Diplomarbeit

Heye Caspers

Entwicklung eines Mikroprozessor-gesteuerten x/y-Schreibers mit USB-Schnittstelle

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Heye Caspers

Entwicklung eines Mikroprozessor-gesteuerten x/y-Schreibers mit USB-Schnittstelle

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.rer.nat. Jochen Schneider
Zweitgutachter : Prof. Dr.-Ing. Franz Schubert

Abgegeben am 2. Juli 2008

Heye Caspers

Thema der Diplomarbeit

Entwicklung eines Mikroprozessor-gesteuerten x/y-Schreibers mit USB-Schnittstelle

Stichworte

x/y-Schreiber, Mikroprozessor, USB, A/D-Umsetzung, Verstärkerschaltung, MFC

Kurzzusammenfassung

Im Zuge dieser Arbeit wird ein Mikroprozessor-gesteuerter x/y-Schreiber entwickelt, dessen Anbindung an einen PC über USB-Schnittstelle die komfortable Verarbeitung der Messdaten ermöglicht. Das Konzept ist mit seiner Hardwarenähe hauptsächlich auf den Lehrbereich ausgelegt. Das Verständnis für die Messung soll hiermit gefördert werden. Das Ziel ist es, die rein analogen x/y-Schreiber, die bis jetzt in den Laboren Verwendung finden, zu ersetzen.
Neben dem x/y-Betrieb im Bereich von einem Millivolt bis zu mehreren Volt ist auch ein Zeitbetrieb zwischen 10 Millisekunden und 2 Minuten möglich. Durch einen frei wählbaren Offset besteht die Möglichkeit der Darstellung negativer Eingangssignale.

Heye Caspers

Title of the paper

Development of a microprocessor-controlled x/y-writer with USB interface

Keywords

x/y-Writer, Microprocessor, USB, A/D-conversion, amplifying circuits, MFC

Abstract

In the course of this work a microprocessor controlled x/y-Writer, whose connection to the PC via the USB interface allows a comfortable way to process the measured data. The concept of staying close to the measured hardware is mainly dedicated to the teaching field. The understanding of the measurement should be hereby encouraged. The aim is to replace purely analogue x / y-Writers, which are used in the laboratories until now.
In addition to the x/y operation in the area of a millivolt to several volts, also a time operation between 10 milliseconds and 2 minutes is possible. Through a free selectable offset negative input signals can be represented.

Inhaltsverzeichnis

Abkürzungsverzeichnis

A/D	Analog/Digital
ADC	Analog/Digital Conversion
CMOS	Complementary Metal Oxyd Semiconductor
CPU	Central Processing Unit
I/O	Input/Output
IC	Integrated Circuit
LCD	Liquid Crystal Display
LC-Display	Liquid Crystal Display
MFC	Microsoft Foundation Class
OP	Operationsverstärker
PC	Personal Computer
RISC	Reduced Instruction Set Computing
t/y	Spannungsmessung über eine eingestellte Zeit
USART	Universal Synchronous and Asynchronous serial Receiver and Transmitter
USB	Universal Serial Bus
x/y	Spannungsmessung relativ zu einer zweiten Spannung

1 Einführung

1.1 Motivation und Zielsetzung

In den Laboren der Hochschulen, wie auch in der Industrie, wird in der Dokumentation von Funktionstests und Praktikumsversuchen fast ausschließlich mit dem PC gearbeitet. Die Ergebnisse von durchgeführten Messungen müssen möglichst einfach in einem Dokument zusammengestellt, für Präsentationen aufbereitet oder über das Internet versendet werden können. Die digitale Ebene wird kaum mehr verlassen.

Analoge Geräte ohne PC-Schnittstellen, die Daten ausschließlich auf Papier darstellen wie die x/y-Schreiber der Serie PM8041 von Philips[1], die in Laboren zur Aufnahme von Kennlinien genutzt werden sind daher nicht mehr zeitgemäß. Die Zeichnung des Koordinatensystems, vor allem des Ursprungs und die Beschriftung der Achsen mit einer entsprechenden Skalierung sind bei diesen Geräten mit einem hohen zusätzlichen Zeitaufwand verbunden.

Mit der Entwicklung eines Mikroprozessor-gesteuerten x/y-Schreibers mit USB-Schnittstelle, der als PC-Peripheriegerät betrieben wird, sollen die analogen Geräte ersetzt werden. Die direkte Anbindung an den PC erhöht den Komfort in der Dokumentation und den Austausch der Daten. Durch eine automatische Erstellung des Koordinatensystem mit sinnvoller Skalierung erfolgt zudem eine Vereinfachung und Zeitersparnis.

Ziel dieser Arbeit ist ein Mikroprozessor-gesteuerter x/y-Schreiber, der vor allem im Lehrgebiet eingesetzt werden soll. Die analogen x/y-Schreiber bieten hier durch ihre Hardwarenähe einen guten Einblick in die Funktion der auszumessenden Bausteine. Die ausschließlich manuelle Einstellung der Aufnahmeparameter fordert eine eingehende Auseinandersetzung mit der Messung und fördert das Verständnis. Diese Hardwarenähe des analogen Gerätes wird deshalb weitestgehend übernommen. Die Einstellung der Parameter erfolgt ausschließlich an dem Peripheriegerät. Die so eingestellten Parameter werden auf einem Display angezeigt, so dass der gesamte Messvorgang unabhängig vom PC vorbereitet wird. Die Datenübertragung über die USB-Schnittstelle erfolgt nur in Richtung des PCs zur Anzeige und Verwaltung der Daten. Die Darstellung der Messdaten soll in „Echtzeit“ noch während der Messung erfolgen. Im PC-Programm kann anschließend an

[1] Siehe [1] Gebrauchsanleitung A4 x-y Recorder.

die Messung eine Bearbeitung und Verwaltung der Messdaten vorgenommen werden. Ziel ist es demnach, die vorhandenen analogen Geräte durch x/y-Schreiber mit direkter PC-Anbindung zu ersetzen, ohne das Verständnis für die durchgeführte Messung durch eine entsprechend hardwarenahe Bedienung zu vermindern.

1.2 Anforderungen

Mindestvoraussetzung für den Mikroprozessor-gesteuerten x/y-Schreiber mit USB-Schnittstelle ist die Umsetzung aller Möglichkeiten des analogen x/y-Schreibers PM8041 . Dabei sollen die Vorzüge, wie die oben erwähnte Hardwarenähe, erhalten bleiben.
Die Amplitude der zu messenden Eingangssignale liegt zwischen einigen Millivolt und mehreren Volt. Die Auflösung der vorhandenen Geräte ist mit 1 mV/cm bis 1 V/cm, bezogen auf ein DIN A3-Blatt, angegeben. Durch Verstärkung oder eine entsprechend hohe Messgenauigkeit soll das Ergebnis der Messung, bezogen auf die Amplitude des Eingangssignals, etwa 12 Bit, das heißt 4096 Punkten, entsprechen. Entsprechend dem PM8041 muss die Aufnahme negativer Eingangsspannungen gegeben sein und beide Kanäle des x/y-Schreibers sind mit einem regelbaren Offset zu versehen. Neben dem x/y-Betrieb muss ein Eingangssignal auch über einen vorherbestimmten Zeitraum gemessen werden können.
Im Gegensatz zu dem analogen x/y-Schreiber, der mit einem Schreiber eine Linie entsprechend der Eingangssignale auf ein Blatt Papier zeichnet, ist in der digitalen Umsetzung auf eine entsprechend hohe Abtastrate zu achten. Während bei den analogen Schreibern die Trägheit des Systems bei schnellen Spannungsänderungen zu Messungenauigkeiten führt, werden diese in der digitalen Umsetzung bei einer geringen Abtastrate nur ungenügend aufgelöst oder gar nicht erkannt. Die Messung der Daten, die Übertragung und die Verarbeitung und Darstellung durch den PC müssen zeitlich optimiert sein. Ein Minimum von 1000 Punktmessungen pro Sekunde, das heißt einer Frequenz von 1000 Hertz, soll erreicht werden. Die Darstellung der Messung soll entsprechend den Vergleichsgeräten während der Messung erfolgen.

1.3 Inhalt der Arbeit

Nach der in diesem Kapitel einleitenden Darlegung der Motivation und Zielsetzung sowie der minimal umzusetzenden Anforderungen ist diese Arbeit wie folgt aufgebaut. Zunächst werden einige Grundlagen, die dem Verständnis der darauf folgenden Kapitel dienen sollen, dargelegt. Anschließend wird die Planung und Realisierung des Mikroprozessorgesteuerten x/y-Schreibers mit USB-Schnittstelle beschrieben, wobei eine Aufteilung in drei Themenbereich erfolgt. Der erste befasst sich mit der Verstärkung der Eingangssignale, der zweite mit deren Steuerung, der Aufnahme der Messwerte und der Datenübertragung zum PC und der dritte behandelt die Darstellung und Verwaltung der Messdaten durch ein PC-Programm. Auf die wesentlichen Funktionen der einzelnen Bereiche wird Bezug nehmend auf die jeweiligen Bauteile oder Blöcke genauer eingegangen. Fehlerbetrachtungen werden durchgeführt und Ausschnitte aus den Programmcodes zum besseren Verständnis gekürzt dargestellt.
Anschließend folgt die Prüfung der einzelnen Funktionen des x/y-Schreibers. Zunächst wird die Steuerung und Datenaufnahme in Verbindung mit der Datenübertragung und Darstellung durch das PC-Programm untersucht. Nach erfolgreicher Inbetriebnahme und Prüfung wird die Verstärkung mit einbezogen. Größere Anpassungen aufgrund der Testergebnisse und deren Auswirkungen auf den Ablauf der Messung werden vergleichend betrachtet.

2 Grundlagen

Die hier dargestellten Grundlagen sollen dem Verständnis der nachfolgenden Kapitel dienlich sein. Behandelt werden im Wesentlichen die Grundlagen der seriellen Kommunikation, der A/D-Umsetzung und einiger Operationsverstärkerschaltungen. Sie sind bei weitem nicht vollständig und sollen lediglich einen Überblick verschaffen. Für weitergehende Informationen sei auf die entsprechende Literatur verwiesen.

2.1 x/y-Schreiber

Messschreiber im Allgemeinen sind Messgeräte, die den Verlauf der Messgröße direkt auf Papier in ein Liniendiagramm zeichnen. In der Regel wird eine Größe über dem zeitlichen Verlauf aufgetragen. Dabei wird meistens ein Papiervorschub verwendet, wie zum Beispiel bei analogen Geräten zur Wetterüberwachung oder EKGs.
x/y-Schreiber haben anstelle der Zeitachse eine zweite Messwertachse. Das Papier liegt statisch auf einer Schreibplatte und verfügt über keinen Vorschub. Da die eine Größe in Abhängigkeit zur zweiten Messgröße dargestellt werden kann, eignen sie sich gut zur Aufnahme von Kennlinien. Diese werden bei analogen x/y-Schreibern durch einen von der Eingangsspannung gesteuerten Schreiber gezeichnet. Einige x/y-Schreiber verfügen zudem über die Möglichkeit, eine Eingangsspannung über eine voreingestellte Zeit aufzunehmen.
Über eine komplexe Verstärkerschaltung ist eine Einstellung für verschiedene Spannungsbereiche möglich, um eine möglichst passende Darstellung auf einem Blatt Papier zu erreichen. Die Skala reicht von einigen Millivolt pro Zentimeter (mV/cm) bis zu einem Volt pro Zentimeter[2] (1 V/cm). Außerdem kann mit einer einstellbaren Offsetspannung der Nullpunkt der Eingangsspannung künstlich verschoben werden, um die Darstellung zu optimieren.

[2] Siehe [1] Gebrauchsanleitung A4 x-y Recorder.

2.2 Serielle Datenübertragung

Bei der seriellen Schnittstelle, wie zum Beispiel bei einer RS232-Schnittstelle, werden einzelne Zeichen nacheinander und grundsätzlich asynchron, das heißt ohne Taktübertragung, über nur eine Leitung bitseriell übertragen. Für den Datentransfer muss vorher ein Protokoll für die angeschlossenen Geräte vereinbart werden, in dem die Übertragungsrate und der verwendete Zeichencode, aber auch die Anzahl an Start-, Stoppbits und das Paritätsbit festgelegt werden[3].

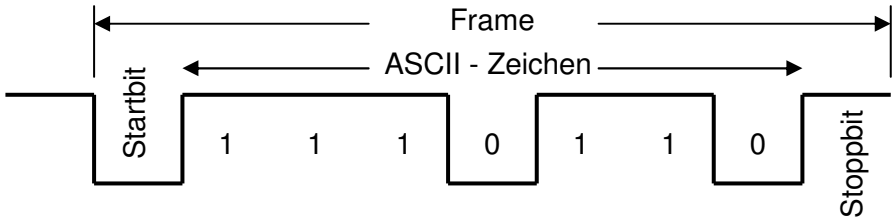


Bild 1: Beispiel eines Datenwortes der seriellen Datenübertragung

Es gibt drei verschiedene Möglichkeiten Daten zwischen zwei Teilnehmern auszutauschen. Dies sind der Ein-Richtungsbetrieb (simplex), der Wechselbetrieb (halbduplex) und der Gegenbetrieb (duplex). Für den Duplexbetrieb werden mindestens zwei Datenleitungen benötigt.

Zur Vermeidung von Datenverlusten muss der Empfänger die Datenübertragung anhalten können, wenn keine weiteren Daten mehr verarbeitet werden können. Dieses kann mit einem Handshake auf zwei verschiedene Arten erreicht werden:

- **Hardware-Handshake:** Der Empfänger steuert über Steuer-Leitungen die Handshake-Eingänge CTS (Clear To Send) und/oder DSR (Data Send Ready) des Senders mit seinem Handshake-Ausgang DTR Data Transmit Ready) oder RTS (Ready To Send).
- **Software-Handshake:** Der Empfänger sendet zur Steuerung des Datenflusses spezielle Zeichen an den Sender (z.B. XON/XOFF).

[3] Vergleiche mit [2] Serielle Datenübertragung.

2.2.1 Bitratenquarz

Bei asynchronen Datenübertragungen wird für die Verbindung eine Bitrate vorgegeben. Diese muss möglichst exakt eingehalten werden, da sonst Fehlabtastungen des Signals auftreten können. Für Projekte mit asynchroner serieller Kommunikation ist es deshalb vorteilhaft Bitratenquarze als Taktgeber zu verwenden, aus denen sich durch ganzzahlige Teilung die Bitrate errechnen lässt. Übliche Bitraten sind unter anderem: 9.600, 115.200 oder 921.600 Bit pro Sekunde. Bitratenquarze haben Werte von zum Beispiel 1,8432, 7,3728 oder 14,7456 MHz.

2.2.2 SPI

Das Serial Peripheral Interface (SPI) ist ein synchroner serieller Datenbus mit dem digitale Schaltungen verbunden werden können. Alle Teilnehmer werden mit drei gemeinsamen Leitungen verbunden:

- SDO (Serial Data Out) oder MOSI (Master Out Slave In)
- SDI (Serial Data In) oder MISO (Master In Slave Out)
- SCLK (Serial Clock)

Für jeden Teilnehmer gibt es eine zusätzliche Leitung SS (Slave Select) oder CS (Chip Select), durch die bestimmt wird, welcher Teilnehmer aktiv ist. Es werden hohe Taktraten bis in den Megahertzbereich erreicht[4]. Ein Vollduplex-Betrieb ist möglich.

2.2.3 USB

Der Universal Serial Bus (USB) sendet Daten bitseriell über zwei verdrillte Leitungen. Das Signal wird dabei auf der einen Leitung unverändert und auf der anderen invertiert übertragen. Der Empfänger bildet aus beiden Signalen eine Differenzspannung, wodurch sich der Spannungshub der logischen Pegel verdoppelt. Da einstrahlende Störungen auf beide Leitungen in etwa gleichermaßen wirken, eliminieren sie sich durch dieses Verfahren weitgehend. Hierdurch wird die Übertragungssicherheit erhöht und hohe Bitraten bis über 12 MBit/s sind möglich. Zwei weitere Kabel dienen zur Spannungsversorgung der angeschlossenen Geräte.

---

[4] Siehe [3] Serial Peripheral interface (SPI).

Start-of-frame-Paket

Feld	PID	FrameNumber	CRC5
Bits	8	11	5

Token-Paket

Feld	PID	ADDR	ENDP	CRC5
Bits	8	7	4	5

Daten-Paket

Feld	PID	DATA	CRC16
Bits	8	0-8192	16

Handshake-Paket

Feld	PID
Bits	8

Bild 2: Pakete des USB-Protokolls

Das Protokoll[5] umfasst verschiedene Paketarten für unterschiedliche Aufgaben. Die Token-Pakete bestimmen eindeutig den Empfänger der nachfolgenden Datenpakete oder Sender von Datenpaketen. Die Start-of-Frame-Pakete werden durch den Host etwa jede Millisekunde versendet. Das Daten-Paket enthält die eigentlichen Daten, die versendet werden sollen. Durch die Handshake-Pakete kann der Status der Übertragung übermittelt werden.

2.3 Analog/Digital-Umsetzung

Ein analoges Signal wie in Bild 3 a) kann innerhalb bestimmter Grenzen theoretisch beliebig viele Werte annehmen. Damit hat es in der Theorie eine unendliche Auflösung und ist amplitudenkontinuierlich. Über die Zeit kontinuierliche Signale sind zu jedem Zeitpunkt eindeutig definiert. Mit zeitdiskreten Signalen (Bild 3 b) können auf nur einer Leitung meh-

[5] Quelle [4] Universal Serial Bus Specification, Seite 195 ff.

rere Signale übertragen werden[6]. Im Zuge dieser Arbeit kommen ausschließlich zeitkontinuierliche analoge Eingangssignale zur Anwendung.

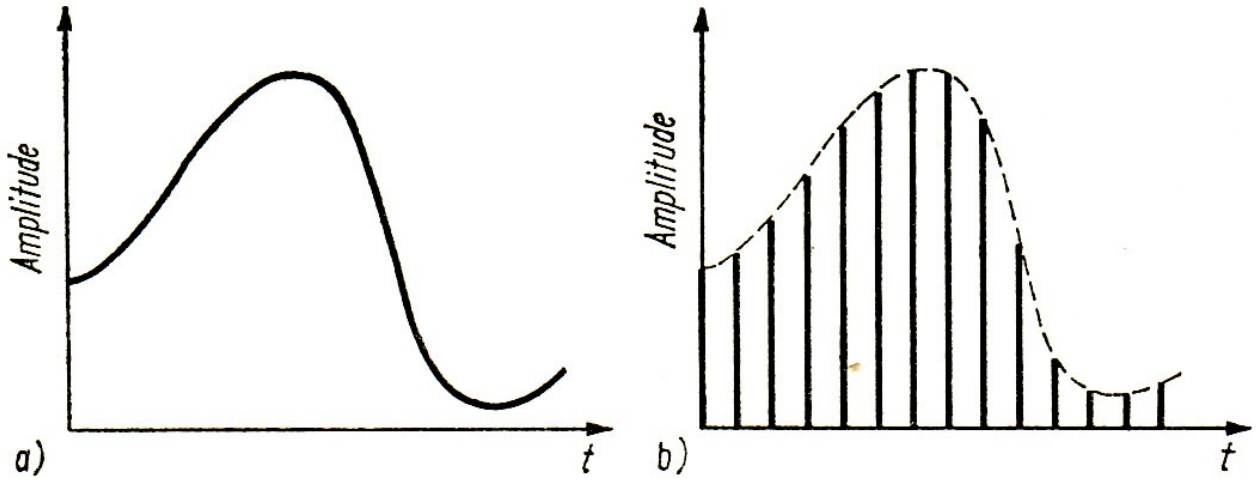


Bild 3: a) amplituden- und zeitkontinuierlich, b) amplitudenkontinuierlich, zeitdiskret

Die Amplitude eines digitalen Signals kann nur in endlich vielen Schritten dargestellt werden und ist somit amplitudendiskret. Durch definierte Zeitabstände zwischen den Messungen, die Abtastfrequenz, ist das digitale Signal ebenfalls zeitdiskret (Bild 4).

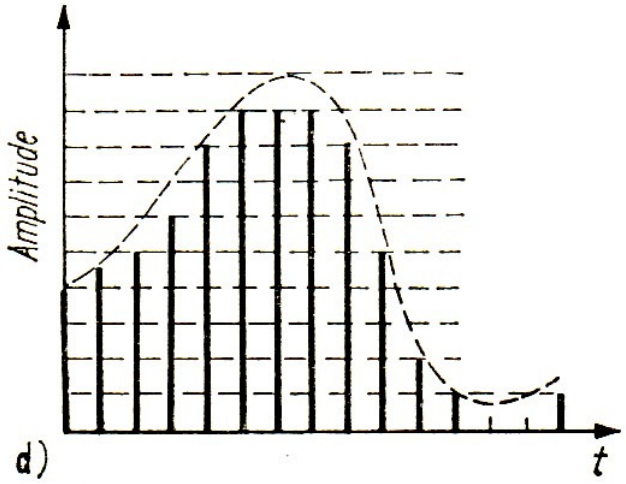


Bild 4: Amplituden- und Zeitdiskret

Bei der Analog- zu Digital- (A/D-) Umsetzung wird ein analoger Eingangswert in einen digitalen Wert umgesetzt. Entsprechend der digitalen Auflösung gehen von dem kontinuierlichen Eingangssignal Informationen verloren. Dadurch spielt die Wahl der Auflösung der

[6] Siehe [5] Digitale Schaltungen, Manfred Seifart.

A/D-Umsetzung eine wesentliche Rolle bei der Genauigkeit der Ergebnisse. Ebenso bedeutend ist die Geschwindigkeit der A/D-Umsetzung und der Datenauswertung, welche die minimale Frequenz der Abtastung bestimmen.

2.4 Operationsverstärker

Ein Operationsverstärker (OP) [7] ist ein mehrstufiger, hoch verstärkender Differenzverstärker, der sowohl Gleichspannungen als auch Wechselspannungen verstärken kann. Seine Wirkweise wird durch die äußere Gegenkopplungsbeschaltung beeinflusst. Der OP hat einen invertierenden und einen nichtinvertierenden Eingang. Entsprechend der Bezeichnung der Eingänge verhält sich die Verstärkung invertierend oder nichtinvertierend. Die Differenz der beiden Spannungen wird verstärkt auf den Ausgang ausgegeben. Durch eine entsprechende Beschaltung können mit den OPs verschiedenste Schaltungen wie Filterschaltungen und Reglerschaltungen realisiert werden. Die in den folgenden Kapiteln behandelten invertierenden Verstärker und Addierverstärker gehören zu den Grundschaltungen[8].

2.4.1 Operationsverstärker OP177

Der OP177[9] ist ein sehr präziser Operationsverstärker mit einer geringen Offsetspannung von maximal 60µV. Diese kann zusätzlich durch eine äußere Beschaltung kompensiert werden. Der OP177 wird für alle Verstärkerschaltungen auf der Platine verwendet, mit Ausnahme der eingangsseitigen Instrumentenverstärker.

2.4.2 Realer Verstärker - Offsetabgleich

Ein idealer Verstärker ist ein vereinfachtes Modell eines realen Verstärkers. Er hat einen unendlich großen Verstärkungsfaktor und Eingangswiderstand, einen Ausgangswiderstand gleich Null und einen Frequenzbereich von null bis unendlich. Dadurch besteht zwischen Eingangs- und Ausgangsspannung ein linearer Zusammenhang. Verzerrungen und

[7] Siehe [6] Operationsverstärkerschaltungen

[8] Siehe [7] Grundlagen zu Operationsverstärkern

[9] Siehe [8] OP177 Datenblatt, Seite 2

Rauschen treten nicht auf. Bei einem realen Verstärker gelten diese Vereinfachungen nicht. Durch die Wahl eines geeigneten Verstärkers und eine Einhaltung von entsprechenden Betriebsbereichen wird versucht, den realen Verstärker dem idealen anzugleichen. Mit zusätzlichen Kompensationsbeschaltungen wie der Offsetkompensation kann sein Verhalten verbessert werden.

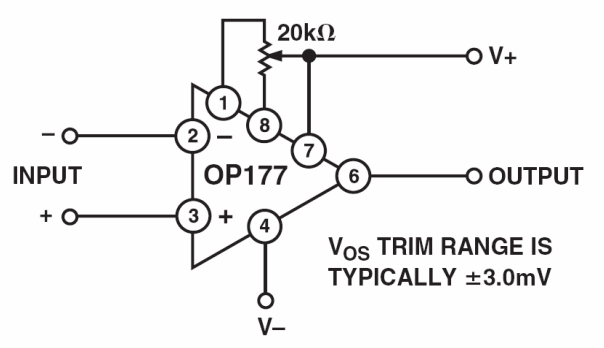
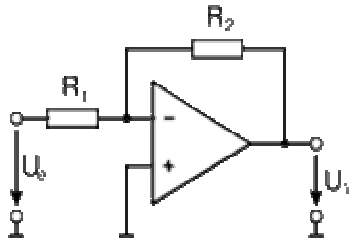


Bild 5: Offsetkompensation beim OP177[10]

Ein Offset tritt durch kleinste Basisströme durch die Eingangswiderstände und Asymmetrien in den symmetrischen Eingangsstufen des Operationsverstärkers auf.

2.4.3 Invertierender Verstärker

Eine Eingangsspannung U_e wird bei dem invertierenden Verstärker mit dem Verstärkungsfaktor v auf die Ausgangsspannung U_a verstärkt.



$$v = -\frac{R_2}{R_1}$$

$$U_a = v \cdot U_e = -\frac{R_2}{R_1} \cdot U_e$$

Bild 6: Invertierender Verstärker

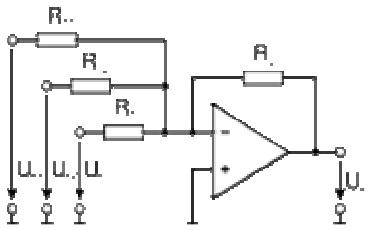
Der Ausgang des Operationsverstärkers wird so gesteuert, dass die Differenzspannung zwischen den beiden Eingängen null beträgt. Dadurch stellt sich am nichtinvertierenden

[10] Siehe [8] OP177 Datenblatt, Seite 4.

Eingang ein Massepotential ein, das als virtuelle Masse bezeichnet wird. Somit liegt der Widerstand R_1 zwischen Eingangsspannung und Masse, R_2 zwischen Ausgangsspannung und Masse. Zudem fließt kein Strom in den invertierenden Eingang des Operationsverstärkers, sondern ausschließlich über R_2.

2.4.4 Addierverstärker

Die Schaltung des Addierers erweitert den invertierenden Verstärker auf mehrere Eingänge. Eine beliebige Anzahl von Eingangsspannungen U_{e1}, U_{e2}, ..., U_{en}, die jeweils durch einen Eingangswiderstand gewichtet werden können, wird aufsummiert und verstärkt.

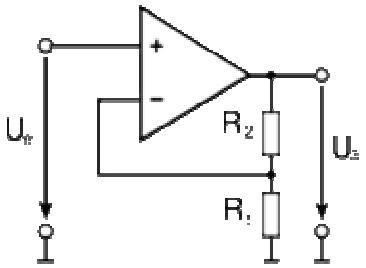


$$U_a = -R_2 \cdot \left(\frac{U_{e1}}{R_{11}} + \frac{U_{e2}}{R_{12}} + \cdots + \frac{U_{en}}{R_{1n}} \right)$$

Bild 7: Addierverstärker mit drei Eingängen

2.4.5 Nichtinvertierender Verstärker

Der nichtinvertierende Verstärker verändert die Polarität der Eingangsspannung im Gegensatz zum invertierenden Verstärker nicht. Durch einen Spannungsteiler wird ein Teil der Ausgangsspannung auf den invertierenden Eingang zurückgeführt. Durch die Regeln des Spannungsteilers und unter der Annahme, dass zwischen den beiden Eingängen des Operationsverstärkers die Spannung null beträgt, lässt sich die Verstärkung v und damit die Ausgangsspannung U_a bestimmen.



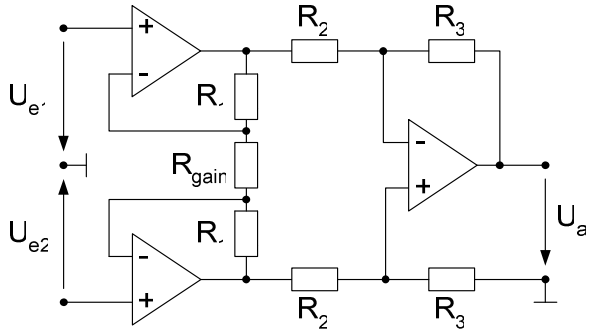
$$v = 1 + \frac{R_2}{R_1}$$

$$U_a = v \cdot U_e = \left(1 + \frac{R_2}{R_1}\right) \cdot U_e$$

Bild 8: Nichtinvertierender Verstärker

2.4.6 Instrumentenverstärker

Der Instrumentenverstärker ist eine Operationsverstärker-Schaltung, bestehend aus meistens drei Operationsverstärkern. Er hat sehr hochohmige Eingänge und eine geringe Offsetspannung. Die Eingänge des Instrumentenverstärkers sind von der Masse des Systems unabhängig, so dass keine Rückwirkungen auftreten können. Der Widerstand R_{gain} kann bei verschiedenen Instrumentenverstärkern extern beschaltet werden und somit die Verstärkung beeinflussen.



$$v = \left(1 + \frac{2 \cdot R_1}{R_{gain}}\right) \cdot \frac{R_3}{R_2}$$

$$v = \left(1 + \frac{2 \cdot R_1}{R_{gain}}\right) \quad \text{für } R_2 = R_3$$

$$U_a = v \cdot (U_{e2} - U_{e1}) = \left(1 + \frac{2 \cdot R_1}{R_{gain}}\right) \cdot (U_{e2} - U_{e1})$$

Bild 9: Instrumentenverstärker mit 3 OPs

3 Realisierung

In diesem Kapitel wird die Realisierung des x/y-Schreibers mit USB-Schnittstelle dargestellt. Zunächst wird der Aufbau der Hardware detailliert beschrieben. Sie besteht aus zwei unterschiedlichen Platinen. Zum Einen ist dies der vorschaltbare Verstärker, zum Anderen die Steuerung mit Datenaufnahme und Kommunikation. Durch die Teilung wird der sehr störungsanfällige analoge Verstärker von der digitalen Steuerung und Kommunikation getrennt. Diese Trennung wird auch in der Dokumentation beibehalten, beginnend mit dem analogen Verstärker und nachfolgend mit der Steuerung. Anschließend an die Hardware folgt die Beschreibung der PC-Software zur Darstellung und Verwaltung der Messergebnisse. Abschließend wird der Ablauf der Bedienung des x/y-Schreibers dargestellt.

3.1 Vorschaltbarer Verstärker

Die meisten A/D-Umsetzer haben einen Spannungsbereich von 0 bis 5 Volt. Der Maximalwert des analogen Eingangssignals des x/y-Schreibers soll jedoch verschiedene Werte von einigen Millivolt bis mindestens 5 V betragen können. Bei einem direkten Umsetzten der Signale würde sich, bezogen auf den Maximalwert, eine unterschiedliche Auflösung ergeben. Daher sollte eine vorherige Verstärkung des Eingangssignals erfolgen, so dass dessen Maximalwert möglichst dem des A/D-Umsetzers entspricht. Dadurch wird die volle Auflösung des Umsetzers für jede Messspannung genutzt.

3.1.1 Übersicht

Der vorschaltbare Verstärker besteht aus zwei identischen Schaltungen für die beiden analogen Eingangssignale (siehe Bild 10). Der Eingangswiderstand beider Verstärkerschaltungen beträgt 1 Megaohm, eine Schutzschaltung über eine Zener-Diode ist mit vorgesehen. Die Steuerung der Verstärkung[11] erfolgt für jeden Eingangskanal durch einen 4 Bit Demultiplexer, der analoge Schalter ansteuert. Durch diese Schalter wird der Abgriff des Ausgangssignals zwischen den einzelnen Verstärkerstufen bestimmt. Der erste Verstärker ist ein Instrumentenverstärker, um die beiden analogen Eingangssignale unabhängig von

[11] Vergleiche den Schaltplan in Anhang A.1.

der Masse des x/y-Schreibers, ausschließlich relativ zueinander, messen zu können. Alternativ kann der negative Eingang extern mit der Masse verbunden werden. Um eine möglichst hohe Genauigkeit zu erzielen, kann der Offset jedes Operationsverstärkers abgeglichen werden außerdem werden in den Verstärkerschaltungen ausschließlich Präzisionswiderstände mit einer maximalen Abweichung von 0,1% verwendet.

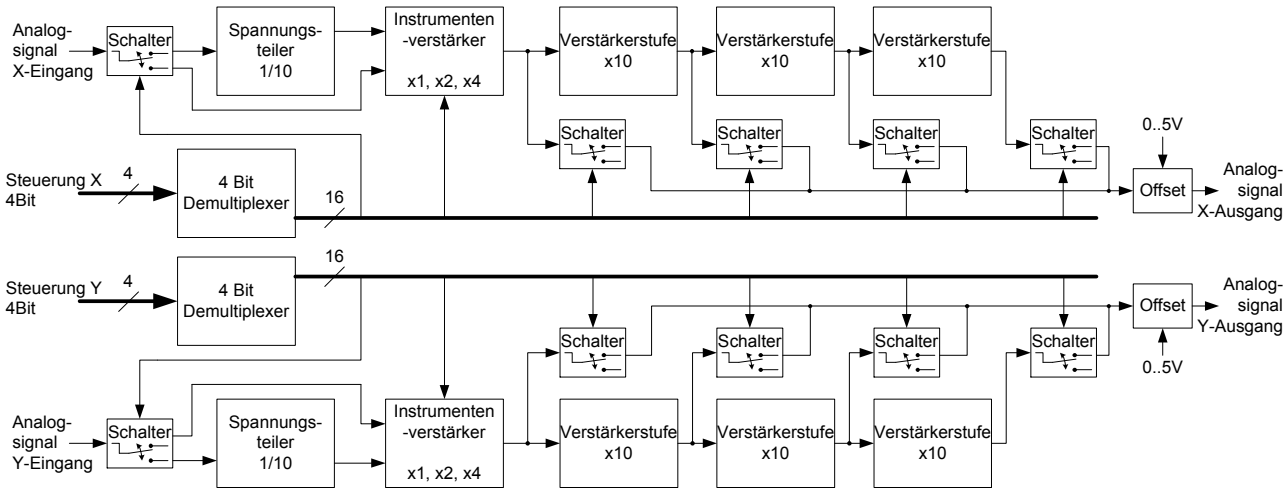


Bild 10: Blockschaltbild des vorschaltbaren Verstärkers

Für die Maximalspannung des Eingangssignals $U_{e,max}$ wird, angelehnt an die Bezeichnung beim analogen x/y-Schreibers PM8041[12], die Auflösung in mV/cm angegeben. Der Darstellungsbereich wird mit etwa 25 cm für ein DIN A4-Blatt vorgegeben. Für die maximale Auflösung des A/D-Umsetzers muss die Eingangsspannung mit den in Tabelle 1 angegebenen Faktoren auf 5 V verstärkt werden.

[12] Siehe [1] Gebrauchsanleitung A4 x-y Recorder.

Nr.	$U_{e,max}$ in mV	Auflösung in mV/cm bei 25cm Achse	Verstärkung auf $U_{a,max}$ = 5V
0	1,25	0,05	4000
1	2,5	0,1	2000
2	5	0,2	1000
3	12,5	0,5	400
4	25	1	200
5	50	2	100
6	125	4	40
7	250	10	20
8	500	20	10
9	1.250	40	4
10	2.500	100	2
11	5.000	200	1
12	12.500	400	0,4
13	25.000	1.000	0,2
14	50.000	2.000	0,1

Tabelle 1: Einstellung der Verstärkung

Die Einstellung der 15 verschiedenen Verstärkungen wird durch fünf kaskadierte Verstärkerstufen erreicht. Die erste Stufe ist ein Spannungsteiler, die zweite der Instrumentenverstärker. Es folgen drei Verstärkerstufen mit jeweils einer 10-fachen Verstärkung. Durch einen zuschaltbaren Offset auf beiden Kanälen kann das Ausgangssignal innerhalb des Messbereiches verschoben werden.

3.1.2 Einstellung der Verstärkung

Von den 16 Ausgängen des 4-Bit Demultiplexers HCF4067[13] werden die Ausgänge 0 bis 14 genutzt. Jeder Ausgang steuert über analoge Schalter den Stromfluss durch verschiedene Verstärkerstufen. Ein Zusammenschluss der Ausgänge bei gemeinsam genutzten Schaltern für eine Verstärkerstufe wird durch ODER-Gatter verhindert. Die erforderlichen Pull-Down-Widerstände an den Ausgängen des Demultiplexers wurden hier nicht dargestellt. Bild 11 zeigt einen Auszug der Beschaltung eines Demultiplexers.

[13] Siehe [9] HCF4067 Datenblatt

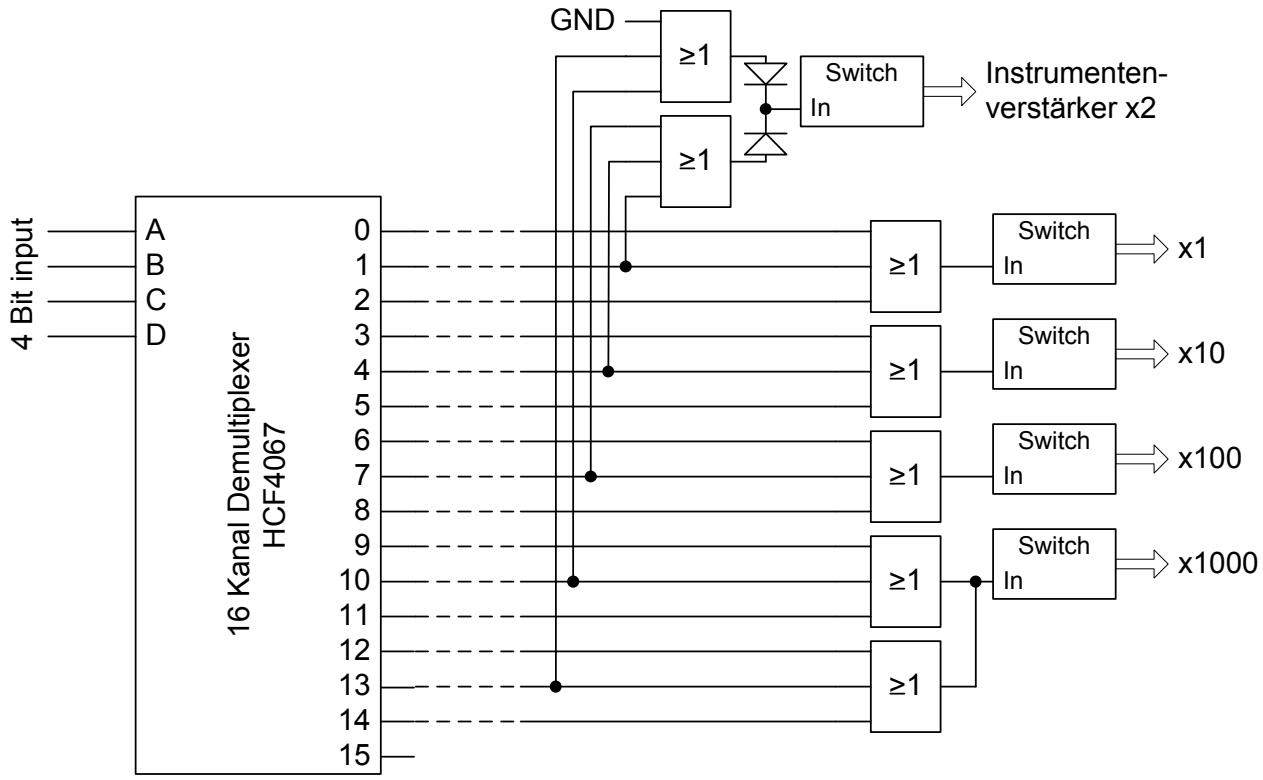


Bild 11: Auszug der Schaltung des Demultiplexers für einen Kanal[14]

Nr.	$U_{e,max}$ in mV	Verstärkung auf $U_{a,max}$ = 5V	Teilverstärkungen				
			Spgs.-teiler	Inst.-V. x1, 2, 4	Stufe 1	Stufe 2	Stufe 3
0	1,25	4000	1	4	10	10	10
1	2,5	2000	1	2	10	10	10
2	5	1000	1	1	10	10	10
3	12,5	400	1	4	10	10	
4	25	200	1	2	10	10	
5	50	100	1	1	10	10	
6	125	40	1	4	10		
7	250	20	1	2	10		
8	500	10	1	1	10		
9	1.250	4	1	4			
10	2.500	2	1	2			
11	5.000	1	1	1			
12	12.500	0,4	1/10	4			
13	25.000	0,2	1/10	2			
14	50.000	0,1	1/10	1			

Tabelle 2: Aufschlüsselung der Verstärkung in einzelne Stufen

[14] Für die vollständige Beschaltung ist auf den Schaltplan in Anhang A.1 verwiesen.

In Tabelle 2 ist die vollständige, durch den Demultiplexer gesteuerte, Verteilung der Gesamtverstärkung auf die einzelnen Verstärkerstufen und den Spannungsteiler dargestellt.

3.1.3 Analoger Schalter MAX313

Als analoger Schalter für die Einstellung der Verstärkung wird der IC MAX313 verwendet. Er beinhaltet vier analoge Schalter mit einem geringen Widerstand R_{DSON} bei Durchschaltung von maximal 10 Ω[15]. Vor allem für die Beschaltung des Instrumentenverstärkers ist dieser geringe Widerstand von Bedeutung, da er innerhalb der Verstärkerschaltung liegt und somit direkten Einfluss auf die Verstärkung nimmt (siehe Kapitel 3.1.5). Die Schaltzeiten haben keine große Bedeutung, da die Einstellung der Verstärkung vor der Messung durchgeführt und während der Aufnahme nicht mehr verändert wird.

3.1.4 Spannungsteiler / Eingangswiderstand

Die erste Verstärkerstufe ist ein 1:10 Spannungsteiler, der zur Reduzierung von hohen Eingangsspannungen vorgesehen ist, da durch eine Schutzschaltung nach dem Spannungsteiler eine Spannung von maximal 5 Volt möglich ist. Durch Schalter kann der Spannungsteiler zugeschaltet werden. Die Widerstände zwischen den zwei Eingängen beider Kanäle sind mit 900 kΩ und 100 kΩ so gewählt, dass ein Eingangswiderstand von 1 MΩ anliegt. Für die maximal (v_{max}) und minimal (v_{min}) mögliche Verstärkung und die relativen Fehler f ergeben sich bei der Verwendung von Präzisionswiderständen (0,1%)

$$v_{max} = \frac{R_{2,max}}{R_{1,min} + R_{2,max}} = \frac{100k\Omega \cdot 1{,}001}{\frac{900k\Omega}{1{,}001} + 100k\Omega \cdot 1{,}001} = 0{,}10018 \quad \Rightarrow \quad f = 0{,}18\%$$

$$v_{min} = \frac{R_{2,min}}{R_{1,max} + R_{2,min}} = \frac{100k\Omega / 1{,}001}{900k\Omega \cdot 1{,}001 + \frac{100k\Omega}{1{,}001}} = 0{,}09982 \quad \Rightarrow \quad f = -0{,}18\%$$

[15] Siehe [10] MAX313 Datenblatt , Seite 2.

3.1.5 Instrumentenverstärker INA114

Der Instrumentenverstärker INA114 ist, bedingt durch seine Bauweise mit drei OPs und getrimmten Widerständen[16], ein sehr präziser Verstärker. Durch die externe Beschaltung des R_{Gain} kann die Verstärkung variiert werden (siehe Bild 12).

Die Eingänge des Instrumentenverstärkers werden durch zwei in Reihe geschaltete 10 kΩ[17] Widerstände gegen hohe Ströme und durch eine Schutzschaltung über zwei Zener-Dioden gegen Überspannungen bzw. Spannungsspitzen, zum Beispiel durch elektrische Entladung, geschützt. Die Eingangsspannung ist mit den beiden Eingängen des Instrumentenverstärkers verbunden, so dass die Messspannung von der Masse der Schaltung unabhängig ist[18].

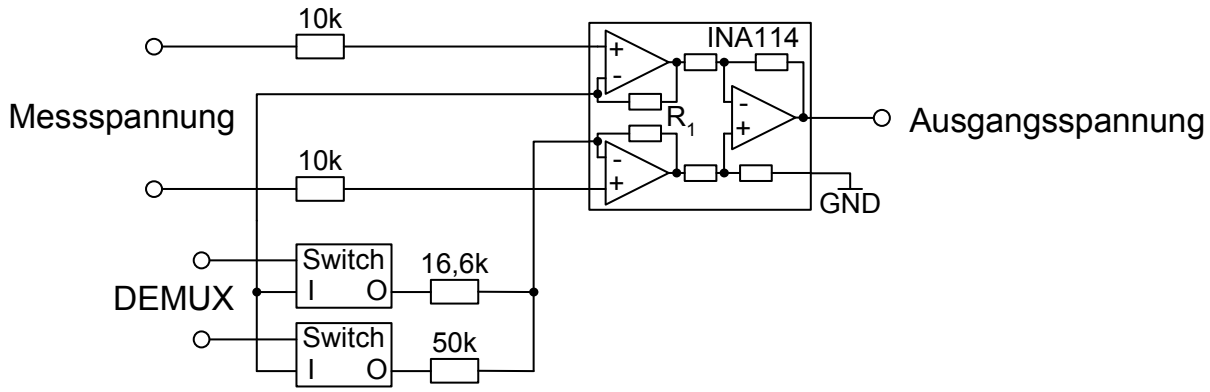


Bild 12: Beschaltung des Instrumentenverstärkers INA114

Durch zwei Schalter können drei verschiedene Widerstandswerte für R_{gain} (siehe Kapitel 2.4.6) eingestellt werden. Diese berechnen sich für eine Verstärkung v durch die Formel[19]

$$v = \left(1 + \frac{2 \cdot R_1}{R_{gain}}\right) \quad \Leftrightarrow \quad R_{gain} = \frac{2 \cdot R_1}{v - 1}.$$

Mit einem internen Widerstand von R_1 = 25 kΩ ergibt sich für die

- 1-fache Verstärkung: $R_{gain} = \frac{2 \cdot 25k\Omega}{1-1} = \infty$ (für einen Nenner gegen Null)

[16] Vergleiche Kapitel 2.4.6 Instrumentenverstärker.

[17] Im Schaltplan (Anhang A.1) sind 100kΩ vorgesehen, um die Stromsicherheit zu erhöhen.

[18] Extern kann der negative Eingang, wenn gewünscht, auf Masse gelegt werden.

[19] Siehe [11] INA114 Datenblatt, Seite 8.

- 2-fache Verstärkung: $R_{gain} = \frac{2 \cdot 25k\Omega}{2-1} = 50k\Omega$
- 4-fache Verstärkung: $R_{gain} = \frac{2 \cdot 25k\Omega}{4-1} = 16{,}66k\Omega$

Für die 4-fache Verstärkung wurden 16 kΩ und 680 Ω gewählt. Der Widerstand $R_{ON,typ}$ = 6,5 Ω[20] bzw. $R_{ON,max}$ = 10 Ω des Schalters MAX313 und eine Toleranz der Widerstände von 0,1% müssen für die Verstärkungen 2 und 4 mit einbezogen werden. Für die maximale (v_{max}) und minimale (v_{min}) Verstärkungen und die relativen Fehler f ergeben sich, bei Vorraussetzung eines idealen Instrumentenverstärkers mit

$$R_{gain,max} = R_{gain} \cdot 1{,}001 + 10\Omega$$

$$R_{gain,min} = \frac{R_{gain}}{1{,}001} + 6{,}5\Omega$$

die in Tabelle 3 dargestellten Fehler.

Verstärkung	$R_{gain,max}$	v_{min}	f	$R_{gain,min}$	v_{max}	f
2	50.060 Ω	-1,9988	-0,06%	49.956,5 Ω	2,0009	0,04%
4	16.706,68 Ω	-3,9928	-0,18%	16.669,84 Ω	3,9994	-0,01%

Tabelle 3: Fehler durch die Instrumentenverstärkerbeschaltung

3.1.6 Offset

Der zuschaltbare Offset wird durch einen invertierenden Addierverstärker (siehe Kapitel 2.4.4) und einen anschließenden Inverter (invertierender Verstärker mit Verstärkung 1, siehe Kapitel 2.4.3) mit jeweils einem Operationsverstärker OP177 realisiert. Die Offsetspannung wird zu der Ausgangsspannung der Verstärkerschaltung kurz von dem A/D-Umsetzer addiert. Die Verstärkung berechnet sich aus

$$v_{Inverter} = -\frac{R_2}{R_1} \quad \text{und} \quad v_{Addierer} = -R_2 \cdot \left(\frac{0{,}5}{R_{11}} + \frac{0{,}5}{R_{12}} \right)$$

Alle Widerstände haben einen Wert von 100 kΩ mit einer Toleranz von 0,1%. Für die maximalen (v_{max}) Verstärkungen und die symmetrischen relativen Fehler f ergeben sich mit

[20] Der minimale Durchschaltwiderstand des MAX313 wird nicht angegeben, daher wird als Minimum der typische Widerstandswert in der Fehlerberechnung angenommen. Der Verstärker an sich wird als ideal angenommen.

$$v_{Inverter,max} = -\frac{R_{2,max}}{R_{1,min}}, \qquad v_{Addierer,max} = -R_{2,max} \cdot \left(\frac{0,5}{R_{11,min}} + \frac{0,5}{R_{12,min}} \right)$$

die errechneten Fehler in folgender Tabelle.

$v_{Inverter,max}$	f	$v_{Addierer,max}$	f
-1,0020	-0,2%	-1,002	0,2%

Der einstellbare Offset zwischen 0 und 5 Volt kann durch das Umsetzen von zwei Jumpern für jeden Kanal, wie in Bild 13 dargestellt, aktiviert oder deaktiviert werden.

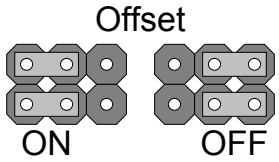


Bild 13: Aktivierung und Deaktivierung des Offsets

3.1.7 Verstärkerstufe

Für beide Kanäle werden drei nichtinvertierende Verstärkerstufen (siehe Kapitel 2.4.5) mit jeweils 10-facher Verstärkung verwendet. Dadurch wird eine Verstärkung um 10, 100 oder 1000 des Eingangssignals erreicht. Die Verstärkung berechnet sich aus

$$v = 1 + \frac{R_2}{R_1}$$

Alle Widerstände haben eine Toleranz von 0,1%. Für die maximale (v_{max}) Verstärkungen und die symmetrischen relativen Fehler f ergeben sich

$$v_{max} = 1 + \frac{R_{2,max}}{R_{1,min}} = 1 + \frac{(430k\Omega + 470k\Omega) \cdot 1,001}{100k\Omega / 1,001} = 1 + \frac{900,900k\Omega}{99,900k\Omega} = 10,018 .$$

$$f = \pm 0,18\%$$

3.1.8 Widerstandstoleranzbedingte Fehler

Die Berechnung des maximalen Verstärkungsfehlers geht von jeweils idealen Operationsverstärkern aus. In die Berechnung fließen lediglich baubedingte Ungenauigkeiten und die

Toleranzen der Widerstände ein. Der Fehler muss für die verschiedenen Verstärkungseinstellungen einzeln berechnet werden, da die unterschiedlichen Verstärkungen des Instrumentenverstärkers unterschiedliche Fehler aufweisen und nicht immer alle Stufen genutzt werden. Tabelle 4 listet die prozentualen Fehler der einzelnen Stufen der Verstärkerschaltung auf und addiert sie zu einem maximalen Gesamtfehler für die resultierende Verstärkung.

Verstärkung	Teilverstärkungen in %						Gesamtfehler in %	
	Spgs.-teiler	Inst.-Ver. x1, 2, 4		Stufe 1	Stufe 2	Stufe 3	min	max
		min	max					
4000		-0,18	-0,01	±0,18	±0,18	±0,18	-0,72	0,53
2000		-0,06	0,04	±0,18	±0,18	±0,18	-0,60	0,58
1000				±0,18	±0,18	±0,18	-0,54	0,54
400		-0,18	-0,01	±0,18	±0,18		-0,54	0,35
200		-0,06	0,04	±0,18	±0,18		-0,30	0,40
100				±0,18	±0,18		-0,36	0,36
40		-0,18	-0,01	±0,18			-0,36	0,17
20		-0,06	0,04	±0,18			-0,24	0,22
10				±0,18			-0,18	0,18
4		-0,18	-0,01				-0,18	-0,01
2		-0,06	0,04				-0,06	0,04
1							0	0
0,4	±0,18	-0,18	-0,01				-0,36	0.17
0,2	±0,18	-0,06	0,04				-0,24	0,22
0,1	±0,18						-0,18	0,18

Tabelle 4: Gesamtfehler der Verstärkung aufgeteilt auf einzelne Komponenten

Der durch Widerstandsungenauigkeiten bedingte Gesamtfehler der Verstärkungen, ist mit unter 0,75% sehr gering. Mit jedem Zuschalten einer Verstärkerstufe nimmt er zu.

3.1.9 Anschlüsse

Die Anschlüsse der Verstärkerplatine bestehen aus einer 1x8 und einer 2x8 Stiftleiste mit Rastermaß 2,54 mm. Die 1x8 Stiftleiste stellt Anschlüsse für die Messspannungen und mehrere Masseverbindungen zur Verfügung. Die 2x8 Stiftleiste ist für die Verbindung mit der Steuer- und Versorgungsplatine vorgesehen. Sie hat Anschlüsse für die Versorgungsspannungen, die zwei 4 Bit Signale zur Ansteuerung der Demultiplexer und die Ausgänge der verstärkten Messspannungen.

Pin	Beschreibung
1	Analog in 1+
2	Analog in 1-
3	Masse
4	Masse
5	Analog in 2+
6	Analog in 2-
7	Masse
8	Masse

Tabelle 5: Stiftleiste 1x8

Pin	Beschreibung	Pin	Beschreibung
1	+12 Volt	2	-12 Volt
3	+5 Volt	4	Masse
5	Analog out 1	6	Offset 1
7	Analog out 2	8	Offset 2
9	DEMUX 1 Bit 0	10	DEMUX 2 Bit 0
11	DEMUX 1 Bit 1	12	DEMUX 2 Bit 1
13	DEMUX 1 Bit 2	14	DEMUX 2 Bit 2
15	DEMUX 1 Bit 3	16	DEMUX 2 Bit 3

Tabelle 6: Stiftleiste 2x8 „Platine“

3.1.10 Platine

Der Schaltplan[21] und das Layout[22] der Platine wurden mit dem Programm Eagle™ erstellt[23]. Bild 14 zeigt die auf dieser Grundlage gefertigte bestückte Platine.

[21] Siehe Anhang A.1.

[22] Siehe Anhang A.2.

[23] Für eine kurze Übersicht des Programms siehe Anhang A. Für weitergehende Informationen sei auf das [12] EAGLE Handbuch verwiesen.

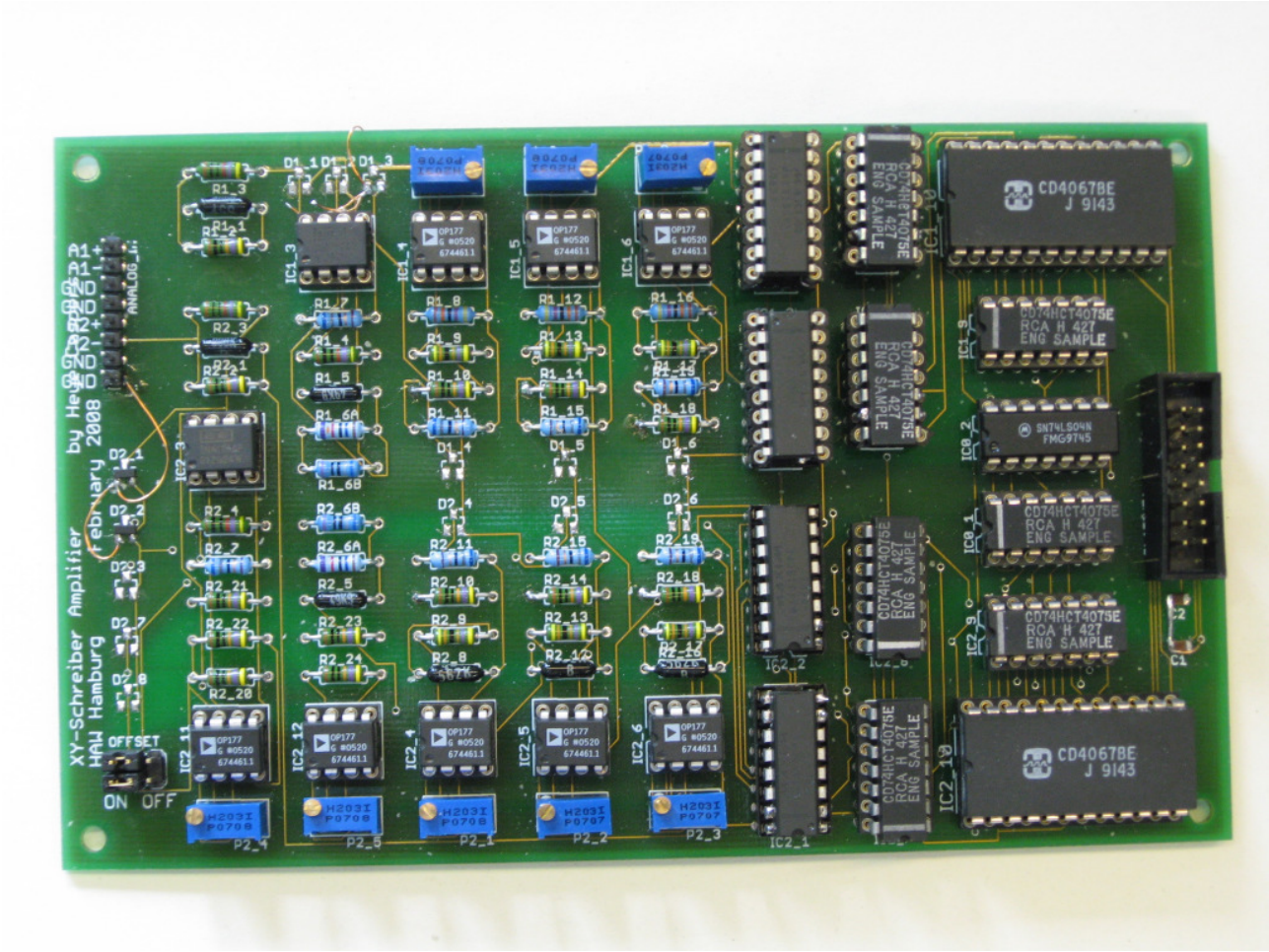


Bild 14: Prototyp der Verstärkerplatine

Kleinere Fehler des Prototypen wurden in dem im Anhang befindlichen Schaltplan und Layout berichtigt. Zudem ist eine Offsetschaltung für den x-Kanal hinzugefügt worden.

3.2 Steuerung

Der mit Steuerung bezeichnete Teil ist der Hauptteil des x/y-Schreibers. Er ist ohne die Verstärkerplatine funktionsfähig, kann jedoch nur Spannungen zwischen 0 und 5 V vernünftig aufgelöst auswerten. Der Mikroprozessor steuert die Datenaufnahme über den A/D-Umsetzer (ADC) MCP3202 und kommuniziert mit dem PC über den USB-Converter FT232RL. Die Einstellung der Auflösung und des Modus erfolgt durch Taster und Schalter an jeweils einem Prozessoreingang und wird auf dem LC-Display dargestellt. Spannungsversorgung und zwei 4 Bit Signale zur Verstärkersteuerung sind für einen vorschaltbaren Verstärker vorgesehen. Die Spannungsversorgung der Steuerplatine sollte mindestens

7 Volt betragen. Wird die Verstärkerplatine mit versorgt, werden etwa +15 und -15 Volt benötig. Durch Spannungsregler werden +5V mit einem Maximalstrom von 1A und plus/minus 12 Volt für die Operationsverstärker der Verstärkerschaltung erzeugt. Der vollständige Schaltplan ist in Anhang A.3 abgebildet.

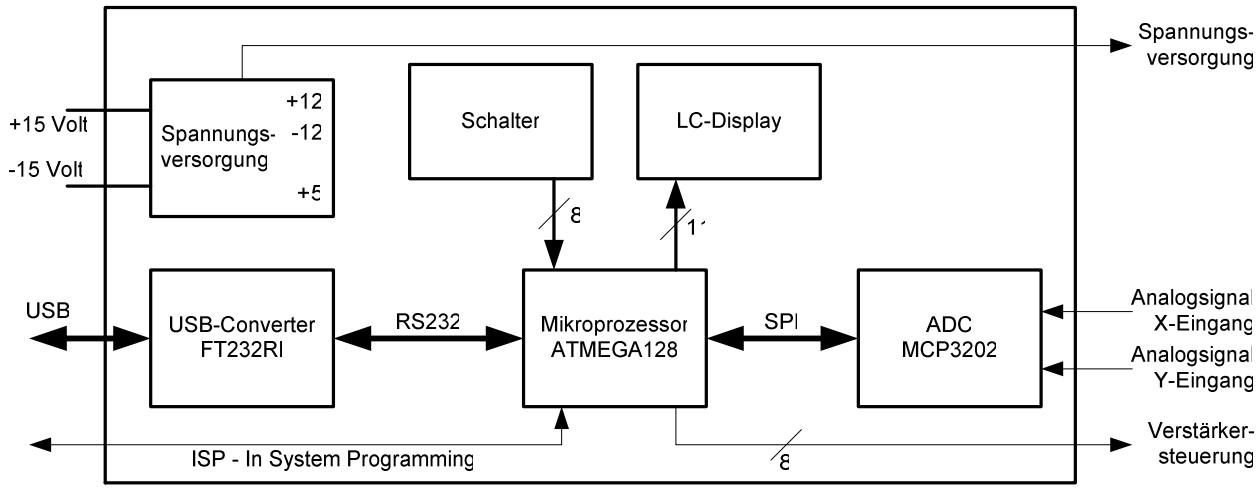


Bild 15: Blockschaltbild der Steuerung

3.2.1 Mikroprozessor

Der verwendete 8-Bit Mikroprozessor ATMEGA128[24] von Atmel bietet mit 128 KByte Speicher, vielen peripheren Funktionen und bis zu 53 voll programmierbaren Eingängen und Ausgängen (I/Os) einen großen Funktionsumfang. In diesem Projekt werden hiervon ein 8 Bit und ein 16 Bit Zähler, der Analog zu Digital Umsetzer, der serielle UART, das SPI Interface sowie verschiedene I/Os genutzt.

Die Funktion des x/y-Schreibers kann man in drei Phasen aufteilen. Erstens, als Einstiegspunkt, die Funktion main() mit der Initialisierung der peripheren Funktionen und der I/Os des Mikroprozessors. In der zweiten Phase werden die Einstellungen für die in der dritten Phase folgende Datenerfassung und Übertragung vorgenommen[25].

[24] Siehe [13] ATMEGA128 Datenblatt.

[25] Als Anleitung zur Programmierung eines Atmel Mikroprozessors wurde das [15] AVR-GCC-Tutorial verwendet. Der gesamte Quellcode ist in Anhang C abgelegt.

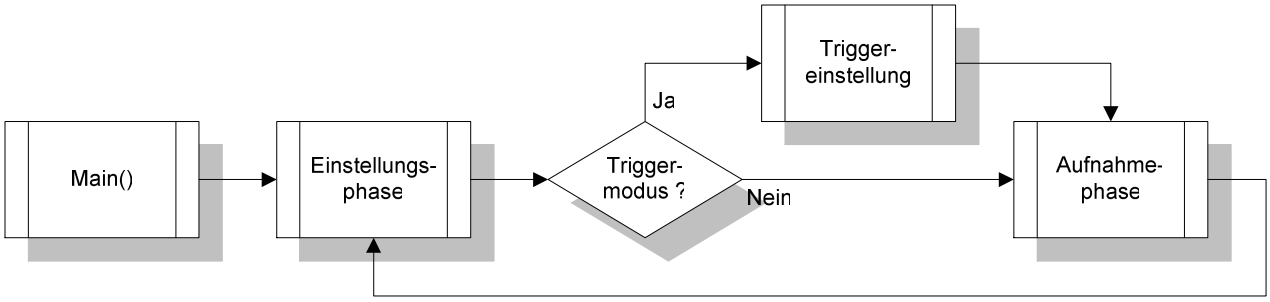


Bild 16: Übersicht des Programmablaufs

Bild 17 zeigt den Ablauf der ersten beiden Phasen detaillierter. Zunächst wird die Initialisierung der I/Os vorgenommen und Standardwerte auf dem LC-Display[26] ausgegeben. Die Einstellung des Modus und der Verstärkung der Aufnahme erfolgt in einer Endlosschleife, in der eine Abfrage der einzelnen Taster durchgeführt wird. Die Einstellungen werden entsprechend der Betätigung der Taster verändert, auf dem Display ausgegeben und über USB versendet. Durch die Taste „Start“ wird die Einstellungsphase verlassen und die Funktion zur Aufnahme von Werten aufgerufen. Im Zeitmodus mit Triggerauslösung wird zuvor noch die Triggerwerteinstellung durchgeführt, welche durch ein erneutes Drücken von „Start“ bestätigt wird.
Auf die Funktion der Tastenerkennung, den A/D-Umsetzer und die Kommunikation über den RS232 zu USB Konverter, wird in den folgenden Kapiteln unter anderem mit Codeausschnitten genauer eingegangen.

[26] Das [16] LCD162C Datenblatt und das [22] AVR-Tutorial: LCD wurden als Anleitung für die Programmierung zu Rate gezogen.

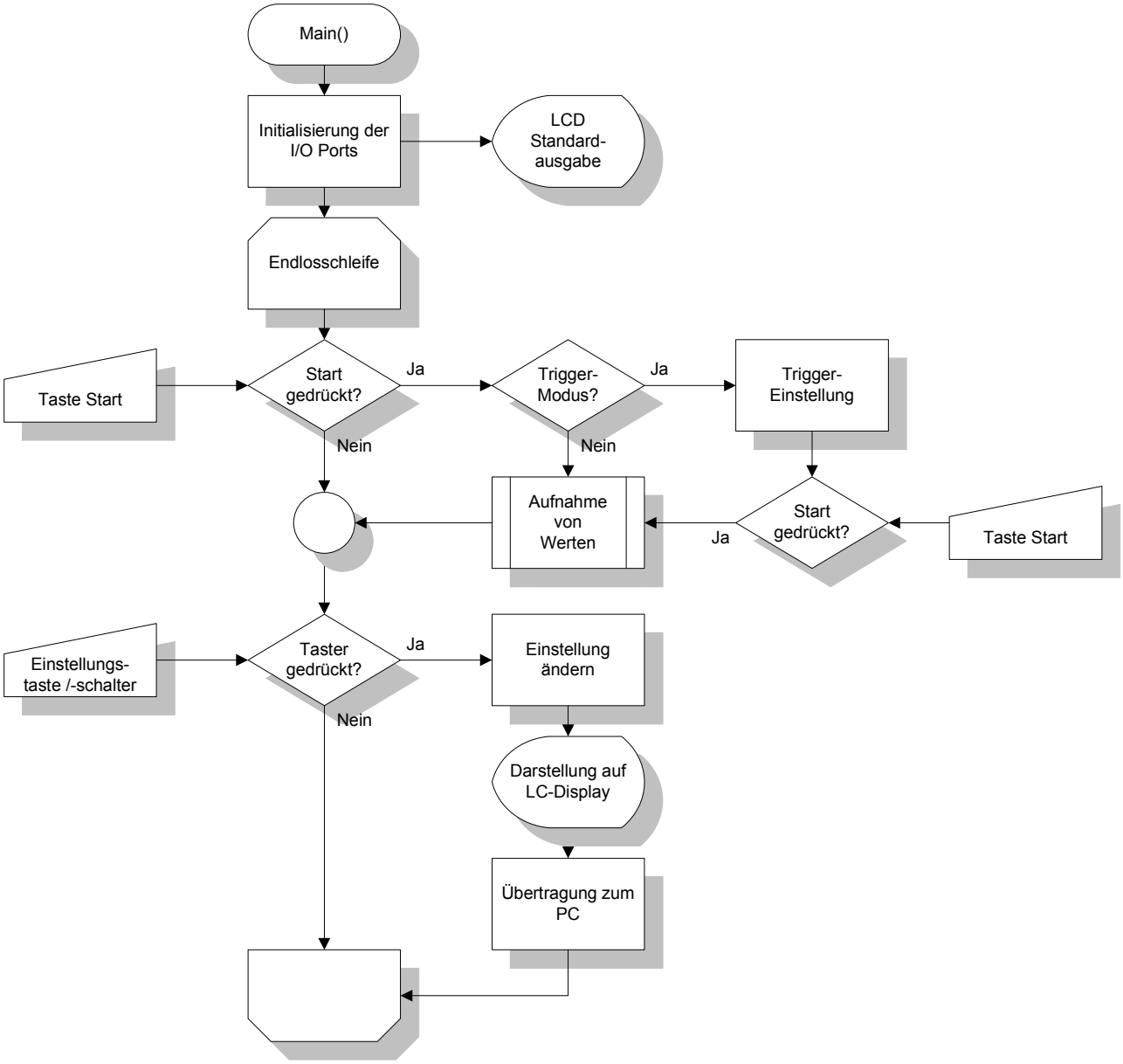


Bild 17: Programmablaufplan des Programmeinstiegs und der Einstellungsphase

Der Ablauf der Aufnahme von Werten ist in Bild 18 dargestellt. Dieser wird durch einen Tastendruck auf „Start“, oder auch im Zeitbetrieb durch den Ablauf der eingestellten Laufzeit, beendet und kehrt in die Einstellungsphase zurück.

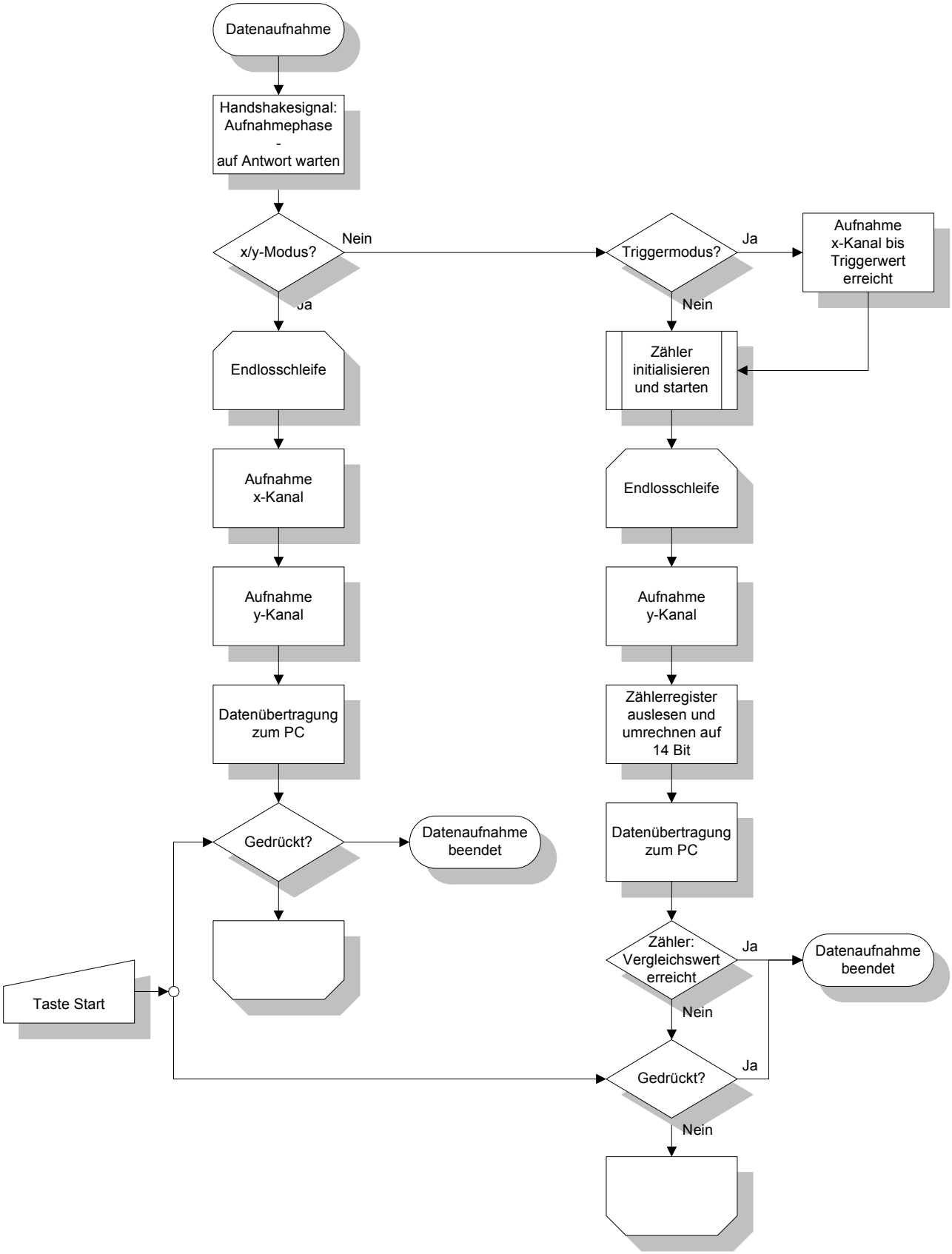


Bild 18: Programmablaufplan der Aufnahme von Daten

Zunächst wird ein Handshake-Signal gesendet, welches dem PC-Programm den Wechsel von der Einstellungs- in die Aufnahmephase mitteilt. Als Antwort wird der RTS# - Ausgang[27] des FT232RL – Bausteins gesetzt, womit die Empfangsbereitschaft erklärt wird. Es folgt entsprechend des eingestellten Modus der Start des x/y- oder Zeitbetriebes. Im x/y-Modus werden in einer Endlosschleife die beiden Kanäle des A/D-Umsetzers nacheinander ausgewertet und als Datenpaar an den PC übertragen. Im Zeitmodus wird nach einer eventuellen Verzögerung durch den Trigger der Zähler initialisiert und gestartet. Der y-Kanal wird mit dem aktuellen Zählerwert, der auf 14 Bit mit einem Maximalwert von 16384 skaliert wird, aufgenommen und übertragen. Auf einen Handshake zur Übertragung der einzelnen Datenpaare wird bewusst verzichtet, um die Frequenz der Datenaufnahme nicht zu verringern. In den durchgeführten Tests in Kapitel 4 zeigten sich keine Datenverluste.

3.2.2 Taster und Schalter

Die Einstellung der Auflösung der beiden Achsen des x/y-Schreibers erfolgt durch vier Taster, die die Auflösung jeweils erhöhen oder verringern. Die Datenaufnahme wird ebenfalls durch einen Taster gestartet oder beendet. Taster „Prellen“ wie alle mechanischen Schalter beim Ein- und Ausschalten[28]. Ein Schaltvorgang kann schematisch dargestellt wie folgt aussehen:

Bild 19: Schematische Darstellung eines Schaltvorganges

[27] Funktionsweise und Beschreibung in [17] FT232R Datenblatt, Seite 8.

[28] Weitere Informationen sind in [18] Entprellung zu finden.

Wenn ein Mikroprozessor dieses Signal auswertet, kann es vorkommen, dass jede durch das Prellen bedingte Pegeländerung über die Schaltschwelle als eigener Schaltvorgang erkannt wird. Dadurch ist eine präzise Einstellung kaum mehr möglich. Für die Entprellung von Schaltern gibt es sowohl Hardware-, als auch Softwarelösungen. Als Hardwarelösung kann ein Tiefpassfilter mit anschließendem Schmitt-Trigger genutzt werden. Die hohen Frequenzen des Prellvorganges werden so herausgefiltert. Bei Verwendung eines Mikroprozessors kann man auf die zusätzliche Hardware verzichten und eine Softwarelösung verwenden. Die einfachste Möglichkeit der Entprellung ist die Auswertung der Dauer einer Schalterbetätigung durch einen Timer. Dabei startet der erste Schaltvorgang den Zähler. Beim zweiten wird der aktuelle Wert des Zählers ausgewertet. Liegt dieser zum Beispiel unter 100 ms wird der Schaltvorgang als Prellen erkannt und nur ein längerer Tastendruck wird gezählt. Der Nachteil dieser Methode liegt in dem Wartezustand, in dem sich der Mikroprozessor befindet, solange der Zähler läuft. In der Zwischenzeit ist er für andere Aufgaben blockiert.
Vorteilhafter ist ein Verfahren, bei dem durch einen Zähler mit Overflow-Interrupt ein Basistakt erzeugt wird. In der Interruptroutine wird dann der am Port anliegende Tastenzustand ausgewertet und mit dem Zustand beim letzten Takt verglichen. Tritt eine Änderung auf, wird diese Flanke erkannt. Erst wenn dieser Zustand in einer bestimmten Anzahl folgender Takte beibehalten wird, wird die Zustandsänderung als Schaltvorgang gewertet. Die Aufrufe der Interruptroutine ohne Zustandsänderung werden intern gezählt.

```
ISR(TIMER0_COMP_vect)        // every 10ms
{
    static uint8_t ct0, ct1, rpt;
    uint8_t i;

    i = key_state ^ ~KEY_PIN;    // key changed ?
    ct0 = ~(ct0 & i);        // count bit ct0 (2Bit for 4 turns)
    ct1 = ct0 ^ (ct1 & i);       // count bit 1 of ct
    i &= ct0 & ct1;          // count until overrun
    key_state ^= i;          // if overrun toggle debounced state
    key_press |= key_state & i;  // 0->1: key press detected
}
```

Listing 1: Interruptroutine der Tastenentprellung

Zusätzlich zu der eigentlichen Tastenentprellung wird noch als Zusatzfunktion die Erkennung eines langen Tastendrucks vorgesehen. Hiermit kann ein schneller Durchlauf bei Halten des Tasters durchgeführt werden.

Die Auswahl des Betriebsmodus x/y, t/y (Zeit) oder t/y mit Trigger erfolgt über einen Schalter mit den drei Schaltstellungen Ein-Aus-Ein. Eine Entprellung ist hier nicht erforderlich, da nicht einzelne Schaltvorgänge ausgewertet werden, sondern ein konstantes Signal anliegt. Der Offset wird durch Potentiometer zwischen 0 und 5 Volt eingestellt. Die eingestellte Spannung wird durch den internen ADC des ATMEGA128 gemessen und zur Einstellung auf dem Display angezeigt.

3.2.3 Analog/Digital-Umsetzer MCP3202

Der MCP3202[29] ist ein Zweikanal, 12 Bit A/D-Umsetzer mit einer, das SPI-Protokoll unterstützenden, seriellen Schnittstelle.

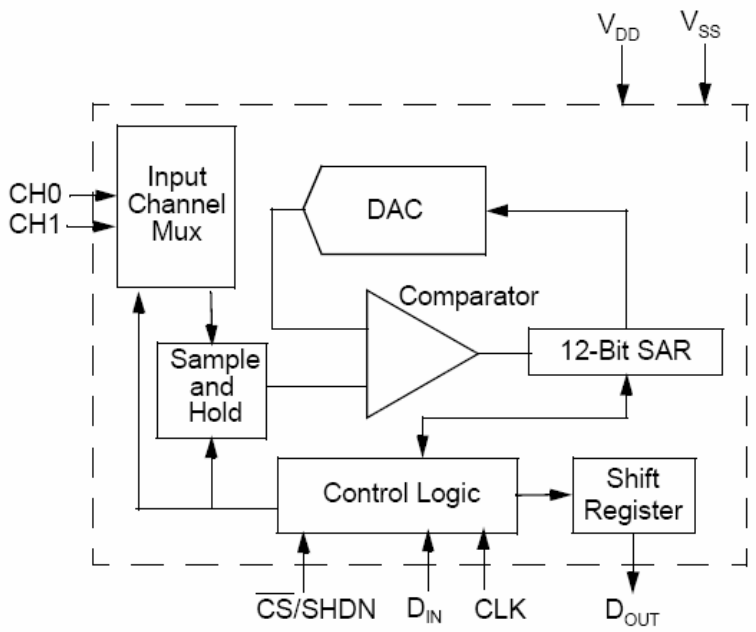


Bild 20: Funktionales Blockdiagramm des MCP3202[30]

Bei einer Eingangsspannungsdifferenz (U_{Diff}) von 5 Volt folgt für eine digitale Auflösung von 12 Bit mit 4096 Werten (d_{Aufl}) eine Stufe (ΔU) von

$$\Delta U = \frac{U_{Diff}}{d_{Aufl}} = \frac{5V}{4096} = 1{,}22mV .$$

[29] Für die Hard- und Softwarebeschreibung siehe [19] MCP3202 Datenblatt.

[30] Bild entnommen aus [19] MCP3202 Datenblatt, Seite 1.

Bei einem maximalen Takt (CLK) von 1,8MHz bei einer Versorgungsspannung von 5V beträgt die Dauer einer reinen Umsetzung (T_U) mit 12 Takten für die 12 Bit

$$T_U = \frac{12}{1{,}8\text{MHz}} = 6{,}67\mu\text{s}\,.$$

Die Frequenz der Umsetzungen wird zudem von der Übertragung der Daten und dem Kommunikationsprotokoll bestimmt. Jede Anfrage erfolgt durch 4 Bits, dem Start- und drei Steuerbits. Die Übertragung des MCP3202 erfolgt mit dem nächsten Takt mit einem Null- und den 12 Datenbits. Mit jedem Taktimpuls wird ein Bit gesendet. Insgesamt werden demnach 17 Bits in 17 Takten übertragen. Bis zur nächsten Anfrage muss das „Chip Select-“ (CS-) Signal zusätzlich für mindestens T_{CSH} = 500 ns deaktiviert sein. Daraus ergibt sich eine Gesamtdauer für eine Umsetzung (T_{CYC}) von

$$T_{CYC} = \frac{17}{\text{CLK}} + T_{CSH} = \frac{17}{1{,}8\text{MHz}} + 500\text{ns} = 9{,}94\mu\text{s}\,.$$

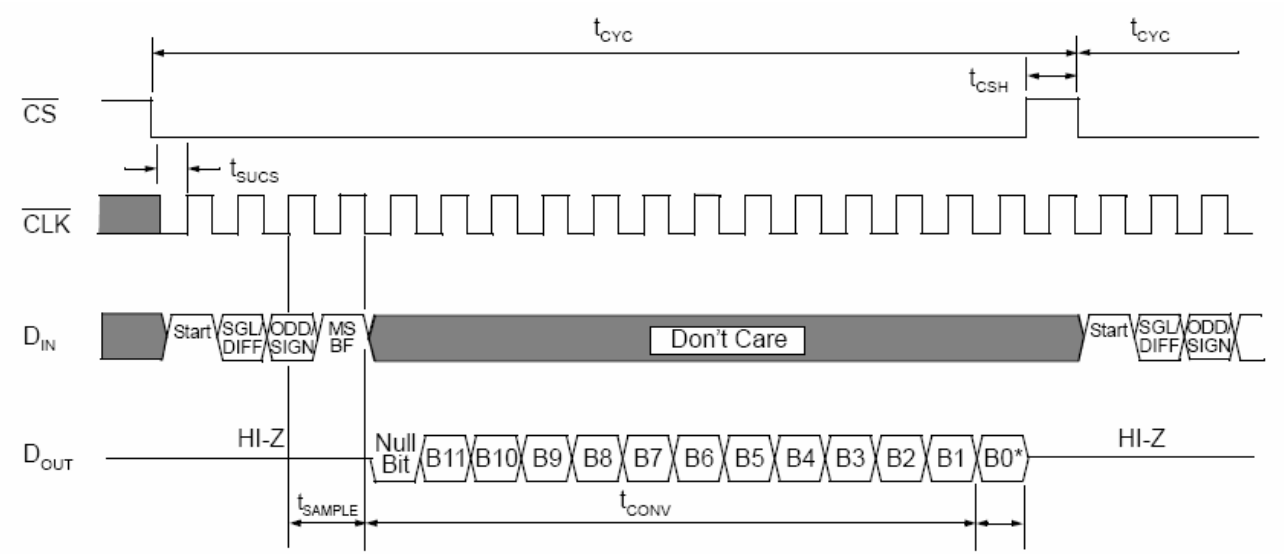


Bild 21: Kommunikation mit dem MCP3202[31]

Bei der Verwendung eines Mikroprozessors mit dem MCP3202 werden in der Regel Gruppen von 8 Bit versendet. Dies wirkt sich vor allem auf das Startbit aus, das in einer Gruppe von 8 Bit versendet wird (siehe Bild 22). Im restlichen Ablauf der Kommunikation werden zwischen dem Versenden einzelner 8 Bit Gruppen noch zwei zusätzliche Takte

[31] Bild entnommen aus [19] MCP3202 Datenblatt, Seite 13.

verloren, so dass sich die Umsetzzeit bei Verwendung eines Mikroprozessors ($T_{CYC,\mu P}$) noch einmal um 9 Taktzyklen erhöht:

$$T_{CYC,\mu P} = \frac{17+9}{CLK} + T_{CSH} = \frac{26}{1{,}8MHz} + 500ns = 14{,}94\mu s\,.$$

Der SPI CLK ist in der Regel der Takt des Mikroprozessors geteilt durch eine Zweierpotenz. Bei der Verwendung eines Bitratenquarzes mit der Frequenz von 14,7456 MHz ergibt sich ein SPI CLK von entweder 1,8432 MHz mit Teiler 8 oder 0,9216 MHz mit Teiler 16. Die Frequenz mit dem Teiler 8 ist minimal höher als die im Datenblatt des MCP3202 mit 1,8 MHz angegebene Maximalfrequenz.

$$T_{CYC,\mu P} = \frac{26}{1{,}8432MHz} + 500ns = 14{,}61\mu s$$

Da nicht davon ausgegangen werden kann, dass alle Bauteile sich gleich verhalten, sollte mit der im Datenblatt garantierten Frequenz und dem Teiler 16 gearbeitet werden, was die Verarbeitungszeit praktisch verdoppelt.

$$T_{CYC,\mu P} = \frac{26}{0{,}9216MHz} + 500ns = 28{,}71\mu s$$

Die vollständige Umsetzung mit Datenübertragung bei Verwendung eines Mikroprozessors mit einem Bitratenquarz der Frequenz 14,7456 MHz erhöht sich auf 28,71µs. Beide Kanäle können also in 57,42µs umgesetzt werden.

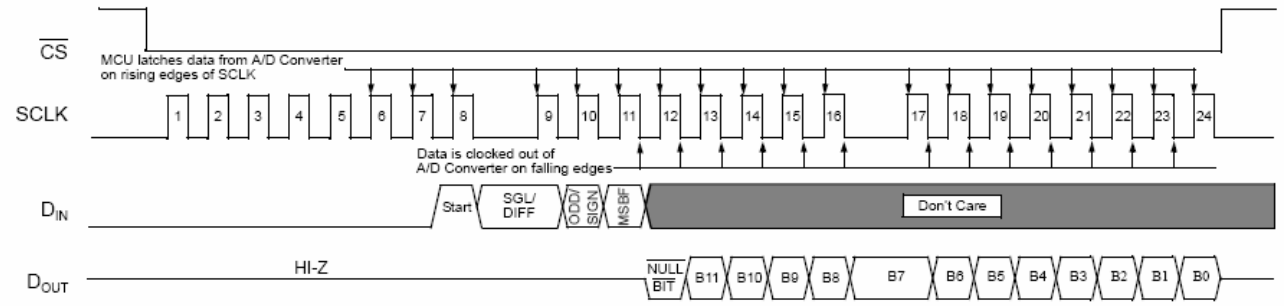


Bild 22 Kommunikation in 8 Bit Segmenten (Mikroprozessor) mit dem MCP3202[32]

Eine Wertabfrage erfolgt wie in Listing 2 mit dem Mikroprozessor ATMEGA128 über SPI in 8 Bit-Sequenzen. Wie oben beschrieben erfolgt zunächst die Übermittlung des Startbits

[32] Bild entnommen aus [19] ,MCP3202 Datenblatt, Seite 15.

und dann der Empfang der beiden Datenbytes. Diese werden zu einem 12 Bit-Wert konvertiert.

```
unsigned short mcp3202_get_value (unsigned char channel)
{
    unsigned short byte_rec = 0;

    SPI_PORT &= ~(1<<AD_CS); // select device
    SPDR = 0x01;          // send start bit (8-bit sequence)
    while(!(SPSR & (1<<SPIF))); // wait until transmission complete

    SPDR = 0b10000000 | (channel << 6); // single ended mode, MSB first
    while(!(SPSR & (1<<SPIF))); // wait until transmission complete
    byte_rec = (SPDR << 8);

    SPDR = 0x00;          // request lower bits
    while(!(SPSR & (1<<SPIF))); // wait until transmission complete
    byte_rec |= SPDR;        // 3 bits + null bit + "12 bit value"

    SPI_PORT |= (1<<AD_CS);      // deselect device
    return (byte_rec & 0b0000111111111111);    // return "12 bit value"
}
```

Listing 2: Wertabfrage des ADC MCP3202 im 8-Bit Modus

3.2.4 USB-Übertragung mit dem FT232RL

Der FT232RL ist ein Konverter von einer USB Schnittstelle zu einem asynchronen seriellen Datentransfer. Das USB-Protokoll wird komplett auf dem Chip durchgeführt. Damit ist keine USB-spezifische Programmierung nötig. Die Datentransferrate beträgt von 300 Bit bis über 1 Megabit pro Sekunde[33] für RS232. Der Chip beinhaltet unter anderem einen 3,3 Volt Spannungsregler und einen Taktgenerator für verschiedene Taktraten.
Die asynchronen seriellen Daten vom Mikroprozessor werden vom chipinternen USART Steuergerät verarbeitet. Die Daten werden um das USB-Protokoll ergänzt und durch die USB Sendeeinheit dem USB 2,0 Standard entsprechend mit 3,3 Volt Pegeln versendet.

[33] Siehe [17] FT232R Datenblatt, Seite 2.

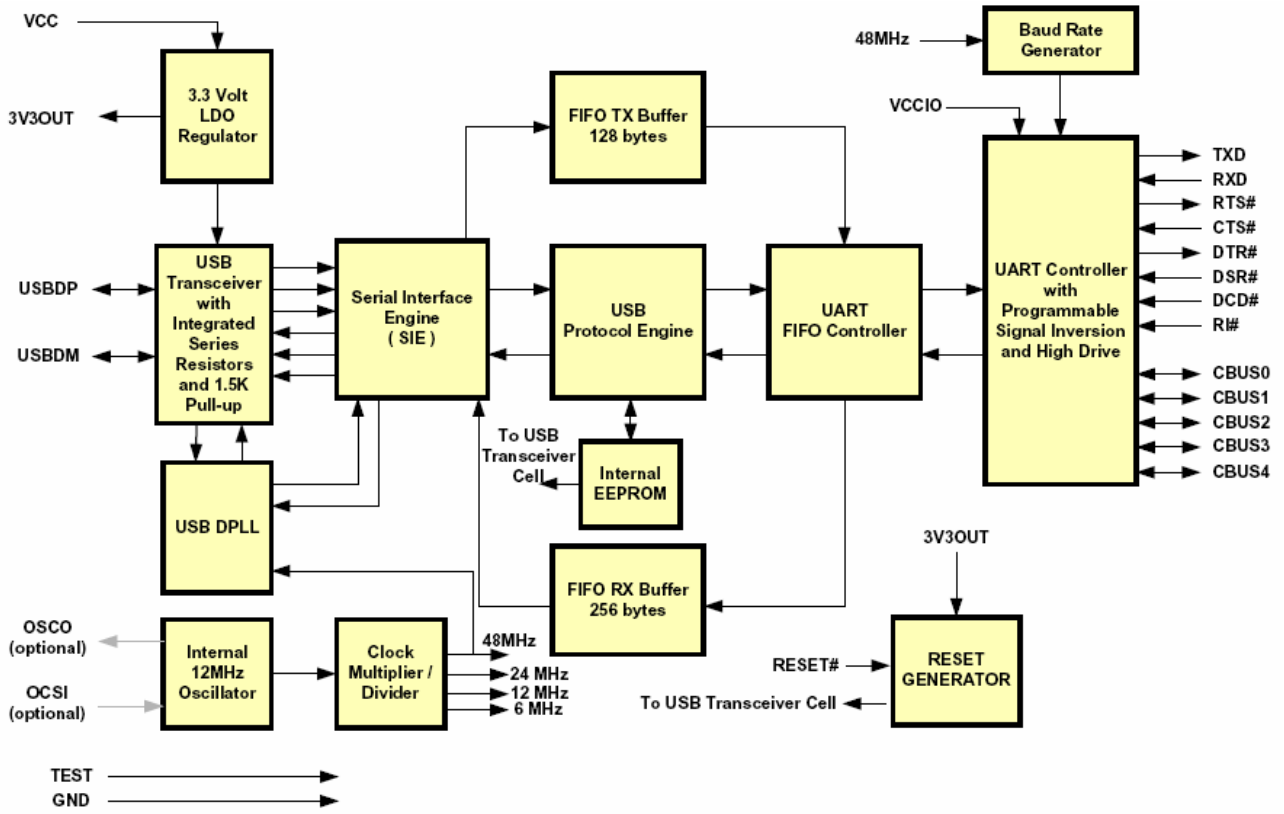


Bild 23: Blockdiagramm des USB Konverters FT232RL[34]

Die Kommunikation zwischen dem FT232RL und dem ATMEGA128 erfolgt über den U-SART. Dieser wird im Mikroprozessor für das Senden und Empfangen initialisiert und zudem ein Interrupt zum Empfangen aktiviert. Eine Bitrate von 921.600 Bit/Sekunde wird für die Übertragung festgelegt.

```
void USART_Init() {
    // transmit and receive enable + interrupt
       UCSRB = (1<<RXCIE) | (1<<RXEN) | (1<<TXEN);
    // Baud rate setting
       UBRRH = (uint8_t) (USART_UBRR_CALC(USART_BAUD_RATE, F_CPU)>>8);
    UBRRL = (uint8_t)  USART_UBRR_CALC(USART_BAUD_RATE, F_CPU);
}
```

Listing 3: Initialisierung des USART für den FT232RL

Ein Interrupt wird ausgelöst, wenn der Eingangsdatenpuffer voll ist. Das gesendete Byte wird dann direkt in der Interruptroutine übernommen. Für das Senden von Einzelzeichen

[34] Bild entnommen aus [17] FT232R Datenblatt, Seite 5.

wird kein Interrupt benötigt, da alle in den Schreib- und Lesepuffer des Mikroprozessors geschriebenen Daten automatisch gesendet werden.

```
volatile uint8_t *transmit_data;
volatile uint8_t receive_data;

ISR (USART_RXC_vect) {       // UART receive complete
    receive_data = UDR;  // read data from register
}

void USART_transmit(uint8_t c) {
    while (!(UCSR0A & (1<<UDRE0))) {}   // register empty
    UDR0 = c;      // transmit byte and step to next
}
```

Listing 4: Kommunikation über den USART mit dem FT232RL

3.2.4.1 Programmierung des EEPROM mit MProg 3.0a

MProg 3.0a ist eine Programmiersoftware zur Einstellung von FTDI-Bauteilen. Nach Installation eines Treibers kann die USB-Verbindung zur Programmierung genutzt werden. Der Bauteiltyp wird automatisch erkannt. Die Standardeinstellungen müssen kaum verändert werden, wichtig ist jedoch die Einstellung der eigenen Spannungsversorgung des x/y-Schreibers. Zudem könnten noch Seriennummern vergeben werden. Ansonsten sind die Einstellungen wie in Bild 24 vorzunehmen.

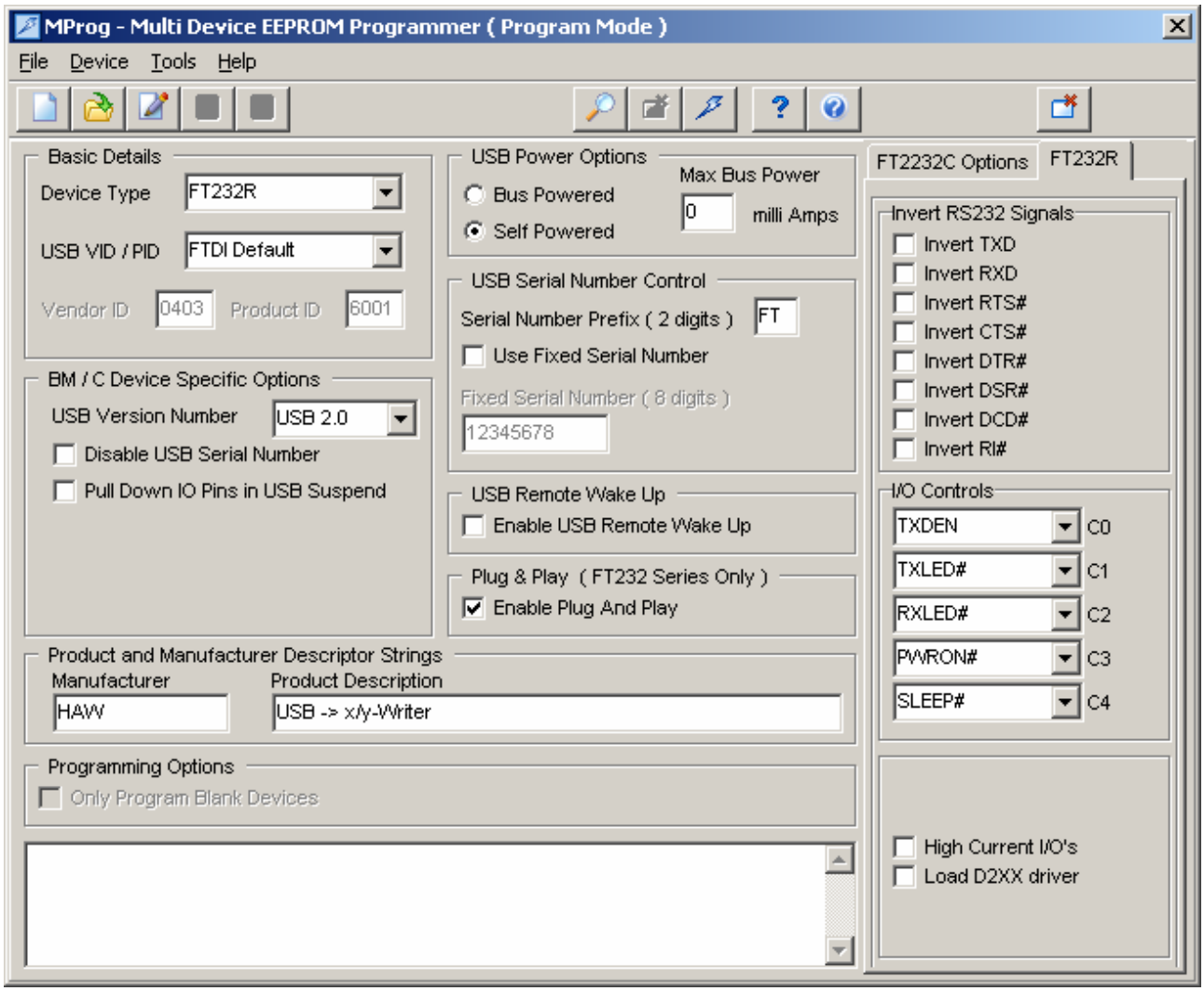


Bild 24: Screenshot des EEPROM-Programmers MProg

3.2.5 Spannungsversorgung

Als Spannungsversorgung sind plus und minus 15 Volt vorgesehen. Beide Anschlüsse sind gegen Verpolung geschützt. Durch Spannungsregler werden +5 V mit einem Maximalstrom von 1 A und plus/minus 12 Volt für die Schaltung erzeugt. Hierdurch ist die Schaltung bis zu einer Eingangsspannung von 35 Volt geschützt. Die Spannungsregler erhitzen schnell bei hohen Spannungen. Sie verfügen über eine Rückflusssicherung, um eine Beschädigung durch eventuell auftretende Ströme aus den Kondensatoren beim Ausschalten zu verhindern.

3.2.6 Anschlüsse

Die Steuerungsplatine verfügt über Anschlüsse für die Versorgungsspannung, die Verstärkerplatine, das Display, die Taster und Schalter und für die Programmierung des Mikroprozessors. Ungenutzte Ein- und Ausgänge des ATMEGA128 werden zusätzlich herausgeführt. Die genaue Beschaltung der Anschlüsse ist im Anhang B, Seite XVI aufgeführt.

3.2.7 Platine

Der Schaltplan[35] und das Layout[36] der Platine wurden mit dem Programm Eagle™ erstellt. Bild 25 zeigt die damit gefertigte, bestückte Platine.

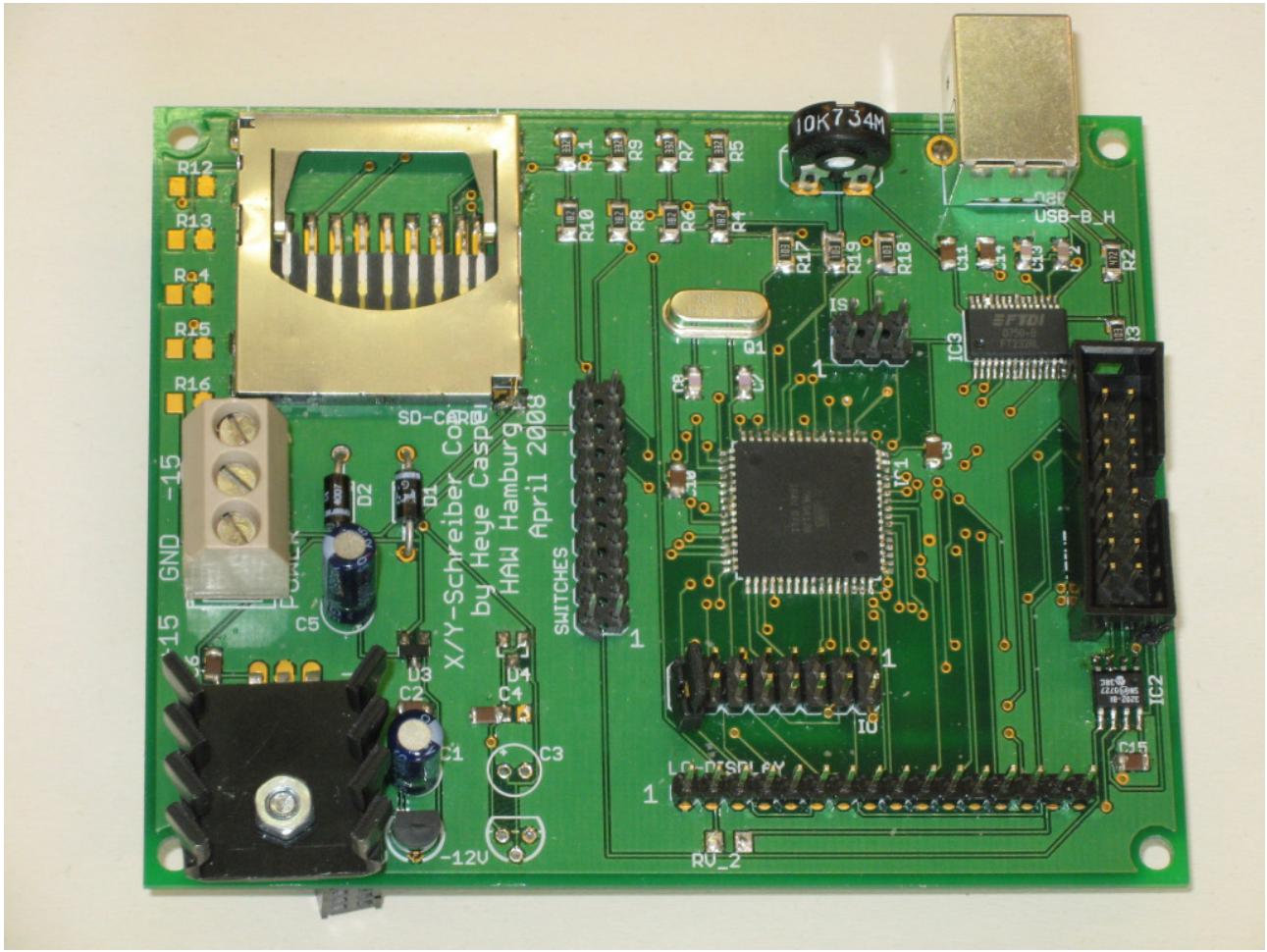


Bild 25: Steuerungsplatine des x/y-Schreibers

[35] Der Schaltplan ist in Anhang A.3 abgebildet.

[36] Das Layout ist in Anhang A.4 abgebildet.

Der Anschluss des Programmiersteckers ist auf der abgebildeten Platine nicht funktional. Die Programmierung des ARMEGA128 erfolgt über die UART-Pins TxD und RxD und nicht über die SPI-Pins MOSI und MISO. Dies ist in dem in Anhang A.3 und A.4 befindlichen Schaltplan und Layout berichtigt. Zudem wurden die Anschlüsse für die Analogeingänge getauscht, um mit der Verstärkerplatine konsistent zu sein.

3.3 PC-Programm des x/y-Schreibers

Die PC-Software des x/y-Schreibers stellt die durch die Hardware erfassten Datenpunkte grafisch dar. Die Einstellung der Datenerfassung wird ausschließlich in dem über eine USB-Schnittstelle angeschlossenen Peripheriegerät durchgeführt. Die Skalierung und Beschriftung der Achsen erfolgt automatisch entsprechend der extern gewählten Auflösung und des Offsets. Das Programm ist als MFC-Anwendung mit einer statischen Bibliothek verwirklicht[37].

[37] Der Quellcode des PC-Programms befindet sich in Anhang D.

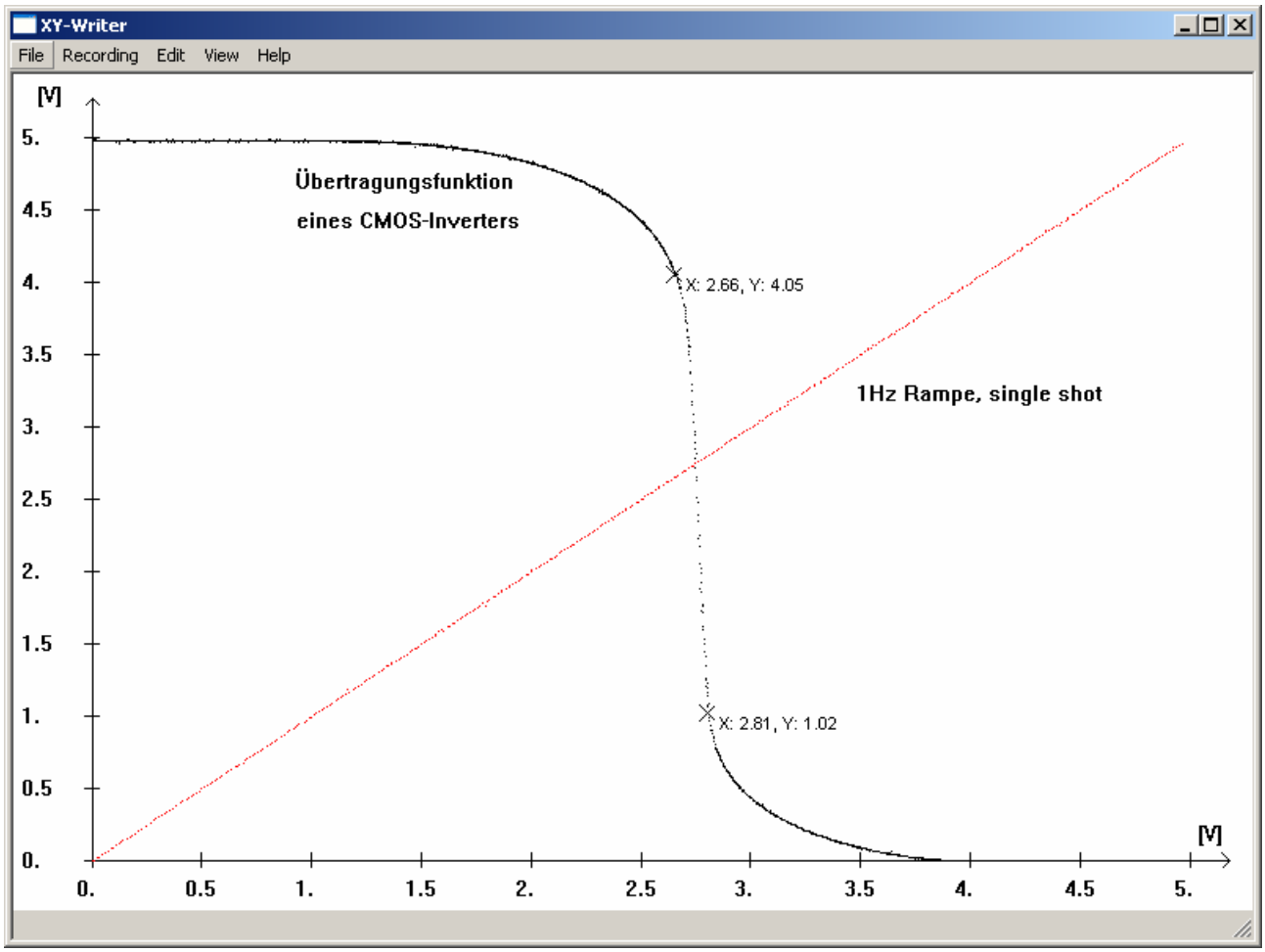


Bild 26: Screenshot des PC-Programms

Beim Zusammenspiel zwischen Hardware und Software ist das oberste Ziel ein dem analogen x/y-Schreiber vergleichbares Verhalten zu erreichen. Die Darstellung und Weiterverarbeitung der Daten soll zudem möglichst komfortabel sein. Die einzelnen Messpunkte werden deshalb in „Echtzeit“ gleich nach ihrer Messung dargestellt. Dadurch erhält der Benutzer schon zu Beginn der Datenaufnahme eine Rückmeldung über seine Messung. Während der Aufnahmephase können keine Änderungen der Messseinstellungen vorgenommen werden. Der Benutzer hat keinen Zugriff auf das Programm, bis die Datenaufnahme beendet ist. Nach erfolgter Aufnahme ist es möglich, die erfassten Datensätze zu speichern, diese wieder zu öffnen und auch mit den eingestellten Aufnahmeparametern grafisch darzustellen. Informationen über einzelne Punkte können durch einen linken Mausklick in die Grafik eingefügt werden. Diese sind wie eingefügter Text und zusätzliche Graphen ausschließlich auf der grafischen Oberfläche dargestellt. Über eine „Undo“-Funktion können diese Zusätze zu der eigentlichen Darstellung der Aufnahme wieder ge-

löscht werden. Der gesamte sichtbare Bildschirm mit allen vorgenommenen Beschriftungen kann ausgedruckt und auch als Bitmap gespeichert werden. Lücken in der Darstellung, die durch die technisch bedingte Abtastfrequenz und die Steilheit des Eingangssignals entstehen können, können grafisch interpoliert werden.

3.3.1 Ablauf der Einstellung und Datenaufnahme

Die Ablaufdiagramme in Bild 27 und Bild 28 stellen den gesamten Ablauf der Einstellung der Messparameter und der Datenaufnahme dar. Für eine über den Menüpunkt Recording/Connect des x/y-Schreiberprogramms hergestellte Verbindung wird über Recording/Record die Funktion startRecording() (Bild 27) als Einstiegspunkt aufgerufen. Sie beginnt mit der Einstellungsphase (record_flag = false) und mit dem Aufruf der Funktion getData() (Bild 28). Ein Datensatz wird durch die Funktion receiveData() eingelesen. Nach der Einstellungsphase wird durch den Aufruf der Funktion initRecord() mit der Aufnahmephase begonnen, in der ebenfalls die Funktion getData() aufgerufen wird. Nach erfolgter Datenaufnahme wird die Verbindung wieder beendet.

Das Umschalten von der Einstellungs- in die Aufnahmephase erfolgt automatisch durch das erste aufgenommene Datenpaar nach dem Druck auf die Taste „Start“ am Peripheriegerät. Die Datensätze der beiden Modi unterscheiden sich grundsätzlich im ersten Byte. Dieses ist in der Einstellungsphase mit „A“ (für Adjustments) und der Aufnahmephase mit „R“ (für Record) belegt. Wird nun während der Einstellungsphase ein Datensatz mit „R“ als erstem Byte empfangen, wird eine Flagge gesetzt (record_flag) und die Einstellungsphase verlassen. Das setzen der Flagge hat zur Folge, dass die Schleife der Funktion startRecording() beendet wird und die Aufnahmephase beginnt. Das Umschalten von Aufnahme- zu Einstellungsphase erfolgt analog. Mit Auftreten eines „A“ im ersten Byte eines Datensatzes wird die Aufnahme beendet. Für das Umschalten zwischen den Phasen ist die Flagge „record_flag“ verantwortlich, die beim ersten Auftreten eines neuen geänderten Startbytes für die Aufnahme gesetzt und für die Einstellungsphase gelöscht wird.

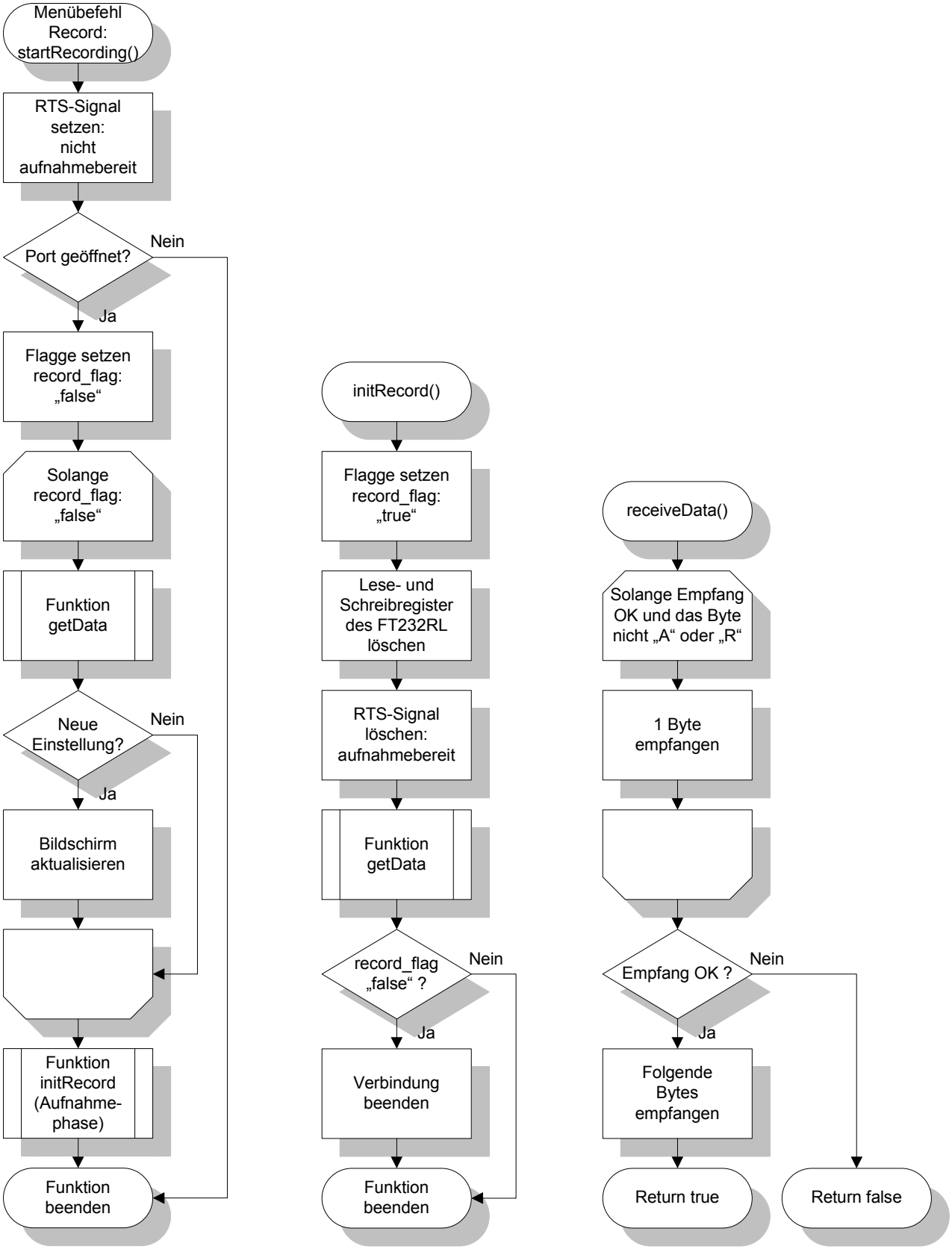


Bild 27: Ablaufdiagramme von startRecording(), initRecord() und receiveData()

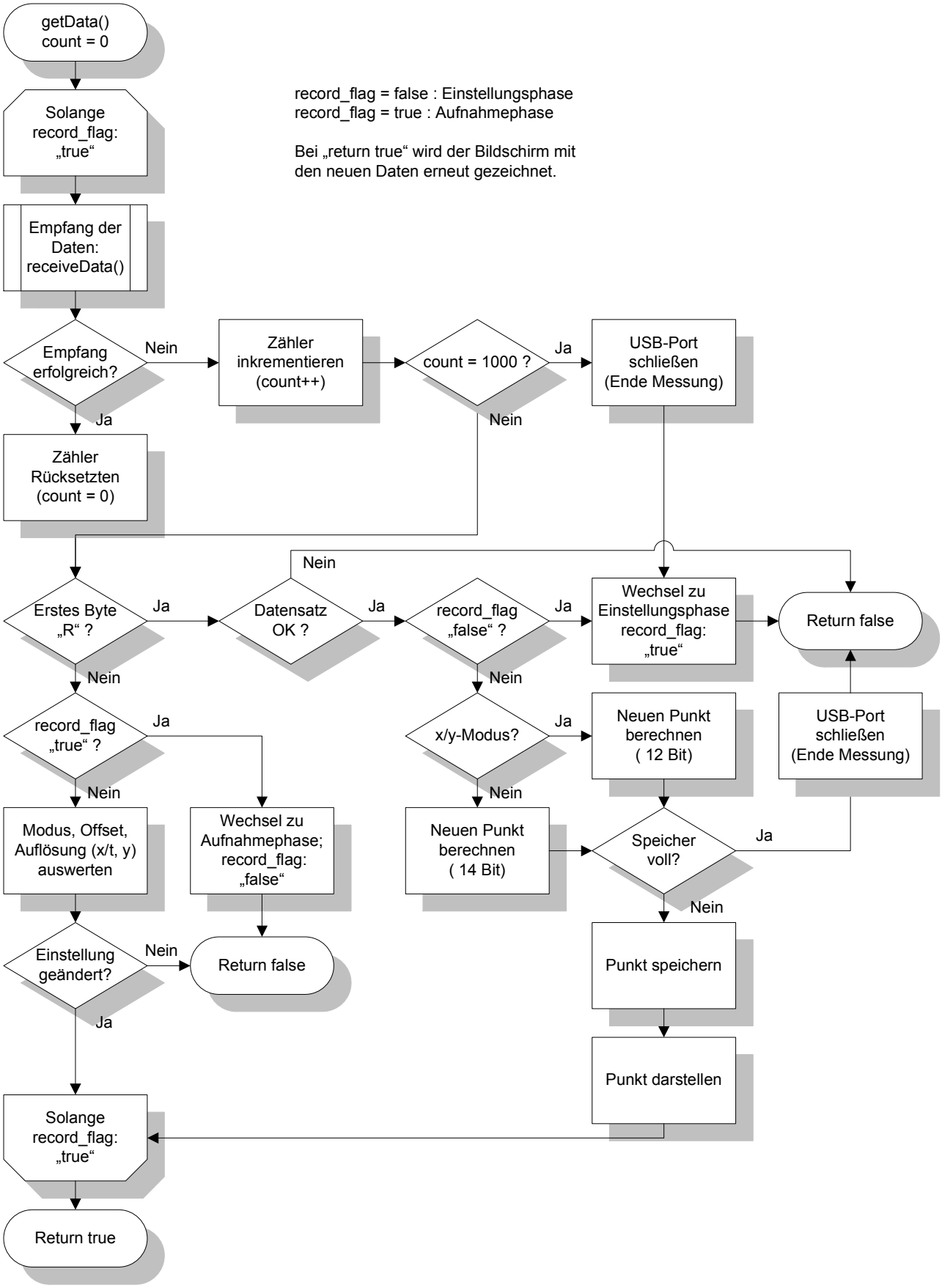


Bild 28: Ablaufdiagramm der Funktion getData()

Zu Beginn der Funktion getData() wird über den Zähler „count“ ein Programmabbruch nach 1000 aufeinander folgenden erfolglosen Empfangsversuchen durchgeführt. Hierdurch wird ein Programmabsturz, hervorgerufen zum Beispiel durch eine Unterbrechung der Verbindung, verhindert. Nach erfolgreichem Datenempfang sind zwei Abläufe voneinander zu trennen, die durch das erste Byte des eingelesenen Datensatzes unterschieden werden. Bei übermitteltem „A“ werden die Einstellungen der Aufnahme aktualisiert. Mit einem „R“ im ersten Byte startet die Datenaufnahme mit einer Unterteilung in x/y und Zeitbetrieb. Die Auflösung der x-Achse ist im Zeitbetrieb 14 Bit, alle anderen Auflösungen sind 12 Bit. Wenn alle Speicherplätze mit Daten belegt sind, wird die Datenaufnahme abgebrochen.

3.3.2 USB-Kommunikation mit FT232RL

Für die Kommunikation mit dem USB zu RS232 Umsetzer FT232RL gibt es zwei unterschiedliche treiberabhängige Möglichkeiten. Zum Einen über einen virtuellen COM Port (VCP), der durch den Treiber emuliert wird. Hierdurch kann auf die gleiche Weise wie bei RS232-Verbindungen kommuniziert werden. Zum Anderen kann durch den D2XX-Treiber direkt auf die USB-Schnittstelle zugegriffen werden. Die Datenrate ist für beide Treiber mit maximal 1 MBit/s für RS232-Verbindungen angegeben. Für den x/y-Schreiber wurde der direkte Zugriff über D2XX gewählt, da die Verbindung hier automatisch durch die Identifikation des angeschlossenen Gerätes erfolgt und kein COM-Port vom Benutzer gewählt werden muss. Es ist zu beachten, dass andere USB-Geräte, speziell von FTDI, den Verbindungsaufbau stören können, da auf explizite Seriennummernerkennung verzichtet wurde.

Der Verbindungsaufbau erfolgt mit[38]

```
ftStatus = FT_Open(0, &ftHandle);
if (ftStatus != FT_OK) { // FT_Open OK, use ftHandle to access device
    return false;        // connection established
}
```

Listing 5: Öffnen einer USB-Verbindung

[38] Für die Programmierung wurde [20] D2XX Programmer´s Guide, Seite 11ff genutzt.

Es darf nur ein Gerät angeschlossen sein. Der zurückgegebene Handle wird für alle Zugriffe auf diese Verbindung verwendet. Mit folgenden Funktionen werden die Parameter für die Kommunikation festgelegt, erstens die Bitrate, zweitens die Datencharakteristik mit 8 Bits, einem Stoppbit und keinem Paritätsbit, drittens die Flusskontrolle und viertens die Timeouts für Lese- und Schreibzugriffe in Millisekunden:

```
ftStatus = FT_SetBaudRate (ftHandle, BaudRate);
FT_SetDataCharacteristics(ftHandle, BITS_8, STOP_BITS_1, PARITY_NONE);
ftStatus = FT_SetFlowControl (ftHandle, FT_FLOW_NONE, NULL, NULL);
ftStatus = FT_SetTimeouts(ftHandle,1000,1000);
```

Listing 6: Einstellen der USB Kommunikationsparameter

Schreibzugriffe werden mit FT_Write() für ein Byte wie folgt durchgeführt. Die zu sendenden Daten sind in TxBuffer enthalten.

```
ftStatus = FT_Write (ftHandle, TxBuffer, 1, &BytesWritten);
if (ftStatus == FT_OK)
    return true;      // FT_Write OK
```

Listing 7: Schreibzugriff auf die USB-Verbindung

Beim Lesezugriff mit der Funktion FT_Read() für RxBytes werden die empfangenen Zeichen im RxBuffer abgelegt. In RxBytesReceived steht die Anzahl der empfangenen Zeichen.

```
ftStatus = FT_Read(ftHandle,&RxBuffer[0],RxBytes,&BytesReceived);
if (ftStatus == FT_OK) {
    if (BytesReceived == RxBytes)
        return true;  // FT_Read OK
    else
        return false; // FT_Read Timeout
}
else
    return false;     // FT Read Failed
```

Listing 8: Lesezugriff auf die USB-Verbindung

Die Verbindung wird mit der Funktion FT_Close() beendet.

```
ftStatus = FT_Close (ftHandle);
if (ftStatus == FT_OK)
    return true;      // connection closed
```

Listing 9: Beenden der USB-Verbindung

3.3.3 Öffnen und Speichern von Dateien

Mit der Funktion GetOpenFileName()[39] der MFC wird ein vordefinierter Dialog zum Öffnen von Dateien aufgerufen, der die Auswahl des Laufwerks, Ordners und Dateinamens ermöglicht. Die Funktion GetSaveFileName() ruft entsprechend einen Dialog zum Speichern in Dateien auf[40]. Die Struktur OPENFILENAME beinhaltet die Informationen, nach denen die Dialogboxen für das Öffnen und Speichern gestaltet werden. Hier werden auch die ausgewählten oder eingegebenen Dateinamen gespeichert.

```
OPENFILENAME ofn;
(...)
// Initialize OPENFILENAME
ZeroMemory(&ofn, sizeof(OPENFILENAME));    // fill memory with zeros
ofn.lStructSize = sizeof (OPENFILENAME);
(...)
ofn.Flags = OFN_PATHMUSTEXIST | OFN_HIDEREADONLY | OFN_CREATEPROMPT;
if( GetOpenFileName(&ofn) )     // Display the open dialog box.
{
    ifstream inp(fileName);     // open from file
    while(!inp.eof())
    {   inp >> x >> y;          // read in x and y
        (...)
    }
    return true;
}
(...)
```

Listing 10: Öffnen einer Datei unter Verwendung der MFC

[39] Als Referenz für die MFC-Programmierung wurde die [21] MSDN Library for Visual Studio 2005 genutzt.

[40] Öffnen und Speichern verlaufen prinzipiell gleich. Für ein Beispiel des Speicherns sei auf den Programmcode in Anhang D verwiesen.

3.3.4 Print und Save As Bitmap

Um den Inhalt des Fensters zu drucken oder als Bild zu speichern, wird dieser zunächst als Bitmap erfasst. Die Struktur BITMAPINFO enthält Farbinformationen und Festlegungen für die Ausmaße und Auflösung der Grafik, die ihrerseits in der Struktur BITMAPINFO-HEADER zusammengefasst sind. Anschließend wird eine Kopie des Fensters mitsamt dessen Inhalt erstellt.

```
BITMAPINFO bmi;
BITMAPFILEHEADER bmfh;
BITMAPINFOHEADER bmih;
ZeroMemory(&bmih, sizeof(BITMAPINFOHEADER));  // initialise bmih
bmih.biSize = sizeof(BITMAPINFOHEADER);
bmih.biHeight = drawGraph.clRect.bottom-20;
(...)
bmi.bmiHeader=bmih;  // load bitmap info and then Create the surface
hbm = CreateDIBSection(dc, &bmi, DIB_RGB_COLORS, (void**)&pBits, 0, 0);
dcMem = CreateCompatibleDC(dc); // Make copy of window
SelectObject(dcMem, hbm);       // assign the object/bitmap to winwow
// Copy the window contents to the memory surface/ into the bitmap
BitBlt(dcMem, 0, 0, (...).right, (...).bottom, dc, 0, 0, SRCCOPY);
```

Listing 11: Erstellung eines Bitmaps des Fensterinhalts mit der MFC

Für das Drucken wird mit der Funktion PrintDlg der Drucken-Dialog geöffnet, der vergleichbar dem Ablauf des Öffnens und Speicherns durch die Struktur PRINTDLG definiert wird. Mit der Datenstruktur DEVMODE können die Einstellungen des Druckers vordefiniert werden. Das Druckverfahren wird durch die Funktion StartDoc und anschließend StartPage zur Aktivierung des Druckertreibers begonnen. StretchBlt passt das Bitmap auf das Papierformat an und übergibt damit die Seite an den Drucker. Der Druckablauf wird mit EndPage und EndDoc beendet.

```
PRINTDLG pdlg;
DOCINFO di;
DIBSECTION ds;
ZeroMemory(&pdlg, sizeof(PRINTDLG));   // fill memory with zeros
pdlg.lStructSize = sizeof (PRINTDLG);
pdlg.Flags = PD_RETURNDEFAULT;
PrintDlg(&pdlg);      // first call for devmode, does not open dialog
DEVMODE* dm = (DEVMODE*)::GlobalLock(pdlg.hDevMode);
dm->dmOrientation = DMORIENT_LANDSCAPE;       // landscape
(...)
pdlg.hwndOwner = drawGraph.hWnd;
pdlg.Flags = PD_RETURNDC;    // order handle for printer attach
if (PrintDlg(&pdlg)) {       // open print dialog
    CDC dcPrint;
    dcPrint.Attach(pdlg.hDC);    // attach dialog info to printer
    (...)
    if (StartDoc(dcPrint, &di) != 0){   // start the print job
        if (StartPage(dcPrint) != 0){   // Prepare driver for data
            // Print the contents of the surface/bitmap
            (...)
            EndPage(dcPrint); // tell driver print job is done
        }
    EndDoc(dcPrint);         // tell driver the document is
}
(...)
```

Listing 12: Drucken eines Bitmaps mit der MFC

Das Abspeichern des Fensters als Bitmap[41] erfolgt über den Dialog Speichern wie in Kapitel 3.3.3 beschrieben. Die erstellte Datei muss neben dem Bitmap mit den Daten des BITMAPINFOHEADERS auch Informationen über das Dateiformat enthalten, die in der Struktur BITMAPFILEHEADER zusammengefasst sind. Diese werden nacheinander in die Datei gespeichert.

[41] Als Anleitung der Erstellung eines Bitmaps diente neben [21] MSDN Library for Visual Studio 2005 auch [23] Bildschirminhalt als Bitmap speichern.

```
(...)
ofn.lStructSize = sizeof (OPENFILENAME);
(...)
if( GetSaveFileName(&ofn) ){ // Display the save dialog box.
    ZeroMemory(&bmfh, sizeof(BITMAPFILEHEADER));
    (...)
    bmfh.bfType=0x4d42;      // BM - Filetype
    bmfh.bfReserved1 = 0;
    bmfh.bfReserved2 = 0;
    fh = CreateFile(ofn.lpstrFile, (...)); // create file
    bytes_write=sizeof(BITMAPFILEHEADER);  // write bitmap file header
    if (!WriteFile(fh, (void*)&bmfh, bytes_write, &bytes_written, 0)){
        OutputDebugString(L"WriteFile failed!\n");
    }
    bytes_write=sizeof(BITMAPINFOHEADER);  // write bitmap info header
    if(!WriteFile(fh, (void*)&bmih, bytes_write, &bytes_written, 0)){
        OutputDebugString(L"WriteFile failed!\n");
    }
    bytes_write = bmih.biSizeImage;        // write bitmap image bits
    if(!WriteFile(fh, (void*)pBits, bytes_write, &bytes_written, 0)){
        OutputDebugString(L"WriteFile failed!\n");
    }
}
(...)
```

Listing 13: Als Bitmap speichern mit der MFC

3.4 Bedienung

Die Bedienung des x/y-Schreibers für den Aufnahmevorgang erfolgt ausschließlich an dem Peripheriegerät, um die Hardwarenähe der bisherigen, rein analogen Geräte zu erhalten. Für die Einstellung der Auflösung der beiden Achsen werden jeweils zwei Taster verwendet. Ein Schalter mit drei möglichen Zuständen regelt die Auswahl des Betriebsmodus und über zwei Potentiometer wird der Offset der x- und y-Achse eingestellt. Der Taster „Start“ startet und beendet die Datenaufnahme, verlässt damit die Phase der Konfiguration oder lässt wieder Einstellungen zu. Der Aufnahmemodus des PC-Programms wird ausschließlich durch das Ende der Aufnahme des Peripheriegerätes beendet.

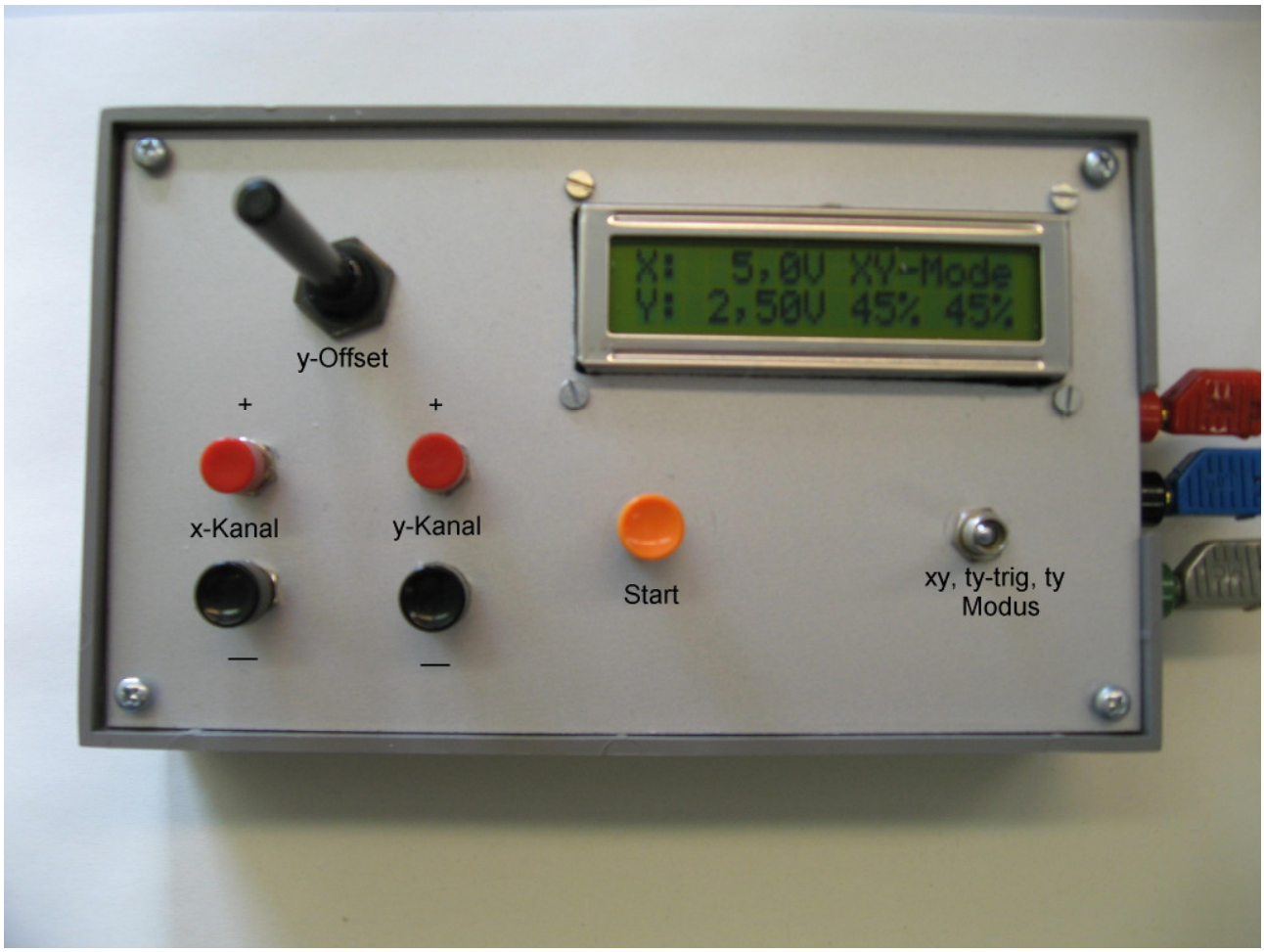


Bild 29: Bedienfeld des x/y-Schreibers mit nur einem Offset

3.4.1 Ablauf eines Aufnahmevorganges

Nach dem Einschalten startet der x/y-Schreiber mit der Einstellungsphase. Im x/y-Betrieb kann über je zwei Taster pro Kanal die gewünschte maximale Spannung des Aufnahmesignals eingestellt werden. Nach hergestellter Verbindung mit dem PC-Programm zur Verwaltung und Darstellung der Messdaten wird die Einstellung an den Achsen automatisch dargestellt. Ein Offset kann zu beiden Kanälen hinzugefügt werden. Dieser ist in Prozent der Auflösungseinstellung angegeben. Durch Drücken des „Start“-Knopfes beginnt die Aufnahme. Auf das PC-Programm kann in dieser Phase nicht zugegriffen werden. Die Messung kann jederzeit durch erneutes Betätigen des Tasters „Start“ beendet werden. Die Verbindung zwischen dem Aufnahmegerät und dem PC-Programm wird mit dem Ende der Aufnahme getrennt und der x/y-Schreiber kehrt wieder in die Einstellungsphase zurück.

Der aufgenommene Verlauf kann abschließend bearbeitet und verwaltet werden. Mit dem Beginn einer neuen Messung wird der Bildschirm gelöscht.
Der Zeitmodus verhält sich äquivalent zum x/y-Modus. Mit den Tastern für den x-Kanal wird hier die Laufzeit eingestellt. Die Aufnahme nach Drücken des Tasters „Start“ wird mit dem Ablauf der Laufzeit beendet, kann aber jederzeit durch erneutes Drücken von „Start“ unterbrochen werden. Eine Besonderheit ist die Möglichkeit eines getriggerten Aufnahmebeginns. Nach dem ersten Betätigen von „Start“ kann über die Taster des y-Kanals eine Triggerschwelle eingestellt werden. Sie ist in Prozent der Eingangsspannung des x-Kanals angegeben. Ein zweites Betätigen von „Start“ aktiviert den Trigger, der auf eine positive oder negative Überschreitung der Schwelle wartet. Bei Überschreiten der Schwelle wird die Messung begonnen. Das PC-Programm befindet sich währenddessen immer in Empfangsbereitschaft.

3.4.2 Bearbeitung der Messdaten

3.4.2.1 Das Menü „File“

Die Messdaten befinden sich nach der Aufnahme ausschließlich in flüchtigem Speicher. Sie können mit „Save As“ aus dem Menü „File“ in einer Textdatei abgelegt werden. Alle empfangenen Datenpaare werden sukzessiv abgelegt. Alle Werte außer den Zeitangaben haben eine Auflösung von 12 Bit, entsprechend 4096 Punkten. Zeitangaben sind mit 14 Bit, entsprechend 16384 aufgelöst. In der ersten Zeile stehen die Parameter der Datenaufnahme (Modus, x- oder Zeitauflösung, y-Auflösung, y-Offset und x-Offset). Zur Eingabe des Dateinamens und Auswahl des Speicherortes dient der Windows Standarddialog.
Mit „New“ werden alle bereits erfolgten Messungen und Zusätze der grafischen Oberfläche gelöscht. Zuvor gespeicherte Messreihen können mit „Open“ wieder geladen werden. Dabei werden auch die Parameter der Datenaufnahme, wie die Auflösung der Achsen, erfasst und in die Darstellung mit einbezogen. Geöffnete Messreihen werden wie neu aufgenommene behandelt.
Mit „Print“ wird das gesamte Fenster ausgedruckt. Einstellungen, wie Quer- oder Hochformat und die Auswahl des Druckers können über das Standarddialogfeld getroffen werden. Mit „Save As Bitmap“ wird das Fenster als Bitmap gespeichert. Die Auflösung des

Originalfensters wird hierbei übernommen, so dass dieses für eine möglichst hohe Auflösung des Abbildes maximiert werden sollte.

3.4.2.2 Das Menü „Recording“

Über „Connect“ wird eine USB-Verbindung zu dem Datenaufnahmegerät aufgebaut. Andere USB Kommunikationsgeräte[42] und insbesondere weitere x/y-Schreiber können einen Verbindungsaufbau verhindern, da nicht zwischen den einzelnen Geräten unterschieden wird. In der Statuszeile wird der Zustand der Verbindung angezeigt. Die aktuelle Einstellung wird bei einem Mausklick oder einer Mausbewegung aktualisiert. Über „Disconnect“ kann die Verbindung wieder beendet werden. Mit „Record“ beginnt die Aufnahmephase. Die Software befindet sich anschließend im Aufnahmemodus und die Messdaten werden punktweise gleich nach ihrer Messung dargestellt. Die Aufnahme kann nur durch das Beenden der Messung am Peripheriegerät unterbrochen werden. Es besteht währenddessen kein Zugriff auf das Programm. Die Verbindung wird nach erfolgter Aufnahme beendet und die Messdaten können bearbeitet werden.

3.4.2.3 Die Menüs „Edit“ und „View“

Das Menü „Edit“ enthält die Funktionen „Interpolate“, „Undo“ und „Add Text“. Mit „Interpolate“ werden die nicht aufgenommenen Werte zwischen den einzelnen Messpunkten linear ergänzt. Es wird nur die Bildschirmoberfläche ergänzt, so dass die Interpolation bei jedem Neuzeichnen des Bildschirminhalts verloren geht.
Nach Auswahl des Punktes „Add Text“ öffnet sich ein Dialogfeld, in das ein beliebiger Text eingegeben werden kann, der durch anschließendes Betätigen der linken Maustaste in der grafischen Oberfläche einmal platziert werden kann. Weiteres Betätigen der Maustaste in dem Darstellungsbereich fügt zu der Position der Maus, relativ zur x-Achse, Punktinformationen an der aufgenommenen Messreihe ein.
Vorher gespeicherte Grafen können über „Add Graph from File“ aus dem Menü „View“ zu der dargestellten Messreihe hinzugefügt werden. So können mehrere abgespeicherte Messreihen mit der jetzigen gleichzeitig dargestellt werden. Dies ist jedoch nur für gleiche Auflösungseinstellungen möglich. Punktinformationen können nur für die aktuelle Messrei-

[42] Ein USB zu RS232 Adapter hat in Tests den Verbindungsaufbau verhindert.

he angegeben werden. Mit der „Undo“-Funktion können alle in die Darstellung eingefügten Texte, Punktinformationen und zusätzlichen Grafen sukzessive rückgängig gemacht werden. Die vollständige Oberfläche kann, wie oben erwähnt, ausgedruckt und als Bild gespeichert werden.

Durch „Redraw“ werden alle grafischen Änderungen gelöscht und ausschließlich die aufgenommenen Daten erneut dargestellt. Mit „Number Of Digits“ werden die Nachkommastellen der Punktinformationen für die Anzeige in der grafischen Oberfläche eingestellt. Vordefinierte Bildschirmauflösungen können über „Screen Resolution“ eingestellt werden, jedoch kann das Fenster auch auf eine beliebige Größe mit der Maus gezogen oder mit der Schaltfläche „Maximieren“ auf Bildschirmgröße vergrößert werden. Nach erfolgter Datenaufnahme sollte die Fenstergröße nicht mehr verändert werden, da die Aktualisierung größerer Datenmengen recht viel Zeit in Anspruch nimmt.

4 Funktionsprüfung und Spezifikation

Die Funktionsprüfung wird für die Steuer- und Verstärkerplatine separat durchgeführt. Mit der zuerst auf ihre Funktionalität geprüften Steuerplatine wird anschließend die Verstärkerplatine getestet. Bei den Untersuchungen der Steuerungsplatine sind, bedingt durch die Testergebnisse, verschiedene Änderungen vorgenommen worden. Auf die Anpassungen wird in den betreffenden Kapiteln eingegangen und ihre Auswirkungen werden vergleichend dargestellt.

Eine von zwei wesentlichen Anpassungen ist die Unterdrückung der Anzeige gemessener Wertepaare auf dem LC-Display, wodurch die Aufnahmefrequenz deutlich erhöht werden konnte (siehe Kapitel 4.1.5). Weiterhin wurde die Verarbeitung der aufgenommenen Daten von einem festen Puffer mit einer Datengröße von 12 Bit (4096 Wertepaare) im x/y-Betrieb und 14 Bit (16384 Wertepaare) im Zeitbetrieb auf grundsätzlich 1.000.000 Speicherplätze erhöht. Es erfolgt keine Mittelwertbildung mehr für wiederholt auftretende x-Werte, sondern alle eingehenden Daten werden nacheinander in den Puffer abgelegt. In Kapitel 4.1.1 wird auf die Auswirkungen genauer eingegangen. Die resultierende Veränderung der Aufnahmefrequenz wird in Kapitel 4.1.4 und 4.1.5 untersucht. Gleichzeitig wird zusätzlich zum y-Kanal ein Offset für den x-Kanal durch die Steuer-Platine und das PC-Programm unterstützt. Für die Übertragung der Offset-Informationen wird die Anzahl der Bytes um zwei auf acht Bytes erhöht. Die Anpassungen der Verstärker-Platine sind im Schaltplan und Layout[43] bereits berücksichtigt.

Die Datenprotokolle der Messungen, die hier nur in Ausschnitten betrachtet werden, sind vollständig auf beiliegender CD enthalten.

4.1 Steuerung

Die folgenden Untersuchungen des Verhaltens des x/y-Schreibers, insbesondere des Zeitverhaltens wurden, sofern nicht anders bezeichnet, mit einem SPI-Takt von 0,9216 MHz und einer USB-Übertragungsrate von 921.600 Bit pro Sekunde durchgeführt. Die Messungen erfolgten ohne Verstärkerplatine, mit den Eingangssignalen direkt an den beiden Analogeingängen des Analog/Digital-Umsetzers. Die Eingangssignale wurden mit

[43] Schaltplan und Layout befinden sich in Anhang A.1 und A.2.

den Funktionsgeneratoren HP 3312A oder PM 5138 erzeugt und mit Hilfe eines Oszilloskopes Tektronix 2465 überprüft, bevor sie für die nachfolgenden Messungen verwendet wurden.

4.1.1 Kennlinienaufnahme im x/y-Betrieb

Für den Nachweis der Funktion des x/y-Betriebes wird die Übertragungskennlinie $U_Q = f (U_I)$ eines CMOS-Inverters CD 4007 aufgenommen.

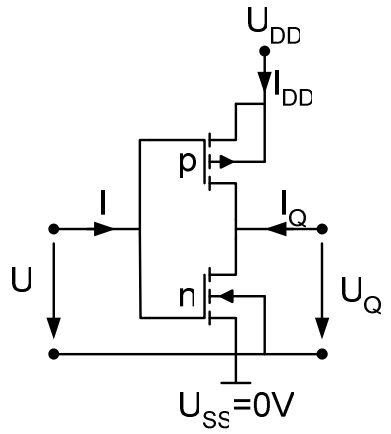


Bild 30: CMOS-Inverter

Kanal x des x/y-Schreibers wird an die Eingangs- und y an die Ausgangsspannung angeschlossen. Die Spannung U_{DD} beträgt, wie die maximale Spannung für U_I, 5V.

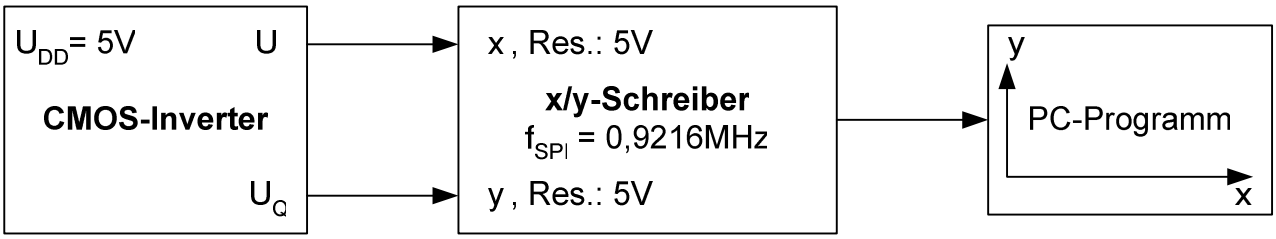


Bild 31: Testaufbau der Übertragungskennlinienaufnahme eines CMOS-Inverters

4.1.1.1 Messung mit 4096 Speicherwerten und Mittelwertbildung

Nach dem Anschließen und Einschalten aller Geräte wird das PC-Programm in Empfangsbereitschaft versetzt. Auflösungsänderungen, die durch die Taster des x/y-Schreibers vorgenommen werden, werden umgesetzt. Nach dem Drücken der Taste

„Start“ am x/y-Schreiber werden die Messpunkte dargestellt. Durch Veränderung der Eingangsspannung U_I am CMOS-Inverter wird die gesamte x-Achse abgefahren. Bild 32 zeigt die aufgenommene Übertragungskennlinie des CMOS-Inverters.

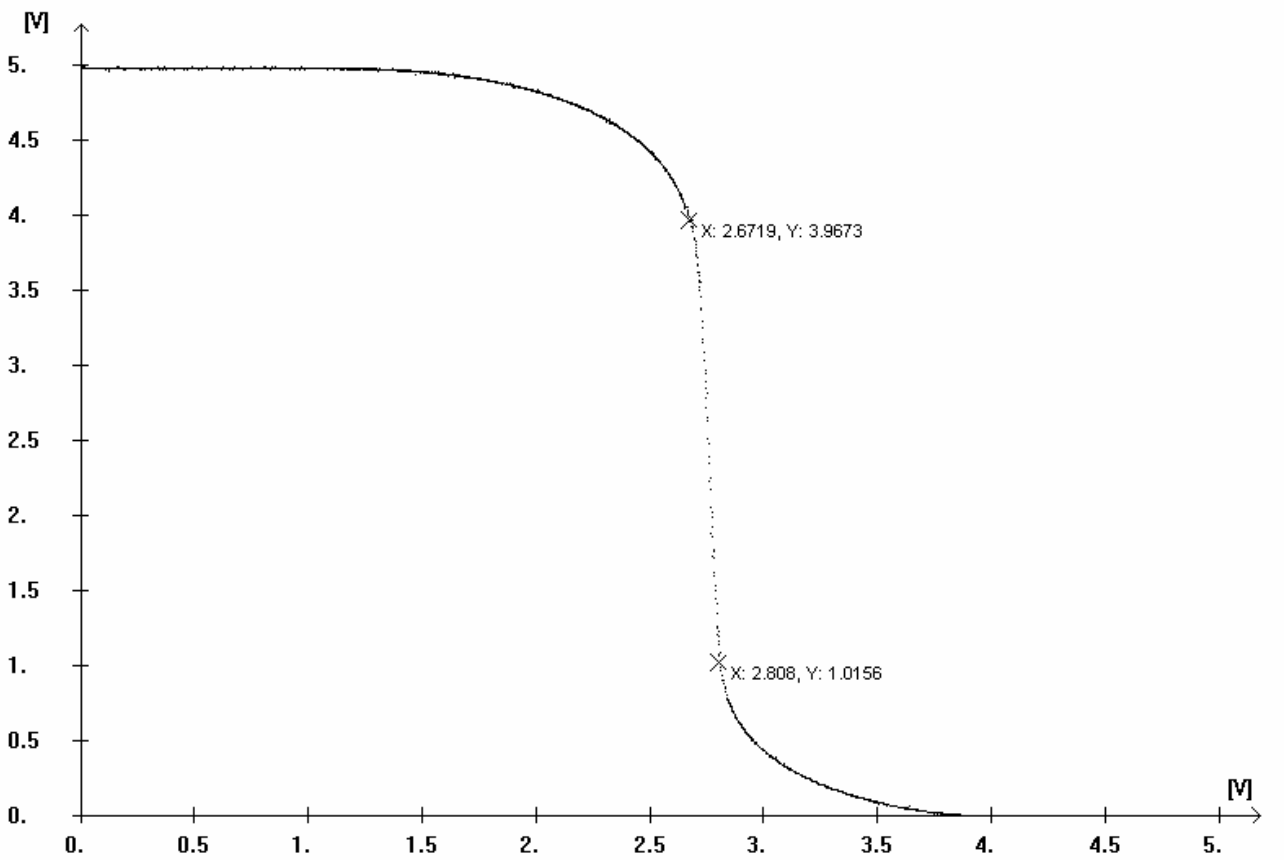


Bild 32: Übertragungskennlinie des CMOS-Inverters, 4096 Speicherpunke[44]

Durch mehrmaliges, bzw. langsames Abfahren des Eingangsspannungsbereiches kann die Aufnahme verbessert werden. Die Lücken in der stark vertikalen Steigung sind technisch bedingt, da die Punkte direkt dargestellt und an die Bildschirmauflösung angepasst werden.

Bei der Messung in Bild 32 wurden die Spannungen für wiederholt auftretende x-Werte gemittelt und in einem für die Auflösung von 12 Bit (4096 Speicherplätze) umfassenden Array abgelegt. Durch diese Mehrfachmessung einzelner Punkte kann die Genauigkeit der Kurve erhöht werden, es ist jedoch nicht möglich, mehrere Messwerte für einen x-Wert wie

[44] Messdaten auf CD in Datei „CMOS1.TXT“.

bei Hysteresekurven nötig, darzustellen. Hierfür müssen alle Messwerte getrennt und nacheinander entsprechend ihrem Auftreten gespeichert werden.

4.1.1.2 Messung mit sukzessiver Datenverwaltung

Die in Bild 33 dargestellten Graphen wurden nach Umstellung unter gleichen Bedingungen wie in Kapitel 4.1.1.1 aufgenommen. Die linke Kurve wurde mit einer Rampe mit der Dauer einer Sekunde als Eingangssignal und die rechte Kurve mit manueller, über einen längeren Zeitraum laufenden Spannungseinstellung aufgenommen. Das Bild wird mit längerer Messdauer, das heißt mehr Messungen für jede Eingangsspannung, unschärfer.

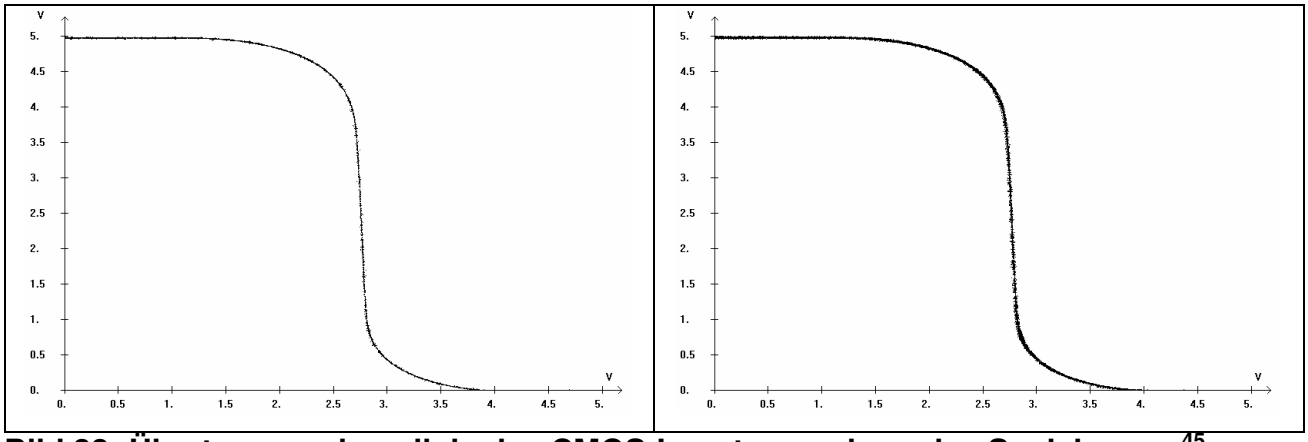


Bild 33: Übertragungskennlinie des CMOS-Inverters, sukzessive Speicherung[45]

Durch die sukzessive Speicherung der Daten ist es möglich, zum Beispiel Hysteresekurven eines Schmitt-Triggers[46] wie in Bild 34 darzustellen. Die vertikalen Linien sind durch Interpolation entstanden, da die Schaltvorgänge des Schmitt-Triggers sehr schnell sind.

[45] Messdaten auf CD “CMOS1_new_Rampe.TXT” und “CMOS1_new_manuell.TXT”.

[46] Es wurde ein Kanal eines vierfach NAND Schmitt-Triggers 4093 verwendet.

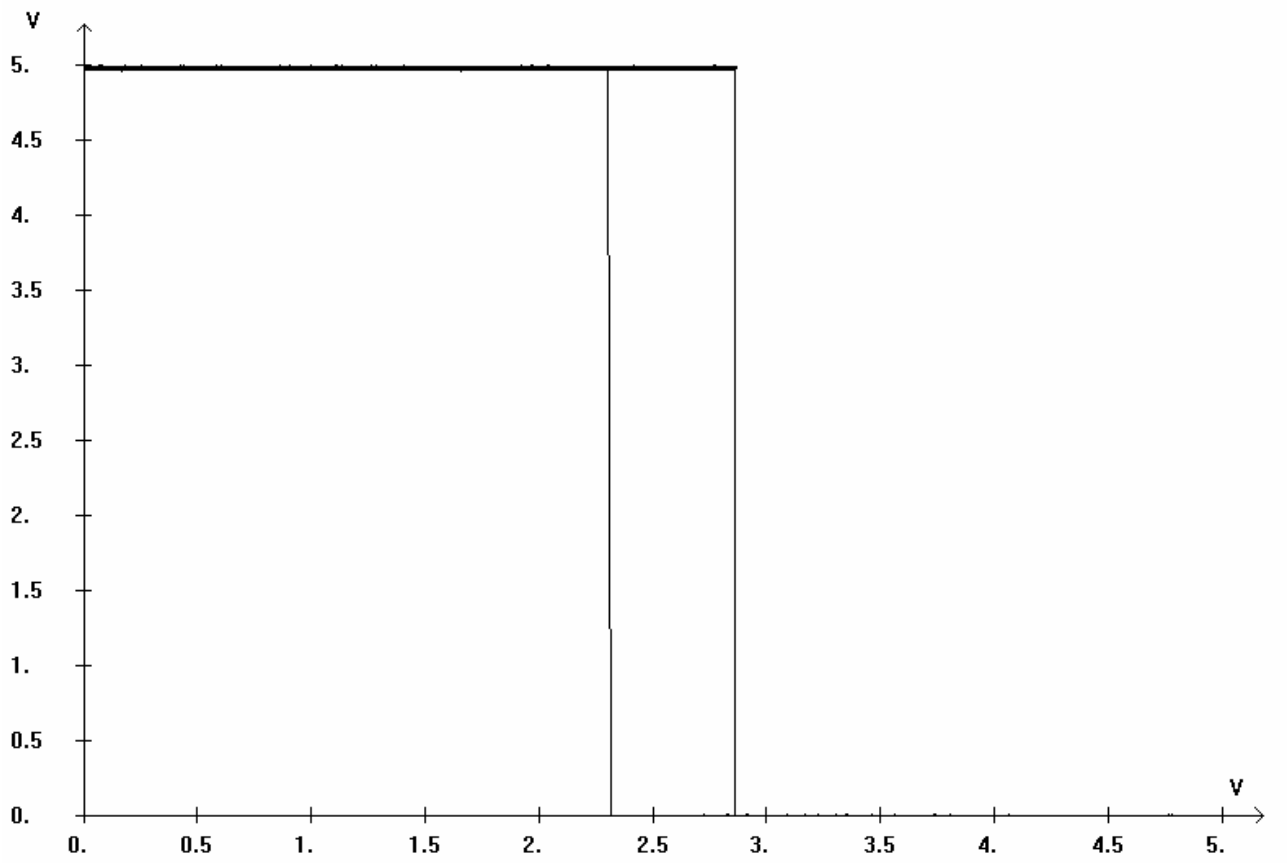


Bild 34: Hysteresekurve eines Schmitt-Triggers[47]

Durch die Umstellung auf die Aufnahme von 4096 auf 1.000.000 Messwerten ist wesentlich mehr Speicher erforderlich. Der Programmfluss vor allem am Start, aber auch beim Aufbau einer Verbindung zum x/y-Schreiber, hat sich dadurch verschlechtert.

4.1.2 Kennlinienaufnahme im Zeitbetrieb

Für den Zeitbetrieb wird kein Signal am x-Eingang des x/y-Schreibers benötigt. An y wurde in diesem Test ein periodisches Sinussignal mit Frequenzen zwischen 10 mHz und 100 Hz sowie einer Amplitude von V_{pp} = 5V angelegt.

[47] Messdaten auf CD „SchmittTrigger.TXT“.

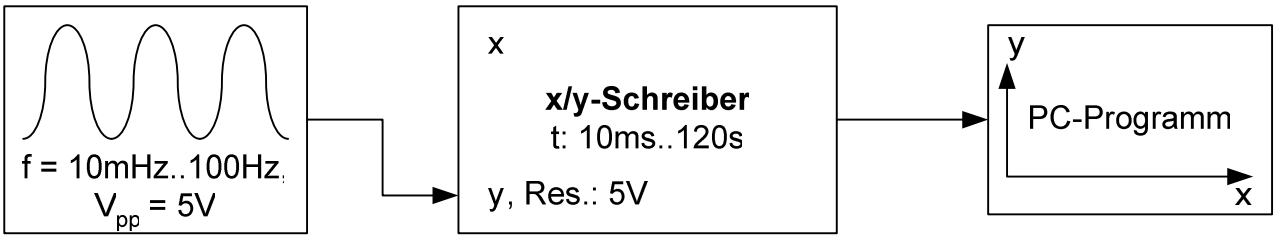


Bild 35: Testaufbau der Aufnahme eines Sinussignals im Zeitbetrieb

Nach Versetzen des PC-Programms in den Aufnahmemodus wird die Laufzeit des x/y-Schreibers entsprechend einer Periode des Eingangssignals eingestellt. Die Aufnahme des bereits anliegenden Sinussignals wird durch Betätigen des Tasters „Start“ begonnen.

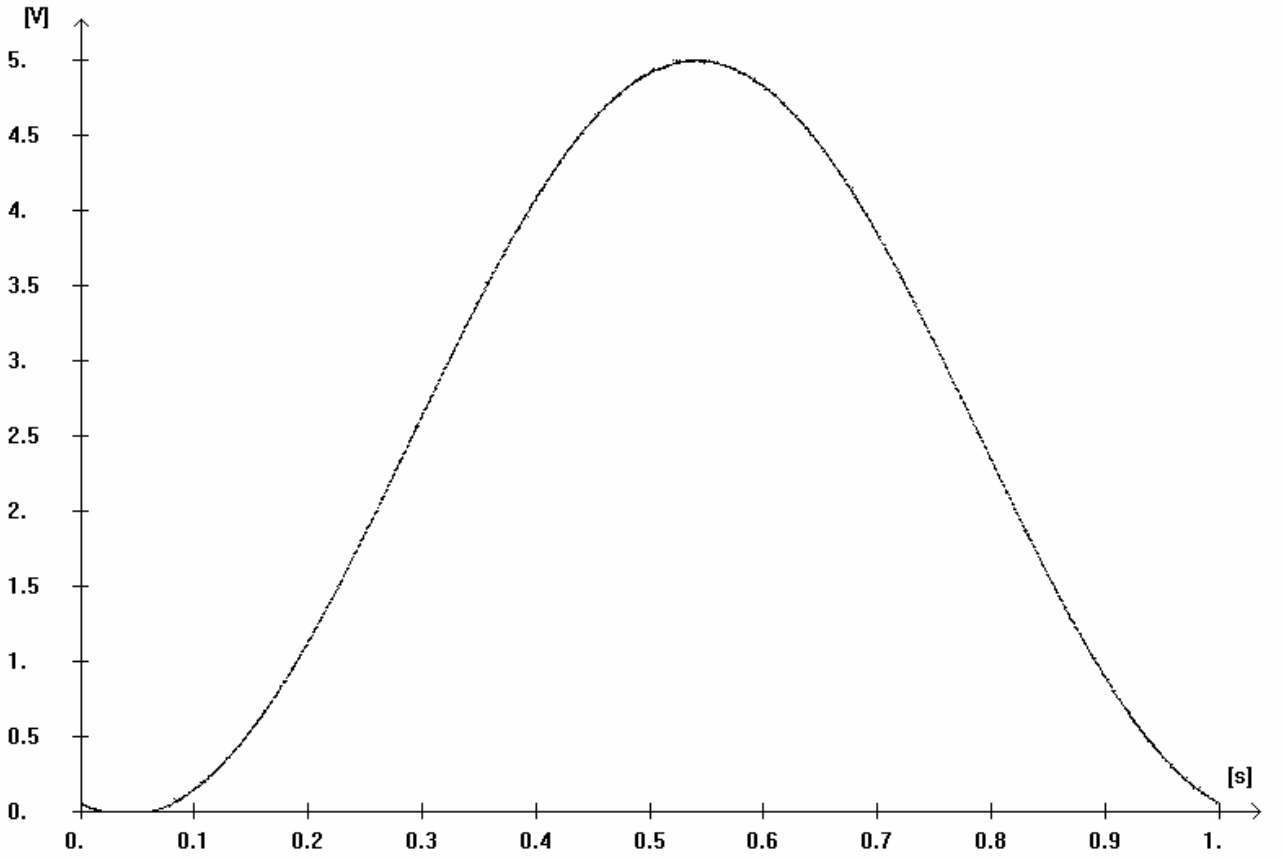


Bild 36: Aufnahme einer Periode eines Sinussignals mit der Frequenz 1 Hz

Die gemessene Kurve in Bild 36 stellt genau eine Periode eines Sinussignals dar. Die genauere Untersuchung der ersten und letzten Messpunkte erfolgt anhand der Messdaten[48].

[48] Messdaten auf CD „tySinus1Hz_SPI16_1s.TXT“.

t[49]	y[50]	y in [mV]	
0	49,33	60,221	linear errechnet
1	49	59,814	aus Messdaten
7	47	57,373	aus Messdaten
16373	53	64,697	aus Messdaten
16380	48	58,594	aus Messdaten
16383	45,86	55,978	linear errechnet

Tabelle 7: Auszug der Messdaten zu Bild 36

Die Überschneidung der beiden Endpunkte ist minimal, kann jedoch nicht genau angegeben werden, da die exakten Werte nicht aufgenommen wurden. Durch eine lineare Kalkulation erhält man für die Laufzeit die in Tabelle 7 für 0 und 16383 errechneten Werte von 60,221 mV und 55,978 mV. Die Überschneidung, bedingt durch die Laufzeit, beträgt damit 4,243 mV, was etwa 0,085% der Amplitude entspricht.
Die Datendichte, bedingt durch die Aufnahme- und Übertragungszeit der Messpunkte, wird in Kapitel 4.1.4 eingehend behandelt.

4.1.3 Kennlinienaufnahme im Zeitbetrieb mit Trigger

Die Funktion der getriggerten Kennlinienaufnahme wird anhand der gleichen Testparameter wie in der Kennlinienaufnahme im Zeitbetrieb in Kapitel 4.1.2 dargestellt. Das Eingangssignal mit einer Frequenz von 1 Hz wird für die Triggerauslösung zusätzlich auf den x-Kanal des x/y-Schreibers gegeben.

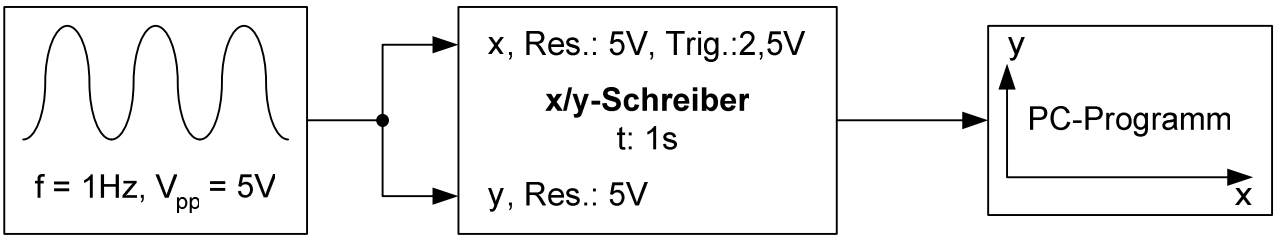


Bild 37: Testaufbau der Aufnahme eines Sinussignals im Zeitbetrieb mit Trigger

[49] Die Laufzeit wird mit einer Genauigkeit von 14 Bit gemessen. Die Zwischenwerte, z.B. zwischen 1 und 7 wurden nicht gemessen.

[50] Die Aufnahme von y erfolgt mit einer Genauigkeit von 12 Bit.

Nach der ersten Betätigung des Tasters „Start“, entsprechend der Abfolge in Kapitel 4.1.2, wird mit den Tastern für die Einstellung des y-Kanals die Triggerschwelle in diesem Fall mit 50% für 2,5 V so gewählt, dass die Graphen in Bild 38 daraus resultieren[51]. Die Einstellung wird mit erneutem Betätigen von „Start“ bestätigt.

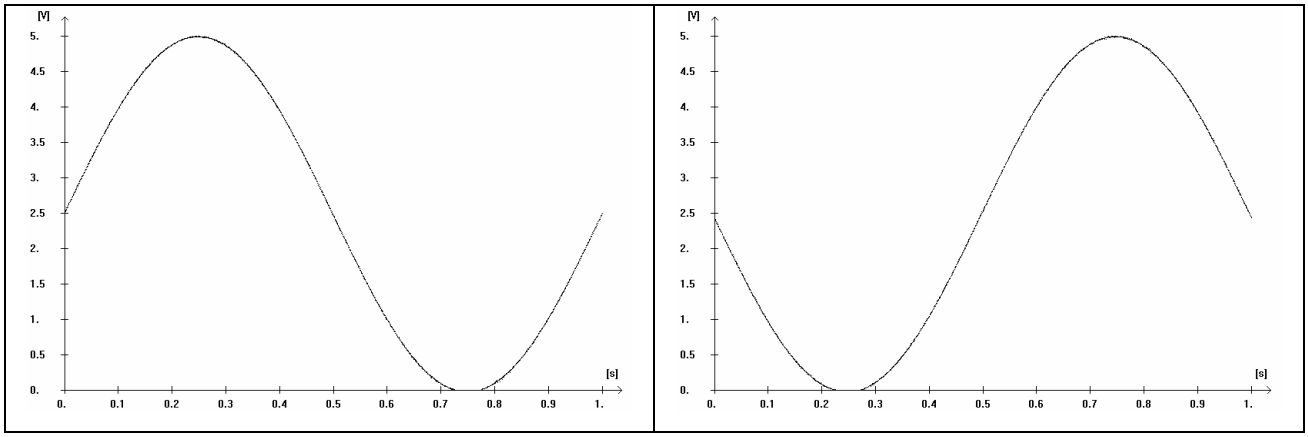


Bild 38: Getriggerte Aufnahme eines Sinussignal (steigende und fallende Flanke)

Die Auswahl, ob auf die steigende oder fallende Flanke getriggert werden soll, kann bei einem stetigen periodischen Signal nicht vorgegeben werden, sondern wird durch den Zeitpunkt des Aufnahmestarts bestimmt.
Bei nichtperiodischen Signalen kann der Triggerwert eingestellt und der x/y-Schreiber durch erneutes Betätigen von „Start“ in Aufnahmebereitschaft versetzt werden. Das PC-Programm sollte ebenfalls bereits in Aufnahmebereitschaft sein. Wird nun das Eingangssignal gestartet, beginnt der wartende x/y-Schreiber bei Überschreiten des Schwellenwertes mit der Aufnahme des Signals.
Eine genauere Betrachtung der Verzögerung nach Triggerauslösung erfolgt in Kapitel 4.1.6.

4.1.4 Aufnahmefrequenz im Zeitbetrieb

Für die Ermittlung der Aufnahmefrequenz im Zeitbetrieb ist der Aufbau entsprechend dem der Messung im x/y-Betrieb (siehe Kapitel 4.1.5). Das Signal am x-Kanal wird zur Trigge-

[51] Messdaten auf CD: „tyTrig25Sinus1Hz_SPI16_rising.TXT„ und „tyTrig25Sinus1Hz_SPI16_falling.TXT“.

rung der Aufnahme genutzt. Die Laufzeit des x/y-Schreibers wird für die Frequenz des Eingangssignals von 10 Hz variiert[52]. Getriggert wird auf 0,1 V.

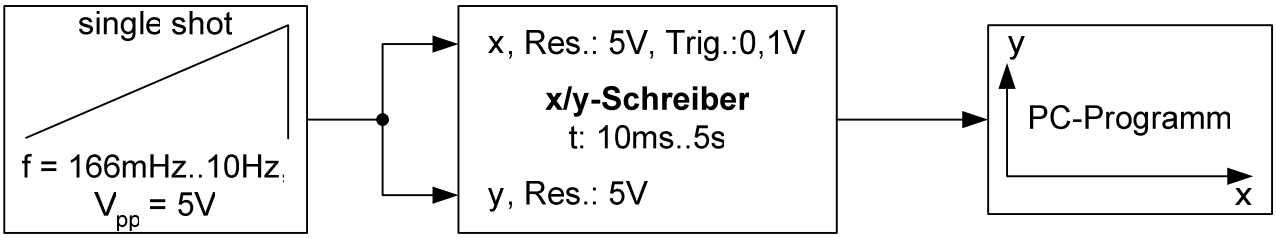


Bild 39: Testaufbau zur Aufnahmefrequenz im Zeitbetrieb

Bei einer Laufzeit von 100 ms ist die Rampe, bedingt durch die Triggerung auf 0,1 V und die leichte Startverzögerung beendet, bevor die Messung abgeschlossen ist (siehe Bild 40).

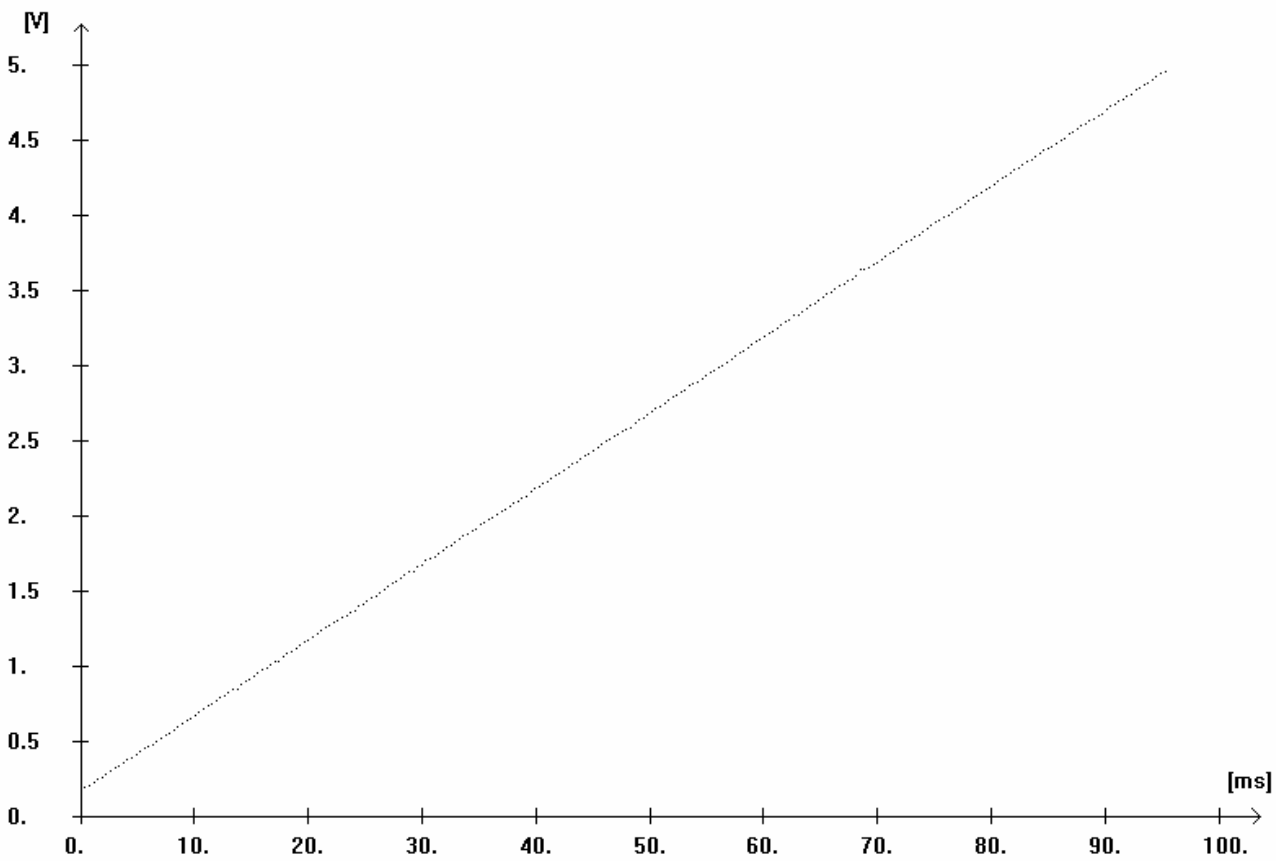


Bild 40: Aufnahme einer Rampe im Zeitbetrieb (Frequenz 10 Hz)

[52] Messdaten auf CD: „xyRamp10Hz_SingleShot_SPI16.TXT“.

Die Aufnahmefrequenz wird aus den Messungen für die Laufzeiten von 50ms[53] und 10ms[54] ermittelt. Für die Messung mit der Laufzeit von 50ms ergibt sich eine durchschnittliche Distanz von 120,46 bei 136 aufgenommenen Werten und einer Auflösung von 16 Bit. Dies entspricht einer Aufnahmezeit von etwa 0,368 ms und einer Frequenz von 2,72 kHz. Das Ergebnis bestätigt sich bei der Auswertung der Messung mit der Laufzeit von 10ms, bei der 27 Werte aufgenommen werden. Dies entspricht einer durchschnittlichen Distanz von 606,78 und der Aufnahmezeit von 0,370 ms.
Der Vergleich mit der später in Kapitel 4.1.5 im x/y-Betrieb erreichten Aufnahmezeit von 0,131 ms zeigt, dass die Umrechnung vom 16 Bit Timer Register auf die 14 Bit Auflösung etwa 0,24 ms benötigt. Weil der Timer für die verschiedenen Laufzeiten unterschiedliche Maximalwerte aufweist, das PC-Programm hingegen den maximal erreichbaren Wert der Auflösung als maximale Laufzeit interpretiert, ist diese Umrechnung notwendig.
Bei der Berechnung von Laufzeiten über einer Sekunde wird zusätzlich die Zählvariable „count“ verwendet, die die Anzahl der abgelaufenen Sekunden zählt. Diese steht dem PC-Programm nicht zur Verfügung, so dass die Berechnung wie folgt mit dem Mikroprozessor durchgeführt werden muss.

$$\text{value}0 = (\text{uint}16\_\text{t})\left((\text{TY\_MAX}-1)\cdot(\text{float})\left(\frac{1{,}0 * \text{value}0}{\text{ocrVal}} + \frac{\text{count}}{\text{count\_max}}\right)\right)$$

value0: Registerwert des Timers und berechnete 14-Bit-Ausgabe
TY_MAX: Maximalwert für 14 Bit (16384), auf die skaliert wird
ocrVal: maximaler Timerwert für die eingestellte Laufzeit
count: aktueller Zählerstand (inkrementiert durch Timerinterrupt)
count_max: maximale Durchlaufzahl für die eingestellte Laufzeit

4.1.4.1 Messung mit sukzessiver Speicherung

Nach der Anpassung der Aufnahme von Werten von einem festen 14 Bit Puffer auf einen Array mit 1.000.000 Speicherplätzen wird der Test aus Kapitel 4.1.4 erneut durchgeführt und die Ergebnisse verglichen. Im Zeitbetrieb sollte keine Änderung der Aufnahmefre-

[53] Messdaten auf CD: „tyTrig01Ramp10Hz_SingleShot_50ms.TXT“.
[54] Messdaten auf CD: „tyTrig01Ramp10Hz_SingleShot_10ms.TXT“.

quenz auftreten, da hier ein mehrfaches Auftreten gleicher Werte ohnehin nicht möglich ist.

Bei der Messung der Rampe mit der Laufzeit von 50 ms werden 136 Punkte aufgenommen. Mit einer Laufzeit von 10 ms werden 27 Punkte gemessen. Beide Messungen entsprechen exakt den Ergebnissen vor den Änderungen. Die Aufnahmefrequenz von 2,72 kHz bleibt erhalten.

Bei der Speicherung der Ergebnisse ergibt sich der Vorteil, dass keine Lücken mit ungemessenen Werten mehr auftreten, da ausschließlich gemessene Wertepaare gespeichert werden und nicht der gesamte Speicherpuffer aufgefüllt wird.

4.1.5 Aufnahmefrequenz im x/y-Betrieb

Anhand einer Rampe, die auf beide Kanäle gleichermaßen gegeben wird, kann die Zeitspanne zwischen den einzelnen Punktmessungen ermittelt werden. Die Messung wurde für verschiedene Frequenzen, das heißt unterschiedliche Steigungen einer einmalig abgefahrenen Rampe, durchgeführt. Die Kommunikationsfrequenzen entsprachen den oben angegebenen für den SPI-Takt von 0,9216 MHz und die USB-Übertragungsrate von 921.600 Bit pro Sekunde.

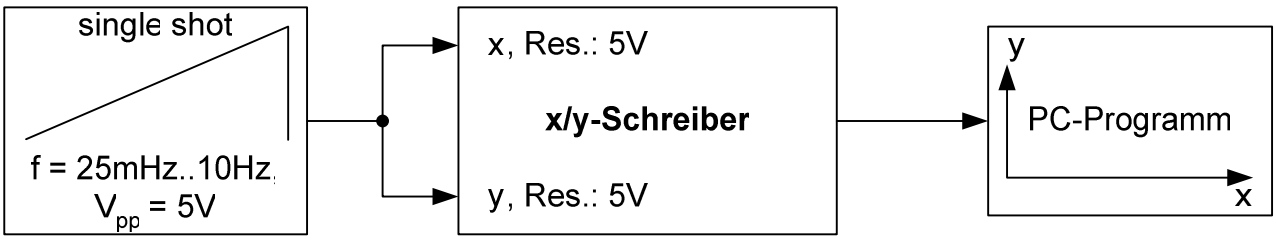


Bild 41: Testaufbau zur Aufnahmefrequenz im x/y-Betrieb

4.1.5.1 Messung mit LCD-Ausgabe

Da sich die Eingangssignale für x und y entsprechen, wird eine Gerade durch den Nullpunkt als Ergebnis erwartet. Als ein Beispiel für die verschiedenen Messungen wird die Aufnahme für die Eingangsrampe mit einer Frequenz von 1 Hz, das heißt einer Laufzeit von einer Sekunde[55], in Bild 42 dargestellt.

[55] Messdaten auf CD: „xyRamp1Hz_SingleShot_SPI16.TXT“.

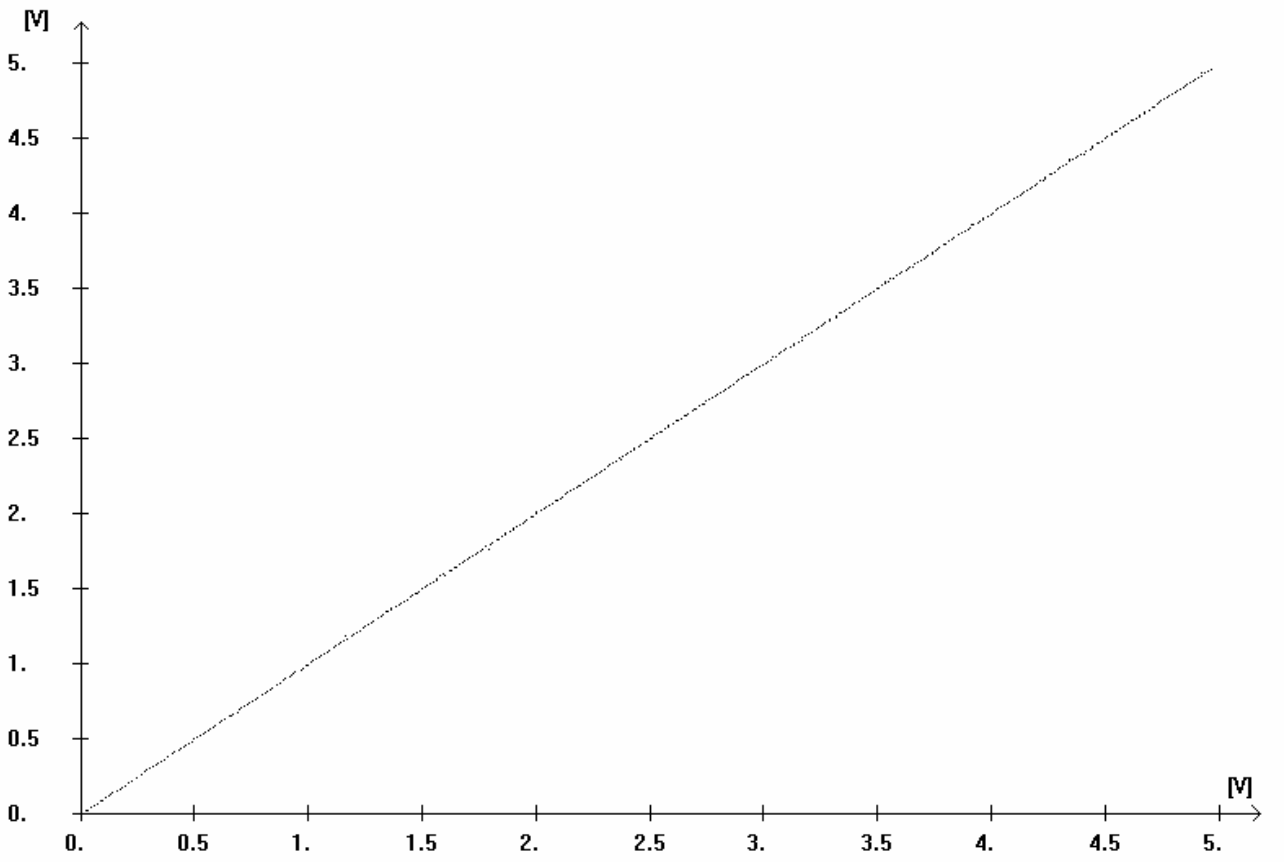


Bild 42: Aufnahme einer Rampe im x/y-Betrieb (Frequenz 1 Hz)

Für die genaue Untersuchung der Aufnahmefrequenz wird die gesamte Anzahl aufgenommener Werte aus den Messdaten ermittelt.

1 Hz			10 Hz		
x	**y**	**Δ x**	**x**	**y**	**Δ x**
2030	2028		1839	1839	
2038	2036	8	1928	1926	89
2047	2047	9	2020	2019	92
2053	2052	6	2119	2120	99
2064	2065	11	2215	2214	94

Tabelle 8: Messdaten einer Rampe im x/y-Betrieb für 1 Hz und 10 Hz (Auszug)

Für die Messung bei der in Bild 42 abgebildeten Rampe mit der Frequenz von 1 Hz ergibt sich für eine durchschnittliche Wertdistanz von 9,68 bei 423 aufgenommenen Werten und

einer Auflösung von 12 Bit eine Aufnahmezeit von etwa 2,36 ms und eine Frequenz von 423 Hz.

Das Ergebnis bestätigt sich bei der Auswertung einer Rampe mit einer Frequenz von 10 Hz[56], bei der 43 Werte aufgenommen werden. Dies entspricht einer Distanz von 95,26 und der Aufnahmezeit von 2,33ms.

Die verstrichene Zeit zwischen den einzelnen Messpunkten variiert zum Teil erheblich, wie auch in dem Auszug aus den Messdaten in Tabelle 8 zu sehen ist.

Im Vergleich zu der Aufnahmefrequenz im Zeitbetrieb mit 2,72 kHz[57] sind die im x/y-Betrieb erreichten 423 Hz sehr langsam. Dies kann nicht durch die Messung des zusätzlichen y-Kanals erklärt werden, da sich hierdurch die Gesamtzeit um nur etwa 28,71 µs[58] verlängern würde. Die Übertragung und Darstellung im PC-Programm verlaufen im Wesentlichen gleich. Es wird daher davon ausgegangen, dass die Umwandlung und die Darstellung der Messwerte auf dem Display viel Zeit in Anspruch nehmen. Belegt wird dies durch die Messungen in Kapitel 4.1.5.2 ohne die LCD-Ausgabe der aktuellen Messwerte.

4.1.5.2 Messung ohne LCD-Ausgabe

Nachfolgend werden die Messungen ohne die Ausgabe des aktuellen Wertes auf dem Display durchgeführt.

1 Hz			10 Hz		
x	**y**	**∆ x**	**x**	**y**	**∆ x**
2046	2047		2034	2039	
2047	2045	1	2044	2041	10
2048	2049	1	2047	2047	3
2049	2050	1	2049	2049	2
2051	2050	2	2056	2048	7

Tabelle 9: Messdaten wie in Tabelle 8, nur ohne LCD-Ausgabe

[56] Messdaten auf CD: „xyRamp10Hz_SingleShot_SPI16.TXT“.

[57] Siehe Kapitel 4.1.4. Aufnahmefrequenz im Zeitbetrieb

[58] Siehe Kapitel 3.2.3. Analog/Digital-Umsetzer MCP3202

Für die Messung einer Rampe der Frequenz 1 Hz ohne die Messausgabe auf dem Display[59] ergibt sich eine durchschnittliche Distanz von 1,24 bei 3309 aufgenommenen Werten und einer Auflösung von 12 Bit. Dies entspricht einer Aufnahmezeit eines Punktes von etwa 0,30 ms und einer Frequenz von 3,309 kHz. Anders als bei der Messung mit LCD-Darstellung wird das Ergebnis durch eine Rampe mit einer Frequenz von 10 Hz[60] nicht bestätigt. Es werden hier 762 Werte aufgenommen, die einer Distanz in den Messdaten von 5,38 entsprechen. Es ergibt sich eine Aufnahmezeit von 0,131 ms und damit einer Frequenz von 7,619 kHz.
Aus diesem Ergebnis lässt sich schließen, dass bei der Rampe mit der Frequenz 1 Hz einige Werte mehrfach aufgenommen wurden. Damit würde sich die durchschnittliche Distanz zwischen den Messpunkten verringern. Die Aufnahmefrequenz von 7,619 kHz wird etwa die maximal erreichbare sein, da zwischen den einzelnen Messungen fast immer eine Distanz Δ x größer als 1 besteht (vergleiche Tabelle 9).

	1 Hz		10 Hz	
	mit LCD	ohne LCD	mit LCD	ohne LCD
Aufnahmezeit in ms	2,33	0,30	2,36	0,131
Aufnahmefrequenz in kHz	423	3,309	424	7,619

Tabelle 10: Vergleich der Aufnahmefrequenz im x/y-Betrieb (mit und ohne LCD)

Aus der Gegenüberstellung der Ergebnisse in Tabelle 10 folgt eine bis zu 18-fache Erhöhung der Aufnahmefrequenz im x/y-Betrieb. Als Folge dieser Messung wurde als Standardeinstellung auf die Anzeige der aktuell gemessenen Werte im LC-Display verzichtet. Sie kann durch Setzten eines Jumpers über die Pins 15 und 16 des I/O-Headers aktiviert werden.

4.1.5.3 Aufnahmefrequenz mit sukzessiver Speicherung

Die Messung wird wie in Kapitel 4.1.5.2, ohne Ausgabe der Messwerte, auf dem LC-Display erneut durchgeführt. Die Ablage der gemessenen Wertepaare erfolgt konsequent

[59] Messdaten auf CD: „xyRamp1Hz_SingleShot_SPI16_woLCD.TXT“.

[60] Messdaten auf CD: „xyRamp10Hz_SingleShot_SPI16_woLCD.TXT“.

sukzessiv. Es wird keine Mittelwertbildung bei einer Mehrfachmessung mehr durchgeführt. Dadurch können verschiedene Messungen für denselben x-Wert aufgenommen und mit dem PC-Programm dargestellt werden.
Bei einer Rampe mit der Frequenz von einem Hertz[61] werden etwa 7919 und bei 10 Hz[62] werden etwa 797 Wertepaare aufgenommen. Genau kann die Anzahl nicht bestimmt werden, da der Beginn der Rampe aus den Messdaten nicht eindeutig bestimmt werden kann. Die Aufnahme des x/y-Schreibers beginnt bevor die Rampe gestartet wird. In Tabelle 11 werden die zu den obigen Anzahlen an Wertepaaren gehörenden Aufnahmefrequenzen der neuen Messungen mit den Ergebnissen aus Kapitel 4.1.5.2 (alt) vergleichend dargestellt.

	1 Hz		10 Hz	
	alt	neu	alt	neu
Aufnahmezeit in ms	2,33	0,126	0,131	0,125
Aufnahmefrequenz in kHz	423	7919	7,619	7,970

Tabelle 11: Vergleich der Aufnahmefrequenz im x/y-Betrieb

Die Aufnahmefrequenz hat sich im x/y-Betrieb noch leicht auf knapp 8 kHz erhöht. Durch den Wegfall der Mittelwertbildung von Mehrfachaufnahmen werden für alle Frequenzen des Eingangssignals gleich viele Werte aufgenommen. Es besteht kein Unterschied mehr zwischen der Aufnahmefrequenz eines Eingangssignals mit 1 Hz oder 10 Hz. Durch den endlichen Puffer für eine Million Messpunkte ergibt sich mit einer Aufnahmefrequenz von 8 kHz eine maximale Messdauer von 125 Sekunden.

Betrachtet man die Übertragungszeit bei der eingestellten Bitrate von 921.000 Bit/s genauer, kommt man der Aufnahmezeit von 0,125 ms recht nahe. Übertragen wird ein Datensatz von 6 Byte, das heißt 48 Bit. Hinzu kommt noch das USB-Protokoll von mindestens 24 Bit pro Datensatz und einem Tokenpaket von ebenfalls 24 Bit[63]. Zusammen ergibt dies 96 Bit, die mit der Bitrate in 0,104 ms übertragen werden können. Hinzu kommen

[61] Messdaten auf CD: „xyRamp1Hz_SingleShot_SPI16_woLCD_new.TXT.

[62] Messdaten auf CD: „xyRamp10Hz_SingleShot_SPI16_woLCD_new.TXT.

[63] Vergleiche das USB-Protokoll in Kapitel 2.2.3.

noch in regelmäßigen Abständen Steuerungs-Pakete. Da mit der Digital/Analog-Umsetzung nicht gewartet wird, bis die Daten übertragen wurden, fließt diese nicht in die Gesamtzeit mit ein, sondern es kann ohne Zeitverlust nach beendeter Übertragung erneut gesendet werden.

4.1.6 Verzögerung nach Triggerauslösung

Die Bestimmung der Verzögerung nach der Triggerauslösung bis zur Messung des ersten Wertes erfolgt anhand zweier aufeinander folgender Rampen mit relativ hohen Frequenzen. Der Trigger wird auf 2,5 V eingestellt.

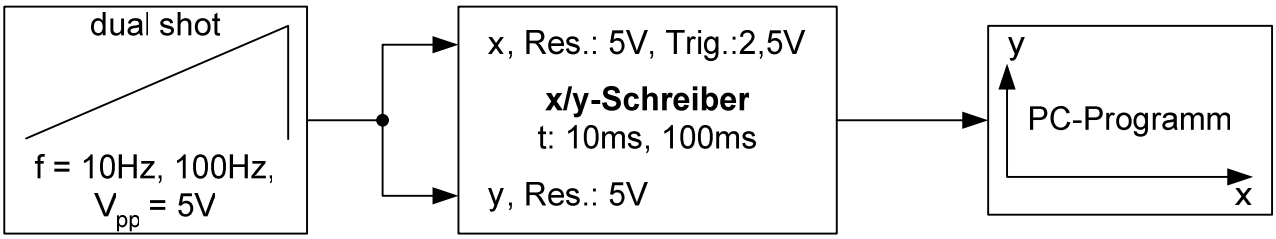


Bild 43: Testaufbau für die Messung der Verzögerung nach Triggerauslösung

4.1.6.1 Messung mit LCD-Ausgabe des Triggerwertes

Im Idealfall beginnt die Aufnahme mit 2,5 V, der Triggerschwelle. Die Messungen werden jeweils für die steigende und die fallende Flanke durchgeführt.
Schon in der grafischen Darstellung für eine Rampe mit 10 Hz[64] in Bild 44 ist die Verzögerung des ersten Messpunktes durch den Spannungsoffset gut zu erkennen.

[64] Messdaten auf CD: „tyTrig25Ramp100Hz_SingleShotx2_10ms_rising.TXT“ und „tyTrig25Ramp100Hz_SingleShotx2_10ms_falling.TXT“.

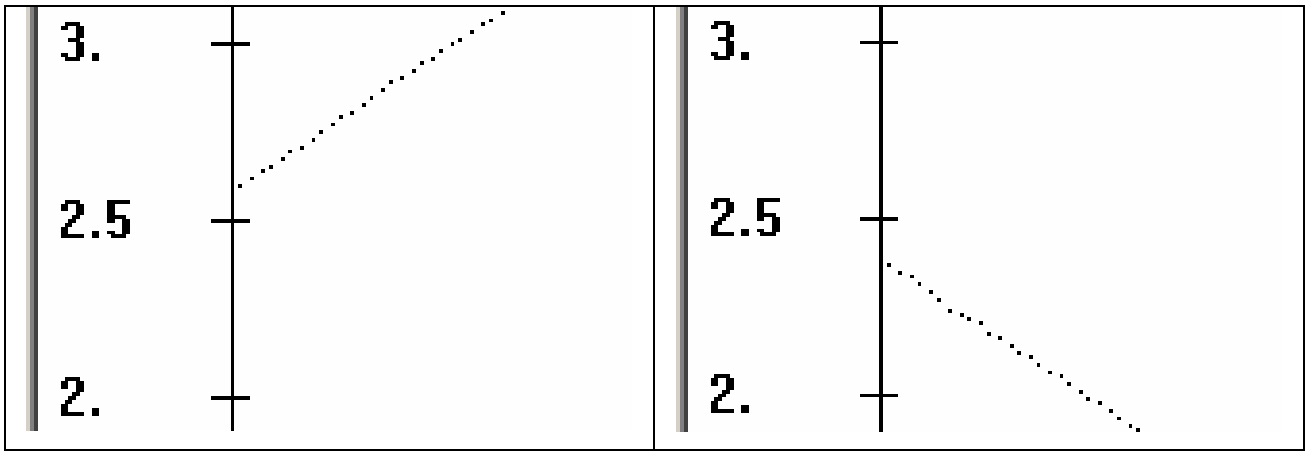


Bild 44: Triggerverzögerung für steigende und fallende Rampe mit 10 Hz

Anhand der Messdaten wird diese Verzögerung für die fallende und steigende Flanke bestimmt. Die Laufzeit der Aufzeichnung beträgt für 10 Hz 100 ms und für 100 Hz 10 ms[65], bei einer Auflösung für die Zeit (t) von 14 Bit und y von 12 Bit.

Frequenz	Flanke	t	y	t in µs	y in V
10 Hz	steigend	6	2119	36,623	2,587
10 Hz	fallend	6	1957	36,623	2,389
100 Hz	steigend	62	2942	37,844	3,591
100 Hz	fallend	62	1271	37,844	1,551

Tabelle 12: Triggerverzögerung für 10 Hz und 100 Hz

Die Messung beginnt um etwa 37 µs verzögert nach Überschreiten der Triggerschwelle. Beim Vergleich der y-Werte wird der Einfluss der Frequenz, und damit der Steigung des Eingangssignals, auf die Auswirkung dieser Verzögerung, bezüglich der Spannungsabweichung, deutlich. Für 100 Hz beträgt der Offset etwa 1 V, sowohl bei der steigenden, wie auch der fallenden Flanke. Die Ausgabe des aktuellen Offset im LC-Display verschlechtert eventuell das Auslösen des Triggers. Dies wird im folgenden Kapitel genauer betrachtet.

[65] Messdaten auf CD: „tyTrig25Ramp10Hz_SingleShotx2_100ms_rising.TXT„ und „tyTrig25Ramp10Hz_SingleShotx2_100ms_falling.TXT“.

4.1.6.2 Messung ohne LCD-Ausgabe des Triggerwertes

Wie schon bei der Messung der Aufnahmefrequenz des x/y-Betriebes wird auch hier der Einfluss der Ausgabe des aktuellen Wertes auf dem LC-Display genauer betrachtet.

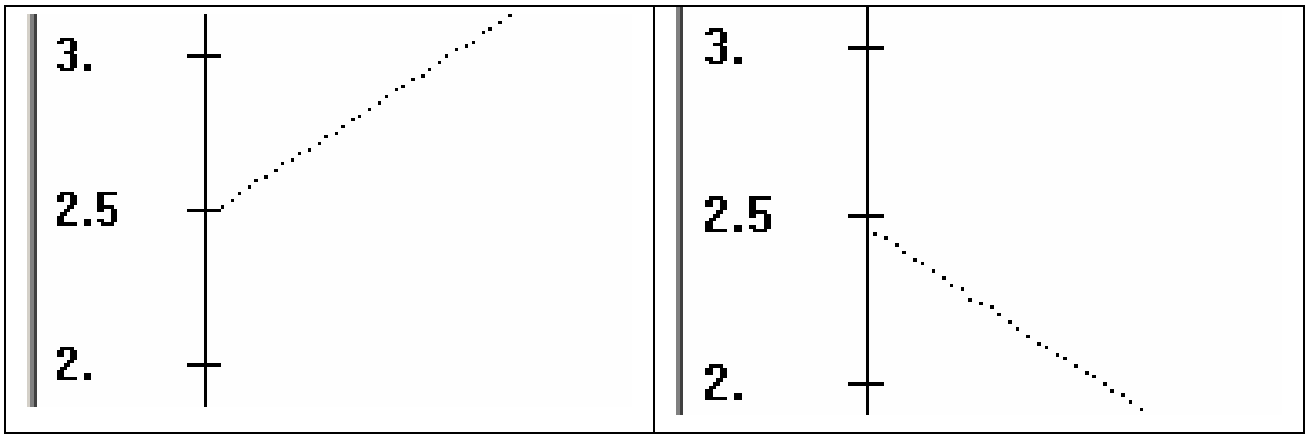


Bild 45: Triggerverzögerung (10 Hz) ohne LCD-Ausgabe

Die Verzögerung ist augenscheinlich geringer geworden. In den Messdaten der beiden Messungen wird dieses noch deutlicher.

Frequenz	Flanke	t	y	t in [µs]	y in [V]
10 Hz[66]	steigend	5	2035	30,052	2,484
10 Hz[67]	fallend	6	2019	36,623	2,465
100 Hz[68]	steigend	62	2095	37,844	2,557
100 Hz[69]	fallend	61	1943	37,234	2,371

Tabelle 13: Triggerverzögerung für 10 Hz und 100 Hz ohne LCD-Ausgabe

Die zeitliche Verzögerung hat sich praktisch nicht verändert, da in beiden Fällen der Timer vor der Digital/Analog-Umsetzung gestartet und danach ausgelesen wird. Das Einsetzen der A/D-Umsetzung erfolgt ohne die Display-Ausgabe früher, wodurch der Offset von der Triggerschwelle von 2,5 V erheblich verringert wird.

---

[66] Messdaten auf CD: „tyTrig25Ramp10Hz_SingleShotx2_100ms_woLCD_rising.TXT“.

[67] Messdaten auf CD: „tyTrig25Ramp10Hz_SingleShotx2_100ms_woLCD_falling.TXT“.

[68] Messdaten auf CD: „tyTrig25Ramp100Hz_SingleShotx2_10ms_woLCD_rising.TXT“.

[69] Messdaten auf CD: „tyTrig25Ramp100Hz_SingleShotx2_10ms_woLCD_falling.TXT“.

		10 Hz		100 Hz	
		mit LCD	**ohne LCD**	**mit LCD**	**ohne LCD**
steigend	t in µs	36,623	30,052	37,844	37,844
	y in V	2,587	2,484	3,591	2,557
fallend	t in µs	36,623	36,623	37,844	37,234
	y in V	2,389	2,465	1,551	2,371

Tabelle 14: Vergleich der Triggerverzögerung (mit und ohne LCD)

Durch Deaktivieren der Eingangsspannungsanzeige für den Triggereingang kann der Offset von der Triggerschwelle besonders für höhere Frequenzen verringert werden.
Die Anzeige des aktuellen Triggersignals wurde deaktiviert, kann aber zusammen mit der Punktanzeige im x/y-Betrieb[70] durch einen Jumper über Pin 15 und 16 des I/O-Headers aktiviert werden. Dadurch erhöht sich der Bedienkomfort, die Genauigkeit wird jedoch verringert.

4.1.7 Zeitfaktor in einem Datensatz des x/y-Betriebes

Für die Datenaufnahme im x/y-Betrieb werden die beiden Eingangskanäle nacheinander mit dem ADC gemessen. In Kapitel 3.2.3 wurde die Erfassung eines Datensatzes anhand der Taktfrequenz des SPI-Busses berechnet. Die Messungen wurden vor der Einführung der sukzessiven Datenspeicherung mit Mittelwertbildung durchgeführt. Dies hat auf diese Untersuchung jedoch keinen Einfluss, da sich hierdurch die Messzeit zwischen den beiden Kanälen nicht verändert.

4.1.7.1 Dreiecksignal bei einem SPI-Takt von 0,9216 MHz

Mit einer Taktfrequenz von 0,9216 MHz, die mit einem Bitratenquarz von 14,7456 MHz laut Datenblatt maximal möglich ist, dauert die errechnete Analog/Digital-Umsetzung 28,71 µs[71].

[70] Siehe Kapitel 4.1.5.2 Messung ohne LCD-Ausgabe.

[71] Siehe Kapitel 3.2.3 und [19] MCP3202 Datenblatt.

Für die Messung des Zeitversatzes zwischen beiden Werten eines Datensatzes wird an beide Eingänge dasselbe symmetrische und periodische Dreieckssignal mit einer Frequenz von 1 kHz und einer Amplitude von 5 V angelegt.

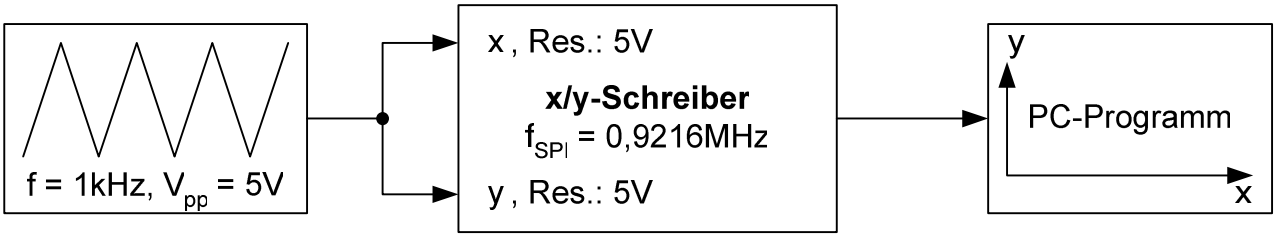


Bild 46: Testaufbau für die Messung des Zeitversatzes in einem Datensatz

Bei der Messung gibt es zu unterscheidende Messpunkte für die drei Phasen des Eingangssignals. Phase eins ist eine Messung während des Spannungsanstieges, Phase zwei während des Abfallens und Phase drei bei einer Messung beim Wechsel von Anstieg zu Abfall oder umgekehrt. In den Messdaten werden bei den ersten beiden Phasen die größten Differenzen zwischen den beiden A/D-Umsetzungen des x- und y-Kanals bestehen. Messpunkte der Phase drei werden sich im Zwischenraum dieser Extremwerte befinden. Da das Signal periodisch ist, werden gleiche Messpunkte, das heißt, das wiederholte Auftreten der gleichen Spannung als x-Wert, miteinander verrechnet und liegen dann ebenfalls zwischen den Maximalabweichungen (siehe Bild 28 für den Ablauf der Datenaufnahme).

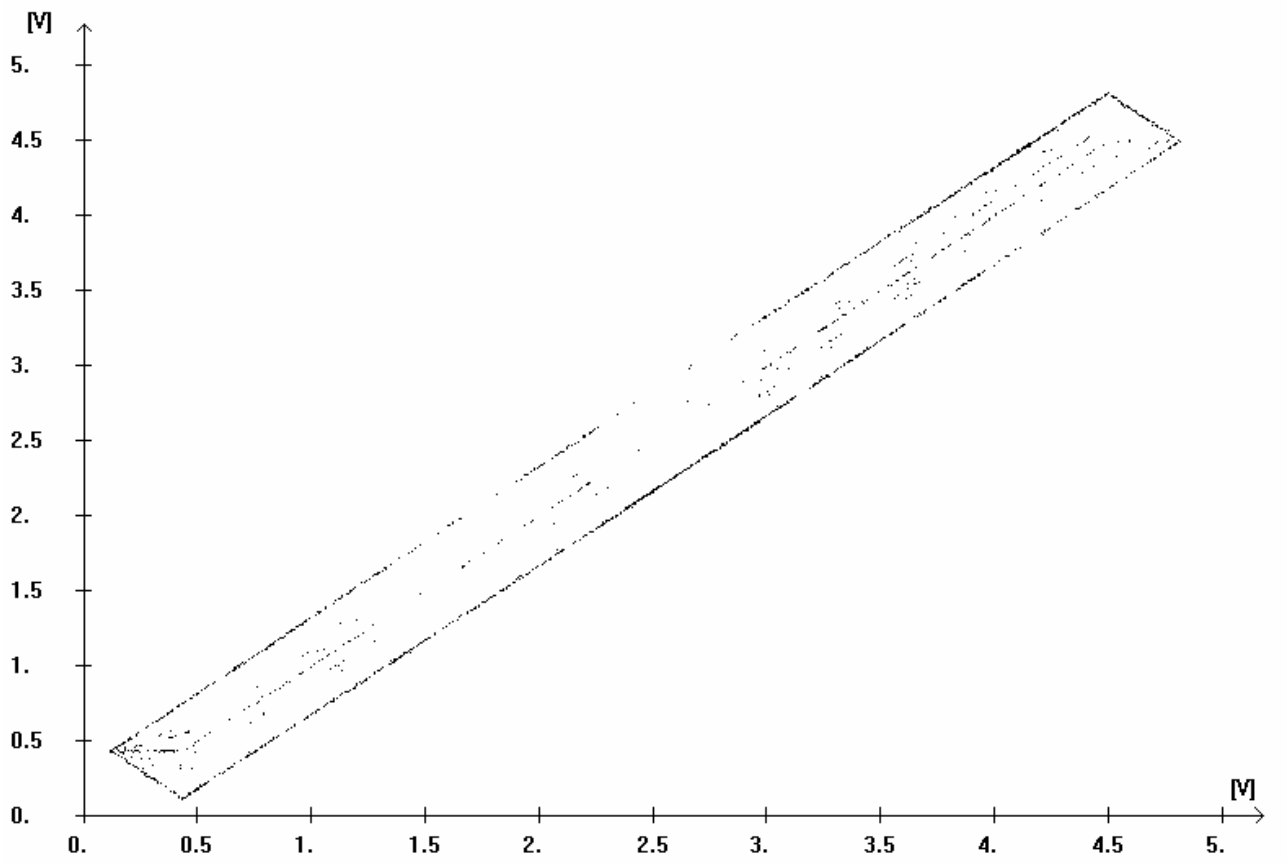


Bild 47: Messung eines Dreiecksignals mit einer Frequenz von 1 kHz

In der Messung ist der parallele lineare Verlauf der maximalen Abweichung an der fallenden und steigenden Flanke des Eingangssignals gut zu erkennen. Die im Zwischenraum befindlichen Punkte sind die oben beschriebenen Mehrfachmessungen oder Messungen an den Maxima der Dreieckspannung. Für eine genauere Untersuchung wird aus den gesamten Messdaten[72] ein geeigneter Ausschnitt mit einem Wechsel zwischen den Messungen einer steigenden und abfallenden Flanke gewählt.

x	**y**	**Δ xy**	**Δ xy in V**
2429	2151	278	0,339
2431	2158	273	0,333
2433	2696	263	0,321
2434	2708	274	0,334

Tabelle 15: Auszug aus den Messdaten eines Dreiecksignals (1 kHz)

[72] Messdaten auf CD: „xyDreieck1kHz_SPI16.TXT“.

Alle dazwischen liegenden Datenpaare waren bisher nicht erfasst oder keine Maximalwerte. Die Messung eines geringeren Wertes für y als für x ergibt sich bei einer fallenden Flanke, umgekehrt bei einer steigenden Flanke des Eingangssignals. Durch die verzögerte Messung tritt bei oben definiertem Eingangssignal eine Abweichung von 0,322 bis 0,355 V auf.

In 0,5 ms steigt oder fällt die Spannung des Eingangssignals um 5 V. Mit Hilfe dieser Steigung kann die Dauer einer A/D-Umsetzung wie folgt berechnet werden. Als Δ xy wird der Mittelwert von 0,321 und 0,339 V zum Ausgleich von Asymmetrien verwendet.

$$t_{ADC} = \frac{\Delta xy \cdot 0{,}5ms}{5V} = \frac{0{,}330V \cdot 0{,}5ms}{5V} = 33{,}00\mu s$$

Mit 33 µs liegt die gemessene Zeit einer A/D-Umsetzung im Bereich der errechneten mit 28,71 µs.

Die Messung wurde mit 750 Hz[73], 500 Hz[74] und 250 Hz[75] erneut durchgeführt und führte zu folgenden Messergebnissen.

x	**y**	**Δ xy**	**Δ xy in V**
2560	2347	213	0,260
2561	2767	206	0,251
2565	2774	209	0,255
2566	2359	207	0,253

Tabelle 16: Auszug aus den Messdaten eines Dreiecksignals (750 Hz)

[73] Messdaten auf CD: „xyDreieck750Hz_SPI16.TXT“.

[74] Messdaten auf CD: „xyDreieck500Hz_SPI16.TXT“.

[75] Messdaten auf CD: „xyDreieck750Hz_SPI16.TXT“.

x	y	Δ xy	Δ xy in V
2511	2369	142	0,173
2512	2648	136	0,166
2516	2377	139	0,170
2518	2657	139	0,170

Tabelle 17: Auszug aus den Messdaten eines Dreiecksignals (500 Hz)

x	y	Δ xy	Δ xy in V
2517	2446	71	0,087
2518	2582	64	0,078
2522	2593	71	0,087
2523	2452	71	0,087

Tabelle 18: Auszug aus den Messdaten eines Dreiecksignals (250 Hz)

Der Mittelwert der Differenzen bei den einzelnen Frequenzen ist in Tabelle 19 dargestellt. Darin kann man schon den linearen Verlauf des Fehlers erkennen, der in Bild 48 eindeutig zu sehen ist.

f [Hz]	Δ xy in V
250	0,08475
500	0,16975
750	0,25475
1000	0,33175

Tabelle 19: Fehler durch Zeitversatz bei verschiedenen Frequenzen

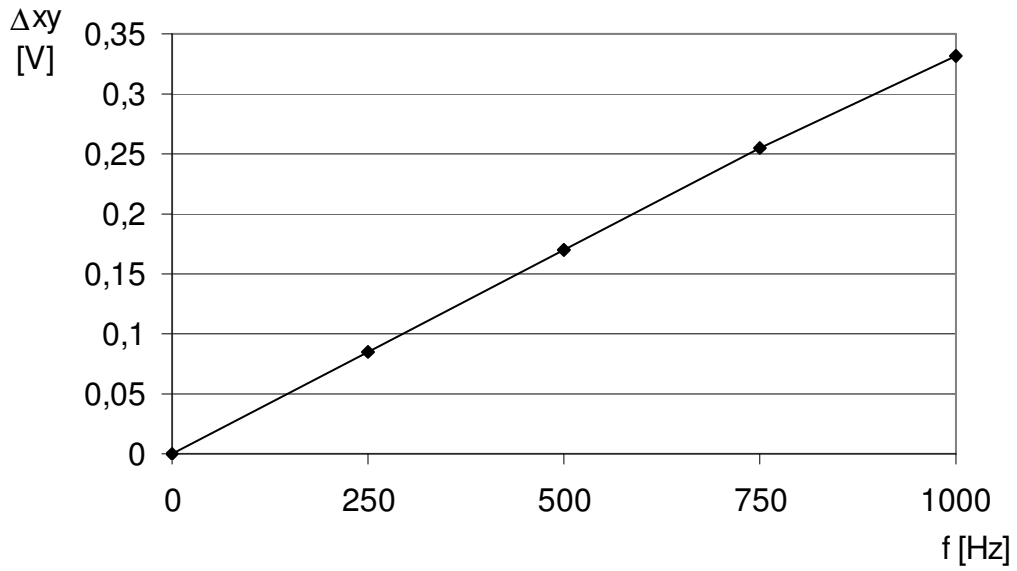


Bild 48: Fehler durch Zeitversatz bei verschiedenen Frequenzen

4.1.7.2 Dreiecksignal bei einem SPI-Takt von 1,8432 MHz

Die Taktfrequenz des SPI-Busses wird in dieser Testreihe auf 1,8432 MHz über die als maximal spezifizierte Frequenz verdoppelt[76].

Es wird wie in Kapitel 4.1.7.1 verfahren und an beiden Eingängen dasselbe symmetrische und periodische Dreiecksignal mit einer Frequenz von 1 kHz und einer Amplitude von 5 V angelegt[77].

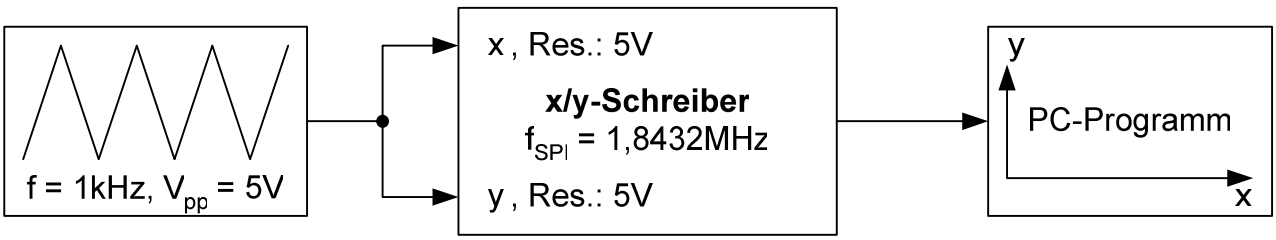


Bild 49: Vergleichsmessung mit SPI-Takt von 1,8432 MHz

[76] MCP3202 Datenblatt [19], Seite 3.

[77] Messdaten auf CD: „xyDreieck1kHz_SPI8.TXT“.

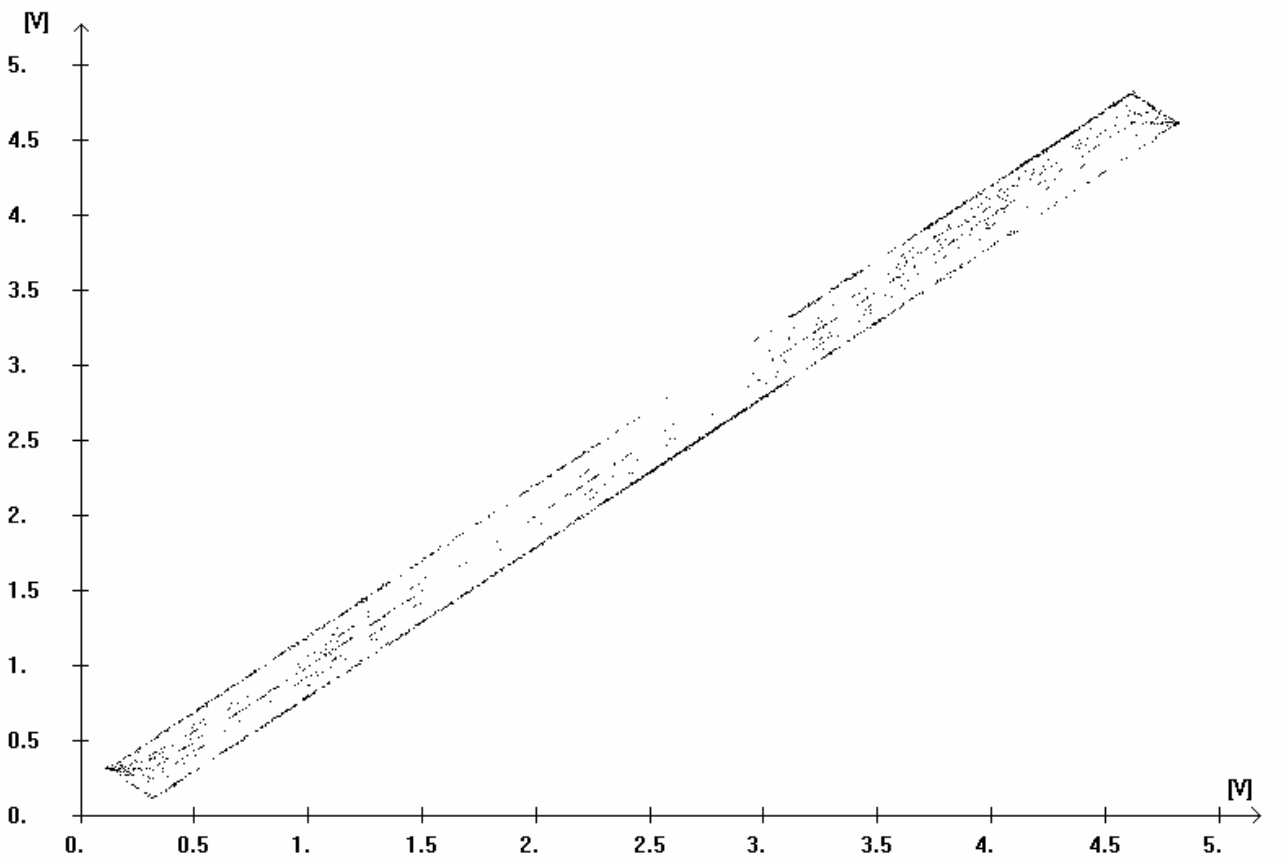


Bild 50: Dreiecksignal mit einer Frequenz von 1 kHz (SPI-Takt 1,8432 MHz)

x	y	Δ xy	Δ xy in V
2646	2816	170	0,208
2649	2477	172	0,210
2650	2478	172	0,210
2651	2819	168	0,205

Tabelle 20: Auszug aus den Messdaten bei 1 MHz (SPI-Takt 1,8432 MHz)

Der Vergleich zwischen den beiden Graphen in Bild 47 mit einem SPI-Takt von 0,9216 MHz und Bild 50 mit 1,8432 MHz zeigt, dass der Abstand der parallelen Linien der maximalen Abweichung mit verdoppeltem SPI-Takt geringer wird. Belegt wird dies durch die Auszüge aus den Messdaten (Tabelle 15 und Tabelle 20).

Für die Berechnung der Dauer einer A/D-Umsetzung t_{ADC} wird für Δ xy wieder der Mittelwert genommen.

$$t_{ADC} = \frac{\Delta xy \cdot 0{,}5ms}{5V} = \frac{0{,}208V \cdot 0{,}5ms}{5V} = 20{,}80\mu s$$

Mit 20,80 µs liegt die gemessene Zeit einer A/D-Umsetzung nur wenig über der in Kapitel 3.2.3 errechneten von 14,61µs. Während bei den errechneten Werten eine Halbierung der Umsetzzeit auftrat, verringert sie sich bei doppeltem SPI-Takt in der Messung nur von 33,00 µs auf 20,80 µs. Eventuell werden einzelne Flanken der über der Spezifikation liegenden Taktfrequenz nicht erkannt, so dass sich eine längere Umsetzzeit ergibt.

4.2 Vorschaltbarer Verstärker

Aufgrund des Tests der ursprünglichen Verstärkerschaltung sind mehrere Änderungen, die nicht mehr vollständig in die Testplatine eingearbeitet werden konnten, an dem Schaltplan und dem Platinenlayout[78] vorgenommen worden. Dies betrifft hauptsächlich die Offsetschaltungen und den Überspannungsschutz. Für beide Kanäle ist jetzt ein Offset vorgesehen, dem direkt vor dem Ausgang der Verstärkerplatine und damit dem A/D-Umsetzer auf der Steuerplatine eine Spannung zwischen 0 und +5 Volt zu der verstärkten Eingangsspannung hinzuaddiert wird.
Die Widerstände an den nichtinvertierenden Eingängen der Operationsverstärker der Dekadenverstärkerstufen betrugen 68 Ω, und nicht 1 kΩ, wie in den Schaltplänen im Anhang angegeben. Diese Änderung soll eine zusätzliche Tiefpass-Filterung des Signals mit der Eingangskapazität des Verstärkers bewirken. Zudem wird empfohlen, die Auswirkung der Trennung der Massen zwischen der Steuerungsplatine und dem Verstärker zusammen mit dem A/D-Umsetzer über einen Tiefpassfilter zu untersuchen.
Alle nachfolgenden Tests wurden mit einer Verstärkerplatine getestet, die nur einen umgerüsteten und funktionstüchtigen Kanal hatte. Der Aufbau der beiden Kanäle ist identisch und die Verstärkung kann vollständig im Zeitbetrieb untersucht werden.

4.2.1 Einstellung der Verstärkung

Der Funktionsnachweis der Einstellung der Verstärkung wird durch die Messung eines Sinussignals mit der Frequenz von einem Hertz und Amplituden von 1 V[79], 0,1 V[80], 10 mV[81] und 2 mV[82] gegeben. Die verschiedenen Spannungen werden in entsprechenden Spannungsbereichen mit dem x/y-Schreiber gemessen. In Tabelle 21 sind die Ergebnisse dargestellt. Die 4000-fache Verstärkung konnte nicht mehr aufgenommen werde, da die minimale Amplitude des Funktionsgenerators HP3312A 2 mV beträgt.

---

[78] Siehe Anhang A.1 und A.2.

[79] Messdaten auf CD: „Sinus_1Hz_1V_1s_5V.TXT“ , „...1s_2.5V.TXT“, „...1s_1.25V.TXT“.

[80] Messdaten auf CD: „Sinus_1Hz_0.1V_1s_0.5V.TXT“, „...1s_0.25V.TXT“, „...1s_0.125V.TXT“.

[81] Messdaten auf CD: „Sinus_1Hz_0.01V_1s_0.05V.TXT“, „...1s_0.025V.TXT“, „...1s_0.0125V.TXT“.

[82] Messdaten auf CD: „Sinus_1Hz_0.002V_1s_0.005V.TXT“, „Sinus_1Hz_0.002V_1s_0.0025V.TXT“.

Die Präzision der Verstärkung und das Rauschen wird in den folgenden Kapiteln genauer betrachtet.

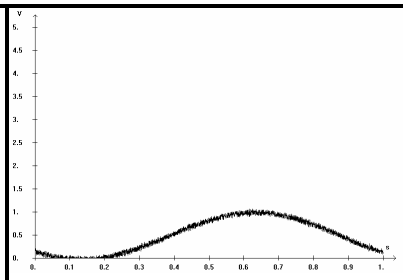
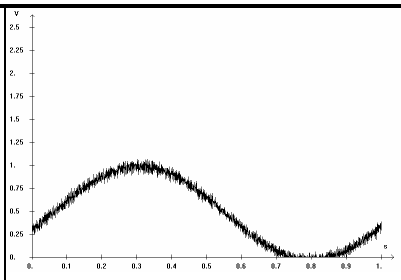
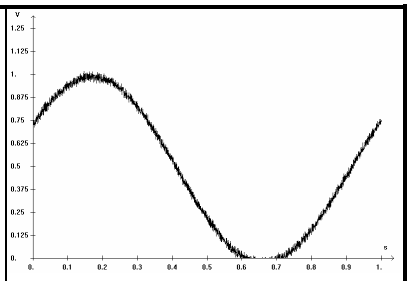
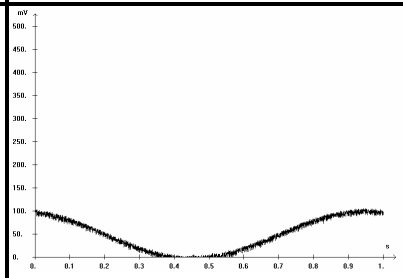
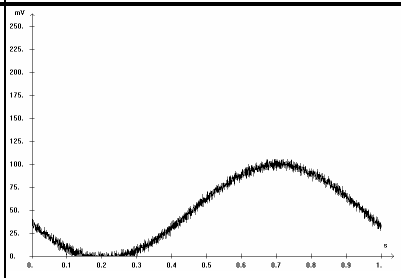
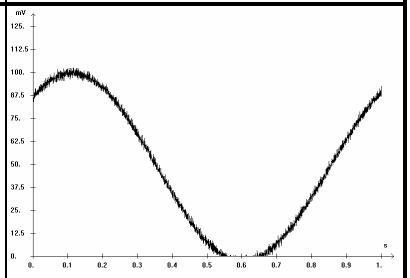
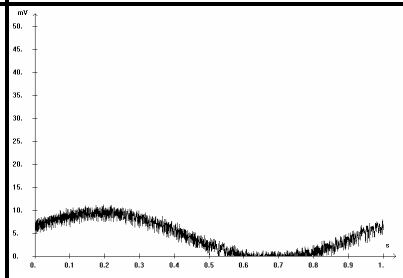
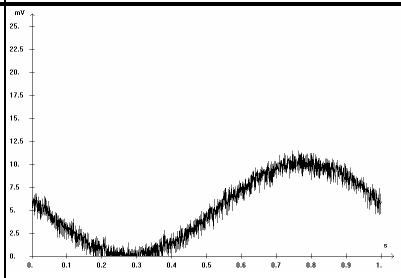
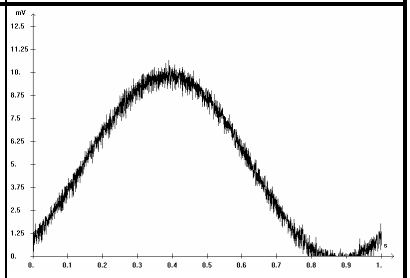
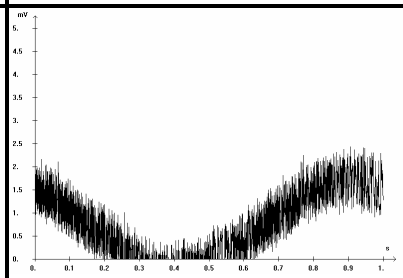
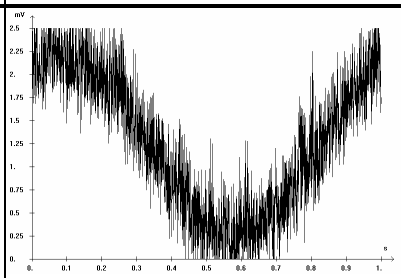
Sinus 1Hz, V_{pp} = 1V	 y-Messbereich: 5 V 1-fache Verstärkung	 y-Messbereich: 2,5 V 2-fache Verstärkung	 y-Messbereich: 1,25 V 4-fache Verstärkung
Sinus 1Hz, V_{pp} = 0,1V	 y-Messbereich: 0,5 V 10-fache Verstärkung	 y-Messbereich: 0,25 V 20-fache Verstärkung	 y-Messbereich: 0,125 V 40-fache Verstärkung
Sinus 1Hz, V_{pp} = 10mV	 y-Messbereich: 50 mV 100-fache Verstärkung	 y-Messbereich: 25 mV 200-fache Verstärkung	 y-Messbereich: 12,5 mV 400-fache Verstärkung
Sinus 1Hz, V_{pp} = 2mV	 y-Messbereich: 5 mV 1000-fache Verstärkung	 y-Messbereich: 2,5 V 2000-fache Verstärkung	

Tabelle 21: Sinussignal bei unterschiedlichen Verstärkungen

4.2.2 Genauigkeit der Verstärkung

Die Genauigkeit der Verstärkung wird hier nur oberflächlich betrachtet. Wesentliche Faktoren, wie der Einfluss von Störsignalen und unterschiedlichen Temperaturen auf die Verstärkerschaltung, werden nicht untersucht.

4.2.2.1 Instrumentenverstärker

An den Eingang des Instrumentenverstärkers wird eine Spannung von möglichst genau einem Volt angelegt. Für die unterschiedlichen Verstärkerschaltungen wird dann die Ausgangsspannung gemessen. Die Spannungsmessungen wurden mit dem Messgerät Fluke 45 durchgeführt. Die Verstärkerschaltung ist in Bild 12 auf Seite 18 dargestellt. Die gemessene Eingangsspannung des Instrumentenverstärkers beträgt 1,0005 V.

Verstärkung	Ausgangsspannung		Abweichung	
	Soll	Ist	in Volt	in %
1-fach	1,0005 V	1,0052 V	0,0047 V	0,470 %
2-fach	2,0010 V	2,0111 V	0,0101 V	0,505 %
4-fach	4,0020 V	4,013 V	0,011 V	0,275 %

Tabelle 22: Genauigkeit der Instrumentenverstärkerschaltung

In der Betrachtung des maximal durch die Widerstände der Verstärkerschaltung verursachten Fehlers in Kapitel 3.1.5 ist der maximale Fehler um mehr als das 10-fache geringer. Der Fehler bei der einfachen Verstärkung ist unabhängig von einer äußeren Beschaltung, so dass dieser von dem Instrumentenverstärker selbst verursacht werden muss.

4.2.2.2 Dekadenverstärkungsstufe

Mit dem Offsetabgleich können die Dekadenverstärkerstufen kalibriert werden. Für die Messungen in Tabelle 23 wurde die Verstärkerschaltung zunächst auf 1 V und danach auf etwa 100 mV kalibriert. In der ersten Messreihe fällt auf, dass sich der Verstärkungsfaktor bei Verringerung der Eingangsspannung ebenfalls verringert. Ein starker Abfall ist für eine

Spannung mit etwa 10 mV festzustellen, bei der nur eine etwa 7-fache statt 10-fache Verstärkung erreicht wird. Nimmt man eine Kalibrierung auf etwa 100 mV vor, so erreicht man für den gesamten Bereich der Eingangsspannung von 10 mV bis 1 V eine ungefähr 10-fache Verstärkung. Die maximale Abweichung liegt dann bei 10 mV bei 1,68 % und bei den anderen Messungen um 0,5 %. Eine grundsätzliche Tendenz der Veränderung der Verstärkung bezüglich der Eingangsspannung ist in der zweiten Messung nicht mehr zu erkennen.
Da Spannungen über 5 V vom A/D-Umsetzer des x/y-Schreibers nicht ausgewertet werden können, ist es sinnvoll, die Dekadenverstärker auf 0,1 V zu kalibrieren, um auch niedrige Spannungen vernünftig zu verstärken.

Kalibrierung auf 1,0000 V			**Kalibrierung auf 100,21 mV**		
U_{IN}	**U_{OUT}**	**Verstärkung**	**U_{IN}**	**U_{OUT}**	**Verstärkung**
1,0000 V	10,000 V	10,000	1,0007 V	10,045 V	10,038
0,5000 V	4,983 V	9,966	0,5011 V	5,030 V	10,038
100,20 mV	0,9696 V	9,677	100,21 mV	1,0021 V	10,000
50,19 mV	0,4675 V	9,315	50,18 mV	0,5052 V	10,068
10,73 mV	71,40 mV	6,654	10,70 mV	108,80 mV	10,168

Tabelle 23: Messung der Spannungsverstärkung eines Dekadenverstärkers

4.2.2.3 Gesamtverstärkung

Mit zwei verschiedenen Gleichspannungen U_{IN} wird die Genauigkeit der Verstärkung der gesamten Verstärkerschaltung untersucht. Nach jeder Verstärkerstufe wird die Ausgangsspannung gemessen.

U_{IN}	**Instrumenten-verstärker**	**Dekadenverstärkerstufen**		
		1	**2**	**3**
10,07 mV	9,63 mV	93,30 mV	0,9330 V	9,375 V
100,73 mV	98,22 mV	982,6 mV	9,822 V	-

Für die Gesamtverstärkung nach drei Stufen ergibt sich mit einer Verstärkung von 931 für das Eingangssignal von 10,07 mV eine Abweichung um 6,9%. Nach zwei Stufen ist es hier

eine etwa 92,7-fache Verstärkung und damit eine Abweichung um 7,3% von der idealen Verstärkung. Für das Eingangssignal ist die Abweichung mit 2,5% bei einer 97,5-fachen Verstärkung deutlich geringer. Beide Gleichspannungen sind in Bild 51 für eine Sekunde aufgetragen.

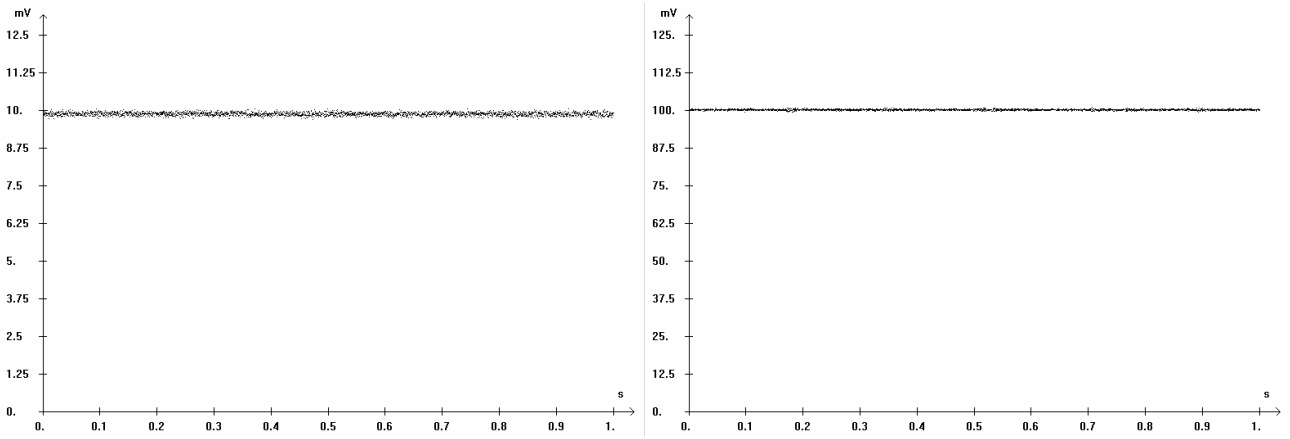


Bild 51: Messung von 10mV und 100mV Gleichspannung

Aus den zu Bild 51 gehörigen Messdaten kann die aufgenommene Spannung genau bestimmt werden. Der Maximalwert der Auflösung entspricht dem Messdatenwert 4096 (12 Bit).

Eingangsspannung	10,07 mV[83]	100,73 mV[84]
Mittelwert der Messdaten	3239	3281
Auflösung	12,5 mV	125 mV
gemessener Mittelwert	9,885 mV	100,128 mV
Messfehler	1,87%	0,60%

Damit ist die Messung mit dem x/y-Schreiber genauer als die vorherige Messung mit dem Messgerät.

[83] Messdaten auf CD: „Constant10.07mVInput.txt“.

[84] Messdaten auf CD: „Constant100.73mVInput.txt“.

4.2.2.4 Rauschen

Das Rauschen der verstärkten Sinussignale aus Kapitel 4.2.1 wird anhand von geeigneten Punkten untersucht. In Bild 52 sind Ausschnitte der Signale mit Punktinformationen für die größte Differenzspannung (Δ V) für einen x-Wert des verstärkten Eingangssignals angegeben. Der Prozentwert bezieht sich auf die Amplitude des Eingangssignals.

Durch eine möglichst gute Ausnutzung des gesamten Messbereichs, wie in der rechten Spalte, wird der Fehler, der durch das Rauschen entsteht, minimiert. Die Verstärkung durch den Instrumentenverstärker wirkt sich mit zunehmender Verstärkung positiv auf das Ergebnis aus. Da das Signal bis zum Instrumentenverstärker unabhängig von der Masse des x/y-Schreibers ist, ist zu vermuten, dass die Störeinstrahlung hauptsächlich durch die Masse bedingt ist. Es sollte daher untersucht werden, ob sich mit der Trennung der Masse der Verstärkerschaltung durch einen Filter von der Steuerschaltungsmasse, die Ergebnisse verbessern.

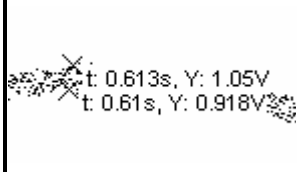
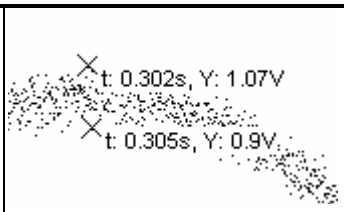
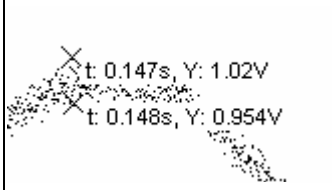
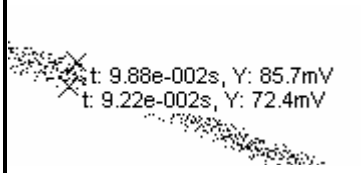
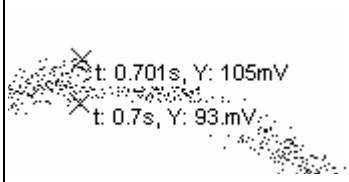
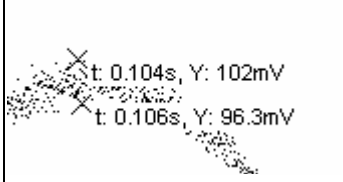
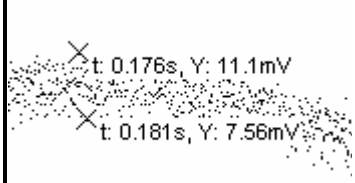
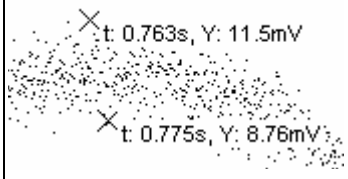
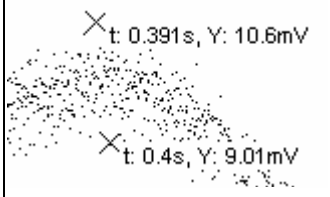
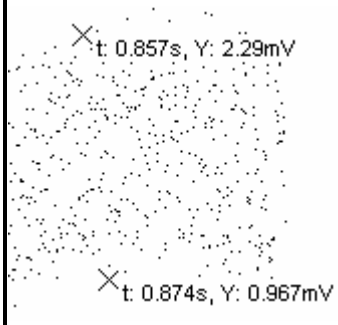
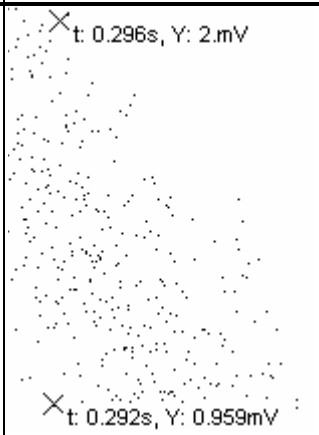
Sinus 1Hz, V_{pp} = 1V	 1-fache Verstärkung Δ V = 0,132V ≙ 13,2%	 2-fache Verstärkung Δ V = 0,170V ≙ 17,0%	 4-fache Verstärkung Δ V = 0,066V ≙ 6,6%
Sinus 1Hz, V_{pp} = 0,1V	 10-fache Verstärkung Δ V = 13,3mV ≙ 13,3%	 20-fache Verstärkung Δ V = 12,0mV ≙ 12,0%	 40-fache Verstärkung Δ V = 5,7mV ≙ 5,7%
Sinus 1Hz, V_{pp} = 10mV	 100-fache Verstärkung Δ V = 3,54mV ≙ 35,4%	 200-fache Verstärkung Δ V = 2,74V ≙ 27,4%	 400-fache Verstärkung Δ V = 1,59mV ≙15,9%
Sinus 1Hz, V_{pp} = 2mV	 1000-fache Verstärkung Δ V = 1,32mV ≙ 66,2%	 2000-fache Verstärkung Δ V = 1,04mV ≙ 52,1%	

Bild 52: Rauschfehler bei unterschiedlichen Verstärkungen

4.2.3 Offset

Die Offset-Funktion kann wegen nur eines funktionstüchtigen Kanals des Verstärkerprototyps nur im Zeitbetrieb gemessen werden. Die Messung des Offsets der x-Achse auf der Steuerungs-Platine und die Übertragung und Darstellung im PC-Programm funktionieren, führen jedoch ohne den zweiten Kanal mit der Offsetschaltung zu einem verzerrten Ergebnis.

Im Zeitmodus wird für die Laufzeit von einer Sekunde im Messbereich von 1,25 V ein Sinussignal mit einer Frequenz von einem Hertz und der Amplitude von einem Volt (-0,5 V bis +0,5 V) gemessen[85]. Der Offset des x/y-Schreibers wird auf 50%, entsprechend 2,5 V direkt am A/D-Umsetzer, eingestellt.

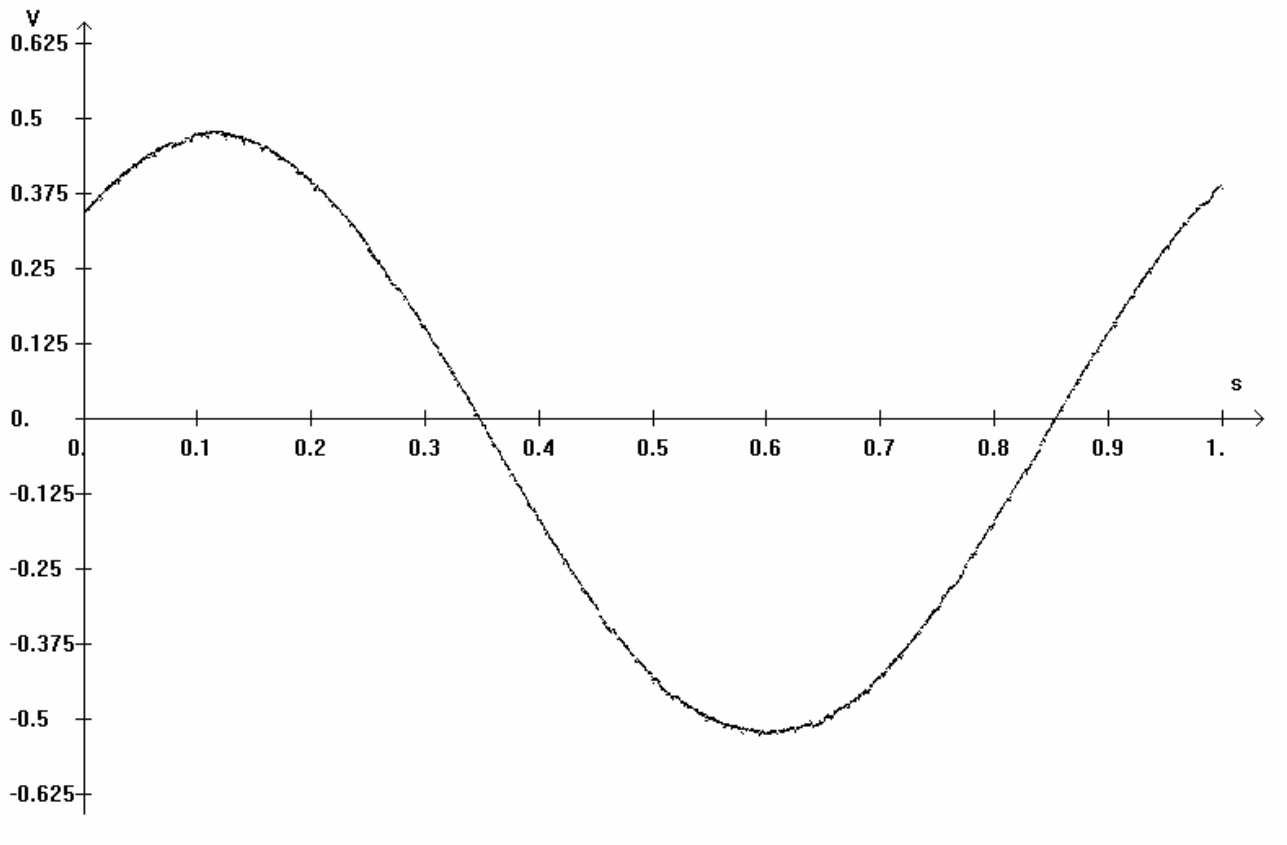


Bild 53: Messung negativer Eingangsspannungen durch einen Offset

Die Messung in Bild 53 entspricht dem Eingangssignal. Durch den Offset ist es demnach möglich, auch negative Spannungen zu messen, bzw. zu simulieren.

[85] Messdaten auf CD: „Sinus1Hz_1V_Offsset50.TXT“.

5 Fazit und Ausblick

Der entwickelte Mikroprozessor-gesteuerte x/y-Schreiber mit USB-Schnittstelle verfügt über alle Funktionen des als Vorbild genommenen analogen PM8041 von Philips. Zusätzliche Funktionen, wie der getriggerte Start der zeitlichen Aufnahme, ermöglichen auch kurze Messungen im Millisekundenbereich. Um die Messung möglichst hardwarenah zu gestalten, werden alle Einstellungen direkt am x/y-Schreiber vorgenommen und übersichtlich dargestellt. Die Anbindung an den PC erfolgt komfortabel und mit einer fast 1 MBit schnellen USB-Verbindung. Die Darstellung am PC richtet sich automatisch nach den bereits vorgenommenen Einstellungen. Die Messdaten können auf verschiedene Art verarbeitet werden. Sie können als Datenpaare in einer Datei abgelegt werden, aus der sie auch wieder geladen werden können, als Bitmap gespeichert oder ausgedruckt werden. Die Auswertung der Messergebnisse muss also nicht nur grafisch erfolgen, sondern kann direkt durch die einzelnen Punktdaten aus den Messdaten unterstützt werden.
Mit einer Aufnahmefrequenz von etwa 8 kHz im x/y-Betrieb und 2,7 kHz im Zeitbetrieb arbeitet der x/y-Schreiber recht schnell, so dass auch schnellere Sprünge des Eingangssignals oder im Zeitbetrieb Signale bis etwa 100 Hz aufgenommen werden können. Die Verstärkung des Eingangssignals ist so präzise, dass der Messfehler selbst im kleinsten Messbereich bei unter 2% liegt. Die Verläufe von Eingangssignalen einer Amplitude von nur einigen Millivolt sind trotz starken Rauschens gut zu erkennen.
Durch die Einstellung eines Offsets können auch negative Signale dargestellt werden. Sie werden in den messbaren Bereich des A/D-Umsetzers zwischen 0 und 5 V verschoben. Die Messdaten werden, wie sie aufgenommen werden, in einer Datei gespeichert.In der Darstellung wird der Offset jedoch berücksichtigt, so dass das eigentliche Eingangssignal mit negativen Anteilen dargestellt wird.

Mit der Entwicklung des Mikroprozessor-gesteuerten x/y-Schreibers mit USB-Schnittstelle ist es gelungen, einen vollständigen Ersatz für entsprechende rein analoge Geräte zu schaffen. Besonderer Wert wurde auf die leichte Bedienbarkeit und auf die komfortable Verwertung der Messergebnisse gelegt.

Für Weiterentwicklungen sollte das Hauptaugenmerk auf die Verbesserung der Verstärkung der Eingangssignale durch Trennung der Massen und weitere Maßnahmen zur

Rauschunterdrückung gelegt werden. Durch die im Layout vorgesehene SD-Speicherkarte ist ein Stand-Alone Betrieb möglich. Für den Einsatz im Lehrbetrieb wird eine noch effektivere Schutzschaltung gegen zu hohe Spannungsversorgungen empfohlen. Die bauteilbedingte Festigkeit der Spannungsregler reicht hierfür möglicherweise nicht aus. Als Orientierung kann die Schutzschaltung für die Messspannungen auf der Verstärkerplatine dienen.

Bilder

Tabellen

Listings

Literaturverzeichnis

[1] Gebrauchsanleitung A4 x-y Recorder, © N.V. Philips, Eindhoven, Niederlande 1977

[2] Serielle Datenübertragung
http://www.dynamo-software.de/serial/knowhow.htm, 26.06.2008

[3] Serial Peripheral interface (SPI)
http://de.wikipedia.org/wiki/Serial_Peripheral_Interface, 25.06.2008

[4] Universal Serial Bus Specification; Revision 2.0; April 27, 2000
http://www.usb.org/developers/docs/usb_20_040908.zip; 25.06.2008

[5] Digitale Schaltungen, Manfred Seifart, 4. bearbeitete Auflage, Verlag Technik GmbH Berlin 1990

[6] Operationsverstärkerschaltungen
http://de.wikipedia.org/wiki/Operationsverst%C3%A4rker; 28.04.2008

[7] Grundlagen zu Operationsverstärkern
http://www.elektronik-kompendium.de/sites/bau/0209092.htm, 28.04.2008

[8] OP177 Datenblatt, Rev. C, 25.06.2008
http://www.datasheetcatalog.org/datasheet/analogdevices/151066330OP177_c.pdf

[9] HCF4067 Datenblatt September 1988
http://www.datasheetcatalog.org/datasheet/SGSThomsonMicroelectronics/mXryquz.pdf, 25.06.2008

[10] MAX313 Datenblatt Rev2.0, 7.02
http://www.datasheetcatalog.org/datasheet/maxim/MAX312-MAX314.pdf, 25.06.2008

[11] INA114 Datenblatt, März 1998
http://www.datasheetcatalog.org/datasheet/BurrBrown/mXsqwzu.pdf, 25.06.2008

[12] EAGLE Handbuch, Version 4.1, Revision 2
ftp://ftp.cadsoft.de/eagle/program/4.16r2/manual-ger.pdf, 25.06.2008

[13] ATMEGA128 Datenblatt, Rev2467R-AVR-06.08
http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf, 25.06.2008

[14] C/C++ Referenz, Nootz/Morick, 5. aktualisierte Auflage, Franzis 2005

[15] AVR-GCC-Tutorial
http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial, 25.06.2008

[16] LCD162C Datenblatt, Version 1.1
http://www.displaytech.com.hk/pdf/char/162c%20series-v11.PDF, 25.06.2008

[17] FT232R Datenblatt, Version 1.04
http://www.ftdichip.com/Documents/DataSheets/DS_FT232R.pdf, 25.06.2008

[18] Entprellung http://www.mikrocontroller.net/articles/Entprellung, 24.04.2008

[19] MCP3202 Datenblatt, c1999
http://www.datasheetarchive.com/pdf/2301953.pdf, 25.06.2008

[20] D2XX Programmer´s Guide, c2006
http://www.ftdichip.com/Documents/ProgramGuides/D2XXPG34.pdf 28.06.2008

[21] MSDN Library for Visual Studio 2005, Recource für MFC-Programmierung
http://www.microsoft.com/downloads/details.aspx?FamilyID=b8704100-0127-4d88-9b5d-896b9b388313&DisplayLang=de, 25.06.2008

[22] AVR-Tutorial: LCD
http://www.mikrocontroller.net/articles/AVR-Tutorial:_LCD, 26.06.2005

[23] Bildschirminhalt als Bitmap speichern, 26.06.2008
http://www.online-tutorials.net/grafik/bitmap-speichern/sourcecodes-t-17-21.html

Anhang

A. EAGLE Schaltplan und Bordentwurf

Die für den x/y-Schreiber verwendeten Platinen wurden mit der Professional Edition des „**E**infach **A**nzuwendenden **G**rafischen **L**ayout **E**ditor“ „EAGLE“ Version 4.11 für Windows, erstellt. Das CAD-Programm besteht aus drei Hauptoberflächen, dem Schaltplanentwurf, der Bauteilbibliothek und dem Platinenentwurf. Der Schaltplan wird mit Bauteilen entworfen, deren Layout in der Bauteilbibliothek entweder bekannt ist, oder neu definiert werden muss. Wechselt man auf die Oberfläche für den Platinenentwurf, werden die im Schaltplan verwendeten Bauteile mit dem Entwurf entsprechenden Verknüpfungen zur Platzierung in einem vordefinierten Platinenbereich angeboten. Nach einer möglichst sinnvollen Platzierung der Bauteile, bei der sowohl die Verknüpfungen als auch der Schaltplan hilfreich sind, können die Verknüpfungen von Bauteil zu Bauteil durch Leiterbahnen ersetzt werden. Nach dem Setzen strategisch wichtiger Leiterbahnen, wie der Spannungsversorgung, hilft auch eine Autorouter-Funktion.

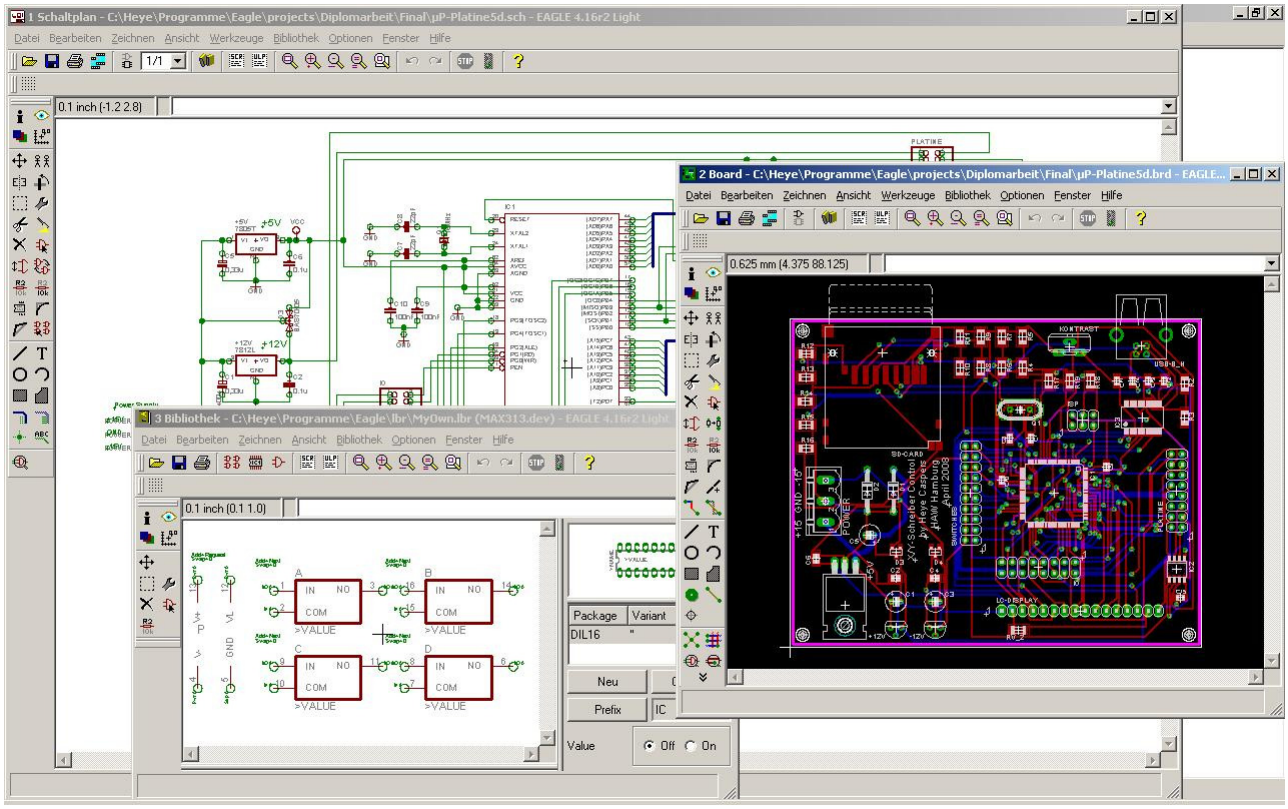
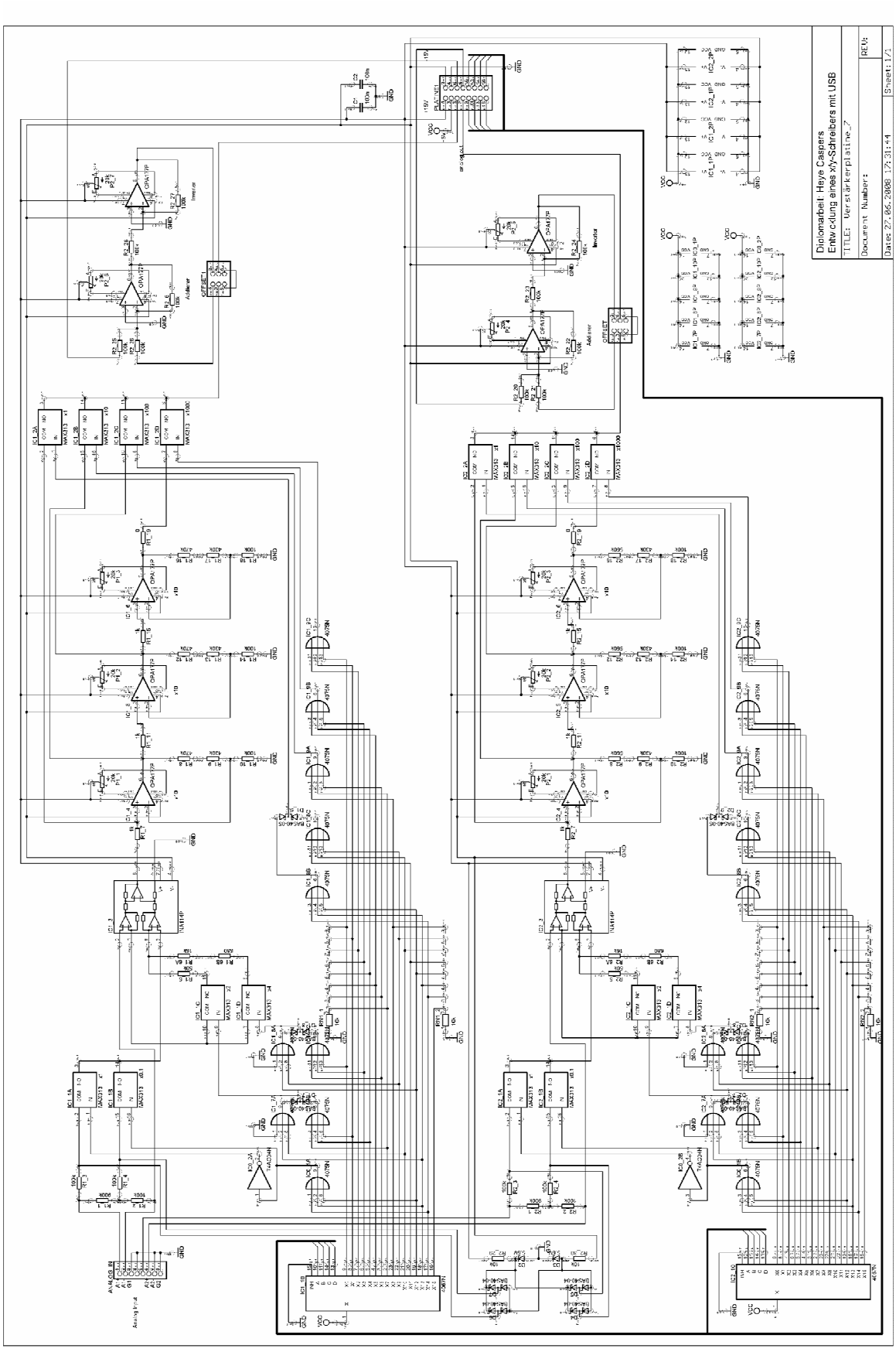
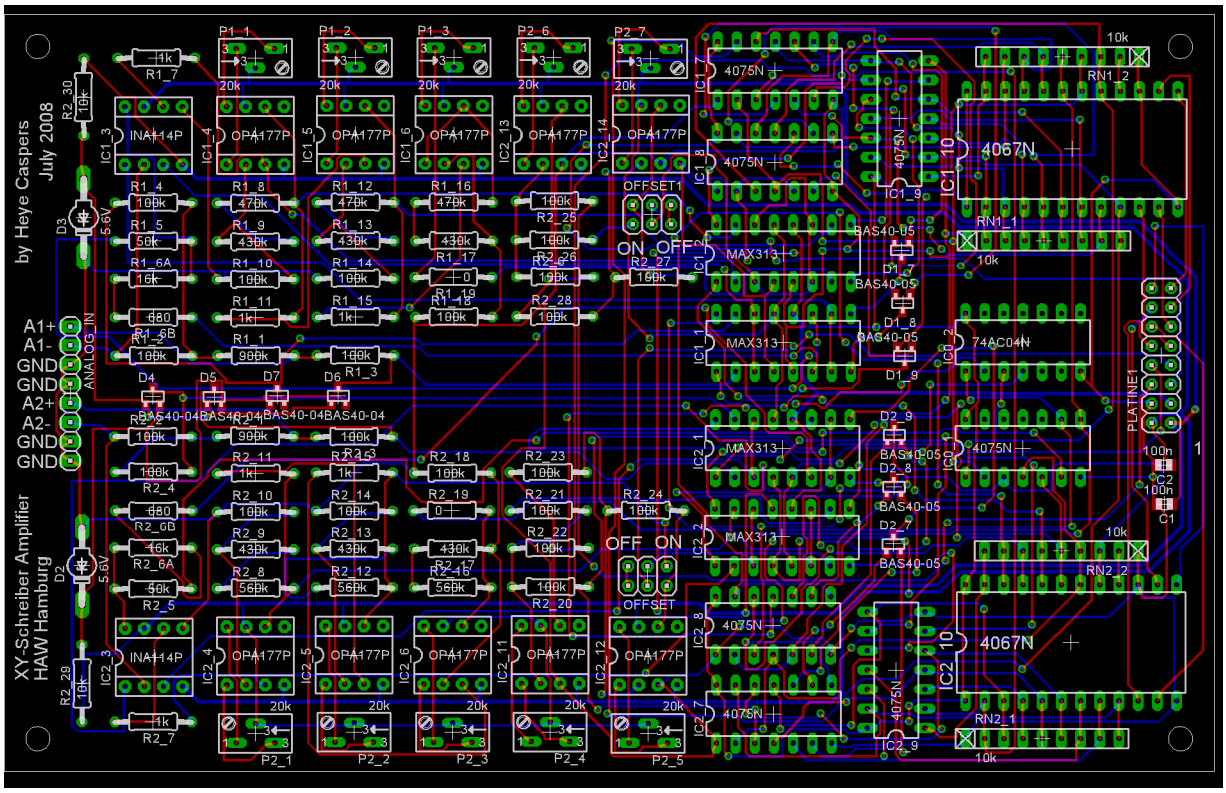


Bild 54: Screenshot des CAD-Programmes EAGLE

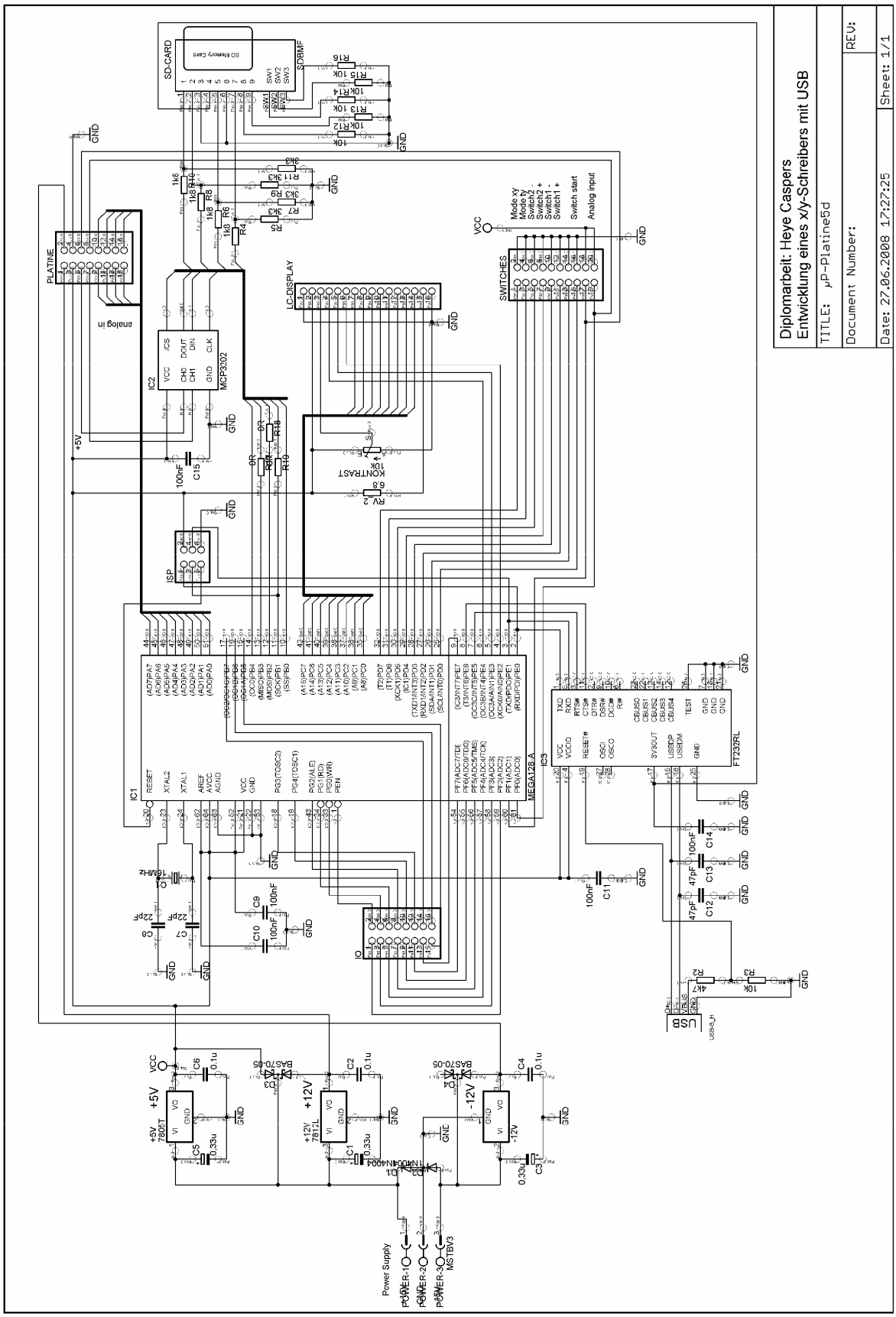
A.1. Schaltplan der Verstärkerplatine



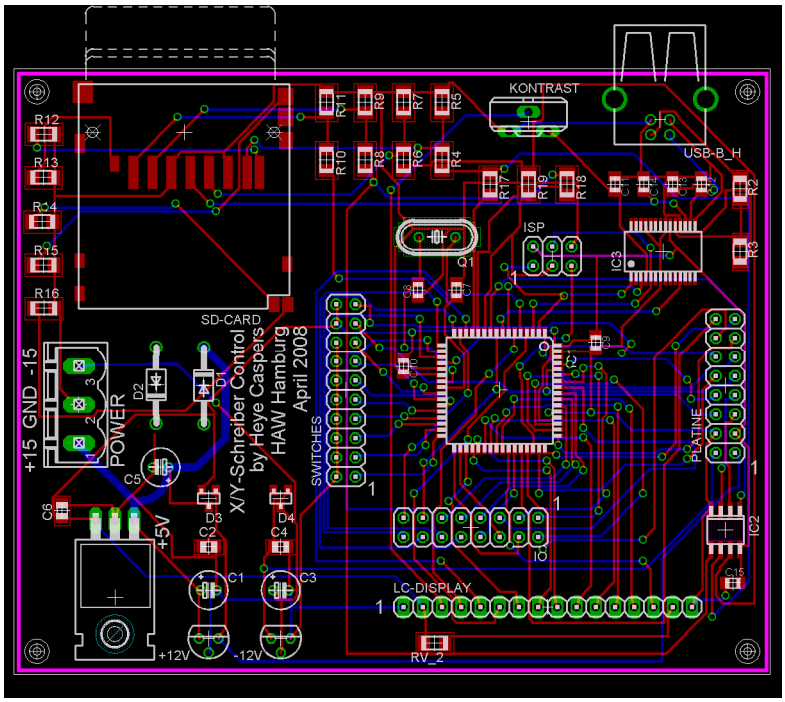
A.2. Layout der Verstärkerplatine



A.3. Schaltplan der Steuerplatine



A.4. Layout der Steuerplatine



B. Anschlüsse der Steuerplatine

Pin	Beschreibung	Pin	Beschreibung
1	+12 Volt	2	-12 Volt
3	+5 Volt	4	Masse
5	Analog in 0[86]	6	n.c.
7	Analog in 1[86]	8	Offset
9	DEMUX 1 Bit 0	10	DEMUX 2 Bit 0
11	DEMUX 1 Bit 1	12	DEMUX 2 Bit 1
13	DEMUX 1 Bit 2	14	DEMUX 2 Bit 2
15	DEMUX 1 Bit 3	16	DEMUX 2 Bit 3

Tabelle 24: Anschluss der Verstärkerplatine

Pin	Beschreibung	Pin	Beschreibung
1	Mode x/y	2	Masse
3	Mode t/y	4	Masse
5	Taster 2 -	6	Masse
7	Taster 2 +	8	Masse
9	Taster 1 -	10	Masse
11	Taster 1 +	12	Masse
13	n.c. (I/O)	14	Masse
15	Taster Start	16	Masse
17	n.c. (analog I/O)	18	+5 Volt
19	Offset	20	+5 Volt

Tabelle 25: Anschluss der Eingabetaster und -schalter

[86] Die Analogeingänge wurden nach Erstellung des Prototyps getauscht, siehe Schaltplan Anhang A.3

Pin	Beschreibung	Pin	Beschreibung	Pin	Beschreibung	Pin	Beschreibung
1	VSS	5	R/W	9	DB2	13	DB6
2	VDD	6	E	10	DB3	14	DB8
3	VO	7	DB0	11	DB4	15	BLA
4	RS	8	DB1	12	DB5	16	BLK

Tabelle 26: Anschluss des Displays[87]

Pin	Beschreibung	Pin	Beschreibung
1	Port E1 (TXD)		+5 Volt
3	SCK		Port E0 (RXD)
5	Reset		Masse

Tabelle 27: Programmierstecker des ATMEGA128

Pin	Beschreibung	Pin	Beschreibung
1	Port F2	2	#PEN
3	Port F3	4	#Port G0
5	Port F4	6	#Port G1
7	Port F5	8	Port G2
9	Port F6	10	Port G4
11	Port F7	12	Port G3
13	Port E7	14	Port B7
15	Port B5	16	Port B6

Tabelle 28: Ausgeführte ungenutzte I/Os des ATMEGA128[88]

[87] Siehe [16] LCD162C Datenblatt.

[88] Für die einzelnen Pinfunktionen siehe [13] ATMEGA128 Datenblatt.

C. Quellcode des Mikroprozessors

Der folgende Quellcode des AVR Mikroprozessors wurde mit dem WINAVR Programmers Notepad 2 Version v2.0.6.1-ella erstellt.

```
//*****************************************************************************
//
// File Name   :   'main.c'
// Title       :   Starting point of x/y-Writer
//
//*****************************************************************************

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include "Key.h"
#include "lcd.h"
#include "xy_ty.h"
#include "SPI_ADC_SD.h"
#include "RS232.h"

int main(void){

    key_init();                     // initialise keys
    lcdInit();                      // initialise LCD
    ADC_OS_init();                  // initialise offset
    SPI_init();                     // initialise SPI port
    USART_init();                   // initialise USART
    lcdClear();                     // clear LCD
    lcdHome();                      // LCD home position (0,0)
    DEMUX_DDR = 0xff;               // output for amplifier control
    DDRB |= (1 << 5);               // Port B, Pin 5 output
    DDRB &= ~(1 << 6);              // Port B, Pin6 input
    // improvised jumper to control LCD output in record phase
    PORTB &= ~(1 << 5);             // Set Pin 5 to ground
    PORTB |= ( 1 << 6);             // Set pull-ups to on

    sei();                          // enable interrupts

    for(;;) {                       // main loop not used normally
        xy_ty_adjust();             // start with adjustment phase
    }
}




//*****************************************************************************
//
// File Name   : 'xy_ty.h'
// Title       : Global variables and functions for adjustments and recording
//
//*****************************************************************************

#ifndef XY_TY_FUNC_H
#define XY_TY_FUNC_H
```

```
#define CLK 14745600                // MCU clock
#define XY_MAX 4096                 // maximum resolution of ADC
#define TY_MAX 16384                // 14 Bit time resolution

#ifndef MUX_PORT
    // port for amplifier demux lines
    #define DEMUX_PORT   PORTA
    #define DEMUX_DDR DDRA
#endif

// ****** global variables ******
uint8_t xRes;                       // x-Axis resolution
uint8_t yRes;                       // y-Axis resolution
uint8_t tRun;                       // Runtime in yt-mode
uint8_t trig;                       // trigger level
uint8_t mode;                       // recording mode (0=xy, 1=ty, 2=ty+trig)
uint8_t count;                      // countdown for timer cycles of 1 second
uint8_t count_max;                  // endvalue for count

// ****** Interrupts ******
ISR(TIMER1_COMPA_vect);             // Timer/Counter0 Compare Match

// ****** Functions ******
// for adjustment phase
void startup(void);                 // insert startup values
void offset_out(void);              // calculate and print offset
void tPrint(uint8_t);               // print on lcd
void xy_ty_adjust(void);            // main function for adjustments
void xyPrint(uint8_t);              // value for xy code
void tyRuntime(uint8_t);            // runtime in ty-mode
void tyTrigger(void);

//for recording phase
void xy_ty_record(void);            // main function for recording
void transmit_twoInt(uint16_t, uint16_t);  // transmits point or two int data
void init_timer(void);              // initialises the timer
uint8_t get_prescaler(void);        // get prescaler for timer register
uint16_t get_ocrVal(uint8_t);       // get value for timer register

#endif
```

```
//*****************************************************************************
//
// File Name   :   'xy_ty_adjust.c'
// Title       :   Adjustments made for recording, ended with push on "Start"
//
//*****************************************************************************

#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include "Key.h"
#include "lcd.h"
#include "xy_ty.h"
#include "RS232.h"
```

```
void xy_ty_adjust(void){            // main function for adjustment phase

    uint8_t change_flag = 0;        // 1 if new data received

    xRes = 11;                      // 5V (startup values)
    tRun = 21;                      // 1s
    yRes = 11;                      // 5V
    mode = 0;                       // xy mode
    trig = 50;                      // Trigger 2,5V (50%)
    DEMUX_PORT = ((yRes<<4)&0xf0) | (xRes&0x0f);  // amplifier control

    startup();                      // show startup values on display
    for(;;){
        // single press
        if ( get_key_press( 1<<KEY_START )){   // "Start"-button
            if( mode == 2)          // in mode 2 set trigger first
                tyTrigger();        // set trigger first then start recording
            ADCSRA &= ~(1<<ADEN);   // ADC disable
            xy_ty_record();         // stop adjustments and start recording
            ADCSRA |= (1<<ADEN);    // enable ADC for offset
        }

        // single press and repeat
        if( get_key_press( 1<<KEY_1P ) || get_key_rpt( 1<<KEY_1P )) {
            // (Channel 1++)
            if (mode==0){           // XY-mode
                if(xRes < 14) xRes++;// 15 different resolutions
                else xRes = 0;
                DEMUX_PORT = ((yRes<<4)&0xf0) | (xRes&0x0f);  // amplifier control
                lcdGotoXY( 0, 0);
                lcdPrintData( "X:" );// LCD output
                xyPrint(xRes);
            }
            else{                   // TY-mode
                if(tRun < 39) tRun++;// 40 different times
                else tRun = 0;      // 0: end with "Start"-button
                if(tRun%10==0 && tRun!=0) tRun++;   // to avoid e.g. 1000ms and 1s
                    lcdGotoXY( 0, 0);
                lcdPrintData( "T: " );

                tyRuntime( tRun );  // output and transmission of runtime
            }
            change_flag = 1;        // value changed
        }
        if( get_key_press( 1<<KEY_1M ) || get_key_rpt( 1<<KEY_1M )) {
            // (Channel 1--)
            if(mode == 0){          // XY-mode
                if(xRes > 0) xRes--;
                else xRes = 14;     // 15 different resolutions
                DEMUX_PORT = ((yRes<<4)&0xf0) | (xRes&0x0f);  // amplifier signal
                lcdGotoXY( 0, 0);
                lcdPrintData( "X:" );// LCD output
                xyPrint(xRes);
            }
            else{                   // TY-mode
                if(tRun > 0) tRun--; // 40 different times
                else tRun=39;
                if(tRun%10==0 && tRun!=0) tRun--;   // to avoid e.g. 1000ms and 1s
                lcdGotoXY( 0, 0);
                lcdPrintData( "T: " );
```

```
            tyRuntime( tRun );    // output and transmission of runtime
        }
        change_flag = 1;          // value changed
    }
    if( get_key_press( 1<<KEY_2P ) || get_key_rpt( 1<<KEY_2P )) {
        // (Channel 2++)
        if(yRes < 14) yRes++;
        else yRes = 0;
        if( mode == 0 )           // amplifier control
            DEMUX_PORT = ((yRes<<4)&0xf0) | (xRes&0x0f);
        else                      // same amplifier for trigger
            DEMUX_PORT = ((yRes<<4)&0xf0) | (yRes&0x0f);
        lcdGotoXY( 0, 1);
        lcdPrintData( "Y:" );     // LCD output
        xyPrint(yRes);
        change_flag = 1;          // value changed
    }
    if( get_key_press( 1<<KEY_2M ) || get_key_rpt( 1<<KEY_2M )) {
        // (Channel 2--)
        if(yRes > 0) yRes--;
        else yRes = 14;
        if( mode == 0 )           // amplifier signal
            DEMUX_PORT = ((yRes<<4)&0xf0) | (xRes&0x0f);
        else                      // same amplifier for trigger
            DEMUX_PORT = ((yRes<<4)&0xf0) | (yRes&0x0f);
        lcdGotoXY( 0, 1);
        lcdPrintData( "Y:" );     // LCD output
        xyPrint(yRes);
        change_flag = 1;          // value changed
    }

    // mode switch
    // XY-Mode
    if ( !(KEY_PIN & (1<<KEY_XY)) && mode!=0 ){
        mode = 0;
        DEMUX_PORT = ((yRes<<4)&0xf0) | (xRes&0x0f);
        lcdGotoXY( 9, 0);
        lcdPrintData( "XY-Mode" );
        lcdGotoXY( 0, 0);         // insert x value at mode switch
        lcdPrintData( "X:" );
        xyPrint(xRes);
        change_flag = 1;          // value changed
    }
    // TY-Mode (time)
    else if ( !(KEY_PIN & (1<<KEY_TY)) && mode!=1 ){
        mode = 1;
        DEMUX_PORT = ((yRes<<4)&0xf0) | (yRes&0x0f);
        lcdGotoXY( 9, 0);
        lcdPrintData( "TY-Mode" );
        lcdGotoXY( 0, 0);         // insert t value at mode switch
        lcdPrintData( "T: " );
        tyRuntime( tRun );
        change_flag = 1;          // value changed
    }
    // TY-Mode with trigger
    else if( (KEY_PIN & (1<<KEY_XY)) && (KEY_PIN & (1<<KEY_TY)) && mode!=2 ){
        mode = 2;
        DEMUX_PORT = ((yRes<<4)&0xf0) | (yRes&0x0f);
        lcdGotoXY( 9, 0);
        lcdPrintData( "TY-trig" );
```

```
            lcdGotoXY( 0, 0);          // insert t value at mode switch
            lcdPrintData( "T: " );
            tyRuntime( tRun );
            change_flag = 1;           // value changed
        }

        // offset calculation and output
        if( offset_flag == 1 ){
            offset_out();              // calculation and output
            change_flag = 1;           // value changed
        }

        // start transmission if value changed
        // byte 1: 'A', 2: mode, 3: xRes or tRun, 4: yRes, 5: y offset
        // upper, 6: y offset lower, 7: x offset upper, 8: x offset lower
        if( change_flag == 1 ){
            cli();                     // disable interrupts
            USART_transmit('A');       // code for Adjustments
            USART_transmit(mode);      // mode (0, 1, 2)
            if(mode == 0)              // depending on mode either
                USART_transmit(xRes);  // x resolution (0..15)
            else                       // or runtime (0..39)
                USART_transmit(tRun);
            USART_transmit(yRes);      // y resolution (0..15)
            // 10 bit y offset (0..1024)
            USART_transmit( (uint8_t) (yOffset>>8) );
            USART_transmit( (uint8_t) (yOffset&0x00ff) );
            // 10 bit x offset (0..1024)
            USART_transmit( (uint8_t) (xOffset>>8) );
            USART_transmit( (uint8_t) (xOffset&0x00ff) );
            change_flag = 0;           // reset flag
            sei();                     // enable interrupts
        }
    }
}

void startup(void){                    // print startup values to lcd
    if (mode==0){                      // XY-mode
        lcdGotoXY( 0, 0);
        lcdPrintData( "X:" );
        xyPrint(xRes);
    }
    else{                              // TY-mode
        if(tRun%10==0 && tRun!=0) tRun++;   // to avoid e.g. 1000ms and 1s
        lcdGotoXY( 0, 0);
        lcdPrintData( "T: " );
        tyRuntime( tRun );             // output and transmission of runtime
    }
    lcdGotoXY( 0, 1);                  // type y-Resolution
    lcdPrintData( "Y:" );
    xyPrint(yRes);
    mode = 3;                          // out of range to force first mode set
}

void offset_out(){
    uint8_t str_os[4];
    uint8_t ios;
    if((ADMUX&0x0f) == 0) {            // y offset
        lcdGotoXY( 13, 1);
        ios = (int) 100*((float) xOffset/1024 );      // Offset in %
```

```
    }
    else if((ADMUX&0x01) == 0x01) { // x offset
        lcdGotoXY( 9, 1);
        ios = (int) 100*((float) xOffset/1024 );      // Offset in %
    }
    itoa(ios, str_os, 10);
    if(ios < 10)
        lcdPrintData( " " );         // only one digit
    lcdPrintData( str_os );
    lcdPrintData( "%" );
    offset_flag = 0;                 // reset offset flag
}

void tPrint(uint8_t xy){             // print time data on lcd
    uint8_t str[3];
    itoa(xy, str, 10);
    lcdPrintData( str );
}

// print appropriate voltage on lcd and set ampilfier signal
void xyPrint(uint8_t xy){
    switch( xy ){
        case 0:    lcdPrintData( "1,25mV" ); break;    // in mV
        case 1:    lcdPrintData( " 2,5mV" ); break;
        case 2:    lcdPrintData( " 5,0mV" ); break;
        case 3:    lcdPrintData( "12,5mV" ); break;
        case 4:    lcdPrintData( "  25mV" ); break;
        case 5:    lcdPrintData( "  50mV" ); break;
        case 6:    lcdPrintData( " 125mV" ); break;
        case 7:    lcdPrintData( " 250mV" ); break;
        case 8:    lcdPrintData( " 0,50V" ); break;    // in V
        case 9:    lcdPrintData( " 1,25V" ); break;
        case 10:   lcdPrintData( " 2,50V" ); break;
        case 11:   lcdPrintData( "  5,0V" ); break;
        case 12:   lcdPrintData( " 12,5V" ); break;
        case 13:   lcdPrintData( " 25,0V" ); break;
        default:   lcdPrintData( " 50,0V" ); break;
    }
}

void tyRuntime(uint8_t t){           // runtime for ty-mode
    if(t==0){
        lcdPrintData( "120" );       // 2 minutes
        lcdPrintData( " s" );
    }
    else if(t < 10){                 // 10 to 90 ms
        lcdPrintData( " " );
        tPrint(t*10);
        lcdPrintData( "ms" );
    }
    else if(t < 20){                 // 100 to 900 ms
        tPrint( (t-10)*10 );         // last 0 comes with ms due char max value
        lcdPrintData( "0ms" );
    }
    else if(t < 30){                 // 1 to 9 seconds
        lcdPrintData( "  " );
        tPrint(t-20);
        lcdPrintData( " s" );
    }
    else{                            // 10 to 90 seconds
```

```
        lcdPrintData( " " );
        tPrint( (t-30)*10 );
        lcdPrintData( " s" );
    }
}

void tyTrigger(void){
    char str[2];
    lcdGotoXY( 9, 0);               // initial trigger value
    lcdPrintData( "TR: " );
    itoa(trig, str, 10);            // percentage output
    if(trig < 10)
        lcdPrintData( " " );
    lcdPrintData( str );
    lcdPrintData( "%" );
    for(;;){
        if( get_key_press( 1<<KEY_2P ) || get_key_rpt( 1<<KEY_2P )) {
            // (Trigger level ++)
            if(trig < 100) trig++;   // level 0..5V in %
            else trig = 0;
            lcdGotoXY( 9, 0);
            lcdPrintData( "TR: " );
            itoa(trig, str, 10);     // percentage output
            if(trig < 10)
                lcdPrintData( " " );
            lcdPrintData( str );
            lcdPrintData( "%" );
        }
        if( get_key_press( 1<<KEY_2M ) || get_key_rpt( 1<<KEY_2M )) {
            // (Trigger level --)
            if(trig > 0) trig--;     // level 0..5V in %
            else trig = 100;
            lcdGotoXY( 9, 0);
            lcdPrintData( "TR: " );
            itoa(trig, str, 10);     // percentage output
            if(trig < 10)
                lcdPrintData( " " );
            lcdPrintData( str );
            lcdPrintData( "%" );
        }
        // offset calculation and output
        if( offset_flag == 1 ){
            offset_out();
        }
        // second press starts recording
        if ( get_key_press( 1<<KEY_START )){
            return;
        }
    }
}
```

```
//*****************************************************************************
//
// File Name   :   'xy_ty_record.c'
// Title       :   Recording, ended with push on "Start" or end of runtime
//
//*****************************************************************************

#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include "Key.h"
#include "lcd.h"
#include "xy_ty.h"
#include "SPI_ADC_SD.h"
#include "RS232.h"

// Timer/Counter1 Compare Match
ISR(TIMER1_COMPA_vect){
    count++;                        // count >1 for times >1 second
}

void xy_ty_record(void){
    uint16_t value0, value1;
    double adc0, adc1;
    uint8_t str_c0[5];              // for 12 bit value
    uint8_t str_c1[5];              // for 12 bit value
    uint16_t ocrVal;                // timer register

    lcdClear();
    lcdGotoXY( 0, 0);
    lcdPrintData( "Recording..." ); // start recording

    transmit_twoInt(0, 0);          // to switch PC-Program to recording mode
    // wait for signal to start recording
    while ( !(PINE & (1 << 5)) && !get_key_press( 1<<KEY_START ));
    if( mode == 0){                 // xy-mode
        do{
        // record data
            cli();
            value0 = mcp3202_get_value(0);  // Channel 0 (x)
            value1 = mcp3202_get_value(1);  // Channel 1 (y)
            sei();
        // send data
            transmit_twoInt(value0, value1);
            if( !(PINB & (1 << 6)) ) {   // output if jumper is set (pin is low)
                adc0 = (double) value0 *5 /4096;    // 5 volts reference, 12 bit
                dtostrf(adc0, 4, 2, str_c0);        // 4 digits, 2 after ","
                lcdGotoXY( 0, 1);
                lcdPrintData ("X:");
                lcdPrintData (str_c0);
                adc1 = (double) value1 *5 /4096;    // 5 volts reference, 12 bit
                dtostrf(adc1, 4, 2, str_c1);        // 4 digits, 2 after ","
                lcdGotoXY( 8, 1);
                lcdPrintData ("Y:");
                lcdPrintData (str_c1);
            }

        } while( !get_key_press( 1<<KEY_START ));
    }
    else{                           // ty-mode
```

```
        ocrVal = get_ocrVal( get_prescaler());     // get counter maximum value
        if(mode == 2){        // trigger-wait for press start or channel x>trig
            if((value0 = mcp3202_get_value(0)) <=
                (uint16_t)trig*(4096/50)) {         // rising edge
                while(!get_key_press( 1<<KEY_START ) && (value0 <=
                    (uint16_t)trig*(4096/50))){
                    value0 = mcp3202_get_value(0);  // Channel 0 (x)
                    // output if jumper is set ( pin is low )
                    if( !(PINB & (1 << 6)) ) {
                        // calculate actual value and display it
                        adc0 = (double) value0 *5 /4096;
                        // 4 digits (with ","), 2 after ","
                        dtostrf(adc0, 4, 2, str_c0);
                        lcdGotoXY( 10, 1);
                        lcdPrintData ("Y:");
                        lcdPrintData (str_c0);
                    }
                }
            }
            else {     // falling edge
                while(!get_key_press( 1<<KEY_START ) && (value0 >
                    (uint16_t)trig*(4096/50))){
                    value0 = mcp3202_get_value(0); // Channel 0 (x)
                    // output if jumper is set ( pin is low )
                    if( !(PINB & (1 << 6)) ) {
                        // calculate actual value and display it
                        adc0 = (double) value0 *5 /4096;
                        // 4 digits (with ","), 2 after ","
                        dtostrf(adc0, 4, 2, str_c0);
                        lcdGotoXY( 10, 1);
                        lcdPrintData ("Y:");
                        lcdPrintData (str_c0);
                    }
                }
            }
        }
        init_timer();               // initialise and start timer
        do{
            cli();
            value1 = mcp3202_get_value(1);  // Channel 1 (y)
            value0 = TCNT1;          // 16 bit value from timer counter
            sei();
        // send data
            // fit value0 to TY_MAX = 16384 (maximum values of PC program
            // for time capture)
            // above one second 57600*count of "one second timer" has to
            // be added to current value
            value0 = (uint16_t)( (TY_MAX-1)*(float)(((1.0*value0/ocrVal) +
                count)/count_max) );
            transmit_twoInt(value0, value1);    // transmit point data
        // stop if time elapsed or start pressed
        } while( !get_key_press( 1<<KEY_START ) && count < count_max);
    }
    lcdClear ();
    lcdHome ();
    startup();        // return to adjustment phase with refreshed display
}

// transmit data points
void transmit_twoInt(uint16_t value0, uint16_t value1){
```

```
    cli();
    USART_transmit('R');     // R for recorded data pair and checksum at end
    USART_transmit( (uint8_t)(value0 >> 8) );     // transmit upper byte
    USART_transmit( (uint8_t)(value0 & 0x00ff) ); // transmit lower byte
    USART_transmit( (uint8_t)(value1 >> 8) );     // transmit upper byte
    USART_transmit( (uint8_t)(value1 & 0x00ff) ); // transmit lower byte
    // sixth byte is checksum: (val1a xor val1b) xor (val2a xor val2b)
    USART_transmit( (uint8_t)(((value0 >> 8) ^ (value0 & 0x00ff)) ^ ((value1 >>
        8) ^ (value1 & 0x00ff))) );
    sei();
}

// initialise the timer for output compare mode
void init_timer(){
    uint8_t prescaler;
    uint16_t ocrVal;

    prescaler = get_prescaler() ;
    // get value for register according to variable tRun
    ocrVal = get_ocrVal(prescaler);
    cli();                          // disable interrupts
    TCNT1 = 0x0000;          // reset timer - importand for repeated recording
    if (prescaler == 8){
        // devide by 8 and CTC (clear timer, timer start)
        TCCR1B |= (1<<CS11) | (1<<WGM12);
        TCCR1B &= ~( (1<<CS12) | (1<<CS10) );
    }
    else if(prescaler == 64){
        // devide by 64 and CTC (clear timer, timer start)
        TCCR1B |= (1<<CS11) | (1<<CS10) | (1<<WGM12);
        TCCR1B &= ~(1<<CS12);
    }
    else{                           // prescaler == 256
        // devide by 256 and CTC (clear timer, timer start)
        TCCR1B |= (1<<CS12) | (1<<WGM12);
        TCCR1B &= ~( (1<<CS11) | (1<<CS10) );
    }
    OCR1A = ocrVal;
    TIMSK |= (1<<OCIE1A);           // output compare match interrupt enable
    sei();                          // enable interrupts
}

// get the appropriate prescaler for runtime tRun
uint8_t get_prescaler(void){
    if(tRun!=0 && tRun<=3)          // prescaler for timer initialisation
        return 8;
    else if(tRun!=0 && tRun <= 10)
        return 64;
    else                            // 0 and 11+
        return 255;                 // +1 is to be added for correct prescaler
}

// get the register value for given runtime tRun and prescaler
uint16_t get_ocrVal(uint8_t prescaler){
    count = 0;
    if(tRun == 0){                  // for 120 seconds
        count_max = 120;
        return 57600;               // 1s = 57600
    }
    else if(tRun <= 9){             // 10 to 90ms
```

```
        count_max = 1;
        return (tRun * (0.01 * CLK / prescaler));
    }
    else if(tRun > 10 && tRun <= 19){   // 100 to 900 ms
        count_max = 1;
        return ((tRun-10) * (0.1*CLK / (prescaler+1)));   // prescaler 255+1
    }
    else if(tRun > 20 && tRun <= 29){   // 1 to 9 s
        count_max = tRun-20;         // times to go 1 second
        return 57600;
    }
    else if(tRun > 30 && tRun <= 39){   // 10 to 90 s
        count_max = (tRun-30) * 10; // times to go 1 second
        return 57600;
    }
    else
        return 0;  // never reached (hopefully)
}



//***************************************************************************
//
// File Name   :   'lcd.h'
// Title       :   Character LCD driver for HD44780/SED1278 displays
//             according to datasheet and mikrocontroller.net
//
//***************************************************************************

#ifndef LCD_H
#define LCD_H

// include project-dependent configurations
#include "lcdconf.h"

// if LCD_DELAY is not defined, this definition sequence
// attempts to find a suitable LCD_DELAY given the F_CPU
#ifndef LCD_DELAY
#if F_CPU >= 16000000
#define LCD_DELAY asm volatile ("nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n
    nop\n nop\n nop\n nop\n nop\n nop\n");
#else
#if F_CPU >= 12000000
#define LCD_DELAY asm volatile ("nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n
    nop\n nop\n");
#else
#if F_CPU >= 8000000
#define LCD_DELAY asm volatile ("nop\n nop\n nop\n nop\n nop\n nop\n nop\n");
#else
#if F_CPU >= 4000000
#define LCD_DELAY asm volatile ("nop\n nop\n nop\n nop\n nop\n");
#else
#define LCD_DELAY asm volatile ("nop\n nop\n nop\n");
#endif
#endif
#endif
#endif
#endif
// HD44780 LCD controller command set
// writing:
```

```
#define LCD_CLR              0      // DB0: clear display
#define LCD_HOME             1      // DB1: return to home position
#define LCD_ENTRY_MODE       2      // DB2: set entry mode
#define LCD_ENTRY_INC        1      // DB1: increment
#define LCD_ENTRY_SHIFT      0      // DB2: shift
#define LCD_ON_CTRL          3      // DB3: turn lcd/cursor on
#define LCD_ON_DISPLAY       2      // DB2: turn display on
#define LCD_ON_CURSOR        1      // DB1: turn cursor on
#define LCD_ON_BLINK         0      // DB0: blinking cursor
#define LCD_MOVE             4      // DB4: move cursor/display
#define LCD_MOVE_DISP        3      // DB3: move display (0-> move cursor)
#define LCD_MOVE_RIGHT       2      // DB2: move right (0-> left)
#define LCD_FUNCTION         5      // DB5: function set
#define LCD_FUNCTION_8BIT    4      // DB4: set 8BIT mode (0->4BIT mode)
#define LCD_FUNCTION_2LINES  3      // DB3: two lines (0->one line)
#define LCD_FUNCTION_10DOTS  2      // DB2: 5x10 font (0->5x7 font)
#define LCD_CGRAM            6      // DB6: set CG RAM address
#define LCD_DDRAM            7      // DB7: set DD RAM address
// reading:
#define LCD_BUSY             7      // DB7: LCD is busy

// default setup loaded on LCD initialization
#ifdef LCD_DATA_4BIT
    #define LCD_FDEF_1       (0<<LCD_FUNCTION_8BIT)
#else
    #define LCD_FDEF_1       (1<<LCD_FUNCTION_8BIT)
#endif
#define LCD_FDEF_2           (1<<LCD_FUNCTION_2LINES)
#define LCD_FUNCTION_DEFAULT ((1<<LCD_FUNCTION) | LCD_FDEF_1 | LCD_FDEF_2)
#define LCD_MODE_DEFAULT     ((1<<LCD_ENTRY_MODE) | (1<<LCD_ENTRY_INC))

// progress bar defines
#define PROGRESSPIXELS_PER_CHAR 6

// initializes I/O pins connected to LCD
void lcdInitHW(void);
// waits until LCD is not busy
void lcdBusyWait(void);
// writes a control command to the LCD
void lcdControlWrite(uint8_t data);
// read the control status from the LCD
uint8_t lcdControlRead(void);
// writes a data byte to the LCD screen at the current position
void lcdDataWrite(uint8_t data);
// reads the data byte on the LCD screen at the current position
uint8_t lcdDataRead(void);

// initializes the LCD display (gets it ready for use)
void lcdInit(void);
// moves the cursor/position to Home (upper left corner)
void lcdHome(void);
// clears the LCD display
void lcdClear(void);
// moves the cursor/position to the row,col requested
void lcdGotoXY(uint8_t row, uint8_t col);
void LCDClearLine(int iLineNO);
// prints a series of bytes/characters to the display
void lcdPrintData(char* data);

#endif
```

```
//*****************************************************************************
//
// File Name  :   'lcdconf.h'
// Title      :   Character LCD driver for HD44780/SED1278 displays
//            (usable in I/O mode)
//            according to datasheet and microcontroller.net
//
//*****************************************************************************

#ifndef LCDCONF_H
#define LCDCONF_H

// Enable interface to LCD
#define LCD_PORT_INTERFACE

// Parameters for LCD_PORT_INTERFACE:
#ifdef LCD_PORT_INTERFACE
    #ifndef LCD_CTRL_PORT
        // port and pins for control lines
        #define LCD_CTRL_PORT PORTE
        #define LCD_CTRL_DDR  DDRE
        #define LCD_CTRL_RS      4
        #define LCD_CTRL_RW      3
        #define LCD_CTRL_E       2
    #endif
    #ifndef LCD_DATA_PORT
        // port for data lines
        #define LCD_DATA_PORT PORTC
        #define LCD_DATA_PIN  PINC
        #define LCD_DATA_DDR  DDRC
    #endif
#endif

// LCD display geometry
#define LCD_LINES           2      // visible lines
#define LCD_LINE_LENGTH     16     // line length (in characters)
// cursor position to DDRAM mapping
#define LCD_LINE0_DDRAMADDR     0x00
#define LCD_LINE1_DDRAMADDR     0x40

#endif



//*****************************************************************************
//
// File Name  :   'lcd.c'
// Title      :   Character LCD driver for HD44780/SED1278 displays
//            (usable in I/O mode)
//            according to datasheet and mikrocontroller.net tutorial
//
//*****************************************************************************

#include <avr/io.h>
#include <string.h>
#include <util/delay.h>
#include "lcd.h"
```

```
/*************************************************************/
/********************** LOCAL FUNCTIONS **********************/
/*************************************************************/
void lcdInitHW(void)
{
    // initialize I/O ports
    // initialize LCD control lines
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_RS);
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_RW);
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_E);
    // initialize LCD control lines to output
    LCD_CTRL_DDR |= (1<<LCD_CTRL_RS);
    LCD_CTRL_DDR |= (1<<LCD_CTRL_RW);
    LCD_CTRL_DDR |= (1<<LCD_CTRL_E);
    // initialize LCD data port to input
    // initialize LCD data lines to pull-up
    LCD_DATA_DDR = 0x00;            // set data I/O lines to input (8bit)
    LCD_DATA_PORT = 0xFF;           // set pull-ups to on (8bit)
}

void lcdBusyWait(void)
{
    // wait until LCD busy bit goes to zero
    // do a read from control register
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_RS);    // set RS to "control"
    LCD_DATA_DDR = 0x00;            // set data I/O lines to input (8bit)
    LCD_DATA_PORT = 0xFF;           // set pull-ups to on (8bit)
    LCD_CTRL_PORT |= (1<<LCD_CTRL_RW);     // set R/W to "read"
    LCD_CTRL_PORT |= (1<<LCD_CTRL_E);      // set "E" line
    LCD_DELAY;                      // wait
    while(LCD_DATA_PIN & 1<<LCD_BUSY)
    {
        LCD_CTRL_PORT &= ~(1<<LCD_CTRL_E); // clear "E" line
        LCD_DELAY;                  // wait
        LCD_DELAY;                  // wait
        LCD_CTRL_PORT |= (1<<LCD_CTRL_E);  // set "E" line
        LCD_DELAY;                  // wait
        LCD_DELAY;                  // wait
    }
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_E);     // clear "E" line
    //  leave data lines in input mode
}

void lcdControlWrite(uint8_t data)
{
// write the control byte to the display controller
    lcdBusyWait();                  // wait until LCD not busy
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_RS);    // set RS to "control"
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_RW);    // set R/W to "write"
    // 8 bit write
    LCD_CTRL_PORT |= (1<<LCD_CTRL_E);      // set "E" line
    LCD_DATA_DDR = 0xFF;            // set data I/O lines to output (8bit)
    LCD_DATA_PORT = data;           // output data, 8bits
    LCD_DELAY;                      // wait
    LCD_DELAY;                      // wait
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_E);     // clear "E" line
    // leave data lines in input mode
    LCD_DATA_DDR = 0x00;            // set data I/O lines to input (8bit)
    LCD_DATA_PORT = 0xFF;           // set pull-ups to on (8bit)
}
```

```
uint8_t lcdControlRead(void)
{
// read the control byte from the display controller
    register uint8_t data;
    lcdBusyWait();                  // wait until LCD not busy
    LCD_DATA_DDR = 0x00;            // set data I/O lines to input (8bit)
    LCD_DATA_PORT = 0xFF;           // set pull-ups to on (8bit)
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_RS);     // set RS to "control"
    LCD_CTRL_PORT |= (1<<LCD_CTRL_RW);      // set R/W to "read"
    // 8 bit read
    LCD_CTRL_PORT |= (1<<LCD_CTRL_E);       // set "E" line
    LCD_DELAY;                      // wait
    LCD_DELAY;                      // wait
    data = LCD_DATA_PIN;            // input data, 8bits
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_E);      // clear "E" line
    // leave data lines in input mode
    return data;
}

void lcdDataWrite(uint8_t data)
{
// write a data byte to the display
    lcdBusyWait();                  // wait until LCD not busy
    LCD_CTRL_PORT |= (1<<LCD_CTRL_RS);      // set RS to "data"
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_RW);     // set R/W to "write"
    // 8 bit write
    LCD_CTRL_PORT |= (1<<LCD_CTRL_E);       // set "E" line
    LCD_DATA_DDR = 0xFF;            // set data I/O lines to output (8bit)
    LCD_DATA_PORT = data;           // output data, 8bits
    LCD_DELAY;                      // wait
    LCD_DELAY;                      // wait
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_E);      // clear "E" line
    // leave data lines in input mode
    LCD_DATA_DDR = 0x00;            // set data I/O lines to input (8bit)
    LCD_DATA_PORT = 0xFF;           // set pull-ups to on (8bit)
}

uint8_t lcdDataRead(void)
{
// read a data byte from the display
    register uint8_t data;
    lcdBusyWait();                  // wait until LCD not busy
    LCD_DATA_DDR = 0x00;            // set data I/O lines to input (8bit)
    LCD_DATA_PORT = 0xFF;           // set pull-ups to on (8bit)
    LCD_CTRL_PORT |= (1<<LCD_CTRL_RS);      // set RS to "data"
    LCD_CTRL_PORT |= (1<<LCD_CTRL_RW);      // set R/W to "read"
    // 8 bit read
    LCD_CTRL_PORT |= (1<<LCD_CTRL_E);       // set "E" line
    LCD_DELAY;                      // wait
    LCD_DELAY;                      // wait
    data = LCD_DATA_PIN;            // input data, 8bits
    LCD_CTRL_PORT &= ~(1<<LCD_CTRL_E);      // clear "E" line
    //  leave data lines in input mode
    return data;
}
```

```
/************************************************************/
/******************** PUBLIC FUNCTIONS **********************/
/************************************************************/
void lcdInit()
{
    lcdInitHW();                    // initialize hardware
    lcdControlWrite(LCD_FUNCTION_DEFAULT);     // LCD function set
    lcdControlWrite(1<<LCD_CLR);        // clear LCD
    _delay_ms(60);                      // wait 60ms
    lcdControlWrite(1<<LCD_ENTRY_MODE | 1<<LCD_ENTRY_INC);   // set entry mode
    lcdControlWrite(1<<LCD_ON_CTRL | 1<<LCD_ON_DISPLAY );    // set display to on
    lcdControlWrite(1<<LCD_HOME);       // move cursor to home
    lcdControlWrite(1<<LCD_DDRAM | 0x00);  // set data address to 0
}

void lcdHome(void)
{
    lcdControlWrite(1<<LCD_HOME);   // move cursor to home
}

void lcdClear(void)
{
    lcdControlWrite(1<<LCD_CLR);    // clear LCD
}

void lcdGotoXY(uint8_t x, uint8_t y)
{
    register uint8_t DDRAMAddr;
    // remap lines into proper order
    switch(y)
    {
    case 0: DDRAMAddr = LCD_LINE0_DDRAMADDR+x; break;
    case 1: DDRAMAddr = LCD_LINE1_DDRAMADDR+x; break;
    default: DDRAMAddr = LCD_LINE0_DDRAMADDR+x;
    }
    lcdControlWrite(1<<LCD_DDRAM | DDRAMAddr);    // set data address
}

void lcdPrintData(char* data)
{
    register uint8_t i;
    uint8_t nBytes=strlen(data);
    if (!data) return;   // check pointer
    for(i=0; i<nBytes; i++)
    {
        lcdDataWrite(data[i]);      // print data
    }
}

void lcdClearLine(int iLineNO)
{
    char cLineData[16],*p_Data;
    p_Data=&cLineData[0];
    p_Data="                ";
    lcdGotoXY(0,iLineNO);
    lcdPrintData(p_Data);
    lcdGotoXY(0,iLineNO);
}
```

```
//*****************************************************************************
//
// File Name   : 'Key.h'
// Title       : Debouncing of input buttons and switches
//
//*****************************************************************************

#ifndef KEY_H
#define KEY_H

#include "KeyConf.h"                // include project dependent configurations
// Key detection behaviour definitions, repeat function at this keys
#define REPEAT_MASK (1<<KEY_1P ^ 1<<KEY_1M ^ 1<<KEY_2P ^ 1<<KEY_2M)
#define REPEAT_START 50      // first repeat after 500ms
#define REPEAT_NEXT 20       // next repeats after 200ms

// global variables
uint8_t key_state;                  // debounced and inverted key state:
                                    // bit = 1: key pressed
uint8_t key_press;                  // key press detect
uint8_t key_rpt;                    // key long press and repeat
uint16_t yOffset;                   // Value of ADC conversion
uint16_t xOffset;                   // Value of ADC conversion
uint8_t offset_flag;

// ****** Interrupts ******
ISR(TIMER0_COMP_vect);              // Timer/Counter0 Compare Match
ISR(ADC_vect);                      // ADC conversion ready

// ****** Functions ******
// IO, timer, adc and interrupt initialization
void key_init(void);
// debounced key press detection
uint8_t get_key_press(uint8_t key_mask);
// repeat function for long press
uint8_t get_key_rpt(uint8_t key_mask);
// initialisation of ADC for offset
void ADC_OS_init (void);

#endif



//*****************************************************************************
//
// File Name   : 'KeyConf.h'
// Title       : Debouncing of input buttons and switches
//
//*****************************************************************************

#ifndef KEYCONF_H
#define KEYCONF_H
#define KEY_PORTS
// Parameters for KEY_PORTS:
#ifdef KEY_PORTS
    #ifndef KEY_PORT
        // port used for data lines
        #define KEY_PORT  PORTD
        #define KEY_PIN      PIND
        #define KEY_DDR      DDRD
```

```
        #define KEY_START     0
        #define KEY_1P        2
        #define KEY_1M        3
        #define KEY_2P        4
        #define KEY_2M        5
        #define KEY_XY        7
        #define KEY_TY        6
    #endif
#endif
#endif



//*****************************************************************************
//
// File Name   :   'Key.c'
// Title       :   Debouncing of input buttons and switches
//             4 time sampling with repeat function
//
//*****************************************************************************

#include <avr/io.h>
#include <avr/interrupt.h>
#include "Key.h"

// ****** Timer Interrupt handler ******
// Timer/Counter0 Compare Match
ISR(TIMER0_COMP_vect)                 // every 10ms
{
    static uint8_t ct0, ct1, rpt, ct_os;
    uint8_t i;
    i = key_state ^ ~KEY_PIN;         // key changed ?
    ct0 = ~(ct0 & i);                 // count bit 0 of ct (2Bit for 4 turns)
    ct1 = ct0 ^ (ct1 & i);            // count bit 1 of ct
    i &= ct0 & ct1;                   // count until overrun
    key_state ^= i;                   // after overrun toggle debounced state
    key_press |= key_state & i;       // 0->1: key press detected

    if( (key_state & REPEAT_MASK) == 0 )   // check repeat function
        rpt = REPEAT_START;           // start delay
    if( --rpt == 0 ) {
        rpt = REPEAT_NEXT;            // repeat delay
        key_rpt |= key_state & REPEAT_MASK;
    }
    if((ADCSRA & (0x00 | (1<<ADEN))) != 0) {   // ADC enabled ?
        // alternating ADC channels, each every 20ms
        if(ct_os == 0) {              // enable channel 0
            ADMUX &= ~(1<<MUX0);
            ct_os++;
        }
        else {                        // enable channel 1
            ADMUX |= (1<<MUX0);
            ct_os = 0;
        }
        ADCSRA |= (1<<ADSC);          // start alternating ADC every 10ms
    }
}
```

```
// ****** ADC Interrupt handler ******
ISR(ADC_vect)                          // interrupt: conversion ready
{
    if((ADMUX&0x0f) == 0) {
        yOffset =  ADCL & 0x00ff;    // Vin * 1024 / 5V
        yOffset |= (ADCH<<8) & 0xff00;
    }
    else if((ADMUX&0x01) == 0x01) {
        xOffset =  ADCL & 0x00ff;    // Vin * 1024 / 5V
        xOffset |= (ADCH<<8) & 0xff00;
    }
    offset_flag = 1;                   // flag: offset calculated
}

// ****** Functions ******
// IO, timer and interrupt initialization
void key_init(void)
{
    KEY_DDR = 0x00;                    // set data I/O lines to input (8bit)
    KEY_PORT = 0xFF;                   // set pull-ups to on (8bit)
    // devide by 1024 (timer start) and CTC
    TCCR0 |= (1<<CS02) | (1<<CS01) | (1<<CS00) | (1<<WGM01);
    OCR0 = (uint8_t) (F_CPU / 1024 * 10e-3 +0.5);     // OCR for 10ms
    TIMSK |= (1<<OCIE0);               // output compare match interrupt enable
}

// debounced key press detection
uint8_t get_key_press( uint8_t key_mask )
{
    cli();                             // read and clear atomic
    key_mask &= key_press;             // read key(s)
    key_press ^= key_mask;             // clear key(s)
    sei();
    return key_mask;
}

// repeat function for long press
uint8_t get_key_rpt( uint8_t key_mask )
{
    cli();                             // read and clear atomic
    key_mask &= key_rpt;               // read key(s)
    key_rpt ^= key_mask;               // clear key(s)
    sei();
    return key_mask;
}

// ****** ADC initialisation ******
void ADC_OS_init (void)
{
    // configure ADC - ADC enable, interrupt enable, F_CPU/128
    ADCSRA |= (1<<ADEN) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
    // external voltage reference
    ADMUX |= (1<<REFS0);
}
```

```
//*****************************************************************************
//
// File Name   : 'SPI_ADC_SD.h'
// Title       : SPI configuration for ADC MCP3202 and SD-Card
//
//*****************************************************************************

#ifndef SPI_ADC_SD_H
#define SPI_ADC_SD_H

// Port and Pin definitions
#define SPI_DDR       DDRB
#define SPI_PORT      PORTB
#define SD_CS         PB4           // Chip select for SD-Card
#define AD_MISO       PB3           // SPI MISO
#define AD_MOSI       PB2           // SPI MOSI
#define AD_SCK        PB1           // SPI clock
#define AD_SS         PB0           // Chip select for ADC MCP3202

/******* Functions ******/
void SPI_init (void);               // SPI port initialisation
extern uint16_t mcp3202_get_value (uint8_t);  // aquire value from adc mcp3202

// enter SD-Card functions here

#endif



//*****************************************************************************
//
// File Name   : 'SPI_ADC_SD.c'
// Title       : Initialisation and communication function for the mcp3202
//           analog to digital converter
//
//*****************************************************************************

#include <avr/io.h>
#include <stdint.h>
#include "SPI_ADC_SD.h"
#include <util/delay.h>
/****** SPI, ADC MCP3202 and SD-Card initialisation ******/
void SPI_init (void)
{
    // configure SPI - SPI enable, Master, F_CPU/16
    SPCR |= (1<<SPE)|(1<<MSTR)|(1<<SPR0);
    // SPSR |= (1<<SPI2X);          // add for F_CPU/8
    // configure DDRx - ADC-chip-select, MOSI, SPI-Clock, SD-cs: output
    SPI_DDR |= (1<<AD_SS) | (1<<AD_MOSI) | (1<<AD_SCK) | (1<<SD_CS);
    // deselect MCP3202 and SD-Card at start
    SPI_PORT |= (1<<AD_SS) | (1<<SD_CS);
}

/****** ADC MCP3202 functions ******/
uint16_t mcp3202_get_value (uint8_t channel)
{
    // value request with SPI communication using 8-bit-sequence
    uint16_t byte_rec = 0;
    SPI_PORT &= ~(1<<AD_SS);        // select device
    SPDR = 0x01;                // send start bit (8-bit sequece)
```

```
    while(!(SPSR & (1<<SPIF))); // wait, until transmission is completed
    // select mode and channel: single ended mode, MSB first
    SPDR = 0b10000000 | (channel << 6);
    while(!(SPSR & (1<<SPIF))); // wait, until transmission is completed
    byte_rec = (SPDR << 8);
    SPDR = 0x00;                // request lower bits
    while(!(SPSR & (1<<SPIF))); // wait, until transmission is completed
    byte_rec |= SPDR;           // 3 unknown + null bit + "12 bit value"
    SPI_PORT |= (1<<AD_SS);     // deselect device
    return (byte_rec & 0x0fff); // return "12 bit value"
}

/****** SD-Card functions ******/
// enter SD-Card functions here



//***************************************************************************
//
// File Name  : 'RS232.h'
// Title      : PC communication
//
//***************************************************************************

#ifndef RS232_H
#define RS232_H

//#define USART_BAUD_RATE 115200    // 9600 bit/s
//#define USART_BAUD_RATE 460800    // 460800 bit/s
#define USART_BAUD_RATE 921600      // 921600 bit/s
#define USART_UBRR_CALC(BAUD_,FREQ_)  (FREQ_/(BAUD_*16l)-1) // BR calculation

// ****** Global varianles ******
volatile uint8_t *transmit_data;
volatile uint8_t receive_data;

// ****** Initialisation ******
void USART_init(void);

// ****** Communication functions ******
uint8_t USART_receive (void);
void USART_transmit (uint8_t);
void USART_transmit_string (void);

#endif



//***************************************************************************
//
// File Name  :   'RS232.c'
// Title      :   PC communication
//
//***************************************************************************

#include <avr/io.h>
#include <avr/interrupt.h>
#include "RS232.h"
```

```
// ****** UART Interrupt handler ******
ISR (USART0_RX_vect) {              // UART receive complete
    receive_data = UDR0;            // read data from register
}

// ****** Initialisation ******
void USART_init() {
    // transmit and receive enable + interrupt receive
    UCSR0B = (1<<RXCIE0) | (1<<RXEN0)| (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00); // 8 bit data
    // Baud rate setting
    UBRR0H = (uint8_t) (USART_UBRR_CALC(USART_BAUD_RATE, F_CPU)>>8);
    UBRR0L = (uint8_t)  USART_UBRR_CALC(USART_BAUD_RATE, F_CPU);
}

// ****** Communication functions ******
void USART_transmit(uint8_t c) {
    while (!(UCSR0A & (1<<UDRE0))) {}   // register empty? (normally true)
    UDR0 = c;                           // transmit byte and step to next
}
```

D. Quellcode des PC-Programms

Der folgende Quellcode des PC-Progamms wurde mit dem Microsoft Visual Studio 2005 Version 8.0.50727.42 erstellt.

```
// ****************************************************************
//
//     File Name: :   stdafx.h
//     Title      :   standard header file
//
// ****************************************************************

#pragma once

#ifndef STDAFX_H
#define STDAFX_H

const unsigned int MAXPOINTS = 1048576;    // 2^20 points to be saved
const unsigned int MAXRES = 16384;         // 14 bit resolution for ty mode
const int XYDIV = 4;     // 16384/4 = 4096 for 12 bit AD resolution for xy mode
static unsigned int savePoint = 1;         // actual point position in Buffer

#ifndef VC_EXTRALEAN
// Selten verwendete Teile der Windows-Header nicht einbinden.
#define VC_EXTRALEAN
#endif
// Lassen Sie die Verwendung spezifischer Features von Windows XP oder später zu
#ifndef WINVER
// Ändern Sie dies in den geeigneten Wert für andere Versionen von Windows.
#define WINVER 0x0501
#endif
// Lassen Sie die Verwendung spezifischer Features von Windows XP oder später zu
#ifndef _WIN32_WINNT
// Ändern Sie dies in den geeigneten Wert für andere Versionen von Windows.
#define _WIN32_WINNT 0x0501
#endif
// Lassen Sie die Verwendung spezifischer Features von Windows 98 oder später zu
#ifndef _WIN32_WINDOWS
// Ändern Sie dies in den geeigneten Wert für Windows Me oder höher.
#define _WIN32_WINDOWS 0x0410
#endif
// Lassen Sie die Verwendung spezifischer Features von IE 6.0 oder später zu.
#ifndef _WIN32_IE
// Ändern Sie dies in den geeigneten Wert für andere Versionen von IE.
#define _WIN32_IE 0x0600
#endif
// Einige CString-Konstruktoren sind explizit.
#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS

#include <afxwin.h>                // MFC-Kern- und Standardkomponenten
#include <afxext.h>                // MFC-Erweiterungen

#ifndef _AFX_NO_OLE_SUPPORT
#include <afxole.h>                // MFC OLE-Klassen
#include <afxodlgs.h>              // MFC OLE-Dialogfeldklassen
#include <afxdisp.h>               // MFC-Automatisierungsklassen
#endif // _AFX_NO_OLE_SUPPORT
```

```
#ifndef _AFX_NO_DB_SUPPORT
#include <afxdb.h>                  // MFC-ODBC-Datenbankklassen
#endif // _AFX_NO_DB_SUPPORT

#ifndef _AFX_NO_DAO_SUPPORT
#include <afxdao.h>                 // MFC-DAO-Datenbankklassen
#endif // _AFX_NO_DAO_SUPPORT

#ifndef _AFX_NO_OLE_SUPPORT
// MFC-Unterstützung für allgemeine Steuerelemente von Internet Explorer 4
#include <afxdtctl.h>
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>   // MFC-Unterstützung für allgemeine Windows-Steuerelemente
#endif // _AFX_NO_AFXCMN_SUPPORT
#include <afxdlgs.h>
#include <ftd2xx.h>                 // FTDI RS232 to USB driver

#endif



// ********************************************************************
//
//     File Name: :   xyWriter.h
//     Title      :   Header file for xyWriter.cpp
//
// ********************************************************************

#pragma once
#ifndef XYWRITER_H
#define XYWRITER_H

#include "resource.h"
#include "OnePoint.h"
#include "OneGraph.h"
#include "DrawGraph.h"
#include "xyOpenFile.h"
#include "xySaveFile.h"
#include "SaveWindowAsBMP.h"
#include "xyPrintWindow.h"
#include "DrawGraphFromFile.h"
#include "AddGraphFromFile.h"
#include "SerialCom.h"

// ********************************************************************
//     Class CxyWriterWnd
// ********************************************************************
class CxyWriterWnd: public CFrameWnd   // windowclass
{
    int noOfDigits;
    bool onClickAddText;
    CPoint pointInfo[50];
    CPoint textPos[20];

// ***** public functions *****
public:
    CxyWriterWnd(void);
    ~CxyWriterWnd(void);
```

```
    void Display_Point(unsigned int);
    void Display_Points(CDC&);

// ***** private functions *****
private:
    CStatusBar xyStatusBar;
    afx_msg void OnPaint();
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    void FileNew();
    void FileOpen();
    void FileSave();
    void FileSaveAsBMP();
    void FilePrint();
    void FileClose();
    void EditInterpolate();
    void EditAddText();
    void ViewRedraw();
    void ViewScreenResolution();
    void HelpInfo();
    void ScreenResolution_640x480();
    void ScreenResolution_800x600();
    void ScreenResolution_1024x768();
    void ScreenResolution_1280x1024();
    void ScreenResolution_1400x1050();
    void ScreenResolution_1600x1200();
    void NumberOfDigits();
    void NumberOfDigits2();
    void NumberOfDigits3();
    void NumberOfDigits4();
    void NumberOfDigits5();
    void AddGraphFromFile();
    void InterpolateLinear();
    void UndoPointInfo();
    void UndoAddGraph();
    void UndoAddText();
    void Connect();
    void Disconnect();
    void StartRecording();
    afx_msg void OnMouseMove(UINT, CPoint);
    afx_msg void OnLButtonDown(UINT, CPoint);
    void DisplayStatusBar(CPoint*);
    void displayPointInfo(CPoint*, unsigned int);
    void redisplayPointInfo(void);
    CString createPointInfoString(CPoint*, unsigned int);
    void drawGraphFromFile(CDC&, int, TCHAR*);
    void redrawGraphFromFile(void);
    void addTextAtPos(int, CPoint*);
    void readTextAtPos(void);
    void delAddInfo(void);
    BOOL CALLBACK PaintHook(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM
    lParam);

    DECLARE_MESSAGE_MAP();
};


// ****************************************************************
//     Class CxyWriterApp
// ****************************************************************
class CxyWriterApp : public CWinApp
```

```
{
public:
    CxyWriterApp();                 // Constructor

public:
    virtual BOOL InitInstance();

    DECLARE_MESSAGE_MAP()
};
#endif



// ********************************************************************
//
//     File Name: :   xyWriter.cpp
//     Title      :   Entrance point for xy-Writer
//                    -> window construction and control
//
// ********************************************************************

#include "stdafx.h"
#include "xyWriter.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// ********************************************************************
//     Global Dialog Box Procedure (must not be in class)
//     http://zfxce.zfx.info/FAQ.php?ID=21 16.4.2008
// ********************************************************************

TCHAR text[20][80];
BOOL CALLBACK AddTextDlg(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam){
    int i = 0;
    TCHAR inText[80];               // text from dialog
    switch (uMsg)
    {
    case WM_COMMAND:
        switch (LOWORD(wParam))
        {
        case IDOK:
            do
            {
                if(*text[i] == NULL) // not assigned yet
                {                    // get text from dialog
                    if (!GetDlgItemText(hwndDlg, IDC_EDIT1, (LPWSTR)inText, 80))
                        *inText = 0;
                    else
                    {
                        for(int j = 0; j < 80; j++)
                            // copy to buffer for up to 20 strings
                            text[i][j] = inText[j];
                        i = 20;       // exit
                    }
                }
                i++;
            } while(i < 20);         // maximum 20!!
                // Fall through.
```

```
        case IDCANCEL:
            EndDialog(hwndDlg, wParam);
            return TRUE;
        }
    }
    return FALSE;
}

CDrawGraph drawGraph;           // to draw items into the window
CAddGraphFromFile agff;         // to add additional graph from file
CSerialCom sc;                  // to access serial connection

// ********************************************************************
//     Class CxyWriterWnd
// ********************************************************************

// Constructor
CxyWriterWnd::CxyWriterWnd(void)
{
    Create(NULL, L"XY-Writer", WS_OVERLAPPEDWINDOW, CRect(0, 0, 800, 600), NULL,
MAKEINTRESOURCE(IDR_MENU1));

    noOfDigits = 3;
    onClickAddText = false;         // flag for mouse click action:
                                    // true: position text, false: point info
    sc.portOpen = false;            // flag, true: connected to device
    for(int i = 0; i < 50; i++)     // up to 50 point infos can be stored
    {
        pointInfo[i].x = -1;
        pointInfo[i].y = -1;
    }
    for(int i = 0; i < 20; i++)     // up to 20 different texts
    {
        *text[i] = NULL;            // added text
        textPos[i].x = -1;          // according position
        textPos[i].y = -1;
    }
    for(int i = 0; i < 5; i++)      // buffer for filenames of added files
        agff.delFileName(i);
}

// Destructor
CxyWriterWnd::~CxyWriterWnd(void)
{
    sc.closePort();
}

BEGIN_MESSAGE_MAP(CxyWriterWnd, CFrameWnd)
    ON_WM_PAINT()
    ON_WM_CREATE()
    ON_COMMAND(ID_FILE_NEW40001, FileNew)
    ON_COMMAND(ID_FILE_OPEN40002, FileOpen)
    ON_COMMAND(ID_FILE_SAVE40003, FileSave)
    ON_COMMAND(IDM_FILE_SAVEAS, FileSave)
    ON_COMMAND(ID_FILE_PRINT40004, FilePrint)
    ON_COMMAND(ID_FILE_CLOSE40005, FileClose)
    ON_COMMAND(ID_EDIT_INTERPOLATE, EditInterpolate)
    ON_COMMAND(ID_VIEW_REDRAW, ViewRedraw)
    ON_COMMAND(ID_VIEW_SCREENRESOLUTION, ViewScreenResolution)
    ON_COMMAND(ID_SCREENRESOLUTION_640X480, ScreenResolution_640x480)
```

```
    ON_COMMAND(ID_SCREENRESOLUTION_800X600, ScreenResolution_800x600)
    ON_COMMAND(ID_SCREENRESOLUTION_1024X768, ScreenResolution_1024x768)
    ON_COMMAND(ID_SCREENRESOLUTION_1280X1024, ScreenResolution_1280x1024)
    ON_COMMAND(ID_SCREENRESOLUTION_1400X1050, ScreenResolution_1400x1050)
    ON_COMMAND(ID_SCREENRESOLUTION_1600X1200, ScreenResolution_1600x1200)
    ON_COMMAND(ID_HELP_INFO, HelpInfo)
    ON_COMMAND(ID_VIEW_NUMBEROFDIGITS, NumberOfDigits)
    ON_COMMAND(ID_NUMBEROFDIGITS_2DIGITS, NumberOfDigits2)
    ON_COMMAND(ID_NUMBEROFDIGITS_3DIGITS, NumberOfDigits3)
    ON_COMMAND(ID_NUMBEROFDIGITS_4DIGITS, NumberOfDigits4)
    ON_COMMAND(ID_NUMBEROFDIGITS_5DIGITS, NumberOfDigits5)
    ON_COMMAND(ID_VIEW_ADDGRAPHFROMFILE, AddGraphFromFile)
    ON_COMMAND(ID_INTERPOLATE_LINEAR, InterpolateLinear)
    ON_COMMAND(ID_UNDO_POINTINFORMATION, UndoPointInfo)
    ON_COMMAND(ID_UNDO_ADDGRAPH, UndoAddGraph)
    ON_COMMAND(ID_FILE_SAVEASBITMAP, FileSaveAsBMP)
    ON_COMMAND(ID_EDIT_ADDTEXT, EditAddText)
    ON_COMMAND(ID_UNDO_ADDTEXT, UndoAddText)
    ON_COMMAND(ID_RECORDING_NEWCONNECTION, Connect)
    ON_COMMAND(ID_RECORDING_DISCONNECT, Disconnect)
    ON_COMMAND(ID_RECORDING_RECORD, StartRecording)
    ON_WM_MOUSEMOVE()
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()

// draw measured points and graph
void CxyWriterWnd::OnPaint()
{
    CPaintDC dc(this);
    COneGraph og;
    // mode[0,1,2], xres[0..14], yres[0..14], offset[mV]
    drawGraph.drawGraph(dc, og.getMode(), og.getXres(), og.getYres(),
    og.getOffset(false), og.getOffset(true));
    Display_Points(dc);             // points
    redisplayPointInfo();           // point info
    redrawGraphFromFile();          // additional graphs
    readTextAtPos();                // text strings in window
}

// add status bar
int CxyWriterWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    UINT nIndicator = ID_SEPARATOR;
    if(CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    xyStatusBar.Create(this);       // creates status bar
    xyStatusBar.SetIndicators(&nIndicator, 1);
    return 0;
}

// Make new graph
void CxyWriterWnd::FileNew()
{
    COneGraph og;
    og.clear_all_points();          // clears all measured points
    delAddInfo();                   // delete stored additional information
    InvalidateRect(NULL, 1);        // Erases window
    OnPaint();                      // draw graph again
}
```

```
// open saved graph from a file
void CxyWriterWnd::FileOpen()
{
    CClientDC dc(this);
    CxyOpenFile of;
    if( of.xyOpenFile(dc) )          // if file was opened in dialog
    {
        delAddInfo();                // delete additional information
        InvalidateRect(NULL, 1);     // Erases window
        OnPaint();                   // Redraw window
    }
}

// save measured points to a file
void CxyWriterWnd::FileSave()
{
    CClientDC dc(this);
    CxySaveFile sf;
    sf.xySaveFile(dc);               // save dilog function
}

// Save screen as bitmap
void CxyWriterWnd::FileSaveAsBMP()
{
    CClientDC dc(this);
    CSaveWindowAsBMP swab;
    swab.saveWindowAsBMP(dc);        // call function to save as Bitmap
}

// print window
void CxyWriterWnd::FilePrint()
{
    CClientDC dc(this);
    CxyPrintWindow pw;
    pw.printWindow(dc);              // call function toprint window contents
}

// close program
void CxyWriterWnd::FileClose()
{
    PostMessage(WM_CLOSE, 0, 0);
}

// Get connection to device
void CxyWriterWnd::Connect()
{
    COneGraph og;
    CClientDC dc(this);
    FileNew();                   // Erase all window content and recorded data
    sc.record_flag = false;
    if(sc.portOpen = sc.openPort()) // Open port
    {
        savePoint = 1;               // reinitialise point position in buffer
        og.clear_all_points();
        sc.purgeFT232();             // erase registers
        if(sc.getData(dc))           // get actual settings from device
        {
            InvalidateRect(NULL, 1); // redraw graph
            OnPaint();
        }
```

```
    }
}

// Disconnect from device
void CxyWriterWnd::Disconnect()
{
    sc.closePort();
}

// Start recording
void CxyWriterWnd::StartRecording()
{
    CClientDC dc(this);
    FT_SetRts(sc.ftHandle);          // set handshake high (normally low)
    if(sc.portOpen)
    {
        sc.record_flag = false;      // adjustment phase
        do{
            if(sc.getData(dc))       // adjustments
            {   // redraw graph with latest values
                InvalidateRect(NULL, 1);
                OnPaint();
            }
        } while(!sc.record_flag);
        if(sc.portOpen)              // if Port is still open
            savePoint = 1;           // reinitialise point position in buffer
            sc.initRecord(dc);       // initialise and start recording
    }
}

// interpolate graph
void CxyWriterWnd::EditInterpolate()
{
}


// Add user defined text position assigned by mouse click
void CxyWriterWnd::EditAddText()
{
    // Open dialog box
    DialogBox(NULL, MAKEINTRESOURCE(IDD_OLE_PROPPAGE_SMALL), drawGraph.hWnd,
(DLGPROC)AddTextDlg);
    onClickAddText = true; // set flag for alternative use of left mouse click
}

// linear interpolation
void CxyWriterWnd::InterpolateLinear()
{
    CClientDC dc(this);
    COneGraph og;
    og.interpolateLinear(dc);
}

// Undo latest point information
void CxyWriterWnd::UndoPointInfo()
{
    int i = 49;
    do
    {
        if(pointInfo[i].x != -1)     // undo last information point
```

```
        {
            pointInfo[i].x = -1;
            i = 0;                      // exit after undo
        }
        i--;
    } while(i >= 0);
    InvalidateRect(NULL, 1);            // Erases window
    OnPaint();
}

// Undo latest Graph addition from file
void CxyWriterWnd::UndoAddGraph()
{
    int i = 4;                          // maximum 5 graphs to add
    do
    {
        if(*agff.getFileName(i) != NULL)
        {
            agff.delFileName(i);        // undo latest graph
            i = 0;
        }
        i--;
    } while(i >= 0);                    // maximum 5!!
    InvalidateRect(NULL, 1);            // Erases window
    OnPaint();
}

// Undo latest text addition
void CxyWriterWnd::UndoAddText()
{
    int i = 9;
    do
    {
        if(*text[i] != NULL)            // undo last information point
        {
            *text[i] = NULL;            // added text
            textPos[i].x = -1;          // according position
            textPos[i].y = -1;
            i = 0;                      // exit after undo
        }
        i--;
    } while(i >= 0);
    InvalidateRect(NULL, 1);            // Erases window
    OnPaint();
}

// redraw measured points and graph
void CxyWriterWnd::ViewRedraw()     // redraws graph with stored values
{
    delAddInfo();                       // delete stored additional information
    InvalidateRect(NULL, 1);            // Erases window
    OnPaint();
}

// change window size - see following functions for values/submenues below
void CxyWriterWnd::ViewScreenResolution()
{
}

// Set resolution for other window size
```

```
void CxyWriterWnd::ScreenResolution_640x480()
{
    drawGraph.screenResX = 640;
    drawGraph.screenResY = 480;
    MoveWindow(0, 0, 640, 480, 1);  // set new window size
}

void CxyWriterWnd::ScreenResolution_800x600()
{
    drawGraph.screenResX = 800;
    drawGraph.screenResY = 600;
    MoveWindow(0, 0, 800, 600, 1);
}

void CxyWriterWnd::ScreenResolution_1024x768()
{
    drawGraph.screenResX = 1024;
    drawGraph.screenResY = 768;
    MoveWindow(0, 0, 1024, 768, 1);

}

void CxyWriterWnd::ScreenResolution_1280x1024()
{
    drawGraph.screenResX = 1280;
    drawGraph.screenResY = 1024;
    MoveWindow(0, 0, 1280, 1024, 1);
}

void CxyWriterWnd::ScreenResolution_1400x1050()
{
    drawGraph.screenResX = 1400;
    drawGraph.screenResY = 1050;
    MoveWindow(0, 0, 1400, 1050, 1);
}

void CxyWriterWnd::ScreenResolution_1600x1200()
{
    drawGraph.screenResX = 1600;
    drawGraph.screenResY = 1200;
    MoveWindow(0, 0, 1600, 1200, 1);
}

// program information
void CxyWriterWnd::HelpInfo()
{
    MessageBox(L"xy-Writer by Heye Caspers, V1.0", L"Info",
        MB_ICONEXCLAMATION|MB_OK);
}

// Set number of digits for display in status bar or as point information
// For value change see following functions/submenus below
void CxyWriterWnd::NumberOfDigits()
{
}

// Set number of digits
void CxyWriterWnd::NumberOfDigits2()
{
    noOfDigits = 2;
```

```
}

void CxyWriterWnd::NumberOfDigits3()
{
    noOfDigits = 3;
}

void CxyWriterWnd::NumberOfDigits4()
{
    noOfDigits = 4;
}

void CxyWriterWnd::NumberOfDigits5()
{
    noOfDigits = 5;
}

// add additional Graph from file
void CxyWriterWnd::AddGraphFromFile()
{
    CClientDC dc(this);
    agff.addGraphFromFile(dc);      // call function to add graph from file
}

// get position for status bar
void CxyWriterWnd::OnMouseMove(UINT nFlags, CPoint point)
{
    DisplayStatusBar(&point);       // mouse position for status bar
    CClientDC dc(this);
    // adjustment phase with open port
    if(sc.record_flag == false && sc.portOpen == true)
    {
        sc.purgeFT232();
        if(sc.getData(dc))          // get actual settings
        {
            InvalidateRect(NULL, 1); // redraw graph
            OnPaint();
        }
    }
}

// on left mouseclick in graph set value lines
void CxyWriterWnd::OnLButtonDown(UINT nFlags, CPoint point)
{
    int i = 0, j = 0;
    CClientDC dc(this);
    // adjustment phase with open port
    if(sc.record_flag == false && sc.portOpen == true)
    {
        sc.purgeFT232();
        if(sc.getData(dc))          // get actual settings
        {
            InvalidateRect(NULL, 1); // redraw graph
            OnPaint();
        }
        if(sc.record_flag)          // if record flag was set by device input
            StartRecording();
    }
    if(sc.portOpen == false)        // editing only if unconnected
    {
```

```
        if(onClickAddText == false) // add point info at mouse x-position
        {
            COneGraph og;
            unsigned int maxPoint = MAXRES;
            if(og.getMode() == 0)
                maxPoint /= XYDIV;
            // 0..MAXPOINTS
            unsigned int x_pt = drawGraph.getXPoint(&point);
            do
            {
                if((pointInfo[i].x == -1) && (x_pt != -1) && (x_pt < maxPoint))
                {                       // store this point for recreation
                    pointInfo[i] = point;
                    i = 50;             // exit when set
                }
                i++;
            } while(i < 50);
            displayPointInfo(&point, x_pt); // display point information
        }
        else                            // add user defined text at mouse position
        {
            do
            {
                if(textPos[i].x == -1)   // store this point for recreation
                {
                    textPos[i] = point;
                    j = i;
                    i = 20;             // exit when set
                }
                i++;
            } while(i < 20);
            addTextAtPos(j, & point);
            onClickAddText = false;
        }
    }
}

// display one point
void CxyWriterWnd::Display_Point(unsigned int no)
{
    CClientDC dc(this);
    COneGraph og;
    og.display_point(dc, no);
}

// display all points
void CxyWriterWnd::Display_Points(CDC& dc)
{
    COneGraph og;
    og.display_points(dc);
}

// displays point at mouse position in status bar
void CxyWriterWnd::DisplayStatusBar(CPoint *pt)
{
    COneGraph og;
    CString str = (CString)"";
    char ch[16];
    unsigned int x_pt = drawGraph.getXPoint(pt);  // 0..MAXPOINTS
    str += createPointInfoString(pt, x_pt);
```

```
    str += ",  x-Offset: ";
    int os = og.getOffset(true);
    // convert value to char[], digits adjustable in menu
    _gcvt_s(ch, 16, (float)5.0*os/1024, noOfDigits);
    str += (CString) ch;
    str += " V,  ";
    str += "y-Offset: ";
    os = og.getOffset(false);
    // convert value to char[], digits adjustable in menu
    _gcvt_s(ch, 16, (float)5.0*os/1024, noOfDigits);
    str += (CString) ch;
    str += " V";
    if(sc.portOpen)
        str += " ,  connected to device";
    else
        str += " ,  not connected";
    if(onClickAddText)
        str += " ,  click to position text";
    xyStatusBar.SetWindowText(str);     // output string in status bar
}

// display point information at mouse click
void CxyWriterWnd::displayPointInfo(CPoint *mpt, unsigned int pt)
{
    CClientDC dc(this);
    COneGraph og;
    unsigned int maxPoint = MAXRES, x_pt, y_pt, i = 0;
    if(og.getMode() == 0)           // mode depending resolution
        maxPoint /= XYDIV;
    if(pt != -1)
    {
        x_pt = og.getXforPt(pt);
        if(x_pt != -1 && x_pt < maxPoint)   // point measured and !!MAXRES-1!!
        {
            y_pt = og.getYforPt(pt); // get appropriate y (0..MAXRES)
            drawGraph.drawPointInfo(dc, x_pt, y_pt, &createPointInfoString(mpt,
                pt));
        }
    }
}

// redisplay up to 50 onscreen point informations
void CxyWriterWnd::redisplayPointInfo()
{
    int i = 0;
    do
    {
        if(pointInfo[i].x != -1)
        {
            displayPointInfo(&pointInfo[i], drawGraph.getXPoint(&pointInfo[i]));

        }
        i++;
    } while(i < 50);
}

// create the point information string for infobar and onscreen information
CString CxyWriterWnd::createPointInfoString(CPoint* mpt, unsigned int pt)
{
    char ch[16];                    // needed to convert value to CString
```

```
    CString str = (CString) "";
    COneGraph og;
    unsigned int maxPoint = MAXRES, x_pt=-1, y_pt=-1;
    if(og.getMode() == 0)
        maxPoint /= XYDIV;
    if(pt != -1)
        x_pt = og.getXforPt(pt);     // get appropriate x (0..MAXRES)
    // mouse inside graph and !!MAXRES-1!!
    if(x_pt != -1 && x_pt < maxPoint && pt != -1)
    {
        y_pt = og.getYforPt(pt);     // get appropriate y (0..MAXRES)
        float x_val = drawGraph.getXValue(mpt, x_pt, og.getMode(),
            og.getOffset(true), og.getXres());  // x-value at axis
        float y_val = drawGraph.getYValue(mpt, y_pt, og.getOffset(false),
            og.getYres());           // y-value at axis
        if(og.getMode() == 0)        // xy-mode
            str += " X: ";
        else
            str += " t: ";
        // convert value to char[], digits adjustable in menu
        _gcvt_s(ch, 16, x_val, noOfDigits);
        str += (CString) ch;         // add to string
        if(og.getMode() != 0)        // ty-mode
        {
            if(og.getXres() < 20)
                str += "ms";
            else
                str += "s";
        }
        else                         // xy-mode
        {
            if(og.getXres() < 9)
                str += "mV";
            else
                str += "V";
        }
        if(y_pt != -1)               // get value only if it is measured yet
        {
            str += ", Y: ";
            _gcvt_s(ch, 16, y_val, noOfDigits);
            str += (CString) ch;
            if(og.getYres() < 9)
                str += "mV";
            else
                str += "V";
        }
        else
            str += ", Y: not measured";
    }
    return str;
}

// Redraw up to the last five graphs from files
void CxyWriterWnd::redrawGraphFromFile()
{
    CClientDC dc(this);
    CDrawGraphFromFile dgff;
    int i = 0;
    do
    {
```

```
        if(*agff.getFileName(i) != NULL)    // draw graph from file
            dgff.drawGraphFromFile(dc, i, agff.getFileName(i));
        i++;
    } while(i < 5);          // maximum 5!!
}

// display text at mouse position
void CxyWriterWnd::addTextAtPos(int i, CPoint *pt)
{
    CClientDC dc(this);
    if(*text[i] != NULL)
    {
        drawGraph.drawTextAtPos(dc, &textPos[i], (CString)text[i]);
    }
}

// redisplay text at stored mouse positions
void CxyWriterWnd::readTextAtPos()
{
    int i = 0;
    do
    {
        if(*text[i] != NULL)
            addTextAtPos(i, &textPos[i]);
        i++;
    } while(i < 20);                // maximum 20!!
}

// delete all information except actual graph
void CxyWriterWnd::delAddInfo()
{
    for(int i = 0; i < 5; i++)      // delete stored filenames of graphs
        agff.delFileName(i);
    for(int i = 0; i < 50; i++)  // delete stored points for point information
    {
        pointInfo[i].x = -1;
        pointInfo[i].y = -1;
    }
    for(int i = 0; i < 20; i++)     // delete stored text
    {
        *text[i] = NULL;            // added text
        textPos[i].x = -1;          // according position
        textPos[i].y = -1;
    }
}

// draw graph from chosen file
void CxyWriterWnd::drawGraphFromFile(CDC& dc, int count, TCHAR
fileName[_MAX_PATH])
{
    CDrawGraphFromFile dgff;
    dgff.drawGraphFromFile(dc, count, &fileName[_MAX_PATH]);
}


//*************************************************************************
//     Class CxyWriterApp
//*************************************************************************

BEGIN_MESSAGE_MAP(CxyWriterApp, CWinApp)
```

```
END_MESSAGE_MAP()

// CxyWriterApp-Erstellung
CxyWriterApp::CxyWriterApp()
{
}
// Das einzige CxyWriterApp-Objekt
CxyWriterApp xyApp;
BOOL CxyWriterApp::InitInstance()
{
    CWinApp::InitInstance();
    m_pMainWnd = new CxyWriterWnd;
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}



// ********************************************************************
//
//     File Name: : OnePoint.h
//     Title      : Classes Point and Graph to archive and manage
//                  measured points
//
// ********************************************************************

#pragma once
#ifndef ONEPOINT_H
#define ONEPOINT_H

#include "DrawGraph.h"
#include "OneGraph.h"

// ********************************************************************
//     Class Point
// ********************************************************************

class COnePoint
{
private:
    CPoint pt;
public:
    COnePoint(void);                // Constructor
    ~COnePoint(void);               // Destructor
    void set_point(unsigned int*, unsigned int*);     // new point
    void clear_point();             // Cleares this point
    unsigned int get_y();           // gives y value
    unsigned int get_x();           // gives x or t value
};

// ********************************************************************
//     Class Graph
// ********************************************************************

class CGraph
{
private:
    COnePoint *pOnePoint[MAXPOINTS];    // 0 to 5V, XYDIV * 4096 points
    int mode;
```

```
    int xres;
    int yres;
    int xOffset;
    int yOffset;
public:
    CGraph(void);                   // Constructor
    ~CGraph(void);                  // Destructor

    bool xy_new_point(CDC&, unsigned int*, unsigned int*); // for xy mode only
    bool ty_new_point(CDC&, unsigned int*, unsigned int*); //for xy mode only
    void clear_point(unsigned int); // Clear one point
    void clear_all_points();        // Clear all points
    bool display_point(CDC&, unsigned int);    // display one point
    void display_points(CDC&);      // display all points (refresh)
    void hide_point(CDC&, unsigned int, unsigned int);// hides a point
    unsigned int getXforPt(unsigned int);      // returns x value for position
    unsigned int getYforPt(unsigned int);      // returns y for given point
    void interpolateLinear(CDC&);              // interpolates unmeasured points
    int getMode(void);              // return and set mode
    void setMode(int);
    int getXres(void);              // return and set xres
    void setXres(int);
    int getYres(void);              // return and set yres
    void setYres(int);
    int getOffset(bool);            // return and set offset
    void setOffset(bool, int);
};
#endif



// ********************************************************************
//
//     File Name: : Point.cpp
//     Title      : Functions to archive measured points and
//                  to manage them
//
// ********************************************************************

#include "stdafx.h"
#include "OnePoint.h"

// ********************************************************************
// ****** Class OnePoint ********************************************
// ********************************************************************

// Constructor
COnePoint::COnePoint(void)
{
    pt.x = -1;                      // never reached
    pt.y = -1;
}

// Destructor
COnePoint::~COnePoint(void)
{
}

// save actual values as a new point
void COnePoint::set_point(unsigned int *x_new, unsigned int *y_new)
```

```
{
    pt.x = *x_new;                  // assign new values to point
    pt.y = *y_new;
}

// delete a point -> set it to (-1,-1)
void COnePoint::clear_point()
{
    pt.x = -1;            // highest possible value (never reached otherwise)
    pt.y = -1;
}

// returns x-value of a point
unsigned int COnePoint::get_x()
{
    return pt.x;
}

// returns y-value of a point
unsigned int COnePoint::get_y()
{
    return pt.y;
}

// ********************************************************************
// ********** Class Graph *********************************************
// ********************************************************************

// Constructor
CGraph::CGraph(void)
{
    for (unsigned int i = 0; i < MAXPOINTS; i++){
        // Number of points depends on resolution and mode
        pOnePoint[i] = new(COnePoint);
    }
    mode = 0;
    xres = 11;
    yres = 11;
    xOffset = 0;
    yOffset = 0;
}

// Destructor
CGraph::~CGraph(void)
{
    for (unsigned int i = 0; i < MAXPOINTS; i++)
        delete pOnePoint[i];
}

// assign a new point or update an already measured (xy-mode)
bool CGraph::xy_new_point(CDC& dc, unsigned int *x_new, unsigned int* y_new)
{
    if(savePoint < MAXPOINTS)
    {
        pOnePoint[savePoint]->set_point(x_new, y_new);    // new point
        display_point(dc, savePoint);   // new point displayed
        savePoint++;
        return true;
    }
    else
```

```
        return false;
}

// assign a new point (ty-mode)
bool CGraph::ty_new_point(CDC& dc, unsigned int* t_new, unsigned int* y_new)
{
    if(savePoint < MAXPOINTS)
    {
        pOnePoint[savePoint]->set_point(t_new, y_new);    // new point
        display_point(dc, savePoint);   // new point displayed
        savePoint++;
        return true;
    }
    else
    {
        savePoint = 1;
        return false;
    }
}

// clear a point (set it to (-1,-1))
void CGraph::clear_point(unsigned int no)
{
    pOnePoint[no]->clear_point();
}

// clears all points
void CGraph::clear_all_points()
{
    for(unsigned int i = MAXPOINTS-1; i > 0; i--){
        clear_point(i);
    }
    savePoint = 1;
}

// display one point
bool CGraph::display_point(CDC& dc, unsigned int no)
{
    CDrawGraph drawGraph;
    unsigned int x_val = pOnePoint[no]->get_x();
    unsigned int y_val = pOnePoint[no]->get_y();
    if(x_val != -1)                 // display only measured points
    {
        drawGraph.drawPoint(dc, x_val, y_val, RGB(0, 0, 0));
        return true;
    }
    else
        return true;      // last point reached
}

// display all points
void CGraph::display_points(CDC& dc)
{
    bool good = true;
    for( unsigned int i = 1; i < MAXPOINTS && good == true; i++)
    {
        good = display_point(dc, i);
    }
}
```

```
// hides a displayed point by drawing it white
void CGraph::hide_point(CDC &dc, unsigned int x, unsigned int y)
{
    CDrawGraph drawGraph;
    drawGraph.drawPoint(dc, x, y, RGB(255, 255, 255));    // draw white
}

// returns x for a given save point
unsigned int CGraph::getXforPt(unsigned int pt)
{
    return (unsigned int) pOnePoint[pt]->get_x();
}

// returns a y for a given save point
unsigned int CGraph::getYforPt(unsigned int pt)
{
    return (unsigned int) pOnePoint[pt]->get_y();
}

// linear interpolation of graph
void CGraph::interpolateLinear(CDC &dc)
{
    CDrawGraph drawGraph;
    unsigned int x_val1=0, x_val2=0, y_val1=0, y_val2=0, y_new=0;
    unsigned int j = 0;
    bool first = true;
    for(unsigned int i = 1; i < MAXPOINTS; i++)
    {
        if(((pOnePoint[i]->get_y()) != -1)) // measured point is found
        {
            if(first == true)        // for first point only
            {
                x_val1 = pOnePoint[i]->get_x(); //load point 1 values
                y_val1 = pOnePoint[i]->get_y();
                j = i + 1;           // +1: start at next point
                first = false;       // first point is found, deactivate flag
            }
            else
            {
                x_val2 = pOnePoint[i]->get_x();   // load point 2 values
                y_val2 = pOnePoint[i]->get_y();
                drawGraph.drawLine(dc, x_val1, y_val1, x_val2, y_val2,RGB(0,0,0));
                x_val1 = x_val2;     // point 2 gets point 1 for next round
                y_val1 = y_val2;
            }
        }
    }
}

//returns actual mode
int CGraph::getMode(void)
{
    return mode;
}

// sets mode
void CGraph::setMode(int m)
{
    mode = m;
}
```

```
//returns actual xres
int CGraph::getXres(void)
{
    return xres;
}

// sets xres
void CGraph::setXres(int x)
{
    xres = x;
}

//returns actual yres
int CGraph::getYres(void)
{
    return yres;
}

// sets yres
void CGraph::setYres(int y)
{
    yres = y;
}

//returns actual offset
int CGraph::getOffset(bool x)
{
    if( x )
        return xOffset;
    else
        return yOffset;
}

// sets offset
void CGraph::setOffset(bool x, int o)
{
    if( x )
        xOffset = o;
    else
        yOffset = o;
}



// ********************************************************************
//
//      File Name: : OneGraph.h
//      Title      : All functions to interact with the ONE graph
//
// ********************************************************************

#pragma once
#ifndef ONEGRAPH_H
#define ONEGRAPH_H

#include "OnePoint.h"
// ********************************************************************
//      Class COneGraph
// ********************************************************************
```

```
class COneGraph
{
public:
    COneGraph(void);
    ~COneGraph(void);

    bool xy_new_point(CDC&, unsigned int, unsigned int);
    bool ty_new_point(CDC&, unsigned int, unsigned int);
    unsigned int getXforPt(unsigned int);
    unsigned int getYforPt(unsigned int);
    void clear_all_points(void);
    void interpolateLinear(CDC&);
    void display_point(CDC&, unsigned int);
    void display_points(CDC&);
    int getMode(void);
    void setMode(int);
    int getXres(void);
    void setXres(int);
    int getYres(void);
    void setYres(int);
    int getOffset(bool);
    void setOffset(bool, int);
};
#endif



// ********************************************************************
//
//     File Name: :   OneGraph.cpp
//     Title      :   All functions for THE ONE graph
//
// ********************************************************************

#include "stdafx.h"
#include "OneGraph.h"

COneGraph::COneGraph(void)
{
}

COneGraph::~COneGraph(void)
{
}

CGraph graph;                    // to manage points

// Add new point to graph in xy mode
bool COneGraph::xy_new_point(CDC &dc, unsigned int x, unsigned int y)
{
    return graph.xy_new_point(dc, &x, &y);
}

// Add new point to graph in ty mode
bool COneGraph::ty_new_point(CDC &dc, unsigned int x, unsigned int y)
{
    return graph.ty_new_point(dc, &x, &y);
}
```

```
// returns the x value for a save point
unsigned int COneGraph::getXforPt(unsigned int Pt)
{
    return graph.getXforPt(Pt);
}

// returns the y value for a save point
unsigned int COneGraph::getYforPt(unsigned int Pt)
{
    return graph.getYforPt(Pt);
}

// clears all points of the graph
void COneGraph::clear_all_points()
{
    graph.clear_all_points();
}

// interpolates "black holes" between points
void COneGraph::interpolateLinear(CDC& dc)
{
    graph.interpolateLinear(dc);
}

// displays one specific point
void COneGraph::display_point(CDC &dc, unsigned int no)
{
    graph.display_point(dc, no);
}

// displays all points of the graph
void COneGraph::display_points(CDC &dc)
{
    graph.display_points(dc);
}

int COneGraph::getMode(void)
{
    return graph.getMode();
}

void COneGraph::setMode(int m)
{
    graph.setMode(m);
}

int COneGraph::getXres(void)
{
    return graph.getXres();
}

void COneGraph::setXres(int x)
{
    graph.setXres(x);
}

int COneGraph::getYres(void)
{
    return graph.getYres();
}
```

```
void COneGraph::setYres(int y)
{
    graph.setYres(y);
}

int COneGraph::getOffset(bool x)
{
    return graph.getOffset(x);
}

void COneGraph::setOffset(bool x, int o)
{
    graph.setOffset(x, o);
}



// ****************************************************************
//
//     File Name: : DrawGraph.h
//     Title      : Class CDrawGraph to draw items in wintow
//
// ****************************************************************

#pragma once
#ifndef DRAWGRAPH_H
#define DRAWGRAPH_H

#include "OneGraph.h"

// ****************************************************************
//     Class CDrawGraph
// ****************************************************************
class CDrawGraph
{
private:
    float xSteps;               // pixel between lines on x axis
    float ySteps;               // pixel between lines on y axis

public:
    int screenResX;             // resolution of total window
    int screenResY;             // can be set by user in menu or by drawing
    HWND hWnd;
    RECT clRect;                // dimensions of client window

// ***** public functions *****
    CDrawGraph(void);
    ~CDrawGraph(void);
    // draw coordinate system with x offset
    void drawGraph(CDC&, int, int, int, int, int);
    // draw point in coordinate system
    void drawPoint(CDC&, unsigned int, unsigned int, COLORREF);
    void drawLine(CDC&, unsigned int, unsigned int, unsigned int, unsigned int,
        COLORREF);                  // for interpolation
    // draw point information line (redraw to clear)
    void drawPointInfo(CDC&, unsigned int, unsigned int, CString*);
    void drawTextAtPos(CDC&, CPoint*, CString); // draw text to mouse position
    unsigned int getXPoint(CPoint*);       // returns x: 0..MAXRES
    // returns x-value: 0..MaxVal
```

```
    float getXValue(CPoint*, unsigned int, int, int, int);
    // returns y-value: 0..MaxVal
    float getYValue(CPoint*, unsigned int, int, int);

// ***** private functions *****
private:
    void drawXAxis(CDC&, int, int, int, int);     // with x offset
    void drawYAxis(CDC&, int, int, int, int);
    void drawValLines(CDC&, int, bool, int, int, int, int);
    float mVforRes(int);            // maximum voltage for resolution
    int secForRes(int);             // maximum time for resolution
};
#endif



// ********************************************************************
//
//     File Name: : DrawGraph.cpp
//     Title      : Class CDrawGraph to draw items in wintow
//
// ********************************************************************

#include "stdafx.h"
#include "DrawGraph.h"

// ********************************************************************
//     Class CDrawGraph
// ********************************************************************

// Constructor
CDrawGraph::CDrawGraph(void)
{
    screenResX = 800;               // set standard resolution
    screenResY = 600;
    hWnd = GetForegroundWindow();   // get handle for active window
    GetClientRect(hWnd, (LPRECT) &clRect);  // get dimensions of client window
}

// Destructor
CDrawGraph::~CDrawGraph(void)
{
}

// entrance function to draw coordinate system
void CDrawGraph::drawGraph(CDC& dc, int mode, int xRes, int yRes, int yOffset,
    int xOffset)                    // additional x offset
{
    hWnd = GetForegroundWindow();   // get handle for active window
    GetClientRect(hWnd, (LPRECT) &clRect);  // get dimensions of client window
    // draw x-axis (complete with values)
    drawXAxis(dc, mode, yOffset, xOffset, xRes);
    // draw y-axis (complete with values)
    drawYAxis(dc, mode, yOffset, xOffset, yRes);
}

// draw x-axis
void CDrawGraph::drawXAxis(CDC& dc, int mode, int yOffset, int xOffset, int
    xRes) // with x offset
{
```

```
    int yos = (int)((((clRect.bottom-50)-(clRect.top+15))*yOffset)/1024);
    dc.MoveTo(clRect.left+50, clRect.bottom-50-yos);
    dc.LineTo(clRect.right-15, clRect.bottom-50-yos); // draw x-Axis
    dc.LineTo(clRect.right-20, clRect.bottom-55-yos); // draw pointer
    dc.LineTo(clRect.right-15, clRect.bottom-50-yos);
    dc.LineTo(clRect.right-20, clRect.bottom-45-yos);
    if((xRes <= 8) && (mode == 0))  //xy mode
        // Axis values in mV
        dc.TextOutW(clRect.right-35, clRect.bottom-75-yos, (CString) "mV");
    else if((xRes > 8) && (mode == 0))
        // Axis values in V
        dc.TextOutW(clRect.right-35, clRect.bottom-75-yos, (CString) "V");
    else if((xRes != 0) && (xRes < 20) && (mode != 0))   // ty mode
        // Axis values in ms
        dc.TextOutW(clRect.right-35, clRect.bottom-80-yos, (CString) "ms");
    else   // Axis values in s
        dc.TextOutW(clRect.right-35, clRect.bottom-80-yos, (CString) "s");
    // 11 lines(including 0) all xStep pixel, 25 pixel off
    xSteps = (float)((clRect.right-50-15-25)/10.0);
    for(int i = 0; i < 11; i++)
    {
        drawValLines(dc, i, true, mode, yos, xOffset, xRes); // true=x axis
    }
}

// draw y-axis
void CDrawGraph::drawYAxis(CDC& dc, int mode, int yOffset, int xOffset, int
    yRes)                           // with x Offset
{
    if(mode != 0)                   // no offset in time mode for x-axis
        xOffset = 0;
    int xos = (int)((((clRect.right-40)-(clRect.left+50))*xOffset)/1024);
    dc.MoveTo(clRect.left+50+xos, clRect.bottom-50);
    dc.LineTo(clRect.left+50+xos, clRect.top+15); // draw y-Axis
    dc.LineTo(clRect.left+45+xos, clRect.top+20); // draw pointer at axis end
    dc.MoveTo(clRect.left+50+xos, clRect.top+15);
    dc.LineTo(clRect.left+55+xos, clRect.top+20);
    if(yRes <= 8)                   // Axis values in mV
        dc.TextOutW(clRect.left+15+xos, clRect.top+5, (CString) "mV");
    else                            // Axis values in V
        dc.TextOutW(clRect.left+15+xos, clRect.top+5, (CString) "V");
    // Axis in 10 pieces, let 25 pixel off
    ySteps = (float)((clRect.bottom-50-15-25)/10.0);
    for(int i = 0; i < 11; i++)     // 11 lines(including 0)
    {
        drawValLines(dc, i, false, 0, yOffset, xos, yRes);   // true=x axis
    }
}

// draw value lines for x- and y-axis
void CDrawGraph::drawValLines(CDC& dc, int lineNo, bool xaxis, int mode, int
yos, int xos, int res)
{   // res is either xRes or yRes, offset is 0 for x-Axis
    char val[16];
    if (xaxis)                      // x-Axis
    {
        if(mode == 0)               // xy mode
        {   // get value in mV or V for resolution code
            float valMax = mVforRes(res);
            _gcvt_s(val, 16, valMax*lineNo, 5);    // convert float to string
```

```
        }
        else                        // ty mode
        {   // get value in ms or s for resolution code
            float valMax = (float)(secForRes(res)/10.0);
            _gcvt_s(val, 16, valMax*lineNo, 5); // convert float to string
            xos = 0;                // no offset in time mode for x-axis
        }
        int xos_axis = (int)((((clRect.right-40)-(clRect.left+50))*xos)/1024);
        lineNo -= (int)(0.5+10*xos/1024);
        if((int)((clRect.left+50)+(lineNo*xSteps)+xos_axis) <= clRect.right-25)
        {
            dc.MoveTo((int)((clRect.left+50)+(lineNo*xSteps)+xos_axis),
                (clRect.bottom-50)-5-yos);      // 10 lines, each 10 pixel long
            dc.LineTo((int)((clRect.left+50)+(lineNo*xSteps)+xos_axis),
                (clRect.bottom-50)+5-yos);
            dc.TextOutW((int)(clRect.left+40+(lineNo*xSteps)+xos_axis),
                (clRect.bottom-50)+10-yos,(CString)val); // set value below x-Axis
        }
    }
    else                            // y-Axis
    {
        int yos_axis = (int)((((clRect.bottom-50)-(clRect.top+15))*yos)/1024);
        float valMax = mVforRes(res);       // axis value for res code
        lineNo -= (int)(0.5+10*yos/1024);
        _gcvt_s(val, 16, valMax*lineNo, 5);
        if((int)((clRect.bottom-50)-(lineNo*ySteps)-yos_axis) >= clRect.top+25)
        {
            dc.MoveTo(clRect.left+50-5+xos, (int)((clRect.bottom-50)-
                (lineNo*ySteps)-yos_axis)); // line every 50 pixel, 10 pixel long
            dc.LineTo(clRect.left+50+5+xos, (int)((clRect.bottom-50)-
                (lineNo*ySteps)-yos_axis));
            dc.TextOutW(clRect.left+5+xos, (int)((clRect.bottom-58)-
                (lineNo*ySteps)-yos_axis),(CString)val); //set value left of yAxis
        }
    }
}

// draw point into coordinate system
void CDrawGraph::drawPoint(CDC& dc, unsigned int x_val, unsigned int y_val,
    COLORREF col)
{
    COneGraph og;
    unsigned int maxPoint = MAXRES;
    if(og.getMode() == 0)           //xy-mode
        maxPoint /= XYDIV;
    // length of (x)-line is: right border -space right -space left - space
    // between maxvalue and pointer
    // fit into screen resolution
    x_val = (int)(x_val*(float)(0.5+(clRect.right-15-50-25))/maxPoint);
    y_val = (int)(y_val*(float)(0.5+(clRect.bottom-50-15-25))/(MAXRES/XYDIV));
    dc.SetPixel((int)(clRect.left+50 + x_val), (int)(clRect.bottom-50 - y_val),
        col);      // standard RGB(0,0,0));
}

void CDrawGraph::drawLine(CDC & dc, unsigned int x1, unsigned int y1, unsigned
    int x2, unsigned int y2, COLORREF col)
{
    COneGraph og;
    unsigned int maxPoint = MAXRES;
    if(og.getMode() == 0)           //xy-mode
```

```
        maxPoint /= XYDIV;
    // fit into screen resolution
    x1 = (int)(x1*(float)(0.5+(clRect.right-15-50-25))/maxPoint);
    y1 = (int)(y1*(float)(0.5+(clRect.bottom-50-15-25))/(MAXRES/XYDIV));
    // fit into screen resolution
    x2 = (int)(x2*(float)(0.5+(clRect.right-15-50-25))/maxPoint);
    y2 = (int)(y2*(float)(0.5+(clRect.bottom-50-15-25))/(MAXRES/XYDIV));
    dc.MoveTo(((clRect.left+50)+x1), (clRect.bottom-50)-y1);
    dc.LineTo(((clRect.left+50)+x2), (clRect.bottom-50)-y2);
}

// returns point with same x point (0..MAXRES) for mouse position
unsigned int CDrawGraph::getXPoint(CPoint *pt)
{
    COneGraph og;
    unsigned int maxPoint = MAXRES, x = -1, y = -1, bestX = -1, i = 1, bestI,
        pt_x, pt_y;
    if(og.getMode() == 0)        //xy-mode
        maxPoint /= XYDIV;
    pt_x = (unsigned int)(0.5+maxPoint*(float)(pt->x-50.0)/(clRect.right-15-
        50-25));
    pt_y = (unsigned int)(0.5+MAXRES/XYDIV*(float)(1 - (pt->y-15.0-25.0) /
        (clRect.bottom-15-50-25)));
    // if mouse is in graph ?
    if((clRect.left+50 <= pt->x) && (pt->x <= clRect.right-15-25) &&
        (clRect.top+15+25 <= pt->y) && (pt->y <= clRect.bottom-50))
    {
        // search for same x value in measured points
        do
        {
            x = og.getXforPt(i);
            y = og.getYforPt(i);
        // take best point but only in small window around mouse pointer
            if(abs(pt_x-x) <= abs(pt_x-bestX) && (x > pt_x-(MAXRES/XYDIV)/50)
                && (x < pt_x+(MAXRES/XYDIV)/50) && (y > pt_y-(MAXRES/XYDIV)/50)
                && (y < pt_y+(MAXRES/XYDIV)/50))
            {
                bestX = x;
                bestI = i;
            }
            i++;
        }
        while(i < MAXPOINTS && (x != -1 && y != -1) && (x != pt_x));
        if(bestX != -1)
            return bestI;
    }
    return -1;                        // if reached
}

// returns x value for point in coordinate system
float CDrawGraph::getXValue(CPoint *pt, unsigned int x_val, int mode, int
offset, int xres)
{

    if(mode == 0)                     // xy-mode
    {
        unsigned int maxPoint = MAXRES/XYDIV;
        float valMax = 10 * mVforRes(xres); // voltage for xres code
        return valMax*(float)(1.0*x_val/maxPoint) - valMax*offset/1024;
    }
```

```
    else                            // ty-mode
    {   // get value in ms or s for resolution code
        int valMax = secForRes(xres);
        return valMax*(float)(1.0*x_val/MAXRES) - valMax*offset/1024;
    }
}

// returns y value for point in coordinate system
float CDrawGraph::getYValue(CPoint *pt, unsigned int y_val, int offset, int
yres)
{
    COneGraph og;
    unsigned int maxPoint = MAXRES/XYDIV;
    // get value in mV or V for resolution code
    float valMax = mVforRes(yres)*10;
    return valMax*(float)(1.0*y_val/maxPoint) - valMax*offset/1024;
}

// draws point information
void CDrawGraph::drawPointInfo(CDC& dc, unsigned int x_val, unsigned int y_val,
CString *str)
{
    COneGraph og;
    unsigned int maxPoint = MAXRES;
    if(og.getMode() == 0)           //xy-mode
        maxPoint /= XYDIV;
    CFont font;
    font.CreatePointFont(80, L"Arial");
    CFont *pFont = dc.SelectObject(&font);
    dc.MoveTo((int)(clRect.left+50+(float)(1.0*x_val/maxPoint)*(clRect.right-
        50-15-25))-5, (int)(clRect.bottom-50-(float)(1.0*y_val/(MAXRES/XYDIV)) *
        (clRect.bottom-50-15-25))-5);
    dc.LineTo((int)(clRect.left+50+(float)(1.0*x_val/maxPoint)*(clRect.right-
        50-15-25))+5, (int)(clRect.bottom-50-  (float)(1.0*y_val/(MAXRES/XYDIV))*
        (clRect.bottom-50-15-25))+5);
    dc.MoveTo((int)(clRect.left+50+(float)(1.0*x_val/maxPoint)*(clRect.right-
        50-15-25))-5, (int)(clRect.bottom-50-  (float)(1.0*y_val/(MAXRES/XYDIV))*
        (clRect.bottom-50-15-25))+5);
    dc.LineTo((int)(clRect.left+50+(float)(1.0*x_val/maxPoint)*(clRect.right-
        50-15-25))+5, (int)(clRect.bottom-50-(float)(1.0*y_val/(MAXRES/XYDIV))*
        (clRect.bottom-50-15-25))-5);
    dc.TextOutW((int)(clRect.left+50+(float)(1.0*x_val/maxPoint)*(clRect.right
        -50-15-25))+5, (int)(clRect.bottom-50-(float)(1.0*y_val/(MAXRES/XYDIV))*
        (clRect.bottom-50-15-25)), *str);
}

// draw text to mouse position
void CDrawGraph::drawTextAtPos(CDC& dc, CPoint* pt, CString str)
{
    dc.TextOutW(pt->x, pt->y, str);
}

// returns the appropriate maximum value for the resolution code
float CDrawGraph::mVforRes(int res)
{
    switch(res)
    {
        case 0: return 0.125; break; // in 0.1 mV
        case 1: return 0.25; break;
        case 2: return 0.5; break;
```

```
        case 3: return 1.25; break;
        case 4: return 2.5; break;
        case 5: return 5.0; break;
        case 6: return 12.5; break;
        case 7: return 25.0; break;
        case 8: return 50.0; break;
        case 9: return 0.125; break;// in 0.1 V
        case 10: return 0.25; break;
        case 11: return 0.5; break;
        case 12: return 1.25; break;
        case 13: return 2.5; break;
        default: return 5.0; break;
    }
}

// returns the appropriate runtime for the time code
int CDrawGraph::secForRes(int res)
{   // to avoid values repeat, otherwise failure here thru calculation
    if(res%10 == 0 && res != 0) res++;
    if(res == 0) return 120;
    else if(res < 10) return 10*res;       // 10..90ms
    else if(res < 20) return 100*(res-10); // 100..900ms
    else if(res < 30) return 1*(res-20);   // 1..9s
    else return 10*(res-30);               // 10..90s
}



// ****************************************************************
//
//     File Name: : DrawGraphFromFile.h
//     Title      : Draws Graph from File to window
//
// ****************************************************************

#pragma once
#ifndef DRAWGRAPHFROMFILE_H
#define DRAWGRAPHFROMFILE_H

#include "DrawGraph.h"
#include "OneGraph.h"
#include <fstream>
#include <iostream>
using namespace std;

// ****************************************************************
//     Class CDrawGraphFromFile
// ****************************************************************

class CDrawGraphFromFile
{
    unsigned int x, y;       // point info
    COLORREF col;         // color of added graph
public:
    CDrawGraphFromFile(void);
    ~CDrawGraphFromFile(void);
    void drawGraphFromFile(CDC&, int, TCHAR*);
};
#endif
```

```
// ****************************************************************
//
//     File Name: :   DrawGraphFromFile.cpp
//     Title      :   Draws previously stored graph from file to window
//
// ****************************************************************

#include "stdafx.h"
#include "DrawGraphFromFile.h"

CDrawGraphFromFile::CDrawGraphFromFile(void)
{
    x = 0, y = 0;
}

CDrawGraphFromFile::~CDrawGraphFromFile(void)
{
}

void CDrawGraphFromFile::drawGraphFromFile(CDC &dc, int count, TCHAR
fileName[_MAX_PATH])
{
    CDrawGraph drawGraph;
    COneGraph og;
    int xRes, yRes;
    switch(count)                   // 5 colors for 5 graphs
    {
        case 0: col = RGB(255, 0, 0); break;       // red
        case 1: col = RGB(0, 128, 0); break;       // green
        case 2: col = RGB(0, 0, 128); break;       // navy
        case 3:col = RGB(128, 0, 128); break;      // purple
        case 4: col = RGB(255, 255, 0); break;     // yellow
        default: col = RGB(0, 0, 0);
    }
    ifstream inp(fileName);         // open from file
    // first line in file: mode, xres, yres, yoffset, xoffset
    inp >> x >> xRes >> yRes >> x >> x;
    // until last valid value, and only if resolutions fit
    while(!inp.eof() && y >= 0 && (xRes == og.getXres()) && (yRes ==
    og.getYres()))
    {
        inp >> x >> y;
        drawGraph.drawPoint(dc, x, y, col);    // draw it only no saving
    }
}



// ****************************************************************
//
//     File Name: : SerialCom.h
//     Title      : Open and manage serial communication
//
// ****************************************************************
#pragma once
#ifndef SERIALCOM_H
#define SERIALCOM_H

#include "OneGraph.h"
static int count = 0;
```

```
// ****************************************************************
//     Class SerialCom
// ****************************************************************

class CSerialCom
{
    FT_STATUS ftStatus;
    unsigned char RxBuffer[8];
    unsigned char lastRx[8];
    DWORD BaudRate;
    bool newData;

// ***** public variables *****
public:
    FT_HANDLE ftHandle;
    bool portOpen;
    bool record_flag;

// ***** public functions *****
public:
    CSerialCom(void);
    ~CSerialCom(void);
    bool openPort(void);
    void initRecord(CDC&);
    bool adjustRecord(void);
    bool getData(CDC&);
    bool closePort(void);
    bool purgeFT232(void);

// ***** private functions *****
private:
    bool transmit(unsigned char*);
    bool receive_byte(void);
    bool receive_dataset(void);
};
#endif



// ****************************************************************
//
//     File Name: : SerialCom.cpp
//     Title      : Functions to open and use a serial communication
//
// ****************************************************************

#include "stdafx.h"
#include "SerialCom.h"

// ****************************************************************
// ****** Class SerialCom *****************************************
// ****************************************************************

// Constructor
CSerialCom::CSerialCom(void)
{
    record_flag = false;            // record mode active
    //BaudRate = 115200;            // Baudrate for USB communication
    //BaudRate = 460800;            // Baudrate for USB communication
```

```
    BaudRate = 921600;              // Baudrate for USB communication
    for(int i = 0; i < 6; i++){
        lastRx[i] = 0;
    }
}

// Destructor
CSerialCom::~CSerialCom(void)
{
}

// initialise and start recording
void CSerialCom::initRecord(CDC& dc)
{
    record_flag = true;
    purgeFT232();
    while(receive_dataset());       // safety check of storage
    FT_ClrRts(ftHandle);            // Request to record data
    getData(dc);
    if(record_flag == false)        // in most cases true
    {
        closePort();                // close connection after recording
    }
}

// record sequence for xy and ty mode
bool CSerialCom::getData(CDC& dc)
{
    unsigned int data0, data1;
    int i = 0;
    COneGraph og;
    newData = false;

    do{
        if( !receive_dataset() )    // get dataset from USB
        {
            count++;
            if(count == 1000)
            {   // disconnect after 1000 false receives in a row
                count = 0;
                record_flag = true;
                closePort();
            }
        }
        else
            count = 0;
    // recording phase
        if(RxBuffer[0] == 'R'&& portOpen){
            if(RxBuffer[5] == ((RxBuffer[1] ^ RxBuffer[2]) ^ (RxBuffer[3]
    ^ RxBuffer[4]))){ // checkbyte
                if(!record_flag)
                {   // switch from adjustment phase to record phase
                    record_flag = true;
                    return false;
                }
                data0 = (0xff00 & (RxBuffer[1]<<8)) | RxBuffer[2];
                data1 = (0xff00 & (RxBuffer[3]<<8)) | RxBuffer[4];
                if(og.getMode() == 0){   // xy-mode
                    if((data0 < MAXRES/XYDIV) && (data1 < MAXRES/XYDIV))
                        // assign new data
```

```
                    if(!og.xy_new_point(dc, data0, data1))
                        closePort();  // buffer is full
                }
                else{                   // ty-mode
                    if((data0 < MAXRES) && (data1 < MAXRES/XYDIV)){
                        // assign new data
                        if(!og.ty_new_point(dc, data0, data1))
                            closePort();
                    }
                }
                newData = true;
            }
        }
    // adjustment phase
        else if(RxBuffer[0] == 'A' && portOpen){
        // 1: mode, 2: xRes or tRun, 3: yRes 4: y offset (upper byte),
        // 5: y offset (lower byte) , 6: x offset (upper), 7: x offset (lower)
            if(record_flag          // end of recording phase
            {
                record_flag = false;
                return false;       // return to adjustments again
            }
            og.setMode(RxBuffer[1] & 0x00ff);   // mode
            og.setXres(RxBuffer[2] & 0x00ff);   // x-res or runtime
            og.setYres(RxBuffer[3] & 0x00ff);   // y-res
            og.setOffset(false, ((RxBuffer[4]<<8) & 0xff00) | (RxBuffer[5]
                & 0x00ff));         // y offset
            og.setOffset(true, ((RxBuffer[6]<<8) & 0xff00) | (RxBuffer[7]
                & 0x00ff));         // x offset

            for(int i = 0; i < 8; i++){
                // change in first 4 bytes detected
                if((lastRx[i] != RxBuffer[i]) && (i < 4))
                {
                    newData = true;
                    lastRx[i] = RxBuffer[i]; // assign new value
                }
                // y offset
                else if((((int)(((lastRx[4]<<8) & 0xff00) | (lastRx[5] & 0x00ff))
                    > (int)((((RxBuffer[4]<<8) & 0xff00) | (RxBuffer[5] &
                    0x00ff))*1.01) || ((int)(((lastRx[4]<<8) & 0xff00) |
                    (lastRx[5] & 0x00ff)) < (int)((((RxBuffer[4]<<8) & 0xff00) |
                    (RxBuffer[5] & 0x00ff))/1.01))) ||
                // x offset
                    ((int)(((lastRx[6]<<8) & 0xff00) | (lastRx[7] & 0x00ff))
                    > (int)((((RxBuffer[6]<<8) & 0xff00) | (RxBuffer[7] & 0x00ff))
                    *1.01) || ((int)(((lastRx[6]<<8) & 0xff00) | (lastRx[7] &
                    0x00ff)) < (int)((((RxBuffer[6]<<8) & 0xff00) | (RxBuffer[7] &
                    0x00ff))/1.01)))) && (i >= 4))
                {       // assign new, if new offset differs more than
                        // 0.5% of previous stored value
                    newData = true;
                    lastRx[i] = RxBuffer[i];
                }
        }                               // end of adjustment phase
    } while(record_flag && portOpen);   // always exit in adjustment phase
    return newData;
}
```

```
// open the port to the ft232 chip
bool CSerialCom::openPort()
{
    ftStatus = FT_Open(0, &ftHandle);
    if (ftStatus != FT_OK) {
        // FT_Open OK, use ftHandle to access device
        return false;     // connection established
    }
    // baudrate setting
    ftStatus = FT_SetBaudRate (ftHandle, BaudRate);
    if (ftStatus != FT_OK)
        return false;
    // Data characteristics: 8 bit, 1 stop bit, no parity bit
    ftStatus = FT_SetDataCharacteristics (ftHandle, FT_BITS_8, FT_STOP_BITS_1,
    FT_PARITY_NONE);
    if (ftStatus != FT_OK)
        return false;
    // no handshaking
    ftStatus = FT_SetFlowControl (ftHandle, FT_FLOW_NONE, NULL, NULL);
    if (ftStatus != FT_OK)
        return false;
    // timeout for read and write access of ft232
    ftStatus = FT_SetTimeouts(ftHandle,1000,1000);
    if (ftStatus != FT_OK)
        return false;
    return true;                    // if succeeded
}

// closes connection
bool CSerialCom::closePort()
{
    ftStatus = FT_Close (ftHandle);
    if (ftStatus == FT_OK) {
        savePoint = 1;
        portOpen = false;
        return true;                // connection closed
    }
    else {
        return false;
    }
}

// transmit one byte
bool CSerialCom::transmit(unsigned char* TxBuffer)
{
    DWORD BytesWritten;
    ftStatus = FT_Write (ftHandle, TxBuffer, 1, &BytesWritten);
    if (ftStatus == FT_OK) {
        return true;                // FT_Write OK
    }
    else {
        return false;               // FT_Write Failed
    }
}

// receive one byte
bool CSerialCom::receive_byte()
{
    DWORD RxBytes = 1;
    DWORD BytesReceived = 0;
```

```
    ftStatus = FT_Read(ftHandle,&RxBuffer[0],RxBytes,&BytesReceived);
    if (ftStatus == FT_OK) {
        if (BytesReceived == RxBytes) {
            return true;              // FT_Read OK
        }
        else {
            return false;             // FT_Read Timeout
        }
    }
    else {
        return false;                 // FT_Read Failed
    }
}

// receive a dataset of adjustment info or recorded point
bool CSerialCom::receive_dataset()
{
    DWORD BytesReceived = 0;
    DWORD Bytes = 0;
    do{                           // get first byte and check if start of dataset
        ftStatus = FT_Read(ftHandle,&RxBuffer[0],1,&BytesReceived);
    }
    while((ftStatus == FT_OK) && (BytesReceived == 1) && (RxBuffer[0]!='A') &&
    (RxBuffer[0]!='R'));
    if (ftStatus == FT_OK) {          // get the rest
        if (BytesReceived == 1) {
            if (RxBuffer[0]=='A')
                // mode, xRes/runtime, yRes, 2*x-Offset, 2*y-Offset
                Bytes = 7;
            else                      // (RxBuffer[0]=='R')
                Bytes = 5;            // xADC_H, xADC_L, yADC_H, yADC_L, checkbyte
            ftStatus = FT_Read(ftHandle,RxBuffer+1,Bytes,&BytesReceived);
            if (ftStatus == FT_OK) {
                if (BytesReceived == Bytes) {
                    return true;      // FT_Read OK
                }
            }
        }
    }
    return false;                     // FT_Read Failed or Timeout
}

// purge the registers for receive and transmit of ft232
bool CSerialCom::purgeFT232(void)
{
    DWORD dwMask = FT_PURGE_RX + FT_PURGE_TX;
    ftStatus = FT_Purge (ftHandle, dwMask);
    if (ftStatus == FT_OK) {
        return true;                  // FT_PURGE OK
    }
    else {
        return false;                 // FT_PURGE Failed
    }
}
```

```
// ********************************************************************
//
//     File Name: : xySaveFile.h
//     Title      : Class xySaveFile to save all points of graph
//                  to file
//
// ********************************************************************

#pragma once
#ifndef XYSAVEFILE_H
#define XYSAVEFILE_H

#include "xyWriter.h"
#include "OnePoint.h"
#include "OneGraph.h"

// ********************************************************************
//     Class xySaveFile
// ********************************************************************

class CxySaveFile
{
    TCHAR filePath[_MAX_PATH];      // buffer for full path
    TCHAR fileName[_MAX_PATH];      // buffer for filename only

// ***** public functions *****
public:
    CxySaveFile(void);
    ~CxySaveFile(void);
    void xySaveFile(CDC&);          // function with save dialog
};
#endif



// ********************************************************************
//
//     File Name: : xyOpenFile.h
//     Title      : Open file and draw ann imported points
//
// ********************************************************************

#pragma once
#ifndef XYOPENFILE_H
#define XYOPENFILE_H

#include "xyWriter.h"
#include "OnePoint.h"
#include "OneGraph.h"
#include <fstream>
#include <iostream>
using namespace std;

// ********************************************************************
//     Class xyOpenFile
// ********************************************************************

class CxyOpenFile
{
private:
```

```
    unsigned int x, y;
    TCHAR filePath[_MAX_PATH];      // buffer for full path
    TCHAR fileName[_MAX_PATH];      // buffer for filename only

public:
    CxyOpenFile(void);
    ~CxyOpenFile(void);
    bool xyOpenFile(CDC&);          // main open file function
};
#endif



// ********************************************************************
//
//     File Name: :   xySaveFile.cpp
//     Title      :   Saves data to a file
//
// ********************************************************************

#include "stdafx.h"
#include "xySaveFile.h"
#include <fstream>
#include <iostream>
using namespace std;

CxySaveFile::CxySaveFile(void)
{
}

CxySaveFile::~CxySaveFile(void)
{
}

void CxySaveFile::xySaveFile(CDC &dc)
{
    COneGraph og;
    OPENFILENAME ofn;
    memset(filePath, 0, sizeof(filePath));     // get memory
    memset(fileName, 0, sizeof(fileName));

    // Initialize OPENFILENAME
    ZeroMemory(&ofn, sizeof(OPENFILENAME));    // fill memory with zeros
    ofn.lStructSize = sizeof (OPENFILENAME);
    ofn.hwndOwner = NULL;
    ofn.lpstrFile = filePath;
    ofn.nMaxFile = sizeof(filePath)-5;
    ofn.lpstrFileTitle = fileName;
    ofn.nMaxFileTitle = sizeof(filePath)-5;
    // filter for showing files
    ofn.lpstrFilter = L"Text (*.txt)\0*.TXT\0All (*.*)\0*.*\0";
    ofn.nFilterIndex = 1;
    ofn.lpstrDefExt = L".txt";      // standard extension
    ofn.Flags = OFN_PATHMUSTEXIST | OFN_HIDEREADONLY | OFN_CREATEPROMPT;

    // Display the save dialog box.
    if( GetSaveFileName(&ofn) )
    {
        unsigned int x = 0, y = 0;
        ofstream out(fileName);     // save to file
```

```
        // first line in file: mode, xres, yres, offset
        out << og.getMode() << " " << og.getXres() << " " << og.getYres() <<
        " " << og.getOffset(false) << " " << og.getOffset(true) << "\n";
        for(unsigned int i = 1; i < MAXPOINTS; i++)   // add points
        {
            x = og.getXforPt(i);
            y = og.getYforPt(i);
            if(x != -1)              // until last measured point
                out << x << " " << y << "\n";   // Format "x y"
        }
    }
    else
        OutputDebugString(L"Failed to open Open dialog!\n");
}



// ********************************************************************
//
//     File Name: :   xyOpenFile.cpp
//     Title      :   Reads data from a file
//
// ********************************************************************

#include "stdafx.h"
#include "xyOpenFile.h"

CxyOpenFile::CxyOpenFile(void)
{
    x = 0, y = 0;
}

CxyOpenFile::~CxyOpenFile(void)
{
}

bool CxyOpenFile::xyOpenFile(CDC& dc)
{
    COneGraph og;
    OPENFILENAME ofn;
    memset(filePath, 0, sizeof(filePath));     // get memory
    memset(fileName, 0, sizeof(fileName));

    // Initialize OPENFILENAME
    ZeroMemory(&ofn, sizeof(OPENFILENAME));    // fill memory with zeros
    ofn.lStructSize = sizeof (OPENFILENAME);
    ofn.hwndOwner = NULL;
    ofn.lpstrFile = filePath;
    ofn.nMaxFile = sizeof(filePath)-5;
    ofn.lpstrFileTitle = fileName;
    ofn.nMaxFileTitle = sizeof(filePath)-5;
    // filter for showing files
    ofn.lpstrFilter = L"Text (*.txt)\0*.TXT\0All (*.*)\0*.*\0";
    ofn.nFilterIndex = 1;
    ofn.lpstrDefExt = L".txt";          // standard extension
    ofn.Flags = OFN_PATHMUSTEXIST | OFN_HIDEREADONLY | OFN_CREATEPROMPT;

    // Display the open dialog box.
    if( GetOpenFileName(&ofn) )
    {
```

```
        ifstream inp(fileName);      // open from file
        // first line in file: mode, xres, yres, yoffset and xOffset
        int mode, xres, yres, yOffset, xOffset;
        inp >> mode >> xres >> yres >> yOffset >> xOffset;
        og.setMode(mode);
        og.setXres(xres);
        og.setYres(yres);
        og.setOffset(true, xOffset);
        og.setOffset(false, yOffset);
        og.clear_all_points();
        savePoint = 1;
        while(!inp.eof())
        {
            inp >> x >> y;           // read in x and y
            if(savePoint < MAXPOINTS)
            {
                og.ty_new_point(dc, x, y);
                savePoint++;
            }
        }
        return true;
    }
    else
    { // Cancel or failure
        return false;
    }
}



// ********************************************************************
//
//     File Name: : AddGraphFromFile.h
//     Title      : Gets graph from file and calls calls a function
//                     to draw it
//
// ********************************************************************

#pragma once
#ifndef ADDGRAPHFROMFILE_H
#define ADDGRAPHFROMFILE_H

#include "DrawGraphFromFile.h"

// ********************************************************************
//     Class CAddGraphFromFile
// ********************************************************************

class CAddGraphFromFile
{
    TCHAR filePath[_MAX_PATH];      // buffer for full path
    TCHAR fileName[_MAX_PATH];      // buffer for filename only
    TCHAR saveFileName[5][_MAX_PATH];   // buffer for filename only

public:
    CAddGraphFromFile(void);
    ~CAddGraphFromFile(void);

    void addGraphFromFile(CDC&);    // opens a file and gets data
    TCHAR* getFileName(int);        // manages opened filenames
```

```
    void delFileName(int);
};
#endif



// ********************************************************************
//
//     File Name: :   AddGraphFromFile.cpp
//     Title      :   Adds previously stored graph from file to window
//
// ********************************************************************

#include "stdafx.h"
#include "AddGraphFromFile.h"

CAddGraphFromFile::CAddGraphFromFile(void)
{
}

CAddGraphFromFile::~CAddGraphFromFile(void)
{
}

void CAddGraphFromFile::addGraphFromFile(CDC &dc)
{
    OPENFILENAME ofn;
    CDrawGraphFromFile dgff;

    memset(filePath, 0, sizeof(filePath));     // get memory
    memset(fileName, 0, sizeof(fileName));

    // Initialize OPENFILENAME
    ZeroMemory(&ofn, sizeof(OPENFILENAME));    // fill memory with zeros
    ofn.lStructSize = sizeof (OPENFILENAME);
    ofn.hwndOwner = NULL;
    ofn.lpstrFile = filePath;
    ofn.nMaxFile = sizeof(filePath)-5;
    ofn.lpstrFileTitle = fileName;
    ofn.nMaxFileTitle = sizeof(filePath)-5;
    // filter for showing files
    ofn.lpstrFilter = L"Text (*.txt)\0*.TXT\0All (*.*)\0*.*\0";
    ofn.nFilterIndex = 1;
    ofn.lpstrDefExt = L".txt";      // standard extension
    ofn.Flags = OFN_PATHMUSTEXIST | OFN_HIDEREADONLY | OFN_CREATEPROMPT;

    // Display the open dialog box.
    if( GetOpenFileName(&ofn) )
    {
        int count = 0;
        do
        {
            if(*saveFileName[count] == NULL)
            {
                for(int i = 0; i < _MAX_PATH; i++)
                    // save filename in buffer (max 5)
                    saveFileName[count][i] = filePath[i];
                // function to draw the file
                dgff.drawGraphFromFile(dc, count, fileName);
                count = 5;
```

```
            }
            count++;
        } while(count < 5);
    }
    else
    { // Cancel or failure
    }
}

TCHAR* CAddGraphFromFile::getFileName(int no)
{
    return saveFileName[no];
}

void CAddGraphFromFile::delFileName(int no)
{
    *saveFileName[no] = NULL;
}



// ********************************************************************
//
//     File Name: : xyPrintWindow.h
//     Title      : Print the contents of the window
//
// ********************************************************************

#pragma once
#ifndef XYPRINTWINDOW_H
#define XYPRINTWINDOW_H

#include "DrawGraph.h"

// ********************************************************************
//     Class CxyPrintWindow
// ********************************************************************

class CxyPrintWindow
{
    int iWidth, iHeight;            // window size
    HBITMAP hbm;
    HDC dcMem;
    DOCINFO di;
    DIBSECTION ds;
    PRINTDLG pdlg;

// ***** public functions *****
public:
    CxyPrintWindow(void);
    ~CxyPrintWindow(void);
    void printWindow(CDC&);         // main print function

// ***** private functions *****
private:
    HBITMAP createBitmapOfWindow(CDC&, int, int);
};
#endif
```

```
// ****************************************************************
//
//      File Name: :   xyPrintWindow.cpp
//      Title      :   Prints window contents
//
// ****************************************************************

#include "stdafx.h"
#include "xyPrintWindow.h"
#include "DrawGraph.h"

CxyPrintWindow::CxyPrintWindow(void)
{
    iWidth = iHeight = 0;
}

CxyPrintWindow::~CxyPrintWindow(void)
{
}

void CxyPrintWindow::printWindow(CDC &dc)
{
    CDrawGraph drawGraph;
    // create a bitmap of the client window without statusbar (-20)
    hbm = createBitmapOfWindow(dc, drawGraph.clRect.right,
drawGraph.clRect.bottom-20);
    dcMem = CreateCompatibleDC(dc); // Prepare the memory surface for drawing.
    SelectObject(dcMem, hbm);       // assign the object/bitmap to the surface
    // Copy the window contents to the memory surface/ into the bitmap
    BitBlt(dcMem, 0, 0, drawGraph.clRect.right, drawGraph.clRect.bottom, dc, 0,
    0, SRCCOPY);

    ZeroMemory(&pdlg, sizeof(PRINTDLG));   // fill memory with zeros
    pdlg.lStructSize = sizeof (PRINTDLG);
    pdlg.Flags = PD_RETURNDEFAULT;
    // first call to get a handles for devmode, does not open print dialog
    PrintDlg(&pdlg);
    // start/standard values of dialog
    DEVMODE* dm = (DEVMODE*)::GlobalLock(pdlg.hDevMode);
    dm->dmOrientation = DMORIENT_LANDSCAPE;    // landscape
    dm->dmPaperSize = DMPAPER_A4_ROTATED;      // DIN A4
    GlobalUnlock(pdlg.hDevMode);
    pdlg.hwndOwner = drawGraph.hWnd;
    pdlg.Flags = PD_RETURNDC;       // order handle for printer attach

    if (PrintDlg(&pdlg)) {          // open print dialog
        CDC dcPrint;
        dcPrint.Attach(pdlg.hDC);   // attach dialog info to printer
        iWidth = GetDeviceCaps(pdlg.hDC, HORZRES); // get paper size
        iHeight = GetDeviceCaps(pdlg.hDC, VERTRES);

        ZeroMemory(&di, sizeof(di));// Prepare the DOCINFO.
        di.cbSize = sizeof(di);
        di.lpszDocName = L"Graph";

        if (StartDoc(dcPrint, &di) != 0)    // start the print job
        {
            if (StartPage(dcPrint) != 0)// Prepare driver to accept data
            {   // Get the information describing the surface/bitmap
                GetObject(hbm, sizeof(DIBSECTION), &ds);
```

```
                // Print the contents of the surface/bitmap
                if(iHeight/ds.dsBm.bmHeight >= iWidth/ds.dsBm.bmWidth)
                    // static width and relative height
                    StretchBlt(dcPrint, 500, 500, iWidth-1000,
                        (int)(ds.dsBm.bmHeight * (float)((iWidth-
                        1000.0)/ds.dsBm.bmWidth)), dcMem, 0, 0, ds.dsBm.bmWidth,
                        ds.dsBm.bmHeight, SRCCOPY);
                if(iHeight/ds.dsBm.bmHeight < iWidth/ds.dsBm.bmWidth)
                    //static height and relative width
                    StretchBlt(dcPrint, 500, 500, (int)(ds.dsBm.bmWidth *
                        (float)((iHeight-1000.0)/ds.dsBm.bmHeight)), iHeight-1000,
                        dcMem, 0, 0, ds.dsBm.bmWidth, ds.dsBm.bmHeight, SRCCOPY);
                EndPage(dcPrint);    // tell driver print job is done
            }   // tell driver the document is done / end print job
            EndDoc(dcPrint);
        }
        DeleteDC(dcPrint);          // Clean up the created objects
        DeleteDC(dcMem);
        dcPrint.Detach();
        DeleteObject(hbm);
    }
    else
        OutputDebugString(L"Failed to open print dialog!\n");
}

// create a bitmap of the current window
HBITMAP CxyPrintWindow::createBitmapOfWindow(CDC &dc, int iWidth, int iHeight)
{
    BITMAPINFO bmi;
    HBITMAP hbm;
    LPBYTE pBits;

    // Initialize to 0s.
    ZeroMemory(&bmi, sizeof(bmi));

    // Initialize the header.
    bmi.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
    bmi.bmiHeader.biWidth = iWidth;
    bmi.bmiHeader.biHeight = iHeight;
    bmi.bmiHeader.biPlanes = 1;
    bmi.bmiHeader.biBitCount = 24;
    bmi.bmiHeader.biCompression = BI_RGB;

    // Create the surface.
    hbm = CreateDIBSection(dc, &bmi, DIB_RGB_COLORS, (void**)&pBits, NULL, 0);

    return(hbm);
}
```

```
// ********************************************************************
//
//      File Name: : SaveWindowAsBMP.h
//      Title      : Saves Window as Bitmap
//
// ********************************************************************

#pragma once

#ifndef SAVEWINDOWASBMP_H
#define SAVEWINDOWASBMP_H

#include "DrawGraph.h"

// ********************************************************************
//      Class SaveWindowAsBMP
// ********************************************************************

class CSaveWindowAsBMP
{
    HBITMAP hbm;
    BITMAPINFO bmi;
    LPBYTE pBits;
    HDC dcMem;
    BITMAPFILEHEADER bmfh;
    BITMAPINFOHEADER bmih;
    DWORD bytes_write;
    DWORD bytes_written;
    HANDLE fh;
    OPENFILENAME ofn;
    TCHAR filePath[_MAX_PATH];      // buffer for full path
    TCHAR fileName[_MAX_PATH];      // buffer for filename only

public:
    CSaveWindowAsBMP(void);
    ~CSaveWindowAsBMP(void);
    void saveWindowAsBMP(CDC&);
};
#endif



// ********************************************************************
//
//      File Name: :   SaveWindowAsBMP.cpp
//      Title      :   Saves the window as bitmap
//
// ********************************************************************

#include "stdafx.h"
#include "SaveWindowAsBMP.h"

CSaveWindowAsBMP::CSaveWindowAsBMP(void)
{
}

CSaveWindowAsBMP::~CSaveWindowAsBMP(void)
{
}
```

```
// according the internet sources http://www.online-tutorials.net/grafik/bitmap-
// speichern/sourcecodes-t-17-21.html 15.4.2008 and
// http://www.wer-weiss-was.de/theme158/article1410552.html  15.4.2008
void CSaveWindowAsBMP::saveWindowAsBMP(CDC &dc)
{
    CDrawGraph drawGraph;

    memset(filePath, 0, sizeof(filePath)); // get memory
    memset(fileName, 0, sizeof(fileName));
    // initialise bitmab info header
    ZeroMemory(&bmih, sizeof(BITMAPINFOHEADER));
    bmih.biSize = sizeof(BITMAPINFOHEADER);
    bmih.biHeight = drawGraph.clRect.bottom-20;
    bmih.biWidth = drawGraph.clRect.right;
    bmih.biPlanes=1;
    bmih.biBitCount=24;             // color depth
    bmih.biCompression=BI_RGB;
    bmih.biSizeImage = ((((bmih.biWidth * bmih.biBitCount) + 31) & ~31) >> 3)
    * bmih.biHeight;
    bmih.biXPelsPerMeter = 0;
    bmih.biYPelsPerMeter = 0;
    bmih.biClrImportant = 0;        // all colors

    bmi.bmiHeader=bmih;             // load bitmap info
    hbm = CreateDIBSection(dc, &bmi, DIB_RGB_COLORS, (void**)&pBits, NULL, 0);
    // Create the surface
    if(hbm==NULL)
    {
        OutputDebugString(L"CreateDIBSection failed!\n");
    }
    dcMem = CreateCompatibleDC(dc); // Prepare the memory surface for drawing.
    SelectObject(dcMem, hbm);       // assign the object/bitmap to the surface
    // Copy the window contents to the memory surface/ into the bitmap
    BitBlt(dcMem, 0, 0, drawGraph.clRect.right, drawGraph.clRect.bottom, dc,
    0, 0, SRCCOPY);

    // Initialize OPENFILENAME
    ZeroMemory(&ofn, sizeof(OPENFILENAME));    // fill memory with zeros
    ofn.lStructSize = sizeof (OPENFILENAME);
    ofn.hwndOwner = NULL;
    ofn.lpstrFile = filePath;
    ofn.nMaxFile = sizeof(filePath)-5;
    ofn.lpstrFileTitle = fileName;
    ofn.nMaxFileTitle = sizeof(filePath)-5;
    // filter for showing files
    ofn.lpstrFilter = L"Bitmap (*.bmp)\0*.BMP\0All (*.*)\0*.*\0";
    ofn.nFilterIndex = 1;
    ofn.lpstrDefExt = L".bmp";      // standard extension
    ofn.Flags = OFN_PATHMUSTEXIST | OFN_HIDEREADONLY | OFN_CREATEPROMPT;

    // Display the save dialog box.
    if( GetSaveFileName(&ofn) )
    {
        ZeroMemory(&bmfh, sizeof(BITMAPFILEHEADER));
        bmfh.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);
        // offset to bitmap image bits
        bmfh.bfSize=(3 * bmih.biHeight * bmih.biWidth) + sizeof(BITMAPFILEHEADER)
        + sizeof(BITMAPINFOHEADER);
        bmfh.bfType=0x4d42;             // BM - Filetype
        bmfh.bfReserved1 = 0;
```

```
        bmfh.bfReserved2 = 0;
        // create file with handle and write and read access
        fh = CreateFile(ofn.lpstrFile, GENERIC_READ|GENERIC_WRITE, (DWORD)0,
        NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, (HANDLE) NULL);

        bytes_write=sizeof(BITMAPFILEHEADER);  // write bitmap file header
        if (!WriteFile(fh, (void*)&bmfh, bytes_write, &bytes_written, NULL))
        {
            OutputDebugString(L"WriteFile failed!\n");
        }
        bytes_write=sizeof(BITMAPINFOHEADER);  // write bitmap info header
        if(!WriteFile(fh, (void*)&bmih, bytes_write, &bytes_written, NULL))
        {
            OutputDebugString(L"WriteFile failed!\n");
        }
        bytes_write = bmih.biSizeImage;     // write bitmap image bits
        if(!WriteFile(fh, (void*)pBits, bytes_write, &bytes_written, NULL))
        {
            OutputDebugString(L"WriteFile failed!\n");
        }
        CloseHandle(fh);             // Clean up the created objects
        DeleteDC(dcMem);
        DeleteObject(hbm);
    }
    else
        OutputDebugString(L"Failed to open save dialog!\n");
}



// ************************************************************************
//
//     File Name: : resource.h
//     Title      : dependencies of program resources
//
// ************************************************************************
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by xyWriter.rc
#define IDR_MENU1                       101
#define IDD_OLE_PROPPAGE_SMALL          104
#define IDC_EDIT1                       1015
#define ID_FILE_NEW40001                40001
#define ID_FILE_OPEN40002               40002
#define ID_FILE_SAVE40003               40003
#define ID_FILE_PRINT40004              40004
#define ID_FILE_CLOSE40005              40005
#define ID_EDIT_INTERPOLATE             40006
#define ID_VIEW_REDRAW                  40007
#define ID_VIEW_SCREENRESOLUTION        40008
#define ID_SCREENRESOLUTION_640X480     40009
#define ID_SCREENRESOLUTION_800X600     40010
#define ID_SCREENRESOLUTION_1024X768    40011
#define ID_SCREENRESOLUTION_1280X1024   40012
#define ID_SCREENRESOLUTION_1400X1050   40013
#define ID_SCREENRESOLUTION_1600X1200   40014
#define ID_HELP_INFO                    40015
#define ID_VIEW_NUMBEROFDIGITS          40016
#define ID_NUMBEROFDIGITS_2DIGITS       40024
#define ID_NUMBEROFDIGITS_3DIGITS       40025
```

```
#define ID_NUMBEROFDIGITS_4DIGITS       40026
#define ID_NUMBEROFDIGITS_5DIGITS       40027
#define IDM_FILE_SAVEAS                 40028
#define ID_VIEW_ADDGRAPHFROMFILE        40029
#define ID_INTERPOLATE_LINEAR           40030
#define ID_EDIT_UNDO40031               40031
#define ID_UNDO_POINTINFORMATION        40032
#define ID_UNDO_ADDGRAPH                40033
#define ID_FILE_SAVEASBITMAP            40034
#define ID_EDIT_ADDTEXT                 40035
#define ID_UNDO_ADDTEXT                 40036
#define ID_EDIT_RECORD                  40037
#define ID_EDIT_STOPRECORDING           40038
#define ID_RECORDING_NEWCONNECTION      40039
#define ID_RECORDING_RECORD             40040
#define ID_RECORDING_DISCONNECT         40043

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        106
#define _APS_NEXT_COMMAND_VALUE         40044
#define _APS_NEXT_CONTROL_VALUE         1018
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

E. Inhalt des beigefügten Datenträgers

- Die Diplomarbeit im PDF Format
- Das Programm x/y-Writer.exe
- Die in den Quellen angegebenen Datenblätter
- Die erstellten EAGLE Projekte
- Der Quellcode des Mikroprozessors
- Das Makefile zur Erstellung des Programmcodes für den Mikroprozessor
- Der Quellcode des PC-Programms
- Die Messdaten zu den durchgeführten Funktionstests
- Der FTDI-Treiber für die USB-Kommunikation (für Windows XP)
- Das Programm MProg3.0a für die Programmierung des FTDI-Chips

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 2. Juli 2008

Ort, Datum

Unterschrift