50,00 ± 0,05) kΩ [59] eingesetzt, entspricht der Fehler der Abweichung des Vorverstärkers $4{,}4 \times 10^{-4}$.

Wie in Abschnitt 11.3.3.1 angegeben, beträgt der Offsetfehler ΔU_{MAX400} eines MAX400 Operationsverstärkers 13 µV.

Der Fehler des Referenzwiderstandes ΔR_{REF} wurde in Kapitel 11.3.3.1 zu 1 Ω [26]für den Messbereich bis 4 mA bestimmt. Für den Messbereich bis 400 mA beträgt dieser Fehler 2,8 mΩ.

In Analogie zum Galvanostaten, siehe Abschnitt 11.3.3.1 wird auch bei der Strommessung die Abhängigkeit des Fehlers des gemessenen Stroms ΔI vom Strom mit Gleichung (54) berechnet, wenn für die Abhängigkeit des Bitwerts vom Strom Gleichung (56) eingesetzt wird[60].

[57] Der Offsetfehler des PGA-Verstärkers kann über den Trimmer vollständig kompensiert werden.

[58] Bei der Offsetkorrektur durch einen auf Massepotenzial gelegten Kanal muss nur der wesentlich kleinere Offset zwischen den einzelnen Kanälen berücksichtigt werden.

[59] Dieser Wert entspricht seiner Toleranz von 0,1 %. Die Fehler, die sich aus den Temperaturkoeffizienten und aus der Langzeitstabilität ergeben, können vernachlässigt werden.

$$BITWERT = \frac{2048}{U_{REF}} I \cdot R_{REF} \cdot X \cdot Y \tag{56}$$

Somit wird folgender Zusammenhang erhalten:

$$\Delta I = \frac{1}{2048}\left(\begin{array}{l}\left|\frac{U_{REF}}{X \cdot Y \cdot R_{REF}}\right|\left|\Delta BITWERT\right| + \left|\frac{2048}{U_{REF}} I\right|\left|\Delta U_{REF}\right| + \\ \left|\frac{2048 \cdot I}{X}\right|\left|\Delta X\right| + \left|\frac{2048 \cdot I}{Y}\right|\left|\Delta Y\right| + \left|\frac{2048 \cdot I}{R_{REF}}\right|\left|\Delta R_{REF}\right|\end{array}\right) + \left|\frac{X \cdot Y \cdot \Delta U_{OFFSET,MAX400}}{R_{REF}}\right| \tag{57}$$

In Abbildung 82 und Abbildung 83 ist die Abhängigkeit der Fehlers der Strommessung vom gemessenen Strom dargestellt. Zusätzlich sind in diesen Abbildungen noch die Beiträge der einzelnen Komponenten zum Gesamtfehler aufgetragen.

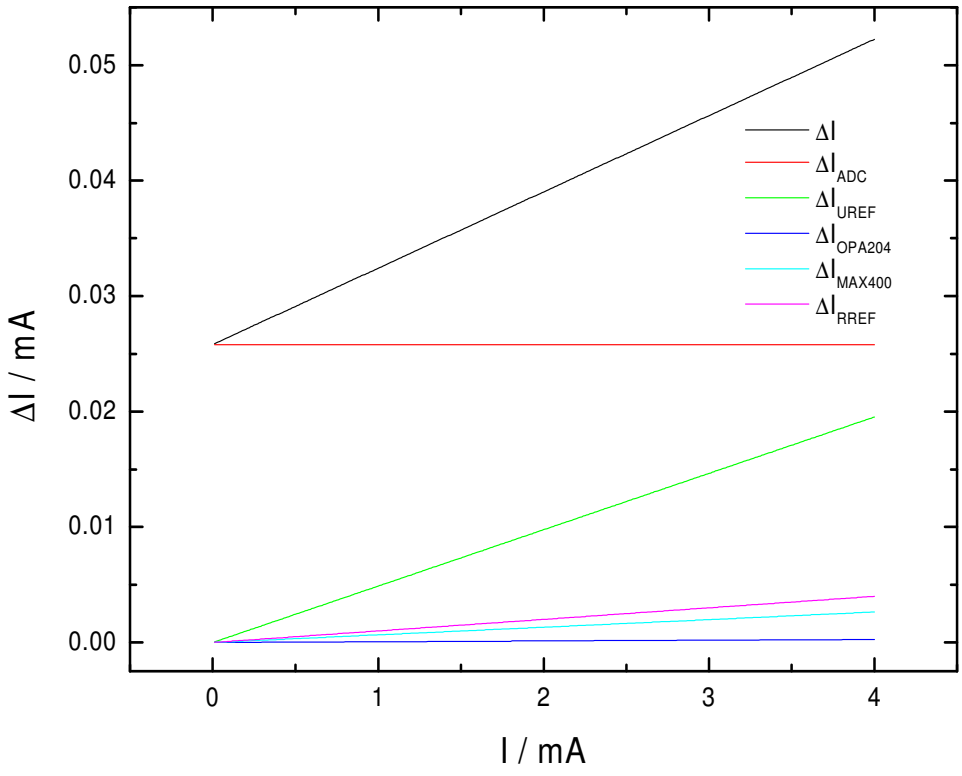


Abbildung 82 Fehler der Strommessung im kleinen Bereich

[60] 2048 entspricht der Auflösung eines bipolaren 12-Bit Wandlers in Richtung positiver oder negativer Spannung.

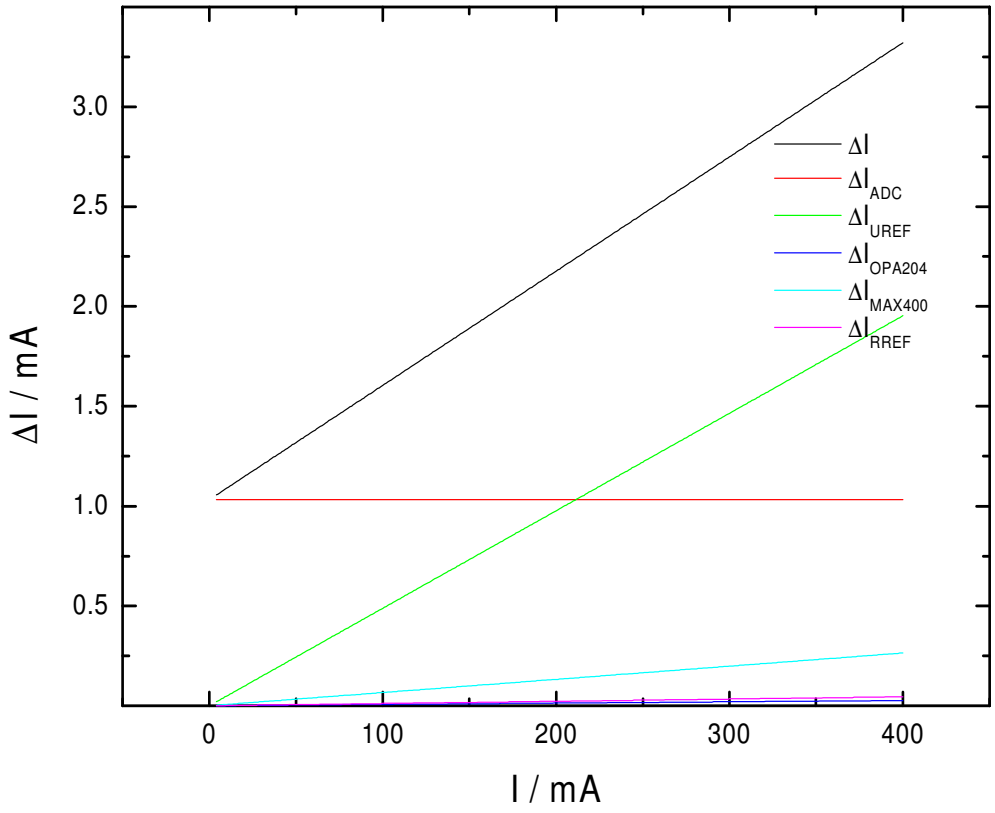


Abbildung 83 Fehler der Strommessung im großen Bereich

Wie auch beim Galvanostaten wird der Hauptanteil des Fehlers nicht durch die Widerstände oder durch die Verstärkerstufen verursacht. Vielmehr liefert der Analog/Digital-Wandler mit seiner integrierten Spannungsreferenz den Hauptbeitrag zum Fehler der Schaltung. Ein erster Schritt zur Erhöhung der Genauigkeit wäre der Einsatz einer externen Spannungsreferenz, wie beispielsweise einer MAX6341, welche die relativ ungenaue interne Spannungsreferenz des MAX128 ersetzt. Eine nochmalige Steigerung der Genauigkeit ließe sich durch einen genaueren Analog/Digital-Wandler erreichen.
Da durch den sehr präzisen Galvanostaten der Strom mit einer Genauigkeit erzeugt wird, die mindestens um den Faktor 5 besser ist als der gemessene Strom, wird für die Berechung der umgesetzten Ladung nicht der gemessene Strom, sondern der Sollstrom verwendet. Die Strommessung dient bei diesem Messgerät daher nur noch als Kontrolle.

11.3.3.3 Spannungsmessung

Die Spannungsmessung der Batteriespannung erfolgt durch Differenzbildung der Spannungen U_A und U_B. Diese Spannungen werden jeweils mit einem Analog/Digitalwandler gemessen, dem eine Vorverstärkerstufe aus zwei Verstärkern vorgeschaltet ist (siehe Abschnitt 5.2.3).

Die Batteriespannung U lässt sich in Abhängigkeit der mit dem Analog/Digital-Wandler eingelesenen Bitwerte durch folgenden Zusammenhang beschreiben:

$$U = \frac{BITWERT_B \cdot U_{REF}}{2048 \cdot X_B \cdot Y} - \frac{BITWERT_A \cdot U_{REF}}{2048 \cdot X_A \cdot Y} \tag{58}$$

Der Fehler der Spannungsmessung ΔU kann dann aus Gleichung (58) durch die Größtfehlermethode bestimmt werden.

$$\Delta U = \frac{1}{2048}\left(\begin{array}{l} \left|\frac{U_{REF}}{X_B \cdot Y}\right|\left|\Delta BITWERT_B\right| + \left|\frac{U_{REF}}{X_A \cdot Y}\right|\left|\Delta BITWERT_A\right| + \\ \left|\frac{BITWERT_B X_A - BITWERT_A X_B}{X_B \cdot Y \cdot X_A}\right|\left|\Delta U_{REF}\right| + \\ \left|\frac{BITWERT_B \cdot U_{REF}}{X_B^2 \cdot Y}\right|\left|\Delta X_B\right| + \left|\frac{BITWERT_A \cdot U_{REF}}{X_A^2 \cdot Y}\right|\left|\Delta X_A\right| + \\ \left|\frac{\left(BITWERT_B X_A - BITWERT_A X_B\right) \cdot U_{REF}}{X_B \cdot Y^2 \cdot X_A}\right|\left|\Delta Y\right| \end{array}\right) \tag{59}$$

Die Fehler des A/D-Wandlers, der Spannungsreferenz und der Vorverstärker sind in Abschnitt 11.3.3.2 beschrieben.
In Abbildung 84 ist der Fehler der Spannungsmessung in Abhängigkeit beider über Analog/Digital-Wandler gemessenen Spannungen dargestellt. Zusätzlich sind noch die einzelnen Beiträge der Komponenten der Schaltung zum Gesamtfehler eingezeichnet.

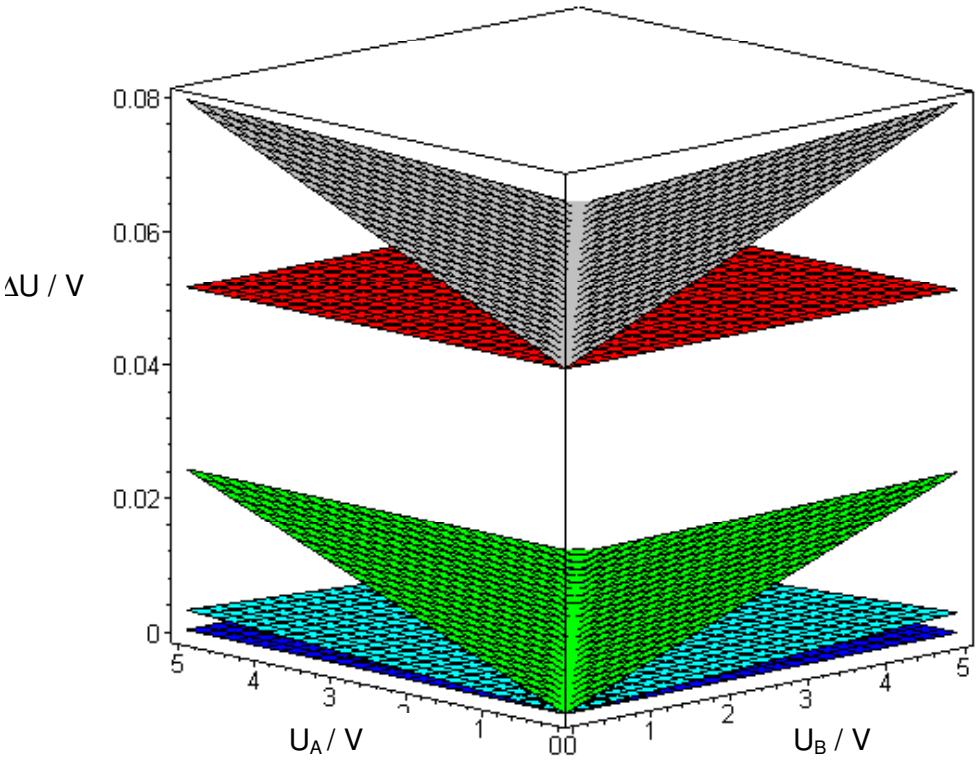


Abbildung 84 Fehler der Spannungsmessung ΔU in Abhängigkeit der mit dem A/D-Wandler gemessenen Spannungen U_A, und U_B. Gesamtfehler der Spannungsmessung (grau), Anteil des A/D-Wandlers (rot), Spannungsreferenz (grün), PGA-Verstärker (blau) und Abschwächer (zyan)

Wie aus dieser Abbildung ersichtlich, ist der Hauptteil der Abweichungen auf den Fehler des Analog/Digital-Wandlers (rot) zurückzuführen. Die Fehler, die durch die beiden Verstärker verursacht werden, sind im Vergleich zum Fehler, den die Spannungsreferenz verursacht, vernachlässigbar.
In Abbildung 85 ist ein Schnitt aus Abbildung 84 dargestellt, der erhalten wird, wenn U_A gleich U_B gesetzt wird.

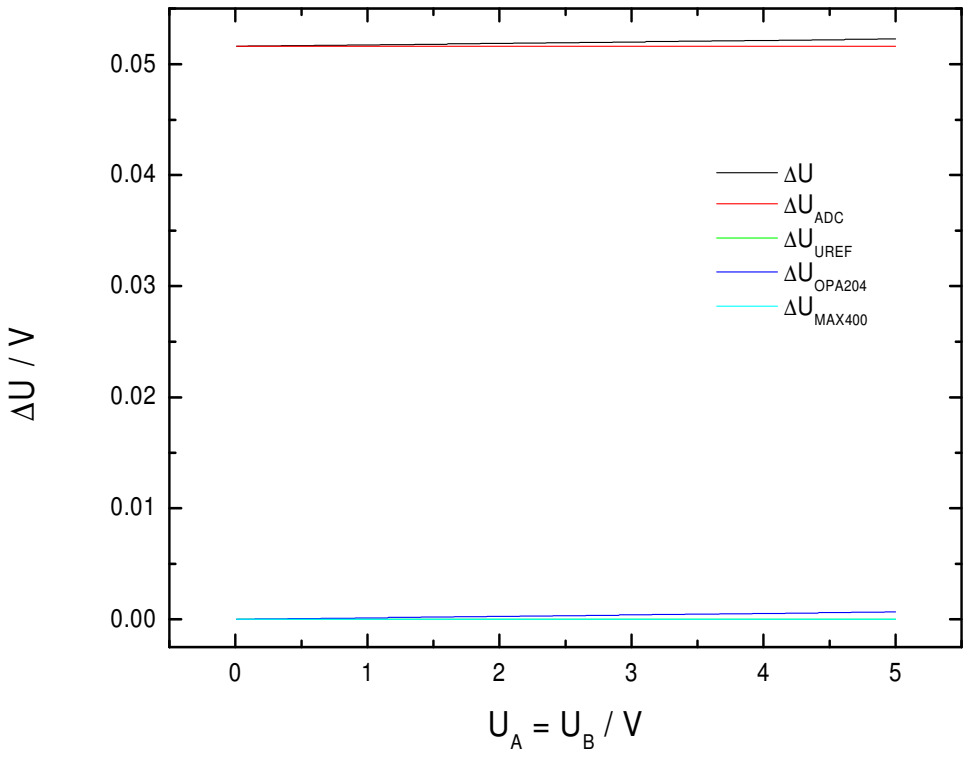


Abbildung 85 Fehler der Spannungsmessung ΔU in Abhängigkeit der mit dem A/D-Wandler gemessenen Spannungen U_A, und U_B

Auch hier wird der gleiche Einfluss der Komponenten deutlich, wobei sich die Fehler der Spannungsreferenz und der Vorverstärker gegenseitig aufheben.
Mit einem maximalen Fehler von 80 mV entspricht die Spannungsmessung den an sie gestellten Anforderungen. Wie durch die Fehlerrechnung gezeigt werden konnte, wäre es für eine Steigerung der Messgenauigkeit sinnvoll, den Analog/Digital-Wandler durch einen genaueren Typen zu ersetzen. Dadurch würde sich auch gleichzeitig die Genauigkeit der Strommessung erhöhen.

11.3.4 Wasserbestimmung mit ^{1}H-NMR

11.3.4.1 Fehlerabschätzung mittels der Größtfehlermethode

Der Fehler der Molalität, Δb_{EC1}, an EC der Messlösung Δb_{EC1} der ersten Stammlösung lässt sich durch Größtfehlerabschätzung mit folgender Formel bestimmen:

$$\Delta b_{EC1} = \left| \frac{1}{M_{EC} \cdot m_{AN}} \right| \left| \Delta m_{EC} \right| + \left| \frac{m_{EC}}{M_{EC} \cdot m_{AN}^{\,2}} \right| \left| \Delta m_{AN} \right| \qquad (60)$$

Mit der Größtfehlerabschätzung wird für den Fehler der Molalität, Δb_{EC2}, an EC der Messlösung Folgendes erhalten:

$$\Delta b_{EC2} = \left|\frac{m_1}{X} + \frac{b_1 \cdot m_1^{\,2}}{X^2}\right| \cdot \left|\Delta b_1\right| + \left|\frac{b_1}{X} + \frac{b_1 \cdot m_1 \cdot \left(1 - \frac{b_1}{M_{EC}}\right)}{X^2}\right| \cdot \left|\Delta m_1\right| + \left|-\frac{b_1 \cdot m_1}{X^2}\right| \cdot \left|\Delta m_{AN}\right| \quad (61)$$

$$\text{mit X} = m_{AN} + m_1 - \frac{b_1 \cdot m_1}{M_{EC}}$$

Der Fehler der Wasserbestimmung, Δw_{Salz}, kann aus Gleichung (32) mit der Größtfehlermethode bestimmt werden, wobei folgende Gleichung erhalten wird:

$$\begin{aligned}\Delta w_{Salz} = &\left|\frac{2 \cdot INT \cdot b_{EC2} \cdot M_{H_2O}}{m_{Salz}}\right| \left|\Delta m_{AN}\right| + \left|\frac{2 \cdot INT \cdot m_{AN} \cdot M_{H_2O}}{m_{Salz}}\right| \left|\Delta b_{EC2}\right| + \\ &+ \left|-\frac{2 \cdot INT \cdot b_{EC2} \cdot m_{AN} \cdot M_{H_2O}}{m_{Salz}^{\,2}}\right| \left|\Delta m_{Salz}\right| + \left|\frac{2 \cdot b_{EC2} \cdot m_{AN} \cdot M_{H_2O}}{m_{Salz}}\right| \left|\Delta INT\right|\end{aligned} \quad (62)$$

Für eine typische Messung wird 1 g der Standardlösung und 100 mg Salz verwendet. Wenn die Lösung mit einer molalen Konzentration von 1,69 mmol/kg eingesetzt wird, so beträgt deren Fehler 0,05 mmol/kg. Die Fehler der Massenbestimmungen betragen 0,001 g. Für den Fehler des Integrals wird, wie in Abschnitt 11.3.4.3 angegeben, ein Wert von 5% vom Messwert angenommen. Bei einem Integral von 0,5, was einem Wassergehalt von 304ppm entspricht, beläuft sich der Fehler einer typischen Messung nach Gleichung (62) 40×10^{-6}. Der Wassergehalt beträgt also (304 ± 40) ppm.

Wird für eine Messung die Standardlösung mit (13,6 ± 0,2) mmol/kg EC eingesetzt, so werden natürlich kleinere Integrale erhalten. Geht man von einem Wassergehalt von 300 ppm aus, so beträgt das Integral 0,061. Nach Gleichung (62) wird hier für den Wassergehalt ein Fehler von 23×10^{-6} erhalten. Der Wassergehalt beträgt somit (300 ± 23) ppm. Die Steigerung der Genauigkeit ist auf die höhere Genauigkeit des EC Standards zurückzuführen, da hier bei der Herstellung der Standardlösung eine größere Menge an EC eingewogen wurde und daher der Fehler in der Massenbestimmung nicht so sehr ins Gewicht fällt.

Für den Fehler der molalen Wasserkonzentration bei den Standardadditionen, $\Delta b_{H2O,rec}$, gilt folgender Zusammenhang:

$$\Delta b_{H_2O,\mathrm{rec}} = \left|\frac{m_{H_2O,s}}{X}\right| \cdot \left|\Delta b_{H_2O,S}\right| + \left|\frac{w_{Salz}}{M_{H_2O}(X)}\right| \cdot \left|\Delta m_{Salz}\right| + \left|\frac{m_{Salz}}{M_{H_2O}(X)}\right| \left|\Delta w_{Salz}\right|$$
$$+ \left|\frac{b_{H_2O,S} m_{H_2O,S} + \frac{m_{Salz} w_{Salz}}{M_{H_2O}}}{X^2}\right| \left|\Delta m_{AN}\right| + \left|\frac{b_{H_2O,S}}{m_{AN} + m_{H_2O,S}} - \frac{b_{H_2O,S} m_{H_2O,S} + \frac{m_{Salz} w_{Salz}}{M_{H_2O}}}{X^2}\right| \left|\Delta m_{H_2O}\right| \quad (63)$$

$$\text{mit } X = m_{AN} + m_{H_2O}$$

Für die Massefehler $\Delta m_{AN} + \Delta m_{Salz} + \Delta m_{H2O,s}$ wird, wie in Abschnitt 7.3.3.1 aufgeführt, ein Fehler von 1 mg angenommen. Der Fehler des Wassergehalts des Salzes ist bei den jeweiligen Standardadditionen dargestellt.

Der Fehler des Sollgehalts an Wasser im Salz bei der Standardaddition ergibt sich mit der Größtfehlerabschätzung gemäß folgender Gleichung:

$$\Delta w_{H_2O,\mathrm{rec}} = \left|\frac{m_{H_2O,S} M_{H_2O}}{m_{Salz}}\right| \left|\Delta b_{H_2O,S}\right| + \left|\frac{b_{H_2O,S} M_{H_2O}}{m_{Salz}}\right| \left|\Delta m_{H_2O,S}\right|$$
$$+ \left|\frac{w_{Salz}}{m_{Salz}} - \frac{\left(b_{H_2O,S} m_{H_2O,S} + \frac{m_{Salz} w_{Salz}}{M_{H_2O}}\right) M_{H_2O}}{m_{Salz}^2}\right| \left|\Delta m_{Salz}\right| + \left|\Delta w_{Salz}\right| \quad (64)$$

11.3.4.2 Erfassungsgrenze

Die Erfassungsgrenze der Methode wird zum einen durch die Erfassungsgrenze der NMR Messung bestimmt. Durch eine möglichst große Anzahl von Messpulsen bei der Erfassung des NMR Spektrums wird ein bestmögliches Signal-Rausch-Verhältnis erreicht, so dass auch sehr schwache Peaks, die von Substanzen in hoher Verdünnung hervorgerufen werden, sicher detektiert werden können.

In Abbildung 86 ist der Wasserpeak des NMR Spektrums von $LiClO_4$ in getrocknetem D-AN mit EC Standard mit einer molalen Konzentration von (13,6 ± 0,2) mmol/kg dargestellt. Die Bedingungen für diese Messung sind in der ersten Zeile der Tabelle 47 angeben.

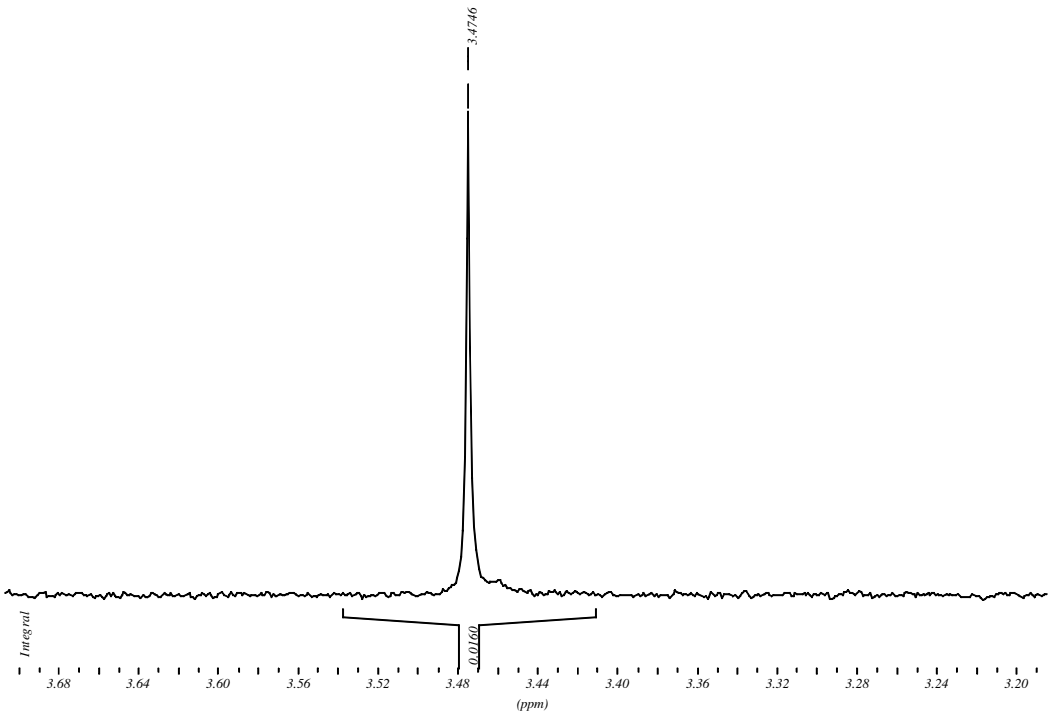


Abbildung 86 Bestimmung der Erfassungsgrenze

In Abschnitt 11.2.5.9 wird für diesen Peak eine molale Wasserkonzentration von 0,44 µmol/g ermittelt. Vergleicht man nun den Peak mit dem Rauschen der Basislinie, so kann man auch noch Peaks, die nur noch über ein Zehntel der Integralfläche verfügen, sicher integrieren, bevor der Fehler durch das Rauschen überhand nimmt. Daher dürften sich Wasserkonzentrationen im Bereich von 0,04 µmol/g noch quantitativ erfassen lassen.

Eine Aussage für die Erfassungsgrenze des Wassergehalts im Salz kann nicht so leicht getroffen werden, da die Wasserkonzentration in der Lösung für die NMR Messung natürlich von der Menge der gelösten Salze abhängt. Für eine sichere Bestimmung einer sehr kleinen Menge Wasser sollte daher eine möglichst große Menge Salz im Lösungsmittel gelöst werden. Der bestimmbare Grenzgehalt an Wasser hängt daher direkt von der Löslichkeit des Salzes im NMR Lösungsmittel ab. Bei der in Abbildung 86 dargestellten Messung betrug die eingesetzte Salzmenge 135 mg, mit der ein Wassergehalt von 55 ppm bestimmt wurde. Wird auch hier ein Zehntel dieses Werts als sicher bestimmbar angenommen, liegt die quantitative Erfassungsgrenze bei etwa 6 ppm.

In der Tabelle 44 ist ein Experiment aufgeführt bei dem 0,012g LiBOB eingewogen wurden sind. Im ^{1}H-NMR Spektrum wurde aber kein Wasserpeak mehr gefunden. Durch diese im Vergleich zur üblichen Einwaage von 100mg sehr kleinen Einwaage wird daher nur eine sehr geringe Menge Wasser in das NMR-Röhrchen eingebracht. Der Wassergehalt des LiBOBs beträgt nach Abschnitt 11.2.5.2 120 ppm. Eine Einwaage von einem Zehntel der Wassermenge entspricht nur einem Zehntel der Wassermenge im Salz. Daher dürfte nach dieser Abschätzung die Nachweisgrenze bei einer Einwaage von 100 mg, bei 10 ppm liegen, was in etwa der oben gemachten Aussage entspricht.

Bei den Angaben der Erfassungsgrenze dürfen aber Kontaminationen der Lösung durch Wasser aus der Umgebung nicht vernachlässigt werden. Die Hauptquelle hierfür dürfte vor allem in der Undichtigkeit der Verschlusskappen der NMR Rohre zu sehen sein. Leider ist es aufgrund der langen Messzeit von circa 30 Minuten nicht möglich, eine Aussage über die Wasseraufnahme in kurzen Zeiträumen zu treffen. Eine weitere nicht zu vernachlässigende Wasserquelle stellt das NMR-Lösungsmittel dar, das zwar unter die Erfassungsgrenze der Karl-Fischer Titration getrocknet wur-

de aber immer noch einen Teil an Wasser beisteuert. Dank der Durchführung der Versuche im Handschuhkasten unter einer Argonatmosphäre mit einem Wassergehalt von 0,2 ppm und durch das Ausheizen der Glasgeräte dürfte die Kontamination hierdurch aber nur sehr gering sein.
Bei der Messung des $LiPF_6$ in der getrockneten Standardlösung, siehe Abschnitt 11.2.5.8, wurde weder ein Wasser-Peak, noch ein HF-Peak gefunden. Aufgrund dieser Beobachtung lässt sich schließen, dass eine Kontamination durch die oben aufgeführten Ursachen so gering ist, dass sie nicht nachweisbar ist.

11.3.4.3 Bestimmung der Wiederholgenauigkeit der NMR-Messungen

Um die Wiederholgenauigkeit der NMR Messungen zu bestimmen, und damit eine Abschätzung für den Fehler der Integration zu bekommen, wurden (0,074 ± 0,001) g LiBOB der Charge SCHK01/29 in (0,793 ± 0,001) g D-AN Standardlösung mit einer molalen Konzentration an EC von (13,6 ± 0,2) mmol/kg gelöst. Diese Lösung wurde dann je zehnmal vermessen. In Tabelle 52 sind die erhaltenen Werte aufgeführt.

INT	b_{H2O} / µmol/g	m_{H2O} /µg	Wasser / ppm
0,0239	0,65	9,06	117
0,0265	0,72	10,1	130
0,0233	0,63	8,82	114
0,0245	0,67	9,29	120
0,0248	0,67	9,44	122
0,0234	0,64	8,9	115
0,0236	0,64	8,98	116
0,0265	0,72	10,1	130
0,0243	0,66	9,21	119
0,0234	0,64	8,9	115

Tabelle 52 Wiederholgenauigkeit der NMR Messung

Der Mittelwert der Integrale des Wasserpeaks INT beträgt 0,0245. Die Messwerte streuen mit einer Standardabweichung von 0,0012 um diesen Mittelwert, also 4,8% vom Messwert. Es liegt somit nahe, für den prozentualen Fehler des Integrals ΔINT einen Wert von 5% anzunehmen.

11.3.4.4 Dichtigkeit der NMR-Kappen

Zum Verschließen der NMR-Röhrchen standen zwei verschiedene Deckeltypen zur Verfügung, Standardkappen aus Polyethylen von RototecSpintec (Biebesheim) und Naturgummi 5mm „Serum Caps“ des gleichen Herstellers, die laut diesem eine deutlich höhere Dichtigkeit aufweisen sollen. Um dies zu überprüfen, wurden für jeden Kappentyp zwei Proben der Charge SCHK01/29 abgefüllt und fünfmal über sieben Tage verteilt vermessen. Als Zeiten wurde die Zeit des Messbeginns nach der Entnahme aus dem Handschuhkasten in Tabelle 53 angegeben.

Naturgummi 1		Naturgummi 2		Polyethylen 1		Polyethylen 2	
t / min	n_{H2O} / µmol	T / min	n_{H2O} / µmol	t / min	n_{H2O} µmol	t / min	n_{H2O} / µmol
75	0,68	104	0,62	132	14,3	160	0,65
1355	0,85	1372	0,80	1412	15,6	1441	2,06
5815	3,64	5843	3,48	5871	196	5899	9,44
8561	5,72	8589	5,76	8618	312	8646	15,0
10053	6,97	10081	6,93	10110	373	10198	17,2

Tabelle 53 Dichtigkeit der NMR-Kappen

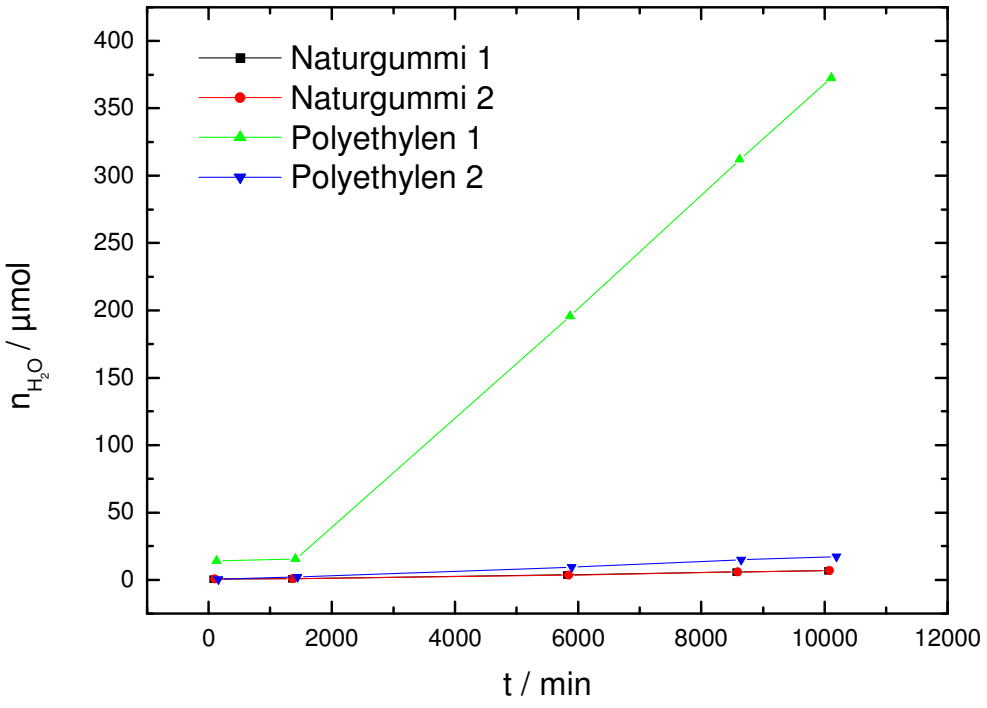


Abbildung 87 Dichtigkeit der NMR-Kappen

Da die Leckrate der Polyethylenkappe 2 (in Abbildung 87 grün dargestellt) so groß ist, dass die anderen Graphen sehr stark zusammengedrückt werden, werden in Abbildung 88 die anderen drei Graphen noch einmal vergrößert dargestellt.

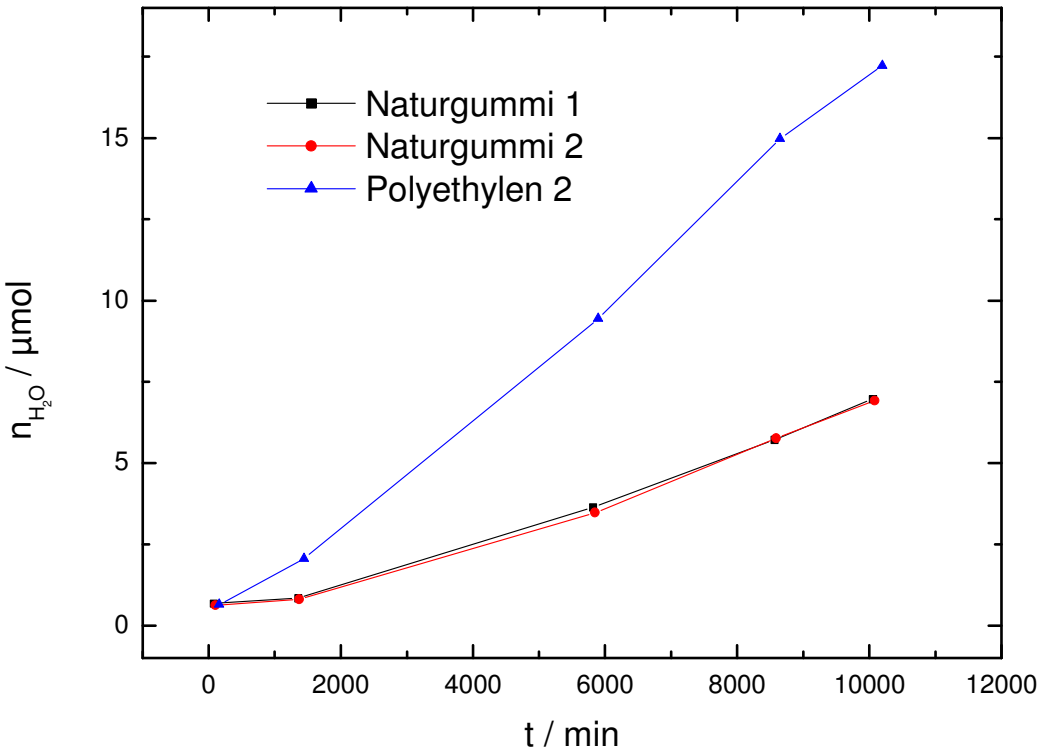


Abbildung 88 Dichtigkeit der NMR-Kappen

Wie den Abbildungen entnommen werden kann, besitzen die Naturgummikappen eine deutlich höhere Dichtigkeit als die Polyethylenkappen. Sie bieten aber auch nur innerhalb einiger Stunden Schutz gegenüber Wasserkontamination. Eine längere Aufbewahrung von wasserempfindlichen Substanzen an Luft in NMR-Rohren, die mit diesen Kappen verschlossen werden, ist aber nicht möglich. Um eine Kontamination der in den Abschnitten 11.2.5.1 bis 11.3.4.3 beschriebenen Proben zu vermeiden, wurden diese sofort nach der Entnahme aus dem Handschuhkasten vermessen.

11.4 Bedienungsanleitungen der neuen Geräte

11.4.1 Phasendiagrammmessgerät

An dieser Stelle sei noch mal auf die von Wudy geschriebe Anleitung zu dieser Anlage verwiesen [30].

11.4.2 Batterietestsystem

11.4.2.1 Aufbau und Installation der Software

Das Messgerät wird über ein Kaltgeräte-Kabel mit Strom versorgt. Die Absicherung erfolgt dabei über zwei 700 mA Glasrohrsicherungen, die im Schaltermodul untergebracht sind. Der Steuerrechner wird über seine serielle Schnittstelle mit einem 9-poligen 1:1 SUB-D Verbindungskabel angeschlossen. Sollen mehr als zwei Geräte damit verbunden werden, so muss die Anzahl der seriellen

Schnittstellen des PC mit kommerziell erhältlichen Erweiterungskarten entsprechend erhöht werden.
Um die Steuersoftware ausführen zu können, ist keine Installation erforderlich. Es genügt, das Programm in einen Unterordner zu kopieren und eine Verknüpfung im Startmenü zu erstellen.

11.4.2.2 Durchführung einer Messung

Zur Durchführung der Messung wird die Steuersoftware gestartet. Dadurch wird das in Abbildung 89 dargestellte Hauptfenster des Zyklisiergeräts sichtbar.

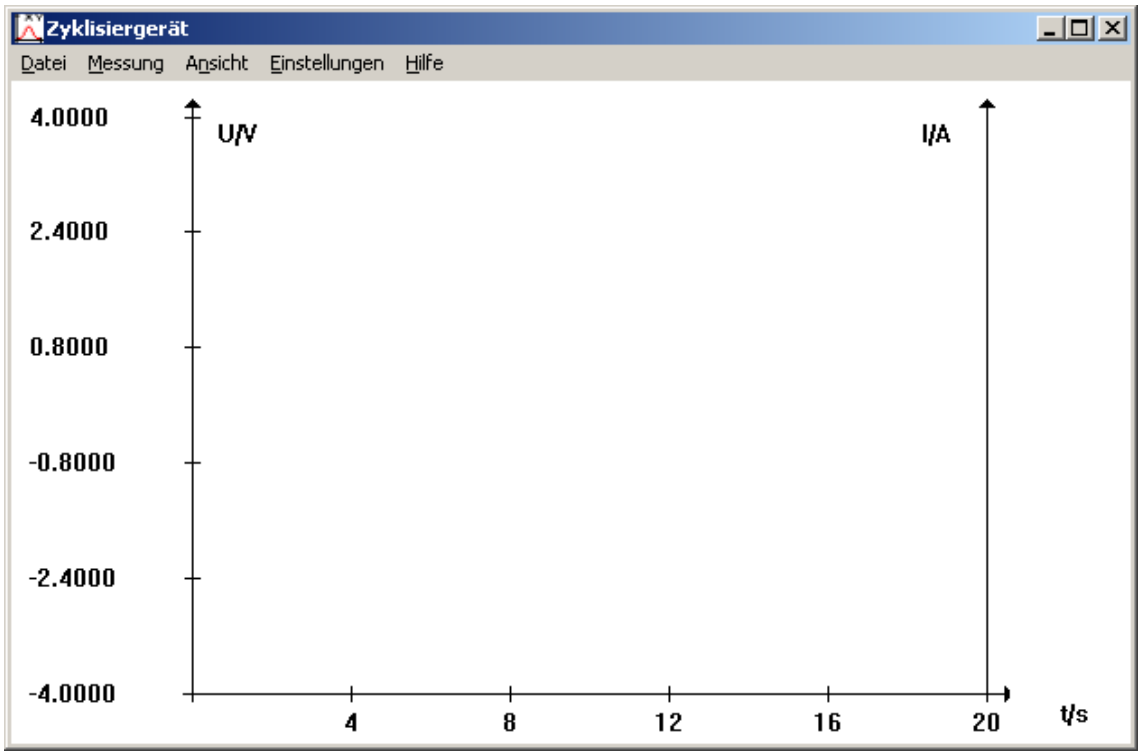


Abbildung 89 Hauptfenster des Programms zum Batterietestsystem

Über den Menüpunkt „Messung, Messung starten“ wird eine Messung gestartet. Die Parameter für diese Messung werden in den Startdialog, siehe Abbildung 90, eingegeben.

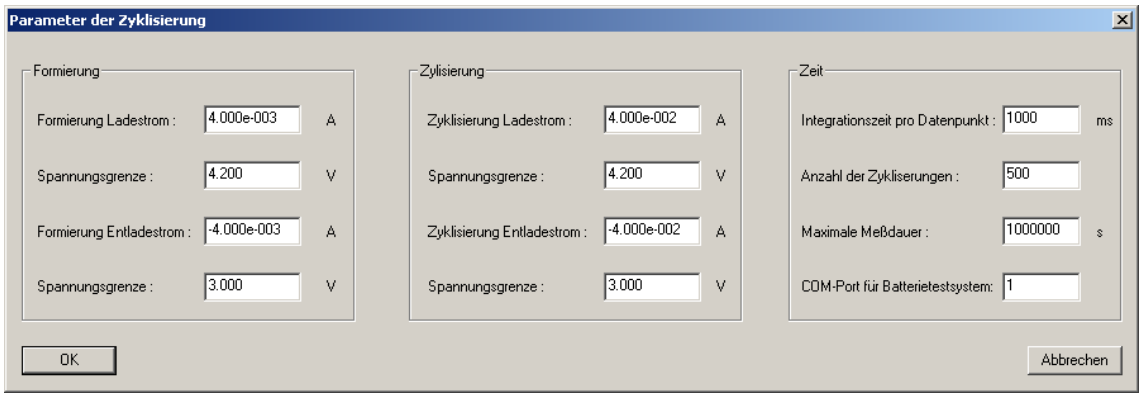


Abbildung 90 Startdialog des Zyklisiergeräts

Sind mehrere Systeme an einen Rechner angeschlossen, muss für jedes Testsystem eine eigene Instanz der Steuersoftware gestartet werden. Die Auswahl eines Geräts erfolgt dann in diesem Dialogfeld über die Nummer des COM-Ports, an den das jeweilige Batterietestsystem angeschlossen ist.
Mit dem OK-Button wird dann die Messung gestartet, woraufhin die aktuellen Messergebnisse in Textform dargestellt werden.

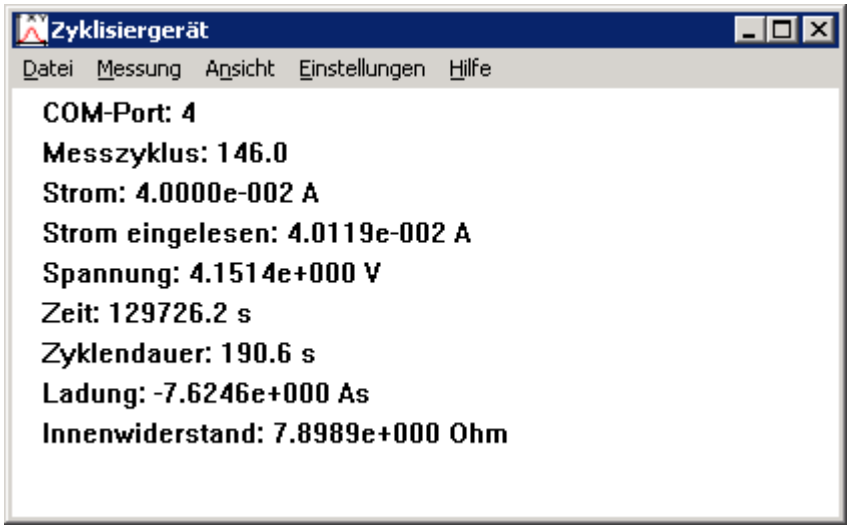


Abbildung 91 Messung mit dem Zyklisiergerät

Über den Menüpunkt „Ansicht, Display umschalten“ kann in eine graphische Darstellung umgeschaltet werden, in welcher der Strom durch die Batterie und ihre Klemmenspannung in Abhängigkeit von der Zeit dargestellt werden. Da aber durch die große Menge der erfassten Datenpunkte für die graphische Darstellung viele Systemressourcen erforderlich sind, besteht die Möglichkeit, das Multitasking dieses Fensters einzuschränken. Die Messung selbst, wie auch die anderen Programme, werden dadurch nicht beeinflusst.
Wurden alle Messzyklen ausgeführt oder wurde die maximale Messdauer erreicht, erscheint ein Dialogfeld, in dem der Benutzer aufgefordert wird, die Messdaten abzuspeichern. Nachdem die Messdaten gesichert sind, können auch noch die Daten der einzelnen Zyklen gespeichert werden, siehe auch Abschnitt 5.3.
Da ein Batterie-Zyklisierungsexperiment mehrere Wochen in Anspruch nehmen kann, besteht die Gefahr, dass die Messung durch Störungen, wie zum Beispiel einen Stromausfall, unterbrochen wird. Die Speicherung der Messdaten erfolgt erst am Ende der Messung, somit wären diese dann verloren. Daher werden alle Messdaten sofort nach der Erfassung in einer temporären Datei zwischengespeichert. Der Dateipfad dieser Dateien kann im INI-File des Messprogramms (zyklisiergerät.ini), das sich im Windowsordner befindet, eingestellt werden. Diese Datei wird mit dem Menüpunkt „Einstellungen, Optionen speichern“ erzeugt.

11.4.2.3 Zusätzliche Funktionen der Software

Mit dem Menüpunkt „Messung, am Zyklenende anhalten“ wird die Messung am Ende des laufenden Zyklus unterbrochen, um beispielsweise ein Impedanzspektrum aufzunehmen. Danach kann die Messung wieder fortgesetzt werden. Ein Messzyklus wird mit dem Menüpunkt „Messung, nächster Zyklus“ sofort beendet. Die Messung wird dann beim nächsten Schritt fortgesetzt. Der Menüpunkt „Messung beenden“ im Menü „Messung“ beendet die Messung sofort.
Über „Datei, speichern“ können Messdaten gespeichert werden. Der COM-Port, an den das Testsystem angeschlossen wird, kann mit dem Menüpunkt „Einstellungen, Optionen“ voreingestellt werden. Diese Einstellung kann mit „Einstellungen, Optionen speichern“ gesichert werden.
Eine Versionsinformation wird mit „Hilfe, Info“ erhalten.

11.4.2.4 Beschreibung des Datenformats

Alle Dateien, die das Batterietestsystem erzeugt, sind ASCII kodiert, wodurch sie leicht in viele Datenauswertungsprogramme importiert werden können. Die Spaltentrennung erfolgt dabei durch ein Tabulatorzeichen.

Rohdaten

Die Rohdaten enthalten die Zeit t, den Strom I, die Zellspannung U und die Spannungen der einzelnen A/D-Wandlerkanäle U_{ADC1} und U_{ADC2}. Ein Beispiel für diese Daten ist in Tabelle 54 aufgeführt.

t / s	I / A	U / V	[61]	U_{ADC1} / V	U_{ADC2} / V	[61]	[61]
1,01717	0,00400903	0,247148	0	4,25618	4,00903	0	0
2,21863	0,00400866	0,438539	0	4,4472	4,00866	0	0
3,4304	0,00400954	0,580442	0	4,58999	4,00955	0	0
4,65214	0,00400884	0,690948	0	4,69979	4,00884	0	0
5,87396	0,00400957	0,779664	0	4,78923	4,00957	0	0
7,08563	0,00400922	0,867783	0	4,877	4,00922	0	0
8,30744	0,00400915	0,945865	0	4,95502	4,00915	0	0

Tabelle 54 Datenformat der Rohdaten

[61] Diese Felder sind für zukünftige Erweiterungen wie zusätzliche Sensoren vorgesehen

Zyklendaten

Die Datei mit den Zyklendaten enthält die Daten, welche die einzelnen Messzyklen charakterisieren. Sie setzen sich aus der Nummer des Zyklus, der Zyklendauer, dem aufgeprägten Strom I, der umgesetzten Ladung Q und dem Innenwiderstand der Zelle R_i zusammen. In Tabelle 55 ist ein Beispiel für diese Daten aufgeführt.

Zyklus	Zyklendauer / s	I / A	Q / As	R_i / Ω
0	2,764978E+04	4,000000E-03	1,105991E+02	3,481625E+01
0,5	1,380256E+04	-4,000000E-03	-5,521023E+01	2,031136E+00
1	1,387906E+03	4,000000E-02	5,551624E+01	1,105250E+00
1,5	1,292399E+03	-4,000000E-02	-5,169594E+01	1,654000E+00
2	1,352255E+03	4,000000E-02	5,409018E+01	9,947500E-01
2,5	1,318516E+03	-4,000000E-02	-5,274064E+01	1,721125E+00
3	1,325426E+03	4,000000E-02	5,301704E+01	1,038250E+00

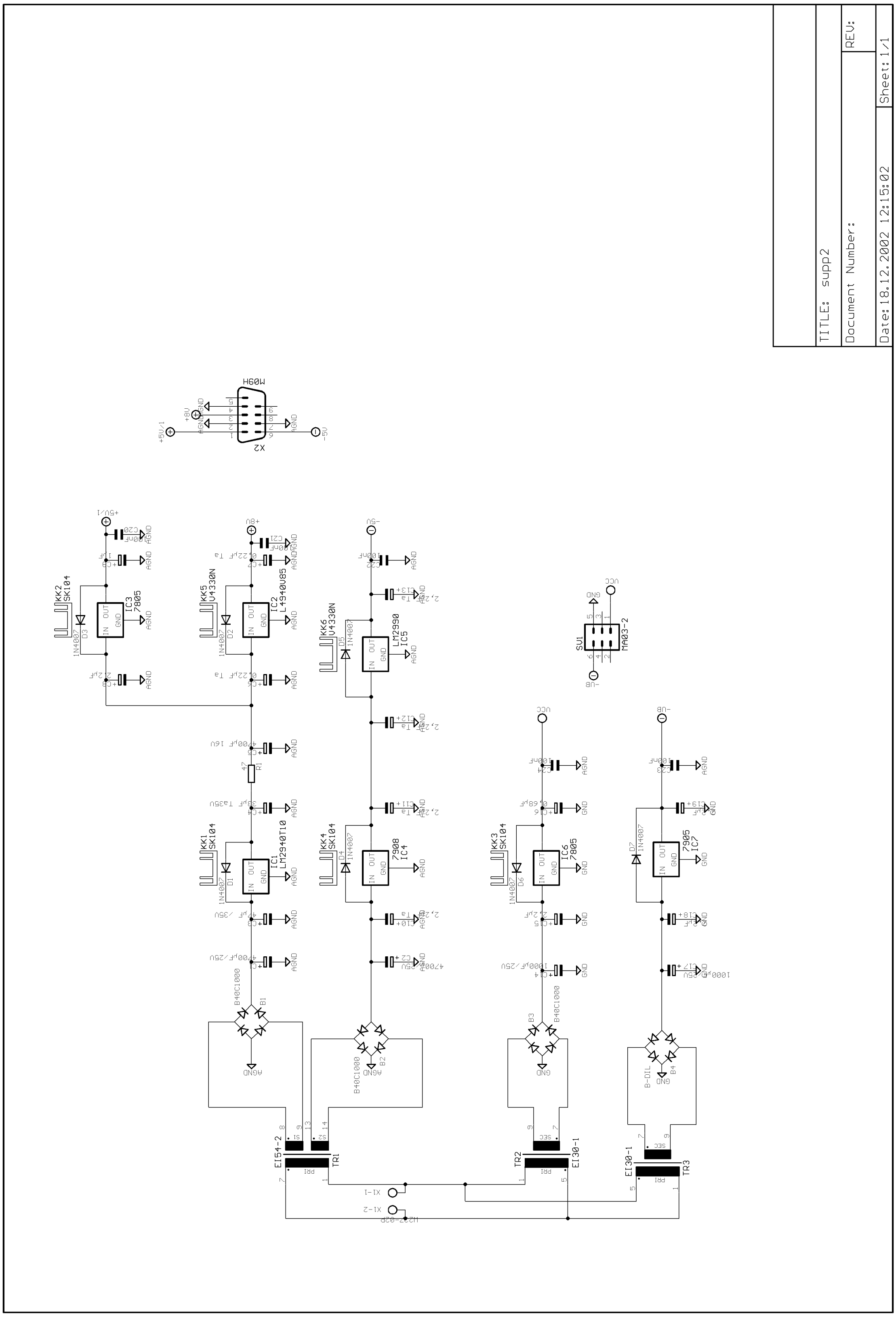
Tabelle 55 Datenformat der Zyklendaten

Ganzzahlige Zyklennummern bezeichnen in dieser Datei Ladezyklen, halbzahlige hingegen Entladezyklen.

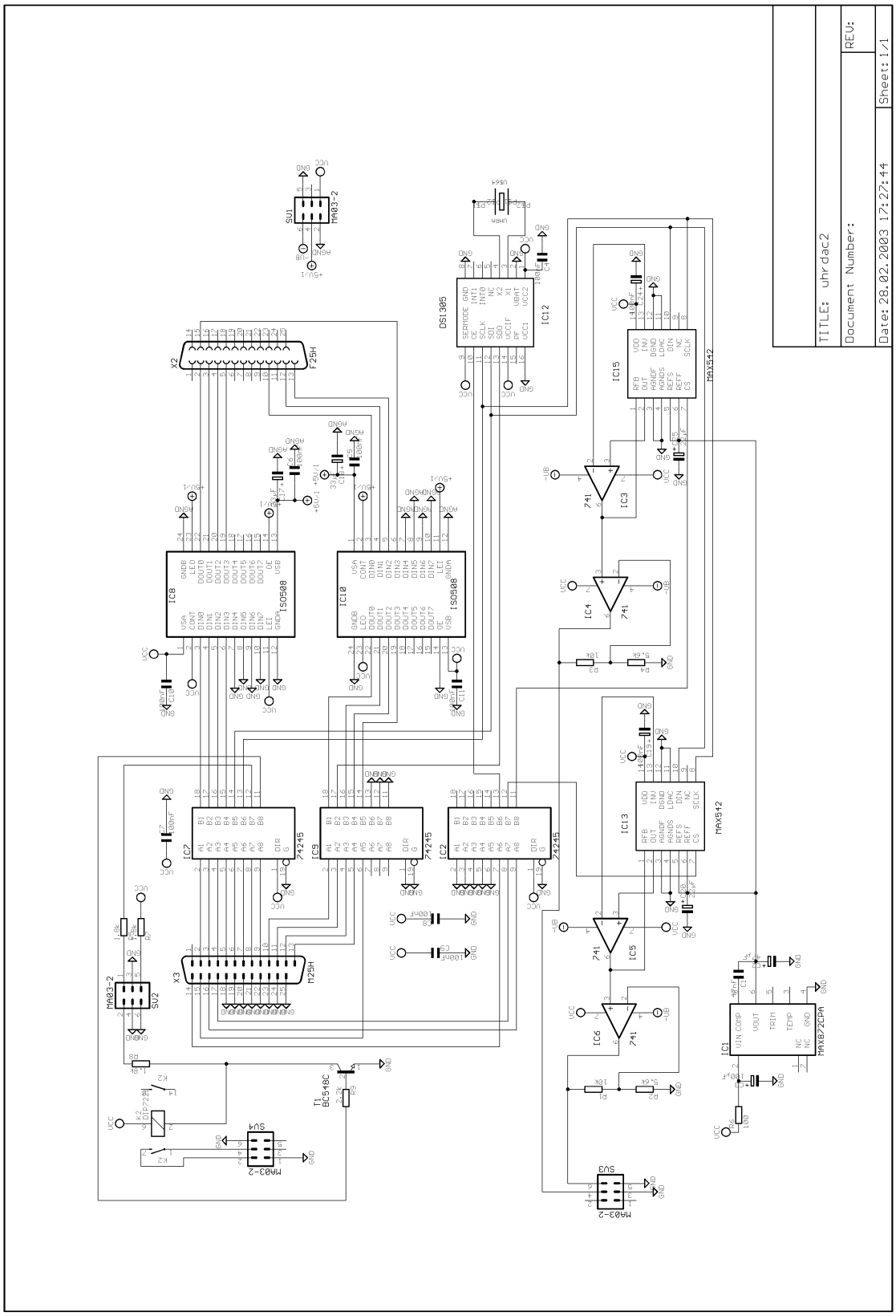
11.5 Schaltpläne

11.5.1 Thermometer

11.5.1.1 Netzteil

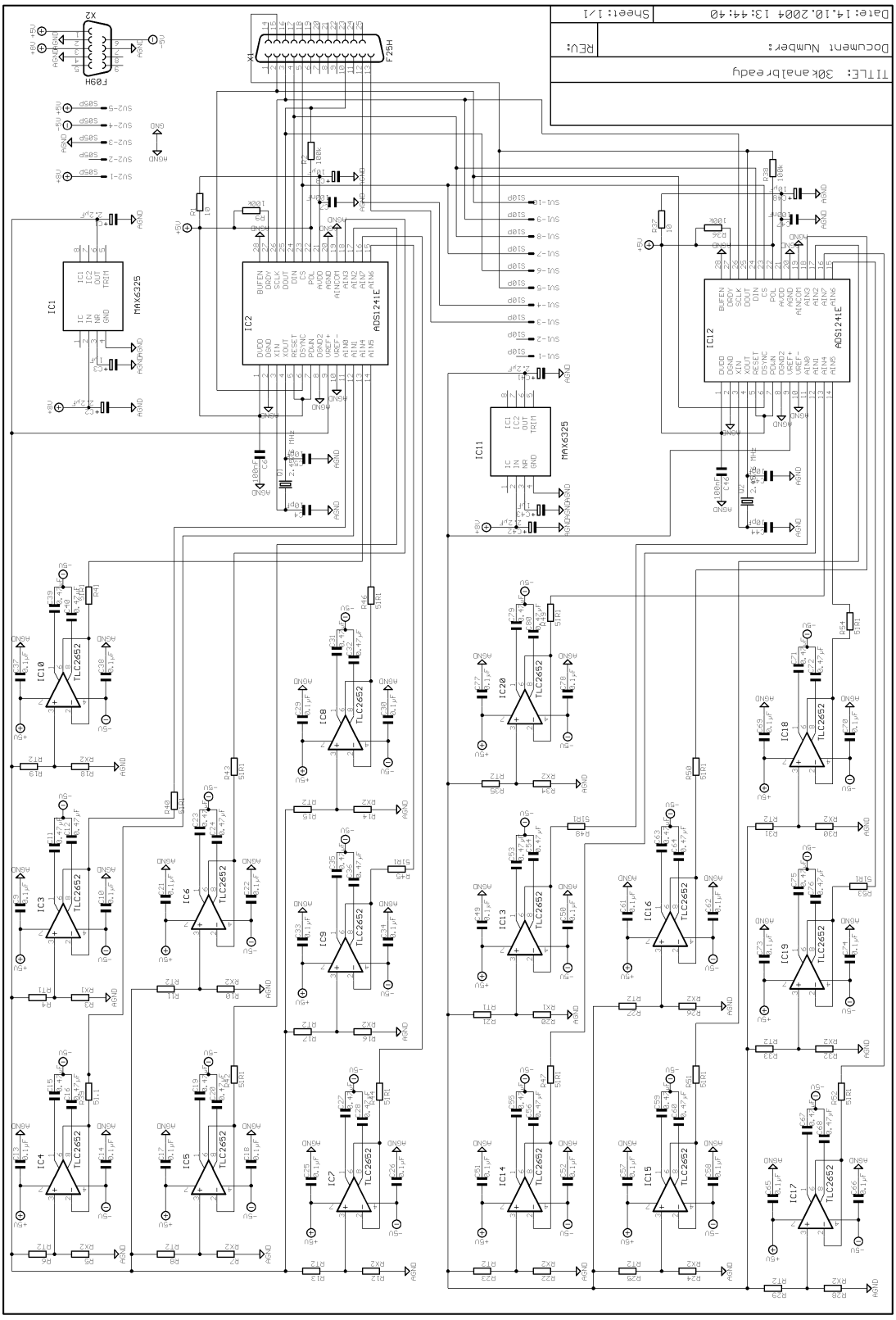

TITLE: supp2
Document Number:
REV:
Date: 18.12.2002 12:15:02
Sheet: 1/1
X2
M09H
SU1
MA03-2
IC1 LM2940T10
IC2 L4940V85
IC3 7805
IC4 7908
IC5 LM2990
IC6 7805
IC7 7905
TR1 EI54-2
TR2 EI30-1
TR3 EI30-1
B40C1000
B-DIL

11.5.1.2 Digitalteil


TITLE: uhrdac2
Document Number:
REV:
Date: 26.02.2003 17:27:44
Sheet: 1/1

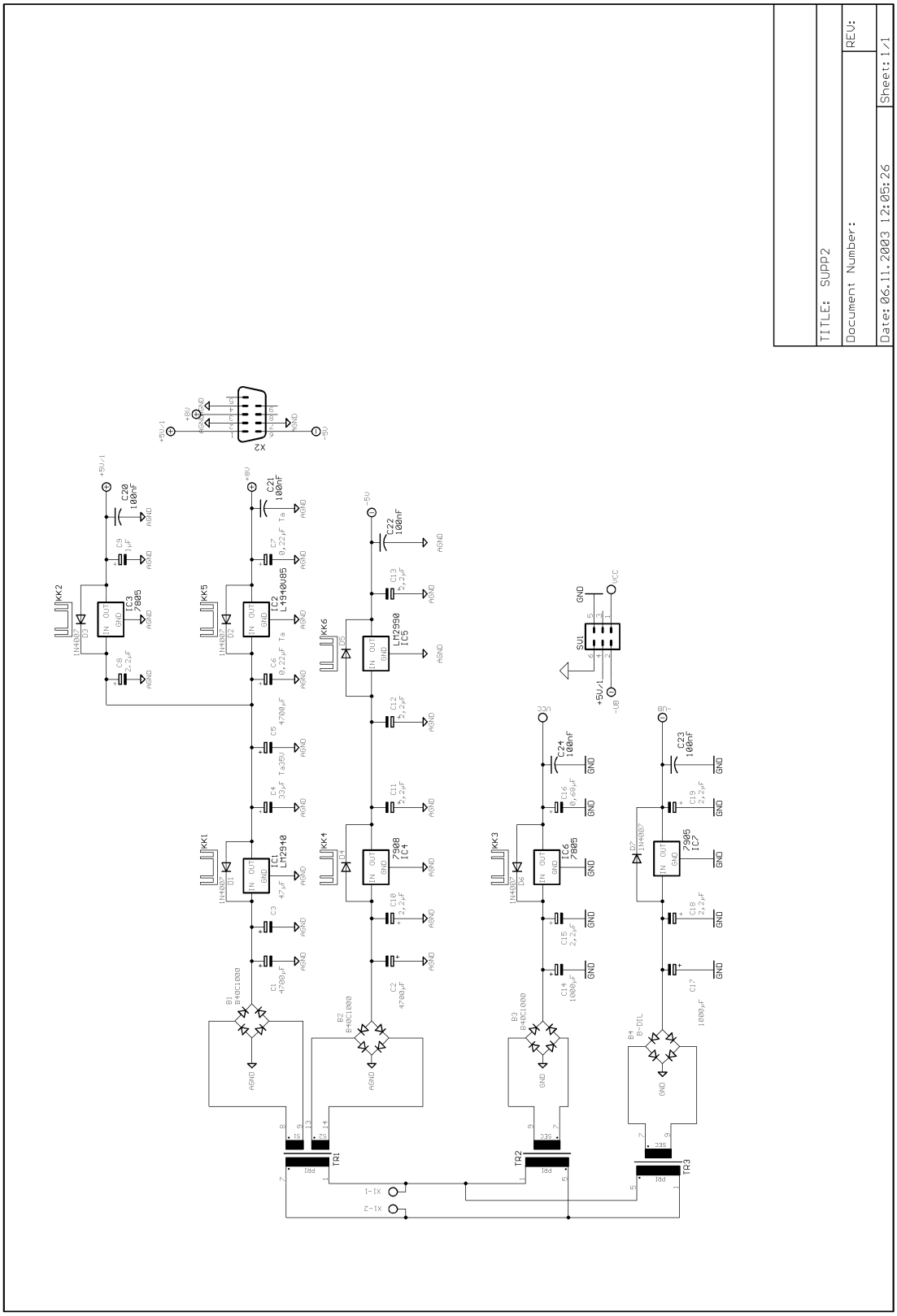
11.5.1.3 Analogteil

Im Schaltplan ist nur eine der beiden Analogplatinen mit je 16-Kanälen dargestellt.

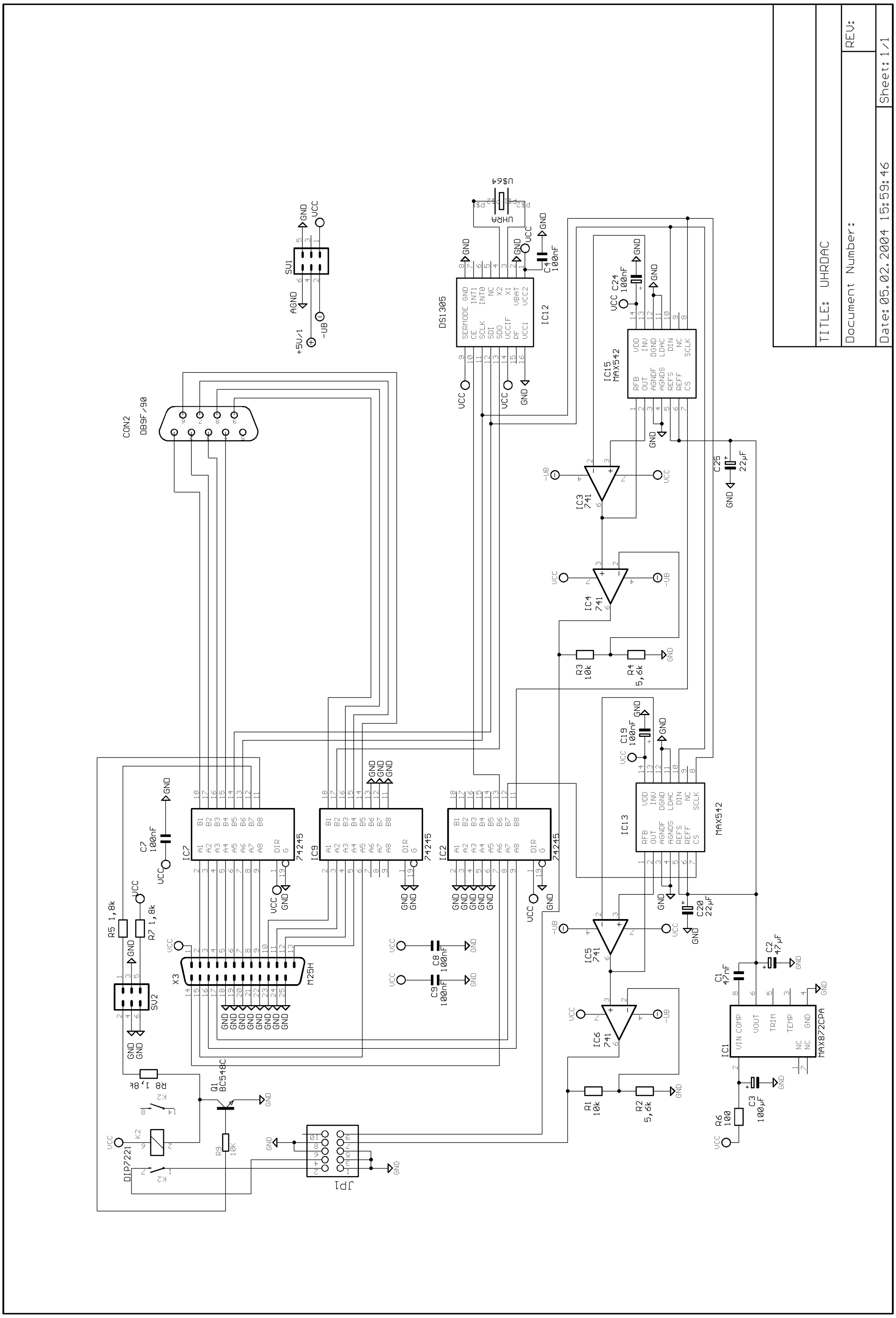

TITLE: 30k analbr eady
Document Number:
REV:
Date: 14.10.2004 13:44:40
Sheet: 1/1
X1
F25H
X2
F09H
IC1
MAX6325
IC2
ADS1241E
IC11
MAX6325
IC12
ADS1241E
IC3
IC4
IC5
IC6
IC7
IC8
IC9
IC10
IC13
IC14
IC15
IC16
IC17
IC18
IC19
IC20
TLC2652
+5U
-5U
AGND
+8U

11.5.2 Leitfähigkeitsmessgerät

11.5.2.1 Netzteil

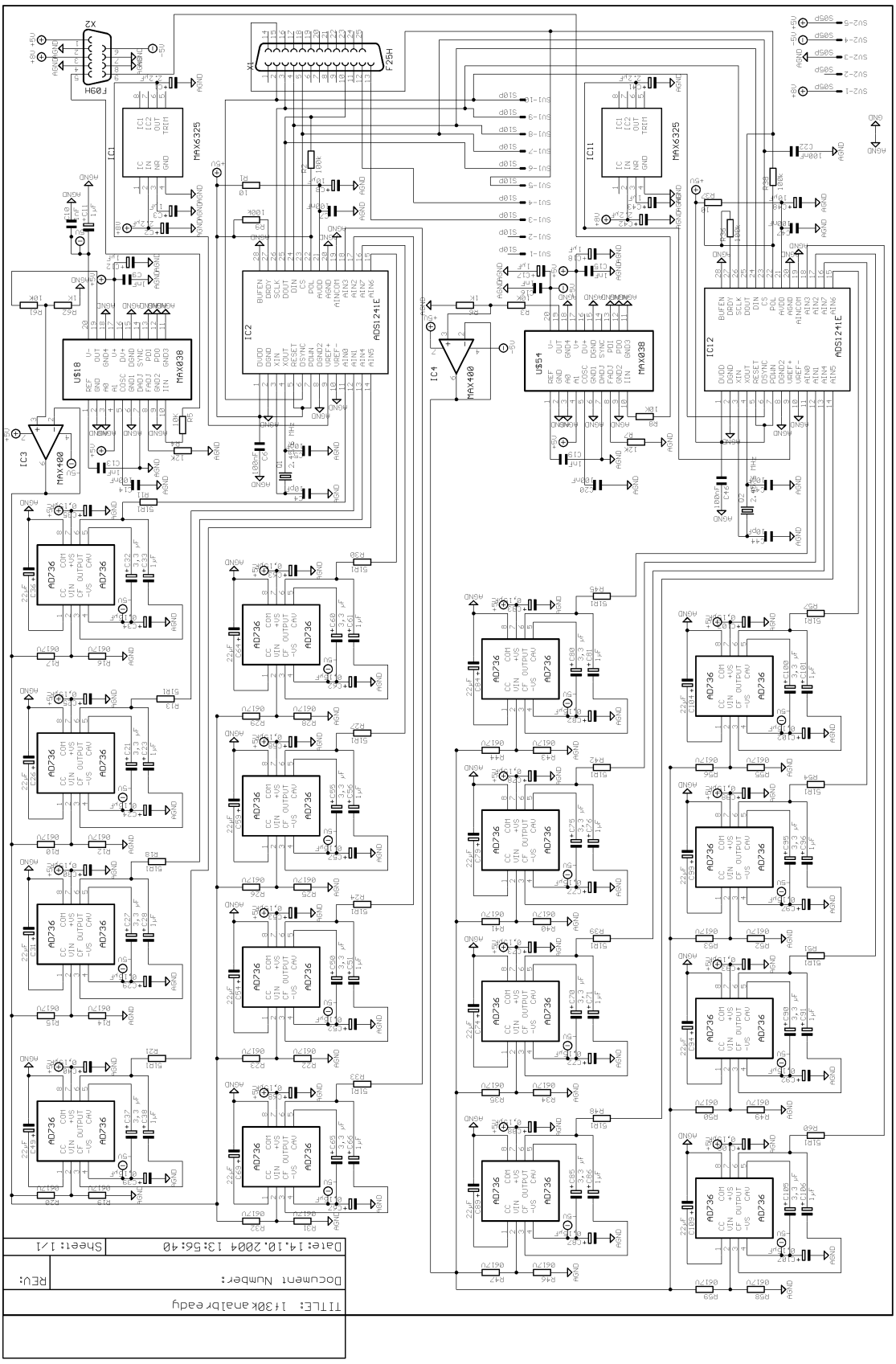

TITLE: SUPP2
Document Number:
REV:
Date: 06.11.2003 12:05:26
Sheet: 1/1

11.5.2.2 Digitalteil

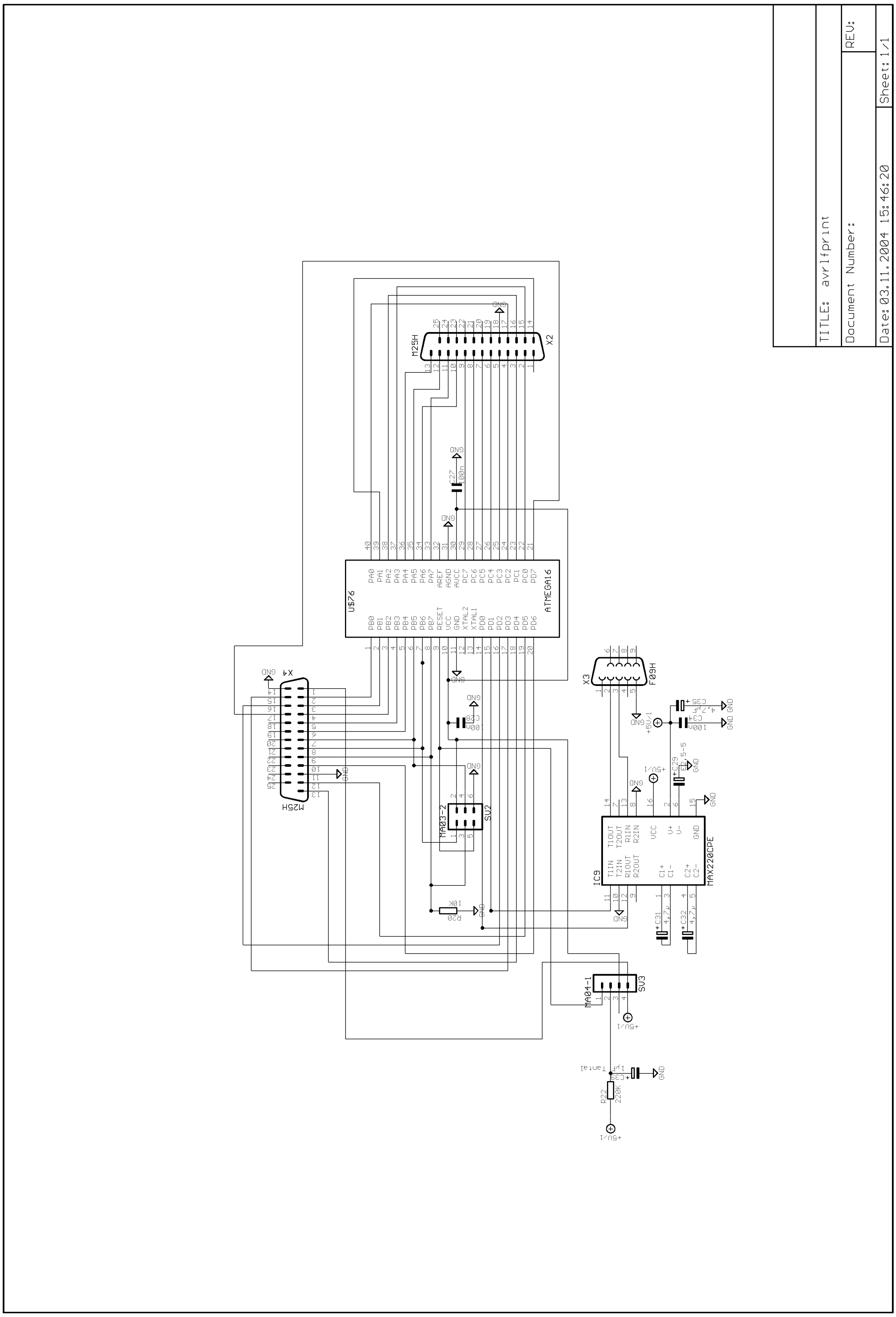

TITLE: UHRDAC
Document Number:
REV:
Date: 05.02.2004 15:59:46
Sheet: 1/1

11.5.2.3 Analogteil

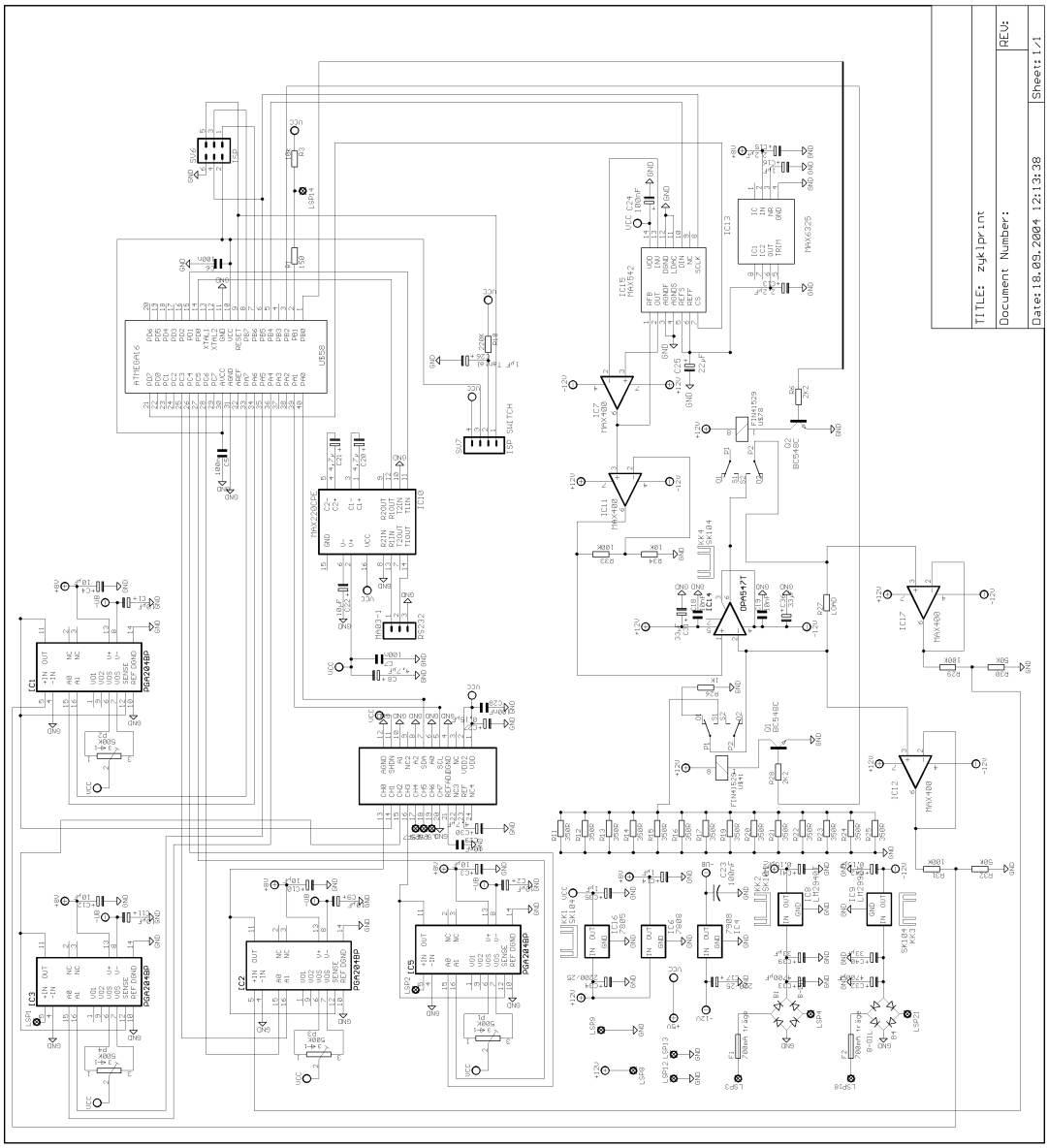
Im Schaltplan ist nur eine der beiden Analogplatinen mit je 16-Kanälen dargestellt.


TITLE: 14bit analbr eady
Document Number:
REV:
Date: 14.10.2004 13:56:40
Sheet: 1/1

11.5.3 Mikrocontrollerplatine


TITLE: avrlfprint
Document Number:
REV:
Date: 03.11.2004 15:46:20
Sheet: 1/1
U$76
ATMEGA16
X2
M25H
X4
M25H
X3
F09H
IC9
MAX220CPE
SV2
MA03-2
SV3
MA04-1
C27 100n
C28 100n
C29 EL5-5
C31 4,7µ
C32 4,7µ
C34 100n
C35 4,7µF
C39 1µF Tantal
R20 10K
R22 220K
GND
+5V/1

11.5.4 Batterietestsystem


TITLE: zyklprint
Document Number:
REV:
Date:18.09.2004 12:13:38
Sheet: 1/1

11.6 Literaturverzeichnis

[1] M. Eberwein, *Darstellung und elektrochemsiche Charakterisierung von oxidationsstabilen Lithiumboraten und Lithiumphosphaten für den Einsatz in Lithium-Ionen-Zellen, Diplomarbeit*, Regensburg (2000)
[2] Y. Chen, T. M. Devine, J. W. Evans, O. R. Monteiro, und I. G. Brown *J. Electrochem. Soc.* **146**, 1310 (1999)
[3] J. W. Braithwaite, A. Gonzales, G. Nagasubramanian, S. J. Lucero, D. E. Peebles, J. A. Ohlhausen, und Wendy R. Cieslak, *J. Electrochem. Soc.* **146**, 448 (1999)
[4] H. Bruglachner, *mündl. Mitteilung,* (2004)
[5] B. Burrows und S. Kirkland, *J. Electrochem. Soc.*, **115**, 1164 (1968)
[6] M. Dekker, *Nonaqueous Electrochemistry*, New York (1999)
[7] D. Aurbach, M. Daroux, P. Faguy und E. Yeager, *J. Electroanal. Chem. Interfacial Electrochem.*, **297**, 225 (1991)
[8] M. Moshkovich, Y. Gofer und D. Aurbach, *J. Electrochem. Soc*, **148**, E155 (2001)
[9] D. Aurbach, *J. Electrochem. Soc.*, **136**, 906 (1989)
[10] H. E. Gottlieb, V. Kotlyar und A. Nudelman, *J. Org. Chem.*, **62**, 7512 (1997)
[11] C. J. Pouchert und J. Behnke, *The Aldrich library of 13 C and 1 H FT NMR Spectra*, Milwaukee
[12] J.-i. Kadokawa, H. Hideyuki, S. Fukamachi, M. Karasu, H. Tagaya und K. Chiba, *J. Chem. Soc., Perkin Trans. 1: Org. and Bio-Org. Chem.*, 2205 (1999)
[13] U. Lischka, U. Wietelmann und M. Wegner, *Lithium bis(oxalato)borate, method for its production and application*, DE 19829030, (1999)
[14] R. Bhanumathi und S. K. Vijayalakshamma, *J. Phys. Chem.*, **90**, 4666 (1986)
[15] V. Maemets und I. Koppel *J. Chem. Soc., Faraday Trans.*, **93**, 1539 (1997)
[16] J. C. Panitz, *mündl. Mitteilung,* (2004)
[17] J. A. Riddick, W. B. Bunger, T. K. Sakano, *Organic Solvents. Physical Properties and Method of Purification, 4th ed., Techniques of Chemistry,* Vol. II (Hrsg. A. Weissberger), Wiley, New York, (1986)
[18] J. F. Coetzee (Ed.*), Recommended Methods for Purification of Solvents and Tests for Impurities*, Pergamon, Oxford, (1982)
[19] D. D. Perrin, W. L. F. Armarego, D. R. Perrin, *Purification of Laboratory Chemicals,* 3rd ed., Pergamon, Oxford, (1988)
[20] H.-G. Schweiger, *Entwicklung einer Präzisionstemperaturmessanlage zur schnellen Messung von Phasendiagrammen und chemische und elektrochemische Charakterisierung von Lithium-bis[oxalato(2-)]borat(1-)*, *Diplomarbeit*, Regensburg (2002)
[21] *Econistor 8E16&8E24,* Rhopoint Components Ltd., Oxted, UK, 2002
[22] *ADS1240/ADS1241 24-Bit analog-to-digital converter,* Texas Instruments Inc., Dallas, TX, 2000
[23] TLC2652, TLC2652A, TLC2652Y, Advanced LinCMOS Precision Chopper-Stabilized Operational Amplifiers, Texas Instruments Inc., Dallas, TX, 2001
[24] MAX038 *High-Frequency Waveform Generator*, Maxim Integrated Products Inc.,Sunnyvale,CA, 2001
[25] Maxim Integrated Products Inc., *MAX400 Ultra-Low Offset Voltage Operational Amplifier,* Sunnyvale (2003)
[26] Rhopoint Components LTD, *Datenblatt zu Econistor 8E16&8E24* (2002)
[27] Maxim Integrated Products Inc., *MAX541, MAX542 +5V, Serial-Input, Voltage-Output 16-Bit DACs,* Sunnyvale (1999)

[28] Maxim Integrated Products Inc., *1ppm/°C, Low-Noise, +2.5V/+4.096V/+5V Voltage References,* Sunnyvale (2001)
[29] Maxim Integrated Products Inc., *MAX127/128 Multirange, +5V, 12-Bit DAS with 2-Wire Serial Interface*, Sunnyvale (1998)
[30] F. Wudy, *Bedienungsanleitung Phasendiagrammmessanlage,* Regensburg 2004

11.7 Quelltexte

11.7.1 Phasendiagrammmessgerät, Mikrokontroller

11.7.1.1 lfundtemp.c

```
/**Steuerprpgramm für Leitfähigkeitsmessgerät und Thermometer*/
#include <stdlib.h>
#include <math.h>
#include <avr/pgmspace.h>
#include <string.h>
#include <stdio.h>
#include <avr/eeprom.h>
#include "ds1305.h"
#include "small.h"
#include "max542.h"
#include "ads1241e.h"

#define UREF 2.5
#define PGA 1
#define KALA 1.068981e-3
#define KALB 2.120700e-4
#define KALC 9.019537e-8
#define KALU 2.5
#define KALR 100e3
#define ADPORT PORTB

// Datentypdefinitionen
typedef unsigned char       BYTE;
typedef unsigned short WORD;

// Globale Variablen

int main(void) {
double u1=1,u2=1,u3=1,u4=1,u5=1,u6=1,u7=1,u8=1;

char intext[50]={""},outtext[200]={""};
char dummy[20]={""};
int j=0,i=0,kanal=0;
double u=0;
double zeit=0;

        DDRA=0x7; // nur A0 bis A1 als output
        DDRB=0xff; // port b als output
        DDRC=0xff; // port b als output
        DDRD=0x84; // nur D7, D2 als output
        USART_Init(25); //19200 Baud bei 8 MHz

        frequenzausgabe(5000);
        ads1241reset();
        ads1241klarmachen();
        startclock();
        USART_Transmit_string("Dateneingeben \r\n"); // an RS232 ausgeben*/
        do
        {
                do
                {
                        intext[i++]=USART_Receive();
                }
                while((i<50) && (intext[i-1]!='\r'));
```

```
                do // parsen
                {
                        i--;
                }
                while( (i>1) && (intext[i]!='h'));

                //hglm1
                if((intext[i+1]=='g') && (intext[i+2]=='l') && (intext[i+3]=='m') && (intext[i+4]=='1'))
// LED messen Leitfähigkeit an
                {
                        ledmessenlf(1);
                }

                //hglm0
                if((intext[i+1]=='g') && (intext[i+2]=='l') && (intext[i+3]=='m') && (intext[i+4]=='0'))
// LED messen Leitfähigkeit aus
                {
                        ledmessenlf(0);
                }

                //hgtm1
                if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='m') && (intext[i+4]=='1'))
// LED messen Thermo an
                {
                        ledmessenthermo(1);
                }

                //hgtm0
                if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='m') && (intext[i+4]=='0'))
// LED messen Thermo aus
                {
                        ledmessenthermo(0);
                }

                //hgts1
                if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='s') && (intext[i+4]=='1')) //
LED Störung Thermo an
                {
                        stoerungthermo(1);
                }

                //hgts0
                if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='s') && (intext[i+4]=='0')) //
LED Störung Thermo aus
                {
                        stoerungthermo(0);
                }
                //hglr1
                if((intext[i+1]=='g') && (intext[i+2]=='l') && (intext[i+3]=='r') && (intext[i+4]=='1')) //
Wandelfeldrührer an
                {
                        wandelfeldruehrer(1);
                }
                //hglr0
                if((intext[i+1]=='g') && (intext[i+2]=='l') && (intext[i+3]=='r') && (intext[i+4]=='0')) //
Wandelfeldrührer aus
                {
                        wandelfeldruehrer(0);
                }

                //hgtr___
                if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='r')) // Magnetrührer an
```

```
{
        if(intext[i+4]=='1' && intext[i+5]=='0' && intext[i+6]=='0')
        {
                motor(100);
        }
        else
        {
                if(intext[i+4]!='\r')
                        dummy[0]=intext[i+4];
                else
                        dummy[0]=0;

                if(intext[i+5]!='\r')
                        dummy[1]=intext[i+5];
                else
                        dummy[1]=0;

                dummy[2]=0;
                u=atol(dummy);
                motor((double)u);
        }
}

//hglf_______
if((intext[i+1]=='g') && (intext[i+2]=='l') && (intext[i+3]=='f')) // Frequenzausgeben
{
        if(intext[i+4]!='\r')
                dummy[0]=intext[i+4];
        else
                dummy[0]=0;

        if(intext[i+5]!='\r')
                dummy[1]=intext[i+5];
        else
                dummy[1]=0;

        if(intext[i+6]!='\r')
                dummy[2]=intext[i+6];
        else
                dummy[2]=0;

        if(intext[i+7]!='\r')
                dummy[3]=intext[i+7];
        else
                dummy[3]=0;

        if(intext[i+8]!='\r')
                dummy[4]=intext[i+8];
        else
                dummy[4]=0;

        dummy[5]=0;

        u=atol(dummy);
        frequenzausgabe((double) u);
}

//hgtu_______
if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='u')) // Steuerspannung für
Heizer ausgeben Spannung in µV
{
        if(intext[i+4]!='\r')
                dummy[0]=intext[i+4];
```

```
                else
                        dummy[0]=0;

                if(intext[i+5]!='\r')
                        dummy[1]=intext[i+5];
                else
                        dummy[1]=0;

                if(intext[i+6]!='\r')
                        dummy[2]=intext[i+6];
                else
                        dummy[2]=0;

                if(intext[i+7]!='\r')
                        dummy[3]=intext[i+7];
                else
                        dummy[3]=0;

                if(intext[i+8]!='\r')
                        dummy[4]=intext[i+8];
                else
                        dummy[4]=0;

                if(intext[i+9]!='\r')
                        dummy[5]=intext[i+9];
                else
                        dummy[5]=0;

                if(intext[i+10]!='\r')
                        dummy[6]=intext[i+10];
                else
                        dummy[6]=0;

                dummy[7]=0;

                //USART_Transmit_string(dummy); // an RS232 ausgeben*/
                u=atol(dummy);

                //sprintf(outtext,"%lf \r\n", zeit);
                u_ausgabe(((double) u)/1e6);
        }

        //hgta1
        if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='a') && (intext[i+4]=='1')) //
Uhr an
        {
                startclock();
        }

        //hgta0
        if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='a') && (intext[i+4]=='0')) //
Uhr aus
        {
                clockoff(); //schaltet den Oszillator auf dem ds1305 aus
        }

        //hgtz
        if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='z')) // Uhrzeit einlesen
        {
                zeit=gettime();
                sprintf(outtext,"%.0lf \r\n", zeit);
                USART_Transmit_string(outtext); // an RS232 ausgeben*/
        }
```

```
                //hgrr
                if((intext[i+1]=='g') && (intext[i+2]=='r') && (intext[i+3]=='r')) // reset
                {
                        ads1241reset();
                        ads1241klarmachen();
                }

                //hg32
                if((intext[i+1]=='g') && (intext[i+2]=='3') && (intext[i+3]=='2')) // alle 32 Kanäle
ausgeben
                {
                        for(j=1;j<=8;j++) // Kanäle am Wandler durchschalten
                        {
                                zeit=gettime();
                                ads1241getu30(1,j,&u1,&u2,&u3,&u4,&u5,&u6,&u7,&u8);
                                sprintf(outtext,"%i %.0lf %lf %lf %lf %lf %lf %lf %lf %lf\r\n",
j,zeit,u1,u4,u3,u2,u5,u8,u8,u8); // achtung kanalzuordung vertauscht
                                USART_Transmit_string(outtext);
                        }
                        USART_Transmit_string("E");
                }

                //hgmm
                if((intext[i+1]=='g') && (intext[i+2]=='m') && (intext[i+3]=='m')) // Kanäle messen
                {
                        kanal=(int) intext[i+4]-'1'+1;
                        zeit=gettime();
                        ads1241getu30(1,kanal,&u1,&u2,&u3,&u4,&u5,&u6,&u7,&u8);
                        sprintf(outtext,"%i %.1lf %lf %lf %lf %lf %lf %lf %lf %lf\r\n",
j,zeit,u1,u4,u3,u2,u5,u8,u6,u7); // achtung kanalzuordung vertauscht
                        USART_Transmit_string(outtext);
                        USART_Transmit_string("E");
                }

                /*if((intext[i+1]=='i') && (intext[i+2]=='l') && (intext[i+3]=='f')&& (intext[i+4]=='e'))
// Hilfe
                {
                        USART_Transmit_string("hgmmX Messung des Kanals X\r\n hg32 Misst alle
Kanäle\r\n hgrr Reset ADCs\r\n hgtz Zeit Uhrzeitan\r\nhgtaX Uhr an/aus (X=1/0) \r\n hgtuX
Spannunsausgabe X in µV\r\nhglfX Frequenzausgabe X in Hz\r\nhglrX Wanderfeldrührer an/aus (X=1/0)
\r\nhgtrX Zahnradrührer X=0-100%\r\n hgtsX Led Störung an/aus (X=1/0) \r\n hgtmX Led Messung
Thermometer an/aus (X=1/0) \r\n hglmX Led Messung Leitfähigkeit an/aus (X=1/0) \r\n");
                }                       */

                i=0;
        }
        while(1);

};
```

11.7.1.2 ds1305.c

```
// funktionen für die Ansteuerung des ds1305 Uhrchips

#include "ds1305.h"
#include "avr/pgmspace.h"
extern void warten1(void);

void startclock(void) //startet den Oszillator auf dem ds1305 und setzt das 24h bit (0x82 bit 6) auf 24h
modus
{
```

```
int data=0;
        ds1305byteread(0x0f,&data);
        ds1305bytewrite(0x8f,data&63); // schreibt in das Kontollregister eine 0 --> oszi an und schreiben
an

        ds1305bytewrite(0x80,0); //ganze Uhr auf null
        ds1305bytewrite(0x81,0);
        ds1305bytewrite(0x82,0); //setzt auch Uhr in 24h modus
        ds1305bytewrite(0x83,0);
        ds1305bytewrite(0x84,0);
        ds1305bytewrite(0x85,0);
        ds1305bytewrite(0x86,0);

        ds1305byteread(0x0f,&data);
        ds1305bytewrite(0x8f,(data&191)+64); // schreibt in das Kontollregister bit 6 eine 1 --> und
schreiben aus*/
}

void clockoff(void) //schaltet den Oszillator auf dem ds1305 aus
{
int data=0;
        ds1305byteread(0x0f,&data);
        ds1305bytewrite(0x8f,(data&127)+128); // schreibt in das Kontollregister auf das bit7 eine 1
}

double gettime(void)
{
double sec1=0,sec2=0;
int dummy=0;

        // das erste mal die Uhrzeit holen
        ds1305byteread(0x04,&dummy); //liest tage
        sec1=(double)( (dummy>>4*10) + (dummy&0x0f) )*24*3600; //BCD dekodieren
        dummy=0;

        ds1305byteread(0x02,&dummy); //liest Stunden
        sec1+=(double)( ((dummy>>4)&0x01)*10+ ((dummy>>5)&0x01)*20 + (dummy&0x0f) )*3600;
//BCD dekodieren
        dummy=0;

        ds1305byteread(0x01,&dummy); //liest Minuten
        sec1+=(double)( (dummy>>4)*10 + (dummy&0x0f) )*60; //BCD dekodieren
        dummy=0;

        ds1305byteread(0x00,&dummy); //liest Sekunden
        sec1+=(double)(dummy>>4)*10 + (dummy&0x0f); //BCD dekodieren
        dummy=0;

        // zum zweiten Mal die Uhrzeit einlesen
        ds1305byteread(0x04,&dummy); //liest Tage
        sec2=(double)( (dummy>>4*10) + (dummy&0x0f) )*24*3600; //BCD dekodieren
        dummy=0;

        ds1305byteread(0x02,&dummy); //liest Stunden
        sec2+=(double)( ((dummy>>4)&0x01)*10+ ((dummy>>5)&0x01)*20 + (dummy&0x0f) )*3600;
//BCD dekodieren
        dummy=0;

        ds1305byteread(0x01,&dummy); //liest Minuten
        sec2+=(double)( (dummy>>4)*10 + (dummy&0x0f) )*60; //BCD dekodieren
        dummy=0;

        ds1305byteread(0x00,&dummy); //liest Sekunden
```

```
	sec2+=(double)(dummy>>4)*10 + (dummy&0x0f); //BCD dekodieren

	if(sec2>=sec1) // falls beim erstenmal einlesen die Uhrzeit umgeschaltet wird, ist der Wert von sec1 kleiner als sec2 und sec2 gültig
		return(sec2);
	else // falls beim zweitenmal einlesen die Uhrzeit umgeschaltet wird, ist der Wert von sec2 kleiner als sec1 und sec1 gültig wenn man noch die nächste Sekunde hinzuaddiert
		return(sec1+1);
}

void ds1305bytewrite(int adresse, int data) // schreibt ein byte data in die Adresse des ds1305 am lpt-port
{
int bitwert,i;

	PORTA=(PINA & 0xfd) + 0x02;//PIN A1 CE auf HI ds1305 für Datenübertragung fertig
	warten1();

	for(i=7;i>=0;i--) //Daten notieren, Adresse wählen
	{
		PORTC=(PINC & 223)+32;//pin C5 HI; sckl auf hi
		warten1();
		bitwert=1<<i;

		if(adresse&bitwert) // HI ausgeben
		{
			PORTC=(PINC & 239) +16;// pin C4 HI; din auf hi
			warten1();
		}
		else // LO ausgeben
		{
			PORTC=(PINC & 239);// pin c4 lo; din auf lo
			warten1();
		}
		PORTC=(PINC & 223);//pin C5 lo; sckl auf lo
		warten1();
		bitwert=1<<i;
	}

	for(i=7;i>=0;i--) //Datenrausschreiben, Byte ausgeben
	{
		PORTC=(PINC & 223)+32;//pin C5 HI; sckl auf hi
		warten1();
		bitwert=1<<i;

		if(data&bitwert) // HI Ausgeben
		{
			PORTC=(PINC & 239) +16;// pin C4 HI; din auf hi
			warten1();
		}
		else // LO ausgeben
		{
			PORTC=(PINC & 239);// pin c4 lo; din auf lo
			warten1();
		}
		PORTC=(PINC & 223);//pin C5 lo; sckl auf lo
		warten1();
	}
	PORTA=(PINC & 0xfd);// CE auf lo ds1305 Datenübertragung Ende
	warten1();
}

void ds1305byteread(int adresse, int *data) // schreibt ein Byte data in die Adresse adresse des ds1305 am lpt-port ba
```

```
{
int bitwert,i,wert=0;

        PORTA=(PINA & 0xfd) + 0x02;//PIN A1 CE auf HI ds1305 für datenübertragung fertig stellen
        warten1();

        for(i=7;i>=0;i--) //Datennotieren, Adresse wählen
        {
                PORTC=(PINC & 223)+32;//pin C5 HI; sckl auf hi
                bitwert=1<<i;
                warten1();

                if(adresse&bitwert) // HI ausgeben
                {
                        PORTC=(PINC & 239) +16;// pin C4 HI; din auf hi
                        warten1();
                }
                else // LO ausgeben
                {
                        PORTC=(PINC & 239);// pin C4 lo; din auf lo
                        warten1();
                }
                PORTC=(PINC & 223);//pin C5 lo; sckl auf lo
                warten1();
        }

        bitwert = 128;

        for(i=7;i>=0;i--) //Daten notieren, Adresse einlesen
        {
                PORTC=(PINC & 223)+32;//pin C5 HI; sckl auf hi
                warten1();


                if((PINA & 0x80) != 0) //PIN C4; DIn einlesen
                        wert += bitwert;

                bitwert /= 2; // nächstniederes bit herrichten
                PORTC=(PINC & 223);//pin C5 lo; sckl auf lo
                warten1();
        }
        PORTA=(PINA & 0xfd);// CE auf lo ds1305 Datenübertragung Ende
        warten1();
        *data=wert;
}
```

11.7.1.3 small.c

```
#include "small.h"
#include "avr/pgmspace.h"
#include <math.h>

// setzt die LED-messen
void ledmessenthermo(int led)
{
int portbuffer=0;
        portbuffer=PINC;
        PORTC=(portbuffer & 0xbf) +0x40*led;
}


// setzt Störung srelais auf aus und Störungs- LED an
```

```
void stoerungthermo(int cond)
{
int portbuffer=0;
        portbuffer=PINC;
        PORTC=(portbuffer & 0x7f) +0x80*(!cond);
}

// setzt die LED-messen
void ledmessenlf(int led)
{
int portbuffer=0;
        portbuffer=PINB;
        PORTB=(portbuffer & 0xbf) +0x40*led;
}


// setzt Störungsrelais auf aus und Störungs- LED an
void wandelfeldruehrer(int cond) // schaltet Wandelfeldrührer an und aus
{
int portbuffer=0;
        portbuffer=PINB;
        PORTB=(portbuffer & 0x7f) +0x80*(!cond);
}

double round(double x)
{
 if((x-floor(x))<=0.5)
     return(floor(x));
   else
      return(ceil(x));
}

void warten1(void) // 1.2 µs Verzögerung
{
int i=0,j=0;
        for(i=0;i<86;i++)
        {
                for(j=0;j<10;j++)
                {
                }
        }
}

void pause (int ms)  // in ms bei 8 Mhz
{
int i,j,k,l;

         for (l=0; l<ms;l++)
         {
          for (i=0; i<88; i++) //88
           for(j=0; j<255;j++) //255
                k++;
         }

}

void USART_Init( unsigned int baud )
{

/* Set baud rate */
UBRRH = (unsigned char)(baud>>8);
UBRRL = (unsigned char)baud;
/* Enable receiver and transmitter */
```

```
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 2stop bit */
UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}

void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) )
;
/* Put data into buffer, sends the data */
UDR = data;
}

void USART_Transmit_string(unsigned char *data)
{
int i=0;

        do
        {
                USART_Transmit(data[i]);
        }
        while(data[++i]!=0);
}

unsigned char USART_Receive( void )
{
        /* Wait for data to be received */
        while ( !(UCSRA & (1<<RXC)) )
        ;

        /* Get and return received data from buffer */
        return UDR;
}
```

11.7.1.4 max542.c

```
// Funktionen für die Ansteuerung des DAC MAX542
#include "max542.h"
#include "hardware.h"
#include <avr/pgmspace.h>

void lpt_ausgebenthermo(unsigned char byte)
{
unsigned char bitwert, portwert;
int bit;
int portbuffer=0;

        portbuffer=PINC;

        for(bit=7; bit >=0; bit--)
        {
                bitwert = 1 << bit;
                if((byte & bitwert) == bitwert) portwert = 0x10; else portwert = 0;
                {
                        PORTC=(portbuffer & 0xef) + portwert; // PIN C4
                }

                portbuffer=(portbuffer&0xDF) + 0x20; /* SCLK = 1 bei DIN gesetzt */
                PORTC=portbuffer; // PIN C5

                portbuffer=portbuffer&0xDF; /* SCLK = 0 bei DIN gesetzt */
```

```
                PORTC=portbuffer; // PIN C5

        }
}

void lpt_ausgebenlf(unsigned char byte)
{
unsigned char bitwert, portwert;
int bit;
int portbuffer=0;

        portbuffer=PINB;

        for(bit=7; bit >=0; bit--)
        {
                bitwert = 1 << bit;
                if((byte & bitwert) == bitwert) portwert = 0x10; else portwert = 0;
                {
                        PORTB=(portbuffer & 0xef) + portwert; // PIN B4 DIN
                }

                portbuffer=(portbuffer&0xDF) + 0x20; /* SCLK = 1 bei DIN gesetzt */
                PORTB=portbuffer; // PIN B5

                portbuffer=portbuffer&0xDF; /* SCLK = 0 bei DIN gesetzt */
                PORTB=portbuffer; // PIN B5

        }
}

void max542thermo(double u,int dac) // gibt eine Spannung aus
{
unsigned short datenwort;
unsigned char lobyte, hibyte;

        if(u >= UREFMAX542) u = UREFMAX542*32767/32768; /* Ausgabe über Vref nicht möglich
*/
        if(u < -UREFMAX542) u = -UREFMAX542;
        u += UREFMAX542;
        datenwort = (unsigned short) 32768 * u / UREFMAX542; // Konvertierung ist absichtlich so
gewählt
        lobyte = datenwort & 255;
        hibyte = datenwort >> 8;

        csthermo(LO,dac);
        lpt_ausgebenthermo(hibyte);
        lpt_ausgebenthermo(lobyte);
        csthermo(HI,dac);
}

void max542lf(double u,int dac) // gibt eine Spannung aus
{
unsigned short datenwort;
unsigned char lobyte, hibyte;

        if(u >= UREFMAX542) u = UREFMAX542*32767/32768; /* Ausgabe über Vref nicht möglich
*/
        if(u < -UREFMAX542) u = -UREFMAX542;
        u += UREFMAX542;
        datenwort = (unsigned short) 32768 * u / UREFMAX542; // Konvertierung ist absichtlich so
gewählt
        lobyte = datenwort & 255;
        hibyte = datenwort >> 8;
```

```
	cslf(LO,dac);
	lpt_ausgebenlf(hibyte);
	lpt_ausgebenlf(lobyte);
	cslf(HI,dac);
}

void csthermo(unsigned char wert,int dac)
{
int portbuffer=0;

	if(dac==1) // 30Kanal DAC1
	{
		portbuffer=PINA;
		portbuffer=(portbuffer & 0xFE)+0x01*wert; //pin A0 ; cs setzen
		//_outp(ba+2,portbuffer);
		PORTA=portbuffer;
	}
	else if(dac==2) // 30Kanal DAC2
	{
		portbuffer=PINA;
		portbuffer=(portbuffer & 0xFb)+0x4*(wert); //PIN A2 ; CS setzen
		//_outp(ba+2,portbuffer);
		PORTA=portbuffer;
	}
}

void cslf(unsigned char wert,int dac)
{
int portbuffer=0;

	if(dac==1) // 30Kanal DAC1
	{
		portbuffer=PIND;
		portbuffer=(portbuffer & 0x7F)+0x80*wert; //pin D7 ; cs setzen
		//_outp(ba+2,portbuffer);
		PORTD=portbuffer;
	}
	else if(dac==2) // 30Kanal DAC2
	{
		portbuffer=PIND;
		portbuffer=(portbuffer & 0xFb)+0x4*(wert); //PIN D2 ; cs setzen
		//_outp(ba+2,portbuffer);
		PORTD=portbuffer;
	}
}


void u_ausgabe(double u) // gibt eine Spannung am Ausgang aus
{
	max542thermo(u/((double)R1DAC/R2DAC+1),1); // Verstärkungsfaktor berechnen und an DAC
1 ausgeben
}

void motor(double u) // steuert den Motor 0 bis 100%
{
	u=u/100*MOTORKORREKTUR;
	max542thermo(u/((double)R1DAC/R2DAC+1),2); // Verstärkungsfaktor berechnen und an DAC
1 ausgeben
}

void frequenzausgabe(double f) // stellt die Frequenz der Leitfähigkeitsmessung ein, beide DACs laufen
synchron
```

```
{
        f=-0.218+5.938e-4*f;
        max542lf(f/((double)R1DAC/R2DAC+1),1);
        max542lf(f/((double)R1DAC/R2DAC+1),2);
//      max542thermo(f/((double)R1DAC/R2DAC+1),1);
//      max542thermo(f/((double)R1DAC/R2DAC+1),2);
}
```

11.7.1.5 ads1241e.c

```
#include <avr/pgmspace.h>
#include <math.h>
#include "hardware.h"
#include "small.h"
#include "stdio.h"

extern void USART_Transmit_string(unsigned char *data);

void ads1241write(int *data,int anzahl) // schreibt Anzahl Byte aus dem Feld data in den adc lpt-port ba
{

int bitwert,i,j,byte;

        PORTC=PINC & 247; // pin C3 lo; CS auf lo adc für Datenübertragung fertig
        PORTB=PINB & 247; // pin b3 lo; CS auf lo adc für Datenübertragung fertig
   warten1();

        for(j=0;j<anzahl;j++)
        {
                byte=data[j];
                for(i=7;i>=0;i--) //Datennotieren
                {

                        bitwert=1<<i;

                        if(byte & bitwert) // HI ausgeben
                        {
                                PORTC=(PINC & 251)+4;
                                PORTB=(PINB & 251)+4;
                                warten1();
                        }
                        else // LO ausgeben
                        {
                                PORTC=(PINC & 251);//bit 3 lo, pin 4 lo; din auf lo
                                PORTB=(PINB & 251);//bit 3 lo, pin 4 lo; din auf lo
                                warten1();
                        }

                        PORTC=PINC & 253;//bit 2 lo, pin 3 lo; sclk auf lo // Daten in den Wandler
schreiben
                        PORTB=PINB & 253;//bit 2 lo, pin 3 lo; sclk auf lo // Daten in den Wandler
schreiben

                        warten1();

                        PORTC=(PINC & 253)+2;//bit 2 HI, pin 3 HI; sckl auf hi
                        PORTB=(PINB & 253)+2;//bit 2 HI, pin 3 HI; sckl auf hi
                        warten1();
                }
        }

        PORTC=(PINC & 247)+8;//bit 4 HI, pin 5 HI; CS auf Hi adc Datenübertragung Ende
        PORTB=(PINB & 247)+8;//bit 4 HI, pin 5 HI; CS auf Hi adc Datenübertragung Ende
```

```
	warten1();
}

void ads1241read(int *readdata1thermo,int *readdata2thermo,int *readdata3thermo,int *readdata4thermo,int
*readdata1lf,int *readdata2lf,int *readdata3lf,int *readdata4lf,int *writedata,int readanzahl,int writeanzahl) //
für 30-Kanal
{
int i,j,byte,bitwert=0,bitwert1=0,bitwert2=0,bitwert3=0,bitwert4=0,wert1,wert2,wert3,wert4;
int bitwertlf=0,bitwert1lf=0,bitwert2lf=0,bitwert3lf=0,bitwert4lf=0,wert1lf,wert2lf,wert3lf,wert4lf;

	PORTC=PINC & 247; //PIN C3 lo; CS auf lo adc für Datenübertragung fertig
	PORTB=PINB & 247; //PIN C3 lo; CS auf lo adc für Datenübertragung fertig
	warten1();

	for(j=0;j<writeanzahl;j++)
	{
		byte=writedata[j];
		for(i=7;i>=0;i--) //Daten rausschreiben
		{
			bitwert=1<<i;

			if(byte & bitwert) // HI ausgeben
			{
				PORTC=(PINC & 251)+4;// pin C2 HI; din auf hi
				PORTB=(PINB & 251)+4;// pin C2 HI; din auf hi
				warten1();

			}
			else // LO ausgeben
			{
				PORTC=(PINC & 251); // pin C2 lo; din auf lo
				PORTB=(PINB & 251); // pin C2 lo; din auf lo
				warten1();
			}

			PORTC=PINC & 253; //pin C1 lo; sclk auf lo // Daten in den Wandler
schreiben;
			PORTB=PINB & 253; //pin C1 lo; sclk auf lo // Daten in den Wandler
schreiben;
			warten1();

			PORTC=(PINC & 253)+2; // pin C1 HI; sckl auf hi
			PORTB=(PINB & 253)+2; // pin C1 HI; sckl auf hi
			warten1();
		}
	}

	PORTC=(PINC & 251)+4; //bit 3 HI, pin 4 HI; din auf hi
	PORTB=(PINB & 251)+4; //bit 3 HI, pin 4 HI; din auf hi

	// eventuell Pause einfügen
	warten1();

	for(j=0;j<readanzahl;j++)
	{
		bitwert=128;
		bitwert1=128;
		bitwert2=128;
		bitwert3=128;
		bitwert4=128;
		wert1=0;
		wert2=0;
		wert3=0;
```

```
wert4=0;

bitwertlf=128;
bitwert1lf=128;
bitwert2lf=128;
bitwert3lf=128;
bitwert4lf=128;
wert1lf=0;
wert2lf=0;
wert3lf=0;
wert4lf=0;

for(i=1;i<=8;i++)
{
        PORTC=PINC & 253; // c1 lo; sclk auf lo
        PORTB=PINB & 253; // c1 lo; sclk auf lo
        warten1();

        if((PINA & 0x40) != 0) // PIN A6, ACK, BIT 6
        {
                wert1 += bitwert1;
        }
        if((PINA & 0x20) != 0) // PIN A5, PE, BIT 5
        {
                wert2 += bitwert2;
        }
        if((PINA & 0x10) != 0) // PIN A4, SELEKT, BIT 4
        {
                wert3 += bitwert3;
        }
        if((PINA & 0x8) != 0) // PIN A3, ERROR, BIT 3
        {
                wert4 += bitwert4;
        }

        if((PIND & 0x40) != 0) // PIN D6, ACK, BIT 6
        {
                wert1lf += bitwert1lf;
        }
        if((PIND & 0x20) != 0) // PIN D5, PE, BIT 5
        {
                wert2lf += bitwert2lf;
        }
        if((PIND & 0x10) != 0) // PIN D4, SELEKT, BIT 4
        {
                wert3lf += bitwert3lf;
        }
        if((PIND & 0x8) != 0) // PIN D3, ERROR, BIT 3
        {
                wert4lf += bitwert4lf;
        }


        PORTC=(PINC & 253)+2; // pin c1 HI; sckl auf hi
        PORTB=(PINB & 253)+2; // pin c1 HI; sckl auf hi
        warten1();

        bitwert1 /= 2;
        bitwert2 /= 2;
        bitwert3 /= 2;
        bitwert4 /= 2;
        bitwert1lf /= 2;
        bitwert2lf /= 2;
```

```
                        bitwert3lf /= 2;
                        bitwert4lf /= 2;
                }

                readdata1thermo[j]=wert1;
                readdata2thermo[j]=wert2;
                readdata3thermo[j]=wert3;
                readdata4thermo[j]=wert4;

                readdata1lf[j]=wert1lf;
                readdata2lf[j]=wert2lf;
                readdata3lf[j]=wert3lf;
                readdata4lf[j]=wert4lf;

        }

        PORTC=(PINC & 247)+8; //bit 4 HI, pin 5 HI; CS auf Hi adc Datenübertragung Ende
        PORTB=(PINB & 247)+8; //bit 4 HI, pin 5 HI; CS auf Hi adc Datenübertragung Ende

        warten1();
}


void ads1241reset(void) //setzt den Wandler auf Ausgangswerte zurück, Ende continuos mode
{
int data=0xfe;//1111 1110

        ads1241write(&data,1);
}

void ads1241dsync(void) //startet Modulator des Wandlers
{
int data=0xfc;//1111 1100

        ads1241write( &data,1);
}

void ads1241selfoffsetgaincal(void) //self offset und gain calibration
{
int data=0xf0;//1111 0000

        ads1241write( &data,1);
}

// setzt positiven Kanal auf ain 1 bis 8 und negativen Kanal auf aincom, gleich agnd
void ads1241kanalselekt(int kanal) // Kanal ist 0 bis 7
{
int data[3],datenraus=0,indata=0,indata1=0,indata2=0,indata3=0,indata4=0;//0000 1000
int indata1lf=0,indata2lf=0,indata3lf=0,indata4lf=0;
int dummy;

        data[0]=0x51; // schreibt in Register 1
        data[1]=0x01; // schreibt ein Byte 1 auf Null gesetzt
        data[2]=kanal*0x10+8;// die oberen vier Bit geben den positiven Bereich an,  negativer Kanal auf
aincom
        dummy=data[2];
        ads1241write(data,3);

        do
        {
                data[0]=0x51; // schreibt in Register 1
                data[1]=0x01; // schreibt ein Byte 1 auf null gesetzt
                datenraus=kanal*0x10+8; // setzt mux bits
```

```
		data[2]=datenraus;
		ads1241write(data,3);

		data[0]=0x11; //liest aus Register 1
		data[1]=0x00; //liest nur ein Register

	ads1241read(&indata1,&indata2,&indata3,&indata4,&indata1lf,&indata2lf,&indata3lf,&indata4lf,dat
a,1,2);

		/*if(indata1!=datenraus)
			indata=indata1;
		else if(indata2!=datenraus)
			indata=indata2;
		else if(indata3!=datenraus)
			indata=indata3;
		else if(indata4!=datenraus)
			indata=indata4;*/
		//else
			indata=datenraus;

	}
	while(indata!=datenraus);
	dummy=0; // debug
}

// setzt pga Verstärker
void ads1241pga(int pga) //Adresse und Verstärkungsfaktor
{
int writedata[3],indata=0,indata1=0,indata2=0,indata3=0,indata4=0,datenraus=0;
int indata1lf=0,indata2lf=0,indata3lf=0,indata4lf=0;
unsigned int pgabit;
	pgabit=(unsigned int) round(log((double)pga)/log(2));
	writedata[0]=0x10; //liest aus Register 0
	writedata[1]=0x00; //liest nur ein Register

	ads1241read(&indata1,&indata2,&indata3,&indata4,&indata1lf,&indata2lf,&indata3lf,&indata4lf,writ
edata,1,2);
	indata=indata1;// alle ADCs haben den gleichen Verstärkungsfaktor, daher reicht einer, für 2Kanal
nur ADC2

	writedata[0]=0x50; // schreibt in Register 0
	writedata[1]=0x01; // schreibt ein Byte von 1 auf Null gesetzt
	datenraus=(indata & 0xf8)+pgabit; // setzt pga bits
	writedata[2]=datenraus;
	ads1241write(writedata,3);

	writedata[0]=0x10; //liest aus Register 0
	writedata[1]=0x00; //liest nur ein Register

	do
	{
		writedata[0]=0x50; // schreibt in Register 0
		writedata[1]=0x01; // schreibt ein Byte
		datenraus=(indata & 0xf8)+pgabit; // setzt pga bits
		writedata[2]=datenraus;
		ads1241write(writedata,3);

		writedata[0]=0x10; //liest aus Register 0
		writedata[1]=0x01; //liest nur ein Register

	ads1241read(&indata1,&indata2,&indata3,&indata4,&indata1lf,&indata2lf,&indata3lf,&indata4lf,writ
edata,1,2);
```

```
			/*if(indata1!=datenraus)
				indata=indata1;
			else if(indata2!=datenraus)
				indata=indata2;
			else if(indata3!=datenraus)
				indata=indata3;
			else if(indata4!=datenraus)
				indata=indata4;
			else
				indata=datenraus;*/
			indata=datenraus;

		}
		while(indata!=datenraus);
}

void ads1241einstellen(void) //drdy0, datenformat unipolar1, speed 1/128 0,buffer aus 0,bitoder 0, range full 0, datarate 15Hz 00,
{
int writedata[3]={0x52,0x00,0xC0};

		ads1241write(writedata,3);

}

void ads1241klarmachen(void) //macht ads1241 für Messung  bereit
{

		pause(100); //100ms warten
		PORTC=(PINC & 254)+1; // pin C0 HI; dsync auf hi
		PORTB=(PINB & 254)+1; // pin B0 HI; dsync auf hi

		ads1241reset();
		pause(100);
		ads1241einstellen();
		pause(100);
}


void ads1241getu30(int pga, int kanal, double *u1thermo, double *u2thermo, double *u3thermo, double *u4thermo,double *u1lf, double *u2lf, double *u3lf, double *u4lf) //30 Kanal liest einmal Spannung vom Wandler an ba, mit Verstärker pga(1-128), und Kanal  ein(1-8),
{
unsigned int readdata1thermo[16],readdata2thermo[16],readdata3thermo[16],readdata4thermo[16],writedata[2]={16,15};
unsigned int readdata1lf[16],readdata2lf[16],readdata3lf[16],readdata4lf[16];

		PORTC=(PINC & 254)+1; //C0 HI; dsync auf hi
		PORTB=(PINB & 254)+1; //B0 HI; dsync auf hi

		ads1241pga(pga);

		pause (10); //40ms warten, 10 sind auch ausreicend

		ads1241kanalselekt(kanal-1);

		pause (10); //60ms warten, 10  sind auch ausreichend

		ads1241selfoffsetgaincal();

		pause(40);//80ms warten , 40  sind auch ausreichend

		PORTC=(PINC & 254); //bit 1 lo, pin 2 lo; dsync auf lo
```

```
        PORTB=(PINB & 254); //bit 1 lo, pin 2 lo; dsync auf lo


        pause(80); //80ms warten

        PORTC=(PINC & 254)+1; //bit 1 hi, pin 2 hi; dsync auf hi
        PORTB=(PINB & 254)+1; //bit 1 hi, pin 2 hi; dsync auf hi

        pause(400);//300ms warten, dann keine Aufspaltung mehr

        ads1241read(readdata1thermo,readdata2thermo,readdata3thermo,readdata4thermo,readdata1lf,readd
ata2lf,readdata3lf,readdata4lf,writedata,16,2);
        *u1thermo=(readdata1thermo[0x0f]+readdata1thermo[0x0e]*256+readdata1thermo[0x0d]*65536)*
UREF/16777215/pga;
        *u2thermo=(readdata2thermo[0x0f]+readdata2thermo[0x0e]*256+readdata2thermo[0x0d]*65536)*
UREF/16777215/pga;
        *u3thermo=(readdata3thermo[0x0f]+readdata3thermo[0x0e]*256+readdata3thermo[0x0d]*65536)*
UREF/16777215/pga;
        *u4thermo=(readdata4thermo[0x0f]+readdata4thermo[0x0e]*256+readdata4thermo[0x0d]*65536)*
UREF/16777215/pga;

        *u1lf=(readdata1lf[0x0f]+readdata1lf[0x0e]*256+readdata1lf[0x0d]*65536)*UREF/16777215/pga;
        *u2lf=(readdata2lf[0x0f]+readdata2lf[0x0e]*256+readdata2lf[0x0d]*65536)*UREF/16777215/pga;
        *u3lf=(readdata3lf[0x0f]+readdata3lf[0x0e]*256+readdata3lf[0x0d]*65536)*UREF/16777215/pga;
        *u4lf=(readdata4lf[0x0f]+readdata4lf[0x0e]*256+readdata4lf[0x0d]*65536)*UREF/16777215/pga;

        return;
}
```

11.7.1.6 hardware.h

```
// Definitionen, die direkt die Hardware betreffen

//Geräte Parameter
#define UREF 2.5 // Spannungsreferenz der Temperaturmessung
#define UREFMAX542 2.5 // Spannungsreferenz des DAC
#define R1DAC 10000  // Spannungsteiler Verstäker DAC
#define R2DAC 5600 // Spannungsteiler Verstäker DAC
#define THERMA 1.068981e-3 //Standardkalibrierdaten für Thermistoren
#define THERMB 2.120700e-4
#define THERMC 9.019537e-8
#define R1 100e3

#define MOTORKORREKTUR 6.2 // Maximaler Ausgangspegel an DAC

#define MAXTEMPERATUR 40 // maximale Messtemperatur
#define MINTEMPERATUR -60 // minimale Messtemperatur

// logigpegel
#define HI 1
#define LO 0
```

11.7.1.7 ds1305.h

```
void ds1305bytewrite(int adresse, int data); //schreibt ein Byte an den ds1305
void ds1305byteread(int adresse, int *data); //liest ein Byte vom ds1305
void startclock(void); //startet den Oszillator auf dem ds1305
void clockoff(void); //schaltet den Oszillator auf dem ds1305 aus
double gettime(void);
//int gettime(void);
```

11.7.1.8 small.h

```
void stoerungthermo(int cond); // setzt Störungsr elais auf aus und Störungs- LED an
void ledmessenthermo(int led);
void wandelfeldruehrer(int cond); // schaltet Wandelfeldrührer an und aus
void ledmessenlf(int led);
double round(double x);
void warten1(void);
void pause (int ms);  // in ms bei 8 Mhz
void USART_Init( unsigned int baud );
void USART_Transmit( unsigned char data );
void USART_Transmit_string(unsigned char *data);
void warten1(void);
unsigned char USART_Receive(void);
```

11.7.1.9 max542.h

```
void lpt_ausgebenthermo(unsigned char byte);
void max542thermo(double u,int dac); // gibt eine Spannung aus
void csthermo(unsigned char wert,int dac);
void lpt_ausgebenlf(unsigned char byte);
void max542lf(double u,int dac); // gibt eine Spannung aus
void cslf(unsigned char wert,int dac);
void u_ausgabe(double u); // gibt eine Spannung am Ausgang aus
void motor(double u); // steuert den Motor 0 bis 100%
void frequenzausgabe(double f); // stellt die Frequenz der Leitfähigkeitsmessung ein, beide DACs laufen synchron
```

11.7.1.10 ads1241e.h

```
void ads1241pga(int pga); //setzt pgaverstärker
void ads1241kanalselekt(int kanal);
void ads1241selfoffsetgaincal(void); //self offset und gain calibration
void ads1241reset(void); //setzt den Wandler auf Ausgangswerte zurück, Ende continuos mode
void ads1241dsync(void); //startet Modulator des Wandlers
void ads1241write(int *data,int anzahl); // schreibt Anzahl Byte aus dem Feld data in den adc lpt-port ba
void ads1241einstellen(void);
void ads1241klarmachen(void); // richtet adc für Messung her
void ads1241getu30(int pga, int kanal, double *u1thermo, double *u2thermo, double *u3thermo, double *u4thermo,double *u1lf, double *u2lf, double *u3lf, double *u4lf); //30 Kanal liest einmal Spannung vom Wandler an ba, mit Verstärker pga(1-128), und Kanal  ein(1-8),
void ads1241read(int *readdata1thermo,int *readdata2thermo,int *readdata3thermo,int *readdata4thermo,int *readdata1lf,int *readdata2lf,int *readdata3lf,int *readdata4lf,int *writedata,int readanzahl,int writeanzahl);
```

11.7.2 Phasendiagrammmessgerät, PC-Software

11.7.2.1 2kanal.cpp

```
// Funktionen für 2 Kanal U/t schreiber mit automatischer Verstärkereinstellung

#include "StdAfx.h"

extern HINSTANCE hInst;
extern MESSSTEUERSTRUCT defmessparam;

extern DATENSATZ * messdaten;
extern HGLOBAL hMemHandle1;
extern REIHENSTEUERSTRUCT messkontrolle;

extern double bereichsumschalter[8]; // Spannungen für die Kanalumschaltung
```

```
void kanal2Utschreiber(HWND hWnd) //2 Kanal T/t schreiber als Test für 30 Kanal
{
int maxwinx,maxwiny,startx,starty,j=0,readcount=0;
long int i=0;
double ywertmax,ywertmin,u=0,messdauer=0,dummy1, dummy2, dummy3, dummy4;
HDC hDc;
RECT Rect;
char outtext[1000];
HPEN hStift[8];
static char  szFilename[810] = ".kal";
HANDLE hFile32;

	if(DialogBox(hInst, (LPCTSTR)IDD_8KanalTt, hWnd, (DLGPROC) StartDialogkanal8Tt)==FALSE)
		return; // bei Abbruch zurück

	hFile32=CreateFile(defmessparam.tempfile,GENERIC_WRITE,FILE_SHARE_READ,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
	CloseHandle(hFile32); // altes Tempfile löschen

	messkontrolle.messtyp=KANAL2UT; //Einstellungen für diesen Messtyp
	messkontrolle.messan=TRUE; // eine Messung läuft
	messkontrolle.maximalerywert=UREF;
	messkontrolle.minmalerywert=0;  // 0V da nicht bipolar

	drawkosy(hWnd, NOPAINTMSG);

	hStift[0]=CreatePen(PS_SOLID,0,RGB(0,0,0)); // Einzelne Stifte herrichten
	hStift[1]=CreatePen(PS_SOLID,0,RGB(128,0,0));

	ywertmax=messkontrolle.maximalerywert;
	ywertmin=messkontrolle.minmalerywert;

	hDc=GetDC(hWnd);// Fenster Parameter
	GetClientRect(hWnd, &Rect);
	maxwinx=Rect.right;
	maxwiny=Rect.bottom;
	starty=(int) (maxwiny-UAB-OAB)+OAB;
	startx=(int) LAB;

	messkontrolle.flagkanal1=TRUE; // Messkanal setzen
	messkontrolle.flagkanal2=TRUE;
	messkontrolle.flagkanal3=FALSE;
	messkontrolle.flagkanal4=FALSE;
	messkontrolle.flagkanal5=FALSE;
	messkontrolle.flagkanal6=FALSE;
	messkontrolle.flagkanal7=FALSE;
	messkontrolle.flagkanal8=FALSE;

	GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden

	ads1241klarmachen(defmessparam.ba);
	startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts
	ads1241getu30(defmessparam.ba,defmessparam.verstearker,defmessparam.kanal,&dummy1,&dummy2,&dummy3,&dummy4);

	messdauer=gettime(defmessparam.ba); // ersten Wert speichern

	do // eigentliche Messschleife
	{
		do
		{
```

```
                }
                while(messdauer==gettime(defmessparam.ba)); //auf nächsten tick warten;
                messdauer=gettime(defmessparam.ba);

                for(j=1;j<=2;j++)
                {
                        ads1241getu30(defmessparam.ba,1,j,&dummy1,&dummy2,&dummy3,&u);      //
Spannung einlesen, um Bereich festzustellen
                        /*if(u<=bereichsumschalter[6])
                                pga=128;
                        else if(u<=bereichsumschalter[5])
                                        pga=64;
                        else if(u<=bereichsumschalter[4])
                                        pga=32;
                        else if(u<=bereichsumschalter[3])
                                        pga=16;
                        else if(u<=bereichsumschalter[2])
                                        pga=8;
                        else if(u<=bereichsumschalter[1])
                                        pga=4;
                        else if(u<=bereichsumschalter[0])
                                        pga=2;
                        else
                                pga=1;
*/
                //
        ads1241getu30(defmessparam.ba,defmessparam.verstearker,j,&u,&dummy2,&dummy3,&dummy4);
// Spannung einlesen

                        messdaten[i].t=messdauer-1;
                        messdaten[i].ywerte[j-1]=u;

                        SelectObject(hDc,hStift[j]);
                        if(i>0)
                        {
                                MoveToEx(hDc,(int)(startx+(messdaten[i-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int)      round(      (starty-(messdaten[i-
1].ywerte[j-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);
                                LineTo(hDc,(int)(startx+(messdaten[i].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round ((starty-(messdaten[i].ywerte[j-
1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))));
                        }

                }
                sprintf(outtext,"%.0lf s,%i: %.3lf V, %i: %.3lf V", messdauer-1,
1,messdaten[i].ywerte[0],2,messdaten[i].ywerte[1]);
            TextOut(hDc,LAB+10,(OAB-TEXTYDY)/2,outtext,strlen(outtext));
                saveuntermessung(hWnd,i); // Daten unter der Messung speichern

                i++;
                if(GetAsyncKeyState('E') != 0)
                {
                        messkontrolle.messdauer=messdauer;
                        break; // Messung abbrechen
                }
        }
        while(messdauer<=messkontrolle.messdauer);

        messkontrolle.anzahl=i;
        clockoff(defmessparam.ba);
        GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden,
um das Speichern nicht zu stören
```

```
        save2Ut(hWnd); // Daten speichern

        for(j=0;j<2;j++) // Stifte wieder aufräumen
                DeleteObject(hStift[j]);
        ReleaseDC(hWnd,hDc);
        messkontrolle.messan=FALSE; // eine Messung läuft

}

int save2Ut(HWND hWnd) // speichert Messdaten
{
static char  szFilename[810] = ".txt";
int iErg=0;
char *feld,zeile[120];
long int i;
HGLOBAL hMemHandleDaten;
HFILE hFile;
unsigned long lBytes;
OPENFILENAME ofn;

        hMemHandleDaten=GlobalAlloc(GHND,sizeof(char)*120*MAXMESSDAUER);  //Größe  des
charfeldes
        feld=(char*)GlobalLock(hMemHandleDaten);

        memset(&ofn, 0, sizeof(OPENFILENAME));
        ofn.lpstrFile = szFilename;
        iErg = FileDialogSave1ut(hWnd, &ofn);

        if(iErg == IDOK)
        {
          if((hFile = _lcreat(ofn.lpstrFile,0)) == HFILE_ERROR)
          {
            MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK);
            SendMessage(hWnd,WM_CLOSE,NULL,NULL);
            GlobalUnlock(hMemHandleDaten);
            GlobalFree(hMemHandleDaten);
            return(FEHLER);
          }

          // an den Dateianfang gehen
          _llseek(hFile, 0L, 0);

          // Datei schreiben

                                for(i=0;i<messkontrolle.anzahl;i++)
          {
                                        sprintf(zeile,"%.0lf                                %.8lf
%.8lf\r\n",messdaten[i].t,messdaten[i].ywerte[0],messdaten[i].ywerte[1]);
                                        lBytes = _hwrite(hFile,zeile,strlen(zeile));
                                        if(lBytes!=strlen(zeile))
                                        {
                                                MessageBox(hWnd, "Da ist was danebengegangen",
"Achtung", MB_OK|MB_ICONSTOP);
                                                _lclose(hFile);
                                        GlobalUnlock(hMemHandleDaten);
                                GlobalFree(hMemHandleDaten);
                        return(FEHLER);
                                        }
                                        else
                                        {
                                                //_llseek(hFile, lBytes, 1);
                                        }
          }
```

```
            _lclose(hFile);
                        }

        GlobalUnlock(hMemHandleDaten);
        GlobalFree(hMemHandleDaten);
        return(OK);
}
```

11.7.2.2 30kanal.cpp

```
// 30kanal.cpp : Definiert den Einsprungpunkt für die Anwendung.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Globale Variablen:
HINSTANCE hInst;                                    // aktuelle Instanz
TCHAR szTitle[MAX_LOADSTRING];              // Text der Titelzeile
TCHAR szWindowClass[MAX_LOADSTRING];// Text der Titelzeile
HWND mainhwnd;

extern TCHAR szWindowClassMess1[MAX_LOADSTRING];
extern TCHAR szWindowClassMess2[MAX_LOADSTRING];
extern TCHAR szWindowClassMess3[MAX_LOADSTRING];
extern TCHAR szWindowClassMess4[MAX_LOADSTRING];
extern TCHAR szWindowClassMessStatus[MAX_LOADSTRING];
extern TCHAR szWindowClassMessHiRes1[MAX_LOADSTRING];
extern TCHAR szWindowClassHiRes2[MAX_LOADSTRING];
extern TCHAR szWindowClassMessHiRes3[MAX_LOADSTRING];
extern TCHAR szWindowClassMessHiRes4[MAX_LOADSTRING];
extern TCHAR szWindowClassASL[MAX_LOADSTRING];
extern TCHAR szWindowClassMessHiResASL[MAX_LOADSTRING];

extern TCHAR szWindowClassMess1lf[MAX_LOADSTRING];
extern TCHAR szWindowClassMess2lf[MAX_LOADSTRING];
extern TCHAR szWindowClassMess3lf[MAX_LOADSTRING];
extern TCHAR szWindowClassMess4lf[MAX_LOADSTRING];
extern TCHAR szWindowClassMessStatuslf[MAX_LOADSTRING];
extern TCHAR szWindowClassMessHiRes1lf[MAX_LOADSTRING];
extern TCHAR szWindowClassHiRes2lf[MAX_LOADSTRING];
extern TCHAR szWindowClassMessHiRes3lf[MAX_LOADSTRING];
extern TCHAR szWindowClassMessHiRes4lf[MAX_LOADSTRING];


extern REIHENSTEUERSTRUCT messkontrolle;

// Vorausdeklarationen von Funktionen, die in diesem Code-Modul enthalten sind:
ATOM                    MyRegisterClass( HINSTANCE hInstance );
BOOL                    InitInstance( HINSTANCE, int );
LRESULT CALLBACK    WndProc( HWND, UINT, WPARAM, LPARAM );
LRESULT CALLBACK    About( HWND, UINT, WPARAM, LPARAM );

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow)
{
LARGE_INTEGER startzeit;
double TimeUnit;

        TimeInit(&TimeUnit);
```

```
	// ZU ERLEDIGEN: Fügen Sie hier den Code ein.
	MSG msg;
	HACCEL hAccelTable;

	// Globale Zeichenfolgen initialisieren
	LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MY8KANAL, szWindowClass, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESS1, szWindowClassMess1, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESS2, szWindowClassMess2, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESS3, szWindowClassMess3, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESS4, szWindowClassMess4, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRES1, szWindowClassMessHiRes1, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRES2, szWindowClassHiRes2, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRES3, szWindowClassMessHiRes3, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRES4, szWindowClassMessHiRes4, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSASL, szWindowClassASL, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRESASL,szWindowClassMessHiResASL,
MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSSTATUS, szWindowClassMessStatus, MAX_LOADSTRING);

	LoadString(hInstance, IDC_MESS1LF, szWindowClassMess1lf, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESS2LF, szWindowClassMess2lf, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESS3LF, szWindowClassMess3lf, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESS4LF, szWindowClassMess4lf, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRES1LF, szWindowClassMessHiRes1lf,
MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRES2LF, szWindowClassHiRes2lf, MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRES3LF, szWindowClassMessHiRes3lf,
MAX_LOADSTRING);
	LoadString(hInstance, IDC_MESSHIRES4LF, szWindowClassMessHiRes4lf,
MAX_LOADSTRING);

	MyRegisterClass(hInstance);// Fensterklassen registrieren
	MyRegisterClassMess1(hInstance);
	MyRegisterClassMess2(hInstance);
	MyRegisterClassMess3(hInstance);
	MyRegisterClassMess4(hInstance);
	MyRegisterClassMessStatus(hInstance);
	MyRegisterClassMessHiRes1(hInstance);
	MyRegisterClassHiRes2(hInstance);
	MyRegisterClassMessHiRes3(hInstance);
	MyRegisterClassMessHiRes4(hInstance);
	MyRegisterClassMessASL(hInstance);
	MyRegisterClassMessASLHiRes(hInstance);

	MyRegisterClassMess1lf(hInstance);
	MyRegisterClassMess2lf(hInstance);
	MyRegisterClassMess3lf(hInstance);
	MyRegisterClassMess4lf(hInstance);
	MyRegisterClassMessHiRes1lf(hInstance);
	MyRegisterClassHiRes2lf(hInstance);
	MyRegisterClassMessHiRes3lf(hInstance);
	MyRegisterClassMessHiRes4lf(hInstance);

	// Initialisierung der Anwendung durchführen:
	if( !InitInstance( hInstance, nCmdShow ) )
	{
		return FALSE;
	}

	hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_MY8KANAL);

	QueryPerformanceCounter(&startzeit); // Startzeit setzen
```

```
/*	// Hauptnachrichtenschleife:
	do
	{
		PeekMessage(&msg, NULL, 0, 0,PM_REMOVE); // liest Message ein
		if( !TranslateAccelerator (msg.hwnd, hAccelTable, &msg) )
		{
			QueryPerformanceCounter(&zeit);
			if(timerstatus==TRUE)
			{
				if(((zeit.QuadPart-startzeit.QuadPart)*TimeUnit)>WARTEZEIT)
				{
					SendMessage(mainhwnd,WM_TIMERHGS,0,0);	//	Timer
erreicht
					startzeit=zeit;
				}
			}

			TranslateMessage( &msg );
			DispatchMessage( &msg );
		}
	}
	while(1);
*/
/*	while( PeekMessage(&msg, NULL, 0, 0,PM_REMOVE) ) //GetMessage(&msg, NULL, 0, 0)
	{
		if( !TranslateAccelerator (msg.hwnd, hAccelTable, &msg) )
		{
		/*	QueryPerformanceCounter(&zeit);
			if(timerstatus==TRUE)
			{
				if(((zeit.QuadPart-startzeit.QuadPart)*TimeUnit)>WARTEZEIT)
				{
					//SendMessage(mainhwnd,WM_TIMERAN,0,0); // Timer an
					startzeit=zeit;
				}
			}

			TranslateMessage( &msg );
			DispatchMessage( &msg );
		}
	}
*/
	// Hauptnachrichtenschleife:
	while( GetMessage(&msg, NULL, 0, 0) )
	{
		if( !TranslateAccelerator (msg.hwnd, hAccelTable, &msg) )
		{
			TranslateMessage( &msg );
			DispatchMessage( &msg );
		}
	}


	return msg.wParam;
}



//
//  FUNKTION: MyRegisterClass()
//
//  AUFGABE: Registriert die Fensterklasse.
```

```
//
//  KOMMENTARE:
//
//    Diese Funktion und ihre Verwendung sind nur notwendig, wenn dieser Code
//    mit Win32-Systemen vor der 'RegisterClassEx'-Funktion kompatibel sein soll,
//    die zu Windows 95 hinzugefügt wurde. Es ist wichtig diese Funktion aufzurufen,
//    damit der Anwendung kleine Symbole mit den richtigen Proportionen zugewiesen
//    werden.
//
ATOM MyRegisterClass( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_MY8KANAL;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}

//
//   FUNKTION: InitInstance(HANDLE, int)
//
//   AUFGABE: Speichert die Instanzzugriffsnummer und erstellt das Hauptfenster
//
//   KOMMENTARE:
//
//        In dieser Funktion wird die Instanzzugriffsnummer in einer globalen Variable
//        gespeichert und das Hauptprogrammfenster erstellt und angezeigt.
//
BOOL InitInstance( HINSTANCE hInstance, int nCmdShow )
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }

  ShowWindow( hWnd, nCmdShow );
  UpdateWindow( hWnd );
  mainhwnd=hWnd;
  return TRUE;
}

//
//  FUNKTION: WndProc(HWND, unsigned, WORD, LONG)
//
```

```
//  AUFGABE:  Verarbeitet Nachrichten für das Hauptfenster.
//
//  WM_COMMAND      - Anwendungsmenü verarbeiten
//  WM_PAINT  - Hauptfenster darstellen
//  WM_DESTROY        - Beendigungsnachricht ausgeben und zurückkehren
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
int wmId, wmEvent;
//        PAINTSTRUCT ps;
//        HDC hdc;
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
DWORD commask=0;

        switch( message )
        {

                case WM_CREATE :
                        globalinit(hWnd);
                        //SetTimer(hWnd,messkontrolle.timer,WARTEZEIT,NULL); // Timer an
                break;

                case WM_COMMAND:
                        wmId=LOWORD(wParam);
                        wmEvent=HIWORD(wParam);
                        // Menüauswahlen analysieren:
                        MenuJob(hWnd,wParam,lParam);

                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT: // Fenster neu zeichnen
                        drawkosy(hWnd, PAINTMSG);
                break;

                case WM_TIMERHGS:
                        if((messkontrolle.messtyp==KANAL30TU)                                   ||
(messkontrolle.messtyp==KANAL30TT)) // Thermometer alleine
                                kanalmess30(hWnd);

                        else                                                                    if
((messkontrolle.messtyp==KANAL30TTLF)||(messkontrolle.messtyp==KANAL30TTLF))    //    beide
Geräte zusammen
                        {
                                kanalmess30lf(hWnd);
                        }
                break;

                case WM_LBUTTONDOWN:
                        if((messkontrolle.messtyp==KAPPATAUSWERT                                ||
messkontrolle.messtyp==KAPPATTAUSWERT || messkontrolle.messtyp==TTAUSWERT) && (wParam
== MK_LBUTTON))
                        {
                                bearbeitemausebutton(lParam,hWnd);
                        }
                break;

                case WM_RBUTTONDOWN:
                        if(             (messkontrolle.messtyp==KAPPATAUSWERT           ||
messkontrolle.messtyp==KAPPATTAUSWERT || messkontrolle.messtyp==TTAUSWERT) && (wParam
== MK_RBUTTON))
```

```
                        {
                                bearbeitemauserechts(lParam,hWnd);
                        }
                break;

                case WM_MBUTTONDOWN:
                        if(             (messkontrolle.messtyp==KAPPATAUSWERT           ||
messkontrolle.messtyp==KAPPATTAUSWERT || messkontrolle.messtyp==TTAUSWERT) && (wParam
== MK_MBUTTON))
                        {
                                bearbeitemausemitte(lParam,hWnd);
                        }

                        if(             (messkontrolle.messtyp==KAPPATAUSWERT           ||
messkontrolle.messtyp==KAPPATTAUSWERT || messkontrolle.messtyp==TTAUSWERT) && (wParam
== (MK_MBUTTON|MK_SHIFT)))
                        {
                                bearbeitemausemitteshift(lParam,hWnd);
                        }

                break;



                case WM_DESTROY:
                        globalexit();
                        PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

/*-- Funktion für die Abarbeitung des Menus --*/

LONG MenuJob(HWND hWnd, WPARAM wParam, LPARAM lParam)
{

        switch(wParam)
        {

                case ID_MESSUNG_ABKHLKURVE:
                        kanal8Ttschreiber(hWnd);
                break;

                case ID_MESSUNG_ASL:
                        aslkanalschreiber(hWnd);
                break;

                case IDM_TESTFUNKTIONEN_1KANALUTSCHREIBER:
                        kanal1utschreiber(hWnd);
                break;

                case IDM_TESTFUNKTIONEN_8KANALUTSCHREIBER:
                        kanal8utschreiber(hWnd);
                break;

                case IDM_TESTFUNKTIONEN_2KANALUT:
                        kanal2Utschreiber(hWnd);
                break;
```

```
                case IDM_TESTFUNKTIONEN32KANALSPANNUNG:
                  DialogBox(hInst, (LPCTSTR)IDD_32KanalUDisplay, hWnd,
(DLGPROC)Kanal32spannungsanzeige);
                break;

                case ID_MESSUNGEN30KANAL_TEMPERATURANZEIGE30KANAL:
                  DialogBox(hInst, (LPCTSTR)IDD_32KanalUDisplay, hWnd,
(DLGPROC)Kanal32temperaturanzeige);
                break;

                case IDM_TESTFUNKTIONEN_SPANNUNGSAUSGABE:
                  DialogBox(hInst, (LPCTSTR)IDD_SPANNUNGSAUSGABE, hWnd,
(DLGPROC)spannungsausgabe);
                break;

                case IDM_EXTRAS_OPTIONEN:
                  DialogBox(hInst, (LPCTSTR)IDD_OPTIONEN, hWnd, (DLGPROC)optionen);
                break;

                case IDM_EXTRAS_OPTIONENSPEICHERN:
                        writeini();
                break;

                case ID_MESSUNGEN30KANAL_KALIBRIERUNG:
                        messkontrolle.messtyp=KANAL30TU; //Einstellungen für diesen Messtyp
                        kanal30TUschreiber(hWnd);
                break;

                case ID_MESSUNGEN30KANAL_TEMPERATURMESSUNG30KANAL:
                        messkontrolle.messtyp=KANAL30TT; //Einstellungen für diesen Messtyp
                        kanal30TUschreiber(hWnd);
                break;

                case
ID_MESSUNGEN30KANAL_TEMPERATURUNDLEITFHIGKEITSMESSUNG30KANAL:
                        messkontrolle.messtyp=KANAL30TTLF; //Einstellungen für diesen Messtyp
                        kanal30TundLF(hWnd);
                break;

                case ID_MESSUNGEN30KANAL_KALIBRIERUNGCOMM:
                        messkontrolle.messtyp=KANAL30TTLFKAL; //Einstellungen für diesen
Messtyp
                        kanal30TundLF(hWnd);
                break;

                case ID_TESTFUNKTIONEN_WTWGT:
                        messkontrolle.messtyp=WTWGT; //Einstellungen für diesen Messtyp
                        wtwkanalschreiber(hWnd);
                break;

                case ID_KALIB_DATENSATZ:
                        kalibdatensatz(hWnd);
                break;

                case IDM_ABOUT:
                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
                break;

                case IDM_EXIT:
                  PostQuitMessage(0);
                break;

                case ID_EXTRAS_THERMOSTAT:
```

```
                    thermomenue(GetMenu(hWnd));
                break;

                case ID_EXTRAS_MOTOR:
                    motormenue(GetMenu(hWnd));
                break;

                case ID_EXTRAS_THERMOSTATCOMM:
                    thermomenuecomm(GetMenu(hWnd));
                break;

                case ID_EXTRAS_MOTORCOMM:
                    motormenuecomm(GetMenu(hWnd));
                break;

                case ID_MESSUNGEN30KANAL_LEITFAEHIGKEITSMESSUNG30KANAL:
                    messkontrolle.messtyp=KANAL30TTLF;
                    kanal30TundLF(hWnd);
                break;

                case ID_AUSWERTUNG_KAPPT:
                    messkontrolle.messtyp=KAPPATAUSWERT;
                    kappatauswert(hWnd);
                break;

                case ID_AUSWERTUNG_KAPPTT:
                    messkontrolle.messtyp=KAPPATTAUSWERT;
                    kappatauswert(hWnd);
                break;

                case ID_AUSWERTUNG_TT:
                    messkontrolle.messtyp=TTAUSWERT;
                    kappatauswert(hWnd);
                break;

                default:
                  return 0;
        }
        return 0;
}
```

11.7.2.3 ads1241e.cpp

```
#include "StdAfx.h"

extern REIHENSTEUERSTRUCT messkontrolle;

void ads1241write(USHORT ba, int *data,int anzahl) // schreibt Anzahl Byte aus dem Feld data in den adc
lpt-port ba
{
int bitwert,i,j,byte;
UCHAR portbuffer=0;


        portbuffer=_inp(ba);
        portbuffer=portbuffer & 247; //bit 4 lo, pin 5 lo; CS auf lo adc für Datenübertragung fertig
        _outp(ba,portbuffer);
   warten1();

        for(j=0;j<anzahl;j++)
        {
                byte=data[j];
                for(i=7;i>=0;i--) //Datenrausschreiben
```

```
		{

			bitwert=1<<i;

			if(byte & bitwert) // HI Ausgeben
			{
				portbuffer=(portbuffer & 251)+4; //bit 3 HI, pin 4 HI; din auf hi
				_outp(ba,portbuffer);
				warten1();
			}
			else // LO ausgeben
			{
				portbuffer=(portbuffer & 251); //bit 3 lo, pin 4 lo; din auf lo
				_outp(ba,portbuffer);
				warten1();
			}

			portbuffer=portbuffer & 253; //bit 2 lo, pin 3 lo; sclk auf lo // Daten in den Wandler schreiben
			_outp(ba,portbuffer);
			warten1();

			portbuffer=(portbuffer & 253)+2; //bit 2 HI, pin 3 HI; sckl auf hi
			_outp(ba,portbuffer);
			warten1();
		}
	}

	portbuffer=(portbuffer & 247)+8; //bit 4 HI, pin 5 HI; CS auf Hi adc Datenübertragung Ende
	_outp(ba,portbuffer);
	warten1();
}


void ads1241read(USHORT ba, int *readdata,int *writedata,int readanzahl,int writeanzahl) // für 8 Kanal
{
	int i,j,byte,bitwert,wert;
	UCHAR portbuffer=255;

	portbuffer=_inp(ba);

	portbuffer=portbuffer & 247; //bit 4 lo, pin 5 lo; CS auf lo adc für Datenübertragung beendet
	_outp(ba,portbuffer);
	warten1();

	for(j=0;j<writeanzahl;j++)
	{
		byte=writedata[j];
		for(i=7;i>=0;i--) //Datennotieren
		{

			bitwert=1<<i;

			if(byte & bitwert) // HI ausgeben
			{
				portbuffer=(portbuffer & 251)+4; //bit 3 HI, pin 4 HI; din auf hi
				_outp(ba,portbuffer);
				warten1();
			}
			else // LO ausgeben
			{
				portbuffer=(portbuffer & 251); //bit 3 lo, pin 4 lo; din auf lo
```

```
                                _outp(ba,portbuffer);
                                warten1();
                        }

                        portbuffer=portbuffer & 253; //bit 2 lo, pin 3 lo; sclk auf lo // Daten in den
Wandler schreiben
                        _outp(ba,portbuffer);
                        warten1();

                        portbuffer=(portbuffer & 253)+2; //bit 2 HI, pin 3 HI; sckl auf hi
                        _outp(ba,portbuffer);
                        warten1();
                }
        }


        portbuffer=(portbuffer & 251)+4; //bit 3 HI, pin 4 HI; din auf hi
        _outp(ba,portbuffer);

        // eventuell Pause einfügen
        warten1();

        for(j=0;j<readanzahl;j++)
        {
                bitwert=128;
                wert=0;

                for(i=1;i<=8;i++)
                {
                        portbuffer=portbuffer & 253; //bit 1 lo, pin 2 lo; sclk auf lo
                        _outp(ba,portbuffer);
                        warten1();

                        if((_inp(ba+1) & 128) == 0)
                        {
                                wert += bitwert;
                        }

                        portbuffer=(portbuffer & 253)+2; //bit 1 HI, pin 2 HI; sckl auf hi
                        _outp(ba,portbuffer);
                        warten1();

                        bitwert /= 2;
                }
                readdata[j]=wert;
        }


        portbuffer=(portbuffer & 247)+8; //bit 4 HI, pin 5 HI; CS auf Hi adc Datenübertragung Ende
        _outp(ba,portbuffer);
        warten1();
}

void ads1241read(USHORT ba, int *readdata1,int *readdata2,int *readdata3,int *readdata4,int *writedata,int
readanzahl,int writeanzahl) // Für 30-Kanal
{
        int i,j,byte,bitwert=0,bitwert1=0,bitwert2=0,bitwert3=0,bitwert4=0,wert1,wert2,wert3,wert4;
        UCHAR portbuffer=255;

        portbuffer=_inp(ba);

        portbuffer=portbuffer & 247; //bit 4 lo, pin 5 lo; CS auf lo adc für Datenübertragung beendet
        _outp(ba,portbuffer);
```

```
warten1();

for(j=0;j<writeanzahl;j++)
{
	byte=writedata[j];
	for(i=7;i>=0;i--) //Datennotieren
	{

		bitwert=1<<i;

		if(byte & bitwert) // HI ausgeben
		{
			portbuffer=(portbuffer & 251)+4; //bit 3 HI, pin 4 HI; din auf hi
			_outp(ba,portbuffer);
			warten1();
		}
		else // LO ausgeben
		{
			portbuffer=(portbuffer & 251); //bit 3 lo, pin 4 lo; din auf lo
			_outp(ba,portbuffer);
			warten1();
		}

		portbuffer=portbuffer & 253; //bit 2 lo, pin 3 lo; sclk auf lo // Daten in den
Wandler schreiben
		_outp(ba,portbuffer);
		warten1();

		portbuffer=(portbuffer & 253)+2; //bit 2 HI, pin 3 HI; sckl auf hi
		_outp(ba,portbuffer);
		warten1();
	}
}


portbuffer=(portbuffer & 251)+4; //bit 3 HI, pin 4 HI; din auf hi
_outp(ba,portbuffer);

// eventuell Pause einfügen
warten1();

for(j=0;j<readanzahl;j++)
{
	bitwert=128;
	bitwert1=128;
	bitwert2=128;
	bitwert3=128;
	bitwert4=128;
	wert1=0;
	wert2=0;
	wert3=0;
	wert4=0;

	if(messkontrolle.typ==KANAL30)
	{
		for(i=1;i<=8;i++)
		{
			portbuffer=portbuffer & 253; //bit 1 lo, pin 2 lo; sclk auf lo
			_outp(ba,portbuffer);
			warten1();

			if((_inp(ba+1) & 0x40) != 0) // ADC1, PIN 10, ACK, BIT 6
			{
```

```
                                        wert1 += bitwert1;
                                }
                                if((_inp(ba+1) & 0x20) != 0) //ADC2, PIN 12, PE, BIT 5
                                {
                                        wert2 += bitwert2;
                                }
                                if((_inp(ba+1) & 0x10) != 0) //ADC3, PIN 13, SELEKT, BIT 4
                                {
                                        wert3 += bitwert3;
                                }
                                if((_inp(ba+1) & 0x8) != 0) //ADC4, PIN 15, ERROR, BIT 3
                                {
                                        wert4 += bitwert4;
                                }

                                portbuffer=(portbuffer & 253)+2; //bit 1 HI, pin 2 HI; sckl auf hi
                                _outp(ba,portbuffer);
                                warten1();

                                bitwert1 /= 2;
                                bitwert2 /= 2;
                                bitwert3 /= 2;
                                bitwert4 /= 2;
                        }

                        readdata1[j]=wert1;
                        readdata2[j]=wert2;
                        readdata3[j]=wert3;
                        readdata4[j]=wert4;
                }

        }


        portbuffer=(portbuffer & 247)+8; //bit 4 HI, pin 5 HI; CS auf Hi adc Datenübertragung Ende
        _outp(ba,portbuffer);
        warten1();
}



void warten1(void) // wartet 10µS
{

double TimeUnit;

        TimeInit(&TimeUnit);
                warten(50e-6,TimeUnit); //10µS warten,  zur Absicherung

}


void ads1241reset(USHORT ba) //setzt den Wandler auf Ausgangswerte zurück, Ende continuos mode
{
int data=0xfe;//1111 1110

        ads1241write(ba, &data,1);
}

void ads1241dsync(USHORT ba) //startet Modulator des Wandlers
{
int data=0xfc;//1111 1100
```

```
	ads1241write(ba, &data,1);
}


void ads1241systemgaincal(USHORT ba) //system gain calibration
{
int data=0xf4;//1111 0100

	ads1241write(ba, &data,1);
}

void ads1241systemoffsetcal(USHORT ba) //system offset calibration
{
int data=0xf3;//1111 0011

	ads1241write(ba, &data,1);
}

void ads1241selfgaincal(USHORT ba) //self gain calibration
{
int data=0xf2;//1111 0010

	ads1241write(ba, &data,1);
}

void ads1241selfoffsetcal(USHORT ba) //self offset calibration
{
int data=0xf1;//1111 0001

	ads1241write(ba, &data,1);
}

void ads1241selfoffsetgaincal(USHORT ba) //self offset und gain calibration
{
int data=0xf0;//1111 0000

	ads1241write(ba, &data,1);
}


void ads1241dataread(USHORT ba,int *data) //liest das Datenregister DOR aus, wird in der Reihenfolge hibyte, midbyte, lobyte in das
{                                                                                    // in das feld data hineingeschrieben, data ist mindestens 3 lang
	int writedata=0x01;

	ads1241read(ba,data,&writedata,3,2);
}

// setzt positiven Kanal auf ain 1 bis 8 und negativen Kanal auf aincom, gleich agnd
void ads1241kanalselekt(USHORT ba, int kanal) // Kanal betrifft 0 bis 7
{
int data[3],datenraus=0,indata=0,indata1=0,indata2=0,indata3=0,indata4=0;//0000 1000
int dummy;

	data[0]=0x51; // schreibt in Register 1
	data[1]=0x01; // schreibt ein Byte 1 auf Null gesetzt
	data[2]=kanal*0x10+8;// die oberen vier Bit geben den positiven Bereich an;  negativer Kanal auf aincom
	dummy=data[2];
	ads1241write(ba,data,3);
```

```
	do
	{
		data[0]=0x51; // schreibt in Register 1
		data[1]=0x01; // schreibt ein Byte 1 auf Null gesetzt
		datenraus=kanal*0x10+8; // setzt mux bits
		data[2]=datenraus;
		ads1241write(ba,data,3);

		data[0]=0x11; //liest aus Register 1
		data[1]=0x00; //liest nur ein Register
		if(messkontrolle.typ==KANAL8)
			ads1241read(ba,&indata,data,1,2);
		else if(messkontrolle.typ==KANAL30)
		{
			ads1241read(ba,&indata1,&indata2,&indata3,&indata4,data,1,2);
			if(indata1!=datenraus)
				indata=indata1;
		/*	else if(indata2!=datenraus)
				indata=indata2;
			else if(indata3!=datenraus)
				indata=indata3;*/
			else if(indata4!=datenraus)
				indata=indata4;
			else
				indata=datenraus;
		}
	}
	while(indata!=datenraus);
	dummy=0; // debug
}

// setzt pga verstärker
void ads1241pga(USHORT ba, int pga) //Adresse und Verstärkungsfaktor
{
int writedata[3],indata=0,indata1=0,indata2=0,indata3=0,indata4=0,datenraus=0;
unsigned int pgabit;


	pgabit=(unsigned int) round(log((double)pga)/log(2));
	writedata[0]=0x10; //liest aus Register 0
	writedata[1]=0x00; //liest nur ein Register

	if(messkontrolle.typ==KANAL8)
		ads1241read(ba,&indata,writedata,1,2);
	else if (messkontrolle.typ==KANAL30)
	{
		ads1241read(ba,&indata1,&indata2,&indata3,&indata4,writedata,1,2);
		indata=indata1;// alle ADCs haben den gleichen Verstärkungsfaktor, daher reicht einer,
für 2Kanal nur ADC2
	}

	writedata[0]=0x50; // schreibt in Register 0
	writedata[1]=0x01; // schreibt ein Byte von 1 auf Null gesetzt
	datenraus=(indata & 0xf8)+pgabit; // setzt pga bits
	writedata[2]=datenraus;
	ads1241write(ba,writedata,3);

	writedata[0]=0x10; //liest aus Register 0
	writedata[1]=0x00; //liest nur ein Register

	do
	{
		writedata[0]=0x50; // schreibt in Register 0
```

```
                writedata[1]=0x01; // schreibt ein Byte
                datenraus=(indata & 0xf8)+pgabit; // setzt pga bits
                writedata[2]=datenraus;
                ads1241write(ba,writedata,3);

                writedata[0]=0x10; //liest aus Register 0
                writedata[1]=0x01; //liest nur ein Register

                if(messkontrolle.typ==KANAL8)
                        ads1241read(ba,&indata,writedata,1,2);
                else if(messkontrolle.typ==KANAL30)
                {
                        ads1241read(ba,&indata1,&indata2,&indata3,&indata4,writedata,1,2);
                        if(indata1!=datenraus)
                                indata=indata1;
                        /*else if(indata2!=datenraus)
                                indata=indata2;
                        else if(indata3!=datenraus)
                                indata=indata3;*/
                        else if(indata4!=datenraus)
                                indata=indata4;
                        else
                                indata=datenraus;
                }

        }
        while(indata!=datenraus);
}

void ads1241einstellen(USHORT ba) //drdy0, Datenformat unipolar1, speed 1/128 0,buffer aus 0,bitoder 0,
range full 0, datarate 15Hz 00,
{
int writedata[3]={0x52,0x00,0xC0};

        ads1241write(ba,writedata,3);

}

void ads1241klarmachen(USHORT ba) //macht ads1241 für messung bereit
{

UCHAR portbuffer=0xff;
double TimeUnit;

        TimeInit(&TimeUnit);

        warten(0.1,TimeUnit); //100ms warten


        portbuffer=_inp(ba);
        portbuffer=(portbuffer & 254)+1; //bit 1 HI, pin 2 HI; dsync auf hi
        _outp(ba,portbuffer);

        ads1241reset(ba);
        warten(0.1,TimeUnit); //100ms warten
        ads1241einstellen(ba);
        warten(0.1,TimeUnit); //100ms warten
        /*for(k=0;k<10000;k++)
        {
                warten1();
        }*/
}
```

```
double ads1241getu(int ba,int pga, int kanal) // liest einmal Spannung vom Wandler an ba, mit Verstärker pga(1-128), und kanal kanal ein(1-8),
{
int readdata[16],writedata[2]={16,15},k=0;
UCHAR portbuffer=0xff;
double u;
double TimeUnit;

        TimeInit(&TimeUnit);

        portbuffer=_inp(ba);
        portbuffer=(portbuffer & 254)+1; //bit 1 HI, pin 2 HI; dsync auf hi
        _outp(ba,portbuffer);

        ads1241pga(ba,pga);
        warten(10e-3,TimeUnit); //40ms warten, 10 sind auch ausreichend (mindestens 10 ms)

        ads1241kanalselekt(ba,kanal-1);
        warten(10e-3,TimeUnit); //60ms warten

        ads1241selfoffsetgaincal(ba);
        warten(40e-3,TimeUnit); //80ms warten

        portbuffer=_inp(ba);
        portbuffer=(portbuffer & 254); //bit 1 lo, pin 2 lo; dsync auf lo

        warten(80e-3,TimeUnit); //4ms warten

        portbuffer=_inp(ba);
        portbuffer=(portbuffer & 254)+1; //bit 1 hi, pin 2 hi; dsync auf hi

        warten(150e-3,TimeUnit); //300ms warten
        ads1241read(ba,readdata,writedata,16,2);

        u=(readdata[0x0f]+readdata[0x0e]*256+readdata[0x0d]*65536)*UREF/16777215/pga;

        //*mux=readdata[0x01];
        return(u);

}


void ads1241getu30(int ba,int pga, int kanal, double *u1, double *u2, double *u3, double *u4) //30 Kanal liest einmal Spannung vom Wandler an ba, mit Verstärker pga(1-128), und Kanal kanal?? ein(1-8),
{
int readdata1[16],readdata2[16],readdata3[16],readdata4[16],writedata[2]={16,15},k=0;
UCHAR portbuffer=0xff;
double TimeUnit;

        TimeInit(&TimeUnit);

        portbuffer=_inp(ba);
        portbuffer=(portbuffer & 254)+1; //bit 1 HI, pin 2 HI; dsync auf hi
        _outp(ba,portbuffer);

        ads1241pga(ba,pga);

        warten(10e-3,TimeUnit); //40ms warten, 10  sind auch ausreichend (mindestens 10 ms )

        ads1241kanalselekt(ba,kanal-1);

        warten(10e-3,TimeUnit); //60ms warten, 10 gehen auch
```

```
	ads1241selfoffsetgaincal(ba);

	warten(40e-3,TimeUnit); //80ms warten , 40 gehen auch


	portbuffer=_inp(ba);
	portbuffer=(portbuffer & 254); //bit 1 lo, pin 2 lo; dsync auf lo

	warten(80e-3,TimeUnit); //80ms warten

	portbuffer=_inp(ba);
	portbuffer=(portbuffer & 254)+1; //bit 1 hi, pin 2 hi; dsync auf hi

	warten(300e-3,TimeUnit); //300ms warten, dann keine Aufspaltung mehr

	ads1241read(ba,readdata1,readdata2,readdata3,readdata4,writedata,16,2);
	*u1=(readdata1[0x0f]+readdata1[0x0e]*256+readdata1[0x0d]*65536)*UREF/16777215/pga;
	*u2=(readdata2[0x0f]+readdata2[0x0e]*256+readdata2[0x0d]*65536)*UREF/16777215/pga;
	*u3=(readdata3[0x0f]+readdata3[0x0e]*256+readdata3[0x0d]*65536)*UREF/16777215/pga;
	*u4=(readdata4[0x0f]+readdata4[0x0e]*256+readdata4[0x0d]*65536)*UREF/16777215/pga;

	return;
}
```

11.7.2.4 asl.cpp

```
//*************************************************************************
// Funktionen für die Kommunikation mit dem ASL-Gerät
//*************************************************************************

#include "StdAfx.h";

extern REIHENSTEUERSTRUCT messkontrolle;

double getasltemp(void)
{
char datain[1200],data[9],dummy[2];
unsigned long countout=2,countin=11;
double temp=0;
int i=0;


	PurgeComm(messkontrolle.hComasl,PURGE_TXCLEAR&&PURGE_RXCLEAR);	// Puffer
leeren
	WriteFile(messkontrolle.hComasl,"T\n",countout,&countout,NULL);
	i=0;
	do
	{
		countin=1;
		ReadFile(messkontrolle.hComasl,dummy,countin,&countin,NULL);
		datain[i++]=dummy[0];
		if(i>12)
			break;
	}
	while(dummy[0]!='\n');

	strncpy(data,datain+sizeof(char),7);
	data[7]=0;
	if((data[1]==' ') && (data[0]=='-'))
	{
		data[1]='-';
		temp=atof(data+sizeof(char));
```

```
	}
	else
		temp=atof(data);

return(temp+273.15); // Temperatur wird in K zurückgegeben
}

void setasl(void) // stellt ASL ein
{
char dataout[100];
unsigned long stringlaenge;

	strcpy(dataout,"L1P0R1U0\n"); //Bedienfeld verriegelt, Kanal A ausgewählt, drei Nachkommastellen, °C
	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComasl,dataout,stringlaenge,&stringlaenge,NULL);
}
```

11.7.2.5 aslTt.cpp

```
// Funktionen für 1 Kanalschreiber

#include "StdAfx.h"

extern DATENSATZ * messdaten;
extern HGLOBAL hMemHandle1;
extern REIHENSTEUERSTRUCT messkontrolle;
extern HINSTANCE hInst;
extern MESSSTEUERSTRUCT defmessparam;

void aslkanalschreiber(HWND hWnd) //1 Kanal T/t schreiber mit ASL
{
int maxwinx,maxwiny,startx,starty,i=0;
double ywertmax,ywertmin,u=0,messdauer=0;
HDC hDc;
RECT Rect;
char outtext[100];
HPEN hStift[8];
double TimeUnit;
DWORD starttime;
HANDLE hFile32;
char pcCommPort[5];
DCB dcb;

	TimeInit(&TimeUnit);

	if(DialogBox(hInst, (LPCTSTR)IDD_8KanalTt, hWnd, (DLGPROC) StartDialogkanal8Tt)==FALSE)
		return; // bei Abbruch zurück

	sprintf(pcCommPort,"COM%i",messkontrolle.aslcom);
	messkontrolle.hComasl = CreateFile( pcCommPort, GENERIC_READ | GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für ASL

	dcb.ByteSize = 28;		// data size, xmit, and rcv
	dcb.BaudRate = CBR_9600;	// set the baud rate
	dcb.fBinary=1;			// binary mode, no EOF check
	dcb.fParity=0;		// enable parity checking
	dcb.fOutxCtsFlow=0;		// CTS output flow control
	dcb.fOutxDsrFlow=0;		// DSR output flow control
	dcb.fDtrControl=1;		// DTR flow control type
	dcb.fDsrSensitivity=0;	// DSR sensitivity
```

```
        dcb.fTXContinueOnXoff=1; // XOFF continues Tx
        dcb.fOutX=0;           // XON/XOFF out flow control
        dcb.fInX=0;            // XON/XOFF in flow control
        dcb.fErrorChar=0;      // enable error replacement
        dcb.fNull=0;           // enable null stripping
        dcb.fRtsControl=1;     // RTS flow control
        dcb.fAbortOnError=0;   // abort reads/writes on error
        dcb.XonLim=2048;             // transmit XON threshold
        dcb.XoffLim=512;             // transmit XOFF threshold
        dcb.ByteSize=8;            // number of bits/byte, 4-8
        dcb.Parity=NOPARITY;              // 0-4=no,odd,even,mark,space
        dcb.StopBits=TWOSTOPBITS;           // 0,1,2 = 1, 1.5, 2
        dcb.XonChar=17;              // Tx and Rx XON character
        dcb.XoffChar=19;             // Tx and Rx XOFF character
        dcb.ErrorChar=0;           // error replacement character
        dcb.EofChar=0;             // end of input character
        dcb.EvtChar=0;             // received event character


        SetCommState(messkontrolle.hComasl, &dcb);
        PurgeComm(messkontrolle.hComasl,PURGE_TXCLEAR&&PURGE_RXCLEAR);   //   Puffer
leeren
        setasl(); // stellt ASL ein


        messkontrolle.messtyp=ASLTt; //Einstellungen für diesen Messtyp
        messkontrolle.messan=TRUE; // eine Messung läuft
        messkontrolle.maximalerywert=50;
        messkontrolle.minmalerywert=-70;

        drawkosy(hWnd, NOPAINTMSG);

        hStift[0]=CreatePen(PS_SOLID,0,RGB(0,0,0)); // einzelne Stifte herrichten

        ywertmax=messkontrolle.maximalerywert;
        ywertmin=messkontrolle.minmalerywert;

        hDc=GetDC(hWnd);// Fenster Parameter
        GetClientRect(hWnd, &Rect);
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;
        starty=(int) (maxwiny-UAB-OAB)+OAB;
        startx=(int) LAB;

        messkontrolle.flagkanal1=TRUE; // Messkanal setzen
        messkontrolle.flagkanal2=FALSE;
        messkontrolle.flagkanal3=FALSE;
        messkontrolle.flagkanal4=FALSE;
        messkontrolle.flagkanal5=FALSE;
        messkontrolle.flagkanal6=FALSE;
        messkontrolle.flagkanal7=FALSE;
        messkontrolle.flagkanal8=FALSE;

        hFile32=CreateFile(defmessparam.tempfile,GENERIC_WRITE,FILE_SHARE_READ,NULL,CR
EATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
        CloseHandle(hFile32); // altes Tempfile löschen

        GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden

        sprintf(outtext,"asdf");
        TextOut(hDc,LAB+10,(OAB-TEXTYDY)/2,outtext,strlen(outtext));

        warten(0.5,TimeUnit); // 3.5s warten, dann Gerät wieder bereit
```

```
	getasltemp(); // ersten Wert auslesen und verwerfen.
	warten(3.5,TimeUnit); // 3.5s warten, dann Gerät wieder bereit
	starttime=GetTickCount();

	do
	{
		messdauer=(GetTickCount()-starttime)/1000;
		messdaten[i].t=messdauer;
		messdaten[i].ywerte[0]=getasltemp()-273.15;

		sprintf(outtext,"%.0lf s, %.3lf °C, %.3lf K", messdauer,
messdaten[i].ywerte[0],messdaten[i].ywerte[0]+273.15);
		TextOut(hDc,LAB+10,(OAB-TEXTYDY)/2,outtext,strlen(outtext));

		SelectObject(hDc,hStift[0]);
		if(i>0)
		{
			MoveToEx(hDc,(int)(startx+(messdaten[i-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[i-
1].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);
			LineTo(hDc,(int)(startx+(messdaten[i].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round ((starty-
(messdaten[i].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))));
		}
		saveuntermessung(hWnd,i); // Daten unter der Messung speichern
		i++;
		warten(3.5,TimeUnit); // 3,5s warten, dann Gerät wieder bereit
		if(GetAsyncKeyState('E') != 0)
		{
			messkontrolle.messdauer=messdauer;
			break; // Messung abbrechen
		}
	}
	while(messdauer<=messkontrolle.messdauer);

	messkontrolle.anzahl=i;

	GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden,
um das Speichern nicht zu stören
	CloseHandle(messkontrolle.hComasl); // Schnittstelle wieder geschlossen
	save1ut(hWnd); // Daten speichern

	DeleteObject(hStift[0]);
	ReleaseDC(hWnd,hDc);
	messkontrolle.messan=FALSE; // keine Messung läuft
}
```

11.7.2.6 auswert.cpp

```
#include "StdAfx.h"
extern DATENSATZ30 * messdaten30;
ZOOMPOS zoompos;
KOORDPOS auswertpunkte,einzelpunkt;
char auswertnam[MAX_PATH];
extern REIHENSTEUERSTRUCT messkontrolle;
extern HWND mainhwnd;

void kappatauswert(HWND hWnd)
{
unsigned long int i=0,j=0;
OPENFILENAME ofn;
static char  szFilename[MAX_PATH] = ".txt";
int iErg=0,readcount=0;
```

```
HFILE hFile;
char readdata[300],read1;
double dummy=0;
long int dateilaenge=0;

	messkontrolle.count=0;
// Daten einlesen
	memset(&ofn, 0, sizeof(OPENFILENAME));
	ofn.lpstrFile = szFilename;
	iErg = FileDialogwert(hWnd, &ofn);

	auswertpunkte.i=0; // Counter für Auswertung auf 0
	einzelpunkt.i=0;
	zoompos.i=0;
	zoompos.status=UNGEZOOMED;
	zoompos.count[0]=0;
//	zoompos.altesdutemp=0;
	if(iErg == IDOK)
	{
		//Datei öffnen mit Fehlertest
		if((hFile = _lopen(ofn.lpstrFile, OF_READ)) == -1)
		{
			MessageBox(hWnd, "Fehler beim Dateneinlesen", "Achtung",
MB_OK|MB_ICONSTOP);
			SendMessage(hWnd,WM_CLOSE,NULL,NULL);
			messkontrolle.messtyp=NIXMESS;
			return;
		}

		dateilaenge=_llseek(hFile, 0L, FILE_END);
		//an den Dateianfang gehen
		_llseek(hFile, 0L, 0);
	// Datei lesen, noch Fehlerprüfung dazufügen
		do
		{

			readcount=0;
			do //eine Zeile einlesen
			{
				_hread(hFile,&read1,1);
				readdata[readcount++]=read1;
			}
			while(readdata[readcount-1]!=13 ); //Zeilenende

			readdata[readcount-1]=0; //string beenden
			_hread(hFile,&read1,1); // noch Zeilenvorschub einlesen

			if(messkontrolle.messtyp==KAPPATAUSWERT)
			{
				sscanf(readdata,"%lf %lf %lf %lf %lf
%lf",&dummy,&dummy,&dummy,&dummy,&messdaten30[messkontrolle.count].ywerte[0],&messdaten30[m
esskontrolle.count].ywerte[32]);
				messkontrolle.count++; // Datenpunktzähler weiterzählen
			}
			else if(messkontrolle.messtyp==TTAUSWERT)
			{
				sscanf(readdata,"%lf %lf %lf %lf %lf
%lf",&messdaten30[messkontrolle.count].ywerte[0],&messdaten30[messkontrolle.count].ywerte[32],&dummy
,&dummy,&dummy,&dummy);
				messkontrolle.count++; // Datenpunktzähler weiterzählen
			}
			else if(messkontrolle.messtyp==KAPPATTAUSWERT)
			{
```

```
                                        sscanf(readdata,"%lf %lf %lf %lf %lf
%lf",&dummy,&dummy,&messdaten30[messkontrolle.count].ywerte[0],&messdaten30[messkontrolle.count].
ywerte[32],&dummy,&dummy);
                                        messkontrolle.count++; // Datenpunktzähler weiterzählen
                                }


                        }
                        while(_llseek(hFile, 0L, FILE_CURRENT)<dateilaenge);

                        _lclose(hFile);
                        messkontrolle.anzahl=messkontrolle.count;
                        zoompos.count[0]=messkontrolle.anzahl;
                        zoompos.dutemp[0]=0;

                        readcount=0;
                        readcount = strlen(ofn.lpstrFile);

                        do
                        {
                                readcount--;
                        }
                        while((ofn.lpstrFile[readcount] != '\\') && (readcount>=0));

                        for(i=readcount+1;i<strlen(ofn.lpstrFile);i++)
                        {
                                auswertnam[j]=ofn.lpstrFile[i];
                                j++;
                                if(j>=MAX_PATH)
                                        break;
                        }
                        auswertnam[j]=0;

                        SendMessage(mainhwnd,WM_PAINT,0,0); // Daten zeichnen

                }
                else
                {
                        messkontrolle.messtyp=NIXMESS;
                        return;
                }
}


int FileDialogwert(HWND hWnd, OPENFILENAME *ofn)
{
        char szFilter[256],
        szText[]= "TEXT-Datei (*.TXT)|*.txt|Alle Dateien (*.*)|*.*||";
        int i;

        for(i=0; szText[i]!= '\0'; ++i)
          szFilter[i] = szText[i] == '|' ? '\0' : szText[i];

        ofn->lStructSize = sizeof(OPENFILENAME);
        ofn->hwndOwner = hWnd;
        ofn->lpstrFilter = szFilter;
        ofn->nMaxFile = _MAX_PATH;
        ofn->nMaxFileTitle = _MAX_FNAME + _MAX_EXT;
        ofn->lpstrDefExt = "*";
                return GetOpenFileName(ofn)? IDOK : IDCANCEL;
}
```

```
void bearbeitekoords(LPARAM lParam,HWND hWnd,double *x,double *y, long int *count ) // berechnet die Koordinaten im kappa / tempsystem
{
int xPos=0, yPos=0;
double kappakoord=0, tempkoord=0;
double xwertmax,ywertmax,ywertmin,xwertmin,xdiffmin=0,ydiffmin=0;
long int auswertcount=0,xcount=0, start=0, ende=0;
int maxwinx,maxwiny,startx,starty;
RECT Rect;

	GetClientRect(hWnd, &Rect);
	InvalidateRect(hWnd,&Rect,0);
	maxwinx=Rect.right;
	maxwiny=Rect.bottom;

	starty=(int) round((maxwiny-UAB-OAB)+OAB);
	startx=(int) round(LAB);

	if(zoompos.status==UNGEZOOMED)
	{
		start=0;
		ende=messkontrolle.anzahl;
	}
	else
	{
		if(zoompos.count[0]<zoompos.count[1])
		{
			start=zoompos.count[0];
			ende=zoompos.count[1];
		}
		else
		{
			start=zoompos.count[1];
			ende=zoompos.count[0];
		}
	}

	// Extrema bestimmen
	xwertmax=messdaten30[start].ywerte[0];

	for(auswertcount=start;auswertcount<ende;auswertcount++)
	{
		if(xwertmax<messdaten30[auswertcount].ywerte[0])
			xwertmax=messdaten30[auswertcount].ywerte[0];
	}

	xwertmin=messdaten30[start].ywerte[0];
	for(auswertcount=start;auswertcount<ende;auswertcount++)
	{
		if(xwertmin>messdaten30[auswertcount].ywerte[0])
			xwertmin=messdaten30[auswertcount].ywerte[0];
	}

	ywertmax=messdaten30[start].ywerte[32];
	for(auswertcount=start;auswertcount<ende;auswertcount++)
	{
		if(ywertmax<messdaten30[auswertcount].ywerte[32])
			ywertmax=messdaten30[auswertcount].ywerte[32];
	}

	ywertmin=messdaten30[start].ywerte[32];
	for(auswertcount=start;auswertcount<ende;auswertcount++)
	{
```

```
                if(ywertmin>messdaten30[auswertcount].ywerte[32])
                        ywertmin=messdaten30[auswertcount].ywerte[32];
        }

        xPos = LOWORD(lParam);  // horizontal position of cursor
        yPos = HIWORD(lParam);  // vertical position of cursor

        // kosytrafo
        tempkoord= (double) (xPos-startx)/(maxwinx-RAB-LAB)*((xwertmax-xwertmin))+xwertmin;
        kappakoord=(double) -(yPos-starty)/(maxwiny-UAB-OAB)*((ywertmax-ywertmin))+ywertmin;

        // Zähler heraussuchen.
        xdiffmin=1e20; // zu großen Wert eingeben
        for(auswertcount=0;auswertcount<messkontrolle.anzahl;auswertcount++)
        {
                if(fabs(tempkoord-messdaten30[auswertcount].ywerte[0])<xdiffmin)
                {
                        xdiffmin=fabs(tempkoord-messdaten30[auswertcount].ywerte[0]);
                        xcount=auswertcount;
                }
        }
        *x=tempkoord;
        *y=kappakoord;
        *count=xcount;
}

void bearbeitemausebutton(LPARAM lParam,HWND hWnd)
{
double a=0,b=0,r=0;
long int start=0,ende=0;

        if(auswertpunkte.i>=4)
                auswertpunkte.i=0;
        bearbeitekoords(lParam,hWnd,&auswertpunkte.dutemp[auswertpunkte.i],&auswertpunkte.lnkappa[a
uswertpunkte.i],&auswertpunkte.count[auswertpunkte.i]);
        auswertpunkte.i+=1;

        // lineare Regression
        if(auswertpunkte.i==2)
        {
                if(auswertpunkte.count[0]<auswertpunkte.count[1])
                {

        linreg(auswertpunkte.count[0],auswertpunkte.count[1],&auswertpunkte.a[0],&auswertpunkte.b[0],&a
uswertpunkte.r[0]);
                }
                else
                {

        linreg(auswertpunkte.count[1],auswertpunkte.count[0],&auswertpunkte.a[0],&auswertpunkte.b[0],&a
uswertpunkte.r[0]);
                }

        }
        else if(auswertpunkte.i==4)
        {
                if(auswertpunkte.count[2]<auswertpunkte.count[3])
                {

        linreg(auswertpunkte.count[2],auswertpunkte.count[3],&auswertpunkte.a[1],&auswertpunkte.b[1],&a
uswertpunkte.r[1]);
                }
                else
```

```
                {

        linreg(auswertpunkte.count[3],auswertpunkte.count[2],&auswertpunkte.a[1],&auswertpunkte.b[1],&a
uswertpunkte.r[1]);
                }
        }

        // Schnittpunkt bestimmen
        if(auswertpunkte.i==4)
                auswertpunkte.schnittx=(auswertpunkte.a[0]-auswertpunkte.a[1])/(auswertpunkte.b[1]-
auswertpunkte.b[0]);

        SendMessage(mainhwnd,WM_PAINT,0,0); // Daten zeichnen
}

/*********************************************************************/
/* Funktion zur linearen Regression                                  */
/* x und y double Felder f□ r Koordinaten n= Anzahl der Datens,,tze   */
/* r□ ckgabe von a b (f□ r y= Bx + A) und Regressionskoeffizient r als*/
/* double Variable
        */
/*********************************************************************/

void linreg(long int start,long int ende,double *a, double *b, double *r)
{
double sx=0,sy=0,s2x=0,s2y=0,sxy=0,xi=0,yi=0;
long int i=0;

        for(i=start;i<ende;i++)
        {
                xi=messdaten30[i].ywerte[0];
                yi=messdaten30[i].ywerte[32];
                sx+=xi;
                sy+=yi;
                sxy+=xi*yi;
                s2x+=pow(xi,2);
                s2y+=pow(yi,2);
        }

        *b=((ende-start) * sxy - sx * sy) / ((ende-start) *s2x - pow(sx,2));
        *a=(sy - *b * sx) / (ende-start);
        *r=((ende-start) * sxy - sx * sy) / sqrt(((ende-start) *s2x - pow(sx,2)) * ((ende-start) *s2y -
pow(sy,2)));

}

void bearbeitemauserechts(LPARAM lParam,HWND hWnd)
{
        if(einzelpunkt.i>=MAXEINZELPUNKT)
                einzelpunkt.i=0;
        bearbeitekoords(lParam,hWnd,&einzelpunkt.dutemp[einzelpunkt.i],&einzelpunkt.lnkappa[einzelpun
kt.i],&einzelpunkt.count[einzelpunkt.i]);
        einzelpunkt.i+=1;
        SendMessage(mainhwnd,WM_PAINT,0,0); // Daten zeichnen
}


void bearbeitemausemitte(LPARAM lParam,HWND hWnd) // einzoomen
{
double dummy;

        if(zoompos.i>=2)
                zoompos.i=0;
```

```
	bearbeitekoords(lParam,hWnd,&zoompos.dutempneu[zoompos.i],&dummy,&zoompos.countneu[z
oompos.i]);

	zoompos.i+=1;

	if(zoompos.i==1)
	{
		zoompos.tempgelbstrich=zoompos.dutempneu[zoompos.i-1];
	}
	else if(zoompos.i==2)
	{
		zoompos.dutemp[0]=zoompos.dutempneu[0];
		zoompos.dutemp[1]=zoompos.dutempneu[1];
		zoompos.count[0]=zoompos.countneu[0];
		zoompos.count[1]=zoompos.countneu[1];
		zoompos.status=GEZOOMED;
	}

	SendMessage(mainhwnd,WM_PAINT,0,0); // Daten zeichnen
}

void bearbeitemausemitteshift(LPARAM lParam,HWND hWnd) // Zoom zurücksetzen
{
	zoompos.i=0;
	/*zoompos.dutemp[0]=;
	zoompos.dutemp[1]=;
	zoompos.count[0]=;
	zoompos.count[1]=;*/
	zoompos.status=UNGEZOOMED;
	SendMessage(mainhwnd,WM_PAINT,0,0); // Daten zeichnen
}
```

11.7.2.7 comlfthermo.cpp

```
// Funktionen für die Kommunikation über RS-232; Schnittstelle für Leitfähigkeit und Thermometer
#include "StdAfx.h"

extern REIHENSTEUERSTRUCT messkontrolle;

void setledmessen(int geraet,int status) // Setzt die Mess-LEDs
{
char dataout[100];
unsigned long stringlaenge;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	if((geraet==THERMO) && (status ==1)) // Thermometer LED an
	{
		strcpy(dataout,"hgtm1\r\n"); //
	}
	else if((geraet==THERMO) && (status ==0)) // Thermometer LED aus
	{
		strcpy(dataout,"hgtm0\r\n"); //
	}
	else if((geraet==LEITFAEHIGKEIT) && (status ==1)) // Thermometer LED an
	{
		strcpy(dataout,"hglm1\r\n"); //
	}
	else if((geraet==LEITFAEHIGKEIT) && (status ==0)) // Thermometer LED aus
	{
		strcpy(dataout,"hglm0\r\n"); //
	}
```

```
	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
}

void setstoerung(int status) // Setzt die LED Störung am Thermometer
{
char dataout[100];
unsigned long stringlaenge;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	if(status==1)
		strcpy(dataout,"hgts1\r\n");
	else
		strcpy(dataout,"hgts0\r\n");

	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
}

void wandelfeld(int status) //  Wandelfeldrührer ein/aus
{
char dataout[100];
unsigned long stringlaenge;


	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren

	if(status==1)
		strcpy(dataout,"hglr1\r\n");
	else
		strcpy(dataout,"hglr0\r\n");

	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
}

void zahnradruehrer(int motor) //  Zahnradrührer
{
char dataout[100];
unsigned long stringlaenge;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	sprintf(dataout,"hgtr%i\r\n",motor);
	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
}

void spannung(double u) //  Spannungsausgabe
{
char dataout[100];
unsigned long stringlaenge;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	sprintf(dataout,"hgtu%lf\r\n",u*1e6); // Daten in in µV übertragen
	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
}
```

```
void frequenz(double f) //  Frequenzausgabe
{
char dataout[100];
unsigned long stringlaenge;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	sprintf(dataout,"hglf%.0lf\r\n",f);
	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
}

void uhrstatus(int status) // Uhr ein bzw. aus
{
char dataout[100];
unsigned long stringlaenge;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	if(status==1)
		strcpy(dataout,"hgta1\r\n");
	else
		strcpy(dataout,"hgta0\r\n");

	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
}

void reset(void)
{
char dataout[100];
unsigned long stringlaenge;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	strcpy(dataout,"hgrr\r\n");
	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);

}

long int uhrzeit(void) // holt Uhrzeit
{
char dataout[100],dummy;
unsigned long stringlaenge;
DWORD  dummy1;
int i=0;
long int zeit=0;
double ddummy=0;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren

	strcpy(dataout,"hgtz\r\n");
	stringlaenge=strlen(dataout);
	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);
	do
	{
		ReadFile(messkontrolle.hComthermolf,&dummy,1,&dummy1,NULL);
		dataout[i++]=dummy;
	}
	while(dummy!='\r');
```

```
	dataout[i-1]=0;
	sscanf(dataout,"%lf",ddummy);
	zeit = (long int) ddummy;
	return(zeit);
}

int messalle(double *u, long int *zeit) //initialisiert Messungen an beiden
{
char dataout[1000], dummy=0, zeile[100];
unsigned long stringlaenge;
double TimeUnit;
double u1,u2,u3,u4,u5,u6,u7,u8;
long int i=0, l=0;
int j=0, m=0;
DWORD dummy1=0;
double ddummy=0;

	TimeInit(&TimeUnit);

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren

	strcpy(dataout,"hg32\r\n");
	stringlaenge=strlen(dataout);

	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);


	do // gesammte Übertragung einlesen
	{
		ReadFile(messkontrolle.hComthermolf,&dummy,1,&dummy1,NULL);
		dataout[i++]=dummy;
	}
	while(dummy!='E');

	dataout[i-1]=0;

	for(j=0;j<8;j++)
	{
		i=0;
		do // einzelne Zeilen parsen
		{
			zeile[i]=dataout[m++];
		}
		while(zeile[i++]!='\r');
		zeile[i-1]=0;
		sscanf(zeile,"%li	%lf	%lf	%lf	%lf	%lf	%lf	%lf	%lf
%lf",&l,&ddummy,&u1,&u2,&u3,&u4,&u5,&u6,&u7,&u8);
		zeit[j]=(long int) ddummy;
		u[4*j]=u1;
		u[4*j+1]=u2;
		u[4*j+2]=u3;
		u[4*j+3]=u4;
		u[4*j+32]=u5;
		u[4*j+1+32]=u6;
		u[4*j+2+32]=u7;
		u[4*j+3+32]=u8;
	}
	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	return(OK);
}
```

```
int messkanal(int kanal,double *u, long int *zeit) //gibt Messungen eines Kanals aus
{
char dataout[1000], dummy=0, zeile[100];
unsigned long stringlaenge;
double TimeUnit;
double u1,u2,u3,u4,u5,u6,u7,u8;
long int i=0, timedummy=0, l=0;
int j=0, m=0;
DWORD dummy1=0;
double ddummy=0;

	TimeInit(&TimeUnit);

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren

	sprintf(dataout,"hgmm%i\r\n",kanal);
	stringlaenge=strlen(dataout);

	WriteFile(messkontrolle.hComthermolf,dataout,stringlaenge,&stringlaenge,NULL);

	do // gesammte Übertragung einlesen
	{
		ReadFile(messkontrolle.hComthermolf,&dummy,1,&dummy1,NULL);
		dataout[i++]=dummy;
	}
	while(dummy!='E');

	dataout[i-1]=0;

	i=0;
	do // einzelne Zeilen parsen
	{
		zeile[i]=dataout[m++];
	}
	while(zeile[i++]!='\r');
	zeile[i-1]=0;
	sscanf(zeile,"%li	%lf	%lf	%lf	%lf	%lf	%lf	%lf	%lf
%lf",&l,&ddummy,&u1,&u2,&u3,&u4,&u5,&u6,&u7,&u8);
	*zeit=(long int) ddummy;
	u[0]=u1; // ab hier thermo
	u[1]=u2;
	u[2]=u3;
	u[3]=u4;
	u[4]=u5; // ab hier LF
	u[5]=u6;
	u[6]=u7;
	u[7]=u8;

	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
	return(OK);
}
```

11.7.2.8 ds1305.cpp

```
// Funktionen für die Ansteuerung des ds1305 Uhrchips
#include "StdAfx.h"

extern REIHENSTEUERSTRUCT messkontrolle;

void startclock(int ba) //startet den Oszillator auf dem ds1305 und setzt das 24h- Bit (0x82 bit 6) auf 24h-
Modus
```

```
{
int data=0;

        ds1305byteread(ba,0x0f,&data);
        ds1305bytewrite(ba,0x8f,data&63); // schreibt in das Kontollregister eine 0 --> oszi an und
schreiben an

        ds1305bytewrite(ba,0x80,0); //ganze Uhr auf null
        ds1305bytewrite(ba,0x81,0);
        ds1305bytewrite(ba,0x82,0); //setzt auch Uhr in 24h- Modus
        ds1305bytewrite(ba,0x83,0);
        ds1305bytewrite(ba,0x84,0);
        ds1305bytewrite(ba,0x85,0);
        ds1305bytewrite(ba,0x86,0);

        ds1305byteread(ba,0x0f,&data);
        ds1305bytewrite(ba,0x8f,(data&191)+64); // schreibt in das Kontollregister bit 6 eine 1 --> und
schreiben aus
}

void clockoff(int ba) //schaltet den Oszillator auf dem ds1305 aus
{
int data=0;

        ds1305byteread(ba,0x0f,&data);
        ds1305bytewrite(ba,0x8f,(data&127)+128); // schreibt in das Kontollregister auf das bit 7 eine 1

}

long int gettime(int ba)
{
long int sec1=0,sec2=0;
int dummy=0;
long int d1h=0,d1l=0,h1h=0,h1m=0,h1l=0,m1h=0,m1l=0,s1h=0,s1l=0;
long int d2h=0,d2l=0,h2h=0,h2m=0,h2l=0,m2h=0,m2l=0,s2h=0,s2l=0;

        // erstes Ablesen der Uhrzeit
        ds1305byteread(ba,0x04,&dummy); //liest tage
        sec1=( (dummy>>4*10) + (dummy&0x0f) )*24*3600; //BCD dekodieren
        dummy=0;

        ds1305byteread(ba,0x02,&dummy); //liest Stunden
        sec1+=( ((dummy>>4)&0x01)*10+ ((dummy>>5)&0x01)*20 + (dummy&0x0f) )*3600; //BCD
dekodieren
        dummy=0;

        ds1305byteread(ba,0x01,&dummy); //liest Minuten
        sec1+=( (dummy>>4)*10 + (dummy&0x0f) )*60; //BCD dekodieren
        dummy=0;

        ds1305byteread(ba,0x00,&dummy); //liest Sekunden
        sec1+=(dummy>>4)*10 + (dummy&0x0f); //BCD dekodieren
        dummy=0;

        // zweites Ablesen der Uhrzeit
        ds1305byteread(ba,0x04,&dummy); //liest Tage
        sec2=( (dummy>>4*10) + (dummy&0x0f) )*24*3600; //BCD dekodieren
        dummy=0;

        ds1305byteread(ba,0x02,&dummy); //liest Stunden
        sec2+=( ((dummy>>4)&0x01)*10+ ((dummy>>5)&0x01)*20 + (dummy&0x0f) )*3600; //BCD
dekodieren
        dummy=0;
```

```
	ds1305byteread(ba,0x01,&dummy); //liest Minuten
	sec2+=( (dummy>>4)*10 + (dummy&0x0f) )*60; //BCD dekodieren
	dummy=0;

	ds1305byteread(ba,0x00,&dummy); //liest Sekunden
	sec2+=(dummy>>4)*10 + (dummy&0x0f); //BCD dekodieren

	if(sec2>=sec1) // falls beim ersten Mal einlesen die Uhrzeit umgeschaltet wird, ist der Wert von sec1 kleiner als sec2 und sec2 gültig
		return(sec2);
	else // falls beim zweiten Mal einlesen die Uhrzeit umgeschaltet wird, ist der Wert von sec2 kleiner als sec1 und sec1 gültig, wenn man noch die nächste Sekunde hinzuaddiert
		return(sec1+1);
}

void ds1305bytewrite(int ba, int adresse, int data) // schreibt ein Byte data in die Adresse Adresse ?? 2 mal?? des ds1305 am lpt-port
{
int bitwert,portbuffer=0,portbufferbi=0,i;
double TimeUnit;

	TimeInit(&TimeUnit);

	portbuffer=_inp(ba); // Staus der Schnittstelle sichern, damit die anderen Bits, die nicht gebraucht
                                                //werden erhalten bleiben

	warten(1e-6,TimeUnit);

	if(messkontrolle.typ==KANAL8)
	{
		portbuffer=(portbuffer & 191)+64; //PIN 8, BIT 7,  CE auf HI ds1305 für Datenübertragung fertig stellen
		_outp(ba,portbuffer);
	}
	else if(messkontrolle.typ==KANAL30)
	{
		portbufferbi=_inp(ba+2); // auch Bidirektionale werden gesichert
		portbufferbi=(portbufferbi & 0xFD); //PIN 14, BIT 2, Achtung invertiert,  CE auf HI ds1305 für Datenübertragung fertig stellen
		_outp(ba+2,portbufferbi);
	}

	warten(1e-6,TimeUnit);
	for(i=7;i>=0;i--) //Datennotieren, Adresse wählen
	{
		portbuffer=(portbuffer & 223)+32; //bit 6 HI, pin 7 HI; sckl auf hi
		_outp(ba,portbuffer);
		warten(1e-6,TimeUnit);
		bitwert=1<<i;

		if(adresse&bitwert) // HI ausgeben
		{
			portbuffer=(portbuffer & 239)+16; //bit 5 HI, pin 6 HI; din auf hi
			_outp(ba,portbuffer);
			warten(1e-6,TimeUnit);
		}
		else // LO ausgeben
		{

			portbuffer=(portbuffer & 239); //bit 5 lo, pin 6 lo; din auf lo
			_outp(ba,portbuffer);
			warten(1e-6,TimeUnit);
```

```
		}

		portbuffer=portbuffer & 223; //bit 6 lo, pin 7 lo; sclk auf lo
		_outp(ba,portbuffer);
		warten(1e-6,TimeUnit);
	}

	for(i=7;i>=0;i--) //Datennotiern, Byte ausgeben
	{
		portbuffer=(portbuffer & 223)+32; //bit 6 HI, pin 7 HI; sckl auf hi
		_outp(ba,portbuffer);
		warten(1e-6,TimeUnit);

		bitwert=1<<i;

		if(data&bitwert) // HI ausgeben
		{
			portbuffer=(portbuffer & 239)+16; //bit 5 HI, pin 6 HI; din auf hi
			_outp(ba,portbuffer);
			warten(1e-6,TimeUnit);
		}
		else // LO ausgeben
		{

			portbuffer=(portbuffer & 239); //bit 5 lo, pin 6 lo; din auf lo
			_outp(ba,portbuffer);
			warten(1e-6,TimeUnit);
		}

		portbuffer=portbuffer & 223; //bit 1 lo, pin 2 lo; sclk auf lo
		_outp(ba,portbuffer);
		warten(1e-6,TimeUnit);
	}

	if(messkontrolle.typ==KANAL8)
	{
		portbuffer=(portbuffer & 191); //bit 7 lo, pin 8 lo; CS auf lo ds1305 Datenübertragung
ende
		_outp(ba,portbuffer);
	}

	else if(messkontrolle.typ==KANAL30)
	{
		portbufferbi=(portbufferbi & 0xFD)+2; //PIN 14, BIT 2,achtung invertiert,  CE auf lo
ds1305 Datenübertragung Ende
		_outp(ba+2,portbufferbi);
	}
	warten(1e-6,TimeUnit);
}

void ds1305byteread(int ba, int adresse, int *data) // schreibt ein Byte data in die Adresse Adresse?? des
ds1305 am lpt-port ba
{
int bitwert,portbuffer=0,portbufferbi=0,i,wert=0;
double TimeUnit;

	TimeInit(&TimeUnit);
	portbuffer=_inp(ba); // Staus der Schnittstelle sichern, damit die anderen Bits, die nicht gebraucht
                                                //werden, erhalten bleiben


	if(messkontrolle.typ==KANAL8)
	{
```

```
                portbuffer=(portbuffer & 191)+64; //PIN 8, BIT 7,  CE auf HI ds1305 für
atenübertragung fertig stellen
                warten(1e-6,TimeUnit);
                _outp(ba,portbuffer);
                warten(1e-6,TimeUnit);
        }
        else if(messkontrolle.typ==KANAL30)
        {
                portbufferbi=_inp(ba+2); // auch Bidirektionale werden gesichert
                portbufferbi=(portbufferbi & 0xFD); //PIN 14, BIT 2, Achtung invertiert,  CE auf HI
ds1305 für Datenübertragung fertig stellen
                warten(1e-6,TimeUnit);
                _outp(ba+2,portbufferbi);
                warten(1e-6,TimeUnit);
        }

        for(i=7;i>=0;i--) //Datennotieren, Adresse wählen
        {
                portbuffer=(portbuffer & 223)+32; //bit 6 HI, pin 7 HI; sckl auf hi
                _outp(ba,portbuffer);
                warten(1e-6,TimeUnit);

                bitwert=1<<i;

                if(adresse&bitwert) // HI ausgeben
                {
                        portbuffer=(portbuffer & 239)+16; //bit 5 HI, pin 6 HI; din auf hi
                        _outp(ba,portbuffer);
                        warten(1e-6,TimeUnit);
                }
                else // LO ausgeben
                {

                        portbuffer=(portbuffer & 239); //bit 5 lo, pin 6 lo; din auf lo
                        _outp(ba,portbuffer);
                        warten(1e-6,TimeUnit);
                }

                portbuffer=portbuffer & 223; //bit 1 lo, pin 2 lo; sclk auf lo
                _outp(ba,portbuffer);
                warten(1e-6,TimeUnit);
        }

        bitwert = 128;

        for(i=7;i>=0;i--) //Datennotieren, Adresse einlesen
        {
                portbuffer=(portbuffer & 223)+32; //bit 6 HI, pin 7 HI; sckl auf hi
                _outp(ba,portbuffer);
                warten(1e-6,TimeUnit);


                if(messkontrolle.typ==KANAL8)
                {
                        if((_inp(ba+1) & 32)) //PIN 12; paperend einlesen, wenn hi dann um passendes
bit höherzählen
                                wert += bitwert;
                }
                else if(messkontrolle.typ==KANAL30)
                {
                        if((_inp(ba+1) & 0x80)==0) //PIN 11; busy einlesen, Achtung invertiert, wenn hi
dann um passendes Bit höherzählen
                                wert += bitwert;
```

```
                }

                bitwert /= 2; // nächstniederes Bit rbereitstellen
                warten(1e-6,TimeUnit);

                portbuffer=portbuffer & 223; //bit 1 lo, pin 2 lo; sclk auf lo
                _outp(ba,portbuffer);
                warten(1e-6,TimeUnit);
        }

        if(messkontrolle.typ==KANAL8)
        {
                portbuffer=(portbuffer & 191); //bit 7 lo, pin 8 lo; CS auf lo ds1305 Datenübertragung
Ende
                _outp(ba,portbuffer);
        }

        else if(messkontrolle.typ==KANAL30)
        {
                portbufferbi=(portbufferbi & 0xFD)+2; //PIN 14, BIT 2,Achtung invertiert,  CE auf lo
ds1305 Datenübertragung Ende
                _outp(ba+2,portbufferbi);
        }
        warten(1e-6,TimeUnit);

        *data=wert;
}


void warten(double sek,double TimeUnit) // wartet sec Sekunden, Timeunit gibt die Zeit pro tick an
{
LARGE_INTEGER startzeit,zeit;

        QueryPerformanceCounter(&startzeit);

        do
        {
                QueryPerformanceCounter(&zeit);
        }
        while( startzeit.QuadPart+sek/TimeUnit > zeit.QuadPart);

}


LARGE_INTEGER TimeInit(double * TimeUnit) //gibt die Zeit eines Ticks des hires counters in s zurück
{
        LARGE_INTEGER f;

        if(!QueryPerformanceFrequency(&f))
        {
                f.QuadPart=-1; return f; // TimeUnit=-1, wenn kein hires counter vorhanden ist, führt
dazu, dass die warte() sofort abgebrochen wird
                                                                //und somit nicht wartet, daher
kein warten auf maschinen, die keinen hires counter haben, mit Glück läuft dann das
                                                                //Programm doch.
        }

        *TimeUnit =(double) 1/f.QuadPart;
        return f;
}
```

11.7.2.9 grafik.cpp

```
/*Funktionen für die graphische Darstellung*/

#include "StdAfx.h"

/*-- Globale Variablen --*/
extern HINSTANCE hInst;
extern DATENSATZ * messdaten;
extern DATENSATZ30 * messdaten30;
extern REIHENSTEUERSTRUCT messkontrolle;
extern KOORDPOS auswertpunkte, einzelpunkt;
extern ZOOMPOS zoompos;
extern char auswertnam[MAX_PATH];

MESSSTEUERSTRUCT defmessparam;

void fensterloeschen(HDC hPaint,int maxwinx,int maxwiny)
{
        SelectObject(hPaint, GetStockObject(NULL_PEN));
   SelectObject(hPaint, GetStockObject(WHITE_BRUSH));
   Rectangle(hPaint,0,0,maxwinx,maxwiny);
}

void DrawBitmap( HDC hdc, HBITMAP bitmap, short x, short y )
{
BITMAP bitmapbuff;
HDC memorydc;
POINT origin;
POINT size;

   memorydc = CreateCompatibleDC( hdc );
   SelectObject( memorydc, bitmap );
   SetMapMode( memorydc, GetMapMode( hdc ) );
   GetObject( bitmap, sizeof( BITMAP ), (LPSTR) &bitmapbuff );

   origin.x = x;
   origin.y = y;
   size.x = bitmapbuff.bmWidth;
   size.y = bitmapbuff.bmHeight;

   DPtoLP( hdc, &origin, 1 );
   DPtoLP( memorydc, &size, 1 );

   BitBlt( hdc, origin.x, origin.y, size.x, size.y, memorydc, 0, 0, SRCCOPY);
   DeleteDC( memorydc );
}

double findmaxdouble(double *feld,unsigned long int count) // findet größte Fließkommazahl in einem Feld
{
double dummy;
unsigned long int i;

        dummy=feld[0];
        for(i=0;i<count;i++)
        {
                if(fabs(feld[i])>=dummy)
                        dummy=fabs(feld[i]);
        }
        return(dummy);
}

void drawstatus(HWND hWnd, int flag) // zeichnet Statusfenster
```

```
{
HDC hDc;
PAINTSTRUCT ps;

int maxwinx,maxwiny,i,outcount=0;
RECT Rect;
long int j=0,k=0;
char outtext[50];
HPEN hStift[8];
COLORREF FARBENDESTEXTES;

        GetClientRect(hWnd, &Rect);// wie groß ist das Fenster?
        InvalidateRect(hWnd,&Rect,0); // alles neu zeichnen
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;

   if(flag==PAINTMSG)
        {
                hDc=BeginPaint(hWnd, &ps);
                fensterloeschen(hDc,maxwinx,maxwiny);
        }
   else
        {
      hDc=GetDC(hWnd);
                fensterloeschen(hDc,maxwinx,maxwiny);
   }
        fensterloeschen(hDc,maxwinx,maxwiny);

        if(messkontrolle.messtyp!=NIXMESS) // zeichnen natürlich sinnvoll, wenn gemessen wurde
        {
                hStift[0]=CreatePen(PS_SOLID,0,FARBE1); // einzelne Stifte herrichten
                hStift[1]=CreatePen(PS_SOLID,0,FARBE2);
                hStift[2]=CreatePen(PS_SOLID,0,FARBE3);
                hStift[3]=CreatePen(PS_SOLID,0,FARBE4);
                hStift[4]=CreatePen(PS_SOLID,0,FARBE5);
                hStift[5]=CreatePen(PS_SOLID,0,FARBE6);
                hStift[6]=CreatePen(PS_SOLID,0,FARBE7);
                hStift[7]=CreatePen(PS_SOLID,0,FARBE8);

                SetTextAlign(hDc,TA_LEFT|TA_TOP);                // Cursor soll von TextOut
aktualisiert werden: siehe Visual C++ 6.0 Seite 154
                switch(messkontrolle.messtyp) // Spannungen der Kanäle werden herausgeschrieben
                {
                        case KANAL30TU: sprintf(outtext," U/V ");

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i: %.3lf V ", 4*i+1,0);
                                        }
                                        else
                                                sprintf(outtext,"%2i:          %.3lf          V          ",
4*i+1,messkontrolle.letztewerte[4*i]);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
```

```
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i: %.3lf V ", 4*i+2,0);
                                        }
                                        else
                                                sprintf(outtext,"%2i:           %.3lf           V           ",
4*i+2,messkontrolle.letztewerte[4*i+1]);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*2,outtext,strlen(outtext));
                                        }

                                        for(i=0;i<8;i++)
                                        {
                                                SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                                if((messkontrolle.anzahl)==0)
                                                {
                                                        sprintf(outtext,"%2i: %.3lf V ", 4*i+3,0);
                                                }
                                                else
                                                        sprintf(outtext,"%2i:           %.3lf           V           ",
4*i+3,messkontrolle.letztewerte[4*i+2]);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*3,outtext,strlen(outtext));
                                        }

                                        for(i=0;i<8;i++)
                                        {
                                                SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                                if((messkontrolle.anzahl)==0)
                                                {
                                                        sprintf(outtext,"%2i: %.3lf V ", 4*i+4,0);
                                                }
                                                else
                                                        sprintf(outtext,"%2i:           %.3lf           V           ",
4*i+4,messkontrolle.letztewerte[4*i+3]);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*4,outtext,strlen(outtext));
                                        }
                                break;

                                case KANAL30TT: sprintf(outtext," °C ");

                                        for(i=0;i<8;i++)
                                        {
                                                SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                                if(messkontrolle.anzahl==0)
                                                {
                                                        sprintf(outtext,"%2i:   - - ", 4*i+1);
                                                }
                                                else if(messkontrolle.letztewerte[4*i] != -273.15)
                                                {
                                                        sprintf(outtext,"%2i:           %.2lf           °C          ",
4*i+1,messkontrolle.letztewerte[4*i]);
                                                }
                                                else
                                                        sprintf(outtext,"%2i:   - - ", 4*i+1);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK,outtext,strlen(outtext));
                                        }
```

```
                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+2);
                                        }
                                        else if(messkontrolle.letztewerte[4*i+1] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:          %.2lf          °C          ",
4*i+2,messkontrolle.letztewerte[4*i+1]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+2);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*2,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+3);
                                        }
                                        else if(messkontrolle.letztewerte[4*i+2] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:          %.2lf          °C          ",
4*i+3,messkontrolle.letztewerte[4*i+2]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+3);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*3,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+4);
                                        }
                                        else if(messkontrolle.letztewerte[4*i+3] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:          %.2lf          °C          ",
4*i+4,messkontrolle.letztewerte[4*i+3]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+4);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*4,outtext,strlen(outtext));
                                }
                        break;

                        case KANAL30TTLF:
                                // Temperturen ausgeben

                                sprintf(outtext," °C ");
```

```
                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if(messkontrolle.anzahl==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+1);
                                        }
                                        else if(messkontrolle.letztewerte[4*i] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:          %.2lf          °C          ",
4*i+1,messkontrolle.letztewerte[4*i]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+1);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+2);
                                        }
                                        else if(messkontrolle.letztewerte[4*i+1] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:          %.2lf          °C          ",
4*i+2,messkontrolle.letztewerte[ 4*i+1]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+2);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*2,outtext,strlen(outtext));
                                }


                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+3);
                                        }
                                        else if(messkontrolle.letztewerte[4*i+2] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:          %.2lf          °C          ",
4*i+3,messkontrolle.letztewerte[4*i+2]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+3);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*3,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
```

```
                                        sprintf(outtext,"%2i:  - - ", 4*i+4);
                                }
                                else if(messkontrolle.letztewerte[4*i+3] != -273.15)
                                {
                                        sprintf(outtext,"%2i:  %.2lf  °C  ",
4*i+4,messkontrolle.letztewerte[4*i+3]);
                                }
                                else
                                        sprintf(outtext,"%2i:  - - ", 4*i+4);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*4,outtext,strlen(outtext));
                                }

                                // Leitfähigkeiten ausgeben
                                sprintf(outtext," mS ");
                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if(messkontrolle.anzahl==0)
                                        {
                                                sprintf(outtext,"%2i:  - - ", 4*i+1);
                                        }
                                        else if(messkontrolle.letztewerte[32+4*i] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:  %.3lf  mS  ",
4*i+1,messkontrolle.letztewerte[32+4*i]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:  - - ", 4*i+1);


        TextOut(hDc,LABTEXT+(i)*TEXTDIFF,OABK*5,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:  - - ", 4*i+2);
                                        }
                                        else if(messkontrolle.letztewerte[32+4*i+1] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:  %.3lf  mS  ",
4*i+2,messkontrolle.letztewerte[32+ 4*i+1]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:  - - ", 4*i+2);


        TextOut(hDc,LABTEXT+(i)*TEXTDIFF,OABK*6,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:  - - ", 4*i+3);
                                        }
                                        else if(messkontrolle.letztewerte[32+4*i+2] != -273.15)
                                        {
```

```
                                        sprintf(outtext,"%2i:   %.3lf   mS   ",
4*i+3,messkontrolle.letztewerte[32+4*i+2]);
                                }
                                else
                                        sprintf(outtext,"%2i:   - - ", 4*i+3);


        TextOut(hDc,LABTEXT+(i)*TEXTDIFF,OABK*7,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+4);
                                        }
                                        else if(messkontrolle.letztewerte[32+4*i+3] != -273.15)
                                        {
                                                sprintf(outtext,"%2i:   %.3lf   mS   ",
4*i+4,messkontrolle.letztewerte[32+4*i+3]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+4);


        TextOut(hDc,LABTEXT+(i)*TEXTDIFF,OABK*8,outtext,strlen(outtext));
                                }
                        break;

                        case KANAL30TTLFKAL:
                                // Temperturen ausgeben

                                sprintf(outtext," V ");
                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if(messkontrolle.anzahl==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+1);
                                        }
                                        else if(messkontrolle.letztewerte[4*i] != 0)
                                        {
                                                sprintf(outtext,"%2i:   %.2lf   V   ",
4*i+1,messkontrolle.letztewerte[4*i]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+1);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+2);
                                        }
                                        else if(messkontrolle.letztewerte[4*i+1] != 0)
                                        {
```

```
                                        sprintf(outtext,"%2i:   %.2lf   V   ",
4*i+2,messkontrolle.letztewerte[ 4*i+1]);
                                }
                                else
                                        sprintf(outtext,"%2i:   - - ", 4*i+2);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*2,outtext,strlen(outtext));
                                }


                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+3);
                                        }
                                        else if(messkontrolle.letztewerte[4*i+2] != 0)
                                        {
                                                sprintf(outtext,"%2i:   %.2lf   V   ",
4*i+3,messkontrolle.letztewerte[4*i+2]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+3);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*3,outtext,strlen(outtext));
                                }

                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if((messkontrolle.anzahl)==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+4);
                                        }
                                        else if(messkontrolle.letztewerte[4*i+3] != 0)
                                        {
                                                sprintf(outtext,"%2i:   %.2lf   V   ",
4*i+4,messkontrolle.letztewerte[4*i+3]);
                                        }
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+4);


        TextOut(hDc,LABTEXT+i*TEXTDIFF,OABK*4,outtext,strlen(outtext));
                                }

                                // Leitfähigkeiten ausgeben
                                sprintf(outtext," mS ");
                                for(i=0;i<8;i++)
                                {
                                        SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                        if(messkontrolle.anzahl==0)
                                        {
                                                sprintf(outtext,"%2i:   - - ", 4*i+1);
                                        }
                                        else if(messkontrolle.letztewerte[32+4*i] != 0)
                                        {
                                                sprintf(outtext,"%2i:   %.3lf   V   ",
4*i+1,messkontrolle.letztewerte[32+4*i]);
                                        }
```

```
                                        else
                                                sprintf(outtext,"%2i:   - - ", 4*i+1);


        TextOut(hDc,LABTEXT+(i)*TEXTDIFF,OABK*5,outtext,strlen(outtext));
                                        }

                                        for(i=0;i<8;i++)
                                        {
                                                SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                                if((messkontrolle.anzahl)==0)
                                                {
                                                        sprintf(outtext,"%2i:   - - ", 4*i+2);
                                                }
                                                else if(messkontrolle.letztewerte[32+4*i+1] != 0)
                                                {
                                                        sprintf(outtext,"%2i:           %.3lf           V           ",
4*i+2,messkontrolle.letztewerte[32+ 4*i+1]);
                                                }
                                                else
                                                        sprintf(outtext,"%2i:   - - ", 4*i+2);


        TextOut(hDc,LABTEXT+(i)*TEXTDIFF,OABK*6,outtext,strlen(outtext));
                                        }

                                        for(i=0;i<8;i++)
                                        {
                                                SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                                if((messkontrolle.anzahl)==0)
                                                {
                                                        sprintf(outtext,"%2i:   - - ", 4*i+3);
                                                }
                                                else if(messkontrolle.letztewerte[32+4*i+2] != 0)
                                                {
                                                        sprintf(outtext,"%2i:           %.3lf           V           ",
4*i+3,messkontrolle.letztewerte[32+4*i+2]);
                                                }
                                                else
                                                        sprintf(outtext,"%2i:   - - ", 4*i+3);


        TextOut(hDc,LABTEXT+(i)*TEXTDIFF,OABK*7,outtext,strlen(outtext));
                                        }

                                        for(i=0;i<8;i++)
                                        {
                                                SetTextColor(hDc,textfarben[i]); // Farbe auswählen
                                                if((messkontrolle.anzahl)==0)
                                                {
                                                        sprintf(outtext,"%2i:   - - ", 4*i+4);
                                                }
                                                else if(messkontrolle.letztewerte[32+4*i+3] != 0)
                                                {
                                                        sprintf(outtext,"%2i:           %.3lf           V           ",
4*i+4,messkontrolle.letztewerte[32+4*i+3]);
                                                }
                                                else
                                                        sprintf(outtext,"%2i:   - - ", 4*i+4);


        TextOut(hDc,LABTEXT+(i)*TEXTDIFF,OABK*8,outtext,strlen(outtext));
                                        }
```

```
                        break;
                }

                SelectObject(hDc, GetStockObject(BLACK_PEN));

                if(flag==PAINTMSG) // Datenausgeben
                {
                        SelectObject(hDc, GetStockObject(BLACK_PEN));
                        for(j=0;j<8;j++) // Stifte wieder aufräumen
                                DeleteObject(hStift[j]);

                        EndPaint(hWnd, &ps);
                }
                else
                {
                        for(j=0;j<8;j++) // Stifte wieder aufräumen
                                DeleteObject(hStift[j]);
                        ReleaseDC(hWnd,hDc);
                }
        }
        else
        {
                if(flag==PAINTMSG)
                  EndPaint(hWnd, &ps);
                else
                {
                        ReleaseDC(hWnd,hDc);
                }
        }

}

/*****************************************************************************************/
/*Funktion zum Zeichnen vom Kosy und Ausgabe der Datenpunkte
                        */
/*mit dem Flag wird eingestellt, ob auf dem maximalen y-Bereich oder auf den      */
/* maximalen gemessenen y skaliert wird.
                                        */
/*****************************************************************************************/
void drawkosy(HWND hWnd, int flag)
{
HDC hDc;
PAINTSTRUCT ps;

long int maxwinx,maxwiny,startx,starty,dx,dy,i;
RECT Rect;
HBITMAP hUp,hRe;
double du,di,xwertmax,ywertmax,ywertmin,xwertmin;
long int j=0,k=0;
char outtext[50];
HPEN hStift[8];
long int auswertcount=0;
double dummy=0,geradex=0;
COLORREF FARBENDESTEXTES;
long int start=0, ende=0;

        GetClientRect(hWnd, &Rect);
        InvalidateRect(hWnd,&Rect,0); // alles neu zeichnen
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;

  if(flag==PAINTMSG)
        {
```

```
    hDc=BeginPaint(hWnd, &ps);
        }
  else
        {
    hDc=GetDC(hWnd);
        }
        fensterloeschen(hDc,maxwinx,maxwiny);
        if((messkontrolle.messtyp!=KAPPATTAUSWERT) && (messkontrolle.messtyp!=TTAUSWERT)
&& (messkontrolle.messtyp!=KAPPATAUSWERT) && (messkontrolle.messtyp!=NIXMESS) &&
(messkontrolle.messtyp!=KANAL30TU)&& (messkontrolle.messtyp!=KANAL30TT)
&&(messkontrolle.messtyp!=KANAL30TTLF)) // Zeichnen natürlich sinnvoll, wenn gemessen wurde
        {
                hStift[0]=CreatePen(PS_SOLID,0,FARBE1); // einzelne Stifte herrichten
                hStift[1]=CreatePen(PS_SOLID,0,FARBE2);
                hStift[2]=CreatePen(PS_SOLID,0,FARBE3);
                hStift[3]=CreatePen(PS_SOLID,0,FARBE4);
                hStift[4]=CreatePen(PS_SOLID,0,FARBE5);
                hStift[5]=CreatePen(PS_SOLID,0,FARBE6);
                hStift[6]=CreatePen(PS_SOLID,0,FARBE7);
                hStift[7]=CreatePen(PS_SOLID,0,FARBE8);

                // Bestimmung der maximalen Koordinatenwerte

                if( (flag==NOPAINTMSGAUSWERTUNG || flag==PAINTMSG) &&
(messkontrolle.anzahl>1) ) // Darstellung skaliert, wenn Daten vorhanden
                {
                        xwertmax=messkontrolle.messdauer;//maximaler Bereich für x-achse in
Sekunden
                        ywertmin=messkontrolle.minmalerywert;
                        ywertmax=messkontrolle.maximalerywert;
                }
                else    // Darstellung auf maximalen Messbereich skaliert
                {
                        xwertmax=messkontrolle.messdauer;//maximaler Bereich für x-achse in
Sekunden
                        ywertmin=messkontrolle.minmalerywert;
                        ywertmax=messkontrolle.maximalerywert;
                }

                // Zeichnen der Linien des kosy
                starty=(int) round((maxwiny-UAB-OAB)+OAB);
                startx=(int) round(LAB);

                hUp=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_UPFEIL));
                hRe=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_LPFEIL));
                SelectObject(hDc, GetStockObject(NULL_PEN));
                SelectObject(hDc, GetStockObject(BLACK_PEN));

                MoveToEx(hDc,0+LAB-BALKENK-1,starty,NULL); // X-Achse
                LineTo(hDc,maxwinx-RAB+20,starty);
                DrawBitmap(hDc,hRe,maxwinx-RAB+20,starty-4); //Pfeil zeichnen

                MoveToEx(hDc,0+LAB,(int) round(maxwiny-UAB+4),NULL); // Y-Achse
                LineTo(hDc,0+LAB,(int) round(0+OAB-14));
                DrawBitmap(hDc,hUp,0+LAB-4,0+OAB-14); //Pfeil zeichnen

                // Zeichnen der Tickmarks

                dy=(int) round(maxwiny-OAB-UAB)/ANZAHLYTICKS; //Berechnen der Abstände für
kleine Striche
                di=(ywertmax-ywertmin)/ANZAHLYTICKS; // Differenz in y in der aufgetragenen
Einheit
```

```
		dx=(int) round((maxwinx-RAB-startx)/ANZAHLXTICKS);
		du=(xwertmax)/ANZAHLXTICKS;// Differenz in x in der aufgetragenen Einheit

		SetTextAlign(hDc,TA_LEFT|TA_TOP); // Beschriftung der Y-Achse setzen
		switch(messkontrolle.messtyp)
		{
			case TEST1KANAL: sprintf(outtext," U/V ");
			break;

			case KANAL8UT: sprintf(outtext," U/V ");
			break;

			case KANAL8TT: sprintf(outtext," T/°C ");
			break;

			case ASLTt: sprintf(outtext," T/°C ");
			break;

			case KANAL2UT: sprintf(outtext," U/V ");
			break;

			case WTWGT: sprintf(outtext," G/mS ");
			break;

			break;
		}
		TextOut(hDc,LAB+10,maxwiny-UAB-(ANZAHLYTICKS*dy)-
TEXTYDY,outtext,strlen(outtext));

		SetTextAlign(hDc,TA_CENTER|TA_BOTTOM);
		sprintf(outtext," t/s ");
		TextOut(hDc,startx+dx*ANZAHLXTICKS,starty-TEXTXDY,outtext,strlen(outtext));

		SetTextAlign(hDc,TA_LEFT|TA_TOP);
		for(i=0;i<=ANZAHLYTICKS;i++) // Y-Achse
		{
			MoveToEx(hDc,LAB-BALKENK,(int)		(starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UAB-OAB)),NULL);
			LineTo(hDc,LAB+BALKENK+1,(int)		(starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UAB-OAB)));

			switch(messkontrolle.messtyp)
			{
				case TEST1KANAL: sprintf(outtext,"% .3lf",(ywertmin+di*i));
				break;

				case KANAL8UT: sprintf(outtext,"% .3lf",(ywertmin+di*i));
				break;

				case KANAL2UT: sprintf(outtext,"% .3lf",(ywertmin+di*i));
				break;

				case KANAL8TT: sprintf(outtext,"%+.1lf",(ywertmin+di*i));
				break;

				case ASLTt: sprintf(outtext,"%+.1lf",(ywertmin+di*i));
				break;

				case WTWGT: sprintf(outtext,"%+.3lf",(ywertmin+di*i));
				break;
			}
```

```
                    TextOut(hDc,LAB+TEXTYDX,(int)                    (starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UAB-OAB))-TEXTYDY,outtext,strlen(outtext));
                }

                SetTextAlign(hDc,TA_CENTER|TA_TOP);

                for(i=1;i<=ANZAHLXTICKS;i++) // X-Achse
                {
                        MoveToEx(hDc,startx+i*dx,starty-BALKENK,NULL);
                        LineTo(hDc,startx+i*dx,starty+BALKENK+1);
                        sprintf(outtext,"%.0lf",0+i*du);
                        TextOut(hDc,startx+i*dx,starty+TEXTXDY,outtext,strlen(outtext));
                }

                SetTextAlign(hDc,TA_LEFT|TA_TOP);

                if(flag==PAINTMSG) // Daten ausgeben
                {
                        SelectObject(hDc, GetStockObject(BLACK_PEN));

                        switch(messkontrolle.messtyp)
                        {
                                case TEST1KANAL:
                                        for(j=1;j<messkontrolle.anzahl;j++) // Daten zeichnen
                                        {
                                                MoveToEx(hDc,(int)(startx+(messdaten[j-1].t-
messdaten[0].t)/(messdaten[messkontrolle.anzahl-1].t-messdaten[0].t)*(maxwinx-RAB-LAB)),(int)      (starty-
messdaten[j-1].ywerte[0]/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)),NULL);
                                                LineTo(hDc,(int)(startx+(messdaten[j].t-
messdaten[0].t)/(messdaten[messkontrolle.anzahl-1].t-messdaten[0].t)*(maxwinx-RAB-LAB)),(int)      (starty-
messdaten[j].ywerte[0]/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)));
                                        }
                                break;

                                case KANAL8TT:
                                        for(k=1;k<messkontrolle.anzahl;k++)
                                        {
                                                for(j=1;j<=8;j++)
                                                {
                                                        SelectObject(hDc,hStift[j]);
                                                        if(i>0)
                                                        {

        MoveToEx(hDc,(int)(startx+(messdaten[k-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int)      round(      (starty-(messdaten[k-
1].ywerte[j-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);

        LineTo(hDc,(int)(startx+(messdaten[k].t-messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-
RAB-LAB)),(int) round ((starty-(messdaten[k].ywerte[j-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-
OAB))));
                                                        }
                                                }
                                        }
                                break;


                                case KANAL8UT:
                                        for(k=1;k<messkontrolle.anzahl;k++)
                                        {
                                                for(j=1;j<=8;j++)
                                                {
                                                        SelectObject(hDc,hStift[j]);
                                                        if(i>0)
```

```
                                                {

        MoveToEx(hDc,(int)(startx+(messdaten[k-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[k-
1].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);

        LineTo(hDc,(int)(startx+(messdaten[k].t-messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-
RAB-LAB)),(int) round ((starty-(messdaten[k].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-
OAB))));
                                                }
                                        }
                                }
                        break;

                        case KANAL2UT:
                                for(k=1;k<messkontrolle.anzahl;k++)
                                {
                                        for(j=1;j<=2;j++)
                                        {
                                                SelectObject(hDc,hStift[j]);
                                                if(i>0)
                                                {

        MoveToEx(hDc,(int)(startx+(messdaten[k-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[k-
1].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);

        LineTo(hDc,(int)(startx+(messdaten[k].t-messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-
RAB-LAB)),(int) round ((starty-(messdaten[k].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-
OAB))));
                                                }
                                        }
                                }
                        break;

                        case ASLTt:
                                for(j=1;j<messkontrolle.anzahl;j++) // Daten zeichnen
                                {
                                        MoveToEx(hDc,(int)(startx+(messdaten[j-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[j-
1].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);
                                        LineTo(hDc,(int)(startx+(messdaten[j].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round ((starty-
(messdaten[j].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))));

                                }
                        break;

                        case WTWGT:
                                for(j=1;j<messkontrolle.anzahl;j++) // Daten zeichnen
                                {
                                        MoveToEx(hDc,(int)(startx+(messdaten[j-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[j-
1].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);
                                        LineTo(hDc,(int)(startx+(messdaten[j].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round ((starty-
(messdaten[j].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))));

                                }
                        break;


                }
```

```
                        SelectObject(hDc, GetStockObject(BLACK_PEN));

                        DeleteObject(hUp);
                        DeleteObject(hRe);

                        for(j=0;j<8;j++) // Stifte wieder aufräumen
                                DeleteObject(hStift[j]);

                        EndPaint(hWnd, &ps);
                }
                else
                {
                        DeleteObject(hUp);
                        DeleteObject(hRe);
                        for(j=0;j<8;j++) // Stifte wieder aufräumen
                                DeleteObject(hStift[j]);
                        ReleaseDC(hWnd,hDc);
                }
        }
        // bei 30 Kanal Messungen
        else if((messkontrolle.messtyp==KANAL30TU) || (messkontrolle.messtyp==KANAL30TT) ||
(messkontrolle.messtyp==KANAL30TTLF) || (messkontrolle.messtyp==KANAL30TTLFKAL) ) // bei
30 kanal ut
        {
                SetTextAlign(hDc,TA_LEFT|TA_TOP);               // Cursor soll von TextOut
aktualisiert werden: siehe Visual C++ 6.0 Seite 154

                sprintf(outtext,"Messdauer: %li",messkontrolle.messzeit);
                TextOut(hDc,LABTEXT/2,OABK/2,outtext,strlen(outtext));

                sprintf(outtext,"ASL: %.3lf °C",messkontrolle.letztewerteasl);
                TextOut(hDc,LABTEXT/2,OABK/2+20,outtext,strlen(outtext));

                sprintf(outtext,"Messung durch Drücken der Taste \"e\" abbrechen");
                TextOut(hDc,LABTEXT/2,OABK/2+40,outtext,strlen(outtext));

                if                      ((messkontrolle.messtyp==KANAL30TT)                     ||
(messkontrolle.messtyp==KANAL30TTLF))
                {
                        sprintf(outtext,"Gehäuse 1: %.2lf °C",messkontrolle.letztewerte[30]);
                        TextOut(hDc,LABTEXT+108,OABK/2,outtext,strlen(outtext));

                        sprintf(outtext,"Gehäuse 2: %.2lf °C",messkontrolle.letztewerte[31]);
                        TextOut(hDc,LABTEXT+108,OABK/2+20,outtext,strlen(outtext));
                }
                else
                {
                        sprintf(outtext,"Gehäusetemperatur              1:              %.2lf
°C",maketempstandard(messkontrolle.letztewerte[30]));
                        TextOut(hDc,LABTEXT+108,OABK/2,outtext,strlen(outtext));

                        sprintf(outtext,"Gehäusetemperatur              2:              %.2lf
°C",maketempstandard(messkontrolle.letztewerte[31]));
                        TextOut(hDc,LABTEXT+108,OABK/2+20,outtext,strlen(outtext));
                }
                //messdaten30[messkontrolle.count].t[31];
                EndPaint(hWnd, &ps);
        }
        else                    if((messkontrolle.messtyp==KAPPATAUSWERT)                       ||
(messkontrolle.messtyp==KAPPATTAUSWERT) || (messkontrolle.messtyp==TTAUSWERT)) // bei
Auswertung
        {
```

```
hStift[0]=CreatePen(PS_SOLID,0,FARBE1); // Einzelne Stifte herrichten
hStift[1]=CreatePen(PS_SOLID,0,FARBE2);
hStift[2]=CreatePen(PS_SOLID,0,FARBE3);
hStift[3]=CreatePen(PS_SOLID,0,FARBE4);
hStift[4]=CreatePen(PS_SOLID,0,FARBE5);
hStift[5]=CreatePen(PS_SOLID,0,FARBE6);
hStift[6]=CreatePen(PS_SOLID,0,FARBE7);
hStift[7]=CreatePen(PS_SOLID,0,FARBE8);


// Bestimmung der maximalen Koordinatenwerte

if(zoompos.status==UNGEZOOMED)
{
	start=0;
	ende=messkontrolle.anzahl;
}
else
{
	if(zoompos.count[0]<zoompos.count[1])
	{
		start=zoompos.count[0];
		ende=zoompos.count[1];
	}
	else
	{
		start=zoompos.count[1];
		ende=zoompos.count[0];
	}
}

xwertmax=messdaten30[start].ywerte[0];
for(auswertcount=start;auswertcount<ende;auswertcount++)
{
	if(xwertmax<messdaten30[auswertcount].ywerte[0])
		xwertmax=messdaten30[auswertcount].ywerte[0];
}

xwertmin=messdaten30[start].ywerte[0];
for(auswertcount=start;auswertcount<ende;auswertcount++)
{
	if(xwertmin>messdaten30[auswertcount].ywerte[0])
		xwertmin=messdaten30[auswertcount].ywerte[0];
}

ywertmax=messdaten30[start].ywerte[32];
for(auswertcount=start;auswertcount<ende;auswertcount++)
{
	if(ywertmax<messdaten30[auswertcount].ywerte[32])
		ywertmax=messdaten30[auswertcount].ywerte[32];
}

ywertmin=messdaten30[start].ywerte[32];
for(auswertcount=start;auswertcount<ende;auswertcount++)
{
	if(ywertmin>messdaten30[auswertcount].ywerte[32])
		ywertmin=messdaten30[auswertcount].ywerte[32];
}


// Zeichnen der Linien des Kosy
starty=(int) round((maxwiny-UAB-OAB)+OAB);
startx=(int) round(LAB);
```

```
		hUp=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_UPFEIL));
		hRe=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_LPFEIL));
		SelectObject(hDc, GetStockObject(NULL_PEN));
		SelectObject(hDc, GetStockObject(BLACK_PEN));

		MoveToEx(hDc,0+LAB-BALKENK-1,starty,NULL); // X-Achse
		LineTo(hDc,maxwinx-RAB+20,starty);
		DrawBitmap(hDc,hRe,maxwinx-RAB+20,starty-4); //Pfeil zeichnen

		MoveToEx(hDc,0+LAB,(int) round(maxwiny-UAB+4),NULL); // Y-Achse
		LineTo(hDc,0+LAB,(int) round(0+OAB-14));
		DrawBitmap(hDc,hUp,0+LAB-4,0+OAB-14); //Pfeil zeichnen

		// Zeichnen der Tickmarks

		dy=(int) round(maxwiny-OAB-UAB)/ANZAHLYTICKS; //Berechnen der Abstände für
kleine Striche
		di=(ywertmax-ywertmin)/ANZAHLYTICKS; // Differenz in y in der aufgetragenen
Einheit

		dx=(int) round((maxwinx-RAB-startx)/ANZAHLXTICKS);
		du=(xwertmax-xwertmin)/ANZAHLXTICKS;// Differenz in x in der aufgetragenen
Einheit

		SetTextAlign(hDc,TA_LEFT|TA_TOP); // Beschriftung der Y-Achse setzen

		if(messkontrolle.messtyp==KAPPATAUSWERT)
			sprintf(outtext," ln(G / mS) ");
		else if(messkontrolle.messtyp==TTAUSWERT)
			sprintf(outtext," T / °C ");
		else if(messkontrolle.messtyp==KAPPATTAUSWERT)
			sprintf(outtext," G / mS ");


		TextOut(hDc,LAB+10,maxwiny-UAB-(ANZAHLYTICKS*dy)-
TEXTYDY,outtext,strlen(outtext));

		SetTextAlign(hDc,TA_CENTER|TA_BOTTOM);

		if(messkontrolle.messtyp==KAPPATAUSWERT)
			sprintf(outtext," 1 / T * 1000 / (1/K) ");
		else
			sprintf(outtext," t / s ");


		TextOut(hDc,startx+dx*ANZAHLXTICKS,starty-TEXTXDY,outtext,strlen(outtext));

		// Ticks zeichnen
		SetTextAlign(hDc,TA_LEFT|TA_TOP);
		for(i=0;i<=ANZAHLYTICKS;i++) // Y-Achse
		{
			MoveToEx(hDc,LAB-BALKENK,(int)		(starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UAB-OAB)),NULL);
			LineTo(hDc,LAB+BALKENK+1,(int)		(starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UAB-OAB)));
	if(messkontrolle.messtyp==KAPPATAUSWERT) // Zahl der Stellen berücksichtigen
				sprintf(outtext,"%.3lf",(ywertmin+di*i));
			else
				sprintf(outtext,"%.3lf",(ywertmin+di*i));
			TextOut(hDc,LAB+TEXTYDX,(int)		(starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UAB-OAB))-TEXTYDY,outtext,strlen(outtext));
		}
```

```
		SetTextAlign(hDc,TA_CENTER|TA_TOP);

		for(i=1;i<=ANZAHLXTICKS;i++) // X-Achse
		{
			MoveToEx(hDc,startx+i*dx,starty-BALKENK,NULL);
			LineTo(hDc,startx+i*dx,starty+BALKENK+1);
	if(messkontrolle.messtyp==KAPPATAUSWERT)
				sprintf(outtext,"%.3lf",(xwertmin+du*i)); // Zahl der Stellen
berücksichtigen
			else
				sprintf(outtext,"%.0lf",(xwertmin+du*i));

			TextOut(hDc,startx+i*dx,starty+TEXTXDY,outtext,strlen(outtext));
		}

		SetTextAlign(hDc,TA_LEFT|TA_TOP);

		if(flag==PAINTMSG) // Daten ausgeben
		{
			SelectObject(hDc, GetStockObject(BLACK_PEN));

			for(j=start+1;j<ende;j++) // Daten zeichnen
			{
				MoveToEx(hDc,(int)(startx+(messdaten30[j-1].ywerte[0]-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)),(int) (starty-(messdaten30[j-1].ywerte[32]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)),NULL);
				LineTo(hDc,(int)(startx+(messdaten30[j].ywerte[0]-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)),(int) (starty-(messdaten30[j].ywerte[32]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)));
			}

			// Punkte für Auswertung zeichnen

			//SelectObject(hDc,hStift[2]); // roter Stift
			for(j=1;j<=auswertpunkte.i;j++)
			{
				if(j<=2)
					SelectObject(hDc,hStift[5]); // blauer Stift
				else
					SelectObject(hDc,hStift[1]); // roter Stift

				MoveToEx(hDc,(int)(startx+(auswertpunkte.dutemp[j-1]-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)),(int) (starty-/*(auswertpunkte.lnkappa[j-1]-
ywertmin)/(ywertmax-ywertmin)**/(maxwiny-UAB-OAB)),NULL);
				LineTo(hDc,(int)(startx+(auswertpunkte.dutemp[j-1]-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)),(int) (starty-/*(auswertpunkte.lnkappa[j-1]-
ywertmin)/(ywertmax-ywertmin)*/0*(maxwiny-UAB-OAB)));
			}

			// Punkte für ZOOM zeichnen

			if(zoompos.i==1)
			{
				SelectObject(hDc,hStift[3]); // gelber Stift

				MoveToEx(hDc,(int)(startx+(zoompos.tempgelbstrich-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)),(int) (starty-/*(auswertpunkte.lnkappa[j-1]-
ywertmin)/(ywertmax-ywertmin)**/(maxwiny-UAB-OAB)),NULL);
				LineTo(hDc,(int)(startx+(zoompos.tempgelbstrich-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)),(int) (starty-/*(auswertpunkte.lnkappa[j-1]-
ywertmin)/(ywertmax-ywertmin)*/0*(maxwiny-UAB-OAB)));
			}
```

```
                        // Geraden für die Auswertung zeichnen

                        if(auswertpunkte.i<=4 && auswertpunkte.i>1) // erste Gerade zeichnen
                        {
                                SelectObject(hDc,hStift[5]); // blauer Stift

                                SetTextAlign(hDc,TA_LEFT|TA_TOP); // Ausgabe der Parameter der
ersten Geraden
                                SetTextColor(hDc,textfarben[5]); // blauer Text
                                sprintf(outtext," y= %.3lf %+.3lf K * 1/T , r=%.3lf ",
auswertpunkte.a[0],auswertpunkte.b[0],auswertpunkte.r[0]);
                                TextOut(hDc,LAB,maxwiny-UAB-(ANZAHLYTICKS*dy)-50-
TEXTYDY,outtext,strlen(outtext));
                                SetTextColor(hDc,textfarben[0]); // schwarzer Text

                                if(auswertpunkte.i!=4) // beim ersten Zeichnen nicht bis zum
Schnittpunkt
                                {
                                        geradex=auswertpunkte.dutemp[0];

                                        MoveToEx(hDc,(int)(startx+(geradex-xwertmin)/(xwertmax-
xwertmin)*(maxwinx-RAB-LAB)),(int)      (starty-((auswertpunkte.a[0]+auswertpunkte.b[0]*geradex)-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)),NULL);
                                        geradex=auswertpunkte.dutemp[1];

                                        LineTo(hDc,(int)(startx+(geradex-xwertmin)/(xwertmax-
xwertmin)*(maxwinx-RAB-LAB)),(int)      (starty-((auswertpunkte.a[0]+auswertpunkte.b[0]*geradex)-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)));
                                }
                                else
                                {
                                        if(fabs(auswertpunkte.dutemp[0]-
auswertpunkte.schnittx)>fabs(auswertpunkte.dutemp[1]-auswertpunkte.schnittx))
                                        {
                                                geradex=auswertpunkte.dutemp[0];

                                        }
                                        else
                                        {
                                                geradex=auswertpunkte.dutemp[1];

                                        }

                                        MoveToEx(hDc,(int)(startx+(geradex-xwertmin)/(xwertmax-
xwertmin)*(maxwinx-RAB-LAB)),(int)      (starty-((auswertpunkte.a[0]+auswertpunkte.b[0]*geradex)-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)),NULL);

                                        if((auswertpunkte.dutemp[0]<auswertpunkte.schnittx)      &&
(auswertpunkte.dutemp[1] <auswertpunkte.schnittx)) // über den Schnittpunkt richtig hinwegzeichnen
                                        {
                                                geradex=auswertpunkte.schnittx+(xwertmax-
xwertmin)/100*5;
                                        }
                                        else if((auswertpunkte.dutemp[0]>auswertpunkte.schnittx) &&
(auswertpunkte.dutemp[1] >auswertpunkte.schnittx))
                                        {
                                                geradex=auswertpunkte.schnittx-(xwertmax-
xwertmin)/100*5;
                                        }
                                        else
                                                geradex=auswertpunkte.schnittx;
```

```
                                        LineTo(hDc,(int)(startx+(geradex-xwertmin)/(xwertmax-
xwertmin)*(maxwinx-RAB-LAB)),(int)          (starty-((auswertpunkte.a[0]+auswertpunkte.b[0]*geradex)-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)));
                                }
                        }
                        if(auswertpunkte.i == 4) // zweite Gerade zeichnen
                        {

                                SetTextAlign(hDc,TA_LEFT|TA_TOP); // Ausgabe der Parameter der
ersten Geraden
                                SetTextColor(hDc,textfarben[1]); // roter Text
                                sprintf(outtext," y= %.3lf %+.3lf K * 1/T , r=%.3lf ",
auswertpunkte.a[1],auswertpunkte.b[1],auswertpunkte.r[1]);
                                TextOut(hDc,LAB,maxwiny-UAB-(ANZAHLYTICKS*dy)-30-
TEXTYDY,outtext,strlen(outtext));
                                SetTextColor(hDc,textfarben[0]); // schwarzer Text

                                SelectObject(hDc,hStift[1]); // roter Stift
                                if(fabs(auswertpunkte.dutemp[2]-
auswertpunkte.schnittx)>fabs(auswertpunkte.dutemp[3]-auswertpunkte.schnittx))
                                {
                                        geradex=auswertpunkte.dutemp[2];

                                }
                                else
                                {
                                        geradex=auswertpunkte.dutemp[3];

                                }
                                MoveToEx(hDc,(int)(startx+(geradex-xwertmin)/(xwertmax-
xwertmin)*(maxwinx-RAB-LAB)),(int)          (starty-((auswertpunkte.a[1]+auswertpunkte.b[1]*geradex)-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)),NULL);

                                if((auswertpunkte.dutemp[2]<auswertpunkte.schnittx)          &&
(auswertpunkte.dutemp[3] <auswertpunkte.schnittx)) // über den Schnittpunkt richtig hinwegzeichnen
                                {
                                        geradex=auswertpunkte.schnittx+(xwertmax-xwertmin)/100*5;
                                }
                                else    if((auswertpunkte.dutemp[2]>auswertpunkte.schnittx)     &&
(auswertpunkte.dutemp[3] >auswertpunkte.schnittx)) // über den Schnittpunkt richtig hinwegzeichnen
                                {
                                        geradex=auswertpunkte.schnittx-(xwertmax-xwertmin)/100*5;
                                }
                                else
                                        geradex=auswertpunkte.schnittx;//-(xwertmax-
xwertmin)/100*5;

                                LineTo(hDc,(int)(startx+(geradex-xwertmin)/(xwertmax-
xwertmin)*(maxwinx-RAB-LAB)),(int)          (starty-((auswertpunkte.a[1]+auswertpunkte.b[1]*geradex)-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)));

                                if(messkontrolle.messtyp==KAPPATAUSWERT)
                                        sprintf(outtext,"Schnittpunkt: x= %.3lf, y= %.3lf, T=%.3lf°C
",auswertpunkte.schnittx,auswertpunkte.a[1]+auswertpunkte.b[1]*auswertpunkte.schnittx,1/auswertpunkte.sc
hnittx*1000-273.15);
                                else if(messkontrolle.messtyp==TTAUSWERT)
                                        sprintf(outtext,"Schnittpunkt: t= %.0lf s,     T=%.3lf °C
",auswertpunkte.schnittx,auswertpunkte.a[1]+auswertpunkte.b[1]*auswertpunkte.schnittx);
                                else if(messkontrolle.messtyp==KAPPATTAUSWERT)
                                        sprintf(outtext,"Schnittpunkt: t= %.0lf s,     G=%.3lf mS
",auswertpunkte.schnittx,auswertpunkte.a[1]+auswertpunkte.b[1]*auswertpunkte.schnittx);
```

```
                                TextOut(hDc,LAB+250,maxwiny-UAB-(ANZAHLYTICKS*dy)-50-
TEXTYDY,outtext,strlen(outtext));

                        }

                        SetTextColor(hDc,textfarben[1]); // roter Text
                        TextOut(hDc,LAB+250,maxwiny-UAB-(ANZAHLYTICKS*dy)-30-
TEXTYDY,auswertnam,strlen(auswertnam));
                        SetTextColor(hDc,textfarben[0]); // schwarzer Text

                        // Einzelpunkte zeichnen
                        SelectObject(hDc,hStift[4]); // grüner Stift
                        SetTextColor(hDc,textfarben[4]); // grüner Text

                        for(j=1;j<=einzelpunkt.i;j++)
                        {
                                if(messkontrolle.messtyp==KAPPATAUSWERT)
                                        sprintf(outtext,"P %li x= %.3lf, y= %.3lf, T=%.3lf°C, ", j,
einzelpunkt.dutemp[j-1],einzelpunkt.lnkappa[j-1],1/einzelpunkt.dutemp[j-1]*1000-273.15);
                                else if(messkontrolle.messtyp==KAPPATTAUSWERT)
                                        sprintf(outtext,"P %li t= %.0lf s, G= %.3lf mS", j,
einzelpunkt.dutemp[j-1],einzelpunkt.lnkappa[j-1]);
                                else if(messkontrolle.messtyp==TTAUSWERT)
                                        sprintf(outtext,"P %li t= %.0lf s, T= %.3lf °C", j,
einzelpunkt.dutemp[j-1],einzelpunkt.lnkappa[j-1]);

                                if(j==1)
                                        TextOut(hDc,LAB+580,maxwiny-UAB-
(ANZAHLYTICKS*dy)-50-TEXTYDY,outtext,strlen(outtext));
                                else if(j==2)
                                        TextOut(hDc,LAB+850,maxwiny-UAB-
(ANZAHLYTICKS*dy)-50-TEXTYDY,outtext,strlen(outtext));
                                else if(j==3)
                                        TextOut(hDc,LAB+580,maxwiny-UAB-
(ANZAHLYTICKS*dy)-30-TEXTYDY,outtext,strlen(outtext));
                                else if(j==4)
                                        TextOut(hDc,LAB+850,maxwiny-UAB-
(ANZAHLYTICKS*dy)-30-TEXTYDY,outtext,strlen(outtext));

                                MoveToEx(hDc,(int)(startx+(einzelpunkt.dutemp[j-1]-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)-BALKENK),(int) (starty-(einzelpunkt.lnkappa[j-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)),NULL);
                                LineTo(hDc,(int)(startx+(einzelpunkt.dutemp[j-1]-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)+BALKENK+1), (int) (starty-
(einzelpunkt.lnkappa[j-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)));

                                MoveToEx(hDc,(int)(startx+(einzelpunkt.dutemp[j-1]-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)),(int) (starty-(einzelpunkt.lnkappa[j-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)-BALKENK),NULL);
                                LineTo(hDc,(int)(startx+(einzelpunkt.dutemp[j-1]-
xwertmin)/(xwertmax-xwertmin)*(maxwinx-RAB-LAB)), (int) (starty-(einzelpunkt.lnkappa[j-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)+BALKENK+1));

                        }
                        SetTextColor(hDc,textfarben[0]); // schwarzer Text
                        SelectObject(hDc, GetStockObject(BLACK_PEN));

                        DeleteObject(hUp);
                        DeleteObject(hRe);

                        for(j=0;j<8;j++) // Stifte wieder aufräumen
                                DeleteObject(hStift[j]);
```

```
                        EndPaint(hWnd, &ps);
                }
                else
                {
                        DeleteObject(hUp);
                        DeleteObject(hRe);
                        for(j=0;j<8;j++) // Stifte wieder aufräumen
                                DeleteObject(hStift[j]);
                        ReleaseDC(hWnd,hDc);
                }

        }
        else
        {
                if(flag==PAINTMSG)
                  EndPaint(hWnd, &ps);
                else
                {
                        ReleaseDC(hWnd,hDc);
                }
        }
}

void drawkosy30kanal(HWND hWnd, int flag,int kanal) // zeichnet Kosy für die kleinen Fenster beim 30-Kanalgerät
{
HDC hDc;
PAINTSTRUCT ps;

int maxwinx,maxwiny,startx,starty,dx,dy,i;
RECT Rect;
HBITMAP hUp,hRe;
double du,di,xwertmax,ywertmax,ywertmin;
long int j=0,k=0;
char outtext[50];
HPEN hStift[8];

        GetClientRect(hWnd, &Rect);// wie groß ist das Fenster?
        InvalidateRect(hWnd,&Rect,0); // alles neu zeichnen
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;

  if(flag==PAINTMSG)
        {
    hDc=BeginPaint(hWnd, &ps);
        }
  else
        {
    hDc=GetDC(hWnd);
  }
        fensterloeschen(hDc,maxwinx,maxwiny);

        if(messkontrolle.messtyp!=NIXMESS) // Zeichnen natürlich sinnvoll, wenn gemessen wurde
        {
                hStift[0]=CreatePen(PS_SOLID,0,FARBE1); // Einzelne Stifte herrichten
                hStift[1]=CreatePen(PS_SOLID,0,FARBE2);
                hStift[2]=CreatePen(PS_SOLID,0,FARBE3);
                hStift[3]=CreatePen(PS_SOLID,0,FARBE4);
                hStift[4]=CreatePen(PS_SOLID,0,FARBE5);
                hStift[5]=CreatePen(PS_SOLID,0,FARBE6);
                hStift[6]=CreatePen(PS_SOLID,0,FARBE7);
                hStift[7]=CreatePen(PS_SOLID,0,FARBE8);
```

```
// Bestimmung der maximalen Koordinatenwerte

xwertmax=messkontrolle.messdauer;//maximaler Bereich für x-achse in Sekunden

if(kanal<=4) // wenn Temperaturfenster
{
        ywertmin=messkontrolle.minmalerywert;
        ywertmax=messkontrolle.maximalerywert;
}
else // Leitfähigkeitsfenster
{
        ywertmin=messkontrolle.minmalerywertlf;
        ywertmax=messkontrolle.maximalerywertlf;
}

if(kanal==ASLPAINT)
{
        ywertmin=MINTEMPERATUR;
        ywertmax=MAXTEMPERATUR;
}

// Zeichnen der Linien des Kosy
starty=(int) round((maxwiny-UABK-OABK)+OABK);
startx=(int) round(LABK);

hUp=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_UPFEIL));
hRe=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_LPFEIL));
SelectObject(hDc, GetStockObject(NULL_PEN));
SelectObject(hDc, GetStockObject(BLACK_PEN));

MoveToEx(hDc,0+LABK-BALKENK-1,starty,NULL); // X-Achse
LineTo(hDc,maxwinx-RABK+0,starty);
DrawBitmap(hDc,hRe,maxwinx-RABK+0,starty-4); //Pfeil zeichnen

MoveToEx(hDc,0+LABK,(int) round(maxwiny-UABK+4),NULL); // Y-Achse
LineTo(hDc,0+LABK,(int) round(0+OABK-14));
DrawBitmap(hDc,hUp,0+LABK-4,0+OABK-14); //Pfeil zeichnen

// Zeichnen der Tickmarks

dy=(int) round(maxwiny-OABK-UABK)/ANZAHLYTICKS; //Berechnen der Abstände
für kleine Striche
di=(ywertmax-ywertmin)/ANZAHLYTICKS; // Differenz in y in der aufgetragenen
Einheit

dx=(int) round((maxwinx-RABK-startx)/ANZAHLXTICKS);
du=(xwertmax)/ANZAHLXTICKS;// Differenz in x in der aufgetragenen Einheit

SetTextAlign(hDc,TA_LEFT|TA_TOP); // Beschriftung der Y-Achse setzen
switch(messkontrolle.messtyp)
{
        case KANAL30TU: sprintf(outtext," U/V ");
        break;

        case KANAL30TT: sprintf(outtext," T/°C ");
        break;

        case KANAL30TTLF:
                if(kanal<=4) // Temperaturen
                        sprintf(outtext," T/°C ");
                else // Leitfähigkeiten
                        sprintf(outtext," G/mS ");
        break;
```

```
                    case KANAL30TTLFKAL:
                        if(kanal<=4) // Temperaturen
                            sprintf(outtext," U/V ");
                        else // Leitfähigkeiten
                            sprintf(outtext," U/V ");
                    break;

                }

                if(kanal==ASLPAINT)
                {
                    sprintf(outtext," T/°C ");
                }

                TextOut(hDc,LABK+10,maxwiny-UABK-(ANZAHLYTICKS*dy)-
TEXTYDY,outtext,strlen(outtext));

                SetTextAlign(hDc,TA_CENTER|TA_BOTTOM);
                sprintf(outtext," t/s ");
                TextOut(hDc,startx+dx*ANZAHLXTICKS,starty-TEXTXDY,outtext,strlen(outtext));

                SetTextAlign(hDc,TA_LEFT|TA_TOP);
                for(i=0;i<=ANZAHLYTICKS;i+=2) // Y-Achse
                {
                    MoveToEx(hDc,LABK-BALKENK,(int)          (starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK)),NULL);
                    LineTo(hDc,LABK+BALKENK+1,(int)          (starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK)));

                    switch(messkontrolle.messtyp) // Text Y-Achse
                    {
                        case KANAL30TU: sprintf(outtext,"% .1lf",(ywertmin+di*i));
                        break;

                        case KANAL30TT: sprintf(outtext,"% .1lf",(ywertmin+di*i));
                        break;

                        case KANAL30TTLF: sprintf(outtext,"% .1lf",(ywertmin+di*i));
                        break;
                    }
                    if(kanal==ASLPAINT)
                    {
                         sprintf(outtext,"% .1lf",(ywertmin+di*i));
                    }

                    TextOut(hDc,LABK+TEXTYDXK,(int)          (starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))-TEXTYDY,outtext,strlen(outtext));
                }

                SetTextAlign(hDc,TA_CENTER|TA_TOP);

                for(i=2;i<=ANZAHLXTICKS;i+=2) // X-Achse
                {
                    MoveToEx(hDc,startx+i*dx,starty-BALKENK,NULL);
                    LineTo(hDc,startx+i*dx,starty+BALKENK+1);
                    sprintf(outtext,"%.0lf",0+i*du);
                    TextOut(hDc,startx+i*dx,starty+TEXTXDY,outtext,strlen(outtext));
                }

                SetTextAlign(hDc,TA_LEFT|TA_TOP);

                if(flag==PAINTMSG) // Daten ausgeben
```

```
{
        SelectObject(hDc, GetStockObject(BLACK_PEN));

        if(kanal==ASLPAINT) // Darstellung des ASL-Fensters
        {
                for(k=1;k<messkontrolle.count;k++)
                {
                        SelectObject(hDc,hStift[0]);
                        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-
messdaten30[0].aslzeit)/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);
                        LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-
messdaten30[0].aslzeit)/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                }
        }
        else
        {
                switch(messkontrolle.messtyp)
                {
                        case KANAL30TU:
                                for(k=1;k<messkontrolle.count;k++)
                                {
                                        for(j=0;j<8;j++)
                                        {
                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-
1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messdaten30[k].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                        }
                                }

                                if(messkontrolle.count>0)
                                {
                                        for(j=0;j<8;j++)
                                        {
                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[0].t[4*j+kanal-1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messkontrolle.letztewerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                        }
                                }
                                break;

                                case KANAL30TT:
                                for(k=1;k<messkontrolle.count;k++)
                                {
                                        for(j=0;j<8;j++)
                                        {
                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
```

```
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-
1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messdaten30[k].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                }
                                        }

                                        if(messkontrolle.count>0)
                                        {
                                                for(j=0;j<8;j++)
                                                {
                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[0].t[4*j+kanal-1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messkontrolle.letztewerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                }
                                        }
                                break;

                                case KANAL30TTLF:
                                        if(kanal<=4) // Temperaturen
                                        {
                                                for(k=1;k<messkontrolle.count;k++)
                                                {
                                                        for(j=0;j<8;j++)
                                                        {
                                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-
1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messdaten30[k].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                        }
                                                }

                                                if(messkontrolle.count>0)
                                                {
                                                        for(j=0;j<8;j++)
                                                        {
                                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[0].t[4*j+kanal-1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[0].t[32+4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messkontrolle.letztewerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                        }
                                                }
```

```
                                        }
                                        else // Leitfähigkeiten
                                        {
                                                for(k=1;k<messkontrolle.count;k++)
                                                {
                                                        for(j=0;j<8;j++)
                                                        {
                                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int)  round(
(starty-(messdaten30[k-1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[32+4*j+kanal-1-4]-messdaten30[0].t[32+4*j+kanal-1-
4])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int)  round  ((starty-
(messdaten30[k].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                        }
                                                }

                                                if(messkontrolle.count>0)
                                                {
                                                        for(j=0;j<8;j++)
                                                        {
                                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int)  round(
(starty-(messdaten30[messkontrolle.count-1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int)  round
((starty-(messkontrolle.letztewerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                        }
                                                }
                                        }
                                break;

                                case KANAL30TTLFKAL:
                                        if(kanal<=4) // Temperaturen
                                        {
                                                for(k=1;k<messkontrolle.count;k++)
                                                {
                                                        for(j=0;j<8;j++)
                                                        {
                                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int)  round(  (starty-(messdaten30[k-
1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int)  round  ((starty-
(messdaten30[k].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                        }
                                                }

                                                if(messkontrolle.count>0)
                                                {
                                                        for(j=0;j<8;j++)
                                                        {
```

```
                                                                SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[0].t[4*j+kanal-1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[0].t[32+4*j+kanal-
1])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messkontrolle.letztewerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                }
                                                else // Leitfähigkeiten
                                                {
                                                        for(k=1;k<messkontrolle.count;k++)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round(
(starty-(messdaten30[k-1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[32+4*j+kanal-1-4]-messdaten30[0].t[32+4*j+kanal-1-
4])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messdaten30[k].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                        if(messkontrolle.count>0)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round(
(starty-(messdaten30[messkontrolle.count-1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(messkontrolle.messdauer)*(maxwinx-RABK-LABK)),(int) round
((starty-(messkontrolle.letztewerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                                }
                                                        }
                                                }
                                        break;
                                }
                        }

                        SelectObject(hDc, GetStockObject(BLACK_PEN));

                        DeleteObject(hUp);
                        DeleteObject(hRe);

                        for(j=0;j<8;j++) // Stifte wieder aufräumen
                                DeleteObject(hStift[j]);
```

```
                        EndPaint(hWnd, &ps);
                }
                else
                {
                        DeleteObject(hUp);
                        DeleteObject(hRe);
                        for(j=0;j<8;j++) // Stifte wieder aufräumen
                                DeleteObject(hStift[j]);
                        ReleaseDC(hWnd,hDc);
                }
        }
        else
        {
                if(flag==PAINTMSG) //
                  EndPaint(hWnd, &ps);
                else
                {
                        ReleaseDC(hWnd,hDc);
                }
        }

}

void drawkosyhires(HWND hWnd, int flag,int kanal) // Zeichnet Kosy für die HIRES FENSTER
{
HDC hDc;
PAINTSTRUCT ps;

int maxwinx,maxwiny,startx,starty,dx,dy,i,welchesfenster;
RECT Rect;
HBITMAP hUp,hRe;
double du,di,xwertmax,ywertmax,ywertmin,dummy=0,maxkanal[8];
long int j=0,k=0,l=0;
char outtext[50];
HPEN hStift[8];

        GetClientRect(hWnd, &Rect);// wie groß ist das Fenster?
        InvalidateRect(hWnd,&Rect,0); // alles neu zeichnen
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;

   if(flag==PAINTMSG)
        {
                hDc=BeginPaint(hWnd, &ps);
        }
   else
        {
      hDc=GetDC(hWnd);
   }
        fensterloeschen(hDc,maxwinx,maxwiny);

        if(messkontrolle.messtyp!=NIXMESS) // Zeichnen natürlich sinnvoll, wenn gemessen wurde
        {
                hStift[0]=CreatePen(PS_SOLID,0,FARBE1); // Einzelne Stifte herrichten
                hStift[1]=CreatePen(PS_SOLID,0,FARBE2);
                hStift[2]=CreatePen(PS_SOLID,0,FARBE3);
                hStift[3]=CreatePen(PS_SOLID,0,FARBE4);
                hStift[4]=CreatePen(PS_SOLID,0,FARBE5);
                hStift[5]=CreatePen(PS_SOLID,0,FARBE6);
                hStift[6]=CreatePen(PS_SOLID,0,FARBE7);
                hStift[7]=CreatePen(PS_SOLID,0,FARBE8);

                // Bestimmung der maximalen Koordinatenwerte
```

```
xwertmax=HIRESMESSDAUER;//maximaler Bereich für x-Achse in Sekunden

// Bestimmung des maximalen y-Werts
if(kanal==ASLPAINT)
{
	if(messkontrolle.letztewerteaslx<200) // sind bereits 200s gemessen?
	{
		// Bestimmung des Maximums eines Messkanals
		dummy=messdaten30[0].asl;
		for(j=0;j<messkontrolle.count;j++)
		{
			if(messdaten30[j].asl>=dummy)
				dummy=messdaten30[j].asl;
		}
		if(messkontrolle.letztewerteasl>=dummy)
			dummy=messkontrolle.letztewerteasl;
		ywertmax=dummy;

		// Bestimmung des Minimums eines Messkanals
		if(messdaten30[0].asl>-273.15)
			dummy=messdaten30[0].asl;

		for(j=0;j<messkontrolle.count;j++)
		{
			if((messdaten30[j].asl<=dummy) && (messdaten30[j].asl > MINTEMPERATUR))
				dummy=messdaten30[j].asl;
		}
		if((messkontrolle.letztewerteasl<=dummy) && (messkontrolle.letztewerteasl > MINTEMPERATUR))
			dummy=messkontrolle.letztewerteasl;
		ywertmin=dummy;
	}
	else
	{
		// Bestimmung des Bereichs, in dem die Daten liegen
		k=messkontrolle.count-1;
		do
		{
		}
		while((messkontrolle.letztewerteaslx-messdaten30[k--].aslzeit)<200);
		k++;

		// Bestimmung des Maximums eines Messkanals
		dummy=messdaten30[k].asl;
		for(j=k;j<messkontrolle.count;j++)
		{
			if(messdaten30[j].asl>=dummy)
				dummy=messdaten30[j].asl;
		}
		if(messkontrolle.letztewerteasl>=dummy)
			dummy=messkontrolle.letztewerteasl;

		ywertmax=dummy;

		// Bestimmung des Minimums eines Messkanals
		if(messdaten30[k].asl>-273.15)
			dummy=messdaten30[k].asl;
		for(j=k;j<messkontrolle.count;j++)
		{
			if((messdaten30[j].asl<=dummy) && (messdaten30[j].asl > MINTEMPERATUR))
```

```
                                        dummy=messdaten30[j].asl;
                                }
                                if((messkontrolle.letztewerteasl<=dummy)                    &&
(messkontrolle.letztewerteasl > MINTEMPERATUR))
                                        dummy=messkontrolle.letztewerteasl;
                                ywertmin=dummy;
                        }
                }
                else // daten der 4 ADCs, sind bereits 200s gemessen?
                {
                        if(kanal<=2 || kanal>4) // Innentemperatur des Geräts nicht im HIRES Fenster
zeichnen
                                welchesfenster=8; // alle Kanäle zeichnen
                        else
                                welchesfenster=7; // nur 1 bis 7 zeichnen; 8 ist Innentemperatur


                        if(messkontrolle.letztewertex[kanal-1]<200)
                        {
                                // Bestimmung des Maximums eines Messkanals
                                if(kanal<=4) // Thermometerdaten
                                {
                                        // Bestimmung des Maximums eines Messkanals
                                        for(i=0;i<welchesfenster;i++)
                                        {
                                                dummy=messdaten30[0].ywerte[4*i+kanal-1];
                                                for(j=0;j<messkontrolle.count;j++)
                                                {
                                                        if((messdaten30[j].ywerte[4*i+kanal-
1])>=dummy)

        dummy=(messdaten30[j].ywerte[4*i+kanal-1]);
                                                }
                                                if(messkontrolle.letztewerte[4*i+kanal-1]>=dummy)

        dummy=messkontrolle.letztewerte[4*i+kanal-1];
                                                maxkanal[i]=dummy;
                                        }
                                }
                                else  // Leitfähigkeitsdaten
                                {
                                        // Bestimmung des Maximums eines Messkanals
                                        for(i=0;i<welchesfenster;i++)
                                        {
                                                dummy=messdaten30[0].ywerte[32+4*i+kanal-1-4];
                                                for(j=0;j<messkontrolle.count;j++)
                                                {
                                                        if((messdaten30[j].ywerte[32+4*i+kanal-1-
4])>=dummy)

        dummy=(messdaten30[j].ywerte[32+4*i+kanal-1-4]);
                                                }
                                                if(messkontrolle.letztewerte[32+4*i+kanal-1-
4]>=dummy)

        dummy=messkontrolle.letztewerte[32+4*i+kanal-1-4];
                                                maxkanal[i]=dummy;
                                        }

                                }

                                // Bestimmung des Maximums der ADCs
                                dummy=maxkanal[0];
```

```
                                for(i=0;i<welchesfenster;i++)
                                {
                                        if((maxkanal[i])>=dummy)
                                                dummy=(maxkanal[i]);
                                }

                                if(messkontrolle.messtyp==KANAL30TT)    //    Temperaturbereich
überschritten?
                                {
                                        if(dummy>MAXTEMPERATUR )
                                                dummy=MAXTEMPERATUR;
                                        if(dummy<MINTEMPERATUR)
                                                dummy=MAXTEMPERATUR;
                                }
                                else            if(messkontrolle.messtyp==KANAL30TTLF)          //
Temperaturbereich überschritten?
                                {
                                        if(kanal<=4)
                                        {
                                                if(dummy>MAXTEMPERATUR )
                                                        dummy=MAXTEMPERATUR;
                                                if(dummy<MINTEMPERATUR)
                                                        dummy=MAXTEMPERATUR;  //
                                        }
                                        else
                                        {
                                                if(dummy>MAXLF )
                                                        dummy=MAXLF;
                                                if(dummy<MINLF)
                                                        dummy=MAXLF;
                                        }
                                }
                                // Spannungsbereiche können nicht überschritten werden
                                ywertmax=dummy;

                                // Bestimmung des Minimum eines Messkanals
                                if(kanal<=4) // Thermometerdaten
                                {
                                        for(i=0;i<welchesfenster;i++)
                                        {
                                                if((messkontrolle.messtyp==KANAL30TT)           ||
(messkontrolle.messtyp==KANAL30TTLF)) // Temperaturen
                                                {
                                                        if(messdaten30[0].ywerte[4*i+kanal-1]>-
273.15)

        dummy=messdaten30[0].ywerte[4*i+kanal-1];
                                                }
                                                else // Spannungen
                                                {
                                                        if(messdaten30[0].ywerte[4*i+kanal-1]>0)

        dummy=messdaten30[0].ywerte[4*i+kanal-1];
                                                }

                                                for(j=0;j<messkontrolle.count;j++)
                                                {

        if((messkontrolle.messtyp==KANAL30TT)||(messkontrolle.messtyp==KANAL30TTLF))        //
Temperaturen
                                                        {
```

```
        if(((messdaten30[j].ywerte[4*i+kanal-1])<=dummy)        &&        (messdaten30[j].ywerte[4*i+kanal-
1]>MINTEMPERATUR) )

        dummy=(messdaten30[j].ywerte[4*i+kanal-1]);
                                                        }
                                                        else // Spannungen
                                                        {

        if(((messdaten30[j].ywerte[4*i+kanal-1])<=dummy) && (messdaten30[j].ywerte[4*i+kanal-1]>0) )

        dummy=(messdaten30[j].ywerte[4*i+kanal-1]);
                                                        }
                                                }


        if((messkontrolle.messtyp==KANAL30TT)||(messkontrolle.messtyp==KANAL30TTLF))        //
Temperaturen
                                                {
                                                        if((messkontrolle.letztewerte[4*i+kanal-
1]<=dummy) &&  (messkontrolle.letztewerte[4*i+kanal-1]>MINTEMPERATUR))

        dummy=messkontrolle.letztewerte[4*i+kanal-1];
                                                }
                                                else // Spannungen
                                                {
                                                        if((messkontrolle.letztewerte[4*i+kanal-
1]<=dummy) &&  (messkontrolle.letztewerte[4*i+kanal-1]>0))

        dummy=messkontrolle.letztewerte[4*i+kanal-1];
                                                }

                                                        maxkanal[i]=dummy;
                                        }
                                }
                                else // Leitfähigkeitsdaten
                                {
                                        // Bestimmung des Minimum eines Messkanals
                                        for(i=0;i<welchesfenster;i++)
                                        {

                                                if(messdaten30[0].ywerte[32+4*i+kanal-1-4]>MINLF)

        dummy=messdaten30[0].ywerte[32+4*i+kanal-1-4];

                                                for(j=0;j<messkontrolle.count;j++)
                                                {
                                                        // Leitfähigkeiten
                                                        if(((messdaten30[j].ywerte[32+4*i+kanal-1-
4])<=dummy) && (messdaten30[j].ywerte[32+4*i+kanal-1-4]>MINLF) )

        dummy=(messdaten30[j].ywerte[32+4*i+kanal-1-4]);
                                                }

                                                 //  Leitfähigkeiten
                                                if((messkontrolle.letztewerte[32+4*i+kanal-1-
4]<=dummy) &&  (messkontrolle.letztewerte[32+4*i+kanal-1-4]>MINLF))

        dummy=messkontrolle.letztewerte[32+4*i+kanal-1-4];


                                                maxkanal[i]=dummy;
                                        }
```

```
                                }

                                // Bestimmung des Minimum der ADCs
                                dummy=maxkanal[0];
                                for(i=0;i<welchesfenster;i++)
                                {
                                        //Messkontrolle.flagkanal30[4*i+kanal-1]=;
                                        if((maxkanal[i])<=dummy)
                                                dummy=(maxkanal[i]);
                                }

                                if(messkontrolle.messtyp==KANAL30TT)    //    Temperaturbereich
überschritten?
                                {
                                        if(dummy<MINTEMPERATUR )
                                        {
                                                dummy=MINTEMPERATUR ;
                                        }
                                }
                                else if(messkontrolle.messtyp==KANAL30TTLF)
                                {
                                        if(kanal<=4)
                                        {
                                                if(dummy>MAXTEMPERATUR )
                                                        dummy=MAXTEMPERATUR;
                                                if(dummy<MINTEMPERATUR)
                                                        dummy=MAXTEMPERATUR;
                                        }
                                        else
                                        {
                                                if(dummy>MAXLF )
                                                        dummy=MAXLF;
                                                if(dummy<MINLF)
                                                        dummy=MAXLF;
                                        }
                                }

                                ywertmin=dummy;
                        }
                        else // nach 200s
                        {
                                // Bestimmung des Bereichs in dem die Daten liegen
                                k=messkontrolle.count-1;
                                do
                                {
                                }
                                while((messkontrolle.letztewertex[kanal-1]-messdaten30[k--].t[kanal-
1])<200);
                                k++;

                                // Bestimmung des Maximums eines Messkanals
                                for(i=0;i<welchesfenster;i++)
                                {
                                        if(kanal<=4) // Thermometerdaten
                                        {
                                                dummy=messdaten30[k].ywerte[4*i+kanal-1];
                                                for(j=k;j<messkontrolle.count;j++)
                                                {
                                                        if((messdaten30[j].ywerte[4*i+kanal-
1])>=dummy)

        dummy=(messdaten30[j].ywerte[4*i+kanal-1]);
                                                }
```

```
                                        if(messkontrolle.letztewerte[4*i+kanal-1]>=dummy)

        dummy=messkontrolle.letztewerte[4*i+kanal-1];
                                        maxkanal[i]=dummy;
                                }
                                else// Leitfähigkeitsdaten
                                {
                                        dummy=messdaten30[k].ywerte[32+4*i+kanal-1-4];
                                        for(j=k;j<messkontrolle.count;j++)
                                        {
                                                if((messdaten30[j].ywerte[32+4*i+kanal-1-
4])>=dummy)

        dummy=(messdaten30[j].ywerte[32+4*i+kanal-1-4]);
                                        }
                                        if(messkontrolle.letztewerte[32+4*i+kanal-1-
4]>=dummy)

        dummy=messkontrolle.letztewerte[32+4*i+kanal-1-4];
                                        maxkanal[i]=dummy;
                                }
                        }

                        // Bestimmung des Maximums des ADCs
                        dummy=maxkanal[0];
                        for(i=0;i<welchesfenster;i++)
                        {
                                if((maxkanal[i])>=dummy)
                                        dummy=(maxkanal[i]);
                        }

                        if(messkontrolle.messtyp==KANAL30TT)    //    Temperaturbereich
überschritten?
                        {
                                if(dummy>MAXTEMPERATUR )
                                {
                                        dummy=MAXTEMPERATUR ;
                                }
                                if(dummy<MINTEMPERATUR)
                                        dummy=MAXTEMPERATUR;
                        }
                        else if(messkontrolle.messtyp==KANAL30TTLF)
                        {
                                if(kanal<=4)
                                {
                                        if(dummy>MAXTEMPERATUR )
                                                dummy=MAXTEMPERATUR;
                                        if(dummy<MINTEMPERATUR)
                                                dummy=MAXTEMPERATUR;
                                }
                                else
                                {
                                        if(dummy>MAXLF )
                                                dummy=MAXLF;
                                        if(dummy<MINLF)
                                                dummy=MAXLF; //???????????
                                }
                        }// Spannungsbereiche können nicht überschritten werden

                        ywertmax=dummy;

                        // Bestimmung des Minimum eines Messkanals
                        if(kanal<=4) // Thermometerdaten
```

```
                                        {
                                                for(i=0;i<welchesfenster;i++)
                                                {
                                                        if((messkontrolle.messtyp==KANAL30TT)           ||
(messkontrolle.messtyp==KANAL30TTLF)) // Temperaturen
                                                        {
                                                                if(messdaten30[k].ywerte[4*i+kanal-1]>-
273.15)

        dummy=messdaten30[k].ywerte[4*i+kanal-1];
                                                        }
                                                        else // Leitfähigkeiten und Spannungen
                                                        {
                                                                if(messdaten30[k].ywerte[4*i+kanal-1]>0)
                                                                dummy=messdaten30[k].ywerte[4*i+kanal-
1];
                                                        }

                                                        for(j=k;j<messkontrolle.count;j++)
                                                        {
                                                                if(messkontrolle.messtyp==KANAL30TT)
                                                                {

        if(((messdaten30[j].ywerte[4*i+kanal-1])<=dummy)        &&        (messdaten30[j].ywerte[4*i+kanal-
1]>MINTEMPERATUR))

        dummy=(messdaten30[j].ywerte[4*i+kanal-1]);
                                                                }
                                                                else
                                                                {

        if(((messdaten30[j].ywerte[4*i+kanal-1])<=dummy) && (messdaten30[j].ywerte[4*i+kanal-1]>0))

        dummy=(messdaten30[j].ywerte[4*i+kanal-1]);
                                                                }
                                                        }

                                                        if((messkontrolle.messtyp==KANAL30TT)           ||
(messkontrolle.messtyp==KANAL30TTLF)) //Temperaturen
                                                        {
                                                                if((messkontrolle.letztewerte[4*i+kanal-
1]<=dummy) && (messkontrolle.letztewerte[4*i+kanal-1]>MINTEMPERATUR))

        dummy=messkontrolle.letztewerte[4*i+kanal-1];

                                                        }
                                                        else // Leitfähigkeiten  und Spannungen
                                                        {
                                                                if((messkontrolle.letztewerte[4*i+kanal-
1]<=dummy) && (messkontrolle.letztewerte[4*i+kanal-1]>0))

        dummy=messkontrolle.letztewerte[4*i+kanal-1];
                                                        }
                                                        maxkanal[i]=dummy;
                                                }
                                        }
                                        else // Leitfähigkeitsdaten
                                        {
                                                for(i=0;i<welchesfenster;i++)
                                                {
                                                        // Leitfähigkeiten
                                                        if(messdaten30[k].ywerte[32+4*i+kanal-1-4]>0)
```

```
dummy=messdaten30[k].ywerte[32+4*i+kanal-1-4];

                                                for(j=k;j<messkontrolle.count;j++)
                                                {
                                                        if(((messdaten30[j].ywerte[32+4*i+kanal-1-
4])<=dummy) && (messdaten30[j].ywerte[32+4*i+kanal-1-4]>0))

dummy=(messdaten30[j].ywerte[32+4*i+kanal-1-4]);
                                                }

                                                if((messkontrolle.letztewerte[32+4*i+kanal-1-
4]<=dummy) && (messkontrolle.letztewerte[32+4*i+kanal-1-4]>0))

dummy=messkontrolle.letztewerte[32+4*i+kanal-1-4];
                                                maxkanal[i]=dummy;
                                        }
                                }
                                // Bestimmung des Minimums des ADCs
                                dummy=maxkanal[0];
                                for(i=0;i<welchesfenster;i++)
                                {
                                        if((maxkanal[i])<=dummy)
                                                dummy=(maxkanal[i]);
                                }

                                if(messkontrolle.messtyp==KANAL30TT)    //    Temperaturbereich
überschritten?
                                {
                                        if(dummy<MINTEMPERATUR )
                                        {
                                                dummy=MINTEMPERATUR ;
                                        }
                                }
                                else if(messkontrolle.messtyp==KANAL30TTLF)
                                {
                                        if(kanal<=4)
                                        {
                                                if(dummy>MAXTEMPERATUR )
                                                        dummy=MAXTEMPERATUR;
                                                if(dummy<MINTEMPERATUR)
                                                        dummy=MAXTEMPERATUR;
                                        }
                                        else
                                        {
                                                if(dummy>MAXLF )
                                                        dummy=MAXLF;
                                                if(dummy<MINLF)
                                                        dummy=MAXLF;
                                        }
                                }

                                ywertmin=dummy;
                        }
                }

                //Pfeile und Linien des Kosy zeichnen
                starty=(int) round((maxwiny-UABK-OABK)+OABK);
                startx=(int) round(LABK);

                hUp=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_UPFEIL));
                hRe=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_LPFEIL));
                SelectObject(hDc, GetStockObject(NULL_PEN));
```

```
		SelectObject(hDc, GetStockObject(BLACK_PEN));

		MoveToEx(hDc,0+LABK-BALKENK-1,starty,NULL); // X-Achse
		LineTo(hDc,maxwinx-RABK+10,starty);
		DrawBitmap(hDc,hRe,maxwinx-RABK+10,starty-4); //Pfeil zeichnen

		MoveToEx(hDc,0+LABK,(int) round(maxwiny-UABK+4),NULL); // Y-Achse
		LineTo(hDc,0+LABK,(int) round(0+OABK-14));
		DrawBitmap(hDc,hUp,0+LABK-4,0+OABK-14); //Pfeil zeichnen

		// Zeichnen der Tickmarks

		dy=(int) round(maxwiny-OABK-UABK)/ANZAHLYTICKS; //Berechnen der Abstände
für kleine Striche
		di=(ywertmax-ywertmin)/ANZAHLYTICKS; // Differenz in y in der aufgetragenen
Einheit

		dx=(int) round((maxwinx-RABK-startx)/ANZAHLXTICKS);
		du=(xwertmax)/ANZAHLXTICKS;// Differenz in x in der aufgetragenen Einheit

		SetTextAlign(hDc,TA_LEFT|TA_TOP); // Beschriftung der Y-Achse setzen
		switch(messkontrolle.messtyp)
		{
			case KANAL30TU: sprintf(outtext," U/V ");
			break;

			case KANAL30TT: sprintf(outtext," T/°C");
			break;

			case KANAL30TTLF:
				if(kanal<=4)
					sprintf(outtext," T/°C");
				else
					sprintf(outtext," G/mS");
			break;

			case KANAL30TTLFKAL:
				if(kanal<=4)
					sprintf(outtext," U/V");
				else
					sprintf(outtext," U/V");
			break;
		}
		TextOut(hDc,LABK+10,maxwiny-UABK-(ANZAHLYTICKS*dy)-
TEXTYDY,outtext,strlen(outtext));

		SetTextAlign(hDc,TA_CENTER|TA_BOTTOM);
		sprintf(outtext," t/s ");
		TextOut(hDc,startx+dx*ANZAHLXTICKS,starty-TEXTXDY,outtext,strlen(outtext));

		SetTextAlign(hDc,TA_LEFT|TA_TOP);
		for(i=0;i<=ANZAHLYTICKS;i+=2) // Y-Achse
		{
			MoveToEx(hDc,LABK-BALKENK,(int)		(starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK)),NULL);
			LineTo(hDc,LABK+BALKENK+1,(int)		(starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK)));

			switch(messkontrolle.messtyp)
			{
				case KANAL30TU: sprintf(outtext,"% .2lf",(ywertmin+di*i)); // Werte
Y-Achse
				break;
```

```
                                case KANAL30TT: sprintf(outtext,"% .2lf",(ywertmin+di*i)); // Werte
Y-Achse
                                break;

                                case KANAL30TTLF: sprintf(outtext,"% .2lf",(ywertmin+di*i)); //
Werte Y-Achse
                                break;
                        }

                        TextOut(hDc,LABK+TEXTYDXK,(int) (starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))-TEXTYDY,outtext,strlen(outtext));
                }

                SetTextAlign(hDc,TA_CENTER|TA_TOP);

                for(i=5;i<=ANZAHLXTICKS;i+=5) // X-Achse
                {
                        MoveToEx(hDc,startx+i*dx,starty-BALKENK,NULL);
                        LineTo(hDc,startx+i*dx,starty+BALKENK+1);
                        sprintf(outtext,"%.0lf",0+i*du);
                        TextOut(hDc,startx+i*dx,starty+TEXTXDY,outtext,strlen(outtext));
                }

                SetTextAlign(hDc,TA_LEFT|TA_TOP);

                if(flag==PAINTMSG) // Daten ausgeben
                {
                        SelectObject(hDc, GetStockObject(BLACK_PEN));

                        switch(messkontrolle.messtyp)
                        {
                                case KANAL30TU:
                                        if(kanal==ASLPAINT) //ASL zeichnen
                                        {
                                                if(messkontrolle.letztewerteaslx<200)   //   wurden
bereits 200s gemessen?
                                                {
                                                        for(k=1;k<messkontrolle.count;k++)
                                                        {
                                                                SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-messdaten30[0].aslzeit)/200*(maxwinx-
RABK-LABK)),(int) round( (starty-(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-messdaten30[0].aslzeit)/200*(maxwinx-RABK-
LABK)),(int) round ((starty-(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                        }

                                                        if(messkontrolle.count>1)
                                                        {
                                                                SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].aslzeit-
messdaten30[0].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewerteaslx -messdaten30[0].aslzeit)/(200)*(maxwinx-
RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerteasl -ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))));
```

```
                                        }
                                }
                                else
                                {
                                        l=messkontrolle.count-1;
                                        do
                                        {
                                        }
                                        while((messkontrolle.letztewerteaslx-
messdaten30[l--].aslzeit)<200);
                                        l++;

                                        for(k=l+2;k<=messkontrolle.count;k++)
                                        {
                                                SelectObject(hDc,hStift[0]);
```

MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-messdaten30[l+1].aslzeit)/200*(maxwinx-RABK-LABK)),(int) round((starty-(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-messdaten30[l+1].aslzeit)/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));

```
                                        }

                                        if(messkontrolle.count>1)
                                        {
                                                SelectObject(hDc,hStift[0]);
```

MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].aslzeit-messdaten30[l+1].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int) round((starty-(messdaten30[messkontrolle.count-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

LineTo(hDc,(int)(startx+(messkontrolle.letztewerteaslx -messdaten30[l+1].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerteasl -ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));

```
                                        }
                                }
                        }
                        else // 4 kleine Fenster zeichnen
                        {
                                if(messkontrolle.letztewertex[kanal-1]<200)          //
wurden bereits 200s gemessen?
                                {
                                        if(kanal<=2 ) // Innentemperatur des Geräts
nicht im HIRES Fenster zeichnen
                                                welchesfenster=8;
                                        else
                                                welchesfenster=7;

                                        for(k=1;k<messkontrolle.count;k++)
                                        {

                                                for(j=0;j<welchesfenster;j++)
                                                {
                                                        SelectObject(hDc,hStift[j]);
```

MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-1])/200*(maxwinx-RABK-LABK)),(int) round((starty-(messdaten30[k-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-

```
1])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                        if(messkontrolle.count>1)
                                                        {
                                                                for(j=0;j<welchesfenster;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[0].t[4*j+kanal-1])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }
                                                }
                                                else
                                                {
                                                        l=messkontrolle.count-1;
                                                        do
                                                        {
                                                        }
                                                        while((messkontrolle.letztewertex[kanal-1]-
messdaten30[l--].t[kanal-1])<200);
                                                        l++;

                                                        for(k=l+1;k<messkontrolle.count;k++)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-1].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                        if(messkontrolle.count>1)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[l].t[4*j+kanal-1])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
```

```
                                                }
                                        }
                                }
                        break;

                        case KANAL30TT:
                                if(kanal==ASLPAINT) //ASL zeichnen
                                {
                                        if(messkontrolle.letztewerteaslx<200)   //   wurden
bereits 200s gemessen?
                                        {
                                                for(k=1;k<messkontrolle.count;k++)
                                                {
                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-messdaten30[0].aslzeit)/200*(maxwinx-
RABK-LABK)),(int)   round(   (starty-(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-messdaten30[0].aslzeit)/200*(maxwinx-RABK-
LABK)),(int)   round   ((starty-(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                }

                                                if(messkontrolle.count>1)
                                                {
                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].aslzeit-
messdaten30[0].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int)   round(   (starty-
(messdaten30[messkontrolle.count-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewerteaslx   -messdaten30[0].aslzeit)/(200)*(maxwinx-
RABK-LABK)),(int)   round   ((starty-(messkontrolle.letztewerteasl   -ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))));
                                                }
                                        }
                                        else
                                        {
                                                l=messkontrolle.count-1;
                                                do
                                                {
                                                }
                                                while((messkontrolle.letztewerteaslx-
messdaten30[l--].aslzeit)<200);
                                                l++;

                                                for(k=l+2;k<=messkontrolle.count;k++)
                                                {
                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-messdaten30[l+1].aslzeit)/200*(maxwinx-
RABK-LABK)),(int)   round(   (starty-(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-messdaten30[l+1].aslzeit)/200*(maxwinx-RABK-
LABK)),(int)   round   ((starty-(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                }

                                                if(messkontrolle.count>1)
                                                {
```

```
                                                                SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].aslzeit-
messdaten30[l+1].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewerteaslx -messdaten30[l+1].aslzeit)/(200)*(maxwinx-
RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerteasl -ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))));
                                                        }
                                                }
                                        }
                                        else // 4 kleine Fenster zeichnen
                                        {
                                                if(messkontrolle.letztewertex[kanal-1]<200)             //
wurden bereits 200s gemessen?
                                                {
                                                        for(k=1;k<messkontrolle.count;k++)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-1].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                        if(messkontrolle.count>1)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[0].t[4*j+kanal-1])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }
                                                }
                                                else
                                                {
                                                        l=messkontrolle.count-1;
                                                        do
                                                        {
                                                        }
                                                        while((messkontrolle.letztewertex[kanal-1]-
messdaten30[l--].t[kanal-1])<200);
                                                        l++;

                                                        for(k=l+1;k<messkontrolle.count;k++)
                                                        {
```

```
                                                                for(j=0;j<8;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-1].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                        if(messkontrolle.count>1)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {
                                                                        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[l].t[4*j+kanal-1])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }
                                                }
                                        }
                                break;

                                case KANAL30TTLF:
                                        if(kanal==ASLPAINT) //ASL zeichnen
                                        {
                                                if(messkontrolle.letztewerteaslx<200)   //   wurden
bereits 200s gemessen?
                                                {
                                                        for(k=1;k<messkontrolle.count;k++)
                                                        {
                                                                SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-messdaten30[0].aslzeit)/200*(maxwinx-
RABK-LABK)),(int) round( (starty-(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-messdaten30[0].aslzeit)/200*(maxwinx-RABK-
LABK)),(int) round ((starty-(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                        }

                                                        if(messkontrolle.count>1)
                                                        {
                                                                SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].aslzeit-
messdaten30[0].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewerteaslx     -messdaten30[0].aslzeit)/(200)*(maxwinx-
```

```
RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerteasl -ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))));
                                                }
                                        }
                                        else
                                        {
                                                l=messkontrolle.count-1;
                                                do
                                                {
                                                }
                                                while((messkontrolle.letztewerteaslx-
messdaten30[l--].aslzeit)<200);
                                                l++;

                                                for(k=l+2;k<=messkontrolle.count;k++)
                                                {
                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-messdaten30[l+1].aslzeit)/200*(maxwinx-
RABK-LABK)),(int) round( (starty-(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-messdaten30[l+1].aslzeit)/200*(maxwinx-RABK-
LABK)),(int) round ((starty-(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                }

                                                if(messkontrolle.count>1)
                                                {
                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].aslzeit-
messdaten30[l+1].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewerteaslx -messdaten30[l+1].aslzeit)/(200)*(maxwinx-
RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerteasl -ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))));
                                                }
                                        }
                                }
                                else // 4 kleine Fenster zeichnen
                                {
                                        if(messkontrolle.letztewertex[kanal-1]<200) //
wurden bereits 200s gemessen?
                                        {
                                                if(kanal<=4) // Thermometerdaten
                                                {

        for(k=1;k<messkontrolle.count;k++)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-1].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
```

```
1])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                                }
                                                                        }

                                                                        if(messkontrolle.count>1)
                                                                        {
                                                                                for(j=0;j<8;j++)
                                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[0].t[4*j+kanal-1])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                                }
                                                                        }
                                                                }
                                                                else // Leitfähigkeitsdaten
                                                                {

        for(k=1;k<messkontrolle.count;k++)
                                                                        {
                                                                                for(j=0;j<8;j++)
                                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-
1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[32+4*j+kanal-1-4]-messdaten30[0].t[32+4*j+kanal-1-
4])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[32+4*j+kanal-1-4]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                                }
                                                                        }

                                                                        if(messkontrolle.count>1)
                                                                        {
                                                                                for(j=0;j<8;j++)
                                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messkontrolle.letztewerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                                }
                                                                        }
                                                                }
                                                        }
                                                        else
```

```
                                        {
                                                l=messkontrolle.count-1;
                                                do
                                                {
                                                }
                                                while((messkontrolle.letztewertex[kanal-1]-
messdaten30[l--].t[kanal-1])<200);
                                                l++;

                                                if(kanal<=4) // Thermometerdaten
                                                {

        for(k=l+1;k<messkontrolle.count;k++)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-1].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                        if(messkontrolle.count>1)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[l].t[4*j+kanal-1])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                }
                                                else // Leitfähigkeitsdaten
                                                {

        for(k=l+1;k<messkontrolle.count;k++)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[32+4*j+kanal-1-4]-
messdaten30[l].t[32+4*j+kanal-1-4])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-
1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[32+4*j+kanal-1-4]-messdaten30[l].t[32+4*j+kanal-1-
```

```
4])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[32+4*j+kanal-1-4]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                                }
                                                                        }

                                                                        if(messkontrolle.count>1)
                                                                        {
                                                                                for(j=0;j<8;j++)
                                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[32+4*j+kanal-1-4]-
messdaten30[l].t[32+4*j+kanal-1-4])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[32+4*j+kanal-1-4]-
messdaten30[l].t[32+4*j+kanal-1-4])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messkontrolle.letztewerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                                }
                                                                        }

                                                                }
                                                        }
                                                }
                                        break;

                                        case KANAL30TTLFKAL:
                                                if(kanal==ASLPAINT) //ASL zeichnen
                                                {
                                                        if(messkontrolle.letztewerteaslx<200)   //      wurden
bereits 200s gemessen?
                                                        {
                                                                for(k=1;k<messkontrolle.count;k++)
                                                                {
                                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-messdaten30[0].aslzeit)/200*(maxwinx-
RABK-LABK)),(int) round( (starty-(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-messdaten30[0].aslzeit)/200*(maxwinx-RABK-
LABK)),(int) round ((starty-(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                                }

                                                                if(messkontrolle.count>1)
                                                                {
                                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].aslzeit-
messdaten30[0].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewerteaslx -messdaten30[0].aslzeit)/(200)*(maxwinx-
RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerteasl -ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }
                                                        else
                                                        {
```

```
                                                l=messkontrolle.count-1;
                                                do
                                                {
                                                }
                                                while((messkontrolle.letztewerteaslx-
messdaten30[l--].aslzeit)<200);
                                                l++;

                                                for(k=l+2;k<=messkontrolle.count;k++)
                                                {
                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].aslzeit-messdaten30[l+1].aslzeit)/200*(maxwinx-
RABK-LABK)),(int) round( (starty-(messdaten30[k-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].aslzeit-messdaten30[l+1].aslzeit)/200*(maxwinx-RABK-
LABK)),(int) round ((starty-(messdaten30[k].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                }

                                                if(messkontrolle.count>1)
                                                {
                                                        SelectObject(hDc,hStift[0]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].aslzeit-
messdaten30[l+1].aslzeit)/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].asl-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewerteaslx -messdaten30[l+1].aslzeit)/(200)*(maxwinx-
RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerteasl -ywertmin)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))));
                                                }
                                        }
                                }
                                else // 4 kleine Fenster zeichnen
                                {
                                        if(messkontrolle.letztewertex[kanal-1]<200)             //
wurden bereits 200s gemessen?
                                        {
                                                if(kanal<=4) // Thermometerdaten
                                                {

        for(k=1;k<messkontrolle.count;k++)
                                                        {
                                                                for(j=0;j<8;j++)
                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-1].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                }
                                                        }

                                                        if(messkontrolle.count>1)
                                                        {
```

```
                                                                    for(j=0;j<8;j++)
                                                                    {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[0].t[4*j+kanal-1])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[0].t[4*j+kanal-
1])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                    }
                                                            }
                                                    }
                                                    else // Leitfähigkeitsdaten
                                                    {

        for(k=1;k<messkontrolle.count;k++)
                                                            {
                                                                    for(j=0;j<8;j++)
                                                                    {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-
1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[32+4*j+kanal-1-4]-messdaten30[0].t[32+4*j+kanal-1-
4])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[32+4*j+kanal-1-4]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                    }
                                                            }

                                                            if(messkontrolle.count>1)
                                                            {
                                                                    for(j=0;j<8;j++)
                                                                    {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[32+4*j+kanal-1-4]-
messdaten30[0].t[32+4*j+kanal-1-4])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messkontrolle.letztewerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                    }
                                                            }
                                                    }
                                            }
                                            else
                                            {
                                                    l=messkontrolle.count-1;
                                                    do
                                                    {
                                                    }
                                                    while((messkontrolle.letztewertex[kanal-1]-
messdaten30[l--].t[kanal-1])<200);
```

```
                                                    l++;

                                                    if(kanal<=4) // Thermometerdaten
                                                    {

        for(k=l+1;k<messkontrolle.count;k++)
                                                            {
                                                                    for(j=0;j<8;j++)
                                                                    {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-1].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                    }
                                                            }

                                                            if(messkontrolle.count>1)
                                                            {
                                                                    for(j=0;j<8;j++)
                                                                    {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[4*j+kanal-1]-
messdaten30[l].t[4*j+kanal-1])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[4*j+kanal-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[4*j+kanal-1]-messdaten30[l].t[4*j+kanal-
1])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-(messkontrolle.letztewerte[4*j+kanal-1]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                    }
                                                            }

                                                    }
                                                    else // Leitfähigkeitsdaten
                                                    {

        for(k=l+1;k<messkontrolle.count;k++)
                                                            {
                                                                    for(j=0;j<8;j++)
                                                                    {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[k-1].t[32+4*j+kanal-1-4]-
messdaten30[l].t[32+4*j+kanal-1-4])/200*(maxwinx-RABK-LABK)),(int) round( (starty-(messdaten30[k-
1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messdaten30[k].t[32+4*j+kanal-1-4]-messdaten30[l].t[32+4*j+kanal-1-
4])/200*(maxwinx-RABK-LABK)),(int) round ((starty-(messdaten30[k].ywerte[32+4*j+kanal-1-4]-
ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                    }
                                                            }

                                                            if(messkontrolle.count>1)
                                                            {
```

```
                                                                                for(j=0;j<8;j++)
                                                                                {

        SelectObject(hDc,hStift[j]);

        MoveToEx(hDc,(int)(startx+(messdaten30[messkontrolle.count-1].t[32+4*j+kanal-1-4]-
messdaten30[l].t[32+4*j+kanal-1-4])/(200)*(maxwinx-RABK-LABK)),(int) round( (starty-
(messdaten30[messkontrolle.count-1].ywerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-
UABK-OABK))),NULL);

        LineTo(hDc,(int)(startx+(messkontrolle.letztewertex[32+4*j+kanal-1-4]-
messdaten30[l].t[32+4*j+kanal-1-4])/(200)*(maxwinx-RABK-LABK)),(int) round ((starty-
(messkontrolle.letztewerte[32+4*j+kanal-1-4]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-OABK))));
                                                                                }
                                                                        }

                                                                }
                                                        }
                                                }
                                        break;
                                }

                                SelectObject(hDc, GetStockObject(BLACK_PEN));

                                DeleteObject(hUp);
                                DeleteObject(hRe);

                                for(j=0;j<8;j++) // Stifte wieder aufräumen
                                        DeleteObject(hStift[j]);

                                EndPaint(hWnd, &ps);
                        }
                        else
                        {
                                DeleteObject(hUp);
                                DeleteObject(hRe);
                                for(j=0;j<8;j++) // Stifte wieder aufräumen
                                        DeleteObject(hStift[j]);
                                ReleaseDC(hWnd,hDc);
                        }
                }
                else
                {
                        if(flag==PAINTMSG) // WM_PAINTMeldung oder nicht Passende Aufräumarbeiten
                          EndPaint(hWnd, &ps);
                        else
                        {
                                ReleaseDC(hWnd,hDc);
                        }
                }

}
```

11.7.2.10 kanal1ut.cpp

```
// Funktionen für 1 Kanal U/t schreiber

#include "StdAfx.h"

extern HINSTANCE hInst;
extern MESSSTEUERSTRUCT defmessparam;

extern DATENSATZ * messdaten;
```

```
extern HGLOBAL hMemHandle1;

REIHENSTEUERSTRUCT messkontrolle;

void kanal1utschreiber(HWND hWnd) //1 Kanal U/t schreiber
{
int maxwinx,maxwiny,startx,starty;
long int i=0;
double ywertmax,ywertmin,u=0,messdauer=0;
HDC hDc;
RECT Rect;
char outtext[100];

        if(DialogBox(hInst, (LPCTSTR)IDD_1Kanal, hWnd, (DLGPROC)
StartDialogkanal1ut)==FALSE)
                return; // bei Abbruch zurück

        messkontrolle.messtyp=TEST1KANAL; //Einstellungen für diesen messtyp
        messkontrolle.messan=TRUE; // eine Messung läuft
        messkontrolle.maximalerywert=UREF/defmessparam.verstearker;
        messkontrolle.minmalerywert=0; // 0V da nicht bipolar

        drawkosy(hWnd, NOPAINTMSG);

        ywertmax=UREF/defmessparam.verstearker;
        ywertmin=0;

        hDc=GetDC(hWnd);// Fenster Parameter
        GetClientRect(hWnd, &Rect);
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;
        starty=(int) (maxwiny-UAB-OAB)+OAB;
        startx=(int) LAB;

        messkontrolle.flagkanal1=FALSE; // Messkanal setzen
        messkontrolle.flagkanal2=FALSE;
        messkontrolle.flagkanal3=FALSE;
        messkontrolle.flagkanal4=FALSE;
        messkontrolle.flagkanal5=FALSE;
        messkontrolle.flagkanal6=FALSE;
        messkontrolle.flagkanal7=FALSE;
        messkontrolle.flagkanal8=FALSE;

        switch(defmessparam.kanal)
        {
                case 1 : messkontrolle.flagkanal1=TRUE;
                break;
                case 2 : messkontrolle.flagkanal2=TRUE;
                break;
                case 3 : messkontrolle.flagkanal3=TRUE;
                break;
                case 4 : messkontrolle.flagkanal4=TRUE;
                break;
                case 5 : messkontrolle.flagkanal5=TRUE;
                break;
                case 6 : messkontrolle.flagkanal6=TRUE;
                break;
                case 7 : messkontrolle.flagkanal7=TRUE;
                break;
                case 8 : messkontrolle.flagkanal8=TRUE;
                break;

        }
```

```
	GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden

	ads1241klarmachen(defmessparam.ba);
	startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts
	ads1241getu(defmessparam.ba,defmessparam.verstearker,defmessparam.kanal);
	messdauer=gettime(defmessparam.ba); // ersten Wert speichern

	do
	{
		do
		{
		}
		while(messdauer==gettime(defmessparam.ba)); //auf nächsten tick warten;
		messdauer=gettime(defmessparam.ba);
		u=ads1241getu(defmessparam.ba,defmessparam.verstearker,defmessparam.kanal);

		messdaten[i].t=messdauer-1;
		messdaten[i].ywerte[defmessparam.kanal-1]=u;

		sprintf(outtext,"%.0lf s, %.5lf V", messdauer-1, u);
	TextOut(hDc,LAB+10,(OAB-TEXTYDY)/2,outtext,strlen(outtext));
		//LAB+10,maxwiny-UAB-(ANZAHLYTICKS*dy)-TEXTYDY

		if(i>0)
		{
			MoveToEx(hDc,(int)(startx+(messdaten[i-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) (starty-messdaten[i-
1].ywerte[defmessparam.kanal-1]/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)),NULL);
			LineTo(hDc,(int)(startx+(messdaten[i].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) (starty-
messdaten[i].ywerte[defmessparam.kanal-1]/(ywertmax-ywertmin)*(maxwiny-UAB-OAB)));
		}

		i++;
		if(GetAsyncKeyState('E') != 0)
		{
			messkontrolle.messdauer=messdauer;
			break; // Messung abbrechen
		}
	}
	while(messdauer<=messkontrolle.messdauer);

	messkontrolle.anzahl=i;
	clockoff(defmessparam.ba);
	GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden,
um das Speichern nicht zu stören

	save1ut(hWnd); // Daten speichern

	ReleaseDC(hWnd,hDc);
	messkontrolle.messan=FALSE; // keine Messung läuft

}

/*****************************************************************************
* Dialog Funkion für das Starten der Messung                                 *
*****************************************************************************/
BOOL FAR PASCAL StartDialogkanal1ut(HWND hDlg, WORD msg, WPARAM wParam, LPARAM
lParam)
{
char dummy[100];
```

```
switch(msg)
{
        case WM_INITDIALOG :

                sprintf(dummy,"%.0lf",messkontrolle.messdauer);//Ausgabe der Messdauer
    SetDlgItemText(hDlg,IDC_MESSDAUER,dummy);

                SendDlgItemMessage(hDlg,IDC_KANAL1, BM_SETCHECK, 0, 0); // alle aus
                SendDlgItemMessage(hDlg,IDC_KANAL2, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_KANAL3, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_KANAL4, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_KANAL5, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_KANAL6, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_KANAL7, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_KANAL8, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_AMP1, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_AMP2, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_AMP4, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_AMP8, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_AMP16, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_AMP32, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_AMP64, BM_SETCHECK, 0, 0);
                SendDlgItemMessage(hDlg,IDC_AMP128, BM_SETCHECK, 0, 0);

                switch(defmessparam.kanal) // Radiobutton Kanal passend setzen
                {
                        case 1: SendDlgItemMessage(hDlg,IDC_KANAL1, BM_SETCHECK,
1, 0);
                                break;
                        case 2: SendDlgItemMessage(hDlg,IDC_KANAL2, BM_SETCHECK,
1, 0);
                                break;
                        case 3: SendDlgItemMessage(hDlg,IDC_KANAL3, BM_SETCHECK,
1, 0);
                                break;
                        case 4: SendDlgItemMessage(hDlg,IDC_KANAL4, BM_SETCHECK,
1, 0);
                                break;
                        case 5: SendDlgItemMessage(hDlg,IDC_KANAL5, BM_SETCHECK,
1, 0);
                                break;
                        case 6: SendDlgItemMessage(hDlg,IDC_KANAL6, BM_SETCHECK,
1, 0);
                                break;
                        case 7: SendDlgItemMessage(hDlg,IDC_KANAL7, BM_SETCHECK,
1, 0);
                                break;
                        case 8: SendDlgItemMessage(hDlg,IDC_KANAL8, BM_SETCHECK,
1, 0);
                                break;
                }

                switch(defmessparam.verstearker) // Radiobutton Kanal passend setzen
                {
                        case 1: SendDlgItemMessage(hDlg,IDC_AMP1, BM_SETCHECK, 1,
0);
                                break;
                        case 2: SendDlgItemMessage(hDlg,IDC_AMP2, BM_SETCHECK, 1,
0);
                                break;
                        case 4: SendDlgItemMessage(hDlg,IDC_AMP4, BM_SETCHECK, 1,
0);
                                break;
```

```
                                case  8: SendDlgItemMessage(hDlg,IDC_AMP8,  BM_SETCHECK,  1,
0);
                                        break;
                                case 16: SendDlgItemMessage(hDlg,IDC_AMP16, BM_SETCHECK, 1,
0);
                                        break;
                                case 32: SendDlgItemMessage(hDlg,IDC_AMP32, BM_SETCHECK, 1,
0);
                                        break;
                                case 64: SendDlgItemMessage(hDlg,IDC_AMP64, BM_SETCHECK, 1,
0);
                                        break;
                                case 128: SendDlgItemMessage(hDlg,IDC_AMP128, BM_SETCHECK,
1, 0);
                                        break;
                        }
            break;

             case WM_COMMAND :
        switch(wParam)
        {
          case IDOK:
            GetDlgItemText(hDlg,IDC_MESSDAUER,dummy,99);
            messkontrolle.messdauer=atof(dummy);//messdauer holen

                                        if(IsDlgButtonChecked(hDlg,IDC_KANAL1)==TRUE)      //
holen der Kanalinfo
                                                defmessparam.kanal=1;
                                        else if(IsDlgButtonChecked(hDlg,IDC_KANAL2)==TRUE)
                                                defmessparam.kanal=2;
                                        else if(IsDlgButtonChecked(hDlg,IDC_KANAL3)==TRUE)
                                                defmessparam.kanal=3;
                                        else if(IsDlgButtonChecked(hDlg,IDC_KANAL4)==TRUE)
                                                defmessparam.kanal=4;
                                        else if(IsDlgButtonChecked(hDlg,IDC_KANAL5)==TRUE)
                                                defmessparam.kanal=5;
                                        else if(IsDlgButtonChecked(hDlg,IDC_KANAL6)==TRUE)
                                                defmessparam.kanal=6;
                                        else if(IsDlgButtonChecked(hDlg,IDC_KANAL7)==TRUE)
                                                defmessparam.kanal=7;
                                        else if(IsDlgButtonChecked(hDlg,IDC_KANAL8)==TRUE)
                                                defmessparam.kanal=8;

                                        if(IsDlgButtonChecked(hDlg,IDC_AMP1)==TRUE) // holen
der Kanalinfo
                                                defmessparam.verstearker=1;
                                        else if(IsDlgButtonChecked(hDlg,IDC_AMP2)==TRUE)
                                                defmessparam.verstearker=2;
                                        else if(IsDlgButtonChecked(hDlg,IDC_AMP4)==TRUE)
                                                defmessparam.verstearker=4;
                                        else if(IsDlgButtonChecked(hDlg,IDC_AMP8)==TRUE)
                                                defmessparam.verstearker=8;
                                        else if(IsDlgButtonChecked(hDlg,IDC_AMP16)==TRUE)
                                                defmessparam.verstearker=16;
                                        else if(IsDlgButtonChecked(hDlg,IDC_AMP32)==TRUE)
                                                defmessparam.verstearker=32;
                                        else if(IsDlgButtonChecked(hDlg,IDC_AMP64)==TRUE)
                                                defmessparam.verstearker=64;
                                        else if(IsDlgButtonChecked(hDlg,IDC_AMP128)==TRUE)
                                                defmessparam.verstearker=128;

            if(messkontrolle.messdauer<=0)
            {
```

```
                MessageBox(hDlg, "Für Messdauer nur Werte größer 0 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
                return FALSE;
              }
                                          else if(messkontrolle.messdauer>MAXMESSDAUER)
              {
                MessageBox(hDlg, "Für Messdauer nur Werte kleiner als 360000 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
                return FALSE;
              }

              EndDialog(hDlg,TRUE);
              return TRUE;
            break;

            case IDCANCEL :
              EndDialog(hDlg,FALSE);
              return TRUE;
            break;

          }
      break;

   }

 return(FALSE);

}

int save1ut(HWND hWnd)
{
static char  szFilename[810] = ".txt";
int iErg=0;
char *feld,zeile[30];
long int i;
HGLOBAL hMemHandleDaten;
HFILE hFile;
unsigned long lBytes;
OPENFILENAME ofn;

        hMemHandleDaten=GlobalAlloc(GHND,sizeof(char)*30*MAXMESSDAUER);    //Größe des
Charfeldes
        feld=(char*)GlobalLock(hMemHandleDaten);

        memset(&ofn, 0, sizeof(OPENFILENAME));
        ofn.lpstrFile = szFilename;
        iErg = FileDialogSave1ut(hWnd, &ofn);

        if(iErg == IDOK)
        {
          if((hFile = _lcreat(ofn.lpstrFile,0)) == HFILE_ERROR)
          {
            MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK);
            SendMessage(hWnd,WM_CLOSE,NULL,NULL);
            GlobalUnlock(hMemHandleDaten);
            GlobalFree(hMemHandleDaten);
            return(FEHLER);
          }

          // an den Dateianfang gehen
          _llseek(hFile, 0L, 0);

          // Datei schreiben
```

```
                                for(i=0;i<messkontrolle.anzahl;i++)
            {
               sprintf(zeile,"%.0lf %.8lf\r\n",messdaten[i].t,messdaten[i].ywerte[defmessparam.kanal-1]);
                                                lBytes = _hwrite(hFile,zeile,strlen(zeile));
                                                if(lBytes!=strlen(zeile))
                                                {
                                                        MessageBox(hWnd, "Da ist was danebengegangen",
"Achtung", MB_OK|MB_ICONSTOP);
                           _lclose(hFile);
                        GlobalUnlock(hMemHandleDaten);
                     GlobalFree(hMemHandleDaten);
                  return(FEHLER);
                                                }
                                                else
                                                {
                                  //_llseek(hFile, lBytes, 1);
                                                }
            }
            _lclose(hFile);
                                }

         GlobalUnlock(hMemHandleDaten);
         GlobalFree(hMemHandleDaten);
         return(OK);
}


int FileDialogSave1ut(HWND hWnd, OPENFILENAME *ofn)
{
      char szFilter[256], szText[]= "Text-Datei (*.TXT)|*.txt|Alle Dateien (*.*)|*.*||";
      int i;

      for(i=0; szText[i]!= '\0'; ++i)
        szFilter[i] = szText[i] == '|' ? '\0' : szText[i];

      ofn->lStructSize = sizeof(OPENFILENAME);
      ofn->hwndOwner = hWnd;
      ofn->lpstrFilter = szFilter;
      ofn->nMaxFile = _MAX_PATH;
      ofn->nMaxFileTitle = _MAX_FNAME + _MAX_EXT;
      ofn->lpstrDefExt = "txt";
         return GetSaveFileName(ofn)? IDOK : IDCANCEL;
}
```

11.7.2.11 kanal8Tt.cpp

```
// Funktionen für 8 Kanal T/t schreiber

#include "StdAfx.h"

extern HINSTANCE hInst;
extern MESSSTEUERSTRUCT defmessparam;

extern DATENSATZ * messdaten;
extern HGLOBAL hMemHandle1;
extern REIHENSTEUERSTRUCT messkontrolle;

double bereichsumschalter[8]={0.164,0.104,0.0633,0.0376,0.0242,0.0149,0.0113}; // Spannungen für die
Kanalumschaltung

void kanal8Ttschreiber(HWND hWnd) //8 Kanal T/t schreiber
{
```

```
int maxwinx,maxwiny,startx,starty,j=0,k,pga,iErg,readcount=0;
long int i=0;
double ywertmax,ywertmin,u=0,messdauer=0, theta;
HDC hDc;
RECT Rect;
char outtext[1000];
HPEN hStift[8];
OPENFILENAME ofn;
static char  szFilename[810] = ".kal";
HFILE hFile;
char readdata[200],read1;
HANDLE hFile32;

        if(DialogBox(hInst, (LPCTSTR)IDD_8KanalTt, hWnd, (DLGPROC)
StartDialogkanal8Tt)==FALSE)
                return; // bei Abbruch zurück

        // Kalibrierdaten einlesen

   memset(&ofn, 0, sizeof(OPENFILENAME));
   ofn.lpstrFile = szFilename;
   iErg = FileDialogKal(hWnd, &ofn);

        if(iErg == IDOK)
   {
      //Datei öffnen mit Fehlertest
      if((hFile = _lopen(ofn.lpstrFile, OF_READ)) == -1)
      {
         MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK|MB_ICONSTOP);
         SendMessage(hWnd,WM_CLOSE,NULL,NULL);
         return;
      }

      //an den Dateianfang gehen
      _llseek(hFile, 0L, 0);

      // Datei lesen,  Fehlerprüfung hinzufügen

                for(k=0;k<8;k++)
                {
                        readcount=0;
                        do //eine Zeile einlesen
                        {
                                _hread(hFile,&read1,1);
                                readdata[readcount++]=read1;
                        }
                        while(readdata[readcount-1]!=13 ); //Zeilenende

                        readdata[readcount-1]=0; //string beenden
                        _hread(hFile,&read1,1); // noch Zeilenvorschub einlesen

                        sscanf(readdata,"%le %le %le %le %le
",&messkontrolle.kalparam[k].a,&messkontrolle.kalparam[k].b,&messkontrolle.kalparam[k].c,&messkontrolle.
kalparam[k].uref,&messkontrolle.kalparam[k].r1);
                }

                _lclose(hFile);

        }
        else
                return;
```

```
	hFile32=CreateFile(defmessparam.tempfile,GENERIC_WRITE,FILE_SHARE_READ,NULL,CR
EATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
	CloseHandle(hFile32); // altes Tempfile löschen

	messkontrolle.messtyp=KANAL8TT; //Einstellungen für diesen Messtyp
	messkontrolle.messan=TRUE; // eine Messung läuft
	messkontrolle.maximalerywert=MAXTEMPERATUR;
	messkontrolle.minmalerywert=MINTEMPERATUR;

	drawkosy(hWnd, NOPAINTMSG);

	hStift[0]=CreatePen(PS_SOLID,0,RGB(0,0,0)); // einzelne Stifte herrichten
	hStift[1]=CreatePen(PS_SOLID,0,RGB(128,0,0));
	hStift[2]=CreatePen(PS_SOLID,0,RGB(255,0,0));
	hStift[3]=CreatePen(PS_SOLID,0,RGB(255,126,0));
	hStift[4]=CreatePen(PS_SOLID,0,RGB(255,255,0));
	hStift[5]=CreatePen(PS_SOLID,0,RGB(0,255,0));
	hStift[6]=CreatePen(PS_SOLID,0,RGB(0,0,255));
	hStift[7]=CreatePen(PS_SOLID,0,RGB(192,192,192));

	ywertmax=messkontrolle.maximalerywert;
	ywertmin=messkontrolle.minmalerywert;

	hDc=GetDC(hWnd);// Fenster Parameter
	GetClientRect(hWnd, &Rect);
	maxwinx=Rect.right;
	maxwiny=Rect.bottom;
	starty=(int) (maxwiny-UAB-OAB)+OAB;
	startx=(int) LAB;

	messkontrolle.flagkanal1=TRUE; // Messkanal setzen
	messkontrolle.flagkanal2=TRUE;
	messkontrolle.flagkanal3=TRUE;
	messkontrolle.flagkanal4=TRUE;
	messkontrolle.flagkanal5=TRUE;
	messkontrolle.flagkanal6=TRUE;
	messkontrolle.flagkanal7=TRUE;
	messkontrolle.flagkanal8=TRUE;

	GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden

	ads1241klarmachen(defmessparam.ba);
	startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts
	ads1241getu(defmessparam.ba,defmessparam.verstearker,defmessparam.kanal);
	messdauer=gettime(defmessparam.ba); // ersten Wert speichern

	do // eigentliche Messschleife
	{
		do
		{
		}
		while(messdauer==gettime(defmessparam.ba)); //auf nächsten tick warten;
		messdauer=gettime(defmessparam.ba);

		for(j=1;j<=8;j++)
		{
			u=ads1241getu(defmessparam.ba,1,j); // Spannung einlesen, um Bereich
festzustellen

			if(u<=bereichsumschalter[6])
				pga=128;
			else if(u<=bereichsumschalter[5])
					pga=64;
```

```
                        else if(u<=bereichsumschalter[4])
                                pga=32;
                        else if(u<=bereichsumschalter[3])
                                pga=16;
                        else if(u<=bereichsumschalter[2])
                                pga=8;
                        else if(u<=bereichsumschalter[1])
                                pga=4;
                        else if(u<=bereichsumschalter[0])
                                pga=2;
                        else
                                pga=1;

                        u=ads1241getu(defmessparam.ba,pga,j); // Spannung einlesen

                        messdaten[i].t=messdauer-1;
                        theta=maketemp(u,j); //berechnet die Temperatur aus der Spannung
                        messdaten[i].ywerte[j-1]=theta;

                        if(theta>messkontrolle.maxbathtemp)
                        {
                                u_ausgabe(0,defmessparam.ba); // Steuerheizer aus
                        }

                        SelectObject(hDc,hStift[j]);
                        if(i>0)
                        {
                                MoveToEx(hDc,(int)(startx+(messdaten[i-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[i-
1].ywerte[j-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);
                                LineTo(hDc,(int)(startx+(messdaten[i].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round ((starty-(messdaten[i].ywerte[j-
1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))));
                        }

                }
                sprintf(outtext,"%.0lf s,%i: %.2lf °C, %i: %.2lf °C, %i: %.2lf °C, %i: %.2lf °C, %i: %.2lf
°C, %i: %.2lf °C, %i: %.2lf °C, %i: %.2lf °C", messdauer-1, 1,messdaten[i].ywerte[0],2,
messdaten[i].ywerte[1],3, messdaten[i].ywerte[2],4, messdaten[i].ywerte[3],5, messdaten[i].ywerte[4],6,
messdaten[i].ywerte[5],7, messdaten[i].ywerte[6],8, messdaten[i].ywerte[7]);
        TextOut(hDc,LAB+10,(OAB-TEXTYDY)/2,outtext,strlen(outtext));
                saveuntermessung(hWnd,i); // Daten  während der Messung speichern

                i++;
                if(GetAsyncKeyState('E') != 0)
                {
                        messkontrolle.messdauer=messdauer;
                        break; // Messung abbrechen
                }
        }
        while(messdauer<=messkontrolle.messdauer);

        messkontrolle.anzahl=i;
        clockoff(defmessparam.ba);
        GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden,
um das Speichern nicht zu stören

        save8Tt(hWnd); // Daten speichern

        for(j=0;j<8;j++) // Stifte wieder aufräumen
                DeleteObject(hStift[j]);
        ReleaseDC(hWnd,hDc);
        messkontrolle.messan=FALSE; // keine Messung läuft
```

```
}


/*****************************************************************************
* Dialog Funkion für das Starten der Messung                                 *
*****************************************************************************/
BOOL FAR PASCAL StartDialogkanal8Tt(HWND hDlg, WORD msg, WPARAM wParam, LPARAM
lParam)
{
char dummy[100];

   switch(msg)
   {
                    case WM_INITDIALOG :
                                sprintf(dummy,"%.0lf",messkontrolle.messdauer);//Ausgabe der Messdauer
         SetDlgItemText(hDlg,IDC_MESSDAUER,dummy);
      break;

                    case WM_COMMAND :
         switch(wParam)
         {
            case IDOK:
               GetDlgItemText(hDlg,IDC_MESSDAUER,dummy,99);
               messkontrolle.messdauer=atof(dummy);//messdauer holen

               if(messkontrolle.messdauer<=0)
               {
                  MessageBox(hDlg, "Für Messdauer nur Werte größer 0 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
                  return FALSE;
               }
                                          else if(messkontrolle.messdauer>MAXMESSDAUER)
               {
                  MessageBox(hDlg, "Für Messdauer nur Werte kleiner als 360000 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
                  return FALSE;
               }

               EndDialog(hDlg,TRUE);
               return TRUE;
            break;

            case IDCANCEL :
               EndDialog(hDlg,FALSE);
               return TRUE;
            break;

         }
      break;

   }

 return(FALSE);

}

void saveuntermessung(HWND hWnd, long int i) // speichert Messdaten während der Messung
{
HANDLE hFile;
char outtext[200];
unsigned long dummy;
```

```
        hFile=CreateFile(defmessparam.tempfile,GENERIC_WRITE,FILE_SHARE_READ,NULL,OPE
N_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);

        // an das Dateiende gehen
        SetFilePointer(hFile,0,NULL,FILE_END);


        // Datei schreiben
        sprintf(outtext,"%.0lf %.8lf %.8lf %.8lf %.8lf %.8lf %.8lf %.8lf
%.8lf\r\n",messdaten[i].t,messdaten[i].ywerte[0],messdaten[i].ywerte[1],messdaten[i].ywerte[2],messdaten[i].y
werte[3],messdaten[i].ywerte[4],messdaten[i].ywerte[5],messdaten[i].ywerte[6],messdaten[i].ywerte[7]);
        WriteFile(hFile,outtext,strlen(outtext),&dummy,NULL);

        CloseHandle(hFile);

        return;
}

int save8Tt(HWND hWnd) // speichert Messdaten
{
static char  szFilename[810] = ".txt";
int iErg=0;
char *feld,zeile[120];
long int i;
HGLOBAL hMemHandleDaten;
HFILE hFile;
unsigned long lBytes;
OPENFILENAME ofn;

        hMemHandleDaten=GlobalAlloc(GHND,sizeof(char)*120*MAXMESSDAUER);  //Größe  des
Charfeldes
        feld=(char*)GlobalLock(hMemHandleDaten);

        memset(&ofn, 0, sizeof(OPENFILENAME));
        ofn.lpstrFile = szFilename;
        iErg = FileDialogSave1ut(hWnd, &ofn);

        if(iErg == IDOK)
        {
           if((hFile = _lcreat(ofn.lpstrFile,0)) == HFILE_ERROR)
           {
              MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK);
              SendMessage(hWnd,WM_CLOSE,NULL,NULL);
              GlobalUnlock(hMemHandleDaten);
              GlobalFree(hMemHandleDaten);
              return(FEHLER);
           }

           // an den Dateianfang gehen
           _llseek(hFile, 0L, 0);

           // Datei schreiben

                              for(i=0;i<messkontrolle.anzahl;i++)
           {
                                        sprintf(zeile,"%.0lf %.8lf %.8lf %.8lf %.8lf %.8lf %.8lf %.8lf
%.8lf\r\n",messdaten[i].t,messdaten[i].ywerte[0],messdaten[i].ywerte[1],messdaten[i].ywerte[2],messdaten[i].y
werte[3],messdaten[i].ywerte[4],messdaten[i].ywerte[5],messdaten[i].ywerte[6],messdaten[i].ywerte[7]);
                                        lBytes = _hwrite(hFile,zeile,strlen(zeile));
                                        if(lBytes!=strlen(zeile))
                                        {
                                                MessageBox(hWnd, "Da ist was danebengegangen",
"Achtung", MB_OK|MB_ICONSTOP);
```

```
                                                    _lclose(hFile);
                                                GlobalUnlock(hMemHandleDaten);
                                        GlobalFree(hMemHandleDaten);
                                return(FEHLER);
                                                }
                                                else
                                                {
                                                        //_llseek(hFile, lBytes, 1);
                                                }
                        }
                        _lclose(hFile);
                                }

                GlobalUnlock(hMemHandleDaten);
                GlobalFree(hMemHandleDaten);
                return(OK);
}

int FileDialogKal(HWND hWnd, OPENFILENAME *ofn)
{
        char szFilter[256],
        szText[]= "KAL-Datei (*.KAL)|*.kal|Alle Dateien (*.*)|*.*||";
        int i;

        for(i=0; szText[i]!= '\0'; ++i)
          szFilter[i] = szText[i] == '|' ? '\0' : szText[i];

        ofn->lStructSize = sizeof(OPENFILENAME);
        ofn->hwndOwner = hWnd;
        ofn->lpstrFilter = szFilter;
        ofn->nMaxFile = _MAX_PATH;
        ofn->nMaxFileTitle = _MAX_FNAME + _MAX_EXT;
        ofn->lpstrDefExt = "*";
                return GetOpenFileName(ofn)? IDOK : IDCANCEL;
}
```

11.7.2.12 kanal8Ut.cpp

```
// Funktionen für 8 Kanal U/t schreiber

#include "StdAfx.h"

extern HINSTANCE hInst;
extern MESSSTEUERSTRUCT defmessparam;

extern DATENSATZ * messdaten;
extern HGLOBAL hMemHandle1;
extern REIHENSTEUERSTRUCT messkontrolle;

extern double bereichsumschalter[8]; // Spannungen für die Kanalumschaltung

void kanal8utschreiber(HWND hWnd) //8 Kanal U/t schreiber
{
int maxwinx,maxwiny,startx,starty,j=0,pga,readcount=0;
long int i=0;
double ywertmax,ywertmin,u=0,messdauer=0;
HDC hDc;
RECT Rect;
char outtext[100];
HPEN hStift[8];

        if(DialogBox(hInst, (LPCTSTR)IDD_8KanalTt, hWnd, (DLGPROC)
StartDialogkanal8Tt)==FALSE)
```

```
                return; // bei Abbruch zurück

        messkontrolle.messtyp=KANAL8UT; //Einstellungen für diesen Messtyp
        messkontrolle.messan=TRUE; // eine Messung läuft
        messkontrolle.maximalerywert=UREF;
        messkontrolle.minmalerywert=0;

        drawkosy(hWnd, NOPAINTMSG);

        hStift[0]=CreatePen(PS_SOLID,0,RGB(0,0,0)); // einzelne Stifte herrichten
        hStift[1]=CreatePen(PS_SOLID,0,RGB(128,0,0));
        hStift[2]=CreatePen(PS_SOLID,0,RGB(255,0,0));
        hStift[3]=CreatePen(PS_SOLID,0,RGB(255,126,0));
        hStift[4]=CreatePen(PS_SOLID,0,RGB(255,255,0));
        hStift[5]=CreatePen(PS_SOLID,0,RGB(0,255,0));
        hStift[6]=CreatePen(PS_SOLID,0,RGB(0,0,255));
        hStift[7]=CreatePen(PS_SOLID,0,RGB(192,192,192));

        ywertmax=messkontrolle.maximalerywert;
        ywertmin=messkontrolle.minmalerywert;

        hDc=GetDC(hWnd);// Fenster Parameter
        GetClientRect(hWnd, &Rect);
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;
        starty=(int) (maxwiny-UAB-OAB)+OAB;
        startx=(int) LAB;

        messkontrolle.flagkanal1=TRUE; // Messkanal setzen
        messkontrolle.flagkanal2=TRUE;
        messkontrolle.flagkanal3=TRUE;
        messkontrolle.flagkanal4=TRUE;
        messkontrolle.flagkanal5=TRUE;
        messkontrolle.flagkanal6=TRUE;
        messkontrolle.flagkanal7=TRUE;
        messkontrolle.flagkanal8=TRUE;

        GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden

        ads1241klarmachen(defmessparam.ba);
        startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts
        ads1241getu(defmessparam.ba,defmessparam.verstearker,defmessparam.kanal);
        messdauer=gettime(defmessparam.ba); // ersten Wert speichern

        do
        {
                do
                {
                }
                while(messdauer==gettime(defmessparam.ba)); //auf nächsten tick warten;
                messdauer=gettime(defmessparam.ba);

                for(j=1;j<=8;j++)
                {
                        u=ads1241getu(defmessparam.ba,1,j); // Spannung einlesen, um Bereich
festzustellen

                        if(u<=bereichsumschalter[6])
                                pga=128;
                        else if(u<=bereichsumschalter[5])
                                        pga=64;
                        else if(u<=bereichsumschalter[4])
                                        pga=32;
```

```
                        else if(u<=bereichsumschalter[3])
                                pga=16;
                        else if(u<=bereichsumschalter[2])
                                pga=8;
                        else if(u<=bereichsumschalter[1])
                                pga=4;
                        else if(u<=bereichsumschalter[0])
                                pga=2;
                        else
                                pga=1;

                        u=ads1241getu(defmessparam.ba,pga,j); // Spannung einlesen

                        messdaten[i].t=messdauer-1;
                        messdaten[i].ywerte[j-1]=u;
                        sprintf(outtext,"%.0lf s, %.5lf V", messdauer-1, messdaten[i].ywerte[j-1]);


                TextOut(hDc,LAB+10,(OAB-TEXTYDY)/2,outtext,strlen(outtext));

                        SelectObject(hDc,hStift[j]);
                        if(i>0)
                        {
                                MoveToEx(hDc,(int)(startx+(messdaten[i-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[i-
1].ywerte[j-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);
                                LineTo(hDc,(int)(startx+(messdaten[i].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round ((starty-(messdaten[i].ywerte[j-
1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))));
                        }

                }
                i++;
                if(GetAsyncKeyState('E') != 0)
                {
                        messkontrolle.messdauer=messdauer;
                        break; // Messung abbrechen
                }
        }
        while(messdauer<=messkontrolle.messdauer);
        messkontrolle.anzahl=i;
        clockoff(defmessparam.ba);

        GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden,
um das Speichern nicht zu stören

        save8Tt(hWnd); // Daten speichern

        for(j=0;j<8;j++) // Stifte wieder aufräumen
                DeleteObject(hStift[j]);
        ReleaseDC(hWnd,hDc);
        messkontrolle.messan=FALSE; // keine Messung läuft
}
```

11.7.2.13 lf597s.cpp

```
//*************************************************************************
// Funktionen für die Kommunikation mit dem WTW LF597-S Leitfähigkeitsmesser
//*************************************************************************

#include "StdAfx.h";

extern REIHENSTEUERSTRUCT messkontrolle;
```

```
double getwtwlf597s(void)
{
char datain[1200],data[9],dummy[2];
unsigned long countout=2,countin=11;
double temp=0,multiplikator;
int i=0;
unsigned long lpEvtMask,lpModemStat;
double TimeUnit;


	TimeInit(&TimeUnit);

	//PurgeComm(messkontrolle.hComwtw,PURGE_TXCLEAR&&PURGE_RXCLEAR); // Puffer
leeren
	//WriteFile(messkontrolle.hComwtw,"T\n",countout,&countout,NULL);

	do
	{
	i=0;
	PurgeComm(messkontrolle.hComwtw,PURGE_RXCLEAR); // Puffer leeren
		do
		{
			do
			{
				WaitCommEvent(messkontrolle.hComwtw,&lpEvtMask,NULL);
			}
			while(lpEvtMask==EV_CTS);
			GetCommModemStatus(messkontrolle.hComwtw,&lpModemStat);
		}
		while(lpModemStat==MS_CTS_ON);

		warten(1,TimeUnit); // Zeit für Datenübertragung abwarten

		do
		{
			countin=1;

	if(ReadFile(messkontrolle.hComwtw,dummy,countin,&countin,NULL)==FALSE)
				break;
			datain[i++]=dummy[0];
		}
		while(dummy[0]!='\n');

		if(datain[4]==' ')
		{
			strncpy(data,datain,4);
			data[4]=0;
			if(datain[5]=='m')
				multiplikator=1e-3;
			else
				multiplikator=1e-6;
		}
		else if(datain[5]==' ')
		{
			strncpy(data,datain,5);
			data[5]=0;

			if(datain[6]=='m')
				multiplikator=1e-3;
			else
				multiplikator=1e-6;
		}
```

```
                if(datain[2]=='O')
                {
                        temp=200e-3; // bei C=0.1 S/cm
                        PurgeComm(messkontrolle.hComwtw,PURGE_RXCLEAR); // Puffer leeren
                        break;
                }
                PurgeComm(messkontrolle.hComwtw,PURGE_RXCLEAR); // Puffer leeren
                temp=atof(data)*multiplikator/0.1;
        }
        while(temp==0);

return(temp); // gibt die gemessene Leitfähigkeit in s zurück
}
```

11.7.2.14 max542.cpp

```
// Funktionen für die Ansteuerung des DAC MAX542
#include "StdAfx.h"
extern REIHENSTEUERSTRUCT messkontrolle;

void lpt_ausgeben(unsigned char byte,int ba)
{
unsigned char bitwert, portwert,dummy=0;
int bit;
UCHAR portbuffer=0;
portbuffer=_inp(ba);

        for(bit=7; bit >=0; bit--)
        {
                bitwert = 1 << bit;
                if((byte & bitwert) == bitwert) portwert = 0x10; else portwert = 0;
                {
                        portbuffer=(portbuffer&0xEF) + portwert;
                        _outp(ba, portbuffer); /* DIN setzen */ //bit 5 , pin 6
                }

                portbuffer=(portbuffer&0xDF) + 0x20; /* SCLK = 1 bei DIN gesetzt */
                _outp(ba, portbuffer);  //bit 8 , pin 7

                portbuffer=portbuffer&0xDF; /* SCLK = 0 bei DIN gesetzt */
                _outp(ba, portbuffer); //bit 8 , pin 7

        }

//      _outp(ba,dummy); // die beiden Zeilen dürfen eigentlich nicht notwendig sein ??
//      portbuffer=dummy; // wie ist es denn mit dem Stromsparmodus? ??
}

void max542(double u,int ba,int dac) // gibt eine Spannung aus
{
unsigned short datenwort;
unsigned char lobyte, hibyte;

        if(u >= UREFMAX542) u = UREFMAX542*32767/32768; /* Ausgabe über Vref nicht möglich
*/
        if(u < -UREFMAX542) u = -UREFMAX542;
        u += UREFMAX542;
        datenwort = (unsigned short) 32768 * u / UREFMAX542; // Konvertierung ist absichtlich so
gewählt
        lobyte = datenwort & 255;
        hibyte = datenwort >> 8;
```

```
	cs(LO,ba,dac);
	lpt_ausgeben(hibyte,ba);
	lpt_ausgeben(lobyte,ba);
	cs(HI,ba,dac);
}

void cs(unsigned char wert,int ba,int dac)
{
UCHAR portbuffer=0;

	if(dac==0) // 8Kanal
	{
		portbuffer=_inp(ba);
		portbuffer=(portbuffer & 0x7F)+0x80*wert; //bit 8 , pin 9 ; cs setzen
		_outp(ba,portbuffer);

	}
	else if(dac==1) // 30Kanal DAC1
	{
		portbuffer=_inp(ba+2);
		portbuffer=(portbuffer & 0xF7)+0x8*(!wert); //SLCK, invertiert , pin 17 ; cs setzen
		_outp(ba+2,portbuffer);
	}
	else if(dac==2) // 30Kanal DAC2
	{
		portbuffer=_inp(ba+2);
		portbuffer=(portbuffer & 0xFb)+0x4*(wert); //init, pin 16 ; cs setzen
		_outp(ba+2,portbuffer);
	}
}

void u_ausgabe(double u,int ba) // gibt eine Spannung am Ausgang aus
{
	if(messkontrolle.typ==KANAL8)
		max542(u/((double)R1DAC/R2DAC+1),ba,0); // Verstärkungsfaktor berechnen und an
DAC ausgeben

	else if(messkontrolle.typ==KANAL30)
		max542(u/((double)R1DAC/R2DAC+1),ba,1); // Verstärkungsfaktor berechnen und an
DAC 1 ausgeben
}

void motor(double u,int ba) // steuert den Motor 0 bis 100%
{
	u=u/100*MOTORKORREKTUR;
	max542(u/((double)R1DAC/R2DAC+1),ba,2); // Verstärkungsfaktor berechnen und an DAC 1
ausgeben
}

void frequenzausgabe(double f, int ba) // stellt die Frequenz der Leitfähigkeitsmessung ein, beide DACs
laufen synchron
{
	f=-0.218+5.938e-4*f;
	max542(f/((double)R1DAC/R2DAC+1),ba,1);
	max542(f/((double)R1DAC/R2DAC+1),ba,2);
}
```

11.7.2.15 mess30kanal.cpp

```
#include "stdafx.h"

#include "resource.h"
```

```
#define MAX_LOADSTRING 100

BOOL FAR PASCAL testdlg(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam);

TCHAR szTitleMess1[MAX_LOADSTRING];
TCHAR szTitleMess2[MAX_LOADSTRING];
TCHAR szTitleMess3[MAX_LOADSTRING];
TCHAR szTitleMess4[MAX_LOADSTRING];
TCHAR szTitleMessStatus[MAX_LOADSTRING];
TCHAR szTitleMessHiRes1[MAX_LOADSTRING];
TCHAR szTitleMessHiRes2[MAX_LOADSTRING];
TCHAR szTitleMessHiRes3[MAX_LOADSTRING];
TCHAR szTitleMessHiRes4[MAX_LOADSTRING];
TCHAR szTitleMessASL[MAX_LOADSTRING];
TCHAR szTitleMessHiResASL[MAX_LOADSTRING];
TCHAR szWindowClassMess1[MAX_LOADSTRING];
TCHAR szWindowClassMess2[MAX_LOADSTRING];
TCHAR szWindowClassMess3[MAX_LOADSTRING];
TCHAR szWindowClassMess4[MAX_LOADSTRING];
TCHAR szWindowClassMessStatus[MAX_LOADSTRING];
TCHAR szWindowClassMessHiRes1[MAX_LOADSTRING];
TCHAR szWindowClassHiRes2[MAX_LOADSTRING];
TCHAR szWindowClassMessHiRes3[MAX_LOADSTRING];
TCHAR szWindowClassMessHiRes4[MAX_LOADSTRING];
TCHAR szWindowClassASL[MAX_LOADSTRING];
TCHAR szWindowClassMessHiResASL[MAX_LOADSTRING];
HWND
mess1hwnd,mess2hwnd,mess3hwnd,mess4hwnd,messstatushwnd,messhi1hwnd,messhi2hwnd,messhi3hwnd,
messhi4hwnd; // Fensterhandle
HWND messASLhwnd, messhiASLhwnd;

extern HWND mainhwnd;
extern MESSSTEUERSTRUCT defmessparam;
extern HINSTANCE hInst;
extern FENSTERKOORDSSTRUCT fensterkoordinaten;
extern REIHENSTEUERSTRUCT messkontrolle;
extern DATENSATZ30 * messdaten30;
extern TIMECAPS multmediatimertc;
extern UINT multmediatimerwTimerRes;
extern                                                                    HWND
lfmess1hwnd,lfmess2hwnd,lfmess3hwnd,lfmess4hwnd,lfmessstatushwnd,lfmesshi1hwnd,lfmesshi2hwnd,lfme
sshi3hwnd,lfmesshi4hwnd; // Fensterhandle

ATOM MyRegisterClassMess1( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMess1;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMess1;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}
```

```
BOOL InitInstanceMess1( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd = CreateWindow(szWindowClassMess1, "ADC 1 Thermometer",WS_VISIBLE |WS_SYSMENU |
WS_OVERLAPPED | WS_CAPTION| WS_THICKFRAME | WS_MINIMIZEBOX |
WS_MAXIMIZEBOX ,
    fensterkoordinaten.xMess1,
fensterkoordinaten.yMess1,fensterkoordinaten.nWidthMess1,fensterkoordinaten.nHeightMess1, hwndParent,
NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);

  UpdateWindow( hWnd );
  mess1hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcMess1(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {

                case WM_CREATE :
                        break;

                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosy30kanal(hWnd, PAINTMSG,1); // zeichnet Kosy für die kleinen
Fenster beim 30-Kanalgerät
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMess2( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);
```

```
	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProcMess2;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= 0;
	wcex.lpszClassName	= szWindowClassMess2;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

	return RegisterClassEx(&wcex);
}

BOOL InitInstanceMess2( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{

  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd = CreateWindow(szWindowClassMess2, "ADC 2 Thermometer",WS_VISIBLE | WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMess2,
fensterkoordinaten.yMess2,fensterkoordinaten.nWidthMess2,fensterkoordinaten.nHeightMess2, hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  mess2hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcMess2(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch( message )
	{
		case WM_COMMAND:
			return DefWindowProc( hWnd, message, wParam, lParam );
		break;

		case WM_PAINT:
			drawkosy30kanal(hWnd, PAINTMSG,2); // zeichnet Kosy für die kleinen Fenster beim 30-Kanalgerät
		break;

		case WM_DESTROY:
		//	globalexit();
			//PostQuitMessage(0);
		break;

		default:
			return DefWindowProc( hWnd, message, wParam, lParam );
  }
```

```
  return 0;
}

ATOM MyRegisterClassMess3( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMess3;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMess3;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}

BOOL InitInstanceMess3( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
HWND hWnd;
        hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

        hWnd = CreateWindow(szWindowClassMess3, "ADC 3 Thermometer",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMess3,
fensterkoordinaten.yMess3,fensterkoordinaten.nWidthMess3,fensterkoordinaten.nHeightMess3, hwndParent,
NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  mess3hwnd=hWnd;
  UpdateWindow( hWnd );

  return TRUE;
}

LRESULT CALLBACK WndProcMess3(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {
                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosy30kanal(hWnd, PAINTMSG,3); // zeichnet Kosy für die kleinen
Fenster beim 30-Kanalgerät
                break;
```

```
                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMess4( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMess4;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMess4;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}


BOOL InitInstanceMess4( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd = CreateWindow(szWindowClassMess4, "ADC 4 Thermometer",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMess4,
fensterkoordinaten.yMess4,fensterkoordinaten.nWidthMess4,fensterkoordinaten.nHeightMess4, hwndParent,
NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  mess4hwnd=hWnd;
  UpdateWindow( hWnd );
  return TRUE;
}

LRESULT CALLBACK WndProcMess4(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {
```

```
                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosy30kanal(hWnd, PAINTMSG,4); // zeichnet Kosy für die kleinen
Fenster beim 30-Kanalgerät
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessStatus( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMessStatus;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMessStatus;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}


BOOL InitInstanceMessStatus( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd          =         CreateWindow(szWindowClassMessStatus,          "Status",WS_VISIBLE          |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessStatus,
fensterkoordinaten.yMessStatus,fensterkoordinaten.nWidthMessStatus,fensterkoordinaten.nHeightMessStatu
s, hwndParent, NULL , hInstance, NULL);


  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  messstatushwnd=hWnd;
  return TRUE;
```

```
}

LRESULT CALLBACK WndProcMessStatus(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch( message )
	{
		case WM_COMMAND:
			return DefWindowProc( hWnd, message, wParam, lParam );
		break;

		case WM_PAINT:
			 drawstatus(hWnd, PAINTMSG); // zeichnet Statusfenster
		break;

		case WM_DESTROY:
		//	globalexit();
			//PostQuitMessage(0);
		break;

		default:
			return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessHiRes1( HINSTANCE hInstance )
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProcMessHiRes1;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= 0;
	wcex.lpszClassName	= szWindowClassMessHiRes1;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

	return RegisterClassEx(&wcex);
}


BOOL InitInstanceMessHiRes1( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

   hWnd = CreateWindow(szWindowClassMessHiRes1, "Hochaufgelöst1",WS_VISIBLE | WS_OVERLAPPEDWINDOW,
     fensterkoordinaten.xMessHiRes1,
fensterkoordinaten.yMessHiRes1,fensterkoordinaten.nWidthMessHiRes1,fensterkoordinaten.nHeightMessHiRes1, hwndParent, NULL , hInstance, NULL);
```

```
  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  messhi1hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcMessHiRes1(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {

                case WM_CREATE :

                break;

                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosyhires(hWnd,PAINTMSG,1); // zeichnet Kosy für die HIRES
FENSTER
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassHiRes2( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcHiRes2;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassHiRes2;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}
```

```
BOOL InitInstanceMessHiRes2( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

   hWnd = CreateWindow(szWindowClassHiRes2, "Hochaufgelöst2",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessHiRes2,
fensterkoordinaten.yMessHiRes2,fensterkoordinaten.nWidthMessHiRes2,fensterkoordinaten.nHeightMessHi
Res2, hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  messhi2hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcHiRes2(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {

                case WM_CREATE :

                break;

                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosyhires(hWnd,PAINTMSG,2); // zeichnet Kosy für die HIRES
FENSTER
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessHiRes3( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);
```

```
	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProcMessHiRes3;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= 0;
	wcex.lpszClassName	= szWindowClassMessHiRes3;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

	return RegisterClassEx(&wcex);
}


BOOL InitInstanceMessHiRes3( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

   hWnd = CreateWindow(szWindowClassMessHiRes3, "Hochaufgelöst3",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessHiRes3,
fensterkoordinaten.yMessHiRes3,fensterkoordinaten.nWidthMessHiRes3,fensterkoordinaten.nHeightMessHi
Res3, hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  messhi3hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcMessHiRes3(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch( message )
	{

		case WM_CREATE :

		break;

		case WM_COMMAND:
			return DefWindowProc( hWnd, message, wParam, lParam );
		break;

		case WM_PAINT:
			drawkosyhires(hWnd,PAINTMSG,3); // zeichnet Kosy für die HIRES
FENSTER
		break;

		case WM_DESTROY:
		//	globalexit();
			//PostQuitMessage(0);
```

```
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessHiRes4( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMessHiRes4;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMessHiRes4;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}


BOOL InitInstanceMessHiRes4( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

   hWnd = CreateWindow(szWindowClassMessHiRes4, "Hochaufgelöst4",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessHiRes4,
fensterkoordinaten.yMessHiRes4,fensterkoordinaten.nWidthMessHiRes4,fensterkoordinaten.nHeightMessHi
Res4, hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  messhi4hwnd=hWnd;
  UpdateWindow( hWnd );
  return TRUE;
}

LRESULT CALLBACK WndProcMessHiRes4(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {
                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;
```

```
                case WM_PAINT:
                        drawkosyhires(hWnd,PAINTMSG,4); // zeichnet Kosy für die HIRES
FENSTER
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessASL( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC) WndProcASL;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassASL;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}


BOOL InitInstanceASL( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

   hWnd         =       CreateWindow(szWindowClassASL,          "ASL-F250",WS_VISIBLE           |
WS_OVERLAPPEDWINDOW,
     fensterkoordinaten.xMessASL,
fensterkoordinaten.yMessASL,fensterkoordinaten.nWidthMessASL,fensterkoordinaten.nHeightMessASL,
hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  messASLhwnd=hWnd;
  UpdateWindow( hWnd );
  return TRUE;
}

LRESULT CALLBACK WndProcASL(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
```

```
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch( message )
	{
		case WM_COMMAND:
			return DefWindowProc( hWnd, message, wParam, lParam );
		break;

		case WM_PAINT:
			drawkosy30kanal(hWnd,PAINTMSG,ASLPAINT);
		break;

		case WM_DESTROY:
		//	globalexit();
			//PostQuitMessage(0);
		break;

		default:
			return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessASLHiRes( HINSTANCE hInstance )
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProcMessHiResASL;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= 0;
	wcex.lpszClassName	= szWindowClassMessHiResASL;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

	return RegisterClassEx(&wcex);
}


BOOL InitInstanceMessHiResASL( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd = CreateWindow(szWindowClassMessHiResASL, "ASl-Hochaufgelöst",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessHiResASL,
fensterkoordinaten.yMessHiResASL,fensterkoordinaten.nWidthMessHiResASL,fensterkoordinaten.nHeight
MessHiResASL, hwndParent, NULL , hInstance, NULL);

	if( !hWnd )
	{
		return FALSE;
	}
```

```
        ShowWindow( hWnd, SW_SHOW);
        messhiASLhwnd=hWnd;
        UpdateWindow( hWnd );
        return TRUE;
}

LRESULT CALLBACK WndProcMessHiResASL(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {
                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosyhires(hWnd,PAINTMSG,ASLPAINT); // zeichnet Kosy für die HIRES
FENSTER
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

/****************************************************************************
* Dialog Funkion für das Starten der Messung  30-Kanal Kalibrierung         *
****************************************************************************/
BOOL FAR PASCAL StartDialogkanal30Ut(HWND hDlg, WORD msg, WPARAM wParam, LPARAM
lParam)
{
char dummy[100];

   switch(msg)
   {
                case WM_INITDIALOG :
                        sprintf(dummy,"%.0lf",messkontrolle.messdauer);//Ausgabe der Messdauer
        SetDlgItemText(hDlg,IDC_MESSDAUER,dummy);

                        if(messkontrolle.asl==TRUE)
                                CheckDlgButton(hDlg,IDC_CHECKASL,1);
                        else
                                CheckDlgButton(hDlg,IDC_CHECKASL,0);

      break;

                case WM_COMMAND :
        switch(wParam)
        {
          case IDOK:
            GetDlgItemText(hDlg,IDC_MESSDAUER,dummy,99);
            messkontrolle.messdauer=atof(dummy);//Messdauer holen
```

```
            if(messkontrolle.messdauer<=0)
            {
               MessageBox(hDlg, "Für Messdauer nur Werte größer 0 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
               return FALSE;
            }
                                          else if(messkontrolle.messdauer>MAXMESSDAUER)
            {
               MessageBox(hDlg, "Für Messdauer nur Werte kleiner als 360000 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
               return FALSE;
            }


         if(IsDlgButtonChecked(hDlg,IDC_CHECKASL)==BST_CHECKED)
                                                messkontrolle.asl=TRUE;
                                          else
                                                messkontrolle.asl=FALSE;


            EndDialog(hDlg,TRUE);
            return TRUE;
          break;

          case IDCANCEL :
            EndDialog(hDlg,FALSE);
            return TRUE;
          break;

        }
      break;

   }

  return(FALSE);

}

// Nachrichtenbehandlungsroutine für "Angeschlossene Fühler"-Feld.
LRESULT CALLBACK Kanal30CeckDialog( HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam )
{
char dummy[100];
double dummy1[8],dummy2[8],dummy3[8],dummy4[8];
int i=0;

         switch( message )
         {
                  case WM_INITDIALOG:
                           return TRUE;

                  case WM_COMMAND:
                     if (LOWORD(wParam) == IDCANCEL)
                     {
                           EndDialog(hDlg, FALSE );
                           return TRUE;
                     }
                     else if(LOWORD(wParam) == IDUPDATE)
                     {
                           ledmessen(1,defmessparam.ba);
                           ads1241klarmachen(defmessparam.ba);
                           startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts
```

```
                                for(i=0;i<8;i++)
                                {

        ads1241getu30(defmessparam.ba,1,i+1,&dummy1[i],&dummy2[i],&dummy3[i],&dummy4[i]);       //
auslesen der Kanäle
                                }
                                ledmessen(0,defmessparam.ba);
                                messkontrolle.adc[1]=0;
                                messkontrolle.adc[2]=0;
                                messkontrolle.adc[3]=0;
                                messkontrolle.adc[4]=0;

                                for(i=0;i<=8;i++) // Kanäle zurücksetzen
                                {
                                        messkontrolle.kanal[i]=0;
                                }

                                for(i=0;i<8;i++) // welche Kanäle sind belegt, welche Wandler werden
verwendet
                                {
                                        if(dummy1[i]<0.01)
                                                messkontrolle.flagkanal30[i*4+1]=FALSE;
                                        else
                                        {
                                                messkontrolle.adc[1]=messkontrolle.adc[1]++;
                                                messkontrolle.flagkanal30[i*4+1]=TRUE;

        messkontrolle.kanal[i+1]=messkontrolle.kanal[i+1]++;
                                        }

                                        if(dummy4[i]<0.01)
                                                messkontrolle.flagkanal30[i*4+2]=FALSE;
                                        else
                                        {
                                                messkontrolle.flagkanal30[i*4+2]=TRUE;
                                                messkontrolle.adc[2]=messkontrolle.adc[2]++;

        messkontrolle.kanal[i+1]=messkontrolle.kanal[i+1]++;
                                        }

                                        if(dummy3[i]<0.01)
                                                messkontrolle.flagkanal30[i*4+3]=FALSE;
                                        else
                                        {
                                                messkontrolle.adc[3]=messkontrolle.adc[3]++;
                                                messkontrolle.flagkanal30[i*4+3]=TRUE;

        messkontrolle.kanal[i+1]=messkontrolle.kanal[i+1]++;
                                        }
                                        if(dummy2[i]<0.01)
                                                messkontrolle.flagkanal30[i*4+4]=FALSE;
                                        else
                                        {
                                                messkontrolle.adc[4]=messkontrolle.adc[4]++;
                                                messkontrolle.flagkanal30[i*4+4]=TRUE;

        messkontrolle.kanal[i+1]=messkontrolle.kanal[i+1]++;
                                        }
                                }

                                if(messkontrolle.flagkanal30[1]==TRUE)
                                        sprintf(dummy,"belegt");
                                else
```

```
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal1, dummy);

if(messkontrolle.flagkanal30[2]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal2, dummy);

if(messkontrolle.flagkanal30[3]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal3, dummy);

if(messkontrolle.flagkanal30[4]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal4, dummy);

if(messkontrolle.flagkanal30[5]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal5, dummy);

if(messkontrolle.flagkanal30[6]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal6, dummy);

if(messkontrolle.flagkanal30[7]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal7, dummy);

if(messkontrolle.flagkanal30[8]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal8, dummy);

if(messkontrolle.flagkanal30[9]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal9, dummy);

if(messkontrolle.flagkanal30[10]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal10, dummy);

if(messkontrolle.flagkanal30[11]==TRUE)
            sprintf(dummy,"belegt");
else
            sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal11, dummy);
```

```
if(messkontrolle.flagkanal30[12]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal12, dummy);

if(messkontrolle.flagkanal30[13]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal13, dummy);

if(messkontrolle.flagkanal30[14]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal14, dummy);

if(messkontrolle.flagkanal30[15]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal15, dummy);

if(messkontrolle.flagkanal30[16]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal16, dummy);

if(messkontrolle.flagkanal30[17]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal17, dummy);

if(messkontrolle.flagkanal30[18]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal18, dummy);

if(messkontrolle.flagkanal30[19]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal19, dummy);

if(messkontrolle.flagkanal30[20]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal20, dummy);

if(messkontrolle.flagkanal30[21]==TRUE)
        sprintf(dummy,"belegt");
else
        sprintf(dummy,"frei");
SetDlgItemText(hDlg,IDC_UKanal21, dummy);

if(messkontrolle.flagkanal30[22]==TRUE)
        sprintf(dummy,"belegt");
else
```

```
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal22, dummy);

                                    if(messkontrolle.flagkanal30[23]==TRUE)
                                        sprintf(dummy,"belegt");
                                    else
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal23, dummy);

                                    if(messkontrolle.flagkanal30[24]==TRUE)
                                        sprintf(dummy,"belegt");
                                    else
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal24, dummy);

                                    if(messkontrolle.flagkanal30[25]==TRUE)
                                        sprintf(dummy,"belegt");
                                    else
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal25, dummy);

                                    if(messkontrolle.flagkanal30[26]==TRUE)
                                        sprintf(dummy,"belegt");
                                    else
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal26, dummy);

                                    if(messkontrolle.flagkanal30[27]==TRUE)
                                        sprintf(dummy,"belegt");
                                    else
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal27, dummy);

                                    if(messkontrolle.flagkanal30[28]==TRUE)
                                        sprintf(dummy,"belegt");
                                    else
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal28, dummy);

                                    if(messkontrolle.flagkanal30[29]==TRUE)
                                        sprintf(dummy,"belegt");
                                    else
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal29, dummy);

                                    if(messkontrolle.flagkanal30[30]==TRUE)
                                        sprintf(dummy,"belegt");
                                    else
                                        sprintf(dummy,"frei");
                                    SetDlgItemText(hDlg,IDC_UKanal30, dummy);

                                }
                                else if(LOWORD(wParam) == IDOK)
                                {
                                        EndDialog(hDlg,TRUE);
            return TRUE;
                                }
                                break;
            }
    return FALSE;
}

int save30ut(HWND hWnd)
```

```
{
static char  szFilename[MAX_PATH] = ".txt";
int iErg=0,j=0;
char *feld,zeile[ZEILENLENG],dummy[200];
long int i;
HGLOBAL hMemHandleDaten;
HFILE hFile;
unsigned long lBytes;
OPENFILENAME ofn;

        hMemHandleDaten=GlobalAlloc(GHND,sizeof(char)*536*MAXMESSDAUER);  //Größe  des
Charfeldes
        feld=(char*)GlobalLock(hMemHandleDaten);

        memset(&ofn, 0, sizeof(OPENFILENAME));
        ofn.lpstrFile = szFilename;
        iErg = FileDialogSave1ut(hWnd, &ofn);

        if(iErg == IDOK)
        {
                if((hFile = _lcreat(ofn.lpstrFile,0)) == HFILE_ERROR)
                {
                        MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK);
                        SendMessage(hWnd,WM_CLOSE,NULL,NULL);
                        GlobalUnlock(hMemHandleDaten);
                        GlobalFree(hMemHandleDaten);
                        return(FEHLER);
                }

                // an den Dateianfang gehen
                _llseek(hFile, 0L, 0);

                // Datei schreiben

                for(i=0;i<messkontrolle.anzahl;i++)
                {
                        zeile[0]=0;
                        for(j=0;j<32;j++)
                        {
                                sprintf(dummy,"%.0lf                                                %.8lf
",messdaten30[i].t[j],messdaten30[i].ywerte[j]);
                                strcat(zeile,dummy);
                        }
                        if(messkontrolle.asl==TRUE)
                        {
                                sprintf(dummy,"%.0lf %.8lf ",messdaten30[i].aslzeit,messdaten30[i].asl);
                                strcat(zeile,dummy);
                        }
                        sprintf(dummy,"\r\n");
                        strcat(zeile,dummy);

                        lBytes = _hwrite(hFile,zeile,strlen(zeile));
                        if(lBytes!=strlen(zeile))
                        {
                                MessageBox(hWnd,  "Da  ist  was  danebengegangen",  "Achtung",
MB_OK|MB_ICONSTOP);
                                _lclose(hFile);
                        GlobalUnlock(hMemHandleDaten);
                GlobalFree(hMemHandleDaten);
                                return(FEHLER);
                        }
                        else
                        {
```

```
                                        //_llseek(hFile, lBytes, 1);
                                }
                        }
                        _lclose(hFile);
                }

                GlobalUnlock(hMemHandleDaten);
                GlobalFree(hMemHandleDaten);
                return(OK);
}

void kanal30TUschreiber(HWND hWnd) // startet die Messung mit dem 30-Kanalgerät
{
double TimeUnit;
int j=0,k,iErg,readcount=0;
long int i=0;
OPENFILENAME ofn;
static char  szFilename[MAX_PATH] = ".kal";
HFILE hFile;
char readdata[300],read1,pcCommPort[5];
DCB dcb;


                TimeInit(&TimeUnit);

                // Fenster zu
                SendMessage(mess1hwnd,WM_CLOSE,0,0);
                SendMessage(mess2hwnd,WM_CLOSE,0,0);
                SendMessage(mess3hwnd,WM_CLOSE,0,0);
                SendMessage(mess4hwnd,WM_CLOSE,0,0);
                SendMessage(messstatushwnd,WM_CLOSE,0,0);
                SendMessage(messhi1hwnd,WM_CLOSE,0,0);
                SendMessage(messhi2hwnd,WM_CLOSE,0,0);
                SendMessage(messhi3hwnd,WM_CLOSE,0,0);
                SendMessage(messhi4hwnd,WM_CLOSE,0,0);

                SendMessage(lfmess1hwnd,WM_CLOSE,0,0);
                SendMessage(lfmess2hwnd,WM_CLOSE,0,0);
                SendMessage(lfmess3hwnd,WM_CLOSE,0,0);
                SendMessage(lfmess4hwnd,WM_CLOSE,0,0);
                SendMessage(lfmesshi1hwnd,WM_CLOSE,0,0);
                SendMessage(lfmesshi2hwnd,WM_CLOSE,0,0);
                SendMessage(lfmesshi3hwnd,WM_CLOSE,0,0);
                SendMessage(lfmesshi4hwnd,WM_CLOSE,0,0);

                if(DialogBox(hInst, (LPCTSTR)IDD_30KANALUT, hWnd, (DLGPROC)
StartDialogkanal30Ut)==FALSE)
                {
                        messkontrolle.messtyp=NIXMESS;
                        warten(0.5,TimeUnit); //0.5s warten
                        return; // bei Abbruch zurück
                }

                if(DialogBox(hInst, (LPCTSTR)IDD_30KANALCHECK, hWnd, (DLGPROC)
Kanal30CeckDialog)==FALSE)
                {
                        messkontrolle.messtyp=NIXMESS;
                        warten(0.5,TimeUnit); //0.5s warten
                        return; // bei Abbruch zurück
                }

                if(messkontrolle.asl==TRUE) // Schnittstelle des ASL klarmachen
                {
```

```
                sprintf(pcCommPort,"COM%i",messkontrolle.aslcom);
                messkontrolle.hComasl = CreateFile( pcCommPort, GENERIC_READ |
GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für ASL

                dcb.ByteSize = 28;            // data size, xmit, and rcv
                dcb.BaudRate = CBR_9600;     // set the baud rate
                dcb.fBinary=1;                                  // binary mode, no EOF check
                dcb.fParity=0;              // enable parity checking
                dcb.fOutxCtsFlow=0;                     // CTS output flow control
                dcb.fOutxDsrFlow=0;                     // DSR output flow control
                dcb.fDtrControl=1;                      // DTR flow control type
                dcb.fDsrSensitivity=0;       // DSR sensitivity
                dcb.fTXContinueOnXoff=1; // XOFF continues Tx
                dcb.fOutX=0;           // XON/XOFF out flow control
                dcb.fInX=0;            // XON/XOFF in flow control
                dcb.fErrorChar=0;       // enable error replacement
                dcb.fNull=0;           // enable null stripping
                dcb.fRtsControl=1;       // RTS flow control
                dcb.fAbortOnError=0;     // abort reads/writes on error
                dcb.XonLim=2048;              // transmit XON threshold
                dcb.XoffLim=512;              // transmit XOFF threshold
                dcb.ByteSize=8;            // number of bits/byte, 4-8
                dcb.Parity=NOPARITY;              // 0-4=no,odd,even,mark,space
                dcb.StopBits=TWOSTOPBITS;            // 0,1,2 = 1, 1.5, 2
                dcb.XonChar=17;              // Tx and Rx XON character
                dcb.XoffChar=19;             // Tx and Rx XOFF character
                dcb.ErrorChar=0;            // error replacement character
                dcb.EofChar=0;              // end of input character
                dcb.EvtChar=0;              // received event character


                SetCommState(messkontrolle.hComasl, &dcb);
                PurgeComm(messkontrolle.hComasl,PURGE_TXCLEAR&&PURGE_RXCLEAR);     //
Puffer leeren
                setasl(); // stellt ASL ein
        }

        // Kalibrierdaten einlesen
        if(messkontrolle.messtyp==KANAL30TT)
        {
                memset(&ofn, 0, sizeof(OPENFILENAME));
                ofn.lpstrFile = szFilename;
                iErg = FileDialogKal(hWnd, &ofn);

                if(iErg == IDOK)
                {
                        //Datei öffnen mit Fehlertest
                        if((hFile = _lopen(ofn.lpstrFile, OF_READ)) == -1)
                        {
                                MessageBox(hWnd, "Fehler beim Dateneinlesen", "Achtung",
MB_OK|MB_ICONSTOP);
                                SendMessage(hWnd,WM_CLOSE,NULL,NULL);
                                messkontrolle.messtyp=NIXMESS;
                                warten(0.5,TimeUnit); //0.5s warten
                                CloseHandle(messkontrolle.hComasl); // Schnittstelle wieder zu
                                return;
                        }

                        //an den Dateianfang gehen
                        _llseek(hFile, 0L, 0);

                        // Datei lesen,  Fehlerprüfung hinzufügen
```

```
                        for(k=0;k<32;k++)
                        {
                                readcount=0;
                                do //eine Zeile einlesen
                                {
                                        _hread(hFile,&read1,1);
                                        readdata[readcount++]=read1;
                                }
                                while(readdata[readcount-1]!=13 ); //Zeilenende

                                readdata[readcount-1]=0; //String beenden
                                _hread(hFile,&read1,1); // noch Zeilenvorschub einlesen

                                sscanf(readdata,"%le %le %le %le %le
",&messkontrolle.kalparam[k].a,&messkontrolle.kalparam[k].b,&messkontrolle.kalparam[k].c,&messkontrolle.
kalparam[k].uref,&messkontrolle.kalparam[k].r1);
                        }

                        _lclose(hFile);

                }
                else
                {
                        messkontrolle.messtyp=NIXMESS;
                        warten(0.5,TimeUnit); //1s warten
                        CloseHandle(messkontrolle.hComasl); // Schnittstelle wieder  schließen
                        return;
                }
        }

        messkontrolle.messan=TRUE; // eine Messung läuft

        if(messkontrolle.messtyp==KANAL30TU)
        {
                messkontrolle.maximalerywert=UREF;
                messkontrolle.minmalerywert=0;
        }
        else if(messkontrolle.messtyp==KANAL30TT)
        {
                messkontrolle.maximalerywert=MAXTEMPERATUR;
                messkontrolle.minmalerywert=MINTEMPERATUR ;
        }

        MoveWindow(hWnd,fensterkoordinaten.xMessMain,fensterkoordinaten.yMessMain,fensterkoordina
ten.nMessWidthMain,fensterkoordinaten.nMessHeighMain,TRUE); // Hauptfenster neu skalieren
        InitInstanceMess1(hInst,SW_SHOWNORMAL,hWnd); // Nebenfenster darstellen
        InitInstanceMess2(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess3(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess4(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessStatus(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes1(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes2(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes3(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes4(hInst,SW_SHOWNORMAL,hWnd);
        if(messkontrolle.asl==TRUE)
        {
                InitInstanceASL(hInst,SW_SHOWNORMAL,hWnd);
                InitInstanceMessHiResASL(hInst,SW_SHOWNORMAL,hWnd);
        }
        setmenue(hWnd); // Menüs passend einstellen

        messkontrolle.count=0;
        for(k=0;k<32;k++) // zurücksetzen
```

```
	{
		messkontrolle.letztewertex[k]=0;
	}
	messkontrolle.letztewerteasl=0;

	ledmessen(1,defmessparam.ba);
	if(messkontrolle.asl==TRUE)
	{
		getasltemp(); // ersten Wert auslesen und verwerfen.
	}
	startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts
	ads1241klarmachen(defmessparam.ba);

	timeSetEvent(100,50,timerroutine,0,TIME_ONESHOT); // Timer an
}

void CALLBACK timerroutine(UINT uID,UINT uMsg,DWORD dwUser,DWORD dw1,DWORD dw2)
{
	SendMessage(mainhwnd,WM_TIMERHGS,0,0);
	return;
}


void kanal30TUschreiberaus(HWND hWnd) // beendet die 30-Kanal Messung
{
	messkontrolle.messan=FALSE; //keine Messung läuft
	messkontrolle.messtyp=NIXMESS;
	setmenue(hWnd); // Menüs passend einstellen

}

void kanalmess30(HWND hWnd) // führt die einzelnen Messschritte durch
{
static int i=0,spikecount[9];
long int messzeit=0;
double dummy1,dummy2,dummy3,dummy4,spike=-273.15;

	ads1241getu30(defmessparam.ba,1,i+1,&dummy1,&dummy2,&dummy3,&dummy4);		//
Spannungen einlesen
	messzeit=gettime(defmessparam.ba);

	messdaten30[messkontrolle.count].t[4*i]=messzeit; // in die entspechenden Speicherbereiche legen
	messdaten30[messkontrolle.count].t[4*i+1]=messzeit;
	messdaten30[messkontrolle.count].t[4*i+2]=messzeit;
	messdaten30[messkontrolle.count].t[4*i+3]=messzeit;
	messkontrolle.letztewertex[4*i]=messzeit;
	messkontrolle.letztewertex[4*i+1]=messzeit;
	messkontrolle.letztewertex[4*i+2]=messzeit;
	messkontrolle.letztewertex[4*i+3]=messzeit;
	messkontrolle.messzeit=messzeit;

	if(messkontrolle.messtyp==KANAL30TU)
	{
		// Spikes rausfiltern; nur Kanal 1 ist betroffen
		spike=dummy1;
		if(messkontrolle.count-1>0)
		{
			if((fabs(messdaten30[messkontrolle.count-1].ywerte[4*i]-spike)>0.5)		&&
(spikecount[i]<1)) // Spike aufgetreten
			{
				spike=messdaten30[messkontrolle.count-1].ywerte[4*i];
				spikecount[i]=1; // Maximal nur daraufolgenden Messwert entfernen
			}
```

```
                        else
                                spikecount[i]=0;
                }

                messdaten30[messkontrolle.count].ywerte[4*i]=spike;
                messdaten30[messkontrolle.count].ywerte[4*i+1]=dummy4;
                messdaten30[messkontrolle.count].ywerte[4*i+2]=dummy3;
                messdaten30[messkontrolle.count].ywerte[4*i+3]=dummy2;
                messkontrolle.letztewerte[4*i]=spike; // Werte für Status und aktueller Wert
                messkontrolle.letztewerte[4*i+1]=dummy4;
                messkontrolle.letztewerte[4*i+2]=dummy3;
                messkontrolle.letztewerte[4*i+3]=dummy2;
        }
        else if(messkontrolle.messtyp==KANAL30TT)
        {

                // Spikes rausfiltern; nur Kanal 1 ist betroffen
                spike=maketemp(dummy1,4*i+1);
                if(messkontrolle.count-1>0)
                {
                        if((fabs(messdaten30[messkontrolle.count-1].ywerte[4*i]-spike)>(double)5)      &&
(spikecount[i]<1)) // Spike aufgetreten
                        {
                                spike=messdaten30[messkontrolle.count-1].ywerte[4*i];
                                spikecount[i]=1; // Maximal nur daraufolgenden Messwert entfernen
                        }
                        else
                                spikecount[i]=0;
                }

                messdaten30[messkontrolle.count].ywerte[4*i]=spike;
                messdaten30[messkontrolle.count].ywerte[4*i+1]=maketemp(dummy4,4*i+2);
                messdaten30[messkontrolle.count].ywerte[4*i+2]=maketemp(dummy3,4*i+3);
                messdaten30[messkontrolle.count].ywerte[4*i+3]=maketemp(dummy2,4*i+4);
                messkontrolle.letztewerte[4*i]=spike; // Werte für Status und aktueller Wert
                messkontrolle.letztewerte[4*i+1]=maketemp(dummy4,4*i+2);
                messkontrolle.letztewerte[4*i+2]=maketemp(dummy3,4*i+3);
                messkontrolle.letztewerte[4*i+3]=maketemp(dummy2,4*i+4);
        }

        if(i<7)// Kanäle durchschalten
        {
                i++;
        }
        else // achter Kanal erreicht, wieder von vorne und asl messen
        {
                if(messkontrolle.asl==TRUE)// bei Messung mit ASL
                {
                        messdaten30[messkontrolle.count].asl=getasltemp()-273.15;
                        messdaten30[messkontrolle.count].aslzeit=messzeit;
                        messkontrolle.letztewerteaslx=messzeit;
                        messkontrolle.letztewerteasl=messdaten30[messkontrolle.count].asl;
                        SendMessage(messASLhwnd,WM_PAINT,0,0);
                        SendMessage(messhiASLhwnd,WM_PAINT,0,0);
                }
                saveuntermessung30(hWnd,messkontrolle.count); // speichert Messdaten während der
Messung
                messkontrolle.count++; // Datenpunktzähler weiterzählen
                messkontrolle.anzahl=messkontrolle.count;
                i=0;
        }

        SendMessage(mess1hwnd,WM_PAINT,0,0); // Graphen zeichnen
```

```
	SendMessage(mess2hwnd,WM_PAINT,0,0);
	SendMessage(mess3hwnd,WM_PAINT,0,0);
	SendMessage(mess4hwnd,WM_PAINT,0,0);
	SendMessage(messstatushwnd,WM_PAINT,0,0);
	SendMessage(messhi1hwnd,WM_PAINT,0,0);
	SendMessage(messhi2hwnd,WM_PAINT,0,0);
	SendMessage(messhi3hwnd,WM_PAINT,0,0);
	SendMessage(messhi4hwnd,WM_PAINT,0,0);
	SendMessage(mainhwnd,WM_PAINT,0,0);

	if(messzeit>=messkontrolle.messdauer || (GetAsyncKeyState('E') != 0)) // Messdauer
überschritten oder Messung abbrechen
	{
		messkontrolle.messan=FALSE; //keine Messung läuft
		clockoff(defmessparam.ba);
		GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen
werden, um das Speichern nicht zu stören
		messkontrolle.anzahl=messkontrolle.count;
		ledmessen(0,defmessparam.ba);
		save30ut(hWnd);
		setmenue(hWnd); // Menüs passend einstellen
		CloseHandle(messkontrolle.hComasl); // Schnittstelle wieder schließen
	}
	else
	{
		timeSetEvent(100,50,timerroutine,0,TIME_ONESHOT);
	}
}

void saveuntermessung30(HWND hWnd, long int i) // speichert Messdaten während der Messung
{
HANDLE hFile;
char zeile[ZEILENLENG+100],dummy[400];
//unsigned long dummy;
int j;
DWORD dummy2;

	zeile[0]=0;
	hFile=CreateFile(defmessparam.tempfile,GENERIC_WRITE,FILE_SHARE_READ,NULL,OPE
N_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);

	// an das Dateiende gehen
	SetFilePointer(hFile,0,NULL,FILE_END);

	for(j=0;j<32;j++)
	{
		sprintf(dummy,"%.0lf %.8lf ",messdaten30[i].t[j],messdaten30[i].ywerte[j]);
		strcat(zeile,dummy);
	}
	if(messkontrolle.asl==TRUE)
	{
		sprintf(dummy,"%.0lf %.8lf ",messdaten30[i].aslzeit,messdaten30[i].asl);
		strcat(zeile,dummy);
	}
	sprintf(dummy,"\r\n");
	strcat(zeile,dummy);

	// Datei schreiben

	WriteFile(hFile,zeile,strlen(zeile),&dummy2,NULL);

	CloseHandle(hFile);
```

```
	return;
}
```

11.7.2.16 mess30thermolf.cpp

```
#include "stdafx.h"

#include "resource.h"

#define MAX_LOADSTRING 100

TCHAR szTitleMess1lf[MAX_LOADSTRING];
TCHAR szTitleMess2lf[MAX_LOADSTRING];
TCHAR szTitleMess3lf[MAX_LOADSTRING];
TCHAR szTitleMess4lf[MAX_LOADSTRING];
TCHAR szTitleMessStatuslf[MAX_LOADSTRING];
TCHAR szTitleMessHiRes1lf[MAX_LOADSTRING];
TCHAR szTitleMessHiRes2lf[MAX_LOADSTRING];
TCHAR szTitleMessHiRes3lf[MAX_LOADSTRING];
TCHAR szTitleMessHiRes4lf[MAX_LOADSTRING];
TCHAR szTitleMessASLlf[MAX_LOADSTRING];
TCHAR szTitleMessHiResASLlf[MAX_LOADSTRING];

TCHAR szWindowClassMess1lf[MAX_LOADSTRING];
TCHAR szWindowClassMess2lf[MAX_LOADSTRING];
TCHAR szWindowClassMess3lf[MAX_LOADSTRING];
TCHAR szWindowClassMess4lf[MAX_LOADSTRING];
TCHAR szWindowClassMessStatuslf[MAX_LOADSTRING];
TCHAR szWindowClassMessHiRes1lf[MAX_LOADSTRING];
TCHAR szWindowClassHiRes2lf[MAX_LOADSTRING];
TCHAR szWindowClassMessHiRes3lf[MAX_LOADSTRING];
TCHAR szWindowClassMessHiRes4lf[MAX_LOADSTRING];

HWND
lfmess1hwnd,lfmess2hwnd,lfmess3hwnd,lfmess4hwnd,lfmessstatushwnd,lfmesshi1hwnd,lfmesshi2hwnd,lfme
sshi3hwnd,lfmesshi4hwnd; // Fensterhandle

extern HWND mainhwnd;
extern MESSSTEUERSTRUCT defmessparam;
extern HINSTANCE hInst;
extern FENSTERKOORDSSTRUCT fensterkoordinaten;
extern REIHENSTEUERSTRUCT messkontrolle;
extern TIMECAPS multmediatimertc;
extern UINT multmediatimerwTimerRes;
extern                                                                                HWND
mess1hwnd,mess2hwnd,mess3hwnd,mess4hwnd,messstatushwnd,messhi1hwnd,messhi2hwnd,messhi3hwnd,
messhi4hwnd; // Fensterhandle
extern HWND messASLhwnd, messhiASLhwnd;
extern DATENSATZ30 * messdaten30;

ATOM MyRegisterClassMess1lf( HINSTANCE hInstance )
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProcMess1lf;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
```

```
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= 0;
	wcex.lpszClassName	= szWindowClassMess1lf;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

	return RegisterClassEx(&wcex);
}

BOOL InitInstanceMess1lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
HWND hWnd;

	SetLastError(0);
	hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

	hWnd = CreateWindow(szWindowClassMess1lf, "ADC 1 Leitfähigkeit",WS_VISIBLE
|WS_SYSMENU | WS_OVERLAPPED | WS_CAPTION| WS_THICKFRAME | WS_MINIMIZEBOX
| WS_MAXIMIZEBOX ,
    fensterkoordinaten.xMess1lf,
fensterkoordinaten.yMess1lf,fensterkoordinaten.nWidthMess1lf,fensterkoordinaten.nHeightMess1lf,
hwndParent, NULL , hInstance, NULL);

	if( !hWnd )
	{
		return FALSE;
	}
	ShowWindow( hWnd, SW_SHOW);

	UpdateWindow( hWnd );
	lfmess1hwnd=hWnd;
	return TRUE;
}

LRESULT CALLBACK WndProcMess1lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch( message )
	{

		case WM_CREATE :
			break;

		case WM_COMMAND:
			return DefWindowProc( hWnd, message, wParam, lParam );
		break;

		case WM_PAINT:
			drawkosy30kanal(hWnd, PAINTMSG,5); // zeichnet Kosy für die kleinen
Fenster beim 30-Kanalgerät
		break;

		case WM_DESTROY:
		//	globalexit();
			//PostQuitMessage(0);
		break;

		default:
			return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
```

```
}

ATOM MyRegisterClassMess2lf( HINSTANCE hInstance )
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProcMess2lf;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= 0;
	wcex.lpszClassName	= szWindowClassMess2lf;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

	return RegisterClassEx(&wcex);
}

BOOL InitInstanceMess2lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{

  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd = CreateWindow(szWindowClassMess2lf, "ADC 2 Leitfähigkeit",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMess2lf,
fensterkoordinaten.yMess2lf,fensterkoordinaten.nWidthMess2lf,fensterkoordinaten.nHeightMess2lf,
hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  lfmess2hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcMess2lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch( message )
	{
		case WM_COMMAND:
			return DefWindowProc( hWnd, message, wParam, lParam );
		break;

		case WM_PAINT:
			drawkosy30kanal(hWnd, PAINTMSG,6); // zeichnet Kosy für die kleinen
Fenster beim 30-Kanalgerät
		break;
```

```
                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMess3lf( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMess3lf;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMess3lf;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}

BOOL InitInstanceMess3lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
HWND hWnd;
        hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

        hWnd = CreateWindow(szWindowClassMess3lf, "ADC 3 Leitfähigkeit",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMess3lf,
fensterkoordinaten.yMess3lf,fensterkoordinaten.nWidthMess3lf,fensterkoordinaten.nHeightMess3lf,
hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  lfmess3hwnd=hWnd;
  UpdateWindow( hWnd );

  return TRUE;
}

LRESULT CALLBACK WndProcMess3lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];

LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {
```

```
                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosy30kanal(hWnd, PAINTMSG,7); // zeichnet Kosy für die kleinen
Fenster beim 30-Kanalgerät
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMess4lf( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMess4lf;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMess4lf;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}


BOOL InitInstanceMess4lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd = CreateWindow(szWindowClassMess4lf, "ADC 4 Leitfähigkeit",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMess4lf,
fensterkoordinaten.yMess4lf,fensterkoordinaten.nWidthMess4lf,fensterkoordinaten.nHeightMess4lf,
hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  lfmess4hwnd=hWnd;
  UpdateWindow( hWnd );
  return TRUE;
}
```

```
LRESULT CALLBACK WndProcMess4lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {
                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosy30kanal(hWnd, PAINTMSG,8); // zeichnet Kosy für die kleinen
Fenster beim 30-Kanalgerät
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessHiRes1lf( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMessHiRes1lf;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMessHiRes1lf;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}


BOOL InitInstanceMessHiRes1lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

   hWnd = CreateWindow(szWindowClassMessHiRes1lf, "Hochaufgelöst1 LF",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessHiRes1lf,
fensterkoordinaten.yMessHiRes1lf,fensterkoordinaten.nWidthMessHiRes1lf,fensterkoordinaten.nHeightMess
HiRes1lf, hwndParent, NULL , hInstance, NULL);
```

```
  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  lfmesshi1hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcMessHiRes1lf(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {

                case WM_CREATE :

                break;

                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosyhires(hWnd,PAINTMSG,5); // zeichnet Kosy für die HIRES
FENSTER
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassHiRes2lf( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcHiRes2lf;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassHiRes2lf;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}
```

```
BOOL InitInstanceMessHiRes2lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

   hWnd = CreateWindow(szWindowClassHiRes2lf, "Hochaufgelöst2 LF",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessHiRes2lf,
fensterkoordinaten.yMessHiRes2lf,fensterkoordinaten.nWidthMessHiRes2lf,fensterkoordinaten.nHeightMess
HiRes2lf, hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  lfmesshi2hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcHiRes2lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {

                case WM_CREATE :

                break;

                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;

                case WM_PAINT:
                        drawkosyhires(hWnd,PAINTMSG,6); // zeichnet Kosy für die HIRES
FENSTER
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessHiRes3lf( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);
```

```
	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProcMessHiRes3lf;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= 0;
	wcex.lpszClassName	= szWindowClassMessHiRes3lf;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

	return RegisterClassEx(&wcex);
}


BOOL InitInstanceMessHiRes3lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

  hWnd = CreateWindow(szWindowClassMessHiRes3lf, "Hochaufgelöst3 LF",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessHiRes3lf,
fensterkoordinaten.yMessHiRes3lf,fensterkoordinaten.nWidthMessHiRes3lf,fensterkoordinaten.nHeightMess
HiRes3lf, hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  UpdateWindow( hWnd );
  lfmesshi3hwnd=hWnd;
  return TRUE;
}

LRESULT CALLBACK WndProcMessHiRes3lf(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch( message )
	{

		case WM_CREATE :

		break;

		case WM_COMMAND:
			return DefWindowProc( hWnd, message, wParam, lParam );
		break;

		case WM_PAINT:
			drawkosyhires(hWnd,PAINTMSG,7); // zeichnet Kosy für die HIRES
FENSTER
		break;

		case WM_DESTROY:
		//	globalexit();
			//PostQuitMessage(0);
```

```
                    break;

                    default:
                            return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

ATOM MyRegisterClassMessHiRes4lf( HINSTANCE hInstance )
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProcMessHiRes4lf;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_MY8KANAL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = 0;
        wcex.lpszClassName      = szWindowClassMessHiRes4lf;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_ICONKLEIN);

        return RegisterClassEx(&wcex);
}


BOOL InitInstanceMessHiRes4lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent)
{
  HWND hWnd;

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variable speichern

   hWnd = CreateWindow(szWindowClassMessHiRes4lf, "Hochaufgelöst4 LF",WS_VISIBLE |
WS_OVERLAPPEDWINDOW,
    fensterkoordinaten.xMessHiRes4lf,
fensterkoordinaten.yMessHiRes4lf,fensterkoordinaten.nWidthMessHiRes4lf,fensterkoordinaten.nHeightMess
HiRes4lf, hwndParent, NULL , hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }
  ShowWindow( hWnd, SW_SHOW);
  lfmesshi4hwnd=hWnd;
  UpdateWindow( hWnd );
  return TRUE;
}

LRESULT CALLBACK WndProcMessHiRes4lf(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {
                case WM_COMMAND:
                        return DefWindowProc( hWnd, message, wParam, lParam );
                break;
```

```
                case WM_PAINT:
                        drawkosyhires(hWnd,PAINTMSG,8); // zeichnet Kosy für die HIRES
FENSTER
                break;

                case WM_DESTROY:
                //      globalexit();
                        //PostQuitMessage(0);
                break;

                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

void kanal30TundLF(HWND hWnd) // startet die Messung mit dem 30-Kanal Thermometer und
Leitfähigkeitsmessgerät
{
double TimeUnit;
int j=0,k,iErg,readcount=0;
long int i=0;
OPENFILENAME ofn;
static char  szFilename[MAX_PATH] = ".kal";
HFILE hFile;
char readdata[300],read1,pcCommPort[5];
DCB dcb;

        TimeInit(&TimeUnit);

        SendMessage(mess1hwnd,WM_CLOSE,0,0);
        SendMessage(mess2hwnd,WM_CLOSE,0,0);
        SendMessage(mess3hwnd,WM_CLOSE,0,0);
        SendMessage(mess4hwnd,WM_CLOSE,0,0);
        SendMessage(messstatushwnd,WM_CLOSE,0,0);
        SendMessage(messhi1hwnd,WM_CLOSE,0,0);
        SendMessage(messhi2hwnd,WM_CLOSE,0,0);
        SendMessage(messhi3hwnd,WM_CLOSE,0,0);
        SendMessage(messhi4hwnd,WM_CLOSE,0,0);

        SendMessage(lfmess1hwnd,WM_CLOSE,0,0);
        SendMessage(lfmess2hwnd,WM_CLOSE,0,0);
        SendMessage(lfmess3hwnd,WM_CLOSE,0,0);
        SendMessage(lfmess4hwnd,WM_CLOSE,0,0);
        SendMessage(lfmesshi1hwnd,WM_CLOSE,0,0);
        SendMessage(lfmesshi2hwnd,WM_CLOSE,0,0);
        SendMessage(lfmesshi3hwnd,WM_CLOSE,0,0);
        SendMessage(lfmesshi4hwnd,WM_CLOSE,0,0);

        if(DialogBox(hInst, (LPCTSTR)IDD_StartMessungTTLF, hWnd, (DLGPROC)
startoptionenttlf)==FALSE)
        {
                messkontrolle.messtyp=NIXMESS;
                warten(0.5,TimeUnit); //0.5s warten
                return; // bei Abbruch zurück
        }

        if(messkontrolle.asl==TRUE) // Schnittstelle des ASL bereitstellen
        {
                sprintf(pcCommPort,"COM%i",messkontrolle.aslcom);
                messkontrolle.hComasl = CreateFile( pcCommPort, GENERIC_READ |
GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für ASL
```

```
		if(messkontrolle.hComasl==INVALID_HANDLE_VALUE)
		{
			messkontrolle.asl=FALSE;
			MessageBox(hWnd, "Schnittstelle für ASL ist bereits belegt", "Achtung",
MB_OK|MB_ICONSTOP);
			return;
		}

		dcb.ByteSize = 28;		// data size, xmit, and rcv
		dcb.BaudRate = CBR_9600;	// set the baud rate
		dcb.fBinary=1;					// binary mode, no EOF check
		dcb.fParity=0;			// enable parity checking
		dcb.fOutxCtsFlow=0;				// CTS output flow control
		dcb.fOutxDsrFlow=0;				// DSR output flow control
		dcb.fDtrControl=1;				// DTR flow control type
		dcb.fDsrSensitivity=0;		// DSR sensitivity
		dcb.fTXContinueOnXoff=1; // XOFF continues Tx
		dcb.fOutX=0;			// XON/XOFF out flow control
		dcb.fInX=0;			// XON/XOFF in flow control
		dcb.fErrorChar=0;		// enable error replacement
		dcb.fNull=0;			// enable null stripping
		dcb.fRtsControl=1;		// RTS flow control
		dcb.fAbortOnError=0;	// abort reads/writes on error
		dcb.XonLim=2048;			// transmit XON threshold
		dcb.XoffLim=512;			// transmit XOFF threshold
		dcb.ByteSize=8;			// number of bits/byte, 4-8
		dcb.Parity=NOPARITY;			// 0-4=no,odd,even,mark,space
		dcb.StopBits=TWOSTOPBITS;			// 0,1,2 = 1, 1.5, 2
		dcb.XonChar=17;			// Tx and Rx XON character
		dcb.XoffChar=19;			// Tx and Rx XOFF character
		dcb.ErrorChar=0;			// error replacement character
		dcb.EofChar=0;			// end of input character
		dcb.EvtChar=0;			// received event character

		SetCommState(messkontrolle.hComasl, &dcb);
		PurgeComm(messkontrolle.hComasl,PURGE_TXCLEAR&&PURGE_RXCLEAR);	//
Puffer leeren
		setasl(); // Stellt ASL ein
	}

	// Schnittstelle für LF und Thermo herrichten bereitstellen
	sprintf(pcCommPort,"COM%i",messkontrolle.thermolfcom);
	messkontrolle.hComthermolf	=	CreateFile(	pcCommPort,	GENERIC_READ	|
GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für LF und Thermo

	if(messkontrolle.hComthermolf==INVALID_HANDLE_VALUE)
	{
		MessageBox(hWnd, "Schnittstelle für Thermometer / Leitfähigkeit ist bereits belegt",
"Achtung", MB_OK|MB_ICONSTOP);
		if(messkontrolle.asl==TRUE)
		{
			CloseHandle(messkontrolle.hComasl);
		}
		return;
	}

	dcb.ByteSize = 28;		// data size, xmit, and rcv
	dcb.BaudRate = CBR_19200;	// set the baud rate
	dcb.fBinary=1;					// binary mode, no EOF check
	dcb.fParity=0;			// enable parity checking
	dcb.fOutxCtsFlow=0;				// CTS output flow control
	dcb.fOutxDsrFlow=0;				// DSR output flow control
```

```
	dcb.fDtrControl=0;                              // DTR flow control type
	dcb.fDsrSensitivity=0;       // DSR sensitivity
	dcb.fTXContinueOnXoff=1; // XOFF continues Tx
	dcb.fOutX=0;           // XON/XOFF out flow control
	dcb.fInX=0;           // XON/XOFF in flow control
	dcb.fErrorChar=0;      // enable error replacement
	dcb.fNull=0;          // enable null stripping
	dcb.fRtsControl=0;      // RTS flow control
	dcb.fAbortOnError=0;    // abort reads/writes on error
	dcb.XonLim=2048;             // transmit XON threshold
	dcb.XoffLim=512;             // transmit XOFF threshold
	dcb.ByteSize=8;           // number of bits/byte, 4-8
	dcb.Parity=NOPARITY;              // 0-4=no,odd,even,mark,space
	dcb.StopBits=TWOSTOPBITS;            // 0,1,2 = 1, 1.5, 2
	dcb.XonChar=17;             // Tx and Rx XON character
	dcb.XoffChar=19;            // Tx and Rx XOFF character
	dcb.ErrorChar=0;           // error replacement character
	dcb.EofChar=0;             // end of input character
	dcb.EvtChar='E';            // received event character

	reset();
	reset();
	reset();
	reset();
	reset();

	SetCommState(messkontrolle.hComthermolf, &dcb);
	PurgeComm(messkontrolle.hComthermolf,PURGE_TXCLEAR&&PURGE_RXCLEAR);         //
Puffer leeren

	// Kalibrierdaten einlesen
	if(messkontrolle.messtyp==KANAL30TTLF)
	{
		memset(&ofn, 0, sizeof(OPENFILENAME));
		ofn.lpstrFile = szFilename;
		iErg = FileDialogKal(hWnd, &ofn);

		if(iErg == IDOK)
		{
			//Datei öffnen mit Fehlertest
			if((hFile = _lopen(ofn.lpstrFile, OF_READ)) == -1)
			{
				MessageBox(hWnd, "Fehler beim Dateneinlesen", "Achtung",
MB_OK|MB_ICONSTOP);
				SendMessage(hWnd,WM_CLOSE,NULL,NULL);
				messkontrolle.messtyp=NIXMESS;
				warten(0.5,TimeUnit); //0.5s warten
				CloseHandle(messkontrolle.hComasl); // Schnittstelle wieder schließen
				CloseHandle(messkontrolle.hComthermolf);
				return;
			}

			//an den Dateianfang gehen
			_llseek(hFile, 0L, 0);

			// Datei lesen,  Fehlerprüfung hinzufügen

			for(k=0;k<32;k++)
			{
				readcount=0;
				do //eine Zeile einlesen
				{
					_hread(hFile,&read1,1);
```

```
                                        readdata[readcount++]=read1;
                                }
                                while(readdata[readcount-1]!=13 ); //Zeilenende

                                readdata[readcount-1]=0; //String beenden
                                _hread(hFile,&read1,1); // noch zeilenvorschub einlesen

                                sscanf(readdata,"%le %le %le %le %le
",&messkontrolle.kalparam[k].a,&messkontrolle.kalparam[k].b,&messkontrolle.kalparam[k].c,&messkontrolle.
kalparam[k].uref,&messkontrolle.kalparam[k].r1);
                        }
                        _lclose(hFile);

                }
                else
                {
                        messkontrolle.messtyp=NIXMESS;
                        warten(0.5,TimeUnit); //0.5s warten
                        CloseHandle(messkontrolle.hComasl); // Schnittstelle wieder schließen
                        CloseHandle(messkontrolle.hComthermolf);
                        return;
                }
        }

        messkontrolle.messan=TRUE; // eine Messung läuft

        if(messkontrolle.messtyp==KANAL30TTLF) // Leitfähigkeit und Temperatur
        {
                messkontrolle.maximalerywert=MAXTEMPERATUR;
                messkontrolle.minmalerywert=MINTEMPERATUR;
                messkontrolle.maximalerywertlf=MAXLF;
                messkontrolle.minmalerywertlf=MINLF;
        }
        else if(messkontrolle.messtyp==KANAL30TTLFKAL) //  Leitfähigkeit und Spannung
        {
                messkontrolle.maximalerywert=UREF;
                messkontrolle.minmalerywert=0;
                messkontrolle.maximalerywertlf=UREF;
                messkontrolle.minmalerywertlf=0;
        }

        MoveWindow(hWnd,fensterkoordinaten.xMessMain,fensterkoordinaten.yMessMain,fensterkoordina
ten.nMessWidthMain,fensterkoordinaten.nMessHeighMain,TRUE); // Hauptfenster neu skalieren

        // Nebenfenster darstellen
        InitInstanceMess1(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess2(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess3(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess4(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessStatus(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes1(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes2(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes3(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes4(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess1lf(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess2lf(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess3lf(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMess4lf(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes1lf(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes2lf(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes3lf(hInst,SW_SHOWNORMAL,hWnd);
        InitInstanceMessHiRes4lf(hInst,SW_SHOWNORMAL,hWnd);
```

```
	if(messkontrolle.asl==TRUE)
	{
		InitInstanceASL(hInst,SW_SHOWNORMAL,hWnd);
		InitInstanceMessHiResASL(hInst,SW_SHOWNORMAL,hWnd);
	}
	setmenue(hWnd); // Menüs passend einstellen

	messkontrolle.count=0;
	for(k=0;k<64;k++) // zurücksetzen
	{
		messkontrolle.letztewertex[k]=0;
	}
	messkontrolle.letztewerteasl=0;

	setledmessen(THERMO,1);
	setledmessen(LEITFAEHIGKEIT,1);
	setstoerung(0); // setzt die LED Störung am Thermometer

	// Rührer an

	if(messkontrolle.ruehrtyp==ZAHNRAD)
	{
		zahnradruehrer(messkontrolle.motor);
		wandelfeld(0);
	}
	else
	{
		zahnradruehrer(0);
		wandelfeld(1);
	}

	if(messkontrolle.asl==TRUE)
	{
		getasltemp(); // ersten Wert auslesen und verwerfen.
	}
	frequenz(messkontrolle.frequenz); //  Frequenzausgabe
	uhrstatus(1);
	//reset();
	timeSetEvent(100,50,timerroutine,0,TIME_ONESHOT); // Timer an
}

void saveuntermessung30lf(HWND hWnd, long int i) // speichert Messdaten während der Messung
{
HANDLE hFile;
char zeile[ZEILENLENG+100],dummy[1000];
//unsigned long dummy;
int j;
DWORD dummy2;

	zeile[0]=0;
	hFile=CreateFile(defmessparam.tempfile,GENERIC_WRITE,FILE_SHARE_READ,NULL,OPE
N_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);

	// an das Dateiende gehen
	SetFilePointer(hFile,0,NULL,FILE_END);

	for(j=0;j<64;j++)
	{
		sprintf(dummy,"%.0lf %.8lf ",messdaten30[i].t[j],messdaten30[i].ywerte[j]);
		strcat(zeile,dummy);
	}
	if(messkontrolle.asl==TRUE)
	{
```

```
                sprintf(dummy,"%.0lf %.8lf ",messdaten30[i].aslzeit,messdaten30[i].asl);
                strcat(zeile,dummy);
        }
        sprintf(dummy,"\r\n");
        strcat(zeile,dummy);

        // Datei schreiben

        WriteFile(hFile,zeile,strlen(zeile),&dummy2,NULL);

        CloseHandle(hFile);

        return;
}


int save30utlf(HWND hWnd)
{
static char szFilename[MAX_PATH] = ".txt",szFilenamekanal[MAX_PATH] = ".txt";
int iErg=0,j=0,kanalcount=0;
char *feld,zeile[ZEILENLENG],dummy[2000], kanaltxt[4];
long int i;
HGLOBAL hMemHandleDaten;
HFILE hFile;
unsigned long lBytes;
OPENFILENAME ofn;


        hMemHandleDaten=GlobalAlloc(GHND,sizeof(char)*536*MAXMESSDAUER);  //Größe  des
Charfeldes
        feld=(char*)GlobalLock(hMemHandleDaten);

        memset(&ofn, 0, sizeof(OPENFILENAME));
        ofn.lpstrFile = szFilename;
        iErg = FileDialogSave1ut(hWnd, &ofn);

        if(iErg == IDOK)
        {
                if((hFile = _lcreat(ofn.lpstrFile,0)) == HFILE_ERROR)
                {
                        MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK);
                        SendMessage(hWnd,WM_CLOSE,NULL,NULL);
                        GlobalUnlock(hMemHandleDaten);
                        GlobalFree(hMemHandleDaten);
                        return(FEHLER);
                }

                // an den Dateianfang gehen
                _llseek(hFile, 0L, 0);

                // Datei schreiben

                for(i=0;i<messkontrolle.anzahl;i++)
                {
                        zeile[0]=0;
                        for(j=0;j<64;j++)
                        {
                                sprintf(dummy,"%.0lf                                                %.8lf
",messdaten30[i].t[j],messdaten30[i].ywerte[j]);
                                strcat(zeile,dummy);
                        }
                        if(messkontrolle.asl==TRUE)
                        {
```

```
                                sprintf(dummy,"%.0lf %.8lf ",messdaten30[i].aslzeit,messdaten30[i].asl);
                                strcat(zeile,dummy);
                        }
                        sprintf(dummy,"\r\n");
                        strcat(zeile,dummy);

                        lBytes = _hwrite(hFile,zeile,strlen(zeile));
                        if(lBytes!=strlen(zeile))
                        {
                                MessageBox(hWnd, "Da ist was danebengegangen", "Achtung",
MB_OK|MB_ICONSTOP);
                                _lclose(hFile);
                        GlobalUnlock(hMemHandleDaten);
                GlobalFree(hMemHandleDaten);
                                return(FEHLER);
                        }
                        else
                        {
                                //_llseek(hFile, lBytes, 1);
                        }
                }
                _lclose(hFile);
        }
        GlobalUnlock(hMemHandleDaten);
        GlobalFree(hMemHandleDaten);

        // einzelne Files rausschreiben
        if(MessageBox(hWnd, "Kanaldateien rauschreiben", "Achtung",
MB_YESNO|MB_ICONQUESTION)==IDYES)
        {
                for(j=0;j<32;j++)
                {
                        // File-Namen der Einzelfiles festlegen
                        kanaltxt[0]=0;
                        szFilenamekanal[0]=0;
                        strncpy(szFilenamekanal,szFilename,strlen(szFilename)-4);
                        szFilenamekanal[strlen(szFilename)-4]=0;
                        sprintf(kanaltxt,"_c%i.txt",j+1);
                        strcat(szFilenamekanal,kanaltxt);


        hMemHandleDaten=GlobalAlloc(GHND,sizeof(char)*536*MAXMESSDAUER); //Größe des
Charfeldes
                        feld=(char*)GlobalLock(hMemHandleDaten);
                        memset(&ofn, 0, sizeof(OPENFILENAME));

                        ofn.lpstrFile = szFilenamekanal;


                        if((hFile = _lcreat(ofn.lpstrFile,0)) == HFILE_ERROR)
                        {
                                MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK);
                                SendMessage(hWnd,WM_CLOSE,NULL,NULL);
                                GlobalUnlock(hMemHandleDaten);
                                GlobalFree(hMemHandleDaten);
                                return(FEHLER);
                        }

                        // an den Dateianfang gehen
                        _llseek(hFile, 0L, 0);

                        // Datei schreiben
```

```
                        for(i=0;i<messkontrolle.anzahl;i++)
                        {
                                zeile[0]=0;
                                sprintf(dummy,"%.0lf %.8lf %.0lf %.8lf %.8lf
%.8lf",messdaten30[i].t[j],messdaten30[i].ywerte[j],messdaten30[i].t[j+32],messdaten30[i].ywerte[j+32],1/(mes
sdaten30[i].ywerte[j]+273.15)/1000,messdaten30[i].ywerte[j+32]);
                                strcat(zeile,dummy);

                                sprintf(dummy,"\r\n");
                                strcat(zeile,dummy);

                                lBytes = _hwrite(hFile,zeile,strlen(zeile));
                                if(lBytes!=strlen(zeile))
                                {
                                        MessageBox(hWnd, "Da ist was danebengegangen", "Achtung",
MB_OK|MB_ICONSTOP);
                                        _lclose(hFile);
                                GlobalUnlock(hMemHandleDaten);
                        GlobalFree(hMemHandleDaten);
                                        return(FEHLER);
                                }
                                else
                                {
                                        //_llseek(hFile, lBytes, 1);
                                }
                        }
                        _lclose(hFile);
                        GlobalUnlock(hMemHandleDaten);
                        GlobalFree(hMemHandleDaten);
                }
                if(messkontrolle.asl==TRUE)
                {
                //                      sprintf(dummy,"%.0lf                                    %.8lf
",messdaten30[i].aslzeit,messdaten30[i].asl);
                //                      strcat(zeile,dummy);
                }
        }


        return(OK);
}


void kanalmess30lf(HWND hWnd) // führt die einzelnen Messchritte durch
{
long int messzeit=0;
double u[8],spike=-273.15;
static int i=0,spikecount[9];

        // Rührer an

        if(messkontrolle.ruehrtyp==ZAHNRAD)
        {
                zahnradruehrer(messkontrolle.motor);
                wandelfeld(0);
        }
        else
        {
                zahnradruehrer(0);
                wandelfeld(1);
        }

        messkanal(i+1,u,&messzeit); // Zeit und Spannung einlesen
```

```
        messdaten30[messkontrolle.count].t[4*i]=messzeit; // in die entspechenden Speicherbereiche legen Thermometer 0 bis 31 sind Thermometerdaten
        messdaten30[messkontrolle.count].t[4*i+1]=messzeit;
        messdaten30[messkontrolle.count].t[4*i+2]=messzeit;
        messdaten30[messkontrolle.count].t[4*i+3]=messzeit;
        messkontrolle.letztewertex[4*i]=messzeit;
        messkontrolle.letztewertex[4*i+1]=messzeit;
        messkontrolle.letztewertex[4*i+2]=messzeit;
        messkontrolle.letztewertex[4*i+3]=messzeit;
        messkontrolle.messzeit=messzeit;


        messdaten30[messkontrolle.count].t[32+4*i]=messzeit; // in die entspechenden Speicherbereiche legen, 32-63 Leitfähigkeit
        messdaten30[messkontrolle.count].t[32+4*i+1]=messzeit;
        messdaten30[messkontrolle.count].t[32+4*i+2]=messzeit;
        messdaten30[messkontrolle.count].t[32+4*i+3]=messzeit;
        messkontrolle.letztewertex[32+4*i]=messzeit;
        messkontrolle.letztewertex[32+4*i+1]=messzeit;
        messkontrolle.letztewertex[32+4*i+2]=messzeit;
        messkontrolle.letztewertex[32+4*i+3]=messzeit;

   if(messkontrolle.messtyp==KANAL30TTLF)
        {
                // Spikes rausfiltern, nur Kanal 1 ist betroffen
                spike=maketemp(u[0],4*i+1);
                if(messkontrolle.count-1>0)
                {
                        if((fabs(messdaten30[messkontrolle.count-1].ywerte[4*i]-spike)>(double)5)      && (spikecount[i]<1)) // Spike aufgetreten
                        {
                                spike=messdaten30[messkontrolle.count-1].ywerte[4*i];
                                spikecount[i]=1; // Maximal nur daraufolgenden Messwert entfernen
                        }
                        else
                                spikecount[i]=0;
                }


                messdaten30[messkontrolle.count].ywerte[4*i]=spike;//maketemp(u[0],4*i+1); // Thermo
                messdaten30[messkontrolle.count].ywerte[4*i+1]=maketemp(u[1],4*i+2);
                messdaten30[messkontrolle.count].ywerte[4*i+2]=maketemp(u[2],4*i+3);
                messdaten30[messkontrolle.count].ywerte[4*i+3]=maketemp(u[3],4*i+4);

                messkontrolle.letztewerte[4*i]=spike;//maketemp(u[0],4*i+1); // Werte für Status und aktueller Wert
                messkontrolle.letztewerte[4*i+1]=maketemp(u[1],4*i+2);
                messkontrolle.letztewerte[4*i+2]=maketemp(u[2],4*i+3);
                messkontrolle.letztewerte[4*i+3]=maketemp(u[3],4*i+4);

                messdaten30[messkontrolle.count].ywerte[32+4*i]=makelf(u[4]); // LF
                messdaten30[messkontrolle.count].ywerte[32+4*i+1]=makelf(u[5]);

                messdaten30[messkontrolle.count].ywerte[32+4*i+2]=0;//makelf(u[6]);
                messdaten30[messkontrolle.count].ywerte[32+4*i+3]=0;//makelf(u[7]);

                messkontrolle.letztewerte[32+4*i]=makelf(u[4]); // Werte für Status und aktueller Wert
                messkontrolle.letztewerte[32+4*i+1]=makelf(u[5]);
                messkontrolle.letztewerte[32+4*i+2]=0;//makelf(u[6]);
                messkontrolle.letztewerte[32+4*i+3]=0;//makelf(u[7]);
        }
        else if(messkontrolle.messtyp==KANAL30TTLFKAL)
```

```
	{
				// Spikes rausfiltern, nur Kanal 1 ist betroffen
		spike=u[0];
		if(messkontrolle.count-1>0)
		{
			if((fabs(messdaten30[messkontrolle.count-1].ywerte[4*i]-spike)>(double)5)	&&
(spikecount[i]<1)) // Spike aufgetreten
			{
				spike=messdaten30[messkontrolle.count-1].ywerte[4*i];
				spikecount[i]=1; // Maximal nur daraufolgenden Messwert entfernen
			}
			else
				spikecount[i]=0;
		}


		messdaten30[messkontrolle.count].ywerte[4*i]=spike;//maketemp(u[0],4*i+1); // Thermo
		messdaten30[messkontrolle.count].ywerte[4*i+1]=u[1];
		messdaten30[messkontrolle.count].ywerte[4*i+2]=u[2];
		messdaten30[messkontrolle.count].ywerte[4*i+3]=u[3];

		messkontrolle.letztewerte[4*i]=spike;//maketemp(u[0],4*i+1); // Werte für Status und
aktueller Wert
		messkontrolle.letztewerte[4*i+1]=u[1];
		messkontrolle.letztewerte[4*i+2]=u[2];
		messkontrolle.letztewerte[4*i+3]=u[3];

		messdaten30[messkontrolle.count].ywerte[32+4*i]=u[4]; // LF
		messdaten30[messkontrolle.count].ywerte[32+4*i+1]=u[5];

		messdaten30[messkontrolle.count].ywerte[32+4*i+2]=0;//u[6];
		messdaten30[messkontrolle.count].ywerte[32+4*i+3]=0;//u[7];

		messkontrolle.letztewerte[32+4*i]=u[4]; // Werte für Status und aktueller Wert
		messkontrolle.letztewerte[32+4*i+1]=u[5];
		messkontrolle.letztewerte[32+4*i+2]=0;//u[6];
		messkontrolle.letztewerte[32+4*i+3]=0;//u[7];
	}

	if(i<7)// Kanäle durchschalten
	{
		i++;
	}
	else // achter Kanal erreicht, wieder von vorne und asl messen
	{
		if(messkontrolle.asl==TRUE)// bei Messung mit ASL
		{
			messdaten30[messkontrolle.count].asl=getasltemp()-273.15;
			messdaten30[messkontrolle.count].aslzeit=messzeit;
			messkontrolle.letztewerteaslx=messzeit;
			messkontrolle.letztewerteasl=messdaten30[messkontrolle.count].asl;
			SendMessage(messASLhwnd,WM_PAINT,0,0);
			SendMessage(messhiASLhwnd,WM_PAINT,0,0);
		}
		saveuntermessung30lf(hWnd,messkontrolle.count); // speichert Messdaten während der
Messung
		messkontrolle.count++; // Datenpunktzähler weiterzählen
		messkontrolle.anzahl=messkontrolle.count;
		i=0;
	}

	SendMessage(mess1hwnd,WM_PAINT,0,0); // Graphen zeichnen
	SendMessage(mess2hwnd,WM_PAINT,0,0);
```

```
	SendMessage(mess3hwnd,WM_PAINT,0,0);
	SendMessage(mess4hwnd,WM_PAINT,0,0);

	SendMessage(lfmess1hwnd,WM_PAINT,0,0); // Graphen zeichnen
	SendMessage(lfmess2hwnd,WM_PAINT,0,0);
	SendMessage(lfmess3hwnd,WM_PAINT,0,0);
	SendMessage(lfmess4hwnd,WM_PAINT,0,0);

	SendMessage(messstatushwnd,WM_PAINT,0,0);
	SendMessage(messhi1hwnd,WM_PAINT,0,0);
	SendMessage(messhi2hwnd,WM_PAINT,0,0);
	SendMessage(messhi3hwnd,WM_PAINT,0,0);
	SendMessage(messhi4hwnd,WM_PAINT,0,0);
	SendMessage(mainhwnd,WM_PAINT,0,0);

	SendMessage(lfmesshi1hwnd,WM_PAINT,0,0);
	SendMessage(lfmesshi2hwnd,WM_PAINT,0,0);
	SendMessage(lfmesshi3hwnd,WM_PAINT,0,0);
	SendMessage(lfmesshi4hwnd,WM_PAINT,0,0);


	if(messzeit>=messkontrolle.messdauer || (GetAsyncKeyState('E') != 0)) // Messdauer
überschritten oder Messung abbrechen
	{
		messkontrolle.messan=FALSE; //keine Messung läuft
		uhrstatus(0);
		GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen
werden, um das Speichern nicht zu stören
		messkontrolle.anzahl=messkontrolle.count;
		setledmessen(THERMO,0);
		setledmessen(LEITFAEHIGKEIT,0);
		save30utlf(hWnd);
		setmenue(hWnd); // Menüs passend einstellen
		CloseHandle(messkontrolle.hComthermolf); // Schnittstelle wieder schließen
		CloseHandle(messkontrolle.hComasl);
		i=0;
	}
	else
	{
		timeSetEvent(100,50,timerroutine,0,TIME_ONESHOT);
	}
}

LRESULT CALLBACK startoptionenttlf( HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam )
{
char dummy[MAX_PATH];;

	switch( message )
	{
		case WM_INITDIALOG:

			if(messkontrolle.asl==TRUE)
				CheckDlgButton(hDlg,IDC_CHECKASL,1);
			else
				CheckDlgButton(hDlg,IDC_CHECKASL,0);

			if(messkontrolle.ruehrtyp==ZAHNRAD)

	CheckDlgButton(hDlg,IDC_RUERGESCHWINDIGKEITZAHNRAD,1);
			else if (messkontrolle.ruehrtyp==WANDELFELD)
				CheckDlgButton(hDlg,IDC_WANDELFELDRUEHRER,1);
			else
```

```
                        {

        CheckDlgButton(hDlg,IDC_RUERGESCHWINDIGKEITZAHNRAD,0);
                                CheckDlgButton(hDlg,IDC_WANDELFELDRUEHRER,0);
                        }

                        SetDlgItemText(hDlg,IDC_PFAD,defmessparam.tempfile);

                        sprintf(dummy,"%.1lf",messkontrolle.maxbathtemp);
                        SetDlgItemText(hDlg,IDC_MAXTEMP,dummy);

                        sprintf(dummy,"%i",messkontrolle.aslcom);
                        SetDlgItemText(hDlg,IDC_COMASL,dummy);

                        sprintf(dummy,"%.1lf",messkontrolle.frequenz);
                        SetDlgItemText(hDlg,IDC_FREQUENZ,dummy);

                        if(messkontrolle.frequenz<=0)
      {
        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben", "Achtung", MB_OK|MB_ICONSTOP);
        return FALSE;
      }

                        if(messkontrolle.aslcom<=0)
      {
        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben", "Achtung", MB_OK|MB_ICONSTOP);
        return FALSE;
      }

                        sprintf(dummy,"%i",messkontrolle.motor);
                        SetDlgItemText(hDlg,IDC_RUEHR,dummy);
                        if(messkontrolle.motor<=0)
      {
        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben", "Achtung", MB_OK|MB_ICONSTOP);
        return FALSE;
      }
                        else if(messkontrolle.motor>100)
      {
        MessageBox(hDlg, "Keine Werte größer als 100 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
        return FALSE;
      }
                        sprintf(dummy,"%.0lf",messkontrolle.messdauer);//Ausgabe der Messdauer
      SetDlgItemText(hDlg,IDC_MESSDAUER,dummy);

                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return FALSE;
                        }
                        else if(LOWORD(wParam) == IDOK)
                        {
                                // Checkboxen der Comports
                                if(IsDlgButtonChecked(hDlg,IDC_CHECKASL)==BST_CHECKED)
                                        messkontrolle.asl=TRUE;
                                else
                                        messkontrolle.asl=FALSE;

                                // Maximale Temperatur
                                GetDlgItemText(hDlg,IDC_MAXTEMP,dummy,99);
```

```
                                        sscanf(dummy, "%lf", &messkontrolle.maxbathtemp);

                                        // Motor
                                        GetDlgItemText(hDlg,IDC_RUEHR,dummy,99);
                                        sscanf(dummy, "%i", &messkontrolle.motor);
                                        if(messkontrolle.motor<=0)
                                        {
                                                MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                                return FALSE;
                                        }
                                        else if(messkontrolle.motor>100)
                                        {
                                                MessageBox(hDlg, "Keine Werte größer als 100 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                                return FALSE;
                                        }

                                        // Tempfile
                                        GetDlgItemText(hDlg,IDC_PFAD,defmessparam.tempfile,99);

                                        // Frequenz
                                        GetDlgItemText(hDlg,IDC_FREQUENZ,dummy,99);
                                        sscanf(dummy, "%lf", &messkontrolle.frequenz);
                                        if(messkontrolle.frequenz<=0)
                                        {
                                                MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                                return FALSE;
                                        }

                                        // Radiobuttons der Rührwerke


        if(IsDlgButtonChecked(hDlg,IDC_RUERGESCHWINDIGKEITZAHNRAD)==BST_CHECKE
D)
                                                messkontrolle.ruehrtyp=ZAHNRAD;
                                        else                                                    if
(IsDlgButtonChecked(hDlg,IDC_WANDELFELDRUEHRER)==BST_CHECKED)
                                                messkontrolle.ruehrtyp=WANDELFELD;
                                        else
                                                messkontrolle.ruehrtyp=0;

                                        GetDlgItemText(hDlg,IDC_MESSDAUER,dummy,99);

          messkontrolle.messdauer=atof(dummy);//Messdauer einlesen

          if(messkontrolle.messdauer<=0)
          {
            MessageBox(hDlg, "Für Messdauer nur Werte größer 0 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
            return FALSE;
          }
                                        else if(messkontrolle.messdauer>MAXMESSDAUER)
          {
            MessageBox(hDlg, "Für Messdauer nur Werte kleiner als 360000 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
            return FALSE;
          }

                                        EndDialog(hDlg, LOWORD(wParam));
                                        return TRUE;
                                }
```

```
                    break;

                    case IDCANCEL :
                            EndDialog(hDlg,FALSE);
                            return TRUE;
      break;

            }
   return FALSE;
}
```

11.7.2.17 messwtw.cpp

```
// Funktionen 1-Kanal-Schreiber

#include "StdAfx.h"

extern DATENSATZ * messdaten;
extern HGLOBAL hMemHandle1;
extern REIHENSTEUERSTRUCT messkontrolle;
extern HINSTANCE hInst;
extern MESSSTEUERSTRUCT defmessparam;

void wtwkanalschreiber(HWND hWnd) //1 Kanal G/t schreiber mit wtw
{
int maxwinx,maxwiny,startx,starty,i=0;
double ywertmax,ywertmin,u=0,messdauer=0;
HDC hDc;
RECT Rect;
char outtext[100];
HPEN hStift[8];
double TimeUnit;
DWORD starttime;
HANDLE hFile32;
DCB dcb;
char string[MAX_PATH],pcCommPort[5];

        if(messkontrolle.wtw==TRUE) // Schnittstelle des WTW bereitstellen
        {
                sprintf(pcCommPort,"COM%i",messkontrolle.wtwcom);
                messkontrolle.hComwtw = CreateFile( pcCommPort, GENERIC_READ |
GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für ASL
                if(messkontrolle.hComwtw==INVALID_HANDLE_VALUE)
                {
                        messkontrolle.wtw=FALSE;
                        MessageBox(hWnd, "Schnittstelle für WTW ist bereits belegt", "Achtung",
MB_OK|MB_ICONSTOP);
                        return;
                }
                dcb.ByteSize = 28;            // data size, xmit, and rcv
                dcb.BaudRate = CBR_4800;     // set the baud rate
                dcb.fBinary=1;                                  // binary mode, no EOF check
                dcb.fParity=0;               // enable parity checking
                dcb.fOutxCtsFlow=1;                     // CTS output flow control
                dcb.fOutxDsrFlow=0;                             // DSR output flow control
                dcb.fDtrControl=0;                              // DTR flow control type
                dcb.fDsrSensitivity=0;       // DSR sensitivity
                dcb.fTXContinueOnXoff=1; // XOFF continues Tx
                dcb.fOutX=0;           // XON/XOFF out flow control
                dcb.fInX=0;            // XON/XOFF in flow control
                dcb.fErrorChar=0;      // enable error replacement
                dcb.fNull=0;           // enable null stripping
                dcb.fRtsControl=1;       // RTS flow control
```

```
                dcb.fAbortOnError=0;     // abort reads/writes on error
                dcb.XonLim=2048;              // transmit XON threshold
                dcb.XoffLim=512;             // transmit XOFF threshold
                dcb.ByteSize=8;            // number of bits/byte, 4-8
                dcb.Parity=NOPARITY;              // 0-4=no,odd,even,mark,space
                dcb.StopBits=ONESTOPBIT;            // 0,1,2 = 1, 1.5, 2
                dcb.XonChar=17;              // Tx and Rx XON character
                dcb.XoffChar=19;             // Tx and Rx XOFF character
                dcb.ErrorChar=0;           // error replacement character
                dcb.EofChar=0;              // end of input character
                dcb.EvtChar=0;              // received event character

                SetCommState(messkontrolle.hComwtw, &dcb);
                PurgeComm(messkontrolle.hComwtw,PURGE_TXCLEAR&&PURGE_RXCLEAR);  //
Puffer leeren
        }



        TimeInit(&TimeUnit);

        if(DialogBox(hInst,              (LPCTSTR)IDD_8KanalTt,              hWnd,              (DLGPROC)
StartDialogkanal8Tt)==FALSE)
                return; // bei Abbruch zurück

        messkontrolle.messtyp=WTWGT; //Einstellungen für diesen Messtyp
        messkontrolle.messan=TRUE; // eine Messung läuft
        messkontrolle.maximalerywert=2;
        messkontrolle.minmalerywert=0;

        drawkosy(hWnd, NOPAINTMSG);

        hStift[0]=CreatePen(PS_SOLID,0,RGB(0,0,0)); // einzelne Stifte herrichten

        ywertmax=messkontrolle.maximalerywert;
        ywertmin=messkontrolle.minmalerywert;

        hDc=GetDC(hWnd);// Fenster Parameter
        GetClientRect(hWnd, &Rect);
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;
        starty=(int) (maxwiny-UAB-OAB)+OAB;
        startx=(int) LAB;

        messkontrolle.flagkanal1=TRUE; // Messkanal setzen
        messkontrolle.flagkanal2=FALSE;
        messkontrolle.flagkanal3=FALSE;
        messkontrolle.flagkanal4=FALSE;
        messkontrolle.flagkanal5=FALSE;
        messkontrolle.flagkanal6=FALSE;
        messkontrolle.flagkanal7=FALSE;
        messkontrolle.flagkanal8=FALSE;

        hFile32=CreateFile(defmessparam.tempfile,GENERIC_WRITE,FILE_SHARE_READ,NULL,CR
EATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
        CloseHandle(hFile32); // altes Tempfile löschen


        GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden

        getwtwlf597s();
        warten(3.5,TimeUnit); // 3.5s warten, dann Gerät wieder bereit
```

```
	starttime=GetTickCount()/1000; // hier Zeit vom System
	do
	{
		messdauer=(GetTickCount()/1000-starttime);
		messdaten[i].t=messdauer;
		messdaten[i].ywerte[0]=getwtwlf597s();

		sprintf(outtext,"%.0lf s, %.3lf S", messdauer, messdaten[i].ywerte[0]);
		TextOut(hDc,LAB+10,(OAB-TEXTYDY)/2,outtext,strlen(outtext));

		SelectObject(hDc,hStift[0]);
		if(i>0)
		{
			MoveToEx(hDc,(int)(startx+(messdaten[i-1].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[i-
1].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);
			LineTo(hDc,(int)(startx+(messdaten[i].t-
messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round ((starty-
(messdaten[i].ywerte[0]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))));
		}
		saveuntermessung(hWnd,i); // Daten während der Messung speichern
		i++;
		//warten(3.5,TimeUnit); // 3,5s warten, dann Gerät wieder bereit
		if(GetAsyncKeyState('E') != 0)
		{
			messkontrolle.messdauer=messdauer;
			break; // Messung abbrechen
		}
	}
	while(messdauer<=messkontrolle.messdauer);

	messkontrolle.anzahl=i;

	GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden,
um das Speichern nicht zu stören

	save1ut(hWnd); // Daten speichern

	DeleteObject(hStift[0]);
	ReleaseDC(hWnd,hDc);
	messkontrolle.messan=FALSE; // keine Messung läuft
	CloseHandle(messkontrolle.hComwtw); // Schnittstelle wieder schließen
}
```

11.7.2.18 small.cpp

```
// enthält kleine Routinen, durchgehend benötigt werden

#include "StdAfx.h"

extern MESSSTEUERSTRUCT defmessparam;
extern REIHENSTEUERSTRUCT messkontrolle;
extern HINSTANCE hInst;

DATENSATZ * messdaten;
HGLOBAL hMemHandle1;

DATENSATZ30 * messdaten30;
HGLOBAL hMemHandle30;

FENSTERKOORDSSTRUCT fensterkoordinaten;
TIMECAPS multmediatimertc;
UINT multmediatimerwTimerRes;
```

```
double werte[33]; // speichert Mittelwerte

void globalinit(HWND hWnd) // stellt grundlegende Parameter ein, wird später durch Einlesen aus ini-datei ersetzt
{
int portbuffer,ba=0x378;
unsigned long stringlaenge;
char string[MAX_PATH];

	stringlaenge=GetPrivateProfileString("Hardware", "LPT-Basisadresse Thermometer", "0x378", string, 29, INIFILENAME);
	sscanf(string, "%x", &defmessparam.ba);

	stringlaenge=GetPrivateProfileString("Hardware", "LPT-Basisadresse Leitfaehigkeit", "0x278", string, 29, INIFILENAME);
	sscanf(string, "%x", &defmessparam.balf);

	_outp(defmessparam.ba,0); // Schnittstelle auf 0
	_outp(defmessparam.ba+2,0x9); // Schnittstelle auf 0, Achtung invertierende Eingänge STROBE und SLKT

	_outp(defmessparam.balf,0); // Schnittstelle auf 0
	_outp(defmessparam.balf+2,0x9); // Schnittstelle auf 0, Achtung invertierende Eingänge STROBE und SLKT

	stoerung(0,defmessparam.ba); // Keine Störung, Messung kann laufen, Thermometer
	stoerung(0,defmessparam.balf); // Keine Störung, Messung kann laufen, Leitfähigkeit

	stringlaenge=GetPrivateProfileString("Hardware", "Tempfile", "g:\\30kanaltemp.txt", defmessparam.tempfile, MAX_PATH, INIFILENAME);

	stringlaenge=GetPrivateProfileString("Hardware", "Maximaletemperatur", "50", string, 29, INIFILENAME);
	sscanf(string, "%lf", &messkontrolle.maxbathtemp);

	stringlaenge=GetPrivateProfileString("Hardware", "ASL", "0", string, 29, INIFILENAME); // Default kein ASL
	sscanf(string, "%i", &messkontrolle.asl);

	stringlaenge=GetPrivateProfileString("Hardware", "ASLCOMM", "1", string, 29, INIFILENAME); // Default COM1
	sscanf(string, "%i", &messkontrolle.aslcom);

	stringlaenge=GetPrivateProfileString("Hardware", "WTW", "0", string, 29, INIFILENAME); // Default kein WTW
	sscanf(string, "%i", &messkontrolle.wtw);

	stringlaenge=GetPrivateProfileString("Hardware", "WTWCOMM", "4", string, 29, INIFILENAME); // Default COM4
	sscanf(string, "%i", &messkontrolle.wtwcom);

	stringlaenge=GetPrivateProfileString("Hardware", "MOTOR", "80", string, 29, INIFILENAME); // Rührleistung Drehrührer
	sscanf(string, "%i", &messkontrolle.motor);

	stringlaenge=GetPrivateProfileString("Hardware", "FREQUENZ", "5000", string, 29, INIFILENAME); // Default 5 kHz
	sscanf(string, "%lf", &messkontrolle.frequenz);

	stringlaenge=GetPrivateProfileString("Hardware", "THERMOLF", "1", string, 29, INIFILENAME); // Default LF und TEMP angeschlossen
	sscanf(string, "%i", &messkontrolle.thermolf);
```

```
	stringlaenge=GetPrivateProfileString("Hardware", "THERMOLFCOMM", "2", string, 29, INIFILENAME); // Default COM2
	sscanf(string, "%i", &messkontrolle.thermolfcom);

	stringlaenge=GetPrivateProfileString("Hardware", "KRYO", "1", string, 29, INIFILENAME); // Default Kryo
	sscanf(string, "%i", &messkontrolle.kryo);
	stringlaenge=GetPrivateProfileString("Hardware", "KRYOCOMM", "3", string, 29, INIFILENAME); // Default COM3
	sscanf(string, "%i", &messkontrolle.kryocom );


	stringlaenge=GetPrivateProfileString("Hardware", "RUEHRTYP", "120", string, 29, INIFILENAME); // Default WANDELFELDRührer
	sscanf(string, "%i", &messkontrolle.ruehrtyp);

	stringlaenge=GetPrivateProfileString("Hardware", "MESSTYP", "64", string, 29, INIFILENAME); // Default WANDELFELDRührer
	sscanf(string, "%i", &messkontrolle.typ);

	hMemHandle1=GlobalAlloc(GHND,sizeof(DATENSATZ)*MAXMESSDAUER); // für 1 bis 8 Kanal
	messdaten=(DATENSATZ*)GlobalLock(hMemHandle1);

	hMemHandle30=GlobalAlloc(GHND,sizeof(DATENSATZ30)*MAXMESSDAUER*2); // für 30Kanal
	messdaten30=(DATENSATZ30*)GlobalLock(hMemHandle30);

	if(hMemHandle1==NULL || messdaten==NULL || hMemHandle30==NULL || messdaten30==NULL)
	{
		MessageBox(hWnd, "Zuwenig Speicher", "Achtung", MB_OK|MB_ICONSTOP);
		globalexit();
		PostQuitMessage(0);
		return;
	}

	defmessparam.verstearker=1;
	defmessparam.kanal=1;

	messkontrolle.messtyp=TEST1KANAL;
	messkontrolle.messdauer=1000;
	messkontrolle.maximalerywert=2.5;

	// für 30Kanal
  portbuffer=_inp(defmessparam.ba);
	portbuffer=(portbuffer & 254)+1; //bit 1 HI, pin 2 HI; dsync auf hi
	_outp(defmessparam.ba,portbuffer);

	portbuffer=(portbuffer & 0x7F)+0x80; //bit 8 , pin 9 ; CS für MAX542 auf Hi, dann kein Datentransfer
	_outp(defmessparam.ba,portbuffer);

  portbuffer=_inp(defmessparam.balf);
	portbuffer=(portbuffer & 254)+1; //bit 1 HI, pin 2 HI; dsync auf hi
	_outp(defmessparam.balf,portbuffer);

	portbuffer=(portbuffer & 0x7F)+0x80; //bit 8 , pin 9 ; CS für MAX542 auf Hi, dann kein Datentransfer
	_outp(defmessparam.balf,portbuffer);


	u_ausgabe(0,defmessparam.ba); // DAC auf 0V
```

```
        motor(0,defmessparam.ba); // Motor aus

        // Fensterkoordinaten einstellen

        stringlaenge=GetPrivateProfileString("Graphik", "xMess1", "0", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMess1);
        stringlaenge=GetPrivateProfileString("Graphik", "yMess1", "140", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMess1);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMess1", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMess1);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMess1", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMess1);

        stringlaenge=GetPrivateProfileString("Graphik", "xMess2", "341", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMess2);
        stringlaenge=GetPrivateProfileString("Graphik", "yMess2", "140", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMess2);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMess2", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMess2);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMess2", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMess2);

        stringlaenge=GetPrivateProfileString("Graphik", "xMess3", "0", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMess3);
        stringlaenge=GetPrivateProfileString("Graphik", "yMess3", "441", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMess3);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMess3", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMess3);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMess3", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMess3);

        stringlaenge=GetPrivateProfileString("Graphik", "xMess4", "341", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMess4);
        stringlaenge=GetPrivateProfileString("Graphik", "yMess4", "441", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMess4);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMess4", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMess4);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMess4", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMess4);

        stringlaenge=GetPrivateProfileString("Graphik", "xMessStatus", "512", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMessStatus);
        stringlaenge=GetPrivateProfileString("Graphik", "yMessStatus", "0", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMessStatus);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMessStatus", "140", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMessStatus);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMessStatus", "512", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMessStatus);

        stringlaenge=GetPrivateProfileString("Graphik", "xMessHiRes1", "682", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMessHiRes1);
```

```
        stringlaenge=GetPrivateProfileString("Graphik", "yMessHiRes1", "140", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMessHiRes1);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMessHiRes1", "602", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiRes1);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMessHiRes1", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiRes1);

        stringlaenge=GetPrivateProfileString("Graphik", "xMessHiRes2", "852", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMessHiRes2);
        stringlaenge=GetPrivateProfileString("Graphik", "yMessHiRes2", "140", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMessHiRes2);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMessHiRes2", "602", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiRes2);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMessHiRes2", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiRes2);

        stringlaenge=GetPrivateProfileString("Graphik", "xMessHiRes3", "682", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMessHiRes3);
        stringlaenge=GetPrivateProfileString("Graphik", "yMessHiRes3", "441", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMessHiRes3);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMessHiRes3", "602", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiRes3);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMessHiRes3", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiRes3);

        stringlaenge=GetPrivateProfileString("Graphik", "xMessHiRes4", "852", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMessHiRes4);
        stringlaenge=GetPrivateProfileString("Graphik", "yMessHiRes4", "441", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMessHiRes4);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMessHiRes4", "602", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiRes4);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMessHiRes4", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiRes4);

        stringlaenge=GetPrivateProfileString("Graphik", "xMessMain", "0", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMessMain);
        stringlaenge=GetPrivateProfileString("Graphik", "yMessMain", "0", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMessMain);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMessMain", "140", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nMessHeighMain);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMessMain", "512", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nMessWidthMain);

        stringlaenge=GetPrivateProfileString("Graphik", "xMessASL", "1022", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMessASL);
        stringlaenge=GetPrivateProfileString("Graphik", "yMessASL", "140", string, 29, INIFILENAME);
```

```
        sscanf(string, "%i", &fensterkoordinaten.yMessASL);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMessASL", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMessASL);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMessASL", "130", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMessASL);

        stringlaenge=GetPrivateProfileString("Graphik", "xMessASLHIRES", "1022", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMessHiResASL);
        stringlaenge=GetPrivateProfileString("Graphik", "yMessASLHIRES", "441", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMessHiResASL);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMessASLHIRES", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiResASL);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMessASLHIRES", "130", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiResASL);

        // Leitfähigkeit
        stringlaenge=GetPrivateProfileString("Graphik", "xMess1lf", "0", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMess1lf);
        stringlaenge=GetPrivateProfileString("Graphik", "yMess1lf", "140", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMess1lf);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMess1lf", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMess1lf);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMess1lf", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMess1lf);

        stringlaenge=GetPrivateProfileString("Graphik", "xMess2lf", "341", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMess2lf);
        stringlaenge=GetPrivateProfileString("Graphik", "yMess2lf", "140", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMess2lf);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMess2lf", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMess2lf);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMess2lf", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMess2lf);

        stringlaenge=GetPrivateProfileString("Graphik", "xMess3lf", "0", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMess3lf);
        stringlaenge=GetPrivateProfileString("Graphik", "yMess3lf", "441", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMess3lf);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMess3lf", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMess3lf);
        stringlaenge=GetPrivateProfileString("Graphik", "WidthMess3lf", "341", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nWidthMess3lf);

        stringlaenge=GetPrivateProfileString("Graphik", "xMess4lf", "341", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.xMess4lf);
        stringlaenge=GetPrivateProfileString("Graphik", "yMess4lf", "441", string, 29, INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.yMess4lf);
        stringlaenge=GetPrivateProfileString("Graphik", "HeightMess4lf", "301", string, 29,
INIFILENAME);
        sscanf(string, "%i", &fensterkoordinaten.nHeightMess4lf);
```

```
	stringlaenge=GetPrivateProfileString("Graphik", "WidthMess4lf", "341", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nWidthMess4lf);

	stringlaenge=GetPrivateProfileString("Graphik", "xMessHiRes1lf", "682", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.xMessHiRes1lf);
	stringlaenge=GetPrivateProfileString("Graphik", "yMessHiRes1lf", "140", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.yMessHiRes1lf);
	stringlaenge=GetPrivateProfileString("Graphik", "HeightMessHiRes1lf", "602", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiRes1lf);
	stringlaenge=GetPrivateProfileString("Graphik", "WidthMessHiRes1lf", "341", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiRes1lf);

	stringlaenge=GetPrivateProfileString("Graphik", "xMessHiRes2lf", "852", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.xMessHiRes2lf);
	stringlaenge=GetPrivateProfileString("Graphik", "yMessHiRes2lf", "140", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.yMessHiRes2lf);
	stringlaenge=GetPrivateProfileString("Graphik", "HeightMessHiRes2lf", "602", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiRes2lf);
	stringlaenge=GetPrivateProfileString("Graphik", "WidthMessHiRes2lf", "341", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiRes2lf);

	stringlaenge=GetPrivateProfileString("Graphik", "xMessHiRes3lf", "682", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.xMessHiRes3lf);
	stringlaenge=GetPrivateProfileString("Graphik", "yMessHiRes3lf", "441", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.yMessHiRes3lf);
	stringlaenge=GetPrivateProfileString("Graphik", "HeightMessHiRes3lf", "602", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiRes3lf);
	stringlaenge=GetPrivateProfileString("Graphik", "WidthMessHiRes3lf", "341", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiRes3lf);

	stringlaenge=GetPrivateProfileString("Graphik", "xMessHiRes4lf", "852", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.xMessHiRes4lf);
	stringlaenge=GetPrivateProfileString("Graphik", "yMessHiRes4lf", "441", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.yMessHiRes4lf);
	stringlaenge=GetPrivateProfileString("Graphik", "HeightMessHiRes4lf", "602", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nHeightMessHiRes4lf);
	stringlaenge=GetPrivateProfileString("Graphik", "WidthMessHiRes4lf", "341", string, 29,
INIFILENAME);
	sscanf(string, "%i", &fensterkoordinaten.nWidthMessHiRes4lf);

	multmediatimerwTimerRes                                                    =
min(max(multmediatimertc.wPeriodMin,50),multmediatimertc.wPeriodMax); // Timer herichten 10ms max
Auflösung
	timeBeginPeriod(multmediatimerwTimerRes);
}

void globalexit(void)
```

```
{
char dataout[100];
unsigned long stringlaenge;
double TimeUnit;

	if(messkontrolle.asl=TRUE) // Schnittstelle des ASL abmelden
	{
		strcpy(dataout,"L0P0R1U0\n"); //Bedienfeld verriegelt, Kanal A ausgewählt, drei Nachkommastellen, °C
		stringlaenge=strlen(dataout);
		WriteFile(messkontrolle.hComasl,dataout,stringlaenge,&stringlaenge,NULL);
		TimeInit(&TimeUnit);

		CloseHandle(messkontrolle.hComasl); // Schnittstelle wieder schließen
	}
	ledmessen(0,defmessparam.ba); // Mess Led aus
	GlobalUnlock(hMemHandle1); //Speicher wieder freigeben
	GlobalFree(hMemHandle1);
	GlobalUnlock(hMemHandle30); //Speicher wieder freigeben
	GlobalFree(hMemHandle30);
}

// Nachrichtenbehandlungsroutine für "Info"-Feld.
LRESULT CALLBACK About( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam )
{
double u1=0,u2=0,u3=0,u4=0;
int data=0;


// Test
char pcCommPort[5];
DCB dcb;
// Test

	switch( message )
	{
		case WM_INITDIALOG:
			return TRUE;

		case WM_COMMAND:
			if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
			{

				sprintf(pcCommPort,"COM%i",messkontrolle.kryocom);
				messkontrolle.hkryo = CreateFile( pcCommPort, GENERIC_READ | GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für LF und Thermo

				dcb.ByteSize = 28;		// data size, xmit, and rcv
				dcb.BaudRate = CBR_4800;	// set the baud rate
				dcb.fBinary=1;				// binary mode, no EOF check
				dcb.fParity=0;			// enable parity checking
				dcb.fOutxCtsFlow=1;			// CTS output flow control
				dcb.fOutxDsrFlow=1;			// DSR output flow control
				dcb.fDtrControl=1;			// DTR flow control type
				dcb.fDsrSensitivity=0;		// DSR sensitivity
				dcb.fTXContinueOnXoff=1; // XOFF continues Tx
				dcb.fOutX=0;			// XON/XOFF out flow control
				dcb.fInX=0;			// XON/XOFF in flow control
				dcb.fErrorChar=0;		// enable error replacement
				dcb.fNull=0;			// enable null stripping
				dcb.fRtsControl=1;		// RTS flow control
```

```
                                    dcb.fAbortOnError=0;     // abort reads/writes on error
                                    dcb.XonLim=2048;              // transmit XON threshold
                                    dcb.XoffLim=512;              // transmit XOFF threshold
                                    dcb.ByteSize=8;             // number of bits/byte, 4-8
                                    dcb.Parity=NOPARITY;              // 0-4=no,odd,even,mark,space
                                    dcb.StopBits=TWOSTOPBITS;             // 0,1,2 = 1, 1.5, 2
                                    dcb.XonChar=17;              // Tx and Rx XON character
                                    dcb.XoffChar=19;             // Tx and Rx XOFF character
                                    dcb.ErrorChar=0;            // error replacement character
                                    dcb.EofChar=0;              // end of input character
                                    dcb.EvtChar='E';             // received event character

                                    SetCommState(messkontrolle.hkryo, &dcb);

        PurgeComm(messkontrolle.hkryo,PURGE_TXCLEAR&&PURGE_RXCLEAR); // Puffer leeren
                                    setkryotemp(20.01);

                                    return TRUE;
                                }
                                break;
        }
   return FALSE;
}

double round(double x)
{
   if((x-floor(x))<=0.5)
      return(floor(x));
   else
      return(ceil(x));
}

double maketemp(double u, int kanal) //berechnet die Temperatur aus der Spannung
{
double theta;

        theta=1/(messkontrolle.kalparam[kanal-1].a          +          messkontrolle.kalparam[kanal-
1].b*log(messkontrolle.kalparam[kanal-1].r1*messkontrolle.kalparam[kanal-1].uref/u-
messkontrolle.kalparam[kanal-1].r1)                +          messkontrolle.kalparam[kanal-
1].c*pow(log(messkontrolle.kalparam[kanal-1].r1*messkontrolle.kalparam[kanal-1].uref/u-
messkontrolle.kalparam[kanal-1].r1),3) );

        return(theta-273.15);
}

double makelf(double u) //berechnet die Leitfähigkeiten
{
double G;

        G=1/(USLF/u*RMLF-RMLF);

        return(G*1000); // Leitfähigkeiten werden in mS innerhalb der Software behandelt.
}

// Nachrichtenbehandlungsroutine für "Spannungsausgabe"-Feld.
LRESULT CALLBACK spannungsausgabe( HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam )
{
char dummy[100];
double u;

        switch( message )
        {
```

```
                case WM_INITDIALOG:
                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        else if(LOWORD(wParam) == IDOK)
                        {
                                GetDlgItemText(hDlg,IDC_SPANNUNG,dummy,99);
        u=atof(dummy);//Spannung einlesen holen
                                u_ausgabe(u,defmessparam.ba);
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        else if(LOWORD(wParam) == IDC_TAKE)
                        {
                                GetDlgItemText(hDlg,IDC_SPANNUNG,dummy,99);
                                u=atof(dummy);//Spannung einlesen holen
                                u_ausgabe(u,defmessparam.ba);
                                return FALSE;
                        }
                        break;
        }
   return FALSE;
}

// setzt die LED-Messen
void ledmessen(int led,int ba)
{
int portbuffer=0;

        portbuffer=_inp(ba); // Status der Schnittstelle sichern, damit die anderen Bits die nicht gebraucht
                                                        //werden erhalten bleiben

        portbuffer=(portbuffer & 191)+64*led; //PIN 8, BIT 7, schaltet LED
        _outp(ba,portbuffer);
}


// setzt Störungsrelais  auf aus und Störungs -LED an
void stoerung(int cond,int ba)
{
int portbuffer=0;

        portbuffer=_inp(ba); // Staus der Schnittstelle sichern, damit die anderen bits die nicht gebraucht
                                                        //werden erhalten bleiben

        portbuffer=(portbuffer & 0x7F)+0x80*(!cond); //PIN 9, BIT 8, schaltet LED-Störung und Relais
        _outp(ba,portbuffer);
}


// Nachrichtenbehandlungsroutine für "32-Kanal-Spannungsausgabe"-Feld.
LRESULT CALLBACK Kanal32spannungsanzeige( HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam )
{
char dummy[100];
double u[33],dummy1[8],dummy2[8],dummy3[8],dummy4[8];
int i=0;
```

```
	switch( message )
	{
		case WM_INITDIALOG:
			{
				sprintf(dummy,"frei");
				SetDlgItemText(hDlg,IDC_UKanal1, dummy);
				SetDlgItemText(hDlg,IDC_UKanal2, dummy);
				SetDlgItemText(hDlg,IDC_UKanal3, dummy);
				SetDlgItemText(hDlg,IDC_UKanal4, dummy);
				SetDlgItemText(hDlg,IDC_UKanal5, dummy);
				SetDlgItemText(hDlg,IDC_UKanal6, dummy);
				SetDlgItemText(hDlg,IDC_UKanal7, dummy);
				SetDlgItemText(hDlg,IDC_UKanal8, dummy);
				SetDlgItemText(hDlg,IDC_UKanal9, dummy);
				SetDlgItemText(hDlg,IDC_UKanal10, dummy);
				SetDlgItemText(hDlg,IDC_UKanal11, dummy);
				SetDlgItemText(hDlg,IDC_UKanal12, dummy);
				SetDlgItemText(hDlg,IDC_UKanal13, dummy);
				SetDlgItemText(hDlg,IDC_UKanal14, dummy);
				SetDlgItemText(hDlg,IDC_UKanal15, dummy);
				SetDlgItemText(hDlg,IDC_UKanal16, dummy);
				SetDlgItemText(hDlg,IDC_UKanal17, dummy);
				SetDlgItemText(hDlg,IDC_UKanal18, dummy);
				SetDlgItemText(hDlg,IDC_UKanal19, dummy);
				SetDlgItemText(hDlg,IDC_UKanal20, dummy);
				SetDlgItemText(hDlg,IDC_UKanal21, dummy);
				SetDlgItemText(hDlg,IDC_UKanal22, dummy);
				SetDlgItemText(hDlg,IDC_UKanal23, dummy);
				SetDlgItemText(hDlg,IDC_UKanal24, dummy);
				SetDlgItemText(hDlg,IDC_UKanal25, dummy);
				SetDlgItemText(hDlg,IDC_UKanal26, dummy);
				SetDlgItemText(hDlg,IDC_UKanal27, dummy);
				SetDlgItemText(hDlg,IDC_UKanal28, dummy);
				SetDlgItemText(hDlg,IDC_UKanal29, dummy);
				SetDlgItemText(hDlg,IDC_UKanal30, dummy);
				SetDlgItemText(hDlg,IDC_UKanal31, dummy);
				SetDlgItemText(hDlg,IDC_UKanal32, dummy);

				return TRUE;
			}

		case WM_COMMAND:
			if (LOWORD(wParam) == IDCANCEL)
			{
				EndDialog(hDlg, LOWORD(wParam));
				return TRUE;
			}
			else if(LOWORD(wParam) == IDUPDATE)
			{
				ledmessen(1,defmessparam.ba);
				ads1241klarmachen(defmessparam.ba);
				startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts

				for(i=0;i<8;i++)
				{

	ads1241getu30(defmessparam.ba,1,i+1,&dummy1[i],&dummy2[i],&dummy3[i],&dummy4[i]);	//
auslesen der Kanäle
				}
				ledmessen(0,defmessparam.ba);
				u[1]=dummy1[0];
				u[2]=dummy4[0];
				u[3]=dummy3[0];
```

```
u[4]=dummy2[0];

u[5]=dummy1[1];
u[6]=dummy4[1];
u[7]=dummy3[1];
u[8]=dummy2[1];

u[9]=dummy1[2];
u[10]=dummy4[2];
u[11]=dummy3[2];
u[12]=dummy2[2];

u[13]=dummy1[3];
u[14]=dummy4[3];
u[15]=dummy3[3];
u[16]=dummy2[3];

u[17]=dummy1[4];
u[18]=dummy4[4];
u[19]=dummy3[4];
u[20]=dummy2[4];

u[21]=dummy1[5];
u[22]=dummy4[5];
u[23]=dummy3[5];
u[24]=dummy2[5];

u[25]=dummy1[6];
u[26]=dummy4[6];
u[27]=dummy3[6];
u[28]=dummy2[6];

u[29]=dummy1[7];
u[30]=dummy4[7];
u[31]=dummy3[7];
u[32]=dummy2[7];

sprintf(dummy,"%.3f V", u[1]);
SetDlgItemText(hDlg,IDC_UKanal1, dummy);
sprintf(dummy,"%.3f V", u[2]);
SetDlgItemText(hDlg,IDC_UKanal2, dummy);
sprintf(dummy,"%.3f V", u[3]);
SetDlgItemText(hDlg,IDC_UKanal3, dummy);
sprintf(dummy,"%.3f V", u[4]);
SetDlgItemText(hDlg,IDC_UKanal4, dummy);
sprintf(dummy,"%.3f V", u[5]);
SetDlgItemText(hDlg,IDC_UKanal5, dummy);
sprintf(dummy,"%.3f V", u[6]);
SetDlgItemText(hDlg,IDC_UKanal6, dummy);
sprintf(dummy,"%.3f V", u[7]);
SetDlgItemText(hDlg,IDC_UKanal7, dummy);
sprintf(dummy,"%.3f V", u[8]);
SetDlgItemText(hDlg,IDC_UKanal8, dummy);
sprintf(dummy,"%.3f V", u[9]);
SetDlgItemText(hDlg,IDC_UKanal9, dummy);
sprintf(dummy,"%.3f V", u[10]);
SetDlgItemText(hDlg,IDC_UKanal10, dummy);
sprintf(dummy,"%.3f V", u[11]);
SetDlgItemText(hDlg,IDC_UKanal11, dummy);
sprintf(dummy,"%.3f V", u[12]);
SetDlgItemText(hDlg,IDC_UKanal12, dummy);
sprintf(dummy,"%.3f V", u[13]);
SetDlgItemText(hDlg,IDC_UKanal13, dummy);
```

```
                                        sprintf(dummy,"%.3f V", u[14]);
                                        SetDlgItemText(hDlg,IDC_UKanal14, dummy);
                                        sprintf(dummy,"%.3f V", u[15]);
                                        SetDlgItemText(hDlg,IDC_UKanal15, dummy);
                                        sprintf(dummy,"%.3f V", u[16]);
                                        SetDlgItemText(hDlg,IDC_UKanal16, dummy);
                                        sprintf(dummy,"%.3f V", u[17]);
                                        SetDlgItemText(hDlg,IDC_UKanal17, dummy);
                                        sprintf(dummy,"%.3f V", u[18]);
                                        SetDlgItemText(hDlg,IDC_UKanal18, dummy);
                                        sprintf(dummy,"%.3f V", u[19]);
                                        SetDlgItemText(hDlg,IDC_UKanal19, dummy);
                                        sprintf(dummy,"%.3f V", u[20]);
                                        SetDlgItemText(hDlg,IDC_UKanal20, dummy);
                                        sprintf(dummy,"%.3f V", u[21]);
                                        SetDlgItemText(hDlg,IDC_UKanal21, dummy);
                                        sprintf(dummy,"%.3f V", u[22]);
                                        SetDlgItemText(hDlg,IDC_UKanal22, dummy);
                                        sprintf(dummy,"%.3f V", u[23]);
                                        SetDlgItemText(hDlg,IDC_UKanal23, dummy);
                                        sprintf(dummy,"%.3f V", u[24]);
                                        SetDlgItemText(hDlg,IDC_UKanal24, dummy);
                                        sprintf(dummy,"%.3f V", u[25]);
                                        SetDlgItemText(hDlg,IDC_UKanal25, dummy);
                                        sprintf(dummy,"%.3f V", u[26]);
                                        SetDlgItemText(hDlg,IDC_UKanal26, dummy);
                                        sprintf(dummy,"%.3f V", u[27]);
                                        SetDlgItemText(hDlg,IDC_UKanal27, dummy);
                                        sprintf(dummy,"%.3f V", u[28]);
                                        SetDlgItemText(hDlg,IDC_UKanal28, dummy);
                                        sprintf(dummy,"%.3f V", u[29]);
                                        SetDlgItemText(hDlg,IDC_UKanal29, dummy);
                                        sprintf(dummy,"%.3f V", u[30]);
                                        SetDlgItemText(hDlg,IDC_UKanal30, dummy);
                                        sprintf(dummy,"%.3f V", u[31]);
                                        SetDlgItemText(hDlg,IDC_UKanal31, dummy);
                                        sprintf(dummy,"%.3f V", u[32]);
                                        SetDlgItemText(hDlg,IDC_UKanal32, dummy);

                                }
                                break;
        }
   return FALSE;
}


// Nachrichtenbehandlungsroutine für "Options"-Feld.
LRESULT CALLBACK optionen( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam )
{
char dummy[MAX_PATH];;

        switch( message )
        {
                case WM_INITDIALOG:

                        if(messkontrolle.asl==TRUE)
                                CheckDlgButton(hDlg,IDC_CHECKASL,1);
                        else
                                CheckDlgButton(hDlg,IDC_CHECKASL,0);

                        if(messkontrolle.wtw==TRUE)
                                CheckDlgButton(hDlg,IDC_CHECKWTW,1);
                        else
```

```
                        CheckDlgButton(hDlg,IDC_CHECKWTW,0);

                    if(messkontrolle.thermolf==TRUE)
                        CheckDlgButton(hDlg,IDC_CHECKTHERMOLF,1);
                    else
                        CheckDlgButton(hDlg,IDC_CHECKTHERMOLF,0);

                    if(messkontrolle.kryo==TRUE)
                        CheckDlgButton(hDlg,IDC_CHECKKRYO,1);
                    else
                        CheckDlgButton(hDlg,IDC_CHECKKRYO,0);


                    if(messkontrolle.ruehrtyp==ZAHNRAD)

 CheckDlgButton(hDlg,IDC_RUERGESCHWINDIGKEITZAHNRAD,1);
                    else if (messkontrolle.ruehrtyp==WANDELFELD)
                        CheckDlgButton(hDlg,IDC_WANDELFELDRUEHRER,1);


                    if(messkontrolle.typ==KANAL30)
                        CheckDlgButton(hDlg,IDC_32KANALBUTTON,1);
                    else if (messkontrolle.typ==KANAL30TLF)
                        CheckDlgButton(hDlg,IDC_32TLFBUTTON,1);
                    else
                    {
                        CheckDlgButton(hDlg,IDC_8Kanalbutton,1);
                    }

                    sprintf(dummy,"0x%x",defmessparam.ba);
                    SetDlgItemText(hDlg,IDC_LPTPORT,dummy);

                    SetDlgItemText(hDlg,IDC_PFAD,defmessparam.tempfile);

                    sprintf(dummy,"%.1lf",messkontrolle.maxbathtemp);
                    SetDlgItemText(hDlg,IDC_MAXTEMP,dummy);

                    sprintf(dummy,"%i",messkontrolle.aslcom);
                    SetDlgItemText(hDlg,IDC_COMASL,dummy);

                    sprintf(dummy,"%.1lf",messkontrolle.frequenz);
                    SetDlgItemText(hDlg,IDC_FREQUENZ,dummy);

                    if(messkontrolle.frequenz<=0)
{
  MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben", "Achtung", MB_OK|MB_ICONSTOP);
  return FALSE;
}

                    if(messkontrolle.aslcom<=0)
{
  MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben", "Achtung", MB_OK|MB_ICONSTOP);
  return FALSE;
}

                    sprintf(dummy,"%i",messkontrolle.wtwcom);
                    SetDlgItemText(hDlg,IDC_COMWTW,dummy);

                    if(messkontrolle.wtwcom<=0)
{
  MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben", "Achtung", MB_OK|MB_ICONSTOP);
  return FALSE;
}
```

```
                        sprintf(dummy,"%i",messkontrolle.thermolfcom);
                        SetDlgItemText(hDlg,IDC_COMLFTHERMO,dummy);

                        if(messkontrolle.wtwcom<=0)
        {
          MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben", "Achtung", MB_OK|MB_ICONSTOP);
          return FALSE;
        }


                        sprintf(dummy,"%i",messkontrolle.motor);
                        SetDlgItemText(hDlg,IDC_RUEHR,dummy);
                        if(messkontrolle.motor<=0)
        {
          MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben", "Achtung", MB_OK|MB_ICONSTOP);
          return FALSE;
        }
                        else if(messkontrolle.motor>100)
        {
          MessageBox(hDlg, "Keine Werte größer als 100 eingeben", "Achtung",
MB_OK|MB_ICONSTOP);
          return FALSE;
        }

                        sprintf(dummy,"%i",messkontrolle.kryocom);
                        SetDlgItemText(hDlg,IDC_COMKRYO,dummy);

                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        else if(LOWORD(wParam) == IDOK)
                        {
                                // Checkboxen der Comports
                                if(IsDlgButtonChecked(hDlg,IDC_CHECKASL)==BST_CHECKED)
                                        messkontrolle.asl=TRUE;
                                else
                                        messkontrolle.asl=FALSE;

        if(IsDlgButtonChecked(hDlg,IDC_CHECKWTW)==BST_CHECKED)
                                        messkontrolle.wtw=TRUE;
                                else
                                        messkontrolle.wtw=FALSE;


        if(IsDlgButtonChecked(hDlg,IDC_CHECKTHERMOLF)==BST_CHECKED)
                                        messkontrolle.thermolf=TRUE;
                                else
                                        messkontrolle.thermolf=FALSE;


        if(IsDlgButtonChecked(hDlg,IDC_CHECKKRYO)==BST_CHECKED)
                                        messkontrolle.kryo=TRUE;
                                else
                                        messkontrolle.kryo=FALSE;

                                GetDlgItemText(hDlg,IDC_COMLFTHERMO,dummy,99);
                                sscanf(dummy, "%i", &messkontrolle.thermolfcom);
```

```
                                GetDlgItemText(hDlg,IDC_COMASL,dummy,99);
                                sscanf(dummy, "%i", &messkontrolle.aslcom);
                                GetDlgItemText(hDlg,IDC_COMWTW,dummy,99);
                                sscanf(dummy, "%i", &messkontrolle.wtwcom);
                                GetDlgItemText(hDlg,IDC_COMKRYO,dummy,99);
                                sscanf(dummy, "%i", &messkontrolle.kryocom);

                                // Comports überprüfen
                                if(messkontrolle.aslcom<=0)
                                {
                                        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                        return FALSE;
                                }
                                if(messkontrolle.wtwcom<=0)
                                {
                                        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                        return FALSE;
                                }
                                if(messkontrolle.thermolfcom <=0)
                                {
                                        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                        return FALSE;
                                }
                                if(messkontrolle.kryocom <=0)
                                {
                                        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                        return FALSE;
                                }

                                // Thermometer Basisadresse
                                GetDlgItemText(hDlg,IDC_LPTPORT,dummy,99);
                                sscanf(dummy, "%x", &defmessparam.ba);
                                if(defmessparam.ba <=0)
                                {
                                        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                        return FALSE;
                                }

                                GetDlgItemText(hDlg,IDC_MAXTEMP,dummy,99);
                                sscanf(dummy, "%lf", &messkontrolle.maxbathtemp);

                                // Motor
                                GetDlgItemText(hDlg,IDC_RUEHR,dummy,99);
                                sscanf(dummy, "%i", &messkontrolle.motor);
                                if(messkontrolle.motor<=0)
                                {
                                        MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                        return FALSE;
                                }
                                else if(messkontrolle.motor>100)
                                {
                                        MessageBox(hDlg, "Keine Werte größer als 100 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                        return FALSE;
                                }

                                // Tempfile
```

```
                                        GetDlgItemText(hDlg,IDC_PFAD,defmessparam.tempfile,99);

                                        // Frequenz
                                        GetDlgItemText(hDlg,IDC_FREQUENZ,dummy,99);
                                        sscanf(dummy, "%lf", &messkontrolle.frequenz);
                                        if(messkontrolle.frequenz<=0)
                                        {
                                                MessageBox(hDlg, "Keine Werte kleiner als 0 eingeben",
"Achtung", MB_OK|MB_ICONSTOP);
                                                return FALSE;
                                        }

                                        // Radiobuttons der Rührwerke


        if(IsDlgButtonChecked(hDlg,IDC_RUERGESCHWINDIGKEITZAHNRAD)==BST_CHECKE
D)
                                                messkontrolle.ruehrtyp=ZAHNRAD;
                                        else                                                    if
(IsDlgButtonChecked(hDlg,IDC_WANDELFELDRUEHRER)==BST_CHECKED)
                                                messkontrolle.ruehrtyp=WANDELFELD;
                                        else
                                                messkontrolle.ruehrtyp=0;

                                        // Radiobuttons der Messgeräte

                                        if(IsDlgButtonChecked(hDlg,IDC_8Kanalbutton)==BST_CHECKED)
                                                messkontrolle.typ=KANAL8;
                                        else                                                    if
(IsDlgButtonChecked(hDlg,IDC_32KANALBUTTON)==BST_CHECKED)
                                                messkontrolle.typ=KANAL30;
                                        else
                                                messkontrolle.typ=KANAL30TLF;


                                        EndDialog(hDlg, LOWORD(wParam));
                                        return TRUE;
                                }
                                break;

        }
   return FALSE;
}

void writeini(void)
{
char dummy[MAX_PATH];

        sprintf(dummy,"0x%x",defmessparam.ba);
        WritePrivateProfileString("Hardware", "LPT-Basisadresse Thermometer",dummy,INIFILENAME);

        sprintf(dummy,"0x%x",defmessparam.balf);
        WritePrivateProfileString("Hardware", "LPT-Basisadresse Leitfaehigkeit",dummy,INIFILENAME);

        WritePrivateProfileString("Hardware", "Tempfile",defmessparam.tempfile,INIFILENAME);

        sprintf(dummy,"%i",messkontrolle.asl);
        WritePrivateProfileString("Hardware", "ASL",dummy,INIFILENAME);
        sprintf(dummy,"%i",messkontrolle.aslcom);
        WritePrivateProfileString("Hardware", "ASLCOMM",dummy,INIFILENAME);

        sprintf(dummy,"%i",messkontrolle.wtw);
        WritePrivateProfileString("Hardware", "WTW",dummy,INIFILENAME);
```

```
	sprintf(dummy,"%i",messkontrolle.wtwcom);
	WritePrivateProfileString("Hardware", "WTWCOMM",dummy,INIFILENAME);

	sprintf(dummy,"%i",messkontrolle.thermolf);
	WritePrivateProfileString("Hardware", "THERMOLF",dummy,INIFILENAME);
	sprintf(dummy,"%i",messkontrolle.thermolfcom);
	WritePrivateProfileString("Hardware", "THERMOLFCOMM",dummy,INIFILENAME);

	sprintf(dummy,"%i",messkontrolle.kryo);
	WritePrivateProfileString("Hardware", "KRYO",dummy,INIFILENAME);
	sprintf(dummy,"%i",messkontrolle.kryocom);
	WritePrivateProfileString("Hardware", "KRYOCOMM",dummy,INIFILENAME);

	sprintf(dummy,"%.1lf",messkontrolle.maxbathtemp);
	WritePrivateProfileString("Hardware", "Maximaletemperatur",dummy,INIFILENAME);

	sprintf(dummy,"%i",messkontrolle.motor);
	WritePrivateProfileString("Hardware", "MOTOR",dummy,INIFILENAME);

	sprintf(dummy,"%.1lf",messkontrolle.frequenz);
	WritePrivateProfileString("Hardware", "FREQUENZ",dummy,INIFILENAME);

	sprintf(dummy,"%i",messkontrolle.ruehrtyp);
	WritePrivateProfileString("Hardware", "RUEHRTYP",dummy,INIFILENAME);

	sprintf(dummy,"%i",messkontrolle.typ);
	WritePrivateProfileString("Hardware", "MESSTYP",dummy,INIFILENAME);
}

// schaltet passende Menüs aus, wenn Messung läuft
void setmenue(HWND hWnd)
{
	if(messkontrolle.messan==FALSE) // keine Messung, dann alle Menüs verfügbar
	{

	EnableMenuItem(GetMenu(hWnd),ID_MESSUNGEN30KANAL_KALIBRIERUNG,MF_ENAB
LED);
		EnableMenuItem(GetMenu(hWnd),ID_MESSUNG_ABKHLKURVE,MF_ENABLED);
		EnableMenuItem(GetMenu(hWnd),ID_MESSUNG_ASL,MF_ENABLED);

	EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN_1KANALUTSCHREIBER,MF_
ENABLED);

	EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN_8KANALUTSCHREIBER,MF_
ENABLED);

	EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN_2KANALUT,MF_ENABLED);

	EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN32KANALSPANNUNG,MF_E
NABLED);

	EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN_SPANNUNGSAUSGABE,MF_
ENABLED);
		EnableMenuItem(GetMenu(hWnd),IDM_EXTRAS_OPTIONEN,MF_ENABLED);

	EnableMenuItem(GetMenu(hWnd),IDM_EXTRAS_OPTIONENSPEICHERN,MF_ENABLED);
		EnableMenuItem(GetMenu(hWnd),IDM_ABOUT,MF_ENABLED);
		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_MESS1,MF_ENABLED);
		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_MESS2,MF_ENABLED);
		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_MESS3,MF_ENABLED);
		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_MESS4,MF_ENABLED);
		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_STATUS,MF_ENABLED);
```

```
		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_HOCHAUFGELST1,MF_ENABLED);

		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_HOCHAUFGELST2,MF_ENABLED);

		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_HOCHAUFGELST3,MF_ENABLED);

		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_HOCHAUFGELST4,MF_ENABLED);
		}
		else
		{

		EnableMenuItem(GetMenu(hWnd),ID_MESSUNGEN30KANAL_KALIBRIERUNG,MF_GRA
YED);
			EnableMenuItem(GetMenu(hWnd),ID_MESSUNG_ABKHLKURVE,MF_GRAYED);
			EnableMenuItem(GetMenu(hWnd),ID_MESSUNG_ASL,MF_GRAYED);

		EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN_1KANALUTSCHREIBER,MF_
GRAYED);

		EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN_8KANALUTSCHREIBER,MF_
GRAYED);

		EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN_2KANALUT,MF_GRAYED);

		EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN32KANALSPANNUNG,MF_G
RAYED);

		EnableMenuItem(GetMenu(hWnd),IDM_TESTFUNKTIONEN_SPANNUNGSAUSGABE,MF_
GRAYED);
			EnableMenuItem(GetMenu(hWnd),IDM_EXTRAS_OPTIONEN,MF_GRAYED);

		EnableMenuItem(GetMenu(hWnd),IDM_EXTRAS_OPTIONENSPEICHERN,MF_GRAYED);
			EnableMenuItem(GetMenu(hWnd),IDM_ABOUT,MF_GRAYED);
			if(messkontrolle.messtyp!=KANAL30TU)
			{
				EnableMenuItem(GetMenu(hWnd),ID_FENSTER_MESS1,MF_GRAYED);
				EnableMenuItem(GetMenu(hWnd),ID_FENSTER_MESS2,MF_GRAYED);
				EnableMenuItem(GetMenu(hWnd),ID_FENSTER_MESS3,MF_GRAYED);
				EnableMenuItem(GetMenu(hWnd),ID_FENSTER_MESS4,MF_GRAYED);
				EnableMenuItem(GetMenu(hWnd),ID_FENSTER_STATUS,MF_GRAYED);

		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_HOCHAUFGELST1,MF_GRAYED);

		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_HOCHAUFGELST2,MF_GRAYED);

		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_HOCHAUFGELST3,MF_GRAYED);

		EnableMenuItem(GetMenu(hWnd),ID_FENSTER_HOCHAUFGELST4,MF_GRAYED);
			}

		}
}

double maketempstandard(double u) //berechnet die Temperatur aus der Spannung für Standardthermistor
{
double theta;

		theta=1/(THERMA + THERMB*log(R1*UREF/u-R1) + THERMC*pow(log(R1*UREF/u-R1),3)
);
		return(theta-273.15);
}
```

```
int FileDialogOpen(HWND hWnd, OPENFILENAME *ofn) // Filedialog für das Öffnen von Dateien
{
      char szFilter[256], szText[]= "Text-Datei (*.TXT)|*.txt|Alle Dateien (*.*)|*.*||";
      int i;

      for(i=0; szText[i]!= '\0'; ++i)
        szFilter[i] = szText[i] == '|' ? '\0' : szText[i];

      ofn->lStructSize = sizeof(OPENFILENAME);
      ofn->hwndOwner = hWnd;
      ofn->lpstrFilter = szFilter;
      ofn->nMaxFile = _MAX_PATH;
      ofn->nMaxFileTitle = _MAX_FNAME + _MAX_EXT;
      ofn->lpstrDefExt = "txt";
         return GetOpenFileName(ofn)? IDOK : IDCANCEL;
}

int kalibdatensatz(HWND hWnd) // liest einzelne Datenfiles ein, und erstellt den Kalibrierdatensatz
{
static char  szFilename[MAX_PATH] = ".txt";
int iErg=0,j=0,k=0,result;
long int i,count=0,error;
HGLOBAL hMemHandleDaten;
HANDLE hFilekalib,hFiledata;
OPENFILENAME ofn,ofndata;
char *feld,dummy,zeile[540],zahl[20];
DWORD nummer=1;
unsigned long filesize;
DWORD dummy2=0;
static int l=0;

          /* Datei, in die die Kalibrierdaten geschrieben werden sollen ausgewählt werden, Daten werden
angehängt*/
          MessageBox(hWnd, "Wählen Sie die Datei aus, in die die Kalibrierdaten geschrieben werden sollen",
"Achtung", MB_OK|MB_ICONINFORMATION);
   memset(&ofn, 0, sizeof(OPENFILENAME));
   ofn.lpstrFile = szFilename;
   iErg = FileDialogOpen(hWnd, &ofn);
          if(iErg == IDOK)
   {
                    hFilekalib=CreateFile(ofn.lpstrFile,GENERIC_WRITE,0,
NULL,OPEN_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
      if( hFilekalib == INVALID_HANDLE_VALUE)
      {
         MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK);
         return(FEHLER);
      }
      // an das Dateiende gehen
                    SetFilePointer(hFilekalib,0,0,FILE_END);

   }
          else
                    return(FEHLER);

   hMemHandleDaten=GlobalAlloc(GHND,sizeof(char)*MAXMESSDAUER*536);      //Größe      des
Charfeldes
   feld=(char*)GlobalLock(hMemHandleDaten);

          do
          {
                    if(l==0)
                    {
```

```
                    MessageBox(hWnd, "Wählen Sie eine Datei aus, die Messdaten enthält",
"Achtung", MB_OK|MB_ICONINFORMATION);
                    memset(&ofndata, 0, sizeof(OPENFILENAME));
                    ofndata.lpstrFile = szFilename;
                    iErg = FileDialogOpen(hWnd, &ofndata);
                    l=1;
                }
                else
                {
                    if(DialogBox(hInst, (LPCTSTR)IDD_MITTEL,hWnd, (DLGPROC)
Kanal32mittelwert)==IDCANCEL)
                    {
                        iErg=IDCANCEL;
                    }
                    else
                    {
                        memset(&ofndata, 0, sizeof(OPENFILENAME));
                        ofndata.lpstrFile = szFilename;
                        iErg = FileDialogOpen(hWnd, &ofndata);
                    }
                }

                if(iErg == IDOK) // Daten einlesen, solange Dateien ausgewählt werden
                {
                    hFiledata=CreateFile(ofndata.lpstrFile,GENERIC_READ,0,
NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
                    if( hFilekalib == INVALID_HANDLE_VALUE)
                    {
                        MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK);
                        CloseHandle(hFiledata);
                      GlobalUnlock(hMemHandleDaten);
                        GlobalFree(hMemHandleDaten);
                    }
                    else
                    {
                        filesize=GetFileSize(hFiledata,NULL);
                        for(i=0;i<=33;i++)
                        {
                            werte[i]=0;
                        }
                        SetFilePointer(hFiledata,0,0,FILE_BEGIN); // an den Dateianfang
gehen
                        count=0;
                        SetLastError(0);
                        do // Daten einlesen
                        {
                            i=0;
                            do // Zeile einlesen
                            {
                                nummer=1;// immer genau ein Zeichen einlesen

        result=ReadFile(hFiledata,&dummy,1,&nummer,NULL);
                                zeile[i++]=dummy;
                            }
                            while(dummy!='\r');
                            error=GetLastError();
                            if((nummer !=0) && (result !=0) )
                            {
                                i=0;
                                j=0;
                                k=0;
                                for(i=0;i<33;i++)
                                {
```

```
                                                j=0;

                                                do // Zeile parsen
                                                {
                                                        zahl[j++]=zeile[k++];
                                                }
                                                while(zeile[k-1]!=' ');
                                                zahl[j]=0;
                                                //Messdaten30[count].t[i]=atof(zahl);

                                                j=0;;
                                                do // Zeile parsen
                                                {
                                                        zahl[j++]=zeile[k++];
                                                }
                                                while(zeile[k-1]!=' ');
                                                zahl[j]=0;
                                                werte[i]=werte[i]+atof(zahl);
                                        }
                                        count++;

                                }
                                else
                                        break;

                                zeile[i]=0;
                                SetFilePointer(hFiledata,1,0,FILE_CURRENT); // ein Zeichen
weiter
                        }
                        while((nummer !=0) && (result !=0)); // solange Daten in Datei
                }
                CloseHandle(hFiledata);
                zahl[0]=0;
                zeile[0]=0;
                for(j=0;j<33;j++) // Mittelwerte bilden
                {
                        werte[j]=werte[j]/count;
                        sprintf(zahl," %.8lf ",werte[j]);
                        strcat(zeile,zahl);
                }
                strcat(zeile,"\r\n");
                WriteFile(hFilekalib,zeile,sizeof(char)*strlen(zeile),&dummy2,NULL);
        }
    }
    while(iErg==IDOK);

    CloseHandle(hFilekalib);
  GlobalUnlock(hMemHandleDaten);
  GlobalFree(hMemHandleDaten);
    l=0;
  return(OK);
}

// Nachrichtenbehandlungsroutine für Mittelwertbildung
LRESULT CALLBACK Kanal32mittelwert( HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam )
{
char dummy[100];
int i=0;

    switch( message )
    {
        case WM_INITDIALOG:
```

```
{
        sprintf(dummy,"%.5f V", werte[0]);
        SetDlgItemText(hDlg,IDC_UKanal1, dummy);
        sprintf(dummy,"%.5f V", werte[1]);
        SetDlgItemText(hDlg,IDC_UKanal2, dummy);
        sprintf(dummy,"%.5f V", werte[2]);
        SetDlgItemText(hDlg,IDC_UKanal3, dummy);
        sprintf(dummy,"%.5f V", werte[3]);
        SetDlgItemText(hDlg,IDC_UKanal4, dummy);
        sprintf(dummy,"%.5f V", werte[4]);
        SetDlgItemText(hDlg,IDC_UKanal5, dummy);
        sprintf(dummy,"%.5f V", werte[5]);
        SetDlgItemText(hDlg,IDC_UKanal6, dummy);
        sprintf(dummy,"%.5f V", werte[6]);
        SetDlgItemText(hDlg,IDC_UKanal7, dummy);
        sprintf(dummy,"%.5f V", werte[7]);
        SetDlgItemText(hDlg,IDC_UKanal8, dummy);
        sprintf(dummy,"%.5f V", werte[8]);
        SetDlgItemText(hDlg,IDC_UKanal9, dummy);
        sprintf(dummy,"%.5f V", werte[9]);
        SetDlgItemText(hDlg,IDC_UKanal10, dummy);
        sprintf(dummy,"%.5f V", werte[10]);
        SetDlgItemText(hDlg,IDC_UKanal11, dummy);
        sprintf(dummy,"%.5f V", werte[11]);
        SetDlgItemText(hDlg,IDC_UKanal12, dummy);
        sprintf(dummy,"%.5f V", werte[12]);
        SetDlgItemText(hDlg,IDC_UKanal13, dummy);
        sprintf(dummy,"%.5f V", werte[13]);
        SetDlgItemText(hDlg,IDC_UKanal14, dummy);
        sprintf(dummy,"%.5f V", werte[14]);
        SetDlgItemText(hDlg,IDC_UKanal15, dummy);
        sprintf(dummy,"%.5f V", werte[15]);
        SetDlgItemText(hDlg,IDC_UKanal16, dummy);
        sprintf(dummy,"%.5f V", werte[16]);
        SetDlgItemText(hDlg,IDC_UKanal17, dummy);
        sprintf(dummy,"%.5f V", werte[17]);
        SetDlgItemText(hDlg,IDC_UKanal18, dummy);
        sprintf(dummy,"%.5f V", werte[18]);
        SetDlgItemText(hDlg,IDC_UKanal19, dummy);
        sprintf(dummy,"%.5f V", werte[19]);
        SetDlgItemText(hDlg,IDC_UKanal20, dummy);
        sprintf(dummy,"%.5f V", werte[20]);
        SetDlgItemText(hDlg,IDC_UKanal21, dummy);
        sprintf(dummy,"%.5f V", werte[21]);
        SetDlgItemText(hDlg,IDC_UKanal22, dummy);
        sprintf(dummy,"%.5f V", werte[22]);
        SetDlgItemText(hDlg,IDC_UKanal23, dummy);
        sprintf(dummy,"%.5f V", werte[23]);
        SetDlgItemText(hDlg,IDC_UKanal24, dummy);
        sprintf(dummy,"%.5f V", werte[24]);
        SetDlgItemText(hDlg,IDC_UKanal25, dummy);
        sprintf(dummy,"%.5f V", werte[25]);
        SetDlgItemText(hDlg,IDC_UKanal26, dummy);
        sprintf(dummy,"%.5f V", werte[26]);
        SetDlgItemText(hDlg,IDC_UKanal27, dummy);
        sprintf(dummy,"%.5f V", werte[27]);
        SetDlgItemText(hDlg,IDC_UKanal28, dummy);
        sprintf(dummy,"%.5f V", werte[28]);
        SetDlgItemText(hDlg,IDC_UKanal29, dummy);
        sprintf(dummy,"%.5f V", werte[29]);
        SetDlgItemText(hDlg,IDC_UKanal30, dummy);
        sprintf(dummy,"%.5f V", werte[30]);
        SetDlgItemText(hDlg,IDC_UKanal31, dummy);
```

```
                                sprintf(dummy,"%.5f V", werte[31]);
                                SetDlgItemText(hDlg,IDC_UKanal32, dummy);
                                sprintf(dummy,"%.5f V", werte[32]);
                                SetDlgItemText(hDlg,IDC_ASL, dummy);
                                return TRUE;
                        }

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return IDCANCEL;
                        }
                        else if(LOWORD(wParam) == IDOK)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                break;
        }
   return FALSE;
}

void motormenue(HMENU hMenu) // Menüfunktion zum Motor ein- und ausschalten
{
        if(messkontrolle.typ!=KANAL30TLF) // nicht bei Therm- und LF- Kopplung
        {
                if(messkontrolle.motoran==FALSE)
                {
                        CheckMenuItem(hMenu,ID_EXTRAS_MOTOR,MF_CHECKED);
                        motor(messkontrolle.motor,defmessparam.ba); // Motor an
                        messkontrolle.motoran=TRUE;
                }
                else
                {
                        CheckMenuItem(hMenu,ID_EXTRAS_MOTOR,MF_UNCHECKED);
                        motor(0,defmessparam.ba); // Motor aus
                        messkontrolle.motoran=FALSE;
                }
        }

}

void thermomenue(HMENU hMenu) // Menüfunktion zum Thermostat ein- und ausschalten
{
        if(messkontrolle.typ!=KANAL30TLF) // nicht bei Therm- und LF- Kopplung
        {
                if(messkontrolle.thermoan==FALSE)
                {
                        CheckMenuItem(hMenu,ID_EXTRAS_THERMOSTAT,MF_CHECKED);
                        stoerung(0,defmessparam.ba); //Thermo an
                        messkontrolle.thermoan=TRUE;
                }
                else
                {
                        CheckMenuItem(hMenu,ID_EXTRAS_THERMOSTAT,MF_UNCHECKED);
                        stoerung(1,defmessparam.ba); //Thermo aus
                        messkontrolle.thermoan=FALSE;
                }
        }
}

void motormenuecomm(HMENU hMenu) // Menüfunktion zum Motor ein- und ausschalten
```

```
{
DCB dcb;
char pcCommPort[5];

	if(messkontrolle.typ==KANAL30TLF) // nur bei Thermo- und LF- Kopplung
	{

		sprintf(pcCommPort,"COM%i",messkontrolle.thermolfcom);
		messkontrolle.hComthermolf = CreateFile( pcCommPort, GENERIC_READ |
GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für LF und Thermo

		if(messkontrolle.hComthermolf==INVALID_HANDLE_VALUE)
		{
			if(messkontrolle.asl==TRUE)
			{
				CloseHandle(messkontrolle.hComthermolf);
			}
			return;
		}

		dcb.ByteSize = 28;		// data size, xmit, and rcv
		dcb.BaudRate = CBR_19200;	// set the baud rate
		dcb.fBinary=1;					// binary mode, no EOF check
		dcb.fParity=0;		// enable parity checking
		dcb.fOutxCtsFlow=0;			// CTS output flow control
		dcb.fOutxDsrFlow=0;				// DSR output flow control
		dcb.fDtrControl=0;				// DTR flow control type
		dcb.fDsrSensitivity=0;	// DSR sensitivity
		dcb.fTXContinueOnXoff=1; // XOFF continues Tx
		dcb.fOutX=0;		// XON/XOFF out flow control
		dcb.fInX=0;		// XON/XOFF in flow control
		dcb.fErrorChar=0;	// enable error replacement
		dcb.fNull=0;		// enable null stripping
		dcb.fRtsControl=0;	// RTS flow control
		dcb.fAbortOnError=0;	// abort reads/writes on error
		dcb.XonLim=2048;		// transmit XON threshold
		dcb.XoffLim=512;		// transmit XOFF threshold
		dcb.ByteSize=8;		// number of bits/byte, 4-8
		dcb.Parity=NOPARITY;		// 0-4=no,odd,even,mark,space
		dcb.StopBits='TWOSTOPBITS;		// 0,1,2 = 1, 1.5, 2
		dcb.XonChar=17;		// Tx and Rx XON character
		dcb.XoffChar=19;		// Tx and Rx XOFF character
		dcb.ErrorChar=0;		// error replacement character
		dcb.EofChar=0;		// end of input character
		dcb.EvtChar='E';		// received event character

		if(messkontrolle.ruehrtyp==ZAHNRAD) // bei Zahnradrührer
		{
			if(messkontrolle.motoran==FALSE)
			{

	CheckMenuItem(hMenu,ID_EXTRAS_MOTORCOMM,MF_CHECKED);
				zahnradruehrer(messkontrolle.motor);// Motor an
				messkontrolle.motoran=TRUE;
			}
			else
			{

	CheckMenuItem(hMenu,ID_EXTRAS_MOTORCOMM,MF_UNCHECKED);
				zahnradruehrer(0); // Motor aus
				messkontrolle.motoran=FALSE;
			}
		}
```

```
		else if(messkontrolle.ruehrtyp==WANDELFELD)
		{
			if(messkontrolle.motoran==FALSE)
			{

	CheckMenuItem(hMenu,ID_EXTRAS_MOTORCOMM,MF_CHECKED);
				wandelfeld(1); // Motor an
				messkontrolle.motoran=TRUE;
			}
			else
			{

	CheckMenuItem(hMenu,ID_EXTRAS_MOTORCOMM,MF_UNCHECKED);
				wandelfeld(0); // Motor aus
				messkontrolle.motoran=FALSE;
			}

		}
		CloseHandle(messkontrolle.hComthermolf);
	}

}

void thermomenuecomm(HMENU hMenu) // Menüfunktion zum Thermostat ein- und ausschalten
{
DCB dcb;
char pcCommPort[5];

	if(messkontrolle.typ==KANAL30TLF) // nur bei Thermo- und LF- Kopplung
	{
		sprintf(pcCommPort,"COM%i",messkontrolle.thermolfcom);
		messkontrolle.hComthermolf = CreateFile( pcCommPort, GENERIC_READ |
GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für LF und Thermo

		if(messkontrolle.hComthermolf==INVALID_HANDLE_VALUE)
		{
			if(messkontrolle.asl==TRUE)
			{
				CloseHandle(messkontrolle.hComthermolf);
			}
			return;
		}

		dcb.ByteSize = 28;		// data size, xmit, and rcv
		dcb.BaudRate = CBR_19200;	// set the baud rate
		dcb.fBinary=1;					// binary mode, no EOF check
		dcb.fParity=0;		// enable parity checking
		dcb.fOutxCtsFlow=0;			// CTS output flow control
		dcb.fOutxDsrFlow=0;				// DSR output flow control
		dcb.fDtrControl=0;				// DTR flow control type
		dcb.fDsrSensitivity=0;	// DSR sensitivity
		dcb.fTXContinueOnXoff=1; // XOFF continues Tx
		dcb.fOutX=0;		// XON/XOFF out flow control
		dcb.fInX=0;		// XON/XOFF in flow control
		dcb.fErrorChar=0;	// enable error replacement
		dcb.fNull=0;		// enable null stripping
		dcb.fRtsControl=0;	// RTS flow control
		dcb.fAbortOnError=0;	// abort reads/writes on error
		dcb.XonLim=2048;		// transmit XON threshold
		dcb.XoffLim=512;		// transmit XOFF threshold
		dcb.ByteSize=8;		// number of bits/byte, 4-8
		dcb.Parity=NOPARITY;		// 0-4=no,odd,even,mark,space
		dcb.StopBits=TWOSTOPBITS;		// 0,1,2 = 1, 1.5, 2
```

```
                dcb.XonChar=17;            // Tx and Rx XON character
                dcb.XoffChar=19;           // Tx and Rx XOFF character
                dcb.ErrorChar=0;          // error replacement character
                dcb.EofChar=0;            // end of input character
                dcb.EvtChar='E';           // received event character

                if(messkontrolle.thermoan==FALSE)
                {

        CheckMenuItem(hMenu,ID_EXTRAS_THERMOSTATCOMM,MF_CHECKED);
                        setstoerung(0); //Thermo an
                        messkontrolle.thermoan=TRUE;
                }
                else
                {

        CheckMenuItem(hMenu,ID_EXTRAS_THERMOSTATCOMM,MF_UNCHECKED);
                        setstoerung(1); //Thermo aus
                        messkontrolle.thermoan=FALSE;
                }
                CloseHandle(messkontrolle.hComthermolf);
        }
}


// Nachrichtenbehandlungsroutine für "32-Kanal-Temperaturanzeige"-Feld.
LRESULT CALLBACK Kanal32temperaturanzeige( HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam )
{
char dummy[100];
double u[33],dummy1[8],dummy2[8],dummy3[8],dummy4[8];
int j=0,k,iErg,readcount=0;
long int i=0;
OPENFILENAME ofn;
static char  szFilename[MAX_PATH] = ".kal";
HFILE hFile;
char readdata[300],read1;

        switch( message )
        {
                case WM_INITDIALOG:
                        {
                                memset(&ofn, 0, sizeof(OPENFILENAME));
                                ofn.lpstrFile = szFilename;
                                iErg = FileDialogKal(hDlg, &ofn);

                                if(iErg == IDOK)
                                {
                                        //Datei öffnen mit Fehlertest
                                        if((hFile = _lopen(ofn.lpstrFile, OF_READ)) == -1)
                                        {
                                                MessageBox(hDlg, "Fehler beim Dateneinlesen",
"Achtung", MB_OK|MB_ICONSTOP);
                                                EndDialog(hDlg, LOWORD(wParam));
                                                return TRUE;
                                        }

                                        //an den Dateianfang gehen
                                        _llseek(hFile, 0L, 0);

                                        // Datei lesen,  Fehlerprüfung hinzufügen

                                        for(k=0;k<32;k++)
```

```
                                        {
                                                readcount=0;
                                                do //eine Zeile einlesen
                                                {
                                                        _hread(hFile,&read1,1);
                                                        readdata[readcount++]=read1;
                                                }
                                                while(readdata[readcount-1]!=13 ); //Zeilenende

                                                readdata[readcount-1]=0; //String beenden
                                                _hread(hFile,&read1,1);  //  noch  Zeilenvorschub
einlesen

                                                sscanf(readdata,"%le  %le  %le  %le  %le
",&messkontrolle.kalparam[k].a,&messkontrolle.kalparam[k].b,&messkontrolle.kalparam[k].c,&messkontrolle.
kalparam[k].uref,&messkontrolle.kalparam[k].r1);
                                        }

                                        _lclose(hFile);

                                }
                                else
                                {
                                        EndDialog(hDlg, LOWORD(wParam));
                                        return TRUE;
                                }

                                sprintf(dummy,"frei");
                                SetDlgItemText(hDlg,IDC_UKanal1, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal2, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal3, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal4, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal5, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal6, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal7, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal8, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal9, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal10, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal11, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal12, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal13, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal14, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal15, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal16, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal17, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal18, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal19, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal20, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal21, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal22, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal23, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal24, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal25, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal26, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal27, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal28, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal29, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal30, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal31, dummy);
                                SetDlgItemText(hDlg,IDC_UKanal32, dummy);

                                return TRUE;
                        }
```

```
		case WM_COMMAND:
			if (LOWORD(wParam) == IDCANCEL)
			{
				EndDialog(hDlg, LOWORD(wParam));
				return TRUE;
			}
			else if(LOWORD(wParam) == IDUPDATE)
			{
				ledmessen(1,defmessparam.ba);
				ads1241klarmachen(defmessparam.ba);
				startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts

				for(i=0;i<8;i++)
				{

	ads1241getu30(defmessparam.ba,1,i+1,&dummy1[i],&dummy2[i],&dummy3[i],&dummy4[i]);	//
Auslesen der Kanäle
				}
				ledmessen(0,defmessparam.ba);

				u[1]=dummy1[0];
				u[2]=dummy4[0];
				u[3]=dummy3[0];
				u[4]=dummy2[0];

				u[5]=dummy1[1];
				u[6]=dummy4[1];
				u[7]=dummy3[1];
				u[8]=dummy2[1];

				u[9]=dummy1[2];
				u[10]=dummy4[2];
				u[11]=dummy3[2];
				u[12]=dummy2[2];

				u[13]=dummy1[3];
				u[14]=dummy4[3];
				u[15]=dummy3[3];
				u[16]=dummy2[3];

				u[17]=dummy1[4];
				u[18]=dummy4[4];
				u[19]=dummy3[4];
				u[20]=dummy2[4];

				u[21]=dummy1[5];
				u[22]=dummy4[5];
				u[23]=dummy3[5];
				u[24]=dummy2[5];

				u[25]=dummy1[6];
				u[26]=dummy4[6];
				u[27]=dummy3[6];
				u[28]=dummy2[6];

				u[29]=dummy1[7];
				u[30]=dummy4[7];
				u[31]=dummy3[7];
				u[32]=dummy2[7];

				for(i=1;i<=32;i++)
				{
```

```
            u[i]=maketemp(u[i],i);
}

sprintf(dummy,"%.3f °C", u[1]);
SetDlgItemText(hDlg,IDC_UKanal1, dummy);
sprintf(dummy,"%.3f°C", u[2]);
SetDlgItemText(hDlg,IDC_UKanal2, dummy);
sprintf(dummy,"%.3f °C", u[3]);
SetDlgItemText(hDlg,IDC_UKanal3, dummy);
sprintf(dummy,"%.3f °C", u[4]);
SetDlgItemText(hDlg,IDC_UKanal4, dummy);
sprintf(dummy,"%.3f °C", u[5]);
SetDlgItemText(hDlg,IDC_UKanal5, dummy);
sprintf(dummy,"%.3f °C", u[6]);
SetDlgItemText(hDlg,IDC_UKanal6, dummy);
sprintf(dummy,"%.3f °C", u[7]);
SetDlgItemText(hDlg,IDC_UKanal7, dummy);
sprintf(dummy,"%.3f °C", u[8]);
SetDlgItemText(hDlg,IDC_UKanal8, dummy);
sprintf(dummy,"%.3f °C", u[9]);
SetDlgItemText(hDlg,IDC_UKanal9, dummy);
sprintf(dummy,"%.3f °C", u[10]);
SetDlgItemText(hDlg,IDC_UKanal10, dummy);
sprintf(dummy,"%.3f °C", u[11]);
SetDlgItemText(hDlg,IDC_UKanal11, dummy);
sprintf(dummy,"%.3f °C", u[12]);
SetDlgItemText(hDlg,IDC_UKanal12, dummy);
sprintf(dummy,"%.3f °C", u[13]);
SetDlgItemText(hDlg,IDC_UKanal13, dummy);
sprintf(dummy,"%.3f °C", u[14]);
SetDlgItemText(hDlg,IDC_UKanal14, dummy);
sprintf(dummy,"%.3f °C", u[15]);
SetDlgItemText(hDlg,IDC_UKanal15, dummy);
sprintf(dummy,"%.3f °C", u[16]);
SetDlgItemText(hDlg,IDC_UKanal16, dummy);
sprintf(dummy,"%.3f °C", u[17]);
SetDlgItemText(hDlg,IDC_UKanal17, dummy);
sprintf(dummy,"%.3f °C", u[18]);
SetDlgItemText(hDlg,IDC_UKanal18, dummy);
sprintf(dummy,"%.3f °C", u[19]);
SetDlgItemText(hDlg,IDC_UKanal19, dummy);
sprintf(dummy,"%.3f °C", u[20]);
SetDlgItemText(hDlg,IDC_UKanal20, dummy);
sprintf(dummy,"%.3f °C", u[21]);
SetDlgItemText(hDlg,IDC_UKanal21, dummy);
sprintf(dummy,"%.3f °C", u[22]);
SetDlgItemText(hDlg,IDC_UKanal22, dummy);
sprintf(dummy,"%.3f °C", u[23]);
SetDlgItemText(hDlg,IDC_UKanal23, dummy);
sprintf(dummy,"%.3f °C", u[24]);
SetDlgItemText(hDlg,IDC_UKanal24, dummy);
sprintf(dummy,"%.3f °C", u[25]);
SetDlgItemText(hDlg,IDC_UKanal25, dummy);
sprintf(dummy,"%.3f °C", u[26]);
SetDlgItemText(hDlg,IDC_UKanal26, dummy);
sprintf(dummy,"%.3f °C", u[27]);
SetDlgItemText(hDlg,IDC_UKanal27, dummy);
sprintf(dummy,"%.3f °C", u[28]);
SetDlgItemText(hDlg,IDC_UKanal28, dummy);
sprintf(dummy,"%.3f °C", u[29]);
SetDlgItemText(hDlg,IDC_UKanal29, dummy);
sprintf(dummy,"%.3f °C", u[30]);
SetDlgItemText(hDlg,IDC_UKanal30, dummy);
```

```
                                    sprintf(dummy,"%.3f °C", u[31]);
                                    SetDlgItemText(hDlg,IDC_UKanal31, dummy);
                                    sprintf(dummy,"%.3f °C", u[32]);
                                    SetDlgItemText(hDlg,IDC_UKanal32, dummy);

                                }
                                break;
        }
   return FALSE;
}
```

11.7.2.19 StdAfx.cpp

```
// stdafx.cpp : Quelltextdatei, die nur die Standard-Includes einbindet
//        8kanal.pch ist die vorkompilierte Header-Datei
//        stdafx.obj enthält die vorkompilierte Typinformation

#include "stdafx.h"
```

11.7.2.20 Tanzeige.cpp

```
// Funktionen für die Temperaturanzeige
#include "StdAfx.h"
/*
extern HINSTANCE hInst;
extern MESSSTEUERSTRUCT defmessparam;

extern DATENSATZ * messdaten;
extern HGLOBAL hMemHandle1;
extern REIHENSTEUERSTRUCT messkontrolle;

double bereichsumschalter[8]={0.164,0.104,0.0633,0.0376,0.0242,0.0149,0.0113}; // Spannungen für die Kanalumschaltung

void kanal8Ttschreiber(HWND hWnd) //8 Kanal T/t schreiber
{
int maxwinx,maxwiny,startx,starty,j=0,k,pga,iErg,readcount=0;
long int i=0;
double ywertmax,ywertmin,u=0,messdauer=0;
HDC hDc;
RECT Rect;
char outtext[1000];
HPEN hStift[8];
OPENFILENAME ofn;
static char  szFilename[810] = ".kal";
HFILE hFile;
char readdata[200],read1;


        // Kalibrierdaten einlesen

   memset(&ofn, 0, sizeof(OPENFILENAME));
   ofn.lpstrFile = szFilename;
   iErg = FileDialogKal(hWnd, &ofn);

        if(iErg == IDOK)
   {
      //Datei öffnen mit Fehlertest
      if((hFile = _lopen(ofn.lpstrFile, OF_READ)) == -1)
      {
         MessageBox(hWnd, "Das war wohl nichts", "Achtung", MB_OK|MB_ICONSTOP);
         SendMessage(hWnd,WM_CLOSE,NULL,NULL);
```

```
        return;
    }

    //an den Dateianfang gehen
    _llseek(hFile, 0L, 0);

    // Datei lesen, Fehlerprüfung hinzufügen

                for(k=0;k<8;k++)
                {
                        readcount=0;
                        do //eine Zeile einlesen
                        {
                                _hread(hFile,&read1,1);
                                readdata[readcount++]=read1;
                        }
                        while(readdata[readcount-1]!=13 ); //Zeilenende

                        readdata[readcount-1]=0; //String beenden
                        _hread(hFile,&read1,1); // noch Zeilenvorschub einlesen

                        sscanf(readdata,"%le %le %le %le %le
",&messkontrolle.kalparam[k].a,&messkontrolle.kalparam[k].b,&messkontrolle.kalparam[k].c,&messkontrolle.
kalparam[k].uref,&messkontrolle.kalparam[k].r1);
                }

                _lclose(hFile);

        }
        else
                return;

        hDc=GetDC(hWnd);// Fenster Parameter
        GetClientRect(hWnd, &Rect);
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;
        starty=(int) (maxwiny-UAB-OAB)+OAB;
        startx=(int) LAB;

        messkontrolle.flagkanal1=TRUE; // Messkanal setzen
        messkontrolle.flagkanal2=TRUE;
        messkontrolle.flagkanal3=TRUE;
        messkontrolle.flagkanal4=TRUE;
        messkontrolle.flagkanal5=TRUE;
        messkontrolle.flagkanal6=TRUE;
        messkontrolle.flagkanal7=TRUE;
        messkontrolle.flagkanal8=TRUE;

        GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden

        ads1241klarmachen(defmessparam.ba);
        startclock(defmessparam.ba); // startet und resetet die Uhr des Geräts
        ads1241getu(defmessparam.ba,defmessparam.verstearker,defmessparam.kanal);
        messdauer=gettime(defmessparam.ba); // ersten Wert speichern

        do
        {
                do
                {
                }
                while(messdauer==gettime(defmessparam.ba)); //auf nächsten tick warten;
                messdauer=gettime(defmessparam.ba);
```

```
                for(j=1;j<=8;j++)
                {
                        u=ads1241getu(defmessparam.ba,1,j); // Spannung einlesen, um Bereich festzustellen

                        if(u<=bereichsumschalter[6])
                                pga=128;
                        else if(u<=bereichsumschalter[5])
                                        pga=64;
                        else if(u<=bereichsumschalter[4])
                                        pga=32;
                        else if(u<=bereichsumschalter[3])
                                        pga=16;
                        else if(u<=bereichsumschalter[2])
                                        pga=8;
                        else if(u<=bereichsumschalter[1])
                                        pga=4;
                        else if(u<=bereichsumschalter[0])
                                        pga=2;
                        else
                                pga=1;

                        u=ads1241getu(defmessparam.ba,pga,j); // Spannung einlesen

                        messdaten[i].t=messdauer-1;
                        messdaten[i].ywerte[j-1]=maketemp(u,j); //berechnet die Temperatur aus der Spannung

                        SelectObject(hDc,hStift[j]);
                        if(i>0)
                        {
                                MoveToEx(hDc,(int)(startx+(messdaten[i-1].t-messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round( (starty-(messdaten[i-1].ywerte[j-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))),NULL);
                                LineTo(hDc,(int)(startx+(messdaten[i].t-messdaten[0].t)/(messkontrolle.messdauer)*(maxwinx-RAB-LAB)),(int) round ((starty-(messdaten[i].ywerte[j-1]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UAB-OAB))));
                        }

                }
                sprintf(outtext,"%.0lf s,%i: %.2lf °C, %i: %.2lf °C, %i: %.2lf °C, %i: %.2lf °C, %i: %.2lf °C, %i: %.2lf °C, %i: %.2lf °C", messdauer-1, 1,messdaten[i].ywerte[0],2, messdaten[i].ywerte[1],3, messdaten[i].ywerte[2],4, messdaten[i].ywerte[3],5, messdaten[i].ywerte[4],6, messdaten[i].ywerte[5],7, messdaten[i].ywerte[6],8, messdaten[i].ywerte[7]);
            TextOut(hDc,LAB+10,(OAB-TEXTYDY)/2,outtext,strlen(outtext));

                i++;
                if(GetAsyncKeyState('E') != 0)
                {
                        messkontrolle.messdauer=messdauer;
                        break; // Messung abbrechen
                }
        }
        while(messdauer<=messkontrolle.messdauer);

        messkontrolle.anzahl=i;
        clockoff(defmessparam.ba);
        GetAsyncKeyState('E'); // Tastatatur einlesen, damit vorherige Tastendrücke verworfen werden, um das Speichern nicht zu stören

        save8Tt(hWnd); // Daten speichern

        for(j=0;j<8;j++) // Stifte wieder aufräumen
```

```
                DeleteObject(hStift[j]);
        ReleaseDC(hWnd,hDc);

}


*/
```

11.7.2.21 thermostat.cpp

```
////**************************************************************************
// Funktionen für die Kommunikation mit dem Laudathermostaten
//**************************************************************************

#include "StdAfx.h";

extern REIHENSTEUERSTRUCT messkontrolle;

/*double getasltemp(void)
{
char datain[1200],data[9],dummy[2];
unsigned long countout=2,countin=11;
double temp=0;
int i=0;


        PurgeComm(messkontrolle.,PURGE_TXCLEAR&&PURGE_RXCLEAR); // Puffer leeren
        WriteFile(messkontrolle.hComasl,"T\n",countout,&countout,NULL);
        i=0;
        do
        {
                countin=1;
                ReadFile(messkontrolle.hComasl,dummy,countin,&countin,NULL);
                datain[i++]=dummy[0];
                if(i>12)
                        break;
        }
        while(dummy[0]!='\n');

        strncpy(data,datain+sizeof(char),7);
        data[7]=0;
        if((data[1]==' ') && (data[0]=='-'))
        {
                data[1]='-';
                temp=atof(data+sizeof(char));
        }
        else
                temp=atof(data);

return(temp+273.15); // Temperatur wird in K zurückgegeben
}*/

void setkryotemp(double t) // setzt Solltemperatur beim kryo in °C
{
char dataout[100];
unsigned long stringlaenge;

        sprintf(dataout,"OUT_%05.2lf",t);
        stringlaenge=strlen(dataout);
        WriteFile(messkontrolle.hkryo,dataout,stringlaenge,&stringlaenge,NULL);
}
```

11.7.2.22 2kanal.h

```
void kanal2Utschreiber(HWND hWnd); //2 Kanal T/t schreiber als Test für 30 Kanal;
int save2Ut(HWND hWnd);
```

11.7.2.23 30kanal.h

```
#if !defined(AFX_8KANAL_H__681F6363_6FA9_4D7E_963A_1703314DEDF7__INCLUDED_)
#define AFX_8KANAL_H__681F6363_6FA9_4D7E_963A_1703314DEDF7__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "resource.h"


#endif // !defined(AFX_8KANAL_H__681F6363_6FA9_4D7E_963A_1703314DEDF7__INCLUDED_)

// Funktionsprototypen

LONG MenuJob(HWND hWnd, WPARAM wParam, LPARAM lParam);
void kanal1utschreiber(HWND hWnd);


// Konstanten

#define OK 1
#define FEHLER 0
```

11.7.2.24 ads1241e.h

```
void ads1241pga(USHORT ba, int pga); //setzt pgaverstärker
void ads1241kanalselekt(USHORT ba, int kanal);
void ads1241selfoffsetgaincal(USHORT ba); //self offset und gain calibration
void ads1241systemoffsetcal(USHORT ba); //system offset calibration
void ads1241selfgaincal(USHORT ba); //self gain calibration
void ads1241selfoffsetcal(USHORT ba); //self offset calibration
void ads1241selfoffsetcal(USHORT ba); //self offset und gain calibration
//overload ads1241dataread;
void ads1241dataread(USHORT ba,int *data); //liest das Datenregister DOR aus, wird in der Reihenfolge hibyte, midbyte, lobyte in das
void ads1241read(USHORT ba, int *readdata1,int *readdata2,int *readdata3,int *readdata4,int *writedata,int readanzahl,int writeanzahl); // für 30-Kanal
void ads1241systemgaincal(USHORT ba); //system gain calibration
void ads1241reset(USHORT ba); //setzt den Wandler auf Ausgangswerte zurück, Ende continuos mode
void ads1241dsync(USHORT ba); //startet Modulator des Wandlers
void ads1241write(USHORT ba, int *data,int anzahl); // schreibt Anzahl Byte aus dem Feld data in den adc lpt-port ba
void ads1241read(USHORT ba, int *readdata,int *writedata,int readanzahl,int writeanzahl);
void warten1(void);
void ads1241einstellen(USHORT ba);
void ads1241klarmachen(USHORT ba); // richtet adc für Messung her
double ads1241getu(int ba,int pga, int kanal); // liest einmal Spannung vom Wandler an ba, mit Verstärker pga, und kanal kanal  ein
void ads1241getu30(int ba,int pga, int kanal, double *u1, double *u2, double *u3, double *u4); //30 Kanal liest einmal Spannung vom Wandler an ba, mit Verstärker pga(1-128), und kanal kanal ein(1-8),
double ads1241getudebug(int ba,int pga, int kanal,int *mux); // liest einmal Spannung vom Wandler an ba, mit Verstärker pga(1-128), und kanal kanal ein(1-8),
```

11.7.2.25 asl.h

```
double getasltemp(void);
void setasl(void); // stellt ASL ein
```

11.7.2.26 aslTt.h

```
void aslkanalschreiber(HWND hWnd);
```

11.7.2.27 auswert.h

```
void kappatauswert(HWND hWnd);
int FileDialogwert(HWND hWnd, OPENFILENAME *ofn);
void bearbeitekoords(LPARAM lParam,HWND hWnd,double *x,double *y, long int *count ); // berechnet die Koordinaten im kappa / tempsystem
void bearbeitemausebutton(LPARAM lParam,HWND hWnd);
void bearbeitemauserechts(LPARAM lParam,HWND hWnd);
void bearbeitemausemitte(LPARAM lParam,HWND hWnd); // einzoomen
void bearbeitemausemitteshift(LPARAM lParam,HWND hWnd); // Zoom zurücksetzen
void linreg(long int start,long int ende,double *a, double *b, double *r);

typedef struct
{
	long int count[4]; // wo liegt der Punkt in der Liste
	double lnkappa[4];
	double dutemp[4];
	int i; // Zähler des Punktes
	double a[2];
	double b[2];
	double r[2];
	double schnittx;
}
KOORDPOS;

typedef struct
{
	long int count[2]; // wo liegt der Punkt in der Liste
	double dutemp[2];
	int i; // Zähler
	int status;
	double tempgelbstrich;
	double dutempneu[2];
	long int countneu[2];
}
ZOOMPOS;
```

11.7.2.28 comlfthermo.h

```
void setledmessen(int geraet,int status); // setzt die Messen LEDs
void setstoerung(int status); // setzt die LED Störung am Thermometer
void wandelfeld(int status); //  Wandelfeldrührer ein/aus
void zahnradruehrer(int motor); //  Zahnradrührer
void spannung(double u); //  Spannungsausgabe
void uhrstatus(int status); // Uhr ein oder aus
void reset(void);
long int uhrzeit(void); // holt Uhrzeit
int messalle(double *u, long int *zeit); // alle 32 Kanäle an beiden Geräten ausgeben
int messkanal(int kanal,double *u, long int *zeit); //gibt Messungen eines Kanals aus
void frequenz(double f); //  Frequenzausgabe
```

11.7.2.29 ds1305.h

```
void ds1305bytewrite(int ba, int adresse, int data); //schreibt ein Byte an den ds1305
```

```
void ds1305byteread(int ba, int adresse, int *data); //liest ein Byte vom ds1305
void startclock(int ba); //startet den Oszillator auf dem ds1305
void clockoff(int ba); //schaltet den Oszillator auf dem ds1305 aus
long int gettime(int ba);
void warten(double sek,double TimeUnit);

LARGE_INTEGER TimeInit(double * TimeUnit);
```

11.7.2.30 grafik.h

```
//define Anweisungen

// Paintkontrolle
#define NOPAINTMSG 2
#define PAINTMSG 1
#define NOPAINTMSGAUSWERTUNG 0

// Abstände für Kosy
#define LAB 70 //für Kosy
#define RAB 70 //Abstände, ab denen die Messwerte aufgetragen werden
#define OAB 70 //OAB=UAB
#define UAB 70
#define BALKENK 4 // muss <als Raender sein
#define TEXTYDX -60
#define TEXTYDY 8
#define TEXTXDX 0
#define TEXTXDY 8
#define ANZAHLXTICKS 10
#define ANZAHLYTICKS 10

#define LABK 44 //für Kosy für die kleinen KOSYs
#define RABK 20 //Abstände ab denen die Messwerte aufgetragen werden
#define OABK 20
#define UABK 28
#define TEXTYDXK -40

#define TEXTDIFF 100 // Textabstände für Statusfenster 80
#define LABTEXT 15

#define ASLPAINT 1111

#define MAXEINZELPUNKT 4// maximale Zahl an Einzelpunkten

#define SCHWARZ RGB(0,0,0)
#define BRAUN RGB(128,0,0)
#define ROT RGB(255,0,0)
#define ORANGE RGB(255,126,0)
#define GELB RGB(240,220,28)
#define GRUEN RGB(51,150,18)
#define BLAU RGB(0,0,255)
#define GRAU RGB(162,162,162)
#define VIOLETT RGB(137,23,153)

#define FARBE1 BRAUN
#define FARBE2 ROT
#define FARBE3 ORANGE
#define FARBE4 GELB
#define FARBE5 GRUEN
#define FARBE6 BLAU
#define FARBE7 VIOLETT
#define FARBE8 GRAU
#define                                                                    FARBENDESTEXTES
textfarben[8]={FARBE1,FARBE2,FARBE3,FARBE4,FARBE5,FARBE6,FARBE7,FARBE8}
```

```
#define GEZOOMED 1
#define UNGEZOOMED 0

/*-- Prototypen --*/
void fensterloeschen(HDC hPaint,int maxwinx,int maxwiny);
void drawkosy(HWND hWnd, int flag); // WM-Paintroutine für das Hauptfenster
void DrawBitmap( HDC hdc, HBITMAP bitmap, short x, short y );
double findmaxdouble(double *feld,unsigned long int count);
void drawkosy30kanal(HWND hWnd, int flag,int kanal); // zeichnet Kosy für die kleinen Fenster beim 30-Kanalgerät
void drawkosyhires(HWND hWnd, int flag,int kanal); // eichnet Kosy für die HIRES FENSTER
void drawstatus(HWND hWnd, int flag); // zeichnet Statusfenster

// Strukturen
typedef struct // Koordinaten der Fenster
{
        int xMess1;
        int yMess1;
        int nWidthMess1;
        int nHeightMess1;

        int xMess2;
        int yMess2;
        int nWidthMess2;
        int nHeightMess2;

        int xMess3;
        int yMess3;
        int nWidthMess3;
        int nHeightMess3;

        int xMess4;
        int yMess4;
        int nWidthMess4;
        int nHeightMess4;

        int xMessStatus;
        int yMessStatus;
        int nWidthMessStatus;
        int nHeightMessStatus;

        int xMessHiRes1;
        int yMessHiRes1;
        int nWidthMessHiRes1;
        int nHeightMessHiRes1;

        int xMessHiRes2;
        int yMessHiRes2;
        int nWidthMessHiRes2;
        int nHeightMessHiRes2;

        int xMessHiRes3;
        int yMessHiRes3;
        int nWidthMessHiRes3;
        int nHeightMessHiRes3;

        int xMessHiRes4;
        int yMessHiRes4;
        int nWidthMessHiRes4;
        int nHeightMessHiRes4;

        int xMess1lf;
```

```
        int yMess1lf;
        int nWidthMess1lf;
        int nHeightMess1lf;

        int xMess2lf;
        int yMess2lf;
        int nWidthMess2lf;
        int nHeightMess2lf;

        int xMess3lf;
        int yMess3lf;
        int nWidthMess3lf;
        int nHeightMess3lf;

        int xMess4lf;
        int yMess4lf;
        int nWidthMess4lf;
        int nHeightMess4lf;

        int xMessHiRes1lf;
        int yMessHiRes1lf;
        int nWidthMessHiRes1lf;
        int nHeightMessHiRes1lf;

        int xMessHiRes2lf;
        int yMessHiRes2lf;
        int nWidthMessHiRes2lf;
        int nHeightMessHiRes2lf;

        int xMessHiRes3lf;
        int yMessHiRes3lf;
        int nWidthMessHiRes3lf;
        int nHeightMessHiRes3lf;

        int xMessHiRes4lf;
        int yMessHiRes4lf;
        int nWidthMessHiRes4lf;
        int nHeightMessHiRes4lf;

        int xMessMain;
        int yMessMain;
        int nMessWidthMain;
        int nMessHeighMain;

        int xMessASL;
        int yMessASL;
        int nWidthMessASL;
        int nHeightMessASL;

        int xMessHiResASL;
        int yMessHiResASL;
        int nWidthMessHiResASL;
        int nHeightMessHiResASL;
}
FENSTERKOORDSSTRUCT;
```

11.7.2.31 Hardware.h

```
// Definitionen, die direkt die Hardware betreffen

//Geräte Parameter
#define UREF 2.5 // Spannungsreferenz der Temperaturmessung
```

```
#define UREFMAX542 2.5 // Spannungsreferenz des DAC
#define R1DAC 10000  // Spannungsteiler Verstäker DAC
#define R2DAC 5600 // Spannungsteiler Verstäker DAC
#define THERMA 1.068981e-3 //Standardkalibrierdaten für Thermistoren
#define THERMB 2.120700e-4
#define THERMC 9.019537e-8
#define R1 100e3

#define USLF 0.0609//0.061 // Speisespannung der Leitfähigkeitsmessung
#define RMLF 5e2 // Messwiderstand in der Leitfähigkeitsmessung

#define MOTORKORREKTUR 6.2 // maximaler Ausgangspegel an DAC

#define MAXTEMPERATUR 40 // maximale Messtemperatur
#define MINTEMPERATUR -60 // minimale Messtemperatur
#define MINLF 1e-6 //mS
#define MAXLF 30 //mS

#define WANDELFELD 100
#define ZAHNRAD 120

// logigpegel
#define HI 1
#define LO 0
```

11.7.2.32 kanal1ut.h

```
int startdlgfkanal1ut(HWND hWnd);/*Gibt TRUE zurück, wenn es weitergehen soll die Messung fortgesetzt werden soll, wenn nicht dann False. Stellt den ersten Dialog für die Eingabe der Messparameter dar*/
BOOL FAR PASCAL StartDialogkanal1ut(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam);
int FileDialogSave1ut(HWND hWnd, OPENFILENAME *ofn);
int save1ut(HWND hWnd);
```

11.7.2.33 kanal8Tt.h

```
// Prototypen
void kanal8Ttschreiber(HWND hWnd); //8 Kanal T/t schreiber
BOOL FAR PASCAL StartDialogkanal8Tt(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam); // Dialog für die Eingabe des Messparameter
int save8Tt(HWND hWnd); // speichert Messdaten
int FileDialogKal(HWND hWnd, OPENFILENAME *ofn); // liest kaldata
void saveuntermessung(HWND hWnd, long int i); // speichert Messdaten während der Messung
```

11.7.2.34 kanal8Ut.h

```
void kanal8utschreiber(HWND hWnd); //8 Kanal U/t schreiber
```

11.7.2.35 lf597s.h

```
double getwtwlf597s(void); // Werte vom Leitfähigkeitsmessgerät einlesen
```

11.7.2.36 max542.h

```
void u_ausgabe(double u,int ba);
void cs(unsigned char wert,int ba,int dac);
void lpt_ausgeben(unsigned char byte,int ba);
void max542(double u,int ba);
void max542(double u,int ba,int dac); // gibt eine Spannung aus
void motor(double u,int ba); // steuert den Motor
void frequenzausgabe(double f, int ba); // stellt die Frequenz der Leitfähigkeitsmessung ein, beide DACs laufen synchron
```

11.7.2.37 mess30kanal.h

```
LRESULT CALLBACK WndProcMess1(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam); // Fensterdefinitionen
BOOL InitInstanceMess1( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMess1( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMess2(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMess2( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMess2( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMess3(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMess3( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMess3( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMess4(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMess4( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMess4( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMessStatus(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMessStatus( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMessStatus( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMessHiRes1(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMessHiRes1( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMessHiRes1( HINSTANCE hInstance );

LRESULT CALLBACK WndProcHiRes2(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMessHiRes2( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassHiRes2( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMessHiRes3(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMessHiRes3( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMessHiRes3( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMessHiRes4(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMessHiRes4( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMessHiRes4( HINSTANCE hInstance );

LRESULT CALLBACK WndProcASL(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceASL( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMessASL( HINSTANCE hInstance );

ATOM MyRegisterClassMessASLHiRes( HINSTANCE hInstance );
BOOL InitInstanceMessHiResASL( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
LRESULT CALLBACK WndProcMessHiResASL(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void kanal30TUschreiber(HWND hWnd); // startet die Kalibrierung des 30-Kanalgeräts
BOOL FAR PASCAL StartDialogkanal30Ut(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam); // Dialog Funktion für das Starten der Messung  30-Kanal Kalibrierung
void kanal30TUschreiberaus(HWND hWnd); // beendet die 30-Kanal Messung
void kanalmess30(HWND hWnd); // führt die einzelnen Messchritte durch
int save30ut(HWND hWnd);
```

```
void CALLBACK timerroutine(UINT uID,UINT uMsg,DWORD dwUser,DWORD dw1,DWORD dw2);
LRESULT CALLBACK Kanal32mittelwert( HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam );
void saveuntermessung30(HWND hWnd, long int i); // speichert Messdaten während der Messung
```

11.7.2.38 mess.h

```
// Headerdatei mit Definitionen ,die unmittelbaren Bezug zur Messung haben

#define NIXMESS 0 // Definition der Messtypen
#define TEST1KANAL 1
#define KANAL8TT 2
#define KANAL8UT 3
#define ASLTt 4
#define KANAL2UT 5
#define KANAL30TU 6
#define KANAL30TT 7
#define WTWGT 8
#define KANAL30TTLF 9
#define KAPPATAUSWERT 10
#define TTAUSWERT 11
#define KAPPATTAUSWERT 12
#define KANAL30TTLFKAL 13

#define KANAL30 30 // Definition der Gerätetypen, hier 30 Kanal-Gerät
#define KANAL8  8  // 8 Kanal Gerät
#define KANAL30TLF 64
#define NIXDRAN -1 // kein Gerät angeschlossen

#define STANDARTKURVE 1 // Kalibrationstypen
#define INIFILENAME "30kanal.ini"
#define MAXMESSDAUER 360000 //maximale Messdauer 360000
#define HIRESMESSDAUER 200
#define WARTEZEIT 0.2 // Zeit für den Timer in s, der die einzelnen Messungen koordiniert
#define ZEILENLENG 2000

#define WM_TIMERHGS WM_USER+1 // Message für eigenen Timer

#define THERMO 1
#define LEITFAEHIGKEIT 2

typedef struct // enthält alle wichtigen Informationen für die Messung; dient zur Einstellung jeder einzelnen
Messung
{
        int verstearker; // Vestärkungsfaktor des adc
        char kanal; // Messkanal
        int ba; //Basisadresse des LPT-Ports Thermometer
        int balf;//Basisadresse des LPT-Ports Leitfähigkeit
        char tempfile[MAX_PATH]; //Datei für temporärere?? Daten
}
MESSSTEUERSTRUCT;

typedef struct // in diesem Format werden die Ergebnisse gespeichert, 8 kanal
{
        double t;
        double ywerte[8];
}
DATENSATZ;

typedef struct // in diesem Format werden die Ergebnisse gespeichert, 30kanal
{
        double t[64];
        double ywerte[64];
```

```
	double asl; // asl-Temperatur
	double aslzeit;
}
DATENSATZ30;

typedef struct // speichert Kalibrierdaten
{
	double a;
	double b;
	double c;
	double r1;
	double uref;
}
KALSTRUCT;

typedef struct //Kontrolle der Messreihe
{
	int messtyp; // beschreibt die Art der Messung
	int messan; // wird gerade gemessen
	long int anzahl; //maximale Zahl der Messpunkte
	int flagkanal1; //welche Kanäle sind belegt für 8-Kanal
	int flagkanal2;
	int flagkanal3;
	int flagkanal4;
	int flagkanal5;
	int flagkanal6;
	int flagkanal7;
	int flagkanal8;
	int flagkanal30[33]; //welche Kanäle sind belegt für 30-Kanal
	double letztewerte[64]; // speichert die aktuellsten Werte in der y-Achse
	double letztewertex[64]; // speichert die aktuellsten Werte in der x-Achse
	double letztewerteasl;
	double letztewerteaslx;
	int adc[5]; // welche ADCs werden verwendet
	int kanal[9];// Kanäle ADCs werden verwendet
	long int count; //Zahl der Messpunkte
	double messdauer; //speichert die maximale Messdauer
	long int messzeit; //Messzeit
	double maximalerywert; // in V oder °C je nach Messtyp
	double minmalerywert;
	double maximalerywertlf; // in V oder °C je nach Messtyp
	double minmalerywertlf;
	KALSTRUCT kalparam[32];
	double maxbathtemp; //maximale Temperatur des Ölbads
	int typ; // Art des Messgeräts, 8 oder 30 Kanal
	int asl; // mit oder ohne ASL-Thermometer
	int aslcom; //Comport, an den das ASL angeschlossen worden ist
	HANDLE hComasl; // Handle für Comport als
	int wtw; // mit oder ohne WTW-Leitfähigkeitsmessung
	int wtwcom; //Comport, an den das ASL angeschlossen worden ist
	HANDLE hComwtw; // Handle für Comport wtw
	int motor; // Rührgeschwindigkeit des Motors
	int motoran; // Motor an oder aus
	int ruehrtyp; // welcher Rührer ist angeschlossen
	int thermoan; // Thermostat an oder aus
	double frequenz; // Frequenz für Leitfähigkeit
	int thermolfcom; // Comport, an den das 32-Kanal Leitfähigkeitsmessgerät angeschlossen worden
ist
	int thermolf; // mit oder ohne 32-Thermometer und 32-Leitfähigkeit
	int kryo; // Kryostat
	int kryocom; //Comport Kryostat
	HANDLE hkryo; // handle für commport kryo
	HANDLE hComthermolf;
```

```
}
REIHENSTEUERSTRUCT;
```

11.7.2.39 mess30thermolf.h

```
LRESULT CALLBACK WndProcMess1lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam); // Fensterdefinitionen
BOOL InitInstanceMess1lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMess1lf( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMess2lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMess2lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMess2lf( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMess3lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMess3lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMess3lf( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMess4lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMess4lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMess4lf( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMessHiRes1lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMessHiRes1lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMessHiRes1lf( HINSTANCE hInstance );

LRESULT CALLBACK WndProcHiRes2lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceHiRes2lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassHiRes2lf( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMessHiRes3lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMessHiRes3lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMessHiRes3lf( HINSTANCE hInstance );

LRESULT CALLBACK WndProcMessHiRes4lf(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitInstanceMessHiRes4lf( HINSTANCE hInstance, int nCmdShow, HWND hwndParent);
ATOM MyRegisterClassMessHiRes4lf( HINSTANCE hInstance );

void kanal30TundLF(HWND hWnd); // startet die Messung mit dem 30-Kanal Thermometer und Leitfähigkeitsmessgerät
void kanalmess30lf(HWND hWnd); // führt die einzelnen Messschritte durch
void saveuntermessung30lf(HWND hWnd, long int i); // speichert Messdaten während der Messung
int save30utlf(HWND hWnd);
LRESULT CALLBACK startoptionenttlf( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam );
```

11.7.2.40 messwtw.h

```
void wtwkanalschreiber(HWND hWnd); //1 Kanal G/t schreiber mit wtw
```

11.7.2.41 resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
```

```
// Used by 30kanal.rc
//
#define IDC_MYICON                  2
#define IDD_MY8KANAL_DIALOG         102
#define IDD_ABOUTBOX                103
#define IDS_APP_TITLE               103
#define IDM_ABOUT                   104
#define IDM_EXIT                    105
#define IDS_HELLO                   106
#define IDI_MY8KANAL                107
#define IDI_SMALL                   108
#define IDC_MY8KANAL                109
#define IDR_MAINFRAME               128
#define IDD_1Kanal                  130
#define IDB_UPFEIL                  137
#define IDB_LPFEIL                  138
#define IDB_DFEIL                   139
#define IDB_RPFEIL                  140
#define IDD_8KanalTt                142
#define IDD_SPANNUNGSAUSGABE        143
#define IDI_ICONKLEIN               146
#define IDD_32KanalUDisplay         147
#define IDD_OPTIONEN                148
#define IDC_MESS1                   150
#define IDC_MESS2                   151
#define IDC_MESS3                   152
#define IDC_MESS4                   153
#define IDC_MESSSTATUS              154
#define IDC_MESSHIRES               155
#define IDC_MESSHIRES1              156
#define IDC_MESSHIRES2              157
#define IDD_30KANALUT               158
#define IDC_MESSHIRES3              158
#define IDC_MESSHIRES4              159
#define IDD_30KANALCHECK            160
#define IDC_MESSASL                 160
#define IDC_MESSHIRESASL            161
#define IDC_MESS1LF                 162
#define IDC_MESS2LF                 163
#define IDC_MESS3LF                 164
#define IDD_TEST                    165
#define IDC_MESS4LF                 165
#define IDD_MITTEL                  166
#define IDC_MESSHIRES1LF            168
#define IDC_MESSHIRES2LF            169
#define IDD_StartMessungTTLF        169
#define IDC_MESSHIRES3LF            170
#define IDD_MESSKONTROLLE           170
#define IDC_MESSHIRES4LF            171
#define IDC_KANAL1                  1000
#define IDC_SPANNUNG                1000
#define IDC_KANAL2                  1001
#define IDC_KANAL3                  1002
#define IDC_TAKE                    1002
#define IDC_KANAL4                  1003
#define IDC_KANAL5                  1004
#define IDC_KANAL6                  1005
#define IDC_KANAL7                  1006
#define IDC_KANAL8                  1007
#define IDC_MESSDAUER               1008
#define IDC_AMP1                    1009
#define IDC_ENDTEMP                 1009
#define IDC_AMP2                    1010
```

```
#define IDC_Geschindigkeit 1010
#define IDC_AMP4 1011
#define IDC_AMP8 1012
#define IDC_AMP16 1013
#define IDC_AMP32 1014
#define IDC_AMP64 1015
#define IDC_UKanal1 1015
#define IDC_AMP128 1016
#define IDC_UKanal2 1016
#define IDC_UKanal3 1017
#define IDC_UKanal4 1018
#define IDC_UKanal5 1019
#define IDC_UKanal6 1020
#define IDC_UKanal7 1021
#define IDC_UKanal8 1022
#define IDC_UKanal9 1023
#define IDC_UKanal10 1024
#define IDC_UKanal11 1025
#define IDC_UKanal12 1026
#define IDC_UKanal13 1027
#define IDC_UKanal14 1028
#define IDC_UKanal15 1029
#define IDC_UKanal16 1030
#define IDC_UKanal17 1031
#define IDC_UKanal18 1032
#define IDC_UKanal19 1033
#define IDC_UKanal20 1034
#define IDC_UKanal21 1035
#define IDC_UKanal22 1036
#define IDC_UKanal23 1037
#define IDC_UKanal24 1038
#define IDC_UKanal25 1039
#define IDC_UKanal26 1040
#define IDC_UKanal27 1041
#define IDC_UKanal28 1042
#define IDC_UKanal29 1043
#define IDC_UKanal30 1044
#define IDC_UKanal31 1045
#define IDC_UKanal32 1046
#define IDUPDATE 1047
#define IDC_ASL 1047
#define IDC_LPTPORT 1049
#define IDC_CHECKASL 1050
#define IDC_CHECKWTW 1051
#define IDC_CHECKTHERMOLF 1052
#define IDC_CHECKKRYO 1053
#define IDC_PFAD 1061
#define IDC_MAXTEMP 1066
#define IDC_FREQUENZ 1067
#define IDC_COMASL 1068
#define IDC_STARTTEMP 1068
#define IDC_COMWTW 1069
#define IDC_COMLFTHERMO 1070
#define IDC_RUEHR 1071
#define IDC_COMKRYO 1072
#define IDC_RUERGESCHWINDIGKEITZAHNRAD 1073
#define IDC_WANDELFELDRUEHRER 1075
#define Geschindigkeit 1076
#define IDC_8Kanalbutton 1080
#define IDC_32KANALBUTTON 1081
#define IDC_32TLFBUTTON 1082
#define ID_MESSUNG_KALIBRIERUNG 32771
#define ID_MESSUNG_ABKHLKURVE 32772
```

```
#define ID_MESSUNG_TEMPERATURANZEIGE    32773
#define IDM_TESTFUNKTIONEN_1KANALUTSCHREIBER 32774
#define ID_EXTRAS_OPTIONEN              32775
#define IDM_EXTRAS_OPTIONEN             32775
#define ID_EXTRAS_OPTIONENSPEICHERN     32776
#define IDM_EXTRAS_OPTIONENSPEICHERN    32776
#define IDM_TESTFUNKTIONEN_8KANALUTSCHREIBER 32778
#define IDM_TESTFUNKTIONEN_SPANNUNGSAUSGABE 32779
#define ID_MESSUNG_ASL                  32780
#define IDM_TESTFUNKTIONEN_2KANALUT     32781
#define IDM_TESTFUNKTIONEN32KANALSPANNUNG 32782
#define ID_FENSTER_MESS1                32783
#define ID_FENSTER_MESS2                32784
#define ID_FENSTER_MESS3                32785
#define ID_FENSTER_MESS4                32786
#define ID_FENSTER_STATUS               32787
#define ID_FENSTER_HOCHAUFGELST1        32788
#define ID_MESSUNGEN30KANAL_KALIBRIERUNG 32789
#define ID_MESSUNGEN30KANAL_TEMPERATURMESSUNG30KANAL 32790
#define ID_MESSUNGEN30KANAL_TEMPERATURANZEIGE30KANAL 32791
#define ID_FENSTER_HOCHAUFGELST2        32792
#define ID_FENSTER_HOCHAUFGELST3        32793
#define ID_FENSTER_HOCHAUFGELST4        32794
#define ID_TESTFUNKTIONEN_WTWGT         32795
#define ID_KALIB_DATENSATZ              32796
#define ID_EXTRAS_MOTOR                 32797
#define ID_EXTRAS_THERMOSTAT            32798
#define ID_MESSUNGEN30KANAL_LEITFAEHIGKEITSMESSUNG30KANAL 32799
#define ID_MESSUNGEN30KANAL_LEITFAEHIGKEITANZEIGE30KANAL 32800
#define     ID_MESSUNGEN30KANAL_TEMPERATURUNDLEITFHIGKEITSMESSUNG30KANAL
32801
#define     ID_MESSUNGEN30KANAL_TEMPERATURUNDLEITFHIGKEITSANZEIGE30KANAL
32802
#define ID_EXTRAS_THERMOSTATCOMM        32803
#define ID_EXTRAS_MOTORCOMM             32804
#define ID_STEUERUNG_KRYOSTAT           32805
#define ID_AUSWERTUNG_KAPPT             32806
#define ID_AUSWERTUNG_TT                32807
#define ID_AUSWERTUNG_KAPPTT            32808
#define ID_MESSUNGEN30KANAL_KALIBRIERUNGCOMM 32809
#define IDC_STATIC                      -1

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        171
#define _APS_NEXT_COMMAND_VALUE         32810
#define _APS_NEXT_CONTROL_VALUE         1083
#define _APS_NEXT_SYMED_VALUE           110
#endif
#endif
```

11.7.2.42 small.h

```
// Funktionsprototypen für small.cpp

void globalinit(HWND hWnd); // stellt grundlegende Parameter ein, wird später durch einlesen aus ini-datei
ersetzt
void globalexit(void); //aufräumen
LRESULT CALLBACK About( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam );
double round(double x); // rundet eine double
double maketemp(double u,int kanal); //berechnet die Temperatur aus der Spannung
```

```
LRESULT CALLBACK spannungsausgabe( HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam );
void ledmessen(int led,int ba); // LED "Messung" ein- bzw. ausschalten
void stoerung(int cond,int ba); //Relais Störung und LED ein- bzw. ausschalten
LRESULT CALLBACK Kanal32spannungsanzeige( HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam );
LRESULT CALLBACK optionen( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam );
// Optionsdialog
void writeini(void); // speichert Optionen
void setmenue(HWND hWnd);// schaltet passende Menüs aus, wenn Messung läuft
double findmaxdouble(double *feld,unsigned long int count); // findet grösste Fließkommazahl in einem
Feld
double maketempstandard(double u); //berechnet die Temperatur aus der Spannung für Standardthermistor
int kalibdatensatz(HWND hWnd); // liest einzelne Datenfiles ein, und erstellt den Kalibrierdatensatz
int FileDialogOpen(HWND hWnd, OPENFILENAME *ofn); // Filedialog für das Öffnen von Dateien
void motormenue(HMENU hMenu); // Menüfunktion zum ein- und ausschalten des Motors
void thermomenue(HMENU hMenu);  // Menüfunktion zum ein- und ausschalten des Thermostats
void motormenuecomm(HMENU hMenu); // Menüfunktion  Motor ein und aus
void thermomenuecomm(HMENU hMenu);  // Menüfunktion Thermostat ein und aus

LRESULT CALLBACK Kanal32temperaturanzeige( HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam ); // Nachrichtenbehandlungsroutine für "32-Kanal-Temperaturanzeige"-Feld.
double makelf(double u); // berechnet Leitfähigkeiten
```

11.7.2.43 StdAfx.h

```
// stdafx.h : Include-Datei für Standard-System-Include-Dateien,
//  oder projektspezifische Include-Dateien, die häufig benutzt, aber
//      in unregelmäßigen Abständen geändert werden.
//

#if !defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)
#define AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define WIN32_LEAN_AND_MEAN                 // Selten benutzte Teile der Windows-Header nicht
einbinden


// Windows-Header-Dateien:
#include <windows.h>
#include <commdlg.h>

// Header-Dateien der C-Laufzeit
#include <conio.h>
#include <malloc.h>
#include <math.h>
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <tchar.h>
#include <mmsystem.h>

// Lokale Header-Dateien
#include "small.h"
#include "mess.h"
#include "grafik.h"
#include "hardware.h"
#include "resource.h"
```

```
#include "30kanal.h"
#include "kanal1ut.h"
#include "ds1305.h"
#include "ads1241e.h"
#include "kanal1ut.h"
#include "kanal8Tt.h"
#include "kanal8Ut.h"
#include "max542.h"
#include "asl.h"
#include "aslTt.h"
#include "2kanal.h"
#include "mess30kanal.h"
#include "lf597s.h"
#include "messwtw.h"
#include "mess30thermolf.h"
#include "comlfthermo.h"
#include "auswert.h"
#include "thermostat.h"

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ fügt zusätzliche Deklarationen unmittelbar vor der vorherigen Zeile ein.

#endif                                                                                                    //
!defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)
```

11.7.2.44 thermostat.h

void setkryotemp(double t); // setzt Solltemperatur beim Kryo in °C

11.7.3 Zyklisiergerät, Mikrocontroller

11.7.3.1 zykl.c

```
/**Steuerpropgramm für Zyklisergerät*/
#include <stdlib.h>
#include <math.h>
#include <avr/pgmspace.h>
#include <string.h>
#include <stdio.h>
#include <avr/eeprom.h>
#include "hardware.h"
#include "small.h"
#include "max128.h"
#include "max542.h"

// Datentypdefinitionen
typedef unsigned char        BYTE;
typedef unsigned short WORD;

// Globale Variablen
double offset=0;
double uref=4.096;
double u[8];
int gain[8]={0,0,0,0,0,0,0,0};
long int intzeit=1;

int main(void)
{
char intext[50]={"\r\n"},outtext[200]={"\r\n"},outtextkurz[30]={""};
int i=0,j=0;
char dummy[30]={""};
```

```
int kanal=0;
double uout=0;
double rstrom=0;
double gainsave[8]={0,0,0,0,0,0,0,0};
long int uaus[8];

    // PORTs setzen
        DDRC=0xff; // port c als output
        DDRA=0xff; // port als output // A0 wird bei Bedarf auf input geschaltet
        DDRB=0xff;
        DDRD=0xff; // D als output


        USART_Init(25); //19200 Baud bei 8 MHz
        init();
        PORTD=0xff;

        USART_Transmit_string("Dateneingeben \r\n"); // an RS232 ausgeben*/

        // Kleine Ströme
        rstrom=RSTROMKLEIN;
        uausgabe(0);
        PORTB=(PORTB & 0xFB) + 0x04; //B2 auf HI, Relais Schalten

        do
        {
                do
                {
                        intext[i++]=USART_Receive();
                }
                while((i<50) && (intext[i-1]!='\r'));

                do // parsen
                {
                        i--;
                }
                while( (i>1) && (intext[i]!='h'));

                //hgof Offset einstellen
                if((intext[i+1]=='g') && (intext[i+2]=='o') && (intext[i+3]=='f')  ) // Offset stellen
                {
                        offset=setoffset();
                }

                //hgmm messen
                if((intext[i+1]=='g') && (intext[i+2]=='m') && (intext[i+3]=='m')  ) // Messen
                {
                        max128getu(u);
                        //snprintf(outtext,20,"%le %le %le %le %le %le %le %le\r\n",
u[0]/(R2ADC/(R2ADC+R1ADC)),u[1]/(R2ADC/(R2ADC+R1ADC)),u[1]/(R2ADC/(R2ADC+R1ADC))
/rstrom,u[0]/(R2ADC/(R2ADC+R1ADC))-u[1]/(R2ADC/(R2ADC+R1ADC)),u[4],u[5],u[6],u[7]);//
Korrektur für Spannungsteiler
                        outtext[0]=0;
                        dtostre(u[0]/(R2ADC/(R2ADC+R1ADC)),outtext,5,0);
                        strcat(outtext," ");

                        dtostre(u[1]/(R2ADC/(R2ADC+R1ADC)),outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[1]/(R2ADC/(R2ADC+R1ADC))/rstrom,outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");
```

```
                        dtostre(u[0]/(R2ADC/(R2ADC+R1ADC))-
u[1]/(R2ADC/(R2ADC+R1ADC)),outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[4],outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[5],outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[6],outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[7],outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," \r\n");
                        /*                              uaus[0]=u[0]/(R2ADC/(R2ADC+R1ADC))*;
                        uaus[1]=u[1]/(R2ADC/(R2ADC+R1ADC));
                        uaus[2]=u[1]/(R2ADC/(R2ADC+R1ADC))/rstrom;
                        uaus[3]=u[0]/(R2ADC/(R2ADC+R1ADC))-u[1]/(R2ADC/(R2ADC+R1ADC));
                        uaus[4]=u[4];
                        uaus[5]=u[5];
                        uaus[6]=u[6];
                        uasu[7]=u[7];*/
                        USART_Transmit_string(outtext);  // Spannung 1, Spannung 2, Strom,
Batteriespannung
                }

                //hgms jeden kanal einmal auslesen, mit faktor 1, wenn gain != 0
                if((intext[i+1]=='g') && (intext[i+2]=='m') && (intext[i+3]=='s')  ) // Messen
                {
                        for(j=0;j<7;j++) // Verstärkungsfaktoren sichern
                        {
                                gainsave[j]=gain[j];
                                if(gain[j]!=0)
                                {
                                        gain[j]=1;
                                }
                        }

                        setpga();
                        max128getufast(u);

                        //sprintf(outtext,"%le  %le  %le  %le  %le  %le  %le  %le\r\n",
u[0]/(R2ADC/(R2ADC+R1ADC)),u[1]/(R2ADC/(R2ADC+R1ADC)),u[1]/(R2ADC/(R2ADC+R1ADC))
/rstrom,u[0]/(R2ADC/(R2ADC+R1ADC))-u[1]/(R2ADC/(R2ADC+R1ADC)),u[4],u[5],u[6],u[7]);//
Korrektur für Spannungsteiler


                        outtext[0]=0;
                        dtostre(u[0]/(R2ADC/(R2ADC+R1ADC)),outtext,5,0);
                        strcat(outtext," ");

                        dtostre(u[1]/(R2ADC/(R2ADC+R1ADC)),outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[1]/(R2ADC/(R2ADC+R1ADC))/rstrom,outtextkurz,5,0);
```

```
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[0]/(R2ADC/(R2ADC+R1ADC))-
u[1]/(R2ADC/(R2ADC+R1ADC)),outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[4],outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[5],outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[6],outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," ");

                        dtostre(u[7],outtextkurz,5,0);
                        strcat(outtext,outtextkurz);
                        strcat(outtext," \r\n");

                        USART_Transmit_string(outtext); // Spannung 1, Spannung 2, Strom,
Batteriespannung

                        for(j=0;j<7;j++) // Verstärkungsfaktoren zurückschreiben
                        {
                                gain[j]=gainsave[j];
                        }
                }

                //hgiz_______ zeitkonstante eingeben
                if((intext[i+1]=='g') && (intext[i+2]=='i') && (intext[i+3]=='z')) // Integrationszeit in
40mS Schritte max  1e6
                {
                        if(intext[i+4]!='\r')
                                dummy[0]=intext[i+4];
                        else
                                dummy[0]=0;

                        if(intext[i+5]!='\r')
                                dummy[1]=intext[i+5];
                        else
                                dummy[1]=0;

                        if(intext[i+6]!='\r')
                                dummy[2]=intext[i+6];
                        else
                                dummy[2]=0;

                        if(intext[i+7]!='\r')
                                dummy[3]=intext[i+7];
                        else
                                dummy[3]=0;

                        if(intext[i+8]!='\r')
                                dummy[4]=intext[i+8];
                        else
                                dummy[4]=0;

                        if(intext[i+9]!='\r')
```

```
                    dummy[5]=intext[i+9];
                else
                    dummy[5]=0;

                if(intext[i+10]!='\r')
                    dummy[6]=intext[i+10];
                else
                    dummy[6]=0;

                dummy[7]=0;

                //USART_Transmit_string(dummy); // an RS232 ausgeben
                intzeit=atol(dummy);
            }

            //hgpa_______ pgaeinstellen
            if((intext[i+1]=='g') && (intext[i+2]=='p') && (intext[i+3]=='a'))
            {
                if(intext[i+4]!='\r')
                {
                    kanal=(int) intext[i+4]- 0x30; // Zahl bestimmen

                    if(intext[i+5]!='\r')
                        dummy[0]=intext[i+5];
                    else
                        dummy[0]=0;

                    if(intext[i+6]!='\r')
                        dummy[1]=intext[i+6];
                    else
                        dummy[1]=0;

                    if(intext[i+7]!='\r')
                        dummy[2]=intext[i+7];
                    else
                        dummy[2]=0;

                    if(intext[i+8]!='\r')
                        dummy[3]=intext[i+8];
                    else
                        dummy[3]=0;
                    dummy[4]=0;

                    gain[kanal]=atoi(dummy);
                    setpga();
                }

            }

            if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='u')) // Steuerspannung
Ausgeben Spannung in µV
            {
                if(intext[i+4]!='\r')
                    dummy[0]=intext[i+4];
                else
                    dummy[0]=0;

                if(intext[i+5]!='\r')
                    dummy[1]=intext[i+5];
                else
                    dummy[1]=0;

                if(intext[i+6]!='\r')
```

```
                    dummy[2]=intext[i+6];
                else
                    dummy[2]=0;

                if(intext[i+7]!='\r')
                    dummy[3]=intext[i+7];
                else
                    dummy[3]=0;

                if(intext[i+8]!='\r')
                    dummy[4]=intext[i+8];
                else
                    dummy[4]=0;

                if(intext[i+9]!='\r')
                    dummy[5]=intext[i+9];
                else
                    dummy[5]=0;

                if(intext[i+10]!='\r')
                    dummy[6]=intext[i+10];
                else
                    dummy[6]=0;

                if(intext[i+11]!='\r')
                    dummy[7]=intext[i+11];
                else
                    dummy[7]=0;

                if(intext[i+12]!='\r')
                    dummy[8]=intext[i+12];
                else
                    dummy[8]=0;

                if(intext[i+13]!='\r')
                    dummy[9]=intext[i+13];
                else
                    dummy[9]=0;

                dummy[10]=0;

                //USART_Transmit_string(dummy); // an RS232 ausgeben*/
                uout=atol(dummy);

                //sprintf(outtext,"%lf \r\n", zeit);
                uausgabe(((double) uout)/1e6);
            }

            if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='i')) // Konstant Strom
Ausgeben Strom in µA
            {
                if(intext[i+4]!='\r')
                    dummy[0]=intext[i+4];
                else
                    dummy[0]=0;

                if(intext[i+5]!='\r')
                    dummy[1]=intext[i+5];
                else
                    dummy[1]=0;

                if(intext[i+6]!='\r')
                    dummy[2]=intext[i+6];
```

```
                else
                        dummy[2]=0;

                if(intext[i+7]!='\r')
                        dummy[3]=intext[i+7];
                else
                        dummy[3]=0;

                if(intext[i+8]!='\r')
                        dummy[4]=intext[i+8];
                else
                        dummy[4]=0;

                if(intext[i+9]!='\r')
                        dummy[5]=intext[i+9];
                else
                        dummy[5]=0;

                if(intext[i+10]!='\r')
                        dummy[6]=intext[i+10];
                else
                        dummy[6]=0;

                if(intext[i+11]!='\r')
                        dummy[7]=intext[i+11];
                else
                        dummy[7]=0;

                dummy[7]=0;

                if(intext[i+12]!='\r')
                        dummy[8]=intext[i+12];
                else
                        dummy[8]=0;

                dummy[9]=0;

                //USART_Transmit_string(dummy); // an RS232 ausgeben*/
                uout=atol(dummy);

                //sprintf(outtext,"%lf \r\n", zeit);
                uausgabe(((double) uout)/1e6*rstrom);
        }

        //hgts testen
        if((intext[i+1]=='g') && (intext[i+2]=='t') && (intext[i+3]=='s') ) // Offset stellen
        {
                //sprintf(outtext,"%li \r\n", intzeit);
                USART_Transmit_string("test\r\n");
        }

        if((intext[i+1]=='g') && (intext[i+2]=='s') && (intext[i+3]=='g')  ) // große Ströme hgsg
        {
                rstrom=RSTROMKLEIN;
                PORTB=(PORTB & 0xFB) + 0x04; //B2 auf HI, Relais Schalten
        }
        else if((intext[i+1]=='g') && (intext[i+2]=='s') && (intext[i+3]=='k')  ) // kleine Ströme
// hgsk
        {
                rstrom=RSTROMGROSS;
                PORTB=(PORTB & 0xFB); //B2 auf LO, Relais Schalten
        }
```

```
                    if((intext[i+1]=='g') && (intext[i+2]=='z') && (intext[i+3]=='1')  ) // Zelle an hgz1
                    {
                            PORTB=(PORTB & 0xFE) + 0x01; //B0 auf HI, Relais Schalten
                    }
                    else if((intext[i+1]=='g') && (intext[i+2]=='z') && (intext[i+3]=='0')  ) // Zelle aus hgz0
                    {
                            PORTB=(PORTB & 0xFE); //B0 auf LO, Relais Schalten
                    }


                    if((intext[i+1]=='i') && (intext[i+2]=='l') && (intext[i+3]=='f')&& (intext[i+4]=='e') ) //
befehle ausgeben
                    {
                            USART_Transmit_string("hgof: Offseteinstellen\r\nhgmm: messen, gibt alle
kanaele zurueck \r\nhgiz: Intergrationzeit in Vielfachen von 40ms einstellen \r\nhgpa: PGA einstellen Kanal
und Faktor (1 bis 1000) \r\nhgtu: Spannung in µV ausgeben \r\nhgtu: Strom in µA ausgeben \r\nhgsg:
grosser Strombereich \r\nhgsk: kleiner Strombereich\r\nhgz1: Zelle an \r\nhgz0: Zelle aus \r\nhgms:
Schnell Messen \r\nhgts: Test \r\n");
                    }
                    i=0;
            }
            while(1);

}
```

11.7.3.2 small.c

```
#include "small.h"
#include "avr/pgmspace.h"
#include <math.h>

double round(double x)
{
   if((x-floor(x))<=0.5)
      return(floor(x));
   else
      return(ceil(x));
}

void warten1(void) // 1.2 µs Verzögerung
{
int i=0,j=0;
            for(i=0;i<86;i++)
            {
                    for(j=0;j<10;j++)
                    {
                    }
            }
}

void pause (int ms)  // in ms bei 8 Mhz
{
int i,j,k,l;

             for (l=0; l<ms;l++)
             {
              for (i=0; i<88; i++) //88
               for(j=0; j<255;j++) //255
                    k++;
             }

}
```

```
void USART_Init( unsigned int baud )
{

/* Set baud rate */
UBRRH = (unsigned char)(baud>>8);
UBRRL = (unsigned char)baud;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 2stop bit */
UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}

void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) )
;
/* Put data into buffer, sends the data */
UDR = data;
}

void USART_Transmit_string(unsigned char *data)
{
int i=0;

        do
        {
                USART_Transmit(data[i]);
        }
        while(data[++i]!=0);
}

unsigned char USART_Receive( void )
{
        /* Wait for data to be received */
        while ( !(UCSRA & (1<<RXC)) )
        ;

        /* Get and return received data from buffer */
        return UDR;
}
```

11.7.3.3 max128.c

```
/* Steuerprogramm für schreiber*/
#include <stdlib.h>
#include <math.h>
#include <avr/pgmspace.h>
#include <string.h>
#include <stdio.h>
#include <avr/eeprom.h>
#include "max128.h"
#include "small.h"

#define delay40ms 19

//double bremse = 0, scl_bremse = 0, adc_scl_bremse=0;
STEUERSTRUCT st;
unsigned char portbuffer=0; // hier wird der Wert gespeichert, der an der Schnittstelle anliegt, dadurch kann
man alles steuern
extern double offset;
```

```
extern int gain[7];
extern long int intzeit;

double setoffset()
{
	st.kanal = 7; // Kanal 7 auf Masse
	initstruct(&st);
	return( (double) wandeln(st));
}

void setpga(void)
{
	if(gain[0]==1)
	{
		PORTC=(PORTC & 0xEF); //A0 0
		PORTC=(PORTC & 0xDF); //A1 0
	}
	else if(gain[0]==10)
	{
		PORTC=(PORTC &0xEF) + 0x10; //A0 1
		PORTC=(PORTC &0xDF); //A1 0
	}
	else if(gain[0]==100)
	{
		PORTC=(PORTC &0xEF);  //A0 0
		PORTC=(PORTC &0xDF) + 0x20; //A1 1
	}
	else if(gain[0]==1000)
	{
		PORTC=(PORTC &0xEF) +0x10; //A0 1
		PORTC=(PORTC &0xDF) + 0x20; //A1 1
	}

	if(gain[1]==1)
	{
		PORTC=(PORTC & 0xBF); //A0 0
		PORTC=(PORTC & 0xF7); //A1 0
	}
	else if(gain[1]==10)
	{
		PORTC=(PORTC &0xBF) + 0x40; //A0 1
		PORTC=(PORTC &0xF7); //A1 0
	}
	else if(gain[1]==100)
	{
		PORTC=(PORTC &0xBF);  //A0 0
		PORTC=(PORTC &0xF7) + 0x8; //A1 1
	}
	else if(gain[1]==1000)
	{
		PORTC=(PORTC &0xBF) +0x40; //A0 1
		PORTC=(PORTC &0xF7) + 0x8; //A1 1
	}

	if(gain[2]==1)
	{
		PORTA=(PORTA & 0xBF); //A0 0
		PORTA=(PORTA & 0x7F); //A1 0
	}
	else if(gain[2]==10)
	{
		PORTA=(PORTA &0xBF) + 0x40; //A0 1
		PORTA=(PORTA &0x7F); //A1 0
```

```
        }
        else if(gain[2]==100)
        {
                PORTA=(PORTA &0xBF) ; //A0 0
                PORTA=(PORTA &0x7F) + 0x80; //A1 1
        }
        else if(gain[2]==1000)
        {
                PORTA=(PORTA &0xBF) +0x40; //A0 1
                PORTA=(PORTA &0x7F) + 0x80; //A1 1
        }

        if(gain[3]==1)
        {
                PORTA=(PORTA & 0xEF); //A0 0
                PORTA=(PORTA & 0xDF); //A1 0
        }
        else if(gain[3]==10)
        {
                PORTA=(PORTA &0xEF) + 0x10; //A0 1
                PORTA=(PORTA &0xDF); //A1 0
        }
        else if(gain[3]==100)
        {
                PORTA=(PORTA &0xEF);  //A0 0
                PORTA=(PORTA &0xDF) + 0x20; //A1 1
        }
        else if(gain[3]==1000)
        {
                PORTA=(PORTA &0xEF) +0x10; //A0 1
                PORTA=(PORTA &0xDF) + 0x20; //A1 1
        }
}

void max128getu(double *u)
{
int i=0,j=0;
long int k=0;

        st.bipolar = 1; // bipolar
   st.range = 1;

        PORTD=0xff;

        for(i=0;i<7;i++)
        {
                u[i]=0;
        }

        for(k=0;k<intzeit;k++)
        {
                for(j=0;j<delay40ms;j++) // 40mS warten
                {
                        for(i=0;i<7;i++)
                        {
                                if(gain[i]!=0)
                                {
                                        st.kanal = i; // Kanal i
                                        initstruct(&st);
                                        u[i]+= ((double) wandeln(st) - offset)/gain[i];
                                }
                                else
                                {
```

```
                                        wandeln(st) - offset;
                                        u[i]=0;
                                }
                        }
                }
        }

        for(i=0;i<7;i++)
        {
                u[i]=u[i]/(intzeit*delay40ms);
        }

                // Offset bei jeder Messung abziehen: sonst Fehler, falls Messwert nahe 0


        return;
}

/*void max128setpgaauto(void) // setzt den pga automatisch für kanal 0 und 1
{
int i=0;
double u[2];

        st.bipolar = 1; // bipolar
   st.range = 1;

        PORTD=0xff;

        for(i=0;i<2;i++)
        {
                u[i]=0;
        }

        gain[0]=1; // Mit großem Messbereich einlesen
        gain[1]=1;
        for(i=0;i<2;i++)
        {
                if(gain[i]!=0)
                {
                        st.kanal = i; // Kanal i
                        initstruct(&st);
                        u[i]+= ((double) wandeln(st) - offset)/gain[i];
                }
                else
                {
                        wandeln(st) - offset;
                        u[i]=0;
                }
        }

        for(i=0;i<2;i++)
        {
                if(abs(u[i])<0.35)
                {
                        gain[i]=10;
                }
                else if(abs(u[i])<0.035)
                {
                        gain[i]=100;
                }
                else
                        gain[i]=1;
```

```
        }

        setpga();

                // Offset bei jeder Messung abziehen: sonst Fehler, falls Messwert nahe 0


        return;
}
*/
float wandeln(STEUERSTRUCT st)
{
 unsigned char lo_byte, hi_byte;
 signed short sdaten_wort; // Achtung: Bei VC ist ein int 32 bit lang!
 unsigned short udaten_wort;
 float u=0;


 start();
 i2c_ausgeben(st.adresse);
 i2c_ausgeben(st.kontrolle);
 stop();

 //for(i=1; i<=bremse;i++);

 start();
 i2c_ausgeben(st.adresse+1); /* 0101000 1 für lesen */
 hi_byte = i2c_einlesen();
 Ack();
 lo_byte = i2c_einlesen();
 keinAck();
 stop();
 if(st.bipolar == 0)
 {
      udaten_wort = (hi_byte << 8) + lo_byte;
      u = udaten_wort * st.faktor;
 }
 else
 {
      sdaten_wort = (signed) ((hi_byte << 8) + lo_byte);
      u = sdaten_wort * st.faktor;
 }
 return(u);
}

void start(void)
{

 //_outp(ba+2, 1); /* SDA = 0 */
 PORTA=PORTA & 0xfe;
 //for(i=1; i<=adc_scl_bremse;i++);    // neu
 //_outp(ba+2, 3); /* SCL = 0, SDA immer noch 0 */
 PORTA=PORTA & 0xfd;
}

void stop(void)
{

        //_outp(ba+2, 3); /* SCL = 0 und SDA = 0 */
        PORTA=PORTA & 0xfc;
   //for(i=1; i<=adc_scl_bremse;i++);    // neu
        //_outp(ba+2, 1); /* SCL = 1 */
        PORTA=(PORTA & 0xfd) + 0x02;
```

```
            //for(i=1; i<=adc_scl_bremse;i++);    // neu
            //_outp(ba+2, 0); /* SDA = 1 */
            PORTA=(PORTA & 0xfe) + 0x01;
}

void i2c_ausgeben(unsigned char byte)
{
 unsigned char bitwert, portwert;
 //unsigned char porteingabe;
 int bit;

 for(bit=7; bit >=0; bit--)
 {
  bitwert = 1 << bit;
  if((byte & bitwert) == bitwert) portwert = 1; else portwert = 0;
  //_outp(ba+2, portwert); /* SDA setzen */
          PORTA=(PORTA & 0xfe) + portwert;

  //for(i=1; i<=adc_scl_bremse;i++);

   //_outp(ba+2, portwert - 2); /* SCL = 1 bei SDA gesetzt */
           PORTA=(PORTA & 0xfd) + 0x02;

  //for(i=1; i<=adc_scl_bremse;i++);

   //_outp(ba+2, portwert); /* SCL wieder = 0 */
           PORTA=(PORTA & 0xfd);
 }
 //_outp(ba+2, 2); /* SDA = 1 und SCL = 0 */
 PORTA=(PORTA & 0xfe) + 1;
 //for(i=1; i<=adc_scl_bremse;i++);
 //_outp(ba+2, 0); // SCL = 1 SDA i2c_einlesen
 PORTA=(PORTA & 0xfd) + 0x02;
 //for(i=1; i<=adc_scl_bremse;i++);

//   porteingabe = _inp(ba+2); // Ack vom MAX128 wird nicht eingelesen, weil es sowieso nicht
ausgewertet wird
//  ls245out();

//  _outp(ba+2, 2); /* SDA = 1 und SCL = 0 */
    PORTA=PORTA & 0xfd ;
}

unsigned char i2c_einlesen(void)
{
 unsigned char bitwert = 128, wert = 0, n;


 //for(i=1; i<=adc_scl_bremse;i++);    // neu
 //_outp(ba+2, 2); /* SDA = 1 und SCL = 0 */
 PORTA=(PORTA & 0xfe) + 1;
 for(n = 1; n <= 8; n++)
 {
  //for(i=1; i<=adc_scl_bremse;i++);   //
  DDRA=0xfe; // A0, SDA auf einlesen
          //_outp(ba+2, 0); /* SCL = 1 SDA i2c_einlesen */
          PORTA=(PORTA & 0xfd) + 0x02;
  //for(i=1; i<=adc_scl_bremse;i++);
  if((PINA & 1) == 1) wert += bitwert;
  //for(i=1; i<=adc_scl_bremse;i++);    // neu
          //_outp(ba+2, 2); /* SCL = 0 */
          PORTA=PORTA & 0xfd;
  //ls245out();
```

```
   bitwert /= 2;
          DDRA=0xff; // A0, SDA aufschreiben
 }
 return(wert);
}

void Ack(void)
{

 //_outp(ba+2, 3); /* SCL = 0 und SDA = 0 */
 PORTA=PORTA & 0xfc;
 //for(i=1; i<=bremse;i++);
 //_outp(ba+2, 1); /* SCL = 1 */
 PORTA=(PORTA & 0xfd) + 0x02;
 //for(i=1;i<=adc_scl_bremse;i++);
 //_outp(ba+2, 3); /* SCL = 0 */
 PORTA=(PORTA & 0xfd);
}

void keinAck(void)
{


 //_outp(ba+2, 2); /* SCL = 0 und SDA = 1 */
 PORTA=(PORTA & 0xfc) + 0x01;

 //for(i=1; i<=bremse;i++);
 //_outp(ba+2, 0); /* SCL = 1 */
 PORTA=(PORTA & 0xfd) + 0x02;
 //for(i=1;i<=adc_scl_bremse;i++);
 //_outp(ba+2, 2); /* SCL = 0 */
 PORTA=PORTA & 0xfd;
}

void init(void)
{
   //_outp(ba+2, 0); /* SDA und SCL = 1 */
          PORTA=(PORTA & 0xfc) + 0x03;
}

void initstruct(STEUERSTRUCT *st)
{
 st->adresse = (1 << 6) + (1 << 4); /* 0101000 0 */

 /* st->kanal = 0; // nur 1.Kanal auslesen */
 /* st->range = 1; // groáer Bereich */
 /* st->bipolar = 0; // 1 heiát bipolar */
 st->kontrolle = (1 << 7)  /* Bit 7 ist immer hi */
             + (st->kanal << 4) /* Bit 6-4 Kanalnummer */
             + (st->range << 3) /* Bit 3 Range */
             + (st->bipolar << 2); /* Bit 2 bipolar */
 /* Bit 1 und 0 sind immer 0, da kein Stromsparmodus aktiviert wird */

 if(st->range == 1) /* also Bereich 0..Vref */
       st->fullscale = (float) Vref; /* Multiplikator * U(Referenz) */
 else st->fullscale = (float) Vref/2;
 if(st->bipolar == 0) st->faktor = st->fullscale / 65535;
 else st->faktor = st->fullscale / 32767;
}

void max128getufast(double *u) // je eine Messung aller Kanäle mit Faktor 1  wenn  Faktor nicht Null ist
{
int i=0;
```

```
        st.bipolar = 1; // bipolar
   st.range = 1;

        PORTD=0xff;

        for(i=0;i<7;i++)
        {
                u[i]=0;
        }

        for(i=0;i<7;i++)
        {
                if(gain[i]!=0)
                {
                        st.kanal = i; // Kanal i
                        initstruct(&st);
                        u[i]+= ((double) wandeln(st) - offset);
                }
                else
                {
                        wandeln(st) - offset;
                        u[i]=0;
                }
        }

                // Offset bei jeder Messung abziehen: sonst Fehler, falls Messwert nahe 0


        return;
}
```

11.7.3.4 max542.c

```
// Funktionen für die Ansteuerung des DAC MAX542
#include "max542.h"
#include "hardware.h"
#include <avr/pgmspace.h>


void lpt_ausgebenlf(unsigned char byte)
{
unsigned char bitwert, portwert;
int bit;
int portbuffer=0;

        portbuffer=PINB;

        for(bit=7; bit >=0; bit--)
        {
                bitwert = 1 << bit;
                if((byte & bitwert) == bitwert) portwert = 0x10; else portwert = 0;
                {
                        PORTB=(portbuffer & 0xef) + portwert; // PIN B4 DIN
                }

                portbuffer=(portbuffer&0xDF) + 0x20; /* SCLK = 1 bei DIN gesetzt */
                PORTB=portbuffer; // PIN B5

                portbuffer=portbuffer&0xDF; /* SCLK = 0 bei DIN gesetzt */
                PORTB=portbuffer; // PIN B5
```

```
        }
}


void max542lf(double u,int dac) // gibt eine Spannung aus
{
unsigned short datenwort;
unsigned char lobyte, hibyte;

        if(u >= UREFMAX542) u = UREFMAX542*32767/32768; /* Ausgabe über Vref nicht möglich
*/
        if(u < -UREFMAX542) u = -UREFMAX542;
        u += UREFMAX542;
        datenwort = (unsigned short) 32768 * u / UREFMAX542; // Konvertierung ist absichtlich so
gewählt
        lobyte = datenwort & 255;
        hibyte = datenwort >> 8;

        cslf(LO,dac);
        lpt_ausgebenlf(hibyte);
        lpt_ausgebenlf(lobyte);
        cslf(HI,dac);
}



void cslf(unsigned char wert,int dac)
{
int portbuffer=0;

        if(dac==1) // 30Kanal DAC1
        {
                portbuffer=PIND;
                portbuffer=(portbuffer & 0x7F)+0x80*wert; //pin D7 ; cs setzen
                //_outp(ba+2,portbuffer);
                PORTD=portbuffer;
        }
        else if(dac==2) // 30Kanal DAC2
        {
                portbuffer=PIND;
                portbuffer=(portbuffer & 0xFb)+0x4*(wert); //PIN D2 ; cs setzen
                //_outp(ba+2,portbuffer);
                PORTD=portbuffer;
        }
}

void uausgabe(double u) // Gibt eine Spannung aus
{
        max542lf(u/((double)R1DAC/R2DAC+1),1);
        //max542lf(u/((double)R1DAC/R2DAC+1),2);
}
```

11.7.3.5 small.h

```
void stoerungthermo(int cond); // setzt Störungsrelais  auf aus und Störungs- LED auf an
void ledmessenthermo(int led);
void wandelfeldruehrer(int cond); // schaltet Wandelfeldrührer an und aus
void ledmessenlf(int led);
double round(double x);
void warten1(void);
void pause (int ms);  // in ms bei 8 Mhz
void USART_Init( unsigned int baud );
```

```
void USART_Transmit( unsigned char data );
void USART_Transmit_string(unsigned char *data);
void warten1(void);
unsigned char USART_Receive(void);
```

11.7.3.6 max128.h

```
double setoffset(void); // Misst den Offsetkanal
void max128getu(double *u); // Spannungenmessen
void setpga(void); //PGA einstellen
void max128setpgaauto(void); // setzt den pga automatisch für Kanal 0 und 1

#define Vref 4.096 // Gerätspezifische Parameter

#define LO 0 // mit Optokoppler 255 MM
#define HI 1 // mit Optokoppler 0 MM
#define FEHLER -1
#define OK 1

typedef struct
{
	unsigned char adresse;
	unsigned char kontrolle;
	unsigned char kanal; // von 0 bis 7
	unsigned char bipolar; // bei 1 wird bipolar gemessen
	unsigned char range; // 1 ist Vref und 0 Vref/2
	float fullscale;
	float faktor;
	float u;
}
STEUERSTRUCT;

void cs(unsigned char wert);
float wandeln(STEUERSTRUCT st);
unsigned char i2c_einlesen(void);
void start(void);
void stop(void);
void i2c_ausgeben(unsigned char byte);
unsigned char i2c_einlesen(void);
void Ack(void);
void keinAck(void);
void init(void);
void initstruct(STEUERSTRUCT *st);
void max128getufast(double *u);
```

11.7.3.7 max542.h

```
void lpt_ausgebenthermo(unsigned char byte);
void max542thermo(double u,int dac); // gibt eine Spannung aus
void csthermo(unsigned char wert,int dac);
void lpt_ausgebenlf(unsigned char byte);
void max542lf(double u,int dac); // gibt eine Spannung aus
void cslf(unsigned char wert,int dac);
void u_ausgabe(double u); // gibt eine Spannung am Ausgang aus
void motor(double u); // steuert den Motor 0 bis 100%
void uausgabe(double u); // gibt eine Spannung u aus
```

11.7.3.8 hardware.h

```
// Definitionen, die direkt die Hardware betreffen
```

```
//Geräte Parameter
#define UREF 2.5 // Spannungsreferenz der Temperaturmessung
#define UREFMAX542 2.5 // Spannungsreferenz des DAC
#define R1DAC 100000  // Spannungsteiler Verstäker DAC
#define R2DAC 10000 // Spannungsteiler Verstäker DAC

#define R1ADC 100e3 // Spannungsteiler ADC Abschwächer
#define R2ADC 50e3 // Spannungsteiler ADC Abschwächer

#define THERMA 1.068981e-3 //Standardkalibrierdaten für Thermistoren
#define THERMB 2.120700e-4
#define THERMC 9.019537e-8
#define R1 100e3
#define RSTROMKLEIN 25
#define RSTROMGROSS 1000

// logigpegel
#define HI 1
#define LO 0
```

11.7.4 Zyklisiergerät, PC-Software

11.7.4.1 grafik.cpp

```
/*Funktionen für die graphische Darstellung*/

#include "StdAfx.h"

//-- Globale Variablen --
VIEWSTRUCT view;
extern HINSTANCE hInst;
extern DATENSTRUCT daten[maxdatasets]; // aus mess.cpp
extern AWSTRUCT aw; // aus mess.cpp
/*void fensterloeschen(HDC hPaint,int maxwinx,int maxwiny)
{
        SelectObject(hPaint, GetStockObject(NULL_PEN));
   SelectObject(hPaint, GetStockObject(WHITE_BRUSH));
   Rectangle(hPaint,0,0,maxwinx,maxwiny);
}
*/
/*****************************************************************************/
/*Funktion zum Zeichnen vom Kosy und Ausgabe der Datenpunkte
                */
/*****************************************************************************/
void drawkosy(HWND hWnd, int flag)
{
        HDC hDc;
        PAINTSTRUCT ps;
        HBITMAP hUp,hRe;
        HPEN hStift[maxdatasets];
        COLORREF farben[maxdatasets] = {        RGB(128,64,0),    // braun
                                                                  RGB(255,0,0),
        // hellrot
                                                                  RGB(215,215,0),
        // gelb
                                                                  RGB(0,255,0),
        // hellgrün
                                                                  RGB(0,128,0),
        // dunkelgrün
```

```
                                                                    RGB(0,0,255),
        // hellblau
                                                                    RGB(0,0,128),
        // dunkelblau
                                                                    RGB(128,0,128),
        // violett

        RGB(128,128,128), // grau
                                                                    RGB(0,0,0)};
        // schwarz

        int startx,starty,dx,dy,i, dataset;
        int xpixel, ypixel; // für das Zeichnen der Auswertung
        double ymin, ymax; // y-Werte für die Tangente beim Zeichnen der Auswertung
        double du, di, xwertmax, ywertmax, xwertmin, ywertmin;
        unsigned long int datenpunkt;
        char outtext[50];

        updateviewstruct(hWnd, DONTCHANGE); // Größe des Fensters feststellen

  if(flag==PAINTMSG)
    hDc=BeginPaint(hWnd, &ps);
  else
        {
    hDc=GetDC(hWnd);
                fensterloeschen(hDc,view.maxwinx,view.maxwiny);
  }

        for(i=0; i<maxdatasets; i++) hStift[i]=CreatePen(PS_SOLID,0,farben[i]); // Stifte erstellen

        // Bestimmung der maximalen Koordinatenwerte für das Zeichnen der Achsen

        // Fallunterscheidung momentan vollkommen unnötig
        if( (flag==NOPAINTMSGAUSWERTUNG || flag==PAINTMSG) && (daten[0].datacount>1) )
// Darstellung skaliert, wenn Daten vorhanden
        {
                xwertmin=view.xmin; // minimaler Bereich X
                xwertmax=view.xmax; // maximaler Bereich für x-Achse
                ywertmin=view.ymin;
                ywertmax=view.ymax;
        }
        else    // Darstellung auf maximalen Messbereich skaliert
        {
                xwertmin=view.xmin;
                xwertmax=view.xmax;
                ywertmin=view.ymin;
                ywertmax=view.ymax;
        }

        // Zeichnen der Linien des kosy
        starty=(int) round((view.maxwiny-UAB-OAB)+OAB);
        startx=(int) round(LAB);

        hUp=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_UPFEIL));
        hRe=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_LPFEIL));
        SelectObject(hDc, GetStockObject(NULL_PEN));
  SelectObject(hDc, GetStockObject(BLACK_PEN));

        // Zeichnen der Achsen

        // X-Achse zeichnen
        MoveToEx(hDc,0+LAB-BALKENK-1,starty,NULL);
        LineTo(hDc,view.maxwinx-RAB+20,starty);
```

```
        DrawBitmap(hDc,hRe,view.maxwinx-RAB+20,starty-4); //Pfeil zeichnen

        // Y-Achse links zeichnen Spannungen
        MoveToEx(hDc,0+LAB,(int) round(view.maxwiny-UAB+4),NULL);
        LineTo(hDc,0+LAB,(int) round(0+OAB-14));
        DrawBitmap(hDc,hUp,0+LAB-4,0+OAB-14); //Pfeil zeichnen

  // Y-Achse rechts zeichnen Strom
        MoveToEx(hDc,0+LAB+10,(int) round(view.maxwiny-UAB+4),NULL);
        LineTo(hDc,0+LAB+10,(int) round(0+OAB-14));
        DrawBitmap(hDc,hUp,view.maxwinx-RAB+20-4,0+OAB-14); //Pfeil zeichnen


        // Zeichnen der Tickmarks

        dy=(int) round(view.maxwiny-OAB-UAB)/ANZAHLYTICKS; //Berechnen der Abstände für
kleine Striche
        di=(ywertmax-ywertmin)/ANZAHLYTICKS; //  Differenz in y in der aufgetragenen Einheit

        dx=(int) round((view.maxwinx-RAB-startx)/ANZAHLXTICKS);
        du=(xwertmax-xwertmin)/ANZAHLXTICKS;// Differenz in x in der aufgetragenen Einheit

        SetTextAlign(hDc,TA_LEFT|TA_TOP); // Beschriftung der Y-Achse  setzen
        switch(view.iuflag)
        {
                case U: sprintf(outtext," U/V ");
                break;

                case I: sprintf(outtext," I/µA ");
                break;
        }

        TextOut(hDc,LAB+10,view.maxwiny-UAB-(ANZAHLYTICKS*dy)-
TEXTYDY,outtext,strlen(outtext));

        // Einheit X-Achse
        SetTextAlign(hDc,TA_CENTER|TA_BOTTOM);
        sprintf(outtext," U/V ");
        TextOut(hDc,startx+dx*ANZAHLXTICKS,starty-TEXTXDY,outtext,strlen(outtext));

        SetTextAlign(hDc,TA_LEFT|TA_TOP);
  for(i=0;i<=ANZAHLYTICKS;i++) // Y-Achse
  {
    MoveToEx(hDc,LAB-BALKENK,(int)          (starty-(di*i)/(ywertmax-ywertmin)*(view.maxwiny-UAB-
OAB)),NULL);
    LineTo(hDc,LAB+BALKENK+1,(int)          (starty-(di*i)/(ywertmax-ywertmin)*(view.maxwiny-UAB-
OAB)));

                sprintf(outtext,"% .3lf",(ywertmin+di*i));
    TextOut(hDc,LAB+TEXTYDX,(int)  (starty-(di*i)/(ywertmax-ywertmin)*(view.maxwiny-UAB-OAB))-
TEXTYDY,outtext,strlen(outtext));
  }

        SetTextAlign(hDc,TA_CENTER|TA_TOP);

  for(i=1;i<=ANZAHLXTICKS;i++) // X-Achse Zahlen und Ticks
  {
    MoveToEx(hDc,startx+i*dx,starty-BALKENK,NULL);
    LineTo(hDc,startx+i*dx,starty+BALKENK+1);
    sprintf(outtext,"%.3lf",xwertmin+i*du);
    TextOut(hDc,startx+i*dx,starty+TEXTXDY,outtext,strlen(outtext));
  }
```

```
        // hier stand mal ein SetTextAlign
  if(flag==PAINTMSG || flag==NOPAINTMSGAUSWERTUNG) // also die Kurven zeichnen
        {
                SelectObject(hDc, GetStockObject(BLACK_PEN));

                // ************************ Ausgabe der Dateinamen oben im Fenster ******************************
                SetTextAlign(hDc,TA_LEFT|TA_TOP|TA_UPDATECP);
                // Cursor soll von TextOut aktualisiert werden: siehe Visual C++ 6.0 Seite 154
                MoveToEx(hDc, 5, 5, NULL); // Anfangsposition für Text
                for(dataset=0; dataset < daten[0].datasets; dataset++) // Schleife über alle Datensätze
                {
                        SetTextColor(hDc, farben[dataset]);
                        sprintf(outtext, "%i=", dataset);
                        TextOut(hDc, 0, 0, outtext, strlen(outtext));
                        TextOut(hDc, 0, 0, daten[dataset].dateiname,strlen(daten[dataset].dateiname));
                        TextOut(hDc, 0, 0, "   ", 3); // noch 3 Leerzeichen zwischen den Dateinamen
                        // Koordinaten werden ignoriert wegen TA_UPDATECP:siehe Visual C++ 6.0 Seite 154
                }

                // ******* Zeichnen der Kurven (zwei Schleifen, damit Cursorposition nicht durcheinander kommt) *****
                for(dataset=0; dataset < daten[0].datasets; dataset++) // Schleife über alle Datensätze
                {
                        SelectObject(hDc,hStift[dataset]);
                        for(datenpunkt = 1; datenpunkt < daten[dataset].datacount; datenpunkt++)
                        // hier  muss < stehen, da 0 mitgezählt wird
                        {
                                punktzeichnen(hWnd, hDc, dataset, datenpunkt, FALSE); // FALSE: keine Schrift dazu
                        } // Datenpunktschleife
                } // Datensatzschleife

                //          ***************************** Zeichnen der Auswertung *********************************
                if(view.awflag == TRUE)
                {
                        SetTextColor(hDc, farben[aw.dataset]); // Farbe für Text einstellen
                     SetTextAlign(hDc,TA_LEFT|TA_TOP);
                     sprintf(outtext,"Maximum X= %+.4f Y= %+.4f", daten[aw.dataset].xpeak, daten[aw.dataset].ypeak);
                        TextOut(hDc, 2*LAB, OAB,outtext,strlen(outtext));
                        sprintf(outtext,"Grundstrom X= %+.4f Y= %+.4f", daten[aw.dataset].xback, daten[aw.dataset].yback);
                        TextOut(hDc, 2*LAB, OAB + 20,outtext,strlen(outtext));
                        sprintf(outtext,"Maximum - Grundstrom Y= %+.4f", daten[aw.dataset].ypeak - daten[aw.dataset].yback);
                        TextOut(hDc, 2*LAB, OAB + 40,outtext,strlen(outtext));

                        // Peakmaximum zeichnen
                        xpixel = LAB + (int)round((daten[aw.dataset].xpeak - view.xmin) / (view.xmax - view.xmin) * (view.maxwinx-RAB-LAB));
                        ypixel = (int) round((view.maxwiny-UAB-OAB)+OAB) - (int)round((daten[aw.dataset].ypeak - view.ymin) / (view.ymax - view.ymin) * (view.maxwiny-UAB-OAB));

                        SelectObject(hDc, GetStockObject(BLACK_PEN)); // Linienfarbe schwarz
                         // SelectObject(hDc,hStift[aw.dataset]); // Linienfarbe entsprechend Dataset nicht günstig

                        MoveToEx(hDc, xpixel, 0, NULL); // senkrechte Linie
                        LineTo(hDc, xpixel, view.starty);
```

```
                        MoveToEx(hDc, xpixel - 30, ypixel, NULL); // waagrechte Linie
                        LineTo(hDc, xpixel + 30, ypixel);

                        // Backscan bzw. Tangente zeichnen
                        if(aw.tangente == TRUE)
                        {
                                if(view.iuflag == I)
                                {
                                        ymin = daten[aw.dataset].yl[aw.xminpunkt] *
daten[aw.dataset].iubereich;
                                        ymax = daten[aw.dataset].yl[aw.xmaxpunkt] *
daten[aw.dataset].iubereich;
                                }
                                else
                                {
                                        ymin = daten[aw.dataset].yl[aw.xminpunkt];
                                        ymax = daten[aw.dataset].yl[aw.xmaxpunkt];
                                }
                                xpixel = LAB + (int)round((daten[aw.dataset].xl[aw.xminpunkt] -
view.xmin) / (view.xmax - view.xmin) * (view.maxwinx-RAB-LAB));
                                ypixel = (int) round((view.maxwiny-UAB-OAB)+OAB) -
(int)round((ymin - view.ymin) / (view.ymax - view.ymin) * (view.maxwiny-UAB-OAB));
                                MoveToEx(hDc, xpixel, ypixel, NULL);
                                xpixel = LAB + (int)round((daten[aw.dataset].xl[aw.xmaxpunkt] -
view.xmin) / (view.xmax - view.xmin) * (view.maxwinx-RAB-LAB));
                                ypixel = (int) round((view.maxwiny-UAB-OAB)+OAB) -
(int)round((ymax - view.ymin) / (view.ymax - view.ymin) * (view.maxwiny-UAB-OAB));
                                LineTo(hDc, xpixel, ypixel);
                        }
                        else // also Backscan-Methode zeichnen
                        {
                                xpixel = LAB + (int)round((daten[aw.dataset].xback - view.xmin) /
(view.xmax - view.xmin) * (view.maxwinx-RAB-LAB));
                                ypixel = (int) round((view.maxwiny-UAB-OAB)+OAB) -
(int)round((daten[aw.dataset].yback - view.ymin) / (view.ymax - view.ymin) * (view.maxwiny-UAB-OAB));

                                MoveToEx(hDc, xpixel, ypixel - 50, NULL); // senkrechte Linie
                                LineTo(hDc, xpixel, view.starty);

                                MoveToEx(hDc, xpixel - 120, ypixel, NULL); // waagrechte Linie
                                LineTo(hDc, xpixel + 120, ypixel);
                        }
                } // Ende von view.awflag == TRUE

                SelectObject(hDc, GetStockObject(BLACK_PEN));

                DeleteObject(hUp);
                DeleteObject(hRe);

                for(dataset = 0; dataset < maxdatasets; dataset++) // Stifte wieder aufräumen
                        DeleteObject(hStift[dataset]);
        } // Ende von if(flag == PAINTMSG ||

        if(flag==PAINTMSG)
        {
                EndPaint(hWnd, &ps);
        }
  else
        {
                DeleteObject(hUp);
                DeleteObject(hRe);
                for(dataset = 0; dataset < maxdatasets; dataset++) // Stifte wieder aufräumen
                        DeleteObject(hStift[dataset]);
```

```
      ReleaseDC(hWnd,hDc);
          }

}

/*
void DrawBitmap( HDC hdc, HBITMAP bitmap, short x, short y )
{
BITMAP bitmapbuff;
HDC memorydc;
POINT origin;
POINT size;

   memorydc = CreateCompatibleDC( hdc );
   SelectObject( memorydc, bitmap );
   SetMapMode( memorydc, GetMapMode( hdc ) );
   GetObject( bitmap, sizeof( BITMAP ), (LPSTR) &bitmapbuff );

   origin.x = x;
   origin.y = y;
   size.x = bitmapbuff.bmWidth;
   size.y = bitmapbuff.bmHeight;

   DPtoLP( hdc, &origin, 1 );
   DPtoLP( memorydc, &size, 1 );

   BitBlt( hdc, origin.x, origin.y, size.x, size.y, memorydc, 0, 0, SRCCOPY);
   DeleteDC( memorydc );
}

double findmaxdouble(double *feld,unsigned long int count) // findet groesste Fließkommazahl in einem
Feld
{
          double dummy;
          unsigned long int i;

          dummy=feld[0];
          for(i=0;i<count;i++)
          {
                    if(feld[i]>=dummy) dummy = feld[i];
          }
          return(dummy);
}
*/
double findmindouble(double *feld,unsigned long int count) // findet kleinste Fließkommazahl in einem
Feld
{
          double dummy;
          unsigned long int i;

          dummy=feld[0];
          for(i=0;i<count;i++)
          {
                    if(feld[i] <= dummy) dummy = feld[i];
          }
          return(dummy);
}

double round(double x)
{
   if((x-floor(x))<=0.5)
      return(floor(x));
   else
```

```
    return(ceil(x));
}


void punktzeichnen(HWND hWnd, HDC hDc, int dataset, unsigned long datenpunkt, int showvalue)
{
        char outtext[101];
        int starty, xpixel, ypixel;
        double xwertmin, xwertmax, ywertmin, ywertmax, y, yalt;
/*      RECT Rect;

        // Fenstergröße feststellen
        GetClientRect(hWnd, &Rect);
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;
*/
        // 0-Koordinaten ausrechnen
        starty=(int) round((view.maxwiny-UAB-OAB)+OAB);
        xwertmin=view.xmin; // minimaler Bereich X
        xwertmax=view.xmax; // maximaler Bereich für x Achse
        ywertmin=view.ymin;
        ywertmax=view.ymax;

        switch(view.iuflag)
        {
                case U: y = daten[dataset].yl[datenpunkt];
                                yalt = daten[dataset].yl[datenpunkt-1];
                                break;
                case I: y = daten[dataset].yl[datenpunkt] * daten[dataset].iubereich;
                                yalt = daten[dataset].yl[datenpunkt-1] * daten[dataset].iubereich;
                                break;
        }
        // Liniensegment zeichnen
        if(datenpunkt == 0) //beim ersten Punkt geht keine Line
        {
                xpixel = LAB + (int)round((daten[dataset].xl[datenpunkt]- xwertmin) / (xwertmax -
xwertmin) * (view.maxwinx-RAB-LAB));
                ypixel = starty - (int)round((y - ywertmin) / (ywertmax - ywertmin) * (view.maxwiny-UAB-
OAB));
                SetPixel(hDc, xpixel, ypixel, RGB(230,0,0));
        }
        else // also ab Datenpunkt 1
        {
                xpixel = LAB + (int)round((daten[dataset].xl[datenpunkt-1]- xwertmin) / (xwertmax -
xwertmin) * (view.maxwinx-RAB-LAB));
                ypixel = starty - (int)round((yalt - ywertmin) / (ywertmax - ywertmin) * (view.maxwiny-
UAB-OAB));
                MoveToEx(hDc, xpixel, ypixel, NULL);

                xpixel = LAB + (int)round((daten[dataset].xl[datenpunkt]- xwertmin) / (xwertmax -
xwertmin) * (view.maxwinx-RAB-LAB));
                ypixel = starty - (int)round((y - ywertmin) / (ywertmax - ywertmin) * (view.maxwiny-UAB-
OAB));
                LineTo(hDc, xpixel, ypixel);
        }

        if(showvalue == TRUE) // und Messwert als Zahl rechts oben ausgeben, falls gewünscht
        {
           SetTextAlign(hDc,TA_RIGHT|TA_TOP);
                sprintf(outtext,"X= %+.3f Y= %+.3f", daten[dataset].xl[datenpunkt], y);
                TextOut(hDc, view.maxwinx-RAB, OAB,outtext,strlen(outtext));
        }
}
```

```
void updateviewstruct(HWND hWnd, int viewflag)
{
	int datensatz;
	double y[maxdatasets]; // Maximal- bzw. Minimalwert der Datensätze
	RECT Rect;

	// Fenstergröße feststellen
	GetClientRect(hWnd, &Rect);
	view.maxwinx=Rect.right;
	view.maxwiny=Rect.bottom;

	view.starty=(int) round((view.maxwiny-UAB-OAB)+OAB); // 0-Koordinate in y-Richtung

	switch(viewflag)
	{
		case MESSBEREICH:
			view.ymin = 0;
			view.ymax = 4;
			if(view.iuflag == I)
			{
				view.ymin *= view.maxiubereich;
				view.ymax *= view.maxiubereich;
			}
			break;
		case MAXDATEN:
			// Minimales y suchen, zunächst als Spannung
			for(datensatz = 0; datensatz < maxdatasets; datensatz++)
			{
				if(daten[datensatz].datacount == 0) // also leerer Datensatz
				{
					y[datensatz] = Vref; // größten Wert setzen, damit
Minimumsuche nicht gestört
					if(view.iuflag == I) y[datensatz] *= view.maxiubereich;
				}
				else
				{
					y[datensatz] = findmindouble(daten[datensatz].yl,
daten[datensatz].datacount);
					if(view.iuflag == I) y[datensatz] *= daten[datensatz].iubereich;
				}
			}
			view.ymin = findmindouble(y, maxdatasets);
			// Maximales y suchen, zunächst als Spannung
			for(datensatz = 0; datensatz < maxdatasets; datensatz++)
			{
				if(daten[datensatz].datacount == 0)
					y[datensatz] = 0;
				else y[datensatz] = findmaxdouble(daten[datensatz].yl,
daten[datensatz].datacount);
				if(view.iuflag == I) y[datensatz] *= daten[datensatz].iubereich;
			}
			view.ymax = findmaxdouble(y, maxdatasets);
			break;
		case DONTCHANGE:
			break;
	}
}

/*************************************************************************
* Dialogfunkion für die Ansichtseinstellungen                            *
* gibt FALSE zurück, falls Abbrechen gewählt wurde
	*
```

```
* gibt TRUE zurück, falls Ok gewählt wurde
                    *
*
                                                                        *
* ändert die Werte in view
*****************************************************************************/

BOOL FAR PASCAL setviewDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam)
{
char dummy[100];

   switch(msg)
   {
                    case WM_INITDIALOG :
                              sprintf(dummy,"%.3g", view.xmin);
                              SetDlgItemText(hDlg,IDC_XMIN, dummy);
                              sprintf(dummy,"%.3g", view.xmax);
                              SetDlgItemText(hDlg,IDC_XMAX, dummy);
                              sprintf(dummy,"%.3g", view.ymin);
                              SetDlgItemText(hDlg,IDC_YMIN, dummy);
                              sprintf(dummy,"%.3g", view.ymax);
                              SetDlgItemText(hDlg,IDC_YMAX, dummy);
                              if(view.iuflag == U) sprintf(dummy,"V");
                              else sprintf(dummy,"µA");
                              SetDlgItemText(hDlg,IDC_EINHEIT, dummy);
                              SetDlgItemText(hDlg,IDC_EINHEIT1, dummy);
      break;

                    case WM_COMMAND :
        switch(wParam)
        {
           case IDOK:
                                        GetDlgItemText(hDlg, IDC_XMIN, dummy, 99);
                                        view.xmin = atof(dummy);
                                        GetDlgItemText(hDlg, IDC_XMAX, dummy, 99);
                                        view.xmax = atof(dummy);
                                        GetDlgItemText(hDlg, IDC_YMIN, dummy, 99);
                                        view.ymin = atof(dummy);
                                        GetDlgItemText(hDlg, IDC_YMAX, dummy, 99);
                                        view.ymax = atof(dummy);

                                        EndDialog(hDlg,TRUE);
              return TRUE;
           break;

           case IDCANCEL :
              EndDialog(hDlg,FALSE);
              return TRUE;
           break;

        }
      break;

   }

 return(FALSE);
}
```

11.7.4.2 hardware.cpp

```
/*Messfunktionen*/
#include "StdAfx.h"
```

```
int ba;
double bremse = 20, scl_bremse = 20, adc_scl_bremse=20, hammerbremse=120000;
STEUERSTRUCT st;
unsigned char portbuffer=0; // hier wird der Wert gespeichert, der an der Schnittstelle anliegt,; dadurch kann man alles steuern


void cs(unsigned char wert)
{
      portbuffer=(portbuffer & 248)+wert;
      _outp(ba,portbuffer); /* CS ist bit 0 */

}

float wandeln(STEUERSTRUCT st)
{
 unsigned char lo_byte, hi_byte;
 signed short sdaten_wort; // Achtung: Bei VC ist ein int 32 bit lang!
 unsigned short udaten_wort;
 float u=0;
 int i=0;

 start();
 i2c_ausgeben(st.adresse);
 i2c_ausgeben(st.kontrolle);
 stop();

 for(i=1; i<=bremse;i++); // funktioniert mit Notebook

 start();
 i2c_ausgeben(st.adresse+1); /* 0101000 1 für lesen */
 hi_byte = i2c_einlesen();
 Ack();
 lo_byte = i2c_einlesen();
 keinAck();
 stop();
 if(st.bipolar == 0)
 {
      udaten_wort = (hi_byte << 8) + lo_byte;
      u = udaten_wort * st.faktor;
 }
 else
 {
      sdaten_wort = (signed) ((hi_byte << 8) + lo_byte);
      u = sdaten_wort * st.faktor;
 }
 return(u);
}

void start(void)
{
 long i;
 _outp(ba+2, 1); /* SDA = 0 */
 for(i=1; i<=adc_scl_bremse;i++);    // neu
 _outp(ba+2, 3); /* SCL = 0, SDA immer noch 0 */
}

void stop(void)
{
          long i;
          _outp(ba+2, 3); /* SCL = 0 und SDA = 0 */
   for(i=1; i<=adc_scl_bremse;i++);    // neu
          _outp(ba+2, 1); /* SCL = 1 */
```

```
            for(i=1; i<=adc_scl_bremse;i++);    // neu
            _outp(ba+2, 0); /* SDA = 1 */
}

void i2c_ausgeben(unsigned char byte)
{
 unsigned char bitwert, portwert;
 unsigned char porteingabe;
 int bit;
 long i;

 for(bit=7; bit >=0; bit--)
 {
  bitwert = 1 << bit;
  if((byte & bitwert) == bitwert) portwert = 2; else portwert = 3;
  _outp(ba+2, portwert); /* SDA setzen */

  for(i=1; i<=adc_scl_bremse;i++);

   _outp(ba+2, portwert - 2); /* SCL = 1 bei SDA gesetzt */

  for(i=1; i<=adc_scl_bremse;i++);

   _outp(ba+2, portwert); /* SCL wieder = 0 */
 }
 _outp(ba+2, 2); /* SDA = 1 und SCL = 0 */
 for(i=1; i<=adc_scl_bremse;i++); //     funktioniert mit Notebook
 _outp(ba+2, 0); // SCL = 1 SDA i2c_einlesen
 for(i=1; i<=adc_scl_bremse;i++);  //    funktioniert mit Notebook

 ls245in();
 porteingabe = _inp(ba+2); // Ack vom MAX128 wird nicht eingelesen, weil es sowieso nicht ausgewertet
wird
 ls245out();
 if((porteingabe & 1) == 0)
 {
      printf("\nIC antwortet nicht! - Taste drücken\n");
  getch();
 }

 _outp(ba+2, 2); /* SDA = 1 und SCL = 0 */
}

unsigned char i2c_einlesen(void)
{
 unsigned char bitwert = 128, wert = 0, n;
 long i;

 for(i=1; i<=adc_scl_bremse;i++);    // neu
 _outp(ba+2, 2); /* SDA = 1 und SCL = 0 */
 for(n = 1; n <= 8; n++)
 {
  for(i=1; i<=adc_scl_bremse;i++);   //  funktioniert mit Notebook
  _outp(ba+2, 0); /* SCL = 1 SDA i2c_einlesen */
  ls245in();
  for(i=1; i<=adc_scl_bremse;i++);    // funktioniert mit Notebook
  if((_inp(ba+2) & 1) == 0) wert += bitwert;
  for(i=1; i<=adc_scl_bremse;i++);    // neu
          _outp(ba+2, 2); /* SCL = 0 */
  ls245out();
  bitwert /= 2;
 }
 return(wert);
```

```
}

void Ack(void)
{
    int i;
  _outp(ba+2, 3); /* SCL = 0 und SDA = 0 */
  for(i=1; i<=bremse;i++);
  _outp(ba+2, 1); /* SCL = 1 */
  for(i=1;i<=adc_scl_bremse;i++);
  _outp(ba+2, 3); /* SCL = 0 */
}

void keinAck(void)
{
  int i;

  _outp(ba+2, 2); /* SCL = 0 und SDA = 1 */
  for(i=1; i<=bremse;i++);
  _outp(ba+2, 0); /* SCL = 1 */
  for(i=1;i<=adc_scl_bremse;i++);
  _outp(ba+2, 2); /* SCL = 0 */
}

void init(void)
{
   _outp(ba+2, 0); /* SDA und SCL = 1 */

   portbuffer=(portbuffer & 247);
   _outp(ba,portbuffer); //Hammer =0
   ls245out(); /* sonst ist der erste Schreibzugriff auf den MAX128 verloren */
}

void initstruct(STEUERSTRUCT *st)
{
  st->adresse = (1 << 6) + (1 << 4); /* 0101000 0 */

  /* st->kanal = 0; // nur 1.Kanal auslesen */
  /* st->range = 1; // groáer Bereich */
  /* st->bipolar = 0; // 1 heiát bipolar */
  st->kontrolle = (1 << 7)  /* Bit 7 ist immer hi */
              + (st->kanal << 4) /* Bit 6-4 Kanalnummer */
              + (st->range << 3) /* Bit 3 Range */
              + (st->bipolar << 2); /* Bit 2 bipolar */
  /* Bit 1 und 0 sind immer 0, da kein Stromsparmodus aktiviert wird */

  if(st->range == 1) /* also Bereich 0..Vref */
       st->fullscale = (float) Vref; /* Multiplikator * U(Referenz) */
  else st->fullscale = (float) Vref/2;
  if(st->bipolar == 0) st->faktor = st->fullscale / 65535;
  else st->faktor = st->fullscale / 32767;
}

void ls245in(void)
{
   portbuffer=(portbuffer & 191) + 64; // Datenbit 6=1 bedeutet vom Polarographen zum Rechner
   /* ************************************************
            * Achtung: nur bei Petrak-Platine, sonst Datenbit 7=1 *
            ************************************************/
            _outp(ba,portbuffer);
}

void ls245out(void)
{
```

```
   portbuffer=(portbuffer & 191); // Datenbit 6=0 bedeutet vom Rechner zum Polarographen
   /* ************************************************
         * Achtung: nur bei Petrak-Platine, sonst Datenbit 7=0 *
         ************************************************/
   _outp(ba,portbuffer);
}
```

11.7.4.3 mess.cpp

```
#include "StdAfx.h"

// Globale Variablen
DATENSTRUCT daten[maxdatasets];
AWSTRUCT aw;
MESSTRUCT messparam; // Parameter für die Messung

extern HINSTANCE hInst; // aus schreiber.cpp
extern VIEWSTRUCT view; // aus grafik.cpp
extern STEUERSTRUCT st; // aus hardware.c
extern int ba; // aus hardware.cpp

void globalinit(HWND hWnd)
{
         HMENU hMenu;
         ba = 0x378;

         hMenu=GetMenu(hWnd); // Handle auf Menü holen

   daten[0].adbereich=1; // 1 ist Vref und 0 Vref/2
   daten[0].bipolar=1; // 1 für bipolar
         daten[0].datasets = 0; // nichts geladen

         messparam.zeit = 100; // 100 ms voreingestellt
         messparam.filter = 0.001; // 1 mV voreingestellt
         messparam.iubereich = 1; // in µA/V
         view.xmin = -4;
   view.xmax = 4;
         view.maxiubereich = 1;

         // Spannungsanzeige einstellen
         view.iuflag = U;
         CheckMenuItem(hMenu, IDM_ANSICHT_I, MF_UNCHECKED);

         // Ansicht auf Meßbereich schalten
         CheckMenuItem(hMenu, IDM_ANSICHT_MEBEREICH, MF_CHECKED);
         CheckMenuItem(hMenu, IDM_ANSICHT_DATENMAX, MF_UNCHECKED);
         updateviewstruct(hWnd, MESSBEREICH);

         aw.xmin=-0.48;
         aw.xmax=-0.30;
         aw.tangente = TRUE;
         aw.manuell = FALSE;
         aw.xpeakmanu=-0.38;
}


/* ********************************************************************************************
*    Messroutine für Nanotracer                                                    *
*********************************************************************************************/

void mess(HWND hWnd)
{
```

```
   HDC hDc;
   double offset;
          int dataset;

   hDc=GetDC(hWnd); // wird für  Punktzeichnen() in Einzelmessungen()  benötigt
   // drawkosy(hWnd,NOPAINTMSGAUSWERTUNG); 9.9.02
          dataset = speicherallokieren(hWnd);
          if(dataset == -1) return; // Abbruch, falls Allokieren nicht geglückt ist

          // Meessparameter abfragen
          if(DialogBoxParam(hInst, (LPCTSTR)IDD_MESS, hWnd, (DLGPROC) MessDialog,
dataset)==FALSE)
          {
                cleanup(dataset); // allokierten Speicher wieder freigeben
                return; // und Messung abbrechen
          }
   init();
   // Offset des Wandlers durch Messen eines auf Masse gelegten Eingangs bestimmen
          offset = getadcoffset(4);
   daten[dataset].datacount = 0;

   GetAsyncKeyState(VK_ESCAPE); // ESC-Status auslesen, damit vorherige Tastendrücke verworfen
werden
   do
   {
      einzelmess(hWnd, hDc, dataset, offset);
   } while((GetAsyncKeyState(VK_ESCAPE) == 0) && (daten[dataset].datacount < maxdatenpunkte));

   // wird nicht ausgeführt, wenn ESC gedrückt wurde: nicht abgeholtes ESC schließt Box
   MessageBox(hWnd, "Messung beendet", "Info", MB_OK|MB_ICONINFORMATION);
   save(hWnd, dataset);
   drawkosy(hWnd,NOPAINTMSGAUSWERTUNG); // damit Dateiname nachgetragen wird
   ReleaseDC(hWnd,hDc);
} /* ********************** Ende mess ******************************/

void einzelmess(HWND hWnd, HDC hDc, int dataset, double offset)
{
          float TimeUnit=0; // für Zeitfunktionen aus My_Timer.cpp
          LARGE_INTEGER StartTime; // für Zeitfunktionen aus My_Timer.cpp

          int messung = 0;
   double ux=0, uy=0, oldux;

   st.bipolar = 1; // bipolar
   st.range = daten[0].adbereich;

   StartTime=TimeInit(&TimeUnit);
   do
   {
      st.kanal = 0; // X auf Kanal 0
      initstruct(&st);
      ux += (double) wandeln(st) - offset;
                // Offset bei jeder Messung abziehen: sonst Fehler, falls Meßwert Messwert nahe 0

      st.kanal = 1; // Y auf Kanal 1
      initstruct(&st);
      uy += (double) wandeln(st) - offset;
                messung++;
   } while(TimeRead(StartTime,TimeUnit) <= messparam.zeit);

          // Mittelwert ausrechnen
          ux = ux / messung;
   uy = uy / messung;
```

```
   if(daten[dataset].datacount > 0)
   {
      oldux = daten[dataset].xl[daten[dataset].datacount - 1];
                  if(fabs(oldux - ux) >= messparam.filter) // nur wenn Messpunkte mehr als Filterwert
auseinanderliegen
      {
                              daten[dataset].xl[daten[dataset].datacount] = ux;
         daten[dataset].yl[daten[dataset].datacount] = uy;
                              punktzeichnen(hWnd, hDc, dataset, daten[dataset].datacount, TRUE);
                              // immer den letzen Punkt; TRUE heißt mit Ausgabe der Messwerte
         daten[dataset].datacount++;
      }
   }
   else // also  ersten Messpunkt , in jedem Fall speichern
   {
                              daten[dataset].xl[daten[dataset].datacount] = ux;
         daten[dataset].yl[daten[dataset].datacount] = uy;
                              punktzeichnen(hWnd, hDc, dataset, daten[dataset].datacount, TRUE);
                              // immer den letzen Punkt; TRUE heißt mit Ausgabe der  Messwerte
         daten[dataset].datacount++;
           }
}

int save(HWND hWnd, int dataset)
{
   static char szFilename[81];
   int iErg=0;
//    char *feld,
   char zeile[40];
   unsigned long int i=0;
   HFILE hFile;
   size_t lBytes; // size_t ist unsigned int laut stdio.h
   OPENFILENAME ofn;

   memset(&ofn, 0, sizeof(OPENFILENAME));
   ofn.lpstrFile = szFilename; // Speicherort für Dateinamen
          ofn.lpstrFileTitle = daten[dataset].dateiname; // Dateiname ohne Pfad
          ofn.nMaxFileTitle = 80;     // maximale Länge des Dateinamens ohne Pfad, vgl. mess.h

   iErg = FileDialogSave(hWnd, &ofn);

   if(iErg == IDOK)
   {
      if((hFile = _lcreat(ofn.lpstrFile,0)) == HFILE_ERROR)
      {
         MessageBox(hWnd, "Datei konnte nicht angelegt werden!", "Fehler", MB_OK);
         return(FEHLER);
      }

      // An den Dateianfang gehen
      _llseek(hFile, 0L, 0);

            //  Messbereich Nanotracer in µA/V schreiben
      sprintf(zeile,"9\t%g\r\n", daten[dataset].iubereich);
                  lBytes = _hwrite(hFile,zeile,strlen(zeile));
      if(lBytes!=strlen(zeile))
      {
                              MessageBox(hWnd, "Fehler beim Schreiben in die Datei.", "Fehler",
MB_OK|MB_ICONSTOP);
         _lclose(hFile);
         return(FEHLER);
      }
```

```
                    // Messdaten schreiben
      while(i < daten[dataset].datacount)
      {
         sprintf(zeile,"%.4g\t%.4g\r\n", daten[dataset].xl[i], daten[dataset].yl[i]);
                                 // alte Formatstring: "%.4le\t%.4le\r\n"
         lBytes = _hwrite(hFile,zeile,strlen(zeile));
         if(lBytes!=strlen(zeile))
         {
            MessageBox(hWnd, "Fehler beim Schreiben in die Datei.", "Fehler", MB_OK|MB_ICONSTOP);
            _lclose(hFile);
            return(FEHLER);
         }
         else
         {
           //_llseek(hFile, lBytes, 1);
         }
         i++;
       }
       _lclose(hFile);
   }
   return(OK);
}


int FileDialogSave(HWND hWnd, OPENFILENAME *ofn)
{
   char szFilter[256], szText[]= "TXT-Datei (*.TXT)|*.txt|Alle Dateien (*.*)|*.*||";
   int i;

   for(i=0; szText[i]!= '\0'; ++i) // Ersetzt | in szFilter und szText durch 0-Bytes, die GetSaveFileName
erwartet
      szFilter[i] = szText[i] == '|' ? '\0' : szText[i];

   ofn->lStructSize = sizeof(OPENFILENAME);
   ofn->hwndOwner = hWnd;
   ofn->lpstrFilter = szFilter;
   ofn->nFilterIndex = 1; // MM: gibt an, welcher Dateitypfilter verwendet wird: 0 = Default, 1 = 1. Eintrag
in szFilter
   ofn->nMaxFile = _MAX_PATH;
   ofn->nMaxFileTitle = _MAX_FNAME + _MAX_EXT;
   ofn->lpstrDefExt = "xy"; // MM: gibt der Anwender keine Erweiterung ein, wird .xy angehängt
   return GetSaveFileName(ofn)? IDOK : IDCANCEL;
}

/*****************************************************************************
* Einlesen von Datensätzen
*****************************************************************************/

int load(HWND hWnd)
{
   static char szFilename[81];
   int iErg=0, dataset, zeichenzahl;

   char zeichen, string[30];
         long lesestatus;

   HFILE hFile;
   OPENFILENAME ofn;

         dataset = speicherallokieren(hWnd); // erst Speicher allokieren
         if(dataset == -1) return(FEHLER);
```

```
   memset(&ofn, 0, sizeof(OPENFILENAME));
   ofn.lpstrFile = szFilename;
         ofn.lpstrFileTitle = daten[dataset].dateiname; // Dateiname ohne Pfad, soll von FileDialogOpen
gefüllt werden
         ofn.nMaxFileTitle = 80;     // maximale Länge des Dateinamens ohne Pfad, vgl. mess.h
   iErg = FileDialogOpen(hWnd, &ofn, 1);

   if(iErg == IDOK)
   {
      //Datei öffnen mit Fehlertest
      if((hFile = _lopen(ofn.lpstrFile, OF_READ)) == -1)
      {
         MessageBox(hWnd, "Datei konnte nicht geöffnet werden!", "Fehler", MB_OK);
         return(FEHLER);
      }
      _llseek(hFile, 0L, 0); // an den Dateianfang gehen


                  //Messbereich für Stromumrechnung einlesen
                  if(readiubereich(hWnd, hFile, dataset) == FEHLER)
                  {
                        _lclose(hFile);
                        cleanup(dataset);
                        daten[0].datasets--;
                        return(FEHLER);
                  }
                  if(daten[dataset].iubereich > view.maxiubereich) view.maxiubereich =
daten[dataset].iubereich;
                  do // Messdaten einlesen
                  {
                        zeichenzahl=0;
                        do // bis zum Tab einlesen
                        {
                              lesestatus = _hread(hFile, &zeichen ,1); // ein Byte lesen

                              string[zeichenzahl++] = zeichen;
                        }
                        while((string[zeichenzahl-1]!='\t') && lesestatus == 1); // bei Tab und EOF
abbrechen
                        if(lesestatus == 0) break;
                        string[zeichenzahl-1]=0; // Tab mit 0-Byte überschreiben
                        daten[dataset].xl[daten[dataset].datacount] = atof(string);

                        zeichenzahl=0;
                        do // bis zur neuen Zeile einlesen
                        {
                              lesestatus = _hread(hFile, &zeichen ,1); // ein Byte lesen
                              string[zeichenzahl++] = zeichen;
                        }
                        while(string[zeichenzahl-1]!='\r' && lesestatus == 1); // bei Return und EOF
abbrechen
                        if(lesestatus == 0) break;
                        string[zeichenzahl-1]=0; // Return mit 0-Byte überschreiben
                        daten[dataset].yl[daten[dataset].datacount] = atof(string);

                        lesestatus = _hread(hFile,&zeichen,1); // noch Zeilenvorschub einlesen

                        daten[dataset].datacount++;
                  } while ((lesestatus == 1) && (daten[dataset].datacount < maxdatenpunkte));

                  _lclose(hFile);

         }
```

```
        else // also auf abbrechen geklickt
        {
                cleanup(dataset); // darin wird auch daten[0].datasets-- ausgeführt
                return(FEHLER);
        }

        drawkosy(hWnd, NOPAINTMSGAUSWERTUNG); // jetzt geladene Daten zeichnen

   return(OK);
}


int FileDialogOpen(HWND hWnd, OPENFILENAME *ofn, short typ)
{
   char szFilter[256], szText[]= "XY-Datei (*.XY)|*.xy|Auswertungsdatei (*.TXT)|*.txt|Alle Dateien
(*.*)|*.*||";
        int i;

   for(i=0; szText[i]!= '\0'; ++i) // Ersetzt | in szFilter und szText durch 0-Bytes, die GetSaveFileName
erwartet
      szFilter[i] = szText[i] == '|' ? '\0' : szText[i];

   ofn->lStructSize = sizeof(OPENFILENAME);
   ofn->hwndOwner = hWnd;
   ofn->lpstrFilter = szFilter;
   ofn->nFilterIndex = typ; // gibt an, welcher Dateitypfilter verwendet wird: 0 = Default, 1 = 1. Eintrag in
szFilter
   ofn->nMaxFile = _MAX_PATH;
   ofn->nMaxFileTitle = _MAX_FNAME + _MAX_EXT;
   switch(typ)
        {
                case 1: ofn->lpstrDefExt = "xy"; // gibt der Anwender keine Erweiterung ein, wird das
angehängt
                        // ofn->lpstrFile = ".xy";
                                break;
                case 2: ofn->lpstrDefExt = "txt";
                                // ofn->lpstrFile = ".txt";
                                ofn->Flags = OFN_CREATEPROMPT; // falls Datei nicht exisitert,
fragen ob sie erzeugt werden soll
                                break;
        }
   return GetOpenFileName(ofn)? IDOK : IDCANCEL;
}

/*****************************************************************************
*                                        Dialogfunkion für die Auswahl des Datensatzes
                                *
* gewählter Datensatz (bei Abbruch 999) wird via Enddialog zurückgegeben                *
* Rückgabewerte via return(): entsprechend Windowsorgabe
                *
* FALSE: Standardbearbeitungsfunktion kümmert sich um Message
                *
* TRUE: Message ist bereits bearbeitet, Standardbearbeitungsfkt. nicht mehr aufrufen*
*****************************************************************************/

BOOL FAR PASCAL DatasetChooseDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM
lParam)
{
        char dummy[100];
        int dataset;

   switch(msg)
   {
```

```
                case WM_INITDIALOG :
                        sprintf(dummy,"%d",daten[0].datasets - 1); // voreingestellt: letzter Datensatz
        SetDlgItemText(hDlg,IDC_AWDATASET,dummy);
                        return TRUE; // TRUE: Standardbearbeitungsroutine nicht mehr aufrufen
                break;

                case WM_COMMAND :
        switch(wParam)
        {
          case IDOK:
            GetDlgItemText(hDlg,IDC_AWDATASET,dummy,99);
            dataset=atoi(dummy);

            if(dataset < 0)
            {
              MessageBox(hDlg, "Kleinster Datensatz ist 0.", "Achtung", MB_OK|MB_ICONSTOP);
              return FALSE;
            }
                                    else if(dataset > daten[0].datasets - 1) // C-Nomenklatur !!
            {
                                                MessageBox(hDlg, "So viele Datensätze sind nicht
geöffnet.", "Achtung", MB_OK|MB_ICONSTOP);
              return FALSE;
            }

            EndDialog(hDlg, dataset);
            return TRUE; // TRUE: Standardbearbeitungsroutine nicht mehr aufrufen
          break;

          case IDCANCEL :
            EndDialog(hDlg, 999); // 999 heißt: Abbruch gedrückt
            return TRUE; // TRUE: Standardbearbeitungsroutine nicht mehr aufrufen
          break;

        } // Ende von switch(wParam)
      break; // aus WM_COMMAND-Zweig raus
   } // Ende von switch(msg)

        return(FALSE); // FALSE: Die Standardbearbeitungsroutine soll sich um die Message kümmern.
}


int speicherallokieren(HWND hWnd)
{
   int dataset;

        // nächsten freien Datensatz suchen
        if(daten[0].datasets >= maxdatasets) // schon alle Datensätze voll
        {
          MessageBox(hWnd, "Zu viele Datensätze offen!", "Fehler", MB_OK);
      return(-1);
        }
        dataset = daten[0].datasets; // Array beginnt bei 0 !!

        // Speicher allokieren
        daten[dataset].xlHandle=GlobalAlloc(GHND,sizeof(double)*maxdatenpunkte);
   daten[dataset].xl= (double*) GlobalLock(daten[dataset].xlHandle);
   daten[dataset].ylHandle=GlobalAlloc(GHND,sizeof(double)*maxdatenpunkte);
   daten[dataset].yl= (double*) GlobalLock(daten[dataset].ylHandle);

        // Voreinstellungen für Parameter setzen
        daten[dataset].datacount = 0; // Datensatz hat momentan 0 Punkte
        daten[dataset].iubereich = messparam.iubereich; // in µA/V entsp. Nanotracer
```

```
	daten[dataset].vol = 0; // für Auswertung

	strncpy(daten[dataset].dateiname, "ungespeichert", sizeof(daten[dataset].dateiname));
	// "ungespeichert" wird beim Laden / Speichern überschrieben
	daten[0].datasets++;
	return(dataset);
}

void cleanup(int dataset)
{
	int lastdataset = daten[0].datasets - 1;

	if(dataset > lastdataset) return; // falls nicht existierender Datensatz gewählt, einfach nichts tun

	// Speicher des zu schließenden Datensatzes freigeben
	GlobalUnlock(daten[dataset].xlHandle);
	GlobalUnlock(daten[dataset].ylHandle);
	GlobalFree(daten[dataset].xlHandle);
	GlobalFree(daten[dataset].ylHandle);

	if(dataset == lastdataset)
	// falls letzter Datensatz geschlossen wird; trifft auch zu, falls nur ein einziger Datensatz offen ist
	{

		// Dateinamen mit leerem String überschreiben, für Dateinamenanzeige
		strncpy(daten[dataset].dateiname, "", sizeof(daten[dataset].dateiname));
		daten[dataset].datacount = 0; // 0 Datenpunkte, für Auswertung etc
		// Falls Auswertung angezeigt wird, das ausschalten
		if(dataset == aw.dataset) view.awflag = FALSE;
	}
	else // also Datensatz in der Mitte schließen
	{
		// Daten vom letzen Datensatz in die "Lücke" kopieren
		daten[dataset].xlHandle = daten[lastdataset].xlHandle;
		daten[dataset].ylHandle = daten[lastdataset].ylHandle;

		daten[dataset].xl = daten[lastdataset].xl;
		daten[dataset].yl = daten[lastdataset].yl;

		daten[dataset].datacount = daten[lastdataset].datacount;

		// adbereich, bipolar werden nicht umkopiert, da nur bei daten[0] genutzt
		daten[dataset].iubereich = daten[lastdataset].iubereich;
		daten[dataset].xpeak = daten[lastdataset].xpeak;
		daten[dataset].ypeak = daten[lastdataset].ypeak;
		daten[dataset].xback = daten[lastdataset].xback;
		daten[dataset].yback = daten[lastdataset].yback;
		daten[dataset].vol = daten[lastdataset].vol;

		strncpy(daten[dataset].dateiname, // Destination
				daten[lastdataset].dateiname, // Source
				sizeof(daten[lastdataset].dateiname)); // Länge

		if(lastdataset == aw.dataset) // also Datensatz mit Auswertung wird umkopiert
			aw.dataset = dataset;
	}
	daten[0].datasets--; // in jedem Fall Zählvariable für Datensätze dekrementieren

}

void auswert(HWND hWnd)
{
	HGLOBAL diffhandle;
```

```
	char outtext[201]; // für Bildschirmausgabe der Werte und Speicher-Dialogbox
	double a, b, ymin, ymax; // für Tangentenmethode
	double *diff, mindiff;
	unsigned long punkt, peakpunkt, mindiffpunkt;

	if(DialogBox(hInst, (LPCTSTR)IDD_AWDLG, hWnd, (DLGPROC) AWDialog)==FALSE)
	return; // bei Abbruch zurück

	// Suchbereich für Maximum Vorwärtsscan festlegen, wird bei Peaksuche UND Darstellung Tangente gebraucht
	aw.xminpunkt = 0;
	while(daten[aw.dataset].xl[aw.xminpunkt] < aw.xmin)
	{
		if(aw.xminpunkt >= daten[aw.dataset].datacount - 1) break; // Abbruch, falls Punkt nicht vorkommt
		aw.xminpunkt++; // erster Punkt INNERHALB der Grenzen
	}
	aw.xmaxpunkt = aw.xminpunkt;
	while(daten[aw.dataset].xl[aw.xmaxpunkt] < aw.xmax)
	{
		if(aw.xmaxpunkt >= daten[aw.dataset].datacount - 1) break; // Abbruch, falls Punkt nicht vorkommt
		aw.xmaxpunkt++;
	}

	if(aw.manuell == FALSE)
	{
		daten[aw.dataset].ypeak = -5; // -5 für den 1. Durchlauf der Schleife

		for(punkt = aw.xminpunkt; punkt < aw.xmaxpunkt; punkt++) // Schleife über Messpunkte im AW-Bereich
		{
			if(daten[aw.dataset].yl[punkt] > daten[aw.dataset].ypeak) // Problem: "Ausreißer"
			{
				daten[aw.dataset].ypeak = daten[aw.dataset].yl[punkt];
				daten[aw.dataset].xpeak = daten[aw.dataset].xl[punkt];
				peakpunkt = punkt;
			}
		}
	} // Ende von aw.manuell == FALSE
	else // also manuelle Maximumsuche
	{
		// Speicher für Differenzliste allokieren
		diffhandle=GlobalAlloc(GHND,sizeof(double)*daten[aw.dataset].datacount);
	   diff = (double*) GlobalLock(diffhandle);

		// Array mit Beträgen der Differenzen erstellen
		for(punkt = aw.xminpunkt; punkt <= aw.xmaxpunkt; punkt++)
		{
			diff[punkt] = fabs(daten[aw.dataset].xl[punkt] - aw.xpeakmanu);
		}

		// Punkt mit der kleinsten Differenz heraussuchen
		mindiff = diff[aw.xmaxpunkt]; //
		for(punkt = aw.xminpunkt; punkt <= aw.xmaxpunkt; punkt++)
		{
			if(diff[punkt] < mindiff)
			{
				mindiffpunkt = punkt;
				mindiff = diff[punkt];
			}
		}
```

```
                // gefundenes Maximum eintragen; Umgerechnet wird außerhalb der Fallunterscheidung
                daten[aw.dataset].xpeak = daten[aw.dataset].xl[mindiffpunkt];
                daten[aw.dataset].ypeak = daten[aw.dataset].yl[mindiffpunkt];

                // Speicher für Differenzliste freigeben
                GlobalUnlock(diffhandle);
                GlobalFree(diffhandle);

        } // Ende von aw.manuell == TRUE

        // falls im Stromanzeigemodus gearbeitet wird, Wert umrechnen
        if(view.iuflag == I) daten[aw.dataset].ypeak *= daten[aw.dataset].iubereich;

        // Fallunterscheidung, ob Tangentenmethode oder Backscanmethode
        if(aw.tangente == TRUE)
        {
                if(view.iuflag == I)
                {
                        ymin = daten[aw.dataset].yl[aw.xminpunkt] * daten[aw.dataset].iubereich;
                        ymax = daten[aw.dataset].yl[aw.xmaxpunkt] * daten[aw.dataset].iubereich;
                }
                else
                {
                        ymin = daten[aw.dataset].yl[aw.xminpunkt];
                        ymax = daten[aw.dataset].yl[aw.xmaxpunkt];
                }
                // Tangente y = a*x + b berechnen
                // Steigung a = (ymax - ymin) / (xmax - xmin)
                a = (ymax - ymin) / (daten[aw.dataset].xl[aw.xmaxpunkt] -
daten[aw.dataset].xl[aw.xminpunkt]);
                // Achsenabschnitt b = y - a*x
                b = ymax - a * daten[aw.dataset].xl[aw.xmaxpunkt];

                daten[aw.dataset].xback = daten[aw.dataset].xpeak; //  funktioniert hier im Gegensatz zum
Backscan exakt
                // y aus Tangentengleichung ausrechnen y = a*x + b
                daten[aw.dataset].yback = a * daten[aw.dataset].xback + b;

                // hier wurde, falls nötig, schon in Strom gerechnet, also ist keine Umrechnung mehr nötig
        }
        else // also aw.tangente == FALSE
        {
                // Wert für Backscan suchen: Speicher für Differenzliste allokieren
                diffhandle=GlobalAlloc(GHND,sizeof(double)*daten[aw.dataset].datacount);
           diff = (double*) GlobalLock(diffhandle);

                // Wert für Backscan suchen: Array mit Beträgen der Differenzen erstellen
                for(punkt = aw.xmaxpunkt; punkt < daten[aw.dataset].datacount - 1; punkt++)
                {
                        diff[punkt] = fabs(daten[aw.dataset].xl[punkt] - daten[aw.dataset].xpeak);
                }

                // Wert für Backscan suchen: Punkt mit der kleinsten Differenz heraussuchen
                mindiff = diff[aw.xmaxpunkt];
                for(punkt = aw.xmaxpunkt; punkt < daten[aw.dataset].datacount - 1; punkt++)
                {
                        if(diff[punkt] < mindiff)
                        {
                                mindiffpunkt = punkt;
                                mindiff = diff[punkt];
                        }
                }
```

```
                    // Backscan: Punkte eintragen
                    daten[aw.dataset].xback = daten[aw.dataset].xl[mindiffpunkt];
                    daten[aw.dataset].yback = daten[aw.dataset].yl[mindiffpunkt];


                    // falls im Stromanzeigemodus gearbeitet wird, Wert umrechnen
                    if(view.iuflag == I) daten[aw.dataset].yback *= daten[aw.dataset].iubereich;

                    // Speicher für Differenzliste freigeben
                    GlobalUnlock(diffhandle);
                    GlobalFree(diffhandle);
            } // Ende von else, Backscanmethode


            // graphische Darstellung des Ergebnisses
    view.awflag = TRUE;
            drawkosy(hWnd,NOPAINTMSGAUSWERTUNG); // AW zeichnen

    sprintf(outtext,"Speichern in %s?", aw.dateiname);
            if(MessageBox(hWnd, outtext, "Auswertung", MB_YESNO|MB_ICONQUESTION) == IDYES)
saveaw(hWnd, aw.dataset);
}

/*****************************************************************************
* Dialogfunkion für die Auswertung                                           *
*****************************************************************************/

BOOL FAR PASCAL AWDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam)
{
char dummy[100];

    switch(msg)
    {
                    case WM_INITDIALOG :
                            SetDlgItemInt(hDlg,IDC_AWDATASET, aw.dataset, TRUE); // TRUE heißt
signed Variable
                            sprintf(dummy,"%.3f", aw.xmin);
                            SetDlgItemText(hDlg,IDC_AWXMIN, dummy);
                            sprintf(dummy,"%.3f", aw.xmax);
                            SetDlgItemText(hDlg,IDC_AWXMAX, dummy);
                            sprintf(dummy,"%.1f", daten[aw.dataset].vol * 1e6); // Umrechnung l in µl
                            SetDlgItemText(hDlg,IDC_AW_VOL, dummy);

                            CheckDlgButton(hDlg,IDC_AW_MANUELL, aw.manuell);
                            sprintf(dummy,"%.3f", aw.xpeakmanu);
                            SetDlgItemText(hDlg,IDC_AW_XPEAK, dummy);

                            CheckDlgButton(hDlg,IDC_TANGENTE, aw.tangente);
                    break;

                    case WM_COMMAND :
            switch(wParam)
            {
               case IDOK:
                                            GetDlgItemText(hDlg, IDC_AWDATASET, dummy, 99);
                                            aw.dataset = atoi(dummy);
                                            GetDlgItemText(hDlg, IDC_AWXMIN, dummy, 99);
                                            aw.xmin = atof(dummy);
                                            GetDlgItemText(hDlg, IDC_AWXMAX, dummy, 99);
                                            aw.xmax = atof(dummy);
                  if(aw.dataset < 0)
                  {
                     MessageBox(hDlg, "Kleinster Datensatz ist 0.", "Achtung", MB_OK|MB_ICONSTOP);
```

```
                return FALSE;
            }
                                        else if(aw.dataset > daten[0].datasets - 1) // C-Nomenklatur !!
            {
                                                MessageBox(hDlg, "So viele Datensätze sind nicht
geöffnet.", "Achtung", MB_OK|MB_ICONSTOP);
                return FALSE;
            }

                                        GetDlgItemText(hDlg, IDC_AW_VOL, dummy, 99);
                                        daten[aw.dataset].vol = atof(dummy) * 1e-6; // Erst setzen,
wenn aw.dataset wirklich ok ist

                                        if(IsDlgButtonChecked(hDlg,
IDC_AW_MANUELL)==BST_CHECKED) aw.manuell = TRUE;
                                        else aw.manuell = FALSE;
                                        GetDlgItemText(hDlg, IDC_AW_XPEAK, dummy, 99);
                                        aw.xpeakmanu = atof(dummy);

                                        if(IsDlgButtonChecked(hDlg,
IDC_TANGENTE)==BST_CHECKED) aw.tangente = TRUE;
                                        else aw.tangente = FALSE;

                                        EndDialog(hDlg,TRUE);
            return TRUE;
          break;

          case IDCANCEL :
            EndDialog(hDlg,FALSE);
            return TRUE;
          break;

        }
      break;

   }

  return(FALSE);
}

void filtern(int dataset)
{
   unsigned long i, new_datacount = 0;
        double *new_x, *new_y;
        HGLOBAL handle_x, handle_y;

        handle_x=GlobalAlloc(GHND,sizeof(double)*maxdatenpunkte);
        handle_y=GlobalAlloc(GHND,sizeof(double)*maxdatenpunkte);

   new_x = (double*) GlobalLock(handle_x);
   new_y = (double*) GlobalLock(handle_y);

        for(i=0; i < daten[dataset].datacount; i++)
        {
                if(daten[dataset].yl[i] >= 0.01)
                {
                        new_x[new_datacount]=daten[dataset].xl[i];
                        new_y[new_datacount]=daten[dataset].yl[i];
                        new_datacount++;
                }
        }

        // jetzt umkopieren
```

```
            memcpy(daten[dataset].xl, new_x, sizeof(double)*new_datacount);
            memcpy(daten[dataset].yl, new_y, sizeof(double)*new_datacount);
            daten[dataset].datacount = new_datacount;

            GlobalUnlock(handle_x);
    GlobalFree(handle_x);
            GlobalUnlock(handle_y);
    GlobalFree(handle_y);
}

double getadcoffset(unsigned char kanal)
{
            float TimeUnit=0; // für Zeitfunktionen aus My_Timer.cpp
            LARGE_INTEGER StartTime; // für Zeitfunktionen aus My_Timer.cpp
            int messung = 0;
    double ux=0;

    st.bipolar = daten[0].bipolar;
    st.range = daten[0].adbereich;
    st.kanal = kanal;
            initstruct(&st);

    StartTime=TimeInit(&TimeUnit);
            do
    {
       ux += (double) wandeln(st);
                    messung++;
            } while(TimeRead(StartTime,TimeUnit) <= 100); // 100 ms filtert 50 Hz gut

            // Mittelwert ausrechnen und zurückgeben
    return(ux / messung);
}

int readiubereich(HWND hWnd, HFILE hFile, int dataset)
{
    int zeichenzahl;
    char zeichen, string[30];
            long lesestatus;

            zeichenzahl=0;
            do // bis zum Tab einlesen
            {
                    lesestatus = _hread(hFile, &zeichen ,1); // ein Byte lesen
                    string[zeichenzahl++] = zeichen;
            }
            while((string[zeichenzahl-1]!='\t') && lesestatus == 1); // bei Tab und EOF abbrechen
            if(lesestatus == 0)
            {
                    MessageBox(hWnd, "Datei vermutlich leer!", "Fehler", MB_OK);
       return(FEHLER);
            }
            string[zeichenzahl-1]=0; // Tab mit 0-Byte überschreiben
            // Prüfen ob Kennzahl 9 für I/U-Bereich gelesen wurde
            if(round(atof(string)) != 9) // kein direkter Vergleich mit double !!
            {
                    MessageBox(hWnd, "Datei enthält keinen Messbereich! Nehme 1 µA/V an.", "Fehler",
MB_OK);
                    daten[dataset].iubereich = 1;
                    _llseek(hFile, 0L, 0); // zurück an den Dateianfang gehen, damit 1. Datenpunkt gelesen
wird
                    return(OK); // kein wirklicher Fehler, also normal weiter
            }
            zeichenzahl=0;
```

```
        do // bis zur neuen Zeile einlesen
        {
                lesestatus = _hread(hFile, &zeichen ,1); // ein Byte lesen
                string[zeichenzahl++] = zeichen;
        }
        while(string[zeichenzahl-1]!='\r' && lesestatus == 1); // bei Return und EOF abbrechen
        if(lesestatus == 0)
        {
                MessageBox(hWnd, "Lesefehler beim Messbereich!", "Fehler", MB_OK);
    return(FEHLER);
        };
        string[zeichenzahl-1]=0; // Return mit 0-Byte überschreiben
        daten[dataset].iubereich = atof(string);

        lesestatus = _hread(hFile,&zeichen,1); // noch Zeilenvorschub einlesen
        return(OK);
}


/*****************************************************************************
* Dialogfunkion zum Eingeben der Messoptionen
                *
* lParam ist der Datensatz, für den der Messbereich abgefragt wird          *
*****************************************************************************/

BOOL FAR PASCAL MessDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam)
{
char dummy[100];
static int dataset;

   switch(msg)
   {
                case WM_INITDIALOG :
                        dataset = lParam;
                        sprintf(dummy,"%.1g", daten[dataset].iubereich);
                        SetDlgItemText(hDlg,IDC_IUBEREICH, dummy);
                        sprintf(dummy,"%.1f", messparam.zeit);
                        SetDlgItemText(hDlg,IDC_MESSZEIT, dummy);
                        sprintf(dummy,"%.1g", messparam.filter);
                        SetDlgItemText(hDlg,IDC_XFILTER, dummy);
    break;

                case WM_COMMAND :
      switch(wParam)
      {
        case IDOK:
                                        GetDlgItemText(hDlg, IDC_IUBEREICH, dummy, 99);
                                        messparam.iubereich = atof(dummy);
                                        daten[dataset].iubereich = messparam.iubereich;
                                        GetDlgItemText(hDlg,IDC_MESSZEIT, dummy, 99);
                                        messparam.zeit = atof(dummy);
                                        GetDlgItemText(hDlg, IDC_XFILTER, dummy, 99);
                                        messparam.filter = atof(dummy);
                                        EndDialog(hDlg,TRUE);
          return TRUE;
        break;

        case IDCANCEL :
          EndDialog(hDlg,FALSE);
          return TRUE;
        break;

      }
```

```
      break;

   }

 return(FALSE);
}

/*****************************************************************************
* getawfile ermittelt mit dem Öffnen-Dialog den Dateinamen für die Auswertungsdatei  *
* Dabei wird die Datei NICHT erstellt, sonder nur der Name in aw.dateiname gspeichert *
*****************************************************************************/

void getawfile(HWND hWnd)
{
   static char szFilename[_MAX_FNAME + _MAX_EXT]; // damit bei Cancel nicht das Original überschrieben wird
   int iErg=0;
   OPENFILENAME ofn;

   memset(&ofn, 0, sizeof(OPENFILENAME));
   ofn.lpstrFile = szFilename;
   iErg = FileDialogOpen(hWnd, &ofn, 2);

         // Umkopieren nur, wenn Ok gedrückt wurde
         if(iErg == IDOK) strncpy(aw.dateiname, szFilename, _MAX_FNAME + _MAX_EXT);
   return;
}

/*****************************************************************************
* saveaw() schreibt die in short dataset übergebene Auswertung in aw.dateiname  *
* verwendet nur 32Bit I/O-Routinen                                              *
*****************************************************************************/

void saveaw(HWND hWnd, short dataset)
{
   char zeile[101], einheit[3];
   HANDLE hFile;
         BOOL status;
         unsigned long schreibbytes = 0; // für Windows95 unbedingt erforderlich

   if((hFile = CreateFile(    aw.dateiname, // lpFileName: Dateiname
                              GENERIC_WRITE, // dwDesiredAccess: Zugriffsmodus schreibend
                              0, // dwShareMode: 0 für kein Sharing
                              NULL, // lpSecurityAttributes: NULL heißt Handle nicht vererbbar
                              OPEN_ALWAYS, // dwCreationDisposition: falls vorhanden Öffnen, falls nicht neu erstellen
                              FILE_ATTRIBUTE_NORMAL, // dwFlagsAndAttributes: normal
                              NULL // hTemplateFile: muss für Windows95 NULL sein
                              )) == INVALID_HANDLE_VALUE)
   {
                  MessageBox(hWnd, "Datei konnte nicht angelegt werden!", "Fehler", MB_OK);
      return;
   }
   // An das Dateiende springen
         SetFilePointer(hFile, 0, // 0 Bytes weit springen
                              NULL, // kein Zeiger auf obere 32 Bit, da 64 Bit nicht nötig sind
                              FILE_END); // vom Dateiende aus
```

```
        // Daten schreiben
   switch(view.iuflag)
        {
                case I: strncpy(einheit,"µA",3); break;
                case U: strncpy(einheit,"V",2); break;
        }
        sprintf(zeile,"%g\t%g %s\t%s\r\n", daten[dataset].vol, daten[dataset].ypeak-daten[dataset].yback,
einheit, daten[dataset].dateiname);
        status = WriteFile(hFile, zeile, strlen(zeile), &schreibbytes, NULL);
   if(status == 0)
   {
                MessageBox(hWnd, "Fehler beim Schreiben in die Datei.", "Fehler",
MB_OK|MB_ICONSTOP);
   }

        CloseHandle(hFile);
}
```

11.7.4.4 my_timer.cpp

```
/*********************************************************
* Diese Datei enthält Funktionen für genaue Timer unter
* 1 ms und Task-Prioritäts-Umschaltefunktionen.
* Damit sind Timer und CPU-Unabhängige Delayzeiten
* möglich.
* Die Datei My_Timers.h enthält die für C nötigen
* Deklarationen.
* Siehe auch "Handbuch der PC-Mess- und Steuertechnik"
* S. 68
*********************************************************/

#include "stdafx.h"

/*********************************************************
* Name: RealTime
* Setzt die Task-Priorität auf MAX.
* Dann sind z.B. KEINE Mausaktionen mehr möglich!!
* Abschalten mit NormalTime NICHT vergessen !!
*********************************************************/
void RealTime(void)
{
        SetPriorityClass(GetCurrentProcess(),REALTIME_PRIORITY_CLASS);
}


/*********************************************************
* Name: NormalTime
* setzt die Task-Priorität wieder auf normal. Dann arbeitet
* das Programm wieder normal. Maus, Tastatur usw.
*********************************************************/
void NormalTime(void)
{
        SetPriorityClass(GetCurrentProcess(),NORMAL_PRIORITY_CLASS);
}


/*********************************************************
* Name: TimeInit
* gibt den momentanen Stand des high-resolution Counters
* als Funktionswert zurück und schreibt die Zeitauflösung
* in die Adresse der Float.
* Falls die PC-Hardware keinec o.g. Timer unterstützt,
```

```
* wird 0 als Funktionswert zurückgeliefert.
* Die Float bleibt dann unverändert.
*********************************************************/
LARGE_INTEGER TimeInit(float * TimeUnit)
{
        LARGE_INTEGER f;
        float fl=1000;
        if(!QueryPerformanceFrequency(&f))
                {f.QuadPart=0; return f;}
        *TimeUnit = fl / f.QuadPart;
        QueryPerformanceCounter(&f);
        return f;
}


/*********************************************************
* Name: TimeRead
* gibt die seit dem letzten Aufruf v. TimeInit verstrichene
* Zeit als Real (ms.us) als Funktionswert zurück.
* Die integer-Startzeit wird als large-Int und die Zeitauf-
* lösung als float ms.us (zuvor mit TimeInit ermittelt)
* übergeben.
*********************************************************/
float TimeRead(LARGE_INTEGER StartTime, float TimeUnit)
{
        LARGE_INTEGER Zeit;
        QueryPerformanceCounter(&Zeit);
        if(Zeit.QuadPart==0) return 0;
        return TimeUnit*(Zeit.QuadPart-StartTime.QuadPart);
}

/*********************************************************
* Name: Delay
* Wartet per TimeStart und TimeRead-Funktionen die als
* float übergebene Zeit (Millisekunden.Mikrosekunden)
* Zuvor muss einmal TimeInit zwecks Berechnung der TimeUnit
* und StartTime aufgerufen worden sein.
*********************************************************/
void Delay(float DelayTime, LARGE_INTEGER StartTime,
        float TimeUnit)
{float TimeStart, TimeStop;
 TimeStart=TimeRead(StartTime,TimeUnit);
 TimeStop=TimeStart+DelayTime;
 while(TimeRead(StartTime,TimeUnit) < TimeStop);
}
```

11.7.4.5 small.cpp

```
#include "StdAfx.h"

extern XYTMESSTRUCT xytmessparam; // zum Einstellen des COM-Ports
extern KONTROLLZYKL kontrolle;

/*****************************************************************************
* Dialogfunkion für die Optionen                                             *
*****************************************************************************/

BOOL FAR PASCAL OPTIONENDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM
lParam)
{
char dummy[100];

   switch(msg)
```

```
   {
                    case WM_INITDIALOG :
                                SetDlgItemInt(hDlg,IDC_COMPORT,  xytmessparam.comport,  TRUE);  //
TRUE heißt signed Variable
                                break;

                    case WM_COMMAND :
         switch(wParam)
         {
            case IDOK:
                                                GetDlgItemText(hDlg, IDC_COMPORT, dummy, 99);
                                                xytmessparam.comport = atoi(dummy);
               if(xytmessparam.comport <= 0)
               {
                  MessageBox(hDlg, "Bitte als COM-Port nur ganze positive Zahlen angeben.", "Achtung",
MB_OK|MB_ICONSTOP);
                  return FALSE;
               }
                                                EndDialog(hDlg,TRUE);
               return TRUE;
            break;

            case IDCANCEL :
               EndDialog(hDlg,FALSE);
               return TRUE;
            break;

         }
      break;

   }

 return(FALSE);
}

void writeini(HWND hWnd)
{
   char string[200]; // wird für das Konvertieren der Zahlen in Strings gebraucht

   sprintf(string, "%i", xytmessparam.comport);
   WritePrivateProfileString("Setup", "COM-Port", string, INIFILENAME);
         WritePrivateProfileString("Setup", "TEMPFILE", kontrolle.defaulttempfile, INIFILENAME);
         WritePrivateProfileString("Setup",  "QQTEMPFILE",  kontrolle.defaulttempfileqq,
INIFILENAME);

   // Benutzer das Schreiben der INI-Datei mitteilen
   sprintf(string, "Optionen in\n%s\nim Windowsverzeichnis gespeichert.", INIFILENAME);
   MessageBox(hWnd, string, "Optionen speichern", MB_OK|MB_ICONINFORMATION);
}

void warten(double sek) // warten in Sekunden
{
double TimeUnit;

LARGE_INTEGER startzeit,zeit;

         TimeInithgs(&TimeUnit);

         QueryPerformanceCounter(&startzeit);
         do
         {
                  QueryPerformanceCounter(&zeit);
         }
```

```
        while( startzeit.QuadPart+sek/TimeUnit > zeit.QuadPart);
}

LARGE_INTEGER TimeInithgs(double * TimeUnit) //gibt die Zeit eines Ticks des hires counters in s zurück
{
        LARGE_INTEGER f;

        if(!QueryPerformanceFrequency(&f))
        {
                f.QuadPart=-1; return f; // TimeUnit=-1, wenn kein hires counter vorhanden ist, führt dazu, da die warte() sofort abgebrochen wird
                                                                                // und somit nicht wartet, daher kein warten auf Maschinen, die keinen hires counter haben, mit Glück läuft dann das
                                                                                //Programm doch.
        }

        *TimeUnit =(double) 1/f.QuadPart;
        return f;
}
```

11.7.4.6 stdafx.cpp

```
// stdafx.cpp : Quelltextdatei, die nur die Standard-Includes einbindet
//      Schreiber.pch ist die vorkompilierte Header-Datei
//      stdafx.obj enthält die vorkompilierte Typinformation

#include "stdafx.h"

// ZU ERLEDIGEN: Verweis auf alle zusätzlichen Header-Dateien, die  in STDAFX.H
// und nicht in dieser Datei benötigt werden
```

11.7.4.7 xyrgrafik.cpp

```
#include "StdAfx.h"

extern XYTMESSTRUCT xytmessparam; // aus xytmess.cpp
extern XYTDATENSTRUCT xytdaten; // aus xytmess.cpp
extern HINSTANCE hInst; // aus schreiber.cpp
extern KONTROLLZYKL kontrolle;
extern QQSTRUCT *qqdaten;
extern MESSCONTR messkontrolle;

void fensterloeschen(HDC hPaint,int maxwinx,int maxwiny)
{
        SelectObject(hPaint, GetStockObject(NULL_PEN));
   SelectObject(hPaint, GetStockObject(WHITE_BRUSH));
   Rectangle(hPaint,0,0,maxwinx,maxwiny);
}

void DrawBitmap( HDC hdc, HBITMAP bitmap, short x, short y )
{
BITMAP bitmapbuff;
HDC memorydc;
POINT origin;
POINT size;

   memorydc = CreateCompatibleDC( hdc );
   SelectObject( memorydc, bitmap );
   SetMapMode( memorydc, GetMapMode( hdc ) );
   GetObject( bitmap, sizeof( BITMAP ), (LPSTR) &bitmapbuff );

   origin.x = x;
```

```
   origin.y = y;
   size.x = bitmapbuff.bmWidth;
   size.y = bitmapbuff.bmHeight;

   DPtoLP( hdc, &origin, 1 );
   DPtoLP( memorydc, &size, 1 );

   BitBlt( hdc, origin.x, origin.y, size.x, size.y, memorydc, 0, 0, SRCCOPY);
   DeleteDC( memorydc );
}

double findmaxdouble(double *feld,unsigned long int count) // findet größte Fließkommazahl in einem Feld
{
double dummy;
unsigned long int i;

        dummy=feld[0];
        for(i=0;i<count;i++)
        {
                if(fabs(feld[i])>=dummy)
                        dummy=fabs(feld[i]);
        }
        return(dummy);
}

void drawxyt(HWND hWnd, int flag)
{
HDC hDc;
PAINTSTRUCT ps;

int kanal;
int maxwinx,maxwiny,startx,starty,dx,dy,i,m=1;
RECT Rect;
HBITMAP hUp,hRe;
double du,di,xwertmax,ywertmax,ywertmin,dummy=0,ywertmaxstrom,ywertminstrom;
unsigned long j=0;
unsigned long int k=0,l=0;
char outtext[50];
HPEN hStift[8];

        GetClientRect(hWnd, &Rect);// wie groß ist das Fenster?
        InvalidateRect(hWnd,&Rect,0); // alles neu zeichnen
        maxwinx=Rect.right;
        maxwiny=Rect.bottom;

   if(flag==PAINTMSG)
        {
                hDc=BeginPaint(hWnd, &ps);
        }
   else
        {
      hDc=GetDC(hWnd);
   }
        fensterloeschen(hDc,maxwinx,maxwiny);

        if(kontrolle.graphikstatus==STATUSFENSTER)
        {
                if(xytdaten.datacount != 0)
                {
                        SelectObject(hDc, GetStockObject(BLACK_PEN));
                        SetTextAlign(hDc,TA_LEFT|TA_BOTTOM);

                        sprintf(outtext,"COM-Port: %i",xytmessparam.comport);
```

```
                    TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));

                    sprintf(outtext,"Messzyklus: %.1lf ", (double )kontrolle.anzahl /2);
                    TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));

                    if(kontrolle.anzahl==0) // Formierung laden
                    {
                            sprintf(outtext,"Strom: %.4le A",kontrolle.formlade);
                            TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));
                    }
                    else if(kontrolle.anzahl==1) // Formierung entladen
                    {
                            sprintf(outtext,"Strom: %.4le A",kontrolle.formentlade);
                            TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));
                    }
                    else if((kontrolle.anzahl%2) == 0) // Laden zyklisieren
                    {
                            sprintf(outtext,"Strom: %.4le A",kontrolle.zyklade);
                            TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));
                    }
                    else if((kontrolle.anzahl%2) ==1 ) // Entladen zyklisieren
                    {
                            sprintf(outtext,"Strom: %.4le A",kontrolle.zyklentlade);
                            TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));
                    }

                    sprintf(outtext,"Strom eingelesen: %.4le A",*(xytdaten.xyl[0] +
xytdaten.datacount-1));
                    TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));


                    sprintf(outtext,"Spannung: %.4le V",*(xytdaten.xyl[1] + xytdaten.datacount-1));
                    TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));

                    sprintf(outtext,"Zeit: %.1lf s",*(xytdaten.tl + xytdaten.datacount-1));
                    TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));


                    if(kontrolle.anzahl>=1)
                    {
                            sprintf(outtext,"Zyklendauer: %.1lf s",qqdaten[kontrolle.anzahl-
1].intzeit);
                            TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));

                            sprintf(outtext,"Ladung: %.4le As",qqdaten[kontrolle.anzahl-1].q);
                            TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));

                            sprintf(outtext,"Innenwiderstand: %.4le Ohm",qqdaten[kontrolle.anzahl-
1].r);
                            TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));

                    }


                    if(messkontrolle.cyclefinish==ENDCYCLE)
                    {
                            sprintf(outtext,"Messung wird am Zyklusende angehalten");
                            TextOut(hDc,LABTEXT,OABK*(m++),outtext,strlen(outtext));
                    }

                    EndPaint(hWnd, &ps);
            }
            else // also kein PAINTMSG
```

```
		{
			ReleaseDC(hWnd,hDc);
		}
	}
	else
	{

		hStift[0]=CreatePen(PS_SOLID,0,RGB(0,0,0)); // Einzelne Stifte herrichten
		hStift[1]=CreatePen(PS_SOLID,0,RGB(128,0,0));
		hStift[2]=CreatePen(PS_SOLID,0,RGB(255,0,0));
		hStift[3]=CreatePen(PS_SOLID,0,RGB(255,126,0));
		hStift[4]=CreatePen(PS_SOLID,0,RGB(255,255,0));
		hStift[5]=CreatePen(PS_SOLID,0,RGB(0,255,0));
		hStift[6]=CreatePen(PS_SOLID,0,RGB(0,0,255));
		hStift[7]=CreatePen(PS_SOLID,0,RGB(192,192,192));

		// Bestimmung des maximalen y-Werts
		if(xytdaten.datacount != 0) // es sind also Messdaten vorhanden
		{
			// Bestimmung des maximalen x-Werts in Schritten
			xwertmax = FENSTERSCHRITT * ceil(xytdaten.tl[xytdaten.datacount-
1]/FENSTERSCHRITT);

			ywertmax = -15; // nur mal auf einen kleinen Wert setzen
			ywertmin = 15; // nur mal auf einen großen Wert setzen
			ywertmaxstrom=-4;
			ywertminstrom=4;

			// Bestimmung des Maximums des Messkanalstroms
			kanal=0;
			if(xytmessparam.pga[kanal] != 0) // keine Extremasuche, falls nicht gemessen
			{
				for(j=0;j<xytdaten.datacount-1;j++)
				{

	if(*(xytdaten.xyl[kanal]+j)*xytmessparam.pga[kanal]>=ywertmaxstrom)

ywertmaxstrom=*(xytdaten.xyl[kanal]+j)*xytmessparam.pga[kanal];
				}

				// Bestimmung des Minimum eines Messkanal
				for(j=0;j<xytdaten.datacount-1;j++)
				{

	if(*(xytdaten.xyl[kanal]+j)*xytmessparam.pga[kanal]<=ywertminstrom)

ywertminstrom=*(xytdaten.xyl[kanal]+j)*xytmessparam.pga[kanal];
				}
			} // Ende if pga[kanal] != 0

			// Bestimmung des Maximums eines Messkanal Spannungen
			for(kanal = 1; kanal < MAXKANAL; kanal++)
			{
				if(xytmessparam.pga[kanal] != 0) // keine Extremasuche, falls nicht
gemessen
				{
					for(j=0;j<xytdaten.datacount-1;j++)
					{

	if(*(xytdaten.xyl[kanal]+j)*xytmessparam.pga[kanal]>=ywertmax)

ywertmax=*(xytdaten.xyl[kanal]+j)*xytmessparam.pga[kanal];
					}
```

```
                                        // Bestimmung des Minimums eines Messkanal
                                        for(j=0;j<xytdaten.datacount-1;j++)
                                        {

        if(*(xytdaten.xyl[kanal]+j)*xytmessparam.pga[kanal]<=ywertmin)

ywertmin=*(xytdaten.xyl[kanal]+j)*xytmessparam.pga[kanal];
                                        }
                                } // Ende if pga[kanal] != 0
                        } // Ende Kanal forschleife

                } // Ende if(xytdaten.datacount != 0)
                else // also keine Daten zur Extremabestimmung
                {
                        xwertmax = FENSTERSCHRITT;
                        ywertmax = 4; // einfach mal so gesetzt, vielleicht besser mit pga ausrechnen
                        ywertmin = -4;
                }

                // Pfeile und Linien des Kosy zeichnen
                starty=(int) round((maxwiny-UABK-OABK)+OABK);
                startx=(int) round(LABK);

                hUp=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_UPFEIL));
                hRe=LoadBitmap(hInst,MAKEINTRESOURCE(IDB_LPFEIL));
                SelectObject(hDc, GetStockObject(NULL_PEN));
                SelectObject(hDc, GetStockObject(BLACK_PEN));

                MoveToEx(hDc,0+LABK-BALKENK-1,starty,NULL); // X-Achse
                LineTo(hDc,maxwinx-RABK+10,starty);
                DrawBitmap(hDc,hRe,maxwinx-RABK+10,starty-4); //Pfeil zeichnen

                MoveToEx(hDc,0+LABK,(int)      round(maxwiny-UABK+4),NULL);      //      Y-Achse
Spannungen
                LineTo(hDc,0+LABK,(int) round(0+OABK-14));
                DrawBitmap(hDc,hUp,0+LABK-4,0+OABK-14); // Pfeil zeichnen

                MoveToEx(hDc,maxwinx-RABK,(int)   round(maxwiny-UABK+4),NULL);  //  Y-Achse
Strom
                LineTo(hDc,maxwinx-RABK,(int) round(0+OABK-14));
                DrawBitmap(hDc,hUp,maxwinx-RABK-4,0+OABK-14); // Pfeil zeichnen

                // Zeichnen der Tickmarks Spannungsachse

                dy=(int) round(maxwiny-OABK-UABK)/ANZAHLYTICKS; //Berechnen der Abstände
für kleine Striche
                di=(ywertmax-ywertmin)/ANZAHLYTICKS; // Differenz in y in der aufgetragenen
Einheit

                dx=(int) round((maxwinx-RABK-startx)/ANZAHLXTICKS);
                du=(xwertmax)/ANZAHLXTICKS;// Differenz in x in der aufgetragenen Einheit

                SetTextAlign(hDc,TA_LEFT|TA_TOP); // Beschriftung der Y-Achsesetzen
                sprintf(outtext," U/V ");
                TextOut(hDc,LABK+10,maxwiny-UABK-(ANZAHLYTICKS*dy)-
TEXTYDY,outtext,strlen(outtext));

                SetTextAlign(hDc,TA_CENTER|TA_BOTTOM);
                sprintf(outtext," t/s ");
                TextOut(hDc,startx+dx*ANZAHLXTICKS,starty-TEXTXDY,outtext,strlen(outtext));

                SetTextAlign(hDc,TA_LEFT|TA_TOP);
```

```
		for(i=0;i<=ANZAHLYTICKS;i+=2) // Y-Achse
		{
			MoveToEx(hDc,LABK-BALKENK,(int) (starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK)),NULL);
			LineTo(hDc,LABK+BALKENK+1,(int) (starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK)));

			sprintf(outtext,"%.4lf",(ywertmin+di*i));
			TextOut(hDc,LABK+TEXTYDXK,(int) (starty-(di*i)/(ywertmax-
ywertmin)*(maxwiny-UABK-OABK))-TEXTYDY,outtext,strlen(outtext));
		}

		SetTextAlign(hDc,TA_CENTER|TA_TOP);

		for(i=2;i<=ANZAHLXTICKS;i+=2) // X-Achse
		{
			MoveToEx(hDc,startx+i*dx,starty-BALKENK,NULL);
			LineTo(hDc,startx+i*dx,starty+BALKENK+1);
			sprintf(outtext,"%.0lf",0+i*du);
			TextOut(hDc,startx+i*dx,starty+TEXTXDY,outtext,strlen(outtext));
		}

		SetTextAlign(hDc,TA_LEFT|TA_TOP);

	// Zeichnen der Tickmarks Stromachse

		dy=(int) round(maxwiny-OABK-UABK)/ANZAHLYTICKS; // Berechnen der Abstände
für kleine Striche
		di=(ywertmaxstrom-ywertminstrom)/ANZAHLYTICKS; // Differenz in y in der
aufgetragenen Einheit

		SetTextAlign(hDc,TA_LEFT|TA_TOP); // Beschriftung der Y-Achse setzen
		sprintf(outtext," I/A ");
		TextOut(hDc,maxwinx-RABK-40,maxwiny-UABK-(ANZAHLYTICKS*dy)-
TEXTYDY,outtext,strlen(outtext));

		SetTextAlign(hDc,TA_LEFT|TA_TOP);
		for(i=0;i<=ANZAHLYTICKS;i+=2) // Y-Achse
		{
			MoveToEx(hDc,maxwinx-RABK-BALKENK,(int) (starty-
(di*i)/(ywertmaxstrom-ywertminstrom)*(maxwiny-UABK-OABK)),NULL);
			LineTo(hDc,maxwinx-RABK+BALKENK+1,(int) (starty-
(di*i)/(ywertmaxstrom-ywertminstrom)*(maxwiny-UABK-OABK)));

			sprintf(outtext,"%.4lf",(ywertminstrom+di*i));
			TextOut(hDc,maxwinx-RABK+20,(int) (starty-(di*i)/(ywertmaxstrom-
ywertminstrom)*(maxwiny-UABK-OABK))-TEXTYDY,outtext,strlen(outtext));
		}

		SetTextAlign(hDc,TA_CENTER|TA_TOP);


		// ab hier werden die Messdaten eingezeichnet

		if(flag==PAINTMSG && xytdaten.datacount != 0 ) // Daten ausgeben, alles Zeichen??
zeichnen
		{
			SelectObject(hDc, GetStockObject(BLACK_PEN));

			for(kanal=1; kanal < MAXKANAL; kanal++)
			{

				if(xytmessparam.pga[kanal] != 0) // falls Kanal aus, auch nicht zeichnen
```

```
                                        {
                                                sprintf(outtext,"%le
V",*(xytdaten.xyl[kanal]+xytdaten.datacount-2));
                                                TextOut(hDc,maxwinx-RABK-100,maxwiny-UABK-
(ANZAHLYTICKS*dy)-TEXTYDY,outtext,strlen(outtext));

                                                for(k=1;k<xytdaten.datacount-1;k++)
                                                {
                                                        SelectObject(hDc,hStift[kanal]);
                                                        MoveToEx(hDc,(int)(startx+xytdaten.tl[k-
1]/xwertmax*(maxwinx-RABK-LABK)),
                                                                                (int) round( (starty-
(*(xytdaten.xyl[kanal]+k-1)*xytmessparam.pga[kanal]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))),NULL);
                                                        LineTo(hDc,
(int)(startx+xytdaten.tl[k]/xwertmax*(maxwinx-RABK-LABK)),
                                                                                (int) round( (starty-
(*(xytdaten.xyl[kanal]+k)*xytmessparam.pga[kanal]-ywertmin)/(ywertmax-ywertmin)*(maxwiny-UABK-
OABK))));
                                                } // Datenpunktschleife
                                        } // Ende if(pga != 0)
                                } // Kanalschleife

                                kanal=0; // Strom zeichnen
                                if(xytmessparam.pga[kanal] != 0) // falls Kanal aus, auch nicht zeichnen
                                {
                                        sprintf(outtext,"%le A",*(xytdaten.xyl[kanal]+xytdaten.datacount-2));
                                        TextOut(hDc,maxwinx-RABK-100,maxwiny-UABK-
(ANZAHLYTICKS*dy)+TEXTYDY*6,outtext,strlen(outtext));

                                        for(k=1;k<xytdaten.datacount-1;k++)
                                        {
                                                SelectObject(hDc,hStift[kanal]);
                                                MoveToEx(hDc,(int)(startx+xytdaten.tl[k-
1]/xwertmax*(maxwinx-RABK-LABK)),
                                                                        (int) round( (starty-
(*(xytdaten.xyl[kanal]+k-1)*xytmessparam.pga[kanal]-ywertminstrom)/(ywertmaxstrom-
ywertminstrom)*(maxwiny-UABK-OABK))),NULL);
                                                LineTo(hDc,    (int)(startx+xytdaten.tl[k]/xwertmax*(maxwinx-
RABK-LABK)),
                                                                        (int) round( (starty-
(*(xytdaten.xyl[kanal]+k)*xytmessparam.pga[kanal]-ywertminstrom)/(ywertmaxstrom-
ywertminstrom)*(maxwiny-UABK-OABK))));

                                                } // Datenpunktschleife
                                } // Ende if(pga != 0)

                                SelectObject(hDc, GetStockObject(BLACK_PEN));

                                DeleteObject(hUp);
                                DeleteObject(hRe);

                                for(j=0;j<8;j++) // Stifte wieder aufräumen
                                        DeleteObject(hStift[j]);

                                EndPaint(hWnd, &ps);
                        }

                        else // also kein PAINTMSG
                        {
                                DeleteObject(hUp);
                                DeleteObject(hRe);
                                for(j=0;j<8;j++) // Stifte wieder aufräumen
```

```
                        DeleteObject(hStift[j]);
                        ReleaseDC(hWnd,hDc);
                }
        }
}
```

11.7.4.8 xytmess.cpp

```
#include "StdAfx.h"

XYTMESSTRUCT xytmessparam; // für XYt-Schreiber
XYTDATENSTRUCT xytdaten;
KONTROLLZYKL kontrolle;
extern HINSTANCE hInst; // aus schreiber.cpp
extern HWND hWnd; // aus schreiber.cpp
QQSTRUCT *qqdaten;
HGLOBAL handleqqdata;
MESSCONTR messkontrolle;

void xytglobalinit(HWND hWnd)
{
   char string[30]; // wird für das Konvertieren der eingelesenen Strings in Zahlen gebraucht

        xytmessparam.pga[0] = 1;
        xytmessparam.pga[1] = 1;
        xytmessparam.pga[2] = 0;
        xytmessparam.pga[3] = 0;
        xytmessparam.pga[4] = 0;
        xytmessparam.pga[5] = 0;
        xytmessparam.pga[6] = 0;

        xytmessparam.messzeit = 1000000; // in s
        xytmessparam.intzeit = 1000; // 40 ms = 2 Perioden Netzbrumm

        kontrolle.maxanzahl=1000;
        kontrolle.formcutoffoben=4.2;
        kontrolle.formcutoffunten=3.0;
        kontrolle.zyklcutoffoben=4.2;
        kontrolle.zyklcutoffunten=3.0;
        kontrolle.formlade=4e-3;
        kontrolle.formentlade=-4e-3;
        kontrolle.zyklade=40e-3;
        kontrolle.zyklentlade=-40e-3;

        xytdaten.datacount = 0; // ist das nötig
//      timeBeginPeriod(2);
   GetPrivateProfileString("Setup", "COM-Port", "1", string, 29, INIFILENAME);
        xytmessparam.comport=atoi(string);

        GetPrivateProfileString("Setup", "TEMPFILE", "C:\\tempfile.txt", kontrolle.defaulttempfile, 29,
INIFILENAME);
        GetPrivateProfileString("Setup", "QQTEMPFILE",
"C:\\tempfileqq.txt",kontrolle.defaulttempfileqq, 29, INIFILENAME);
}

int getxytspeicher(HWND hWnd)
{
   int i;

        if(xytdaten.datacount != 0) cleanupxy(); // falls vorher schon etwas allokiert ist, freigeben

        xytdaten.tHandle=GlobalAlloc(GHND,sizeof(double)*xytmessparam.maxpunkte+1);
        xytdaten.tl = (double*) GlobalLock(xytdaten.tHandle);
```

```
//        xytdaten.tl = (double*) malloc(xytmessparam.maxpunkte*sizeof(double));
          for(i=0; i<MAXKANAL; i++)
          {
                    xytdaten.xyHandle[i]=GlobalAlloc(GHND,sizeof(double) * xytmessparam.maxpunkte+1);
                    xytdaten.xyl[i] = (double*) GlobalLock(xytdaten.xyHandle[i]);

//                  xytdaten.xyl[i]=(double*) malloc((sizeof(double) * xytmessparam.maxpunkte)+1);
 }

          xytdaten.datacount = 0; // Datensatz hat momentan 0 Punkte

          return(0);
}


/*****************************************************************************
* Dialogfunkion zum Eingeben der Messoptionen XYt-Schreiber
          *
* lParam ist der Datensatz, für den der Messbereich abgefragt wird          *
*****************************************************************************/

BOOL FAR PASCAL XYtMessDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam)
{
          char dummy[100];
          unsigned long intzeit;

   switch(msg)
   {
                    case WM_INITDIALOG :
         SendDlgItemMessage(hDlg,IDC_KANAL11, BM_SETCHECK, 0, 0);
         SendDlgItemMessage(hDlg,IDC_KANAL110, BM_SETCHECK, 0, 0);
         SendDlgItemMessage(hDlg,IDC_KANAL1100, BM_SETCHECK, 0, 0);
         SendDlgItemMessage(hDlg,IDC_KANAL11000, BM_SETCHECK, 0, 0);
         SendDlgItemMessage(hDlg,IDC_KANAL1NIX, BM_SETCHECK, 0, 0);

         if(xytmessparam.pga[0]==1)
              SendDlgItemMessage(hDlg,IDC_KANAL11, BM_SETCHECK, 1, 0);
         else if(xytmessparam.pga[0]==10)
               SendDlgItemMessage(hDlg,IDC_KANAL110, BM_SETCHECK, 1, 0);
         else if(xytmessparam.pga[0]==100)
               SendDlgItemMessage(hDlg,IDC_KANAL1100, BM_SETCHECK, 1, 0);
         else if(xytmessparam.pga[0]==1000)
               SendDlgItemMessage(hDlg,IDC_KANAL11000, BM_SETCHECK, 1, 0);
         else if(xytmessparam.pga[0]==0)
               SendDlgItemMessage(hDlg,IDC_KANAL1NIX, BM_SETCHECK, 1, 0);

         SendDlgItemMessage(hDlg,IDC_KANAL21, BM_SETCHECK, 0, 0);
         SendDlgItemMessage(hDlg,IDC_KANAL210, BM_SETCHECK, 0, 0);
         SendDlgItemMessage(hDlg,IDC_KANAL2100, BM_SETCHECK, 0, 0);
         SendDlgItemMessage(hDlg,IDC_KANAL21000, BM_SETCHECK, 0, 0);
         SendDlgItemMessage(hDlg,IDC_KANAL2NIX, BM_SETCHECK, 0, 0);

         if(xytmessparam.pga[1]==1)
              SendDlgItemMessage(hDlg,IDC_KANAL21, BM_SETCHECK, 1, 0);
         else if(xytmessparam.pga[1]==10)
               SendDlgItemMessage(hDlg,IDC_KANAL210, BM_SETCHECK, 1, 0);
         else if(xytmessparam.pga[1]==100)
               SendDlgItemMessage(hDlg,IDC_KANAL2100, BM_SETCHECK, 1, 0);
         else if(xytmessparam.pga[1]==1000)
               SendDlgItemMessage(hDlg,IDC_KANAL21000, BM_SETCHECK, 1, 0);
         else if(xytmessparam.pga[1]==0)
               SendDlgItemMessage(hDlg,IDC_KANAL2NIX, BM_SETCHECK, 1, 0);
```

```
                        sprintf(dummy,"%lu", xytmessparam.messzeit);
                        SetDlgItemText(hDlg,IDC_MAXMESSDAUER, dummy);
                        sprintf(dummy,"%lu", xytmessparam.intzeit);
                        SetDlgItemText(hDlg,IDC_INTZEIT, dummy);

                        sprintf(dummy,"%.0lf", (double )kontrolle.maxanzahl /2);
                        SetDlgItemText(hDlg,IDC_ANZAHL, dummy);

                        sprintf(dummy,"%.3le", kontrolle.formlade);
                        SetDlgItemText(hDlg,IDC_FORMLADE, dummy);

                        sprintf(dummy,"%.3le", kontrolle.formentlade);
                        SetDlgItemText(hDlg,IDC_FORMENTLADE, dummy);

                        sprintf(dummy,"%.3le", kontrolle.zyklade);
                        SetDlgItemText(hDlg,IDC_ZYKLADE, dummy);

                        sprintf(dummy,"%.3le", kontrolle.zyklentlade);
                        SetDlgItemText(hDlg,IDC_ZYKENTLADE, dummy);

                        sprintf(dummy,"%.3lf", kontrolle.formcutoffoben);
                        SetDlgItemText(hDlg,IDC_FORMUOBEN, dummy);

                        sprintf(dummy,"%.3lf", kontrolle.formcutoffunten);
                        SetDlgItemText(hDlg,IDC_FORMUUNTEN, dummy);

                        sprintf(dummy,"%.3lf", kontrolle.zyklcutoffoben);
                        SetDlgItemText(hDlg,IDC_ZYKLUOBEN, dummy);

                        sprintf(dummy,"%.3lf", kontrolle.zyklcutoffunten);
                        SetDlgItemText(hDlg,IDC_ZYKLUUNTEN, dummy);

                        SetDlgItemInt(hDlg,IDC_COMPORT, xytmessparam.comport, TRUE); //
TRUE heißt signed Variable

      break;

                case WM_COMMAND :
        switch(wParam)
        {
          case IDOK:
            if(IsDlgButtonChecked(hDlg,IDC_KANAL11)==TRUE)
                  xytmessparam.pga[0]=1;
            else if(IsDlgButtonChecked(hDlg,IDC_KANAL110)==TRUE)
                  xytmessparam.pga[0]=10;
            else if(IsDlgButtonChecked(hDlg,IDC_KANAL1100)==TRUE)
                  xytmessparam.pga[0]=100;
            else if(IsDlgButtonChecked(hDlg,IDC_KANAL11000)==TRUE)
                  xytmessparam.pga[0]=1000;
            else if(IsDlgButtonChecked(hDlg,IDC_KANAL1NIX)==TRUE)
                  xytmessparam.pga[0]=0;

            if(IsDlgButtonChecked(hDlg,IDC_KANAL21)==TRUE)
                  xytmessparam.pga[1]=1;
            else if(IsDlgButtonChecked(hDlg,IDC_KANAL210)==TRUE)
                  xytmessparam.pga[1]=10;
            else if(IsDlgButtonChecked(hDlg,IDC_KANAL2100)==TRUE)
                  xytmessparam.pga[1]=100;
            else if(IsDlgButtonChecked(hDlg,IDC_KANAL21000)==TRUE)
                  xytmessparam.pga[1]=1000;
            else if(IsDlgButtonChecked(hDlg,IDC_KANAL2NIX)==TRUE)
                  xytmessparam.pga[1]=0;
```

```
                                        GetDlgItemText(hDlg,IDC_MAXMESSDAUER, dummy, 99);
                                        xytmessparam.messzeit = atol(dummy);

                                        GetDlgItemText(hDlg, IDC_INTZEIT, dummy, 99);
                                        // erst einlesen, was der Benutzer will
                                        intzeit = atol(dummy);
                                        // jetzt auf Vielfache von 40 ms quanteln
                                        xytmessparam.intzeit = 40 * (int) floor(intzeit / 40.0);
                                        // jetzt untere Grenze beachten: unter 40 ms geht nicht
                                        if(xytmessparam.intzeit < 40) xytmessparam.intzeit = 40;
                                        // obere Grenze für Controller 40e6 s
                                        if(xytmessparam.intzeit >= 40000000) xytmessparam.intzeit =
40000000;

                                        GetDlgItemText(hDlg, IDC_ANZAHL, dummy, 99);
                                        sscanf(dummy,"%lu", &kontrolle.maxanzahl);
                                        kontrolle.maxanzahl*=2; // auf Halbzyklen umrechnen

                                        GetDlgItemText(hDlg,IDC_FORMLADE, dummy, 99);
                                        sscanf(dummy,"%le", &kontrolle.formlade);

                                        GetDlgItemText(hDlg,IDC_FORMENTLADE, dummy, 99);
                                        sscanf(dummy,"%le",&kontrolle.formentlade);


                                        GetDlgItemText(hDlg,IDC_ZYKLADE, dummy, 99);
                                        sscanf(dummy,"%le",&kontrolle.zyklade);


                                        GetDlgItemText(hDlg,IDC_ZYKENTLADE, dummy, 99);
                                        sscanf(dummy,"%le",&kontrolle.zyklentlade);


                                        GetDlgItemText(hDlg,IDC_FORMUOBEN, dummy, 99);
                                        sscanf(dummy,"%lf",&kontrolle.formcutoffoben);


                                        GetDlgItemText(hDlg,IDC_FORMUUNTEN, dummy, 99);
                                        sscanf(dummy,"%lf",&kontrolle.formcutoffunten);


                                        GetDlgItemText(hDlg,IDC_ZYKLUOBEN, dummy, 99);
                                        sscanf(dummy,"%lf",&kontrolle.zyklcutoffoben);


                                        GetDlgItemText(hDlg,IDC_ZYKLUUNTEN, dummy, 99);
                                        sscanf(dummy,"%lf",&kontrolle.zyklcutoffunten);

                                        GetDlgItemText(hDlg, IDC_COMPORT, dummy, 99);
                                        xytmessparam.comport = atoi(dummy);
                if(xytmessparam.comport <= 0)
                {
                  MessageBox(hDlg, "Bitte als COM-Port nur ganze positive Zahlen angeben.", "Achtung",
MB_OK|MB_ICONSTOP);
                  return FALSE;
                }
                                        EndDialog(hDlg,TRUE);

                                        EndDialog(hDlg,TRUE);
                return TRUE;
              break;
```

```
            case IDCANCEL :
               EndDialog(hDlg,FALSE);
               return TRUE;
            break;

         }
      break;

   }

 return(FALSE);
}

/* ********************************************************************************************
*    startet Messung                                                                          *
**********************************************************************************************/

void startxytmess(HWND hWnd)
{
unsigned long countout; // für Writefile
char string[200];
int kanal;
char dummy[10];

        // Messparameter abfragen
        if(DialogBox(hInst, (LPCTSTR)IDD_STARTMESSXY, hWnd, (DLGPROC)
XYtMessDialog)==FALSE)
        {
                return; // abbrechen
        }
        xytmessparam.maxpunkte = xytmessparam.messzeit * 1000 / xytmessparam.intzeit;
        // messzeit von s in ms umrechnen !

        if(getxytspeicher(hWnd) == -1) return; // Abbruch, falls Allokieren nicht geglückt ist

        // Schnittstelle aufmachen
        comopen();
        if(checkschreiber() == FALSE)
        {
           MessageBox(hWnd, "Schreiber antwortet nicht!", "Fehler", MB_OK|MB_ICONSTOP);
                return;
        }

        strcpy(kontrolle.tempfile,kontrolle.defaulttempfile);
        kontrolle.tempfile[strlen(kontrolle.tempfile)-4]=0;
        sprintf(dummy,"comm_%i.txt",xytmessparam.comport);
        strcat(kontrolle.tempfile,dummy);

        strcpy(kontrolle.tempfileqq,kontrolle.defaulttempfileqq);
        kontrolle.tempfileqq[strlen(kontrolle.tempfileqq)-4]=0;
        sprintf(dummy,"comm_%i.txt",xytmessparam.comport);
        strcat(kontrolle.tempfileqq,dummy);

        // Kontrolle.tempfileqq

        messkontrolle.cyclefinish=0; // Menuekontrolle zurücksetzen
        messkontrolle.nextcycle=0;
        messkontrolle.startstop=0;
        kontrolle.graphikstatus=STATUSFENSTER;

        kontrolle.zeitkorr=0;

        EnableMenuItem(GetMenu(hWnd),IDM_MESSUNG_XYT,MF_GRAYED); // Menu ausschalten
```

```
handleqqdata=GlobalAlloc(GHND,sizeof(QQSTRUCT) * kontrolle.maxanzahl+1);
qqdaten = (QQSTRUCT*)  GlobalLock(handleqqdata);

// Integrationszeit an Wandler schicken: nur die Vielfachen von 40 ms wissen
sprintf(string, "hgiz%lu\r\n", xytmessparam.intzeit / 40);
countout = strlen(string);
WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);

// PGA-Info an Wandler schicken
for(kanal = 0; kanal < MAXKANAL; kanal++)
{
        sprintf(string, "hgpa%i%i\r\n", kanal, xytmessparam.pga[kanal]);
        countout = strlen(string);
        WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);
}
kontrolle.anzahl=0;
// Offset korrigieren lassen
sprintf(string, "hgof\r\n");
countout = strlen(string);
WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);

// Strombereich einstellen
if(fabs(kontrolle.formlade)>=STROMSCHWELLE)
{
        kontrolle.rstrom=RSTROMGROSS;
        sprintf(string, "hgsg\r\n");
}
else
{
        kontrolle.rstrom=RSTROMKLEIN;
        sprintf(string, "hgsk\r\n");
}
countout = strlen(string);
WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);

// Temfiles löschen
cleartempfile(hWnd,kontrolle.tempfile);
cleartempfile(hWnd,kontrolle.tempfileqq);

// Stromeinstellen, Laden mit Formierstrom
sprintf(string, "hgtu%lf\r\n",kontrolle.formlade*1e6*kontrolle.rstrom);
countout = strlen(string);
WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);

// Zelle ein
sprintf(string, "hgz1\r\n");
countout = strlen(string);
WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);
kontrolle.anzahl=0;

// warten
warten(0.5);

// erste Messung anfordern
sprintf(string, "hgmm\r\n");
countout = strlen(string);
WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);

// Zeitpunkt Messbeginn feststellen
xytmessparam.StartTime=TimeInit(&xytmessparam.TimeUnit);

// Timer starten
```

```
	xytmessparam.intzeitaufl = xytmessparam.intzeit / 10;
	if(xytmessparam.intzeitaufl > 50) xytmessparam.intzeitaufl = 50; // also 10% Fehler bis 0.5s, darüber 50 ms fest
	timeSetEvent(xytmessparam.intzeit, xytmessparam.intzeitaufl ,timerroutine,0,TIME_ONESHOT); // Timer an

} /* ********************** Ende startxytmess ******************************/


void comopen(void)
{
	DCB dcb;
	char pcCommPort[6];
//	COMMTIMEOUTS commtimeout; besser in Checkschreiber

	sprintf(pcCommPort,"COM%i",xytmessparam.comport);
	xytmessparam.Comhandle = CreateFile( pcCommPort, GENERIC_READ | GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0, NULL ); // RS232 für ASL

	dcb.DCBlength=sizeof(DCB);
	dcb.BaudRate = CBR_19200;     // set the baud rate
	dcb.fBinary=1;                                  // binary mode, no EOF check
	dcb.fParity=0;               // enable parity checking
	dcb.fOutxCtsFlow=0;                     // CTS output flow control
	dcb.fOutxDsrFlow=0;                             // DSR output flow control
	dcb.fDtrControl=1;                             // DTR flow control type
	dcb.fDsrSensitivity=0;       // DSR sensitivity
	dcb.fTXContinueOnXoff=1; // XOFF continues Tx
	dcb.fOutX=0;            // XON/XOFF out flow control
	dcb.fInX=0;             // XON/XOFF in flow control
	dcb.fErrorChar=0;       // enable error replacement
	dcb.fNull=0;            // enable null stripping
	dcb.fRtsControl=1;       // RTS flow control
	dcb.fAbortOnError=0;     // abort reads/writes on error
	dcb.XonLim=2048;              // transmit XON threshold
	dcb.XoffLim=512;              // transmit XOFF threshold
	dcb.ByteSize=8;             // number of bits/byte, 4-8
	dcb.Parity=NOPARITY;               // 0-4=no,odd,even,mark,space
	dcb.StopBits=TWOSTOPBITS;             // 0,1,2 = 1, 1.5, 2
	dcb.XonChar=17;               // Tx and Rx XON character
	dcb.XoffChar=19;              // Tx and Rx XOFF character
	dcb.ErrorChar=0;             // error replacement character
	dcb.EofChar=0;               // end of input character
	dcb.EvtChar=0;               // received event character

	SetCommState(xytmessparam.Comhandle, &dcb);
	PurgeComm(xytmessparam.Comhandle,PURGE_TXCLEAR&&PURGE_RXCLEAR);  //  Puffer leeren
//	SetCommMask(xytmessparam.Comhandle, EV_RXCHAR); // Auf Zeichenempfang achten
//	GetCommTimeouts(xytmessparam.Comhandle, &commtimeout);
//   commtimeout.ReadTotalTimeoutMultiplier = 100;
//   commtimeout.ReadTotalTimeoutConstant = 100;
//	SetCommTimeouts(xytmessparam.Comhandle, &commtimeout);
}

int checkschreiber(void)
{
	char datain[20],dummy[2];
	unsigned long countout = 6, countin = 11;
	int i=0, versuch;
	COMMTIMEOUTS commtimeout;
```

```
        // Timeout festlegen, damit Programm nicht hängt, falls kein Schreiber da ist
        GetCommTimeouts(xytmessparam.Comhandle, &commtimeout);
//    commtimeout.ReadTotalTimeoutMultiplier = 100;
   commtimeout.ReadTotalTimeoutConstant = 100;
        SetCommTimeouts(xytmessparam.Comhandle, &commtimeout);

        for(versuch = 1; versuch <= 2; versuch++)
        {
                PurgeComm(xytmessparam.Comhandle,PURGE_TXCLEAR&&PURGE_RXCLEAR);
// zur Sicherheit Puffer leeren
                WriteFile(xytmessparam.Comhandle,"hgts\r\n",countout,&countout,NULL);
                // Modem 4 Zeichen: "AT\r\n"
                i=0;
                do
                {
                        countin=1;
                        ReadFile(xytmessparam.Comhandle,dummy,countin,&countin,NULL);
                        datain[i++]=dummy[0];
                        if(i>12) break;
                }
                while(dummy[0]!='\n');
                if(strncmp("test\r\n", datain, 6) == 0)
                {
                   commtimeout.ReadTotalTimeoutConstant = 0;
                        SetCommTimeouts(xytmessparam.Comhandle, &commtimeout);
                        return(TRUE); // Schreiber gefunden
                }
                // Modem 4 Zeichen: AT\r\n
        }
        // Schleife ist ohne Erfolg 2x durchgelaufen, also kein Schreiber vorhanden
   commtimeout.ReadTotalTimeoutConstant = 0;
        SetCommTimeouts(xytmessparam.Comhandle, &commtimeout);
        return(FALSE);
}

void getmessdaten(HWND hWnd)
{
char datain[201]="0.05 1.05 2.05 3.05 4.05 5.05 6.05\r\n" ,dummy[2];
unsigned long countout = 6, countin = 11;
int i;
static double letztezeit=0;
char string[200];
double uvor=0,unach=0;
char ausgabetext[50];

        xytdaten.tl[xytdaten.datacount]  =  TimeRead(xytmessparam.StartTime,xytmessparam.TimeUnit)  /
1000 - kontrolle.zeitkorr;
        // Timeread rechnet selbst die Differenz zur StartTime aus, von ms in s umrechnen
        // hier die alten Messdaten abholen

        i=0;
        do
        {
                countin=1;
                ReadFile(xytmessparam.Comhandle,dummy,countin,&countin,NULL);
                // abfragen, ob Readfile Fehler = 0 zurückliefert ?
                datain[i++]=dummy[0];
                if(i>200) break;
        }
        while(dummy[0]!='\n'); // bis Datenzeile komplett
        PurgeComm(xytmessparam.Comhandle,PURGE_RXCLEAR);  // abgeholte Daten aus Puffer
löschen
        // eingelesene Zeile parsen
```

```
//	test = xytdaten.xyl[0] + xytdaten.datacount;
	// wird in der Reihenfolge eingelesen: Spannung 1, Spannung 2, Strom, Batteriespannung, AUX,
	// Daten als U[V] Kanal 0 Strom in A, Kanal 1 Spannung an der Batterie in V,  Kanal 2 AUX Spannung, Kanal 3 Spannung 1, Kanal 4 Spannung2
	sscanf(datain, "%le %le %le %le %le %le %le",
				xytdaten.xyl[3] + xytdaten.datacount, // Spannung 1
				xytdaten.xyl[4] + xytdaten.datacount, // Spannung 2
				xytdaten.xyl[0] + xytdaten.datacount, // Strom
				xytdaten.xyl[1] + xytdaten.datacount, // Batteriespannung
				xytdaten.xyl[2] + xytdaten.datacount, // Auxspannung
				xytdaten.xyl[5] + xytdaten.datacount,
				xytdaten.xyl[6] + xytdaten.datacount);

	// Achtung: double* + 1 addiert 1* sizeof(double) !!!!


	if(kontrolle.anzahl >= kontrolle.maxanzahl|| xytdaten.datacount >= xytmessparam.maxpunkte-1|| xytdaten.tl[xytdaten.datacount]>=xytmessparam.messzeit || (messkontrolle.startstop == MESSENDE)) // Messdauer überschritten oder Messung abbrechen
	{
		// Zelle aus
		sprintf(string, "hgz0\r\n");
		countout = strlen(string);
		WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);

		// Strom 0
		sprintf(string, "hgtu0\r\n");
		countout = strlen(string);
		WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);

		CloseHandle(xytmessparam.Comhandle); // Schnittstelle wieder zu
		EnableMenuItem(GetMenu(hWnd),IDM_MESSUNG_XYT,MF_ENABLED); // Menu einschalten
		savexyt();
	}
	else // also weiter messen
	{

		if(kontrolle.anzahl==0) // Formierung laden
		{
			if(*(xytdaten.xyl[1] + xytdaten.datacount)>=kontrolle.formcutoffoben || (messkontrolle.nextcycle==NEXTCYCLE)) // Ladespannung erreicht
			{
				qqdaten[0].intzeit=*(xytdaten.tl + xytdaten.datacount); // Zyklendauer berechnen
				letztezeit=*(xytdaten.tl + xytdaten.datacount);
				qqdaten[0].q=qqdaten[0].intzeit*kontrolle.formlade;	// Ladung berechnen

				// Integrationszeit auf 40mS
				sprintf(string, "hgiz%lu\r\n", 1);
				countout = strlen(string);
				WriteFile(xytmessparam.Comhandle, string, countout, &countout, NULL);

				// Spannung für Innenwiderstand bestimmen
				uvor=spannunginnenwiderstand();

				// Strom einstellen, Entladen mit Formierstrom
				// Strombereich einstellen
				sprintf(string, "hgtu%lf\r\n",0); // Strom kurz auschalten
				countout = strlen(string);
```

```
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                if(messkontrolle.cyclefinish==ENDCYCLE)
                                {
                                        // Zelle aus
                                        sprintf(string, "hgz0\r\n");
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout,
&countout, NULL);
                                        sprintf(ausgabetext,"Com: %i",xytmessparam.comport);
                                        MessageBox(hWnd, "Messung fortsetzen", ausgabetext,
MB_OK|MB_ICONQUESTION);
                                        // Zelle an
                                        sprintf(string, "hgz1\r\n");
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout,
&countout, NULL);


        kontrolle.zeitkorr=TimeRead(xytmessparam.StartTime,xytmessparam.TimeUnit) / 1000 -
*(xytdaten.tl + xytdaten.datacount);
                                        messkontrolle.cyclefinish=0;
                                }

                                if(fabs(kontrolle.formentlade)>=STROMSCHWELLE)
                                {
                                        kontrolle.rstrom=RSTROMGROSS;
                                        sprintf(string, "hgsg\r\n");
                                }
                                else
                                {
                                        kontrolle.rstrom=RSTROMKLEIN;
                                        sprintf(string, "hgsk\r\n");
                                }
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                sprintf(string,
"hgtu%lf\r\n",kontrolle.formentlade*1e6*kontrolle.rstrom);
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                // Spannung für Innenwiderstand bestimmen
                                unach=spannunginnenwiderstand();

                                qqdaten[0].r=fabs(uvor-unach)/fabs(kontrolle.formlade-
kontrolle.formentlade); // Innenwiderstand
                                qqdaten[0].i=kontrolle.formlade;

                                        // Integrationszeit wieder zurücksetzen
                                sprintf(string, "hgiz%lu\r\n", xytmessparam.intzeit / 40);
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                saveuntermessungqq(hWnd,kontrolle.anzahl); // QQ-Tempfile
speichern
                                kontrolle.anzahl++;
                                messkontrolle.nextcycle=0;
```

```
                        }
                }
                else if(kontrolle.anzahl==1) // Formierung entladen
                {
                        if(*(xytdaten.xyl[1] + xytdaten.datacount)<=kontrolle.formcutoffunten ||
(messkontrolle.nextcycle==NEXTCYCLE)) // Entladespannung erreicht
                        {
                                qqdaten[1].intzeit=*(xytdaten.tl + xytdaten.datacount) - letztezeit;//
Zyklendauer berechnen
                                qqdaten[1].q=qqdaten[1].intzeit*kontrolle.formentlade; // Ladung
berechnen
                                letztezeit=*(xytdaten.tl + xytdaten.datacount);

                                // Integrationszeit auf 40mS
                                sprintf(string, "hgiz%lu\r\n", 1);
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                // Spannung für Innenwiderstand bestimmen
                                uvor=spannunginnenwiderstand();

                                // Strombereich einstellen
                                sprintf(string, "hgtu%lf\r\n",0); // Strom kurz auschalten
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                if(messkontrolle.cyclefinish==ENDCYCLE)
                                {
                                        // Zelle aus
                                        sprintf(string, "hgz0\r\n");
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout,
&countout, NULL);

                                        sprintf(ausgabetext,"Com: %i",xytmessparam.comport);
                                        MessageBox(hWnd, "Messung fortsetzen", ausgabetext,
MB_OK|MB_ICONQUESTION);

                                        // Zelle an
                                        sprintf(string, "hgz1\r\n");
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout,
&countout, NULL);


        kontrolle.zeitkorr=TimeRead(xytmessparam.StartTime,xytmessparam.TimeUnit) / 1000 -
*(xytdaten.tl + xytdaten.datacount);
                                        messkontrolle.cyclefinish=0;
                                }

                                if(fabs(kontrolle.zyklade)>=STROMSCHWELLE)
                                {
                                        kontrolle.rstrom=RSTROMGROSS;
                                        sprintf(string, "hgsg\r\n");
                                }
                                else
                                {
                                        kontrolle.rstrom=RSTROMKLEIN;
                                        sprintf(string, "hgsk\r\n");
                                }
                                countout = strlen(string);
```

```
                                        WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                        // Strom einstellen, Laden mit Ladestrom
                                        sprintf(string, "hgtu%lf\r\n",kontrolle.zyklade*1e6*kontrolle.rstrom);
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                        // Spannung für Innenwiderstand bestimmen
                                        unach=spannunginnenwiderstand();

                                        qqdaten[1].r=fabs(uvor-unach)/fabs(kontrolle.formentlade-
kontrolle.zyklade); // Innenwiderstand
                                        qqdaten[1].i=kontrolle.formentlade;

                                                // Integrationszeit wieder zurücksetzen
                                        sprintf(string, "hgiz%lu\r\n", xytmessparam.intzeit / 40);
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                        saveuntermessungqq(hWnd,kontrolle.anzahl); // QQ-Tempfile
speichern
                                        kontrolle.anzahl++;

                                        messkontrolle.nextcycle=0;
                                }
                        }
                        else if((kontrolle.anzahl%2) == 0) // Laden zyklisieren
                        {
                                if(*(xytdaten.xyl[1] + xytdaten.datacount)>=kontrolle.zyklcutoffoben ||
(messkontrolle.nextcycle==NEXTCYCLE)) // Ladespannung erreicht
                                {
                                        qqdaten[kontrolle.anzahl].intzeit=*(xytdaten.tl + xytdaten.datacount)-
letztezeit;// Zyklendauer berechnen
                                        letztezeit=*(xytdaten.tl + xytdaten.datacount);

        qqdaten[kontrolle.anzahl].q=qqdaten[kontrolle.anzahl].intzeit*kontrolle.zyklade; // Ladung
berechnen

                                        // Integrationszeit auf 40mS
                                        sprintf(string, "hgiz%lu\r\n", 1);
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                        // Spannung für Innenwiderstand bestimmen
                                        uvor=spannunginnenwiderstand();

                                        // Strombereich einstellen
                                        sprintf(string, "hgtu%lf\r\n",0); // Strom kurz auschalten
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                        if(messkontrolle.cyclefinish==ENDCYCLE)
                                        {
                                                // Zelle aus
                                                sprintf(string, "hgz0\r\n");
                                                countout = strlen(string);
                                                WriteFile(xytmessparam.Comhandle, string, countout,
&countout, NULL);
```

```
                                        sprintf(ausgabetext,"Com: %i",xytmessparam.comport);
                                        MessageBox(hWnd, "Messung fortsetzen", ausgabetext,
MB_OK|MB_ICONQUESTION);

                                        // Zelle an
                                        sprintf(string, "hgz1\r\n");
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout,
&countout, NULL);


        kontrolle.zeitkorr=TimeRead(xytmessparam.StartTime,xytmessparam.TimeUnit) / 1000 -
*(xytdaten.tl + xytdaten.datacount);
                                        messkontrolle.cyclefinish=0;
                                }

                                if(fabs(kontrolle.zyklentlade)>=STROMSCHWELLE)
                                {
                                        kontrolle.rstrom=RSTROMGROSS;
                                        sprintf(string, "hgsg\r\n");
                                }
                                else
                                {
                                        kontrolle.rstrom=RSTROMKLEIN;
                                        sprintf(string, "hgsk\r\n");
                                }
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                // Strom einstellen, Entladen mit Ladestrom
                                sprintf(string,
"hgtu%lf\r\n",kontrolle.zyklentlade*1e6*kontrolle.rstrom);
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                // Spannung für Innenwiderstand bestimmen
                                unach=spannunginnenwiderstand();

                                qqdaten[kontrolle.anzahl].r=fabs(uvor-unach)/fabs(kontrolle.zyklade-
kontrolle.zyklentlade); // Innenwiderstand
                                qqdaten[kontrolle.anzahl].i=kontrolle.zyklade;

                                        // Integrationszeit wieder zurücksetzen
                                sprintf(string, "hgiz%lu\r\n", xytmessparam.intzeit / 40);
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                saveuntermessungqq(hWnd,kontrolle.anzahl); // QQ-Tempfile
speichern
                                kontrolle.anzahl++;
                                messkontrolle.nextcycle=0;

                        }
                }
                else if((kontrolle.anzahl%2) ==1 ) // Entladen zyklisieren
                {
                        if(*(xytdaten.xyl[1] + xytdaten.datacount)<=kontrolle.zyklcutoffunten ||
(messkontrolle.nextcycle==NEXTCYCLE)) // Entladespannung erreicht
                        {
```

```
                                qqdaten[kontrolle.anzahl].intzeit=*(xytdaten.tl + xytdaten.datacount)-
letztezeit;// Zyklendauer berechnen
                                letztezeit=*(xytdaten.tl + xytdaten.datacount);

        qqdaten[kontrolle.anzahl].q=qqdaten[kontrolle.anzahl].intzeit*kontrolle.zyklentlade ; // Ladung
berechnen

                                // Integrationszeit auf 40mS
                                sprintf(string, "hgiz%lu\r\n", 1);
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                // Spannung für Innenwiderstand bestimmen
                                uvor=spannunginnenwiderstand();

                                // Strombereich einstellen
                                sprintf(string, "hgtu%lf\r\n",0); // Strom kurz auschalten
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                if(messkontrolle.cyclefinish==ENDCYCLE)
                                {
                                        // Zelle aus
                                        sprintf(string, "hgz0\r\n");
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout,
&countout, NULL);

                                        sprintf(ausgabetext,"Com: %i",xytmessparam.comport);
                                        MessageBox(hWnd, "Messung fortsetzen", ausgabetext,
MB_OK|MB_ICONQUESTION);

                                        // Zelle an
                                        sprintf(string, "hgz1\r\n");
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout,
&countout, NULL);


        kontrolle.zeitkorr=TimeRead(xytmessparam.StartTime,xytmessparam.TimeUnit) / 1000 -
*(xytdaten.tl + xytdaten.datacount);
                                        messkontrolle.cyclefinish=0;
                                }

                                if(fabs(kontrolle.zyklade)>=STROMSCHWELLE)
                                {
                                        kontrolle.rstrom=RSTROMGROSS;
                                        sprintf(string, "hgsg\r\n");
                                }
                                else
                                {
                                        kontrolle.rstrom=RSTROMKLEIN;
                                        sprintf(string, "hgsk\r\n");
                                }
                                countout = strlen(string);
                                WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                // Strom einstellen, Laden mit Ladestrom
                                sprintf(string, "hgtu%lf\r\n",kontrolle.zyklade*1e6*kontrolle.rstrom);
                                countout = strlen(string);
```

```
                                        WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                        // Spannung für Innenwiderstand bestimmen
                                        unach=spannunginnenwiderstand();

                                        qqdaten[kontrolle.anzahl].r=fabs(uvor-unach)/fabs(kontrolle.zyklade-
kontrolle.zyklentlade); // Innenwiderstand
                                        qqdaten[kontrolle.anzahl].i=kontrolle.zyklentlade;

                                                // Integrationszeit wieder zurücksetzen
                                        sprintf(string, "hgiz%lu\r\n", xytmessparam.intzeit / 40);
                                        countout = strlen(string);
                                        WriteFile(xytmessparam.Comhandle, string, countout, &countout,
NULL);

                                        saveuntermessungqq(hWnd,kontrolle.anzahl);      //      QQ-Tempfile
speichern
                                        kontrolle.anzahl++;
                                        messkontrolle.nextcycle=0;

                                }
                        }
                        saveuntermessung(hWnd,xytdaten.datacount);

                        // neue Messung anfordern
                        WriteFile(xytmessparam.Comhandle, "hgmm\r\n", countout, &countout, NULL);

                        // Timer starten
                        timeSetEvent(xytmessparam.intzeit,                      xytmessparam.intzeitaufl
,timerroutine,0,TIME_ONESHOT); // Timer an

                        xytdaten.datacount++; // für nächsten Durchlauf hochsetzen
                        PostMessage(hWnd, WM_PAINT, 0, 0); // damit die neuen Punkte auch gezeichnet
werden
                }
}

void CALLBACK timerroutine(UINT uID,UINT uMsg,DWORD dwUser,DWORD dw1,DWORD dw2)
{
        SendMessage(hWnd,WM_TIMERHGS,0,0);
        return;
}

void cleanupxy(void)
{
        int i;

        if(xytdaten.datacount == 0) return; // Aufruf sinnlos, sowieso nichts allokiert
        xytdaten.datacount = 0;
        GlobalUnlock(xytdaten.tHandle);
        GlobalFree(xytdaten.tHandle);
//      free(xytdaten.tl);
        for(i=0; i<MAXKANAL; i++)
        {
                GlobalUnlock(xytdaten.xyHandle[i]);
                GlobalFree(xytdaten.xyHandle[i]);
//              free(xytdaten.xyl[i]);
        }
}

/*****************************************************************************************
```

```
*                   Speichern der xyt-Daten, verwendet nur 32-Bit Routinen
                                                *
*********************************************************************************************/

void savexyt(void)
{
char zeile[1010],kanaltxt[4];
static char szFilename[_MAX_FNAME + _MAX_EXT],szFilenameQQ[_MAX_FNAME + _MAX_EXT];
// damit bei Cancel nicht das Original überschrieben wird
int kanal, iErg=0;
OPENFILENAME ofn;
HANDLE hFile;
BOOL status;
unsigned long i; // Datensatz
unsigned long schreibbytes = 0; // für Windows95 unbedingt erforderlich

   memset(&ofn, 0, sizeof(OPENFILENAME));
   ofn.lpstrFile = szFilename;
   iErg = FileDialogSave(hWnd, &ofn);

         if(iErg == IDCANCEL) return;

   if((hFile = CreateFile(    szFilename, // lpFileName: Dateiname
                                                GENERIC_WRITE,  //  dwDesiredAccess:
Zugriffsmodus schreibend
                                                0, // dwShareMode: 0 für kein Sharing
                                                NULL, // lpSecurityAttributes: NULL heißt
Handle nicht vererbbar
                                                CREATE_ALWAYS,                  //
dwCreationDisposition: falls vorhanden überschreiben, falls nicht neu erstellen
                                                FILE_ATTRIBUTE_NORMAL,          //
dwFlagsAndAttributes: normal
                                                NULL    //    hTemplateFile:    muss    für
Windows95 NULL sein
                                                )) == INVALID_HANDLE_VALUE)
   {
                  MessageBox(hWnd,    "Datei    konnte    nicht    angelegt    werden!",    "Fehler",
MB_OK|MB_ICONSTOP);
      return;
   }

         // Daten schreiben
         i = 0;
         do
         {
                  sprintf(zeile,"%lg", xytdaten.tl[i]); // Zeit Ausgeben
                  status = WriteFile(hFile, zeile, strlen(zeile), &schreibbytes, NULL);
                  if(status == 0) break; // damit ein Fehler nicht durch nachfolgende Erfolge überschrieben
wird
                  for(kanal = 0; kanal < MAXKANAL; kanal++) // Messwerte ausgeben
                  {
                           sprintf(zeile,"\t%lg", *(xytdaten.xyl[kanal] + i));
                           status = WriteFile(hFile, zeile, strlen(zeile), &schreibbytes, NULL);
                           if(status == 0) break; // damit ein Fehler nicht durch nachfolgende Erfolge
überschrieben wird
                  }
                  status = WriteFile(hFile, "\r\n", 2, &schreibbytes, NULL);
                  i++;
         } while(i < xytdaten.datacount && status != 0);

         if(status == 0)
         {
```

```
                MessageBox(hWnd, "Fehler beim Schreiben in die Datei.", "Fehler",
MB_OK|MB_ICONSTOP);
        }
        CloseHandle(hFile);

        if(MessageBox(hWnd, "Q+/Q- Daten rauschreiben ?", "Achtung",
MB_YESNO|MB_ICONQUESTION)==IDYES)
        {
                // Filenamenbasteln
                strncpy(szFilenameQQ,szFilename,strlen(szFilename)-4);
                szFilenameQQ[strlen(szFilename)-4]=0;
                sprintf(kanaltxt,"_qq.txt");
                strcat(szFilenameQQ,kanaltxt);

                if((hFile = CreateFile(      szFilenameQQ, // lpFileName: Dateiname
                                                                GENERIC_WRITE,          //
dwDesiredAccess: Zugriffsmodus schreibend
                                                                0, // dwShareMode: 0 für kein
Sharing
                                                                NULL, // lpSecurityAttributes:
NULL heißt Handle nicht vererbbar
                                                                CREATE_ALWAYS,          //
dwCreationDisposition: falls vorhanden überschreiben, falls nicht neu erstellen
                                                                FILE_ATTRIBUTE_NORMAL,
// dwFlagsAndAttributes: normal
                                                                NULL // hTemplateFile: muss für
Windows95 NULL sein
                                                                )) ==
INVALID_HANDLE_VALUE)
                {
                        MessageBox(hWnd, "Datei konnte nicht angelegt werden!", "Fehler",
MB_OK|MB_ICONSTOP);
                        return;
                }

                // Daten schreiben
                i = 0;
                do
                {
                        //sprintf(zeile,"%le, %le", qqdaten[i].intzeit,qqdaten[i].q);
                        sprintf(zeile,"%.1lf %le %le %le %le", (double) i /2, qqdaten[i].intzeit, qqdaten[i].i
,qqdaten[i].q,qqdaten[i].r);
                        status = WriteFile(hFile, zeile, strlen(zeile), &schreibbytes, NULL);
                        if(status == 0) break; // damit ein Fehler nicht durch nachfolgende Erfolge
überschrieben wird
                        status = WriteFile(hFile, "\r\n", 2, &schreibbytes, NULL);
                        i++;
                } while(i < kontrolle.anzahl && status != 0);

                if(status == 0)
                {
                        MessageBox(hWnd, "Fehler beim Schreiben in die Datei.", "Fehler",
MB_OK|MB_ICONSTOP);
                }
                CloseHandle(hFile);

        }

        GlobalUnlock(handleqqdata); //Speicher wieder freigeben
        GlobalFree(handleqqdata);
}

void saveuntermessung(HWND hWnd, long int i) // speichert Messdaten wärend der Messung
```

```
{
HANDLE hFile;
char zeile[1000];
//unsigned long dummy;
int kanal;
unsigned long schreibbytes = 0; // für Windows95 unbedingt erforderlich

	zeile[0]=0;
	hFile=CreateFile(kontrolle.tempfile,GENERIC_WRITE,FILE_SHARE_READ,NULL,OPEN_AL
WAYS,FILE_ATTRIBUTE_NORMAL,NULL);

	// an das Dateiende gehen
	SetFilePointer(hFile,0,NULL,FILE_END);

	sprintf(zeile,"%lg", xytdaten.tl[i]); // Zeit ausgeben
	WriteFile(hFile, zeile, strlen(zeile), &schreibbytes, NULL);

	for(kanal = 0; kanal < MAXKANAL; kanal++) // Messwerte ausgeben
	{
		sprintf(zeile,"\t%lg", *(xytdaten.xyl[kanal] + i));
		WriteFile(hFile, zeile, strlen(zeile), &schreibbytes, NULL);
	}
	WriteFile(hFile, "\r\n", 2, &schreibbytes, NULL);

	CloseHandle(hFile);

	return;
}

void saveuntermessungqq(HWND hWnd, long int i) // speichert Messdaten während derMessung
{
HANDLE hFile;
char zeile[1000];
//unsigned long dummy;
unsigned long schreibbytes = 0; // für Windows95 unbedingt erforderlich

	zeile[0]=0;
	hFile=CreateFile(kontrolle.tempfileqq,GENERIC_WRITE,FILE_SHARE_READ,NULL,OPEN_
ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);

	// an das Dateiende gehen
	SetFilePointer(hFile,0,NULL,FILE_END);

	sprintf(zeile,"%.1lf %le %le %le %le", (double ) i /2, qqdaten[i].intzeit, qqdaten[i].i
,qqdaten[i].q,qqdaten[i].r);
	WriteFile(hFile, zeile, strlen(zeile), &schreibbytes, NULL);

	WriteFile(hFile, "\r\n", 2, &schreibbytes, NULL);

	CloseHandle(hFile);

	return;
}

void cleartempfile(HWND hWnd, char *filename) // löscht das tempfile
{

	DeleteFile(filename);

	return;
}

double spannunginnenwiderstand(void)
```

```
{
double dummy1,dummy2,dummy3,dummy4,dummy5,dummy6;
char datain[201]="0.05 1.05 2.05 3.05 4.05 5.05 6.05\r\n" ,dummy[2];
unsigned long countout = 6, countin = 11;
int i;
double uout=0;

        // Spannung für Innenwiderstand bestimmen
        WriteFile(xytmessparam.Comhandle, "hgmm\r\n", countout, &countout, NULL);
        i=0;
        do
        {
                countin=1;
                ReadFile(xytmessparam.Comhandle,dummy,countin,&countin,NULL);
                // abfragen, ob Readfile Fehler = 0 zurückliefert ?
                datain[i++]=dummy[0];
                if(i>200) break;
        }
        while(dummy[0]!='\n'); // bis Datenzeile komplett
        PurgeComm(xytmessparam.Comhandle,PURGE_RXCLEAR); // abgeholte Daten aus Puffer
löschen
        // eingelesene Zeile parsen
//      test = xytdaten.xyl[0] + xytdaten.datacount;
        // wird in der Reihenfolge eingelesen: Spannung 1, Spannung 2, Strom, Batteriespannung, AUX,
        // Daten als U[V] Kanal 0 Strom in A, Kanal 1 Spannung an der Batterie in V,  Kanal 2 AUX
Spannung, Kanal 3 Spannung 1, Kanal 4 Spannung2
        sscanf(datain, "%le %le %le %le %le %le %le",
                        &dummy1, // Spannung 1
                        &dummy2, // Spannung 2
                        &dummy3, // Strom
                        &uout, // Batteriespannung
                        &dummy4, // Auxspannung
                        &dummy5,
                        &dummy6);
        return(uout);
}
```

11.7.4.9 zykl.cpp

```
// Schreiber.cpp : Definiert den Einsprungpunkt für die Anwendung.

#include "stdafx.h"

#define MAX_LOADSTRING 100

// Globale Variablen:
extern AWSTRUCT aw; // aus mess.cpp
extern VIEWSTRUCT view; // aus grafik.cpp
HINSTANCE hInst;                                        // aktuelle Instanz
TCHAR szTitle[MAX_LOADSTRING];
        // Text der Titelzeile
TCHAR szWindowClass[MAX_LOADSTRING];
                // Text der Titelzeile
HWND hWnd; // früher lokal in Initinstance
extern MESSCONTR messkontrolle;
extern KONTROLLZYKL kontrolle;

// Vorausdeklarationen von Funktionen, die in diesem Code-Modul enthalten sind:
ATOM                    MyRegisterClass( HINSTANCE hInstance );
BOOL                    InitInstance( HINSTANCE, int );
LRESULT CALLBACK    WndProc( HWND, UINT, WPARAM, LPARAM );
LRESULT CALLBACK    About( HWND, UINT, WPARAM, LPARAM );
```

```
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,int
nCmdShow )
{

	MSG msg;
	HACCEL hAccelTable;

	// Globale Zeichenfolgen initialisieren
	LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
	LoadString(hInstance, IDC_SCHREIBER, szWindowClass, MAX_LOADSTRING);
	MyRegisterClass(hInstance);

	// Initialisierung der Anwendung durchführen:
	if( !InitInstance( hInstance, nCmdShow ) )
	{
		return FALSE;
	}

	hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_SCHREIBER);

	// Hauptnachrichtenschleife:
	while( GetMessage(&msg, NULL, 0, 0) )
	{
		if( !TranslateAccelerator (msg.hwnd, hAccelTable, &msg) )
		{
			TranslateMessage( &msg );
			DispatchMessage( &msg );
		}
	}

	return msg.wParam;
}



//
//  FUNKTION: MyRegisterClass()
//
//  AUFGABE: Registriert die Fensterklasse.
//
//  KOMMENTARE:
//
//    Diese Funktion und ihre Verwendung sind nur notwendig, wenn dieser Code
//    mit Win32-Systemen vor der 'RegisterClassEx'-Funktion kompatibel sein soll,
//    die zu Windows95 hinzugefügt wurde. Es ist wichtig diese Funktion aufzurufen,
//    damit der Anwendung kleine Symbole mit den richtigen Proportionen zugewiesen
//    werden.
//
ATOM MyRegisterClass( HINSTANCE hInstance )
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProc;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_SCHREIBER);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= (LPCSTR)IDC_SCHREIBER;
```

```
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNKTION: InitInstance(HANDLE, int)
//
//   AUFGABE: Speichert die Instanzzugriffsnummer und erstellt das Hauptfenster
//
//   KOMMENTARE:
//
//        In dieser Funktion wird die Instanzzugriffsnummer in einer globalen Variablen
//        gespeichert und das Hauptprogrammfenster erstellt und angezeigt.
//
BOOL InitInstance( HINSTANCE hInstance, int nCmdShow )
{
//   HWND hWnd; ist jetzt global

  hInst = hInstance; // Instanzzugriffsnummer in unserer globalen Variablen speichern

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if( !hWnd )
  {
    return FALSE;
  }

  ShowWindow( hWnd, nCmdShow );
  UpdateWindow( hWnd );

  return TRUE;
}

//
//  FUNKTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  AUFGABE:  Verarbeitet Nachrichten für das Hauptfenster.
//
//  WM_COMMAND      - Anwendungsmenü verarbeiten
//  WM_PAINT  - Hauptfenster darstellen
//  WM_DESTROY        - Beendigungsnachricht ausgeben und zurückkehren
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        int dataset;
//      HMENU hMenu;
        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch( message )
        {
                case WM_CREATE:
                        xytglobalinit(hWnd); // in xytmess.cpp
                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
```

```
                        // Menüauswahlen analysieren:
                        switch( wmId )
                        {
                                case IDM_DATEI_OPEN:
                                        load(hWnd);
                                        break;
                                case IDM_DATEI_SAVE:
                                        savexyt();
                                        break;
                                case IDM_DATEI_CLOSE:
                                        cleanupxy();
                                        PostMessage(hWnd, WM_PAINT, 0, 0); // damit Darstellung
entfernt wird
                                        break;
                                case IDM_MESSUNG_XYT:
                                        startxytmess(hWnd);
                                        break;
                                case IDM_OPTIONEN:
                                        DialogBox(hInst,    (LPCTSTR)IDD_OPTIONEN,    hWnd,
(DLGPROC) OPTIONENDialog);
                                        break;
                                case IDM_OPTIONENSPEICHERN:
                                        writeini(hWnd);
                                        break;

                                case ID_STEUERUNG_MESSUNGENDE:
                                        messkontrolle.startstop=MESSENDE;
                                break;

                                case ID_STEUERUNG_NCHSTERZYKLUS:
                                        messkontrolle.nextcycle=NEXTCYCLE;
                                break;

                                case ID_STEUERUNG_BISZYKLENENDEMESSEN:
                                        messkontrolle.cyclefinish=ENDCYCLE;
                                break;

                                case ID_ANSICHT_DISPLAYUMSCHALTEN:

                                        if(kontrolle.graphikstatus==STATUSFENSTER)
                                        {
                                                kontrolle.graphikstatus=GRAPHIKFENSTER;
                                        }
                                        else
                                        {
                                                kontrolle.graphikstatus=STATUSFENSTER;
                                        }

                                break;


                                case IDM_ABOUT:
                                  DialogBox(hInst,        (LPCTSTR)IDD_ABOUTBOX,        hWnd,
(DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow( hWnd );
                                  break;
                                default:
                                  return DefWindowProc( hWnd, message, wParam, lParam );
                        }
                        break;
                case WM_TIMERHGS:
```

```
                        getmessdaten(hWnd);
                        break;
                case WM_PAINT:
                        drawxyt(hWnd, PAINTMSG);
                        break;
                case WM_DESTROY: // letztes Stück Code vor Programmende
                        cleanupxy();
                        for(dataset = 0; dataset < maxdatasets; dataset++) cleanup(dataset); // Speicher
freigeben ...
                        PostQuitMessage( 0 );
                        break;
                default:
                        return DefWindowProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

// Nachrichtenbehandlungsroutine für "Info"-Feld.
LRESULT CALLBACK About( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam )
{
//extern STEUERSTRUCT st; // aus hardware.c
//double ux;

        switch( message )
        {
                case WM_INITDIALOG:
/*
   st.bipolar = 1;
        st.range = 1;
        st.kanal = 0;
        do
   {
                initstruct(&st);
      ux = (double) wandeln(st);
        } while(1);

*/                                      return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
   return FALSE;
}
```

11.7.4.10 grafik.h

```
//#define LAB 70 //für Kosy
//#define RAB 40 //Abstände, ab denen die Messwerte aufgetragen werden
//#define OAB 40 //OAB=UAB früher 30
//#define UAB 30 // früher 20
//#define BALKENK 4 // muss <als Raender sein
//#define TEXTYDX -60
//#define TEXTYDY 8
//#define TEXTXDX 0
//#define TEXTXDY 8

//#define NOPAINTMSGAUSWERTUNG 2
```

```
//#define PAINTMSG 1
//#define NOPAINTMSG 0

#define STATUSFENSTER 1
#define GRAPHIKFENSTER 0

#define ANZAHLXTICKS 10
#define ANZAHLYTICKS 10

// für updateviewstruct
#define MESSBEREICH 2
#define MAXDATEN 1
#define DONTCHANGE 0

// Werte von view.iuflag
#define U 0
#define I 1

// Prototypen für grafik.cpp
//void fensterlöschen(HDC hPaint,int maxwinx,int maxwiny);
void drawkosy(HWND hWnd, int flag);
//void DrawBitmap( HDC hdc, HBITMAP bitmap, short x, short y );
//double findmaxdouble(double *feld,unsigned long int count);
double findmindouble(double *feld,unsigned long int count);
double round(double x);
void punktzeichnen(HWND hWnd, HDC hDc, int dataset, unsigned long datenpunkt, int showvalue);
void updateviewstruct(HWND hWnd, int viewflag);
BOOL FAR PASCAL setviewDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam);

// Datenstrukturen
typedef struct
{
   // Infos zu den Messdaten für die Darstellung
   double xmin, xmax; // alle in V
   double ymin, ymax;

         int maxwinx, maxwiny; // Größe des Fensters in Pixel
         int starty; // 0-Koordinate in y-Richtung

         double maxiubereich; // größter Bereich des Nanotracers, der angezeigt wird
         short iuflag; // gibt an, ob in y-Richtung Strom oder Spannung aufgetragen wird
         short awflag; // gibt an, ob die (letzte) Auswertung sichtbar ist
}
VIEWSTRUCT;
```

11.7.4.11 hardware.h

```
#define Vref 4.096 // Gerätspezifische Parameter

#define LO 0 // mit Optokoppler 255 MM
#define HI 1 // mit Optokoppler 0 MM
#define FEHLER -1
#define OK 1
#define RSTROMGROSS 25
#define RSTROMKLEIN 1000
#define STROMSCHWELLE 9e-3

typedef struct
{
      unsigned char adresse;
      unsigned char kontrolle;
      unsigned char kanal; // von 0 bis 7
      unsigned char bipolar; // bei 1 wird bipolar gemessen
```

```
	unsigned char range; // 1 ist Vref und 0 Vref/2
	float fullscale;
	float faktor;
	float u;
}
STEUERSTRUCT;

void lpt_ausgeben(unsigned char byte);
void u_ausgabe(float u);
void cs(unsigned char wert);
float wandeln(STEUERSTRUCT st);
unsigned char i2c_einlesen(void);
void start(void);
void stop(void);
void i2c_ausgeben(unsigned char byte);
unsigned char i2c_einlesen(void);
void Ack(void);
void keinAck(void);
void init(void);
void initstruct(STEUERSTRUCT *st);
void ls245out(void);
void ls245in(void);
```

11.7.4.12 mess.h

```
#define maxdatenpunkte 2000 // Zahl der Datenpunkte, die standardmäßig allokiert werden
#define maxdatasets 10 // Maximalzahl der simultan geöffneten Kurven, wegen Farben !

#define MESSENDE 1
#define NEXTCYCLE 1
#define ENDCYCLE 1
#define GRAFIK 1
#define TEXTL 2


void globalinit(HWND hWnd);
void mess(HWND hWnd);
void einzelmess(HWND hWnd, HDC hDc, int dataset, double offset);
int save(HWND hWnd, int dataset); // speichert die Daten; verwendet 16 Bit Routinen
int FileDialogSave(HWND hWnd, OPENFILENAME *ofn); // stellt den Savedialog dar
int load(HWND hWnd); // Öffnen von Datendateien; verwendet 16 Bit Routinen
int FileDialogOpen(HWND hWnd, OPENFILENAME *ofn, short typ); // stellt den Dialog zum
Dateiöffnen dar
// Typ: Ausgewählte Dateiextension: 1 = .XY, 2 = .TXT
BOOL FAR PASCAL DatasetChooseDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM
lParam);
int speicherallokieren(HWND hWnd); // allokiert den nächsten freien Datensatz und gibt die Nummer
// in C-Syntax zurück
void cleanup(int dataset);
void auswert(HWND hWnd);
BOOL FAR PASCAL AWDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam);
void filtern(int dataset);
double getadcoffset(unsigned char kanal); // Kanal, der auf Masse gelegt wird in C-Nomenklatur
int readiubereich(HWND hWnd, HFILE hFile, int dataset);

BOOL FAR PASCAL MessDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam);
void getawfile(HWND hWnd); // speichert den Dateinamen der AW global in aw.dateiname
void saveaw(HWND hWnd, short dataset); // speichert die Auswertung von Datensatz dataset auf Platte
// saveaw arbeitet nur mit 32 Bit Dateiroutinen

typedef struct
{
```

```
   // Variablen für Daten- und Speicherverwaltung
        HGLOBAL xlHandle;
        HGLOBAL ylHandle;
   double *xl; // x-Daten als U[V]
   double *yl; // y-Daten als U oder I
   unsigned long datacount; // Zahl der Datenpunkte, wenn = 0 dann ist Datensatz ungültig
        int datasets; // Zahl der tatsächlich geladenen Datensätze, wird nur auf Index 0 benutzt

   // Gerätespezifische Variablen
   int adbereich; // 1 ist Vref und 0 Vref/2
   int bipolar; // 1 für bipolar
   double iubereich; // in µA/V entsp. Nanotracer

        // Variablen der Auswertung
        double xpeak; // Spannung am Peakmaximum
        double ypeak; // und y-Wert dazu
        double xback; // zugehöriger x-Wert Backscan, wegen Messungenauigkeit bei Backscanmethode
nötig
        double yback; // zugehöriger y-Wert Backscan
        double vol;          // Volumen der Addition
        char dateiname[81]; // Dateiname der XY-Datei, wie oben im Fenster angezeigt
}
DATENSTRUCT;

typedef struct
{
        int dataset;
        double xmin, xmax;
        double xpeakmanu; // zur manuellen Eingabe der Peakspannung
        BOOL manuell;
        BOOL tangente;
        char dateiname[_MAX_FNAME + _MAX_EXT]; // ist 512 Zeichen
        unsigned long xminpunkt, xmaxpunkt; // Datenpunkte für Anfang- und Ende der Peaksuche &
Tangente
}
AWSTRUCT;

typedef struct
{
   double iubereich; // in µA/V entsp. Nanotracer
        double zeit; // Meßzeit pro Datenpunkt in ms
        double filter; // Filter in V
}
MESSTRUCT;

typedef struct //  Kontrolliert die Messung
{
        int startstop;
        int nextcycle;
        int cyclefinish;
}
MESSCONTR;
```

11.7.4.13 my_timer.h

```
/*********************************************************
* Diese Datei enthaeält Funktionen für genaue Timer unter
* 1 ms und Task-Prioritäts-Umschaltefunktionen.
* Damit sind Timer und CPU-Unabhängige Delayzeiten
* möglich.
* Die Datei My_Timers.c enthält die Implementierung der
* Funktionen
*********************************************************/
```

```
#ifndef my_timerH
#define my_timerH

//LARGE_INTEGER StartTime;
//float TimeUnit;

/*********************************************************
* Name: RealTime
* Setzt die Task-Priorität auf MAX.
* Dann sind z.B. KEINE Mausaktionen mehr möglich!!
* Abschalten mit NormalTime NICHT vergessen !!
*********************************************************/
extern "C" void RealTime(void);

/*********************************************************
* Name: NormalTime
* setzt die Task-Priorität wieder auf normal. Dann arbeitet
* das Programm wieder normal. Maus, Tastatur usw.
*********************************************************/
extern "C" void NormalTime(void);

/*********************************************************
* Name: TimeInit
* gibt den momentanen Stand des high-resolution Counters
* als Funktionswert zurück und schreibt die Zeitauflösung
* in die Adresse der Float.
* Falls die PC-Hardware keinec o.g. Timer unterstützt,
* wird 0 als Funktionswert zurückgeliefert.
* Die Float bleibt dann unverändert.
*********************************************************/
extern "C" LARGE_INTEGER TimeInit(float *);

/*********************************************************
* Name: TimeRead
* gibt die seit dem letzten Aufruf v. TimeInit verstrichene
* Zeit als Real (ms.us) als Funktionswert zurück.
* Die integer-Startzeit wird als large-Int und die Zeitauf-
* lösung als float ms.us (zuvor mit TimeInit ermittelt)
* übergeben.
*********************************************************/
extern "C" float TimeRead(LARGE_INTEGER,float);

/*********************************************************
* Name: Delay
* Wartet per TimeStart und TimeRead-Funktionen die als
* float übergebene Zeit (Millisekunden.Mikrosekunden)
* Zuvor muss einmal TimeInit zwecks Berechnung der TimeUnit
* und StartTime aufgerufen worden sein.
*********************************************************/
extern "C" void Delay(float,LARGE_INTEGER,float);


/************  Beispiele *****************
* Beispiele:

void __fastcall TForm1::DelayButtonClick(TObject *Sender)
{LARGE_INTEGER StartTime;
 float TimeUnit,T;
 AnsiString aString;
 T=float(ScrollBar1->Position)/1000;
 RealTime();
 StartTime=TimeInit(&TimeUnit);
```

```
 for(int i=500; i; i--)                    // 500*2 Delayzeiten
  {SetPortByte(HwCtrl,AD16BaseAdr+oDIG_OUT,0);
   Delay(T,StartTime,TimeUnit);
   SetPortByte(HwCtrl,AD16BaseAdr+oDIG_OUT,1);
   Delay(T,StartTime,TimeUnit);
  }
 T=TimeRead(StartTime,TimeUnit);
 NormalTime();
 aString.sprintf("%.4f ms",T);
 Label6->Caption=aString;
}


void __fastcall TForm1::WhileButtonClick(TObject *Sender)
{LARGE_INTEGER StartTime;
 float TimeUnit,T;
 T=float(ScrollBar1->Position)/1000;
 RealTime();
 StartTime=TimeInit(&TimeUnit);
 for(int i=0; i<500; i++)                  // 500*2 Delayzeiten
  {SetPortByte(HwCtrl,AD16BaseAdr+oDIG_OUT,0);
   while(TimeRead(StartTime,TimeUnit)<(2*i-1)*T); // eliminiert die Verzögerung
   SetPortByte(HwCtrl,AD16BaseAdr+oDIG_OUT,1);    // durch die Funktionsaufrufe
   while(TimeRead(StartTime,TimeUnit)<(2*i)*T);
  }
 T=TimeRead(StartTime,TimeUnit);
 NormalTime();
}
*****************************************/

#endif
```

11.7.4.14 small.h

```
BOOL FAR PASCAL OPTIONENDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM
lParam);
void writeini(HWND hWnd);
void warten(double sek); // warten in Sekunden
LARGE_INTEGER TimeInithgs(double * TimeUnit); //gibt die Zeit eines Ticks des Hires Counters in s
zurück
```

11.7.4.15 stdafx.h

```
// stdafx.h : Include-Datei für Standard-System-Include-Dateien,
//  oder projektspezifische Include-Dateien, die häufig benutzt, aber
//      in unregelmäßigen Abständen geändert werden.
//

#if !defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)
#define AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define WIN32_LEAN_AND_MEAN                 // Selten benutzte Teile der Windows-Header nicht
einbinden


// Windows-Header-Dateien:
#include <windows.h>

// Header-Dateien der C-Laufzeit
```

```
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>

// Lokale Header-Dateien

// ZU ERLEDIGEN: Verweisen Sie hier auf zusätzliche Header-Dateien, die Ihr Programm benötigt
#include <conio.h> // für _outp und _inp in hardware.cpp
#include <math.h> // für fabs in mess.cpp
#include <commdlg.h> // für GetSaveFileName in mess.cpp
#include <stdio.h> // für sprintf in mess.cpp
#include <mmsystem.h> // für Multimediacounter timeSetEvent
#include "resource.h" // damit die IDs in allen Quelltextdateien bekannt sind
#include "hardware.h"
#include "mess.h"
#include "grafik.h"
#include "xytgrafik.h"
#include "My_Timer.h"
#include "xytmess.h"
#include "small.h"

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ fügt zusätzliche Deklarationen unmittelbar vor der vorherigen Zeile ein.

#endif                                                                                                    //
!defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)
```

11.7.4.16 xyrgrafik.h

```
//define Anweisungen

// Paintkontrolle
#define NOPAINTMSG 2
#define PAINTMSG 1
#define NOPAINTMSGAUSWERTUNG 0

// Abstände für Kosy
#define LAB 70 //für Kosy
#define RAB 70 //Abstände, ab denen die Messwerte aufgetragen werden
#define OAB 70 //OAB=UAB
#define UAB 70
#define BALKENK 4 // muss <als Raender sein
#define TEXTYDX -60
#define TEXTYDY 8
#define TEXTXDX 0
#define TEXTXDY 8
#define ANZAHLXTICKS 10
#define ANZAHLYTICKS 10

#define LABK 100//44 //für Kosy für die kleine KOSYs
#define RABK 80 //Abstände, ab denen die Messwerte aufgetragen werden
#define OABK 20
#define UABK 28
#define TEXTYDXK -90// -40

#define TEXTDIFF 80 // Textabstände für Statusfenster
#define LABTEXT 15

#define FENSTERSCHRITT 20
#define ASLPAINT 1111

/*-- Prototypen --*/
```

```
void fensterloeschen(HDC hPaint,int maxwinx,int maxwiny);
void DrawBitmap( HDC hdc, HBITMAP bitmap, short x, short y );
double findmaxdouble(double *feld,unsigned long int count);
void drawxyt(HWND hWnd, int flag); // Zeichnetkosy

// Strukturen
typedef struct // Koordinaten der Fenster
{
	int xMess1;
	int yMess1;
	int nWidthMess1;
	int nHeightMess1;

	int xMess2;
	int yMess2;
	int nWidthMess2;
	int nHeightMess2;

	int xMess3;
	int yMess3;
	int nWidthMess3;
	int nHeightMess3;

	int xMess4;
	int yMess4;
	int nWidthMess4;
	int nHeightMess4;

	int xMessStatus;
	int yMessStatus;
	int nWidthMessStatus;
	int nHeightMessStatus;

	int xMessHiRes1;
	int yMessHiRes1;
	int nWidthMessHiRes1;
	int nHeightMessHiRes1;

	int xMessHiRes2;
	int yMessHiRes2;
	int nWidthMessHiRes2;
	int nHeightMessHiRes2;

	int xMessHiRes3;
	int yMessHiRes3;
	int nWidthMessHiRes3;
	int nHeightMessHiRes3;

	int xMessHiRes4;
	int yMessHiRes4;
	int nWidthMessHiRes4;
	int nHeightMessHiRes4;

	int xMessMain;
	int yMessMain;
	int nMessWidthMain;
	int nMessHeighMain;

	int xMessASL;
	int yMessASL;
	int nWidthMessASL;
	int nHeightMessASL;
```

```
        int xMessHiResASL;
        int yMessHiResASL;
        int nWidthMessHiResASL;
        int nHeightMessHiResASL;
}
FENSTERKOORDSSTRUCT;
```

11.7.4.17 xytmess.h

```
#define MAXKANAL 7
#define WM_TIMERHGS WM_USER+1 // Message für eigenen Timer
#define INIFILENAME "zyklisiergerät.ini"

void xytglobalinit(HWND hWnd);
int getxytspeicher(HWND hWnd);
BOOL FAR PASCAL XYtMessDialog(HWND hDlg, WORD msg, WPARAM wParam, LPARAM lParam);
void startxytmess(HWND hWnd);
void comopen(void);
int checkschreiber(void);
void getmessdaten(HWND hWnd);
void CALLBACK timerroutine(UINT uID,UINT uMsg,DWORD dwUser,DWORD dw1,DWORD dw2);
void cleanupxy(void); // gibt allokierten Speicher für xyt-Daten wieder frei
void savexyt(void);
void saveuntermessung(HWND hWnd, long int i); // speichert Messdaten während der Messung
void saveuntermessungqq(HWND hWnd, long int i); // speichert Messdaten während der Messung
void cleartempfile(HWND hWnd, char *filename); // löscht das tempfile
double spannunginnenwiderstand(void);

typedef struct
{
        unsigned long intzeit; // Messzeit pro Datenpunkt in ms
        unsigned int intzeitaufl; // Auflösung der Integrationszeit für MMCounter
   unsigned long messzeit; // gesamte Messdauer in s
        unsigned long maxpunkte; // maximale Anzahl Messpunkte laut Messzeit und Intzeit
        LARGE_INTEGER StartTime; // für Zeitfunktionen aus My_Timer.cpp
        float TimeUnit; // Zeitauflösung für Zeitfunktionen aus My_Timer.cpp
        int pga[MAXKANAL]; // 7 Kanäle: 0 = nix, 1, 10, 100, 1000
        int comport;
        HANDLE Comhandle;
        int messtyp;
}
XYTMESSTRUCT;

typedef struct
{
   // Variablen für Daten- und Speicherverwaltung
        HGLOBAL tHandle;
        HGLOBAL xyHandle[MAXKANAL];

        double *tl; // Zeit in s
   double *xyl[MAXKANAL]; // anal 0 Strom in A, Daten als U[V] Kanal 1 Spannung an der Batterie in V,
K Kanal 2 AUX Spannung, Kanal 3 Spannung 1, Kanal 4 Spannung 2
   unsigned long datacount; // aktuelle Zahl der Datenpunkte

//      char dateiname[81]; // Dateiname der XY-Datei, wie oben im Fenster angezeigt
}
XYTDATENSTRUCT;

typedef struct
{
        unsigned long int anzahl; // Anzahl der Schritte
        unsigned long int maxanzahl; // maximale Anzahl der Schritte
        double formlade; // Ladestrom Formierung
```

```
	double formentlade; // Entladestrom Formierung
	double zyklade; // Ladestrom
	double zyklentlade; // Entladestrom
	double formcutoffoben; // Obere Spannungsgrenze Formierung
	double formcutoffunten; // Untere Spannungsgrenze Formierung
	double zyklcutoffoben; // Obere Spannungsgrenze
	double zyklcutoffunten; // Untere Spannungsgrenze
	double rstrom;
	char tempfileqq[_MAX_FNAME + _MAX_EXT]; // Pfad für qq tempfile
	char tempfile[_MAX_FNAME + _MAX_EXT]; // Pfad für tempfile
	char defaulttempfileqq[_MAX_FNAME + _MAX_EXT]; // Pfad für qq tempfile
	char defaulttempfile[_MAX_FNAME + _MAX_EXT]; // Pfad für tempfile

	int graphikstatus;
	double zeitkorr; // Korrektur der Zeit durch Messunterbrechungen
}
KONTROLLZYKL;

typedef struct
{
	double intzeit; // Messzeit pro Datenpunkt in ms
	double q; // Ladung
	double r; // Innenwiderstand
	double i; // Strom
}
QQSTRUCT;
```

11.7.4.18 zykl.h

```
#if !defined(AFX_SCHREIBER_H__B31230D6_1166_4D84_9E14_F5C5EC3707C8__INCLUDED_)
#define AFX_SCHREIBER_H__B31230D6_1166_4D84_9E14_F5C5EC3707C8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

// #include "resource.h"

#endif                                                                                                    //
!defined(AFX_SCHREIBER_H__B31230D6_1166_4D84_9E14_F5C5EC3707C8__INCLUDED_)
```

11.7.4.19 resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by zykl.rc
//
#define IDC_MYICON                    2
#define IDD_SCHREIBER_DIALOG          102
#define IDD_ABOUTBOX                  103
#define IDS_APP_TITLE                 103
#define IDM_ABOUT                     104
#define IDM_EXIT                      105
#define IDS_HELLO                     106
#define IDI_SCHREIBER                 107
#define IDI_SMALL                     108
#define IDC_SCHREIBER                 109
#define IDR_MAINFRAME                 128
#define IDD_AWDLG                     129
#define IDD_GETDATASETDLG             130
#define IDD_ANSICHT                   131
#define IDB_LPFEIL                    133
```

```
#define IDB_UPFEIL 135
#define IDD_MESS 136
#define IDD_STARTMESS 137
#define IDD_STARTMESSXY 137
#define IDD_OPTIONEN 138
#define IDC_AWXMIN 1000
#define IDC_AWDATASET 1001
#define IDC_AWXMAX 1002
#define IDC_XMAX 1003
#define IDC_XMIN 1004
#define IDC_YMAX 1005
#define IDC_YMIN 1006
#define IDC_EINHEIT 1007
#define IDC_EINHEIT1 1008
#define IDC_IUBEREICH 1008
#define IDC_MESSZEIT 1009
#define IDC_XFILTER 1010
#define IDC_TANGENTE 1011
#define IDC_AW_VOL 1012
#define IDC_AW_MANUELL 1013
#define IDC_AW_XPEAK 1014
#define IDC_KANAL1100 1015
#define IDC_KANAL11000 1016
#define IDC_RADIO1NIX 1017
#define IDC_KANAL1NIX 1017
#define IDC_KANAL110 1018
#define IDC_KANAL11 1019
#define IDC_KANAL1101 1020
#define IDC_KANAL2100 1020
#define IDC_KANAL11001 1021
#define IDC_KANAL21000 1021
#define IDC_RADIO1NIX2 1022
#define IDC_KANAL2NIX 1022
#define IDC_KANAL111 1023
#define IDC_KANAL210 1023
#define IDC_KANAL12 1024
#define IDC_KANAL21 1024
#define IDC_MESSMAXMESSDAUER 1025
#define IDC_MAXMESSDAUER 1025
#define IDC_INTZEIT 1026
#define IDC_COMPORT 1027
#define IDC_FORMLADE 1028
#define IDC_FORMENTLADE 1029
#define IDC_KANAL1102 1030
#define IDC_KANAL11002 1031
#define IDC_RADIO1NIX3 1032
#define IDC_KANAL112 1033
#define IDC_KANAL13 1034
#define IDC_KANAL1103 1035
#define IDC_KANAL11003 1036
#define IDC_RADIO1NIX4 1037
#define IDC_KANAL113 1038
#define IDC_KANAL14 1039
#define IDC_ZYKLADE 1040
#define IDC_ZYKENTLADE 1041
#define IDC_ANZAHL 1042
#define IDC_FORMUOBEN 1043
#define IDC_FORMUUNTEN 1044
#define IDC_ZYKLUOBEN 1045
#define IDC_ZYKLUUNTEN 1046
#define IDM_MESSUNG_START 32771
#define ID_DATEI_OPEN 32772
#define IDM_DATEI_OPEN 32773
```

```
#define IDM_FINDMAX                     32774
#define IDM_ANSICHT_MEBEREICH           32775
#define IDM_ANSICHT_DATENMAX            32776
#define IDM_DATEI_SAVE                  32777
#define IDM_AW_FILTER                   32778
#define IDM_DATEI_CLOSE                 32779
#define IDM_ANSICHT_EINSTELLUNGEN       32780
#define IDM_ANSICHT_I                   32782
#define IDM_DATEI_AW                    32783
#define IDM_MESSUNG_XY                  32784
#define IDM_MESSUNG_XYT                 32784
#define IDM_OPTIONEN                    32785
#define IDM_OPTIONENSPEICHERN           32786
#define ID_STEUERUNG_MESSUNGENDE        32787
#define ID_STEUERUNG_NCHSTERZYKLUS      32788
#define ID_STEUERUNG_BISZYKLENENDEMESSEN 32789
#define ID_ANSICHT_DISPLAYUMSCHALTEN    32790
#define IDC_STATIC                      -1

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        138
#define _APS_NEXT_COMMAND_VALUE         32792
#define _APS_NEXT_CONTROL_VALUE         1044
#define _APS_NEXT_SYMED_VALUE           110
#endif
#endif
```