Praktikum am *Max-Planck-Institut für Quantenoptik* (MPQ) in Garching:

Mikrocontrollergesteuerte Strahlstabilisierung für Kurzpulslaser

Jan Krieger

Adresse: *In der Neckarhelle 82, 69118 Heidelberg*
Telefon: *06221 / 89 38 63*
e-mail: *jan jkrieger.de*
Matrikel-Nr.: *2264735*

2. Dezember 2003

Inhaltsverzeichnis

1 Einleitung **3**

2 Messprinzip, optischer Aufbau und Strahlansteuerung **4**

3 Die Regelungsroutine **8**

4 Benutzerinterface **9**

5 Das Microcontroller-System **10**

6 Aufbau der Software **11**

7 Beschreibung der Elektronik **13**
7.1 Quadrantendioden-Platinen 13
7.2 Spannungsversorgung und Hochspannungsverstärker 14
7.3 Microcontroller-Platine 15
7.4 AD/DA-Platine 17

8 Zusammenfassung und Ergebnisse **20**

A Sourcecode **21**
A.1 Makefile für WinAVR/gcc 21
A.2 Tool-Funktionen (`alignment.c`) 22
A.3 Hauptprogramm (`stabilise1.c`) 26

Literaturverzeichnis **33**

Danksagung **33**

1 Einleitung

Dies ist der Abschluss-Bericht zu einem Praktikum, das ich in der Laser-PLasma-Gruppe von Prof. Witte am Max-Planck-Institut für Quantenoptik in Garching bei München[1] absolviert habe. Das Thema ist die räumliche Stabilisierung eines mit bis zu 200 Hz gepulsten Laserstrahles mit sehr kurzer Pulslänge (Nano- bis Femtosekundenbereich $\hat{=}10^{-9}..10^{-15}$ s).

Eine räumliche Stabilisierung ist zum Beispiel nötig, um Temperaturdriften auszugleichen. Die Regelung wurde konkret für den folgenden Anwendungsfall entwickelt: Für die Erzeugung eines energiereichen Femstosekunden-Laserpulses muss ein schwacher, kurzer Puls (fs-Bereich) in einen Verstärkerkristall geschossen werden, der vorher mit einem starken, dafür aber längeren, Puls (ns-Bereich) gepumpt wurde. Der Fokus des Pump-Strahles im Kristall beträgt dabei etwa 0.7 mm, der Fokus des Probe-Strahles ist 0.4 mm. Außerdem läuft der Probe-Puls zehn mal durch den Verstärkerkristall (Multipass-Verstärker). Dabei ist eine genaue Ausrichtung der Fokusse sehr wichtig, um eine möglichst große Verstärkung und ein gutes Strahlprofil (gaußförmig) zu erreichen. Da nun der Pump-Laser und der Verstärker auf verschiedenen Experimentiertischen untergebracht und die Strahlwege mit mehreren Metern sehr lang sind, ist eine räumliche Stabilisierung der Strahlen unbedingt nötig. Eine kurze Rechnung zeigt etwa, dass die räumliche Strahlabweichung Δx schon bei einer Winkeländerung des Einfallstrahles von 0.05° und einem Strahlweg von $l = 5\,\mathrm{m}$ mit $\Delta x = l \cdot \sin(0.05°) = 4.4\,\mathrm{mm}$ sehr groß gegen den Fokusdruchmesser ist. Die Abweichungen, die ausgeglichen werden, bewegen sich auf einer Zeitskala von einigen Sekunden und länger. Sie entstehen z.B. durch thermische Driften des Aufbaus. Der beschriebene Laserverstärker wird von Dr. Michael Jost aufgebaut, der mich auch während meines Praktikums betreute.

Ein weiterer Anwednugsfall ist dann die Positionierung des erzeugten Femtosekunden-Laserpulses im Experiment von Dipl. Phys. Matthias Dreher. Hier stellt sich ein ganz ähnliches Problem. Das Experiment versucht superradiante Verstärkung von Laserpulsen in Plasmen nachzuweisen. Dazu wird mit einem langen, energiereichen Pump-Impuls (ps-Bereich) ein Plasma in einem Gasjet erzeugt. Entgegengesetzt zur Pump-Strahlrichtung wird dann der fs-Impuls durch das Plasma geschickt. Dabei erhält man eine Verstärkung des fs-Pulses. Da die Pulse hierfür über lange Strecken (einige Meter) herangeführt werden, reagiert das Experiment sehr sensitiv auf kleine Winkeländerungen. Außerdem müssen die Strahlen noch zeitlich überlagert werden (siehe [Dreher 2003]).

Das Praktikum dauerte vom 1.7.2003 ab acht Wochen. Ich habe vor einem Jahr bereits ein Praktikum in der selben Arbeitsgruppe absolviert. Damals entstanden erste Ideen und Entwürfe für die Strahlstabilisierung. Die Umsetzung und der Aufbau erfolgte aber während dieses Jahres.

In diesem Bericht werde ich zuerst den grundlegenden Aufbau, sowie das Funktionsprinzip erläutern. Später folgen dann noch Abschnitte über die Implementierung in Hard- und Software.

[1] Max-Planck-Institut für Quantenoptik (MPQ), Hans-Kopfermann-Straße 1, D-85748 Garching, `http://www.mpq.mpg.de`

2 Messprinzip, optischer Aufbau und Strahlansteuerung

Zur räumlichen Stabilisierung eines Laserstrahles sind zwei feste Raumpunkte nötig. Mit nur einem lässt sich zwar die Position eines Strahles konstant halten, aber nicht seine Richtung. Die Position an den festen Raumpunkten wird über je eine Quadrantendiode gemessen. Dabei handelt es sich um vier extrem genau aufeinander abgestimmte Photodioden-Segmente, die durch einen feinen Spalt getrennt sind (siehe Abb. 2). Trifft ein Lichtfleck auf einen solchen Detektor, dann teilt sich die Lichtleistung auf die einzelnen Quadranten auf. Aus den Photoströmen der einzelnen Quadranten I_n lässt sich dann ein Wert für die Abweichung $I(\Delta x), I(\Delta y)$ des Lichtfleckes vom Zentrum der aktiven Fläche berechnen:

$$I(\Delta x) = \frac{I_A + I_C - (I_B + I_D)}{I_A + I_B + I_C + I_D} \qquad I(\Delta y) = \frac{I_A + I_B - (I_C + I_D)}{I_A + I_B + I_C + I_D}$$

Die Diode wird also jeweils in zwei Hälften aufgeteilt, indem zwei benachbarte Quadranten addiert werden. Die Division durch die Summe über alle Ströme entspricht einer Normierung auf die Strahlintensität, um deren zeitliche Instabilität auszugleichen.

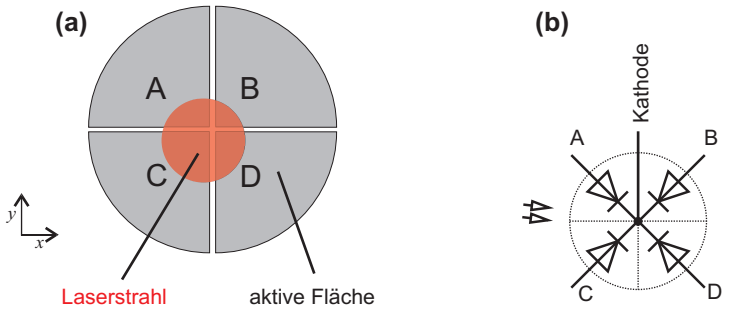

Abb. 1: *Quadrantendiode:* ***(a)*** *Schemazeichnung;* ***(b)*** *Schaltplan*

Eine Quadrantendiode kann nur dann ein Positionssignal liefern, wenn der Strahl den Spalt zwischen den Quadranten überdeckt. Sonst bleibt der Photostrom unabhängig von der Position des Strahles konstant. Dies kann allerdings noch als sinnvolles Eingangssignal für eine Regelung interpretiert werden (Strahl weit weg). Es beschränkt aber die maximal messbare, d.h. auflösbare Weglänge auf einen Strahldurchmesser. Durch Defokussieren des Strahles kann eine höhere Auflösung erreicht werden. Die Beziehung zwischen Positionsänderung eines Lichtflecks und Änderung der Photoströme ist nichtlinear (siehe auch [Laser 2000 2003b]). Dies kann für eine Regelung, die auf den Mittelpunkt der Diode regelt ($\Delta x = 0; \quad \Delta y = 0$) aber vernachlässigt werden.

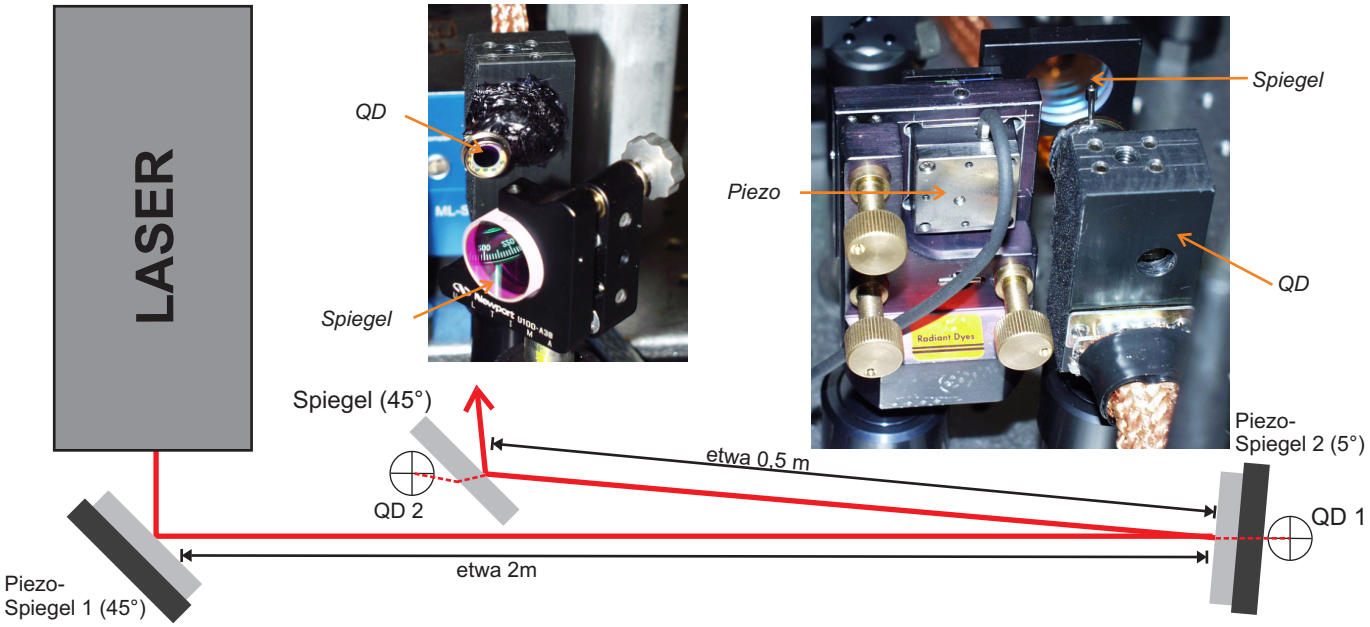

Abb. 2: *Beispiel für den Aufbau einer Strahlstabilisierung am MPQ.*

Die Steuerung des Strahles erfolgt über Piezo-getriebene Spiegel. Dabei kann jeder Spiegel eine Verkippung in x- und y-Richtung ausführen. Jeder Diode ist dabei ein Spiegel zugeordnet (QD 1 $\leftrightarrow$ Spiegel 1, QD 2 $\leftrightarrow$

Spiegel 2), sodass jedes Paar als unabhängiger Regelkreis funktioniert. Der Aufbau kann beispielsweise wie in Abb. 2 erfolgen. Die Strecken zwischen Diode und Spiegel sollten lang gestaltet werden, um eine möglichst hohe Auflösung der Position auf den Dioden zu erhalten (kleine Winkeländerungen verursachen größere Ausschläge). Allerdings ist zu beachten, dass dann die Anzahl der nutzbaren Spiegelstellungen bei digitaler Ansteuerung kleiner wird und dies die Regelgenauigkeit beschränkt.

Da auch gute Spiegel immer nur einen Teil des Lichtes reflektieren, kann der restliche, transmitierte Anteil ($\approx 0.1\%$) auf die Quadrantendioden geleitet werden. So kann die Position des Strahles ohne Eingriffe in den Strahlengang bestimmt werden. Außerdem können so auch Laserstrahlen mit höherer Intensität vermessen werden, weil durch den Spiegel bereits eine Abschwächung des Strahles stattfindet. Diese Methode hat allerdings den Nachteil, dass durch die Dicke d des Glasträgers hinter der spiegelnden Schicht je nach Einstrahlwinkel ein vom Einfallswinkel abhängiger, paralleler Strahlversatz $\delta(\alpha)$ auftritt. Er berechnet sich nach [Stöcker 2000] zu (α, β, n, d: siehe Abb. 2(a)):

$$\delta(\alpha) = d \cdot \frac{\sin(\alpha - \beta)}{\cos\beta} \qquad \text{mit} \qquad \beta = \sin^{-1}\left(\frac{\sin(\alpha)}{n}\right).$$

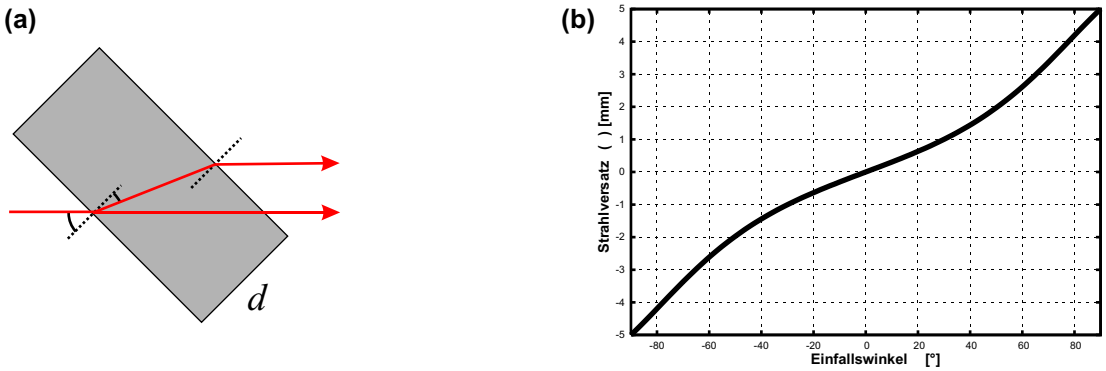

Abb. 3: *Paralleler Strahlversatz δ an einer Glasplatte (Dicke d, Brechungsindex n) in Abhängigkeit vom Einfallswinkel α eines Lichtsrahles.* ***(a)*** *Schemazeichnung;* ***(b)*** *funktionaler Zusammenhang zwischen Einfallswinkel α und Strahlversatz $\delta(\alpha)$ für $n = 1.53; d = 5$ mm.*

Abb 2(b) zeigt den funktionalen Zusammenhang zwischen Strahlversatz und Einfallswinkel. Wie man leicht erkennen kann, ist die Steigung der Kurve um den Einfallswinkel $\alpha = 0°$ flacher, als bei größeren Winkeln. Dort ist also die Änderung des Strahlversatzes bei gleicher Änderungen des Einfallswinkels kleiner, als etwa bei $\alpha = 45°$. Der Strahlversatz kommt bei einer Verschiebung des Lichtpunktes auf dem Spiegel, bei der sich ebenfalls der Einfallswinkel ändert, zu dieser Verschiebung als systematischer, variabler Fehler hinzu. Das Ziel ist also, diesen Fehler möglichst Einfallswinkel-unabhängig zu gestalten. Dies erreicht man offensichtlich durch möglichst kleine Einfallswinkel.

Eine Regelung (siehe Abb. 2(a)) sollte also versuchen mittels Verstellungen am Spiegel die Position des Strahles in der Mitte der Diode zu halten, also die Abweichungen Δx bzw. Δy zu minimieren. Für einen kontinuierlichen Laserstrahl (CW-Laser) kann dies über einen analogen Proportional/Integral-Regler (PI-Regler) erfolgen (siehe Abb. 2(b)). Dabei sorgt der Proportional-Anteil für das Ausregeln von schnelleren und der Integralteil (Tiefpassverhalten) für das Ausgleichen von längerfristigen Schwankungen.

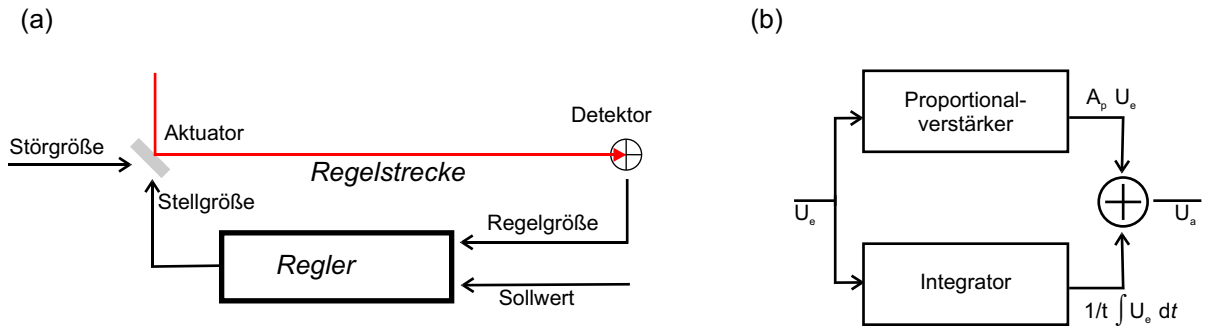

Abb. 4: ***(a)*** *Regelkreis der Strahlstabilisierung;* ***(b)*** *Blockdiagramm eines Proportional-Integralreglers, nach [Tietze, Schenk 2002]*

Bei langsam gepulsten Lasern (bis zu einigen kHz) versagt dieser Ansatz, weil die analoge Regelung während der Pausen zwischen zwei Pulsen kein Eingabesignal bekäme, und somit keine sinnvolle Funktion erfüllen kann. Daher wurde für die hier vorliegenden, extrem kurzen Pulse (ns/fs) ein digitales Regel-Verfahren gewählt. Dies ist außerdem auch nötig, da bei dem verwendeten fs/ns-Oszillator Schuss-zu-Schuss-Schwankungen sowohl in der Intensität, als auch in der Position auftreten (zufälliges „Wackeln“ des Strahles). Diese Störung kann, wegen ihrer zufälligen Natur nicht ausgeregelt werden. Es muss also über mehrere Schüsse die Position gemittelt und erst dann auf den Mittelwert geregelt werden. Die Regelung digitalisiert also die Signale von den Dioden, berechnet daraus eine Position, mittelt diese über mehrere Pulse und steuert dann den Spiegel an.

Pro Regel-Box wurden zwei Regelstrecken implementiert, sodass mit einer Box ein Strahl richtungsstabilisiert werden kann. Die zwei Kanäle sind dabei so aufeinander abgestimmt, dass sie um eine halbe Regelperiode versetzt arbeiten, da sie sich sonst gegenseitig stören würden. Das bedeutet allerdings auch, dass beide Regelungen über dieselbe Anzahl an Schüssen mitteln, was aber ein guter Kompromiss ist, da diese Anzahl so gut wie nur vom Laser abhängt und somit für beide Regelungen gleich angesetzt werden kann.

Ein weiteres Problem ergibt sich durch die kurze Pulslänge, da immer nur einer von acht Quadranten zur selben Zeit ausgelesen werden kann und die Digitalisierung dieses Wertes mit ca. 20 μs wesentlich länger als ein Schuss ($\leq$ 10 ns) dauert. Deswegen wird nicht der Photostrom I_{Photo} direkt gemessen, sondern die Ladung, die er auf einen Kondensator bringt. Abb. 2(a) zeigt den Aufbau des gesamten Systems. Dort erkennt man auch die Beschaltung der Dioden.

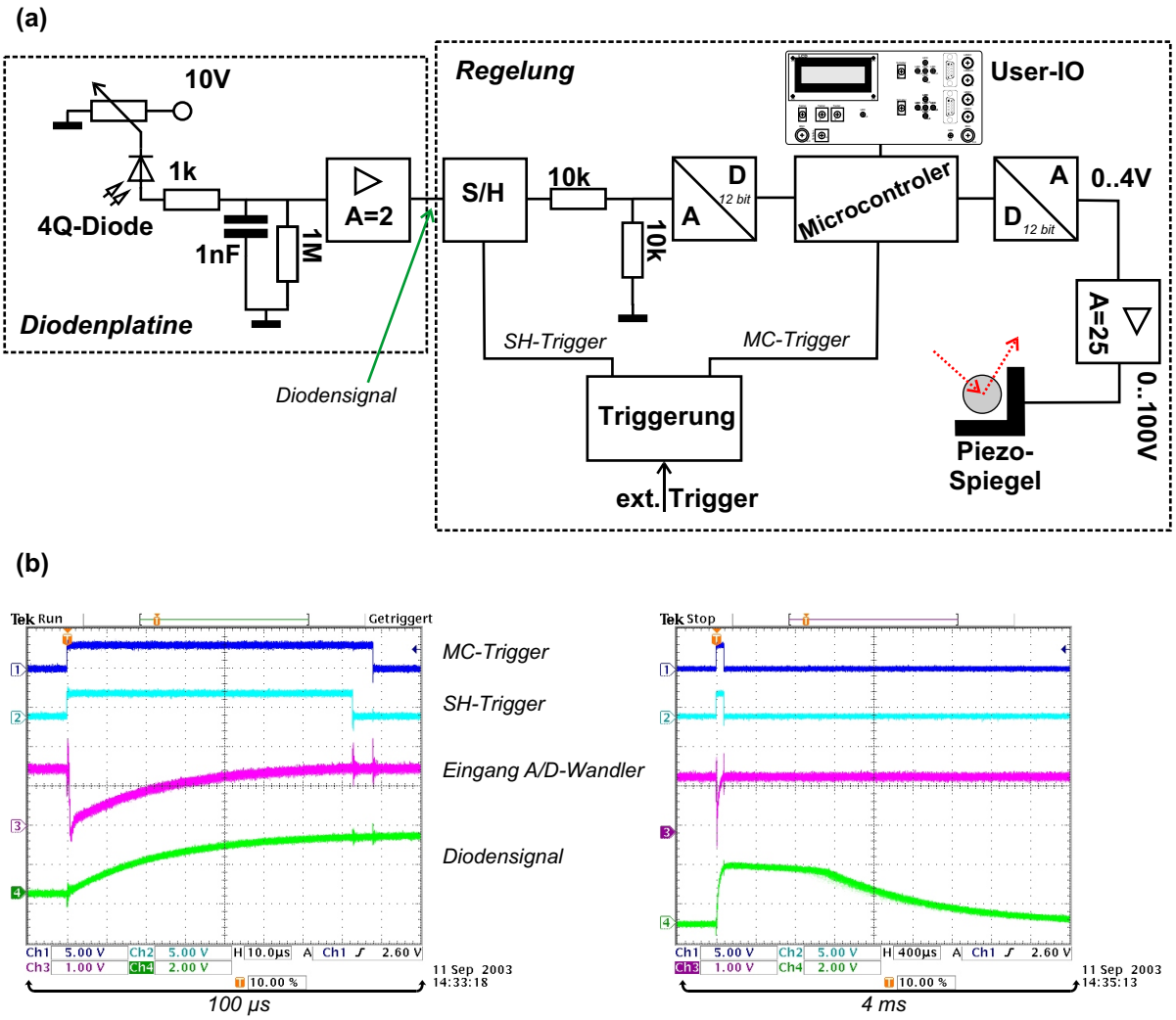

Abb. 5: *(a) Überblick über die elektronische Schaltung der Strahlstabilisierung;* **(b)** *Oszillogramme für den extern getriggerten Betrieb bei* 10 Hz *(MC = Microcontroller; SH = Sample-and-Hold)*

Da für den Strom $I = \frac{\Delta Q}{\Delta t}$ gilt, ist die Ladung auf dem Kondensator (1 nF) nach den Dioden sowohl proportional zur konstanten Pulsdauer Δt, als auch zum Photostrom I_{Photo}. Dieser wiederum hängt von der eingestrahlten Lichtleistung und damit von der bestrahlten Fläche ab. Somit ist die Ladung auf dem Kondensator ein direktes Maß für die eingestrahlte Energie auf einem Quadranten ($\Delta Q = I_{Photo} \cdot \Delta t \propto$ Fläche).

Diese Ladung wird über die am Kondensator nach $U = \frac{Q}{C}$ abfallende Spannung gemessen. Diese Spannugen (je eine für jeden Quadranten) wird kurz nach dem Impuls in eine Sample-and-Hold-Stufe (SH-Stufe, Prinzipschaltbild siehe Abb. 2) gespeichert. Der Zeitpunkt des Speicherns wird dort gesetzt, wo sich die Spannung am Kondensator weitgehend stabilisiert hat, die Ladekurve also in die Entladekurve übergeht (siehe Abb. 2(b)). Bei einer SH-Stufe handelt es sich um einen Kondensator mit einem Analogschalter, dem ein hochohmiger Verstärker nachgeschaltet ist. Im Sample-Modus folgt die Spannung am Kondensator der angelegten Spannung. Im Hold-Modus wird der Kondensator vom Eingangssignal abgekoppelt und auf den hochohmigen Eingang des Ausgangsverstärkers gelegt. Somit kann ein Analog-Wert über eine relativ lange Zeit (einige ms genügen hier) gut konstant gehalten werden, da die Spannung am Kondensator nur langsam über den Leckstrom durch den Ausgangsverstärker absinkt.

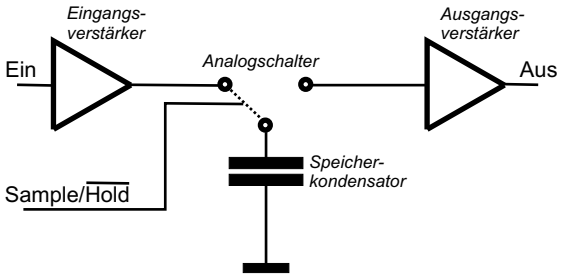

Abb. 6: *Prinzipschaltbild einer Sample and Hold-Stufe*

Wie man in Abb. 2 erkennt, ist dem Ladungsmesskondensator, ein 1 MΩ-Widerstand parallelgeschaltet, der dafür sorgt, dass die Ladung vom Kondensator auch wieder abfließt und die Spannung am Kondesator vor jedem Schuss 0 V beträgt. Dieser Widerstand muss an die Pulsfrequenz angepasst werden. Der 1 kΩ-Widerstand dient zur Strombegrenzung beim Laden des Kondensators.

Das zweite Oszillogramm in Abb. 2 zeigt, wie der Strom, der von den Messkondensatoren kommt (*Diodensignal*) nach dem Schuss langsam wieder auf 0 V absinkt, während das AD-Wandler-Eingangssignal konstant gehalten wird. Der in der SH-Stufe gespeichert Wert wird dann, kurz nach dem Schuss, digitalisiert und verarbeitet. Dieser gesamte Vorgang muss also synchron zum ankommenden Schuss getriggert werden. Dazu verfügt die Regelung über einen Triggereingang, der eine steigende Flanke eines TTL-Signals ($0 \rightarrow 5$ V) erwartet, die gleichzeitig mit dem Schuss ankommen sollte. Die exakte Abstimmung ist aber unkritisch. Es kann also z.B. mit der in Abb. 2 gezeigten Schaltung ein Triggersignal aus einem Reflex des Laserstrahles erzeugt werden. Dabei schaltet der Comparator U1 dann den Ausgang auf 1, wenn der Photostrom einen bestimmten, über P1 einstellbaren Schwellenwert überschreitet.

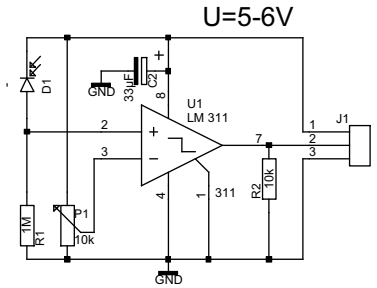

Abb. 7: *Photodiodenbeschaltung zur Triggerung der Reglung (D1: Photodiode; U1: LM311-Komparator-IC)*

Aus dem so erzeugten Triggersignal werden zwei Signale durch Verzögerung abgeleitet: Einmal der SH-Trigger, der die SH-Stufen schaltet. Er ist ab dem Eintreffen des Pulses auf *high* und geht mit einer Verzögerung von etwa 72 μs wieder auf *low*. In dieser Zeit werden Spannungen von den Dioden gesampelt, danach gehalten. Etwas später (etwa 78 μs nach Eintreffen des Triggersignals) wird dann der Microcontroller benachrichtigt, dass er die Spannungen digitalisieren kann. Das Timing zeigen auch die zwei Oszillogramme in Abb. 2. Die Triggerung erfolgt dabei durch eine positive Flanke am Triggereingang; der Microcontroller reagiert auf die erzeugte, negative Flanke.

Zusätzlich zur externen Triggerung ist auch eine interne Triggerung möglich. Dabei läuft die Regelung so schnell wie möglich (etwa 270 Hz). Dieser Modus war vorgesehen, um auch CW-Laser nachregeln zu können. Allerdings hat sich gezeigt, dass dafür die Regelfrequenz zu niedrig ist, und so bereits Schwankungen im Bereich von einigen Hertz nicht mehr ausgeregelt werden konnten.

Nachdem eine neue Spiegelstellung berechnet wurde, wird sie in eine korrespondierende Spannung umgerechnet, die dann über DA-Wandler vom Microcontroller ausgegeben wird. Diese Ausgangsspannung liegt im Bereich von 0..4.096 V. Die Piezos werden mit 0..100 V ausgesteuert, was etwa einer Verkippung von 10 μm entspricht. Deshalb muss die Spannung noch über Hochspannungsverstärker mit einer Verstärkung von etwa $A = 25$ auf diesen Größenbereich gebracht werden. Da die Piezos keine Ohm'schen, sondern kapazitive Lasten darstellen, sind hier keine Leistungsverstärker nötig, nur für Schaltstrombegrenzung ist zu sorgen. Denkbar wäre auch eine Modifikation des Ausgangsteils und der Software, um Spiegel über Servo-Motoren anzusteuern. Dann müssten die Hochspannungsverstärker durch Leistunsgverstärker ersetzt werden.

3 Die Regelungsroutine

An der Regelung können über ein LC-Display verschiedene Parameter eingestellt werden. Sie werden im folgenden Text in der Schriftart `monospace` dargestellt. Die Abb 3 zeigt ein Flussdiagramm dieser Routine.

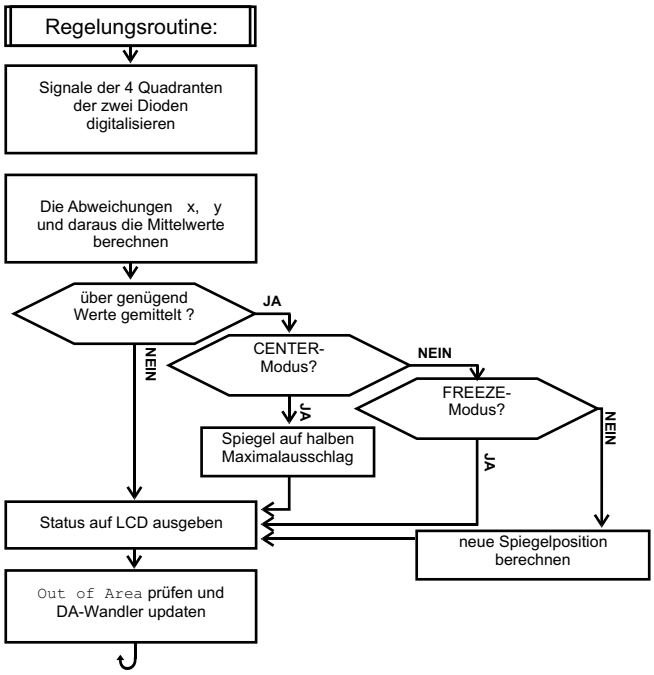

Abb. 8: *Flussdiagramm der Regelroutine*

Aus den vier digitalisierten Spannungen A, B, C und D (jeweils zwischen -2047 und 2047) für die einzelnen Quadranten werden dann, wie in Abschnitt 2 erklärt, zwei Werte Δx und Δy berechnet, die ein Maß für die Abweichung der Strahlposition vom Mittelpunkt sind. Für die Grund-Ausrichtung der Diode ergibt sich:

$$\Delta x = \frac{(A+C)-(B+D)}{A+B+C+D}; \qquad \Delta y = \frac{(A+B)-(C+D)}{A+B+C+D}$$

Die Ausrichtung der Diode kann an der Regelung mit dem Parameter `DiodeDirection` in 90°-Schritten eingestellt werden. Hier sind die Eingaben `left, right, up, down` möglich.

Mit diesen Signalen werden dann Mittelwerte $\overline{x}, \overline{y}$ nach folgender Formel gebildet:

$$\overline{x} = \frac{MeanFactor}{10} \cdot \overline{x}_{alt} + \frac{10 - MeanFactor}{10} \cdot \Delta x$$

Der Parameter `MeanFactor` $\in$ [0..10] gibt an, mit welcher Gewichtung die neuen Werte in den gleitenden Mittelwert $(\overline{x}, \overline{y})$ einfließen. Alte Werte werden oft mit $\frac{MeanFactor}{10}$ multipliziert und „fallen“ somit langsam aus dem Mittelwert heraus.

Die Regelung mittelt so über die im Parameter `Samples` festgelegte Anzahl von Messwerten, bevor eine Nachregelung der Position erfolgt. Über den Parameter `Samples` kann also die Trägheit der Regelung und ihre Empfindlichkeit gegenüber zufälligen Positionsschwankungen eingestellt werden. Für 10Hz-Betrieb haben sich hier Werte zwischen 2 (5 Nachstellungen/s) und 10 (1 Nachstellung/s) als gute Wahl erwiesen. Hier muss vor allem berücksichtigt werden, wie stark der Strahl „wackelt“.

Aus dem so berechneten Mittelwert wird dann eine Änderung $\Delta U_x, \Delta U_y$ der Ausgangsspannung berechnet. Dazu dient folgende Formel:

$$\Delta U_x = SignX \cdot \frac{RegFactor}{RegFraction} \cdot \overline{x}$$

Der Parameter `SignX (SignY)` $\in [-1, +1]$ gibt die Richtung an, in die sich die Ausgangsspannung ändern soll. Dies ermöglicht es, die Position des Piezos (Drehungen um 180°) zu berücksichtigen. Eine Drehung der Piezos um 90° muss durch Umstecken der X- und Y-Spannungen an diesen eingestellt werden. Mit dem Term $\frac{RegFactor}{RegFraction}$ kann die Reaktion des Reglers auf eine Abweichung eingestellt werden (Hier ist eine Aufteilung in zwei Zahlen erforderlich, da der Microcontroller nur mit Integer-Arithmetik arbeitet, um zeitintensive Gleitkomma-Arithmetik zu umgehen). Große Werte bewirken eine starke Reaktion auf eine Abweichung und somit eine schnelle Regelung auf die Null-Position, aber erhöhen auch die Gefahr einer Übersteuerung (→ mehr oder weniger gedämpfte Schwingung um den Nullpunkt). Kleine Werte führen zu einer langsameren Einstellung des Nullpunktes, verhindern aber auch Übersteuerung (→ Kriechfall).

Sollte die Regelung einmal die maximale bzw. minimale Ausgangsspannung erreichen, so gibt es die Möglichkeit, ihr Verhalten über den Parameter `Out of Area` festzulegen. Ist diese Funktion aktiviert (true), so stellt die Regelung beim Erreichen dieser Schranken auf den halben Maximalausschlag zurück und ermöglicht so ein neues Auffinden der Zentralposition. Wird dies nicht gewünscht, so kann die Funktion deaktiviert werden. Der Piezo verharrt dann in der Position der maximalen bzw. minimalen Auslenkung.

Die zwei Kanäle sind wie gesagt so programmiert, dass sie jeweils um eine halbe Regelperiode ($Samples/2$) versetzt ihre Änderungen ausgeben. Dies erhöht die Stabilität der Regelung, da das „Gegeneinanderarbeiten“ der Kanäle weitgehend verhindert wird.

Zusätzlich zum eben beschriebenen Regelmodus können die Kanäle einzeln eingefroren werden (FREEZE-Modus) und es ist möglich, die Piezos auf den halben Maximalausschlag zu stellen (CENTER-Modus). Dies ermöglicht es, die Optik zu justieren, da hierbei der mögliche Ausschlag zum Ausregeln von Störungen nach beiden Seiten maximal ist.

4 Benutzerinterface

Zur Interaktion mit dem Benutzer verfügt die Regelungs-Box über verschiedene Schalter und Taster, sowie ein LC-Display (siehe Abb. 4).

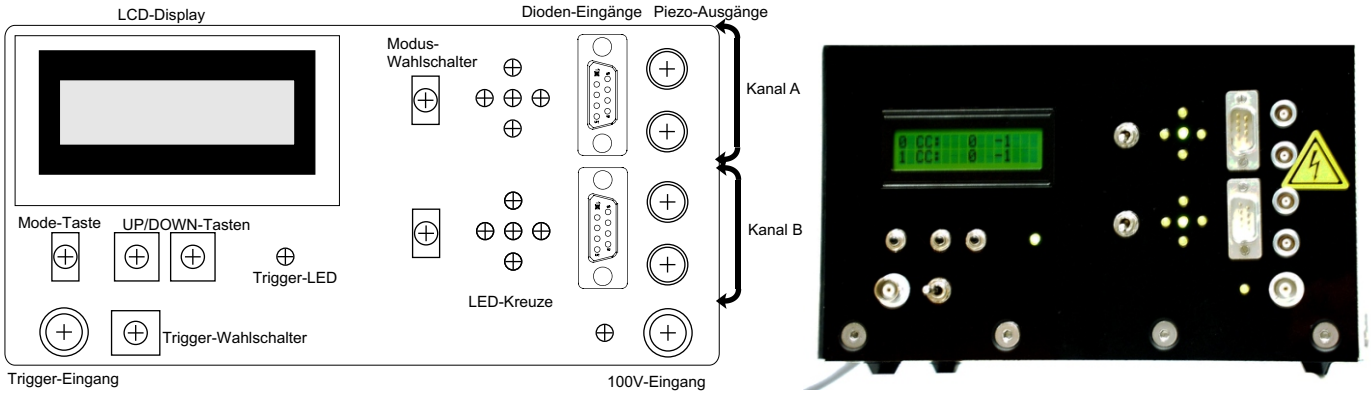

Abb. 9: *Frontplatte der Regelung*

Mit den Mode-Schaltern können die einzelnen Kanäle eingefroren (FREEZE) und auf eine mittlere Spiegelstellung (CENTER, zum Justieren) gebracht werden (siehe Abschnitt 3). In all diesen Modi werden die aktuellen Abweichungen als Zahlenwerte auf dem LC-Display angezeigt. So kann man einerseits die Funktion der Regelung überprüfen und hat andererseits einen gutes Anzeigegerät für die Justage der Optik. Zusätzlich gibt es je ein Kreuz aus fünf LEDs, das anzeigt, ob die Strahlposition auf jeder Diode zentriert ist. Die Schwelle, ab der eine äußere LED aufleuchtet legt dabei der Parameter `LEDTolerance` fest.

Außerdem existiert ein Editiermodus, in dem die Regelparameter aus Abschnitt 3 eingestellt werden können. Dieser ist dann aktiviert, wenn beide Kanäle auf FREEZE stehen. Er wird über die drei Taster MODE, UP und DOWN bedient. Mit MODE kann man durch die einzelnen Parameter schalten. Mit UP und DOWN werden sie verändert. Veränderungen müssen nicht bestätigt werden. Sie werden direkt abgespeichert und verwendet.

Die Parameter werden im internen EEPROM-Datenspeicher (nicht-flüchtig) gespeichert und stehen so auch nach dem Ausschalten der Versorgungsspannung noch zur Verfügung.

Auf der Rückseite der Regel-Box sind die beiden Triggersignale und die Dioden-, sowie SH-Spannung des ersten Quadranten der ersten Diode über Testbuchsen herausgeführt. Mit diesen Signalen kann eine Justage der Elektronik (vor Allem der Triggerung) erfolgen.

5 Das Microcontroller-System

Als Steuer-Computer wird ein ATMEL-Microcontroller ATmega32 der AVR-Serie[2] verwendet. Es handelt sich um einen 8-bit-RISC-Prozessor mit 32kB-Flash-Speicher für das Programm und 2 kB SRAM. Zusätzlich verfügt der ATmega32 über 1k EEPROM Datenspeicher. Da fast alle Befehle im RISC-Kern des Controllers in einem Taktzyklus abgearbeitet werden erreicht er bei 16 MHz externem Takt fast 16 MIPS. Abb. 5 zeigt ein Blockdiagramm des Controllers, das sich auf die, für die Regelung wichtigen, Komponenten beschränkt. In [Atmel 2003] findet sich auf Seite 2 ein vollständiges Diagramm. Hier wurden unter Anderem der AD-Wandler und die I^2C-Schnittstelle weggelassen. Wie man am Blockdiagramm erkennt, sind der Programm- (Flash) und der Hauptspeicher (SRAM) in getrennten Speicher- und Adressbereichen untergebracht (Harvard-Architektur).

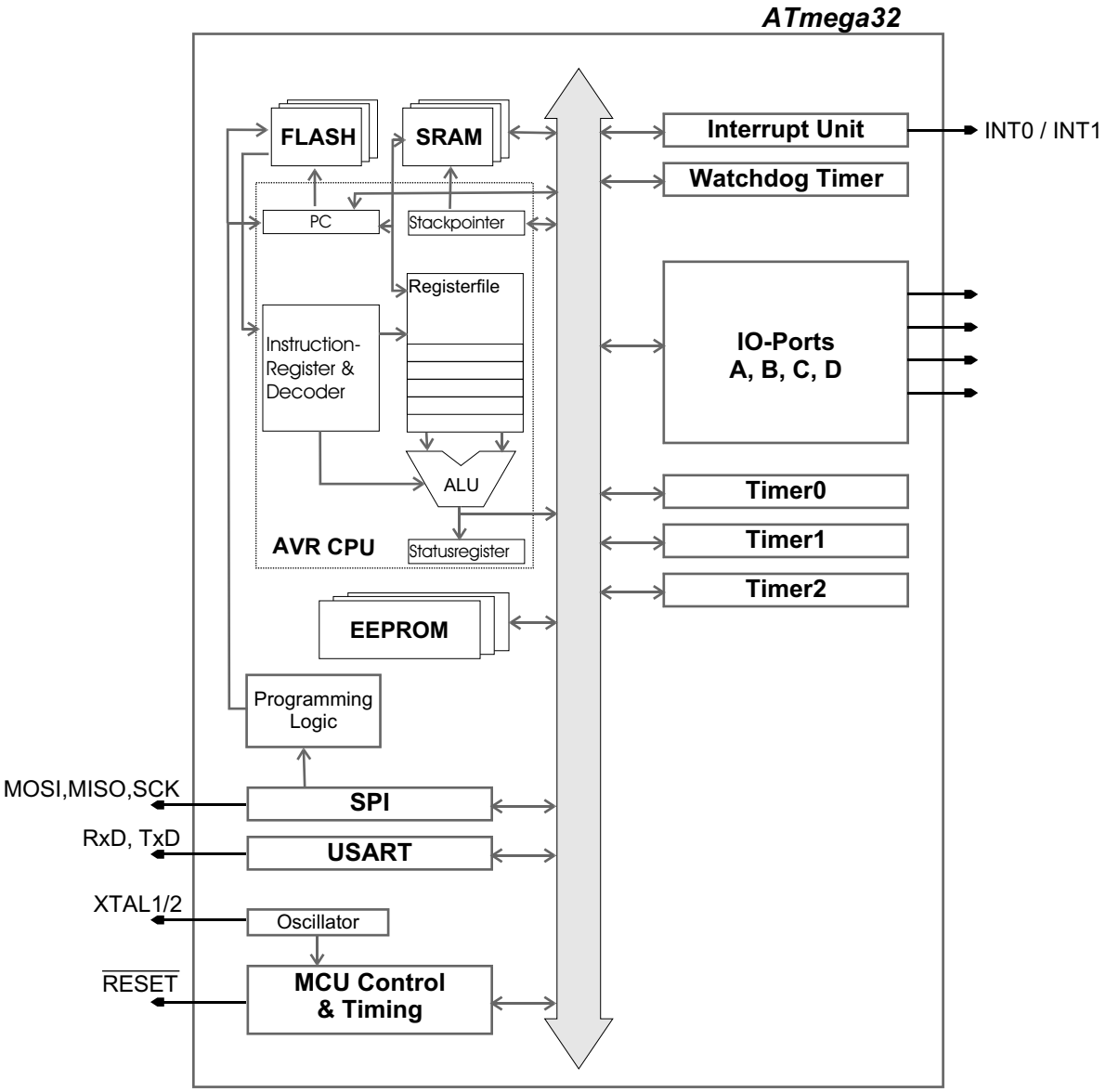

Abb. 10: *Blockdiagramm des AVR-Microcontrollers ATmega32, nach [Atmel 2003]*

Das System besteht neben dem Controller selber noch aus einigen wenigen Peripheriegruppen: Zum einen

[2]Homepage des Herstellers: `http://www.atmel.com/`, Datenblatt [Atmel 2003]

ist eine Programmierschnittstelle integriert, die nach außen hin zugänglich ist und so eine einfache Reprogrammierung der Software erlaubt. Dazu wird das Programm PonyProg 2000[3], eine Freeware von Claudio Lanconelli, verwendet. Zusätzlich ist ein Anschluss für die serielle Schnittstelle (RS-232) vorgesehen, der vor Allem während der Entwicklungsphase zum Debugging diente und von der endgültigen Software-Version nicht genutzt wird. Außerdem ist der Controller natürlich mit der restlichen Hardware der Regelung (ADC, DAC, LCD ...) verbunden. Die Datenwandler benutzen dabei die selbe SPI-Schnittstelle (Serial Peripheral Interface = synchrone serielle Schnittstelle mit 3 Leitungen), die auch zur Programmierung dient. zusätzlich ist zur Adressierung der Datenwandler noch ein 2-zu-4-Dekoder/Demultiplexer (74 139, siehe auch Abschnitt 7) eingebaut. Externer Speicher ist wegen der guten Ausstattung des Chips nicht nötig und kann an den verwendeten Typ auch nicht angeschlossen werden.

6 Aufbau der Software

Dieser Abschnitt soll kurz den Aufbau des gesamten Software-Systems erläutern. Die Quellcodes des Programmes finden sich im Anhang A. Die folgenden Erläuterungen beziehen sich auf diese. Abbildung 6 zeigt ein Flussdiagramm der Software.

Die Software der Regelung selber ist in C geschrieben und kann mit dem GNU-Compiler gcc compiliert werden. Ich habe hierzu die Distribution WinAVR[4] unter Win2000 benutzt. Sie enthält neben dem Compiler noch zusätzliche Tools speziell für AVR-Microcontroller. Alle verwendete Software existiert auch für LINUX.

Nachdem in der Hauptroutine `int main` in `stabilise1.c` die Hardware und die Regelparameter aus dem EEPROM initialisiert wurden, geht das Hauptprogramm in eine Endlosschleife über, in der in jedem Durchlauf geprüft wird, ob sich die Schalterstellungen geändert haben. Trifft dies zu, so passt das Programm seine Parameter entsprechend an, bzw. startet oder beendet den Editiermodus. Ist der Editiermodus nicht aktiv, so wird bei interner Triggerung bei jedem Schleifendurchlauf die Regelroutine aufgerufen. Bei externen Triggerung wird in der Hauptroutine nur sichergestellt, dass der Hardware-Interrupt INT0 aktiviert ist.

Der externe Triggermodus ist also Interrupt-getrieben. Das bedeutet, dass immer wenn von der Triggerelektronik ein Impuls kommt (siehe Abschnitt 2), das Programm unterbrochen und die Regelungsroutine ausgeführt wird. Der Aufruf erfolgt in der Interrupt-Behandlungsroutine `SIGNAL (SIG_INTERRUPT0)`. Befindet sich das Programm im Editormodus, oder in zeitkritischen Routinen (Hardware auslesen ...), so wird dieses Verhalten abgeschaltet. Im Normalbetrieb stören die Unterbrechnungen allerdings nicht, da die Hauptschleife keine zeitkritischen Operationen enthält.

Auch der Editor-Modus ist Interrupt-getrieben. Allerdings wird hier einer der internen Timer (16-bit TIMER1) so konfiguriert, dass er etwa 10 mal pro Sekunde einen Interrupt auslöst (Compare-Interrupt: `SIGNAL (SIG_OUTPUT_COMPARE1A)`). In der Behandlungsroutine werden die Taster abgefragt. Ist der MODE-Taster gedrückt, so wird auf den nächsten Parameter weitergeschaltet. Ist einer der Tasten UP und DOWN gedrückt, so wird der aktuelle Parameter entsprechend geändert. Zum Schluss werden die Parameter angezeigt und im nichtflüchtigen EEPROM-Speicher abgelegt. Für die beiden letztgenannten Tasten wird je eine Zählervariable geführt, die die Anzahl der aufeinanderfolgenden Timer-Interrupts zählt, während derer die Taste gedrückt war. Dies ermöglicht es, durch längeres Drücken der Taste die Verstellung der Werte zu beschleunigen, was vor Allem bei Parametern mit großen Wertebereich eine deutliche Verbesserung der Bedienbarkeit bedeutet.

Die gesamte Software wurde vor Allem auf Geschwindigkeit und Speicherbedarf hin optimiert, wobei letzteres der wesentliche Punkt war. Dies führt an einigen Punkten zu „unschönem Programmierstil“, der aber die Hauptlast vom Hauptspeicher in den Programmspeicher auslagert. Die Software in der aktuellen Version belegt etwa 12 kB von 32 kB Programmspeicher.

[3] PonyProg 2000 für MS Windows und Linux: `http://www.lancos.com/prog.html`
[4] WinAVR-Projekt: `http://sourceforge.net/projects/winavr`

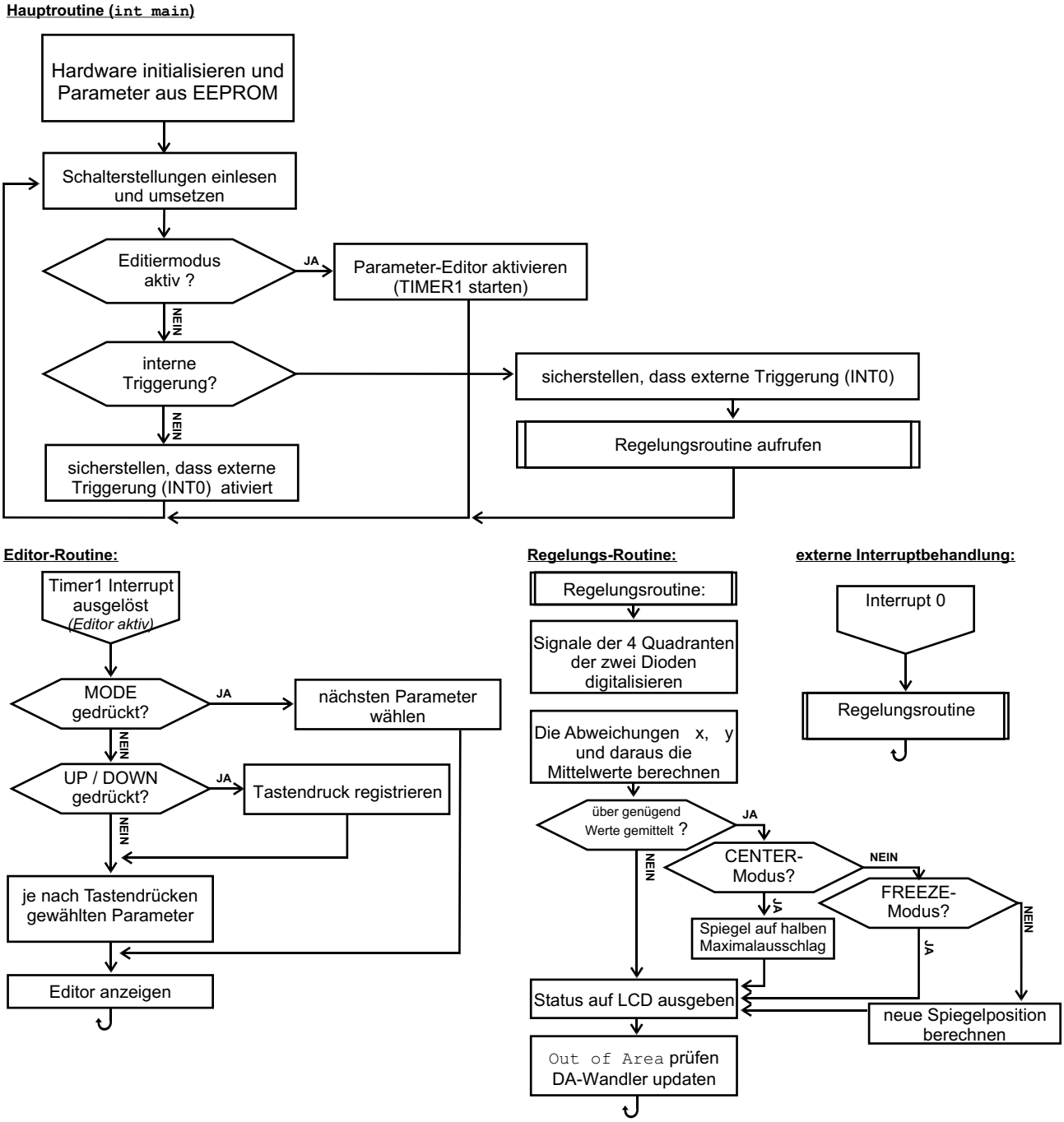

Abb. 11: *Flussdiagramm der Regelsoftware*

7 Beschreibung der Elektronik

Die bereits in Abschnitt 2 skizzierte Elektronik wurde auf vier Platinen verteilt. Nur die Quadrantendiodenplatine (siehe Abschnitt 7.1) befindet sich außerhalb der Regelungsbox; alle anderen sind innerhalb dieser montiert. Alle Platinen wurden als gedruckte Schaltungen gefertigt. Zum Entwurf habe ich das Programm ULTIcap und ULTIboard von Ultimate benutzt. Die folgenden Unterabschnitte erläutern den Aufbau und die Funktion jeder einzelnen Platine.

7.1 Quadrantendioden-Platinen

Die Quadrantendiode ist zusammen mit dem Ladunsgmess-Kondensator und den Vorverstärkern auf zwei kleinen Platinen (ca 2.5 cm^2) untergebracht, die als Stapel aufeinandermontiert wurden. Die Abb 7.1 zeigt den Schaltplan und das Layout. Die Bestückung erfolgte vollständig in SMD-Technik, was die kleinen Abmessungen erst ermöglichte. Die Platinen wurden in Plastik-Gehäuse eingebaut, wie sie in Abb. 2 zu sehen sind. Diese Gehäuse enthalten Gewinde, um sie auf den verwendeten optischen Tischen zu montieren. An den Gehäusen ist ein neunpoliger SUB-D-Stecker angebracht, über den die Elektroik mit Strom versorgt wird und die Signale zur Regel-Box geleitet werden.

Die Unterbringung der Dioden mit Ansteuerelektronik auf einer gesonderten Platine ermöglicht es auch, nachträglich andere Sensoren statt Quadrantendioden einzusetzen. Mit sog. Tetra-Lateraleffekt-Detektoren ist etwa (bei einem wesentlich höheren Preis) eine bessere Ortsauflösung möglich. Hierfür müsste nur eine entsprechende Adapterplatine entwickelt werden, die eine Quadrantendiode vorspiegelt.

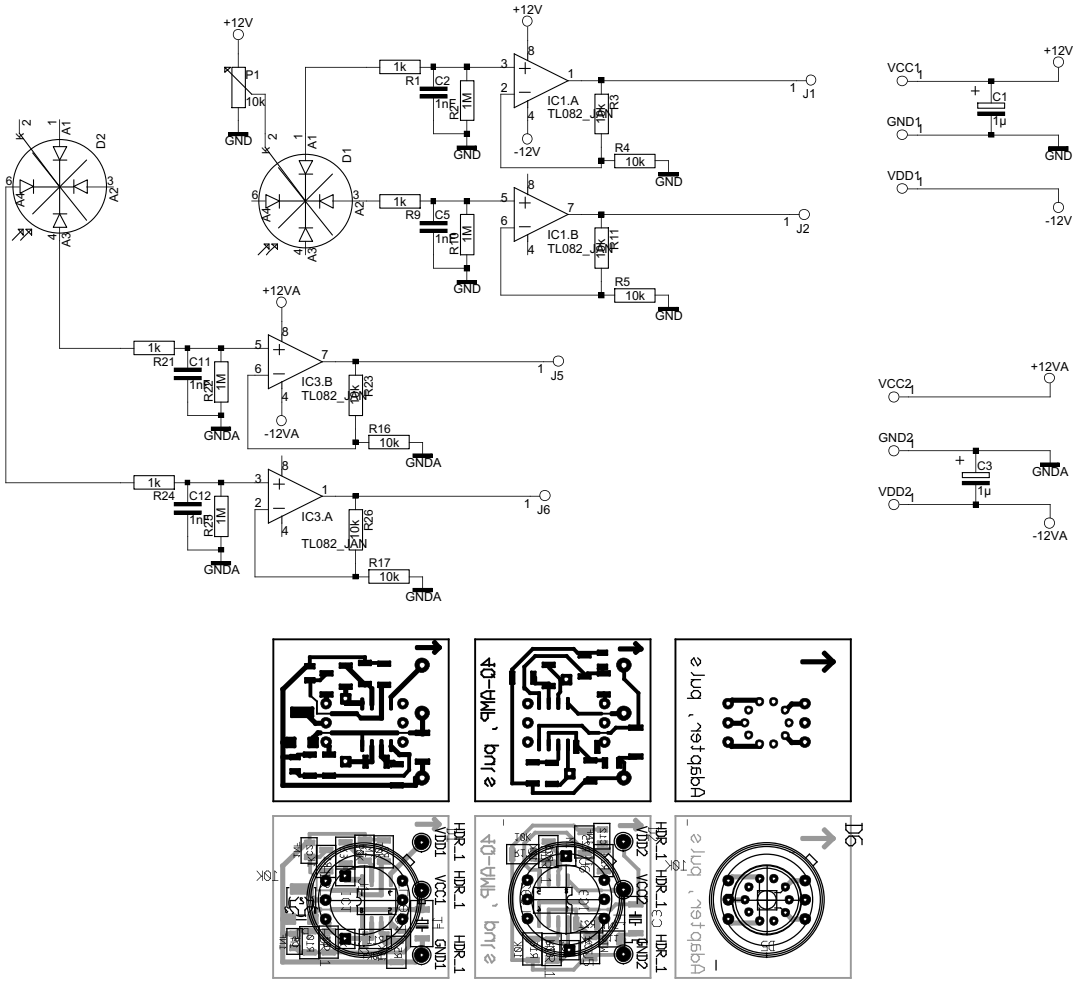

Abb. 12: *Quadrantendioden-Platinen.* ***oben:*** *Schaltplan;* ***unten:*** *Layout und Bestückungsplan (SMD-Bestückungsseite unten, Quadrantendiode oben)*

Abb 7.1 zeigt den genauen Schaltplan einer Diodenplatine. Man erkennt deutlich die Aufteilung in zwei einzelne Platinen. Die Schaltungen für jeden Quadranten sind aber gleich. Die Elektrolyt-Kondensatoren C1 und C3 dienen zur Pufferung der Betriebsspannung. Über das Potentiometer P1 lässt sich der maximale Strom durch die Photodioden einstellen und damit auch die maximale Ausgangsspannung und die Empfindlichkeit

der Schaltung. Die folgenden Erläuterungen beziehen sich auf den Kanal A mit IC1.A. Die Photodiode wird in Sperrichtung betrieben. Über R1 wird der Ladungsmess-Kondensator C2 durch einen Lichtpuls aufgeladen. Über R2 kann er sich wieder entladen. Der Operationsverstärker IC1.A ist als nicht-invertierender Verstärker mit dem Verstärkungsfaktor $A = 2$ ($A = 1 + \frac{R_3}{R4}$) beschaltet. Als Operationsverstärker wurde ein Zweifach-Verstärker TL082 gewählt (siehe [Texas Instruments 1999]). Er weist einen hohen Eingangswiederstand ($R_{ein} = 10^{12}\ \Omega$) und vor Allem eine kurze Anstiegszeit (slew rate) von 13 V/μs auf, die für eine schnelle Reaktion auf ankommende Diodensignale wichtig ist.

Für die Widerstände wurden Typen mit 1% Toleranz verwendet. Die Ladunsgmess-Kondensatoren weisen eine höhere Toleranz von ca. 10% auf. Dies ist aber für den Betrieb weitgehend unerheblich, da die Bauteilwerte gut zeitlich konstant bleiben. Ein Fehler eines Wertes verursacht somit nur eine kleine (zeitlich feste) Verschiebung des Zentrums auf der Diode.

7.2 Spannungsversorgung und Hochspannungsverstärker

Abb 7.2 zeigt den Schaltplan eines Hochspannungsverstärkers und des Netzteiles der Regelungs-Box. Der Hochspannungsverstärker findet sich in vierfacher Ausführung auf der Platine (je zwei pro xy-Piezo). Es handelt sich im Prinzip um die Standard-Beschaltung des Verstärker-ICs PA140 von APEX Microtechnology (siehe [APEX Microtechnology 2002]), ein integrierter Hochspannungsverstärker, der speziell für Piezo-Anwendungen ausgelegt ist. Er wird hier dazu benutzt, das Signal aus der Regelung (0..4 V) auf 0..100 V zu verstärken, um damit die Piezos anzusteuern. Das IC ist als nichtinvertierender Verstärker beschaltet. Mit P8 kann die Verstärkung auf $A = 25$ eingestellt werden. R10 dient der Strombegrenzung, da sich bei kapazitiven Lasten (wie etwa Piezos) zwar keine Dauerströme, dafür aber hohe Impulsströme ergeben können. D5 und D6 kappen zu hohe Spannungen und schützen so das IC. Die Supressor-Dioden D3 und D4 dienen ebenfalls dem Überspannungsschutz. Zur Versorgung der Verstärker ist eine zusätzliche Spannung von 100 V_{DC} nötig, die von einem externen Netzteil geliefert werden muss.

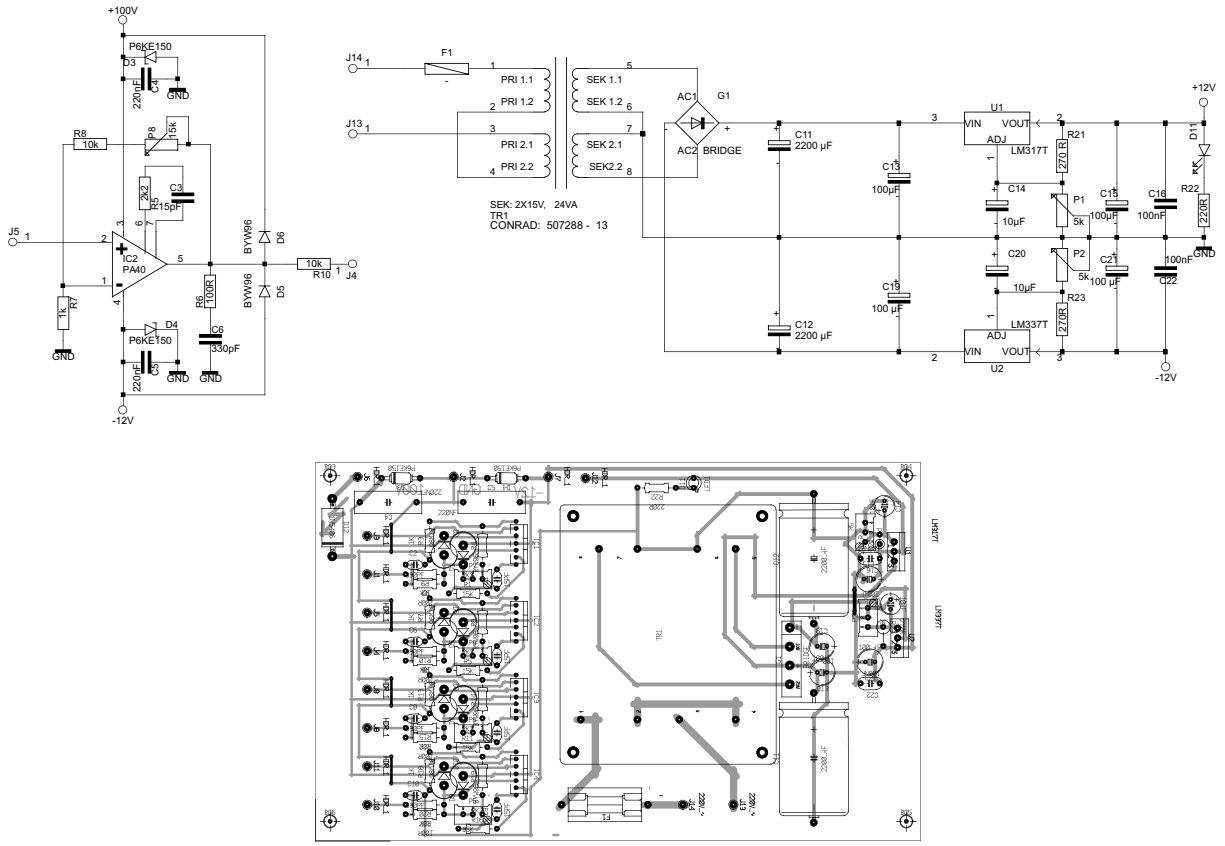

Abb. 13: *Netzteil- und Hochspannungsverstärker-Platine.* ***oben:*** *Schaltplan;* ***unten:*** *Layout und Bestückungsplan (Bestückungsseite oben, 50% Originalgröße)*

Das Netzteil besteht aus einem Trafo, dessen Ausgangsspannung gleichgerichtet und gesiebt wird (G1, C11, C12). Nachgeschaltet sind jeweils ein Spannungsregler vom Typ LM317/337, mit dem die Ausgangsspannung auf den gewünschten Wert eingestellt werden kann. Dabei regelt U1 eine positive und U2 eine negative Spannung.

Beide werden mittels P1 bzw. P2 auf $|U| = 10\,\mathrm{V}$ eingestellt. Diese Spannung dient zur Versorgung einiger Teile der analogen Elektronik und vor Allem der Dioden-Platinen. Auf der AD-DA-Platine (siehe Abschnitt 7.4) wird aus dieser Spannung außerdem noch die Versorgungsspannung für die Logik-Schaltungen ($\pm 5\,\mathrm{V}$) abgeleitet.

Wie sich zeigte, stören Spannungsspitzen von bis zu 50 mV auf der Betriebsspannung nicht. Dies liegt vor Allem am Regelverfahren, das darauf beruht, dass Differenzen der vier Quadrantenspannungen gebildet werden. Dabei fallen etwaige Störsignale, die alle Kanäle gleichermaßen betreffen, heraus.

7.3 Microcontroller-Platine

Auf der Microcontroller-Platine (siehe Abb 7.3) befindet sich als zentrales Bauteil der Steuerrechner ATmega32 (siehe auch Abschnitt 5 bzw. [Atmel 2003]) von ATMEL mit seiner Beschaltung. Der Controller wird vom 16 MHz-Quarz X1 getaktet. Die Taste Ta1 dient zusammen mit R1, D1 und C1 als Reset-Schaltung. Die SUB-D-Buchse PS1PROG bildet zusammen mit R3-R6 , sowie D2, D3 und T1 ein in-circuit-Programmierinterface, über das neue Programme geladen werden können. Das Design orientiert sich dabei am Referenz-Design für die verwendete Freeware-Programmiersoftware PonyProg2000[5]. Mit dem Pegelwandler-IC (0/5V $\leftrightarrow$ $\pm$12V) MAX232 von MAXIM steht außerdem noch eine serielle RS-232-Schnittstelle zur Verfügung, die z.B. zu Debugging-Zwecken dienen kann.

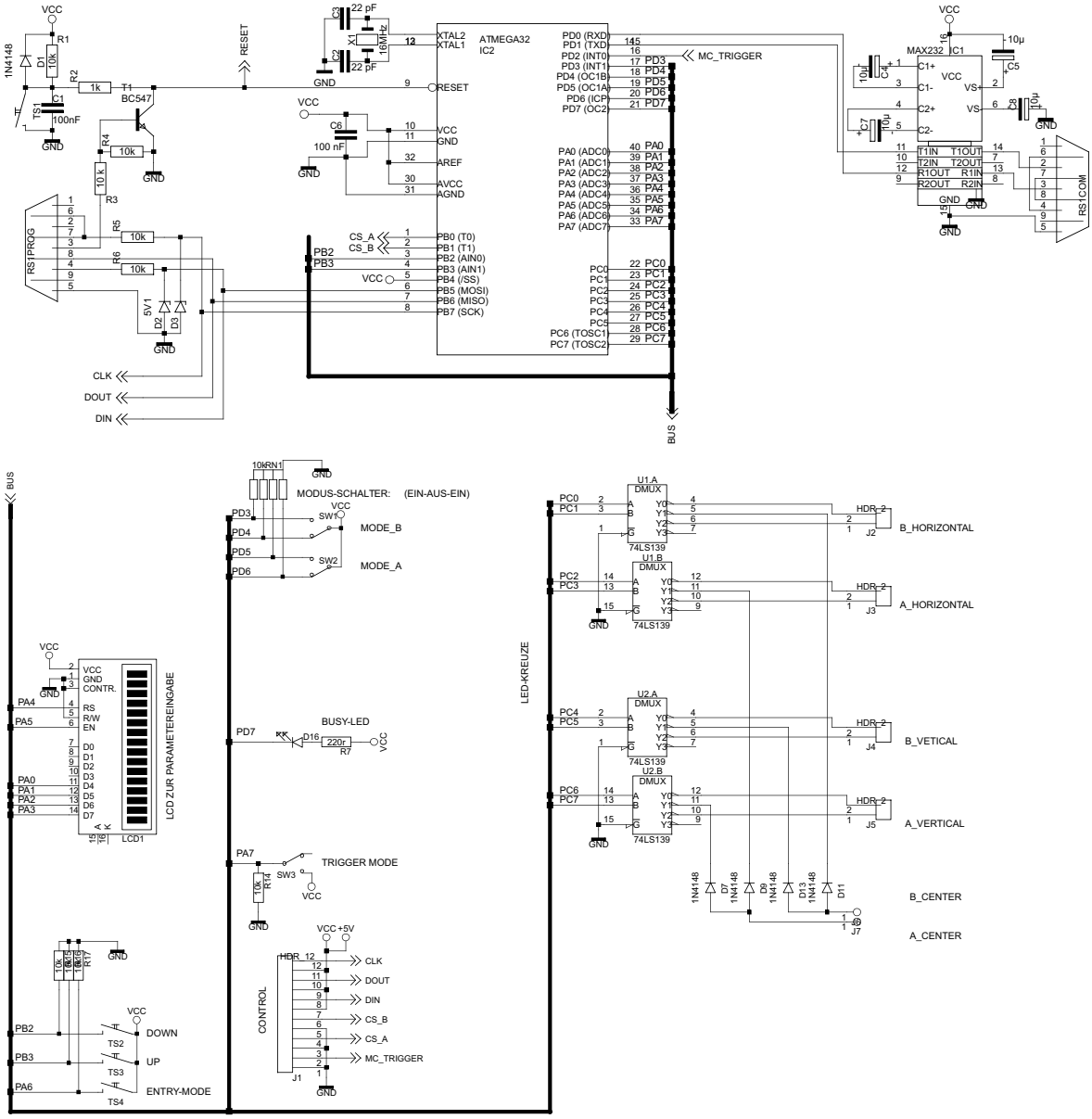

Abb. 14: *Schaltplan der Microcontroller-Platine.*

Der zweite Teil des Schaltplans zeigt, wie die verschiedenen Bedienelemente (Taster, LEDs, LCD ...) an

[5] siehe `http://www.lancos.com/prog.html`

den Microcontroller angeschlossen werden. Eine etwas außergewöhnliche Schaltung wurde für die LED-Kreuze gewählt, um die Anzahl der erforderlichen Pins zu reduzieren: Pro Kreuz werden je zwei halbe 74LS139 (2-zu-4-Dekoder) benutzt, eine für den horizontalen und einer für den vertikalen Wert. Dabei bedeutet eine 1 am Eingang A,B, dass die mittlere LED leuchtet, eine 0 steuert die obere/linke und eine 3 die untere/rechte an. Die zwei Kanäle für die mittlere LED sind über zwei Dioden ODER-Verknüpft, sodass nur eine LED in der Mitte benötigt wird und trotzdem die Ansteuerung der einzelnen Achsen vollkommen gleich ist.

Über den Pfostenstecker J1 wird die Microcontroller-Platine mit der AD/DA-Platine verbunden. Er dient zur Stromversorgung des Controllers und zur Ansteuerung der Bauteile auf der AD-DA-Platine. Dazu werden Signale CLK, DOUT (=MISO), DIN (=MOSI) der SPI-Schnittstelle des ATmega32 nach außen geführt. Die beiden Signale CS_A und CS_B dienen zur Adressierung der ICs. Über MC_Trigger erhält der Microcontroller sein Synchronisationssignal.

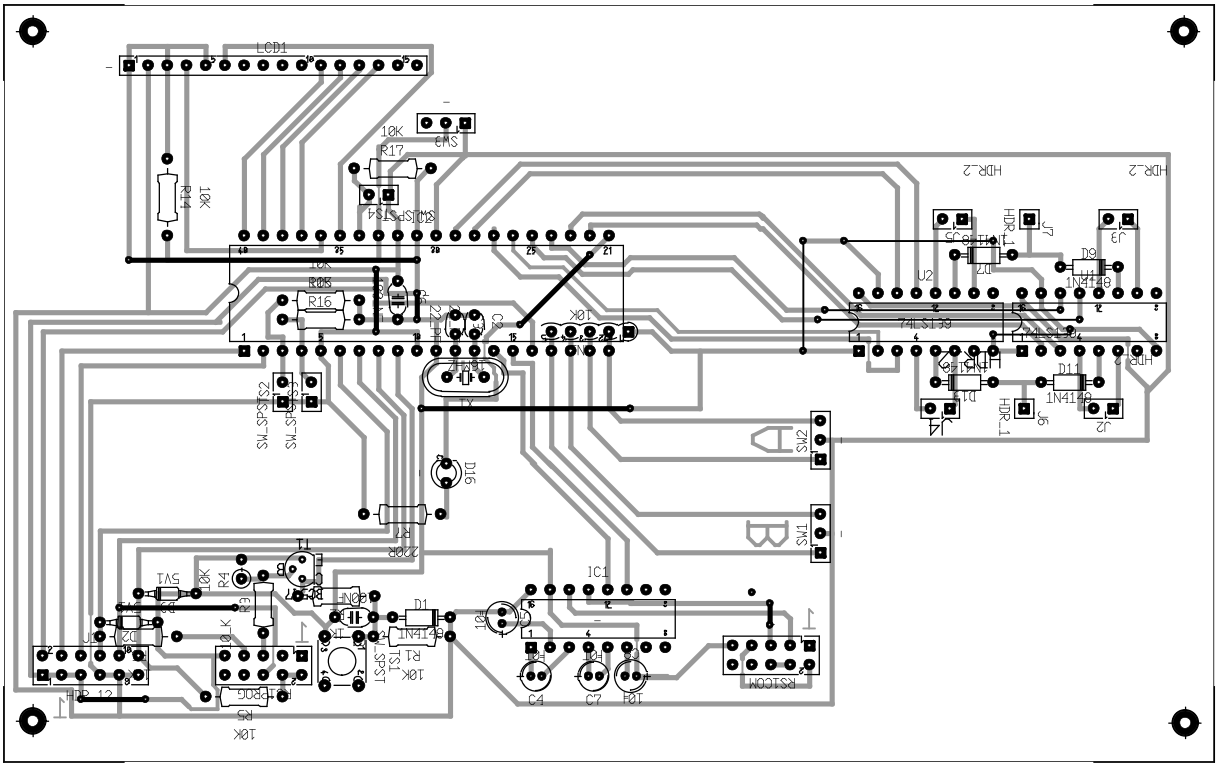

Abb. 15: Platinenlayout der Microcontroller-Platine (Bestückungsseite oben)

7.4 AD/DA-Platine

Die vierte Platine beherbergt den Rest der Elektronik. Dieser umfasst den 8-Kanal-AD-Wandler LT1293 (siehe auch [Linear Technologies 1992]), zwei 2-Kanal-DA-Wandler LTC1446 (siehe auch [Linear Technologies 1996]), die Triggerschaltung, eine ±5 V-Spannungsversorgung und Anpassverstärker für die Ausgänge. Abb. 7.4 zeigt den Schaltplan und Abb. 7.4 den Bestückungsplan.

Die Spannungsversorgung mit den beiden Regler-ICs LM317/337 (U2, U3) ist wie diejenige auf der Netzteil-Platine (siehe Abschnitt 7.2) aufgebaut. Allerdings wird hier die Spannung mit P3 bzw. P4 auf $|U| = 5$ V eingestellt. Die Schottky-Dioden D4 und D5 schützen den AD-Wandler vor Fehlspannungen.

Über HDR12 wird die Microcontroller-Platine (siehe Abschnitt 7.3) angeschlossen. An J1 wird das externe Triggersignal (siehe Abschnitt 2) angeschlossen. Die beiden flankengetriggerten TTL-Monoflops U1.A und U1.B erzeugen daraus die erforderlichen Triggersignale, wie es auch in Abb. 2 dargestellt ist. Über die Potis P1 und P2 lassen sich die Verzögerungen an den Aufbau anpassen.

Der 2-zu-4-Dekoder U4.A erzeugt die Chip-Select-Signale für die AD- und DA-Wandler aus den beiden Signalen CS_A und CS_B. Die DA-Wandler IC1 und IC2 besitzen bereits eine interne Referenzspannung von 4.096 V, weshalb die Beschaltung denkbar einfach ist. An sie wird nur je eine CS-Leitung und der SPI-Bus angeschlossen. Die Verbindung von DOUT ist dabei nicht nötig, weil die Wandler ja keine Signale zurücksenden müssen. Verwendet man die Regelung in Verbindung mit Piezos, so kann die Spannung direkt hinter den DA-Wandlern abgegriffen und zu den Hochspannungsverstärkern geleitet werden. Um eine Ansteuerung von Motoren zu ermöglichen, wurden noch Subtraktionsverstärker mit IC5 und IC6 nachgeschaltet, die die Ausgangsspannung um ein einstellbares Offset (P5) vermindern. So kann z.B. ein Bereich von $-2..+2$ V ausgesteuert werden, womit man Motoren sich vorwärts und rückwärts drehen lassen kann. Dies kann nötig werden, wenn große Spiegel zu verstellen sind. Die Hochspannungsverstärker müssen dann natürlich durch passende Leistungsverstärker ersetzt werden.

Der AD-Wandler IC4 ist mit einer Referenzspannungsquelle LM336 beschaltet. Ist diese nicht gewünscht, so kann die Referenzspannung für die Wandlung mittels J2 auf die Betriebsspannung gesetzt werden. Normalerweise ist dies ausreichend. Mit Hilfe einer Refrenzdiode kann man aber so den Eingangsspannungsbereich herabsetzen und so an unterschiedliche Detektoren anpassen. Die Eingänge des AD-Wandlers sind mit Spannungsteilern (1:2) aus je zwei Widerständen und den Sample-and-Hold-Stufen LF398 beschaltet. Die Spannungsteiler dienen dazu, die maximale Ausgangsspannung der SH-Stufen von ±10 V auf den Bereich ±5 V herabzusetzen, da der AD-Wandler sehr empfindlich auf Überspannungen reagiert. Die maximale Ausgangsspannung der SH-Stufen ist dabei durch deren Betriebsspannung gegeben. Sie muss über der Referenzspannung der AD-Wandlers liegen, um zu gewährleisten, dass der gesamte Wandelbereich (und damit die maximale Auflösung) ausgenutzt wrid.

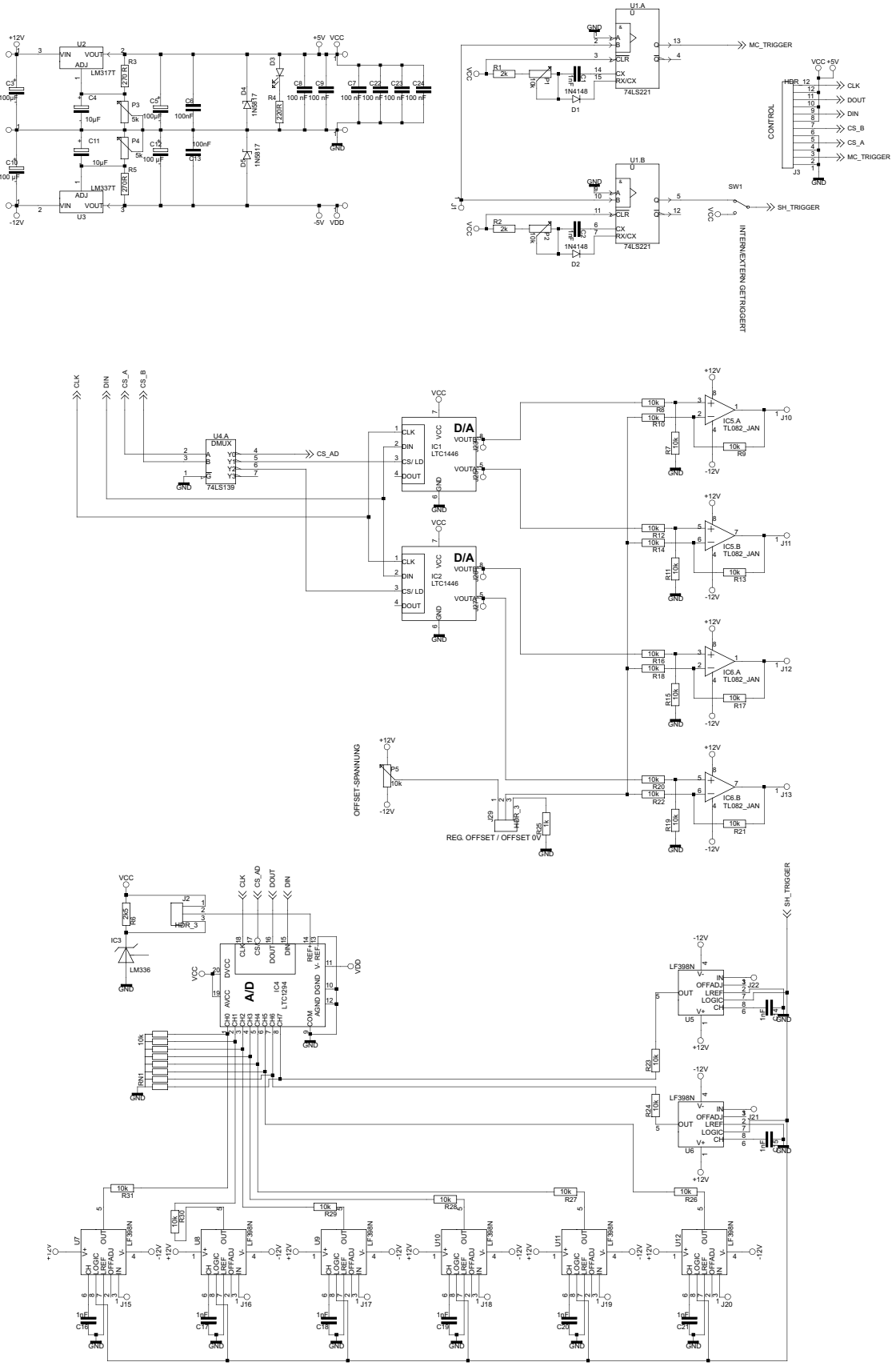

Abb. 16: *Schaltpläne der AD/DA-Platine*

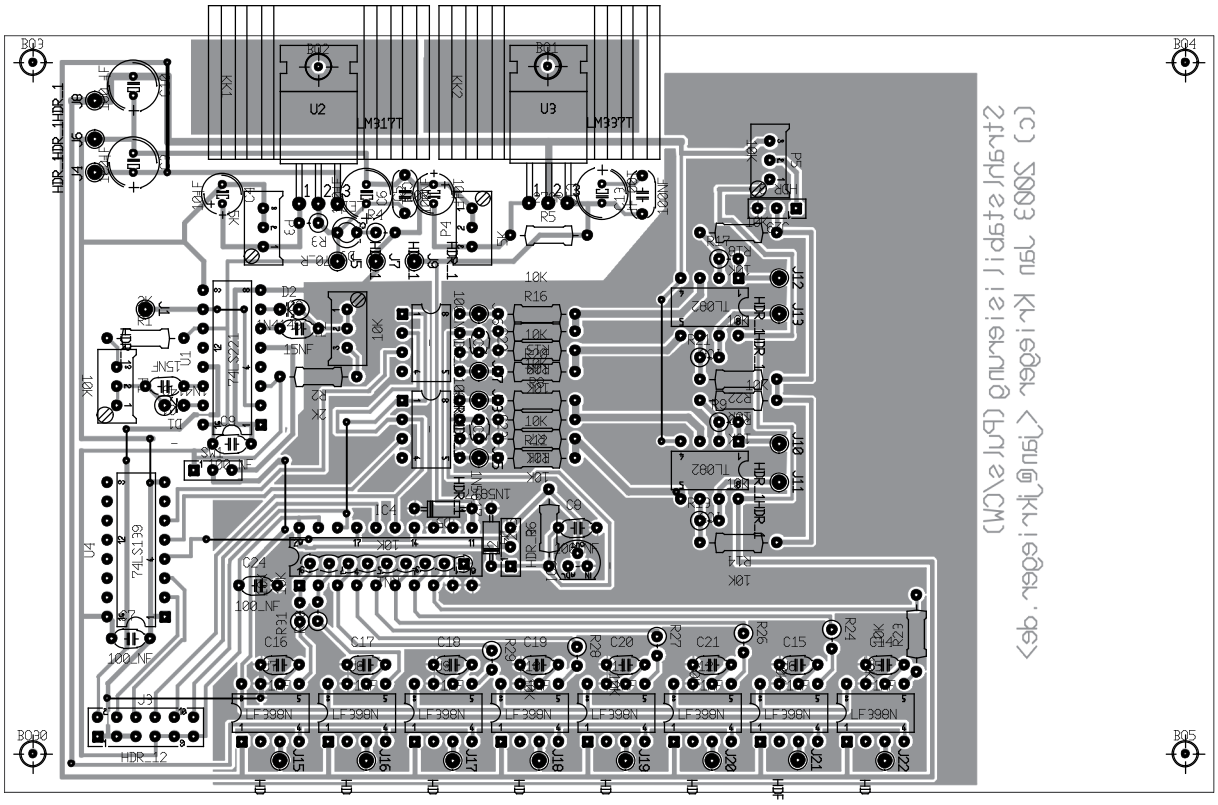

Abb. 17: *AD-DA-Platine: Platinenlayout (Bestückungsseite oben)*

8 Zusammenfassung und Ergebnisse

Es ist mir während meines Praktikums gelungen, ein System zur räumlichen Stabilisierung von Laserstrahlen aus Kurzpulslasern zu entwickeln und in vierfacher Ausfertigung aufzubauen. Um Probleme mit den kurzen Laserpulsen (hier: einige Femto- bis Nanosekunden, bei zehn Schüssen pro Sekunde) zu vermeiden, wurde hierzu ein Ansatz unter Verwendung eines 8-bit-Microcontrollers gewählt. Der gewählte Ansatz funktioniert, im Gegensatz zu analogen Reglern auch mit den hier auftretenden Signal-Pause-Verhältnissen von $10^{-7}..10^{-10}$. Die Regelung kann mittels zweier fester Raumpunkte, auf die geregelt wird, einen Laserstrahl raumfest halten. Dies ist z.B. nötig, wenn ein Experiment über mehrere Tische verteilt ist, oder um langsame Temperaturdriften des optischen Aufbaus auszugleichen. In dem Fall am MPQ ging es darum, einen 10Hz-Pump-Laserstrahl raumfest in einen Verstärker-Kristall einzukoppeln.

Die entwickelte Elektronik schafft es, die Position des Strahles ausreichend konstant zu halten. Zur Beurteilung wurde der Strahl unter Anderem über einige Stunden mit Hilfe einer CCD-Kamera, die ein Bild (des aufgeweiteten) Strahles liefert, aufgenommen. Die Korrektur längerer Driften wurde dadurch gezeigt, dass der Strahl künstlich um etwa einen halben Strahldurchmesser verschoben wurde. Diese Störung konnte die Regelung innerhalb etwa $30 - 60$ s vollständig ausgleichen. Danach konnte auf dem Bild der CCD-Kamera keine merkliche Abweichung von der Idealposition mehr gesehen werden. Die Stabilität des Nullpunktes über einige Tage wurde über eine zusätzliche Quadrantendiode kontrolliert, die mit einer Auswerteelektronik an ein Oszilloskop angeschlossen war. Dabei wird auf dem Oszilloskop im x-y-Betrieb eine Paar von Spannungen angezeigt, die die Abweichungen vom Nullpunkt der Diode angeben. Die Nullpunkt wurde mit obigem Aufbau, mit einer durch die Schuss-zu-Schuss-Schwankungen begrenzten Genauigkeit wieder eingeregelt, was den Anforderungen an die Regelung entspricht.

Die Regelung kann diese Schuss-zu-Schuss-Schwankungen des Lasers nicht ausgleichen, da diese statistisch um die Nulllage verteilt auftreten. Sie werden von der Regel-Software durch eine Mittelung über mehrere Schüsse aus der Regelgröße eliminiert. Im Anwendungsfall wurde so eine Nachstellrate von $1 - 2$ Hz bei einer Schussrate von 10 Hz erreicht. Die Regelung kann prinzipiell bis zu einer Schussrate von ca. 200 Hz eingesetzt werden. Dies beschränkt die Anwendung der Regelung auf Störungen, die sich im Bereich von einigen Sekunden oder länger auswirken.

Beim Versuch, die Regelung auch für kontinuierliche Laser (CW-Laser) verwendbar zu machen stieß das Design an seine Grenzen. Hier traten zum einen im optischen Aufbau Probleme durch den sehr geringen Strahldurchmesser (ca. 1 mm), im Verhältnis zum recht großen Dioden-Durchmesser (ca. 10 mm) auf. Zum anderen ist es bei CW-Lasern interessant, schnelle Schwankungen z.B. durch Schwingungen des Aufbaus auszugleichen, da sich diese aber in Bereichen von einigen Herz bis zu einigen Kilo-Herz bewegen war es mit der freilaufenden Regelung (Regelfrequenz: ca. 200 Hz) nicht möglich, eine Verbesserung zu erzielen. Hier sind analoge PI-Regeler weit besser geeignet.

A Sourcecode

A.1 Makefile für WinAVR/gcc

Das Makefile basiert auf dem Beispiel-Makefile von WinAVR. Es wurden vor Allem die Pfade angepasst.

```
# WinAVR Sample makefile written by Eric B. Weddington, et al.
# Released to the Public Domain
# Please read the make user manual!
#
# On command line:
# make all = Make software.
# make clean = Clean out built project files.
# make coff = Convert ELF to COFF using objtool.
#
# To rebuild project do "make clean" then "make all".

# MCU name
MCU = atmega32

# Output format. (can be srec, ihex)
FORMAT = ihex

# Target file name (without extension).
TARGET = stabilise1

# Optimization level (can be 0, 1, 2, 3, s)
# (Note: 3 is not always the best optimization level. See avr-libc FAQ)
OPT = 3

# List C source files here. (C dependencies are automatically generated.)
SRC = $(TARGET).c \
lcd.c

# Optional compiler flags.
CFLAGS = -O$(OPT) -Wall -Wstrict-prototypes -Wa,-ahlms=$(<:.c=.lst)

# Optional assembler flags.
ASFLAGS = -Wa,-ahlms=$(<:.S=.lst),-gstabs

# Optional linker flags.
LDFLAGS = -Wl,-Map=$(TARGET).map,--cref

# ----------------------------------------------------------------------------
# Define directories, if needed.
DIRAVR = e:/programme/avrgcc
DIRAVRBIN = $(DIRAVR)/bin
DIRAVRUTILS = $(DIRAVR)/utils/bin
DIRINC = .
DIRLIB = $(DIRAVR)/avr/lib

# Define programs and commands.
SHELL = sh
CC = avr-gcc
OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
REMOVE = rm -f
COPY = cp
ELFCOFF = objtool
HEXSIZE = avr-size --target=$(FORMAT) $(TARGET).hex
ELFSIZE = avr-size -A $(TARGET).elf
FINISH = echo Errors: none
BEGIN = echo -------- begin --------
END = echo -------- end --------

# Define all object files.
OBJ = $(SRC:.c=.o) $(ASRC:.S=.o)

# Define all listing files.
LST = $(ASRC:.S=.lst) $(SRC:.c=.lst)

# Combine all necessary flags and optional flags. Add target processor to flags.
ALL_CFLAGS = -mmcu=$(MCU) -I. $(CFLAGS)
ALL_ASFLAGS = -mmcu=$(MCU) -I. -x assembler-with-cpp $(ASFLAGS)

# Default target.
all: begin gccversion sizebefore $(TARGET).elf $(TARGET).hex $(TARGET).eep \
$(TARGET).lss sizeafter finished end

# Eye candy.
begin:
	@$(BEGIN)
finished:
	@$(FINISH)
end:
	@$(END)

# Display size of file.
sizebefore:
	@if [ -f $(TARGET).elf ]; then echo Size before:; $(ELFSIZE);fi
sizeafter:
	@if [ -f $(TARGET).elf ]; then echo Size after:; $(ELFSIZE);fi

# Display compiler version information.
gccversion :
	$(CC) --version

# Target: Convert ELF to COFF for use in debugging / simulating in AVR Studio.
coff: $(TARGET).cof end
%.cof: %.elf
	$(ELFCOFF) loadelf $< mapfile $*.map writecof $@

# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
```

```
        $(OBJCOPY) -O $(FORMAT) -R .eeprom $< $@
%.eep: %.elf
        -$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O $(FORMAT) $< $@

# Create extended listing file from ELF output file.
%.lss: %.elf
        $(OBJDUMP) -h -S $< > $@

# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(OBJ)
%.elf: $(OBJ)
        $(CC) $(ALL_CFLAGS) $(OBJ) --output $@ $(LDFLAGS)

# Compile: create object files from C source files.
%.o : %.c
        $(CC) -c $(ALL_CFLAGS) $< -o $@

# Compile: create assembler files from C source files.
%.s : %.c
        $(CC) -S $(ALL_CFLAGS) $< -o $@

# Assemble: create object files from assembler source files.
%.o : %.S
        $(CC) -c $(ALL_ASFLAGS) $< -o $@

# Target: clean project.
clean: begin clean_list finished end
clean_list :
        $(REMOVE) $(TARGET).hex
        $(REMOVE) $(TARGET).eep
        $(REMOVE) $(TARGET).obj
        $(REMOVE) $(TARGET).cof
        $(REMOVE) $(TARGET).elf
        $(REMOVE) $(TARGET).map
        $(REMOVE) $(TARGET).obj
        $(REMOVE) $(TARGET).a90
        $(REMOVE) $(TARGET).sym
        $(REMOVE) $(TARGET).lnk
        $(REMOVE) $(TARGET).lss
        $(REMOVE) $(OBJ)
        $(REMOVE) $(LST)
        $(REMOVE) $(SRC:.c=.s)
        $(REMOVE) $(SRC:.c=.d)

# Automatically generate C source code dependencies.
# (Code originally taken from the GNU make user manual and modified (See README.txt Credits).)
# Note that this will work with sh (bash) and sed that is shipped with WinAVR (see the SHELL variable defined above).
# This may not work with other shells or other seds.
%.d: %.c
        set -e; $(CC) -MM $(ALL_CFLAGS) $< \
        | sed 's,\(.*\)\.o [ :]*,\1.o \1.d : ,g' > $@; \
        [ -s $@ ] || rm -f $@

# Remove the '-' if you want to see the dependency files generated.
-include $(SRC:.c=.d)

# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion coff clean clean_list
```

A.2 Tool-Funktionen (alignment.c)

```
/*
  Hardware-Treiber fr Laser-Stabilisierung
  (c) 2003 by Jan Krieger
  jan@jkrieger.de, www.jkrieger.de

  Zielsystem: Atmel ATmega32, ohne externes RAM !!!
              Quarz: 8/16 MHz !!!
  Compiler:   gcc 3.3.1
*/

#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/io.h>
#include "lcd.h"


// QFreq gibt die Quarz-Frequenz in MHz an
#define QFreq 16




typedef enum {over, ok, under} ledstate;
typedef enum {AD, DA1, DA2, none} dataconverter;
typedef enum {regulate, freeze, center} reg_mode; //Modus der Regelung
typedef enum {cm_zero=0, cm_center=1, cm_high=2} center_mode; //Einstellungen fr den Center-Modus der Regelung
typedef enum {external, internal} trigger_mode; //Trigger-Modus
typedef enum {up, left, right, down} diode_direction; //gibt die Richtung an, in der die Diode eingebaut ist
                                                       // up ist standard (Pfeil auf Diodenplatinen nach oben!)
/*
  InitHardware() initialisiert den Controller
*/
void InitHardware(void) {
  /* PA0-PA5: Ausgnge fr LCD
     PA6:     Eingang, Bedientaster: UP
     PA7:     Eingang, Trigger-Mode-Schalter (1: interner Trigger/CW, 0: externener Trigger, puls)


     PB0, PB1: Ausgnge fr Adress-Demultiplexer
     PB2-PB3: Eingnge fr Bedientaster (1: gedrckt , 0 losgelassen)
```

```
                PB2: ENTRY-MODE
                PB3: DOWN
   PB4-PB7: SPI-Schnittstelle zu Datenwandlern.
                PB4: /SS (auf VCC gezogen)
                PB5: MOSI, Ausgang
                PB6: MISO, Eingang
                PB7: SCK, Clock-Ausgang


   PortC:    Ausgnge  fr  LED-Kreuze an 74139


   PD0, PD1: RS-232-Schnittstelle (PD0: Eingang/RxD, PD1: Ausgang/TxD)
   PD2:       INT0/Trigger-Eingang
   PD3/4:     Eingnge , Modus, Kanal B
   PD5/6:     Eingnge , Modus, Kanal A
   PD7:       Ausgang, Busy-LED

   Portprogrammierung: Datenblatt S. 47ff
*/

//Datenrichtungen setzen : 1: Ausgang, 0: Eingang
outp(0x3F, DDRA); // 00111111 = 0x3F
outp(0xB3, DDRB); // 10110011 = 0xB3
outp(0xFF, DDRC); // 11111111 = 0xFF
outp(0x82, DDRD); // 10000010 = 0x82

/*
   externer INT0 aktivieren (Trigger-Signal) siehe Datenblatt S. 64

   der Interrupt wird auf die fallen Flanke des Signals an PD2/INT0 getriggert:

   MCUCR: ISC11=0, ISC10=0, ISC01=1, ISC00=0 => MCUCR = 00000010 = 0x02
   GICR:    INT0=1, Rest: 0                          => GICR = 01000000 = 0x40

*/
MCUCR=0x02;
GICR=0x40;


/*
  SPI-Schnittstelle aktivieren (Serial Peripheral Interface im datenblatt ab S. 130)
   fr  Zugriff auf AD und DA-Wandler. Es handelt sich um eine synchrone 3-Leitungs-Scnittstelle (2*Daten, 1* Clock)


  SPCR:
    SPIE=0;          kein Interrupt
    SPE=1;           SPI enabled
    DOWRD=0;         MSB-first dataorder
    MSTR=1:          Master-Mode
    CPOL=0:          non-inverted clock
    CPHA=0:          sample on leading edge
    SPR0=1, SPR1=1: f_osc/128
        ==> SPCR = 01010011 = 0x53

  SPSR:
    SPI2X=0:          single-speed mode (see also SPR0/1)
        ==> SPSR=0
*/
SPCR=0x53;
SPSR=0;


/*
  RS-232-SChnittstelle aktivieren und konfigurieren (USART im Datenblatt ab S. 138)

     Modus: 9600 baud, 8bit, 1stop, noparity

  9600 baud => UBRR=51 bei 8MHz
               UBRR=103 bei 16 MHz
  57.6k              8 bei 8MHz
                     16 bei 16MHz

  BAUD gibt die Baud-Rate der seriellen RS-232-Schnittstelle an
*/
UBRRH = 0;
#if QFreq==8
  UBRRL = 8;
#else
  UBRRL = 16;
#endif

/*
  UCSRA
    U2X=0     single-speed mode
    MPCM=0 no multi-processor comunication mode
      ==> UCSRA = 0

  UCSRB
    RXCIE=0 no RX Complete interrupt
    TXCIE=0 no Tx Complete interrupt
    UDRIE=0 no data register empty interrupt
    RXEN=1 Receiver enabled
    TXEN=1 Transmitter enabled
    UCSZ2=0 8-bit characters (see also UCSZ1/0 and Table 66 on page 161 in datasheet)
    RXB8=0
    TXB8=0  additional 9th data bits
      ==> UCSRB = 00011000 = 0x18

  UCSRC
    URSEL=1 write to UCRSRC register
    UMSEL=0 asynchronous mode
    UPM1=0  | no parity
    UPM0=0  |
    USBS=0  1 stop bit
    UCSZ1=1 | 8 bit character (see also UCSZ2 in UCSRB register)
    UCSZ0=1 |
```

```
      UCPOL=0 ignored in asynchronous mode
        ==> UCSRC = 10000110 = 0x86
  */
  UCSRA=0;
  UCSRB=0x18;
  UCSRC=0x86;

  /* Enable receiver and transmitter */
  UCSRB = (1<<RXEN)|(1<<TXEN);
  /* Set frame format: 8data, 2stop bit */
  UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);


  /*
     Initialisierung von Timer1 zur Interruptgesteuerten Tastenabfrage

    Der Timer luft bei 16MHz mit 16MHz/1024 = 15,625 kHz
    ==> zhlen von 0 bis 1563. Dort Compare-Match, ergibt etwa 10 Overflow-Interruptr pro Sekunde
    Fr den Editiermodus wird CompareA benutzt, im Center-Modus CompareB

    TCCR1A = 0 keine PWM oder Ausgangs-Modi
    TCCR1B
      ICNC1=0
      ICES1=0
      Bit5=0
      WGM13=0
      WGM12=0
      CS12=1 |
      CS11=0 | Clk/1024 (Vorteiler)
      CS10=1 |
        ==> TCCR1B= 00000101= 0x05

    TIMSK:
      TOIE1=1/0 => aktiviert/deaktiviert den T/C1-Overflow Interrupr
  */

  TIMSK=0; //alle Timer-Interrupts disabled T/C interrupt mask register f
  TCNT1=0;
  TCCR1A=0;
  TCCR1B=0x05;
  OCR1A=1563;
  OCR1B=1563;
  sei (); //alle Interrupt aktivieren (cli () deaktiviert sie)
}


/*
  procedures zum aktivieren/deaktivieren der Interrupt-Steuerung (an/abschalten von INT0)
  siehe auch InitHardware-Funktion. DOrt sind die Register erklrt .
*/
#define enableINT0() cli(); MCUCR=0x02; GICR=0x40; sei();
#define disableINT0() cli(); MCUCR=0; GICR=0; sei();

//Abfrage der Eingabetasten:
#define button_UP() bit_is_set(PINB, PINB2)
#define button_DOWN() bit_is_set(PINB, PINB3)
#define button_MODE() bit_is_set(PINA, PINA6)

//Timer/Counter 1 enable/disable
#define timer1_enable() cli(); sbi(TIMSK, OCIE1A); TCNT1=0; OCR1A=1563; sei();
#define timer1_reset() TCNT1=0;
#define timer1_disable() cli(); cbi(TIMSK, OCIE1A); sei();

#define timer2_enable() cli(); sbi(TIMSK, OCIE1B); TCNT1=0; OCR1B=1563; sei();
#define timer2_reset() TCNT1=0;
#define timer2_disable() cli(); cbi(TIMSK, OCIE1B); sei();

// gibt ein Zeichen auf LCD aus
// \n setzt den Cursor in die erste Zeile
// \r setzt den Cursor in die zweite Zeile
int LCDwrite(char data) {
  if (data=='\n'){
    lcd_cls ();
    lcd_goto(0x00);
  } else {
    if (data=='\n'){ lcd_goto(0x40); }
    else { lcd_putch(data); }
  }
  return 0;                          // 0 zurckgeben (erfolgreiches versenden)
}

// gibt ein Byte ber RS-232 aus
int RS232write(char data) {
   loop_until_bit_is_set (UCSRA, UDRE); //warten bis Transmitter bereit.
  UDR=data;                          //Daten ausgeben
  return 0;                          // 0 zurckgeben (erfolgreiches versenden)
}

// empfaengt ein Byte ber RS-232
int RS232read(void) {
  /* Wait for data to be received */
  while ( !(UCSRA & (1<<RXC)) );
  /* Get and return received data from buffer */
  return UDR;
}

//setze Busy-LEDs (0: aus; !=0: an)
void SetBusyLED(unsigned char led) {
  if (led==0) { sbi(PORTD, PD7); } else { cbi(PORTD, PD7); }
}

// liesst den am Modus-Schalter fpr den entsprechenden Kanal eingestelten Modus aus
reg_mode getMode(unsigned char channel) {
  if (channel==0) {
    if (( bit_is_clear (PIND, PIND5))&&(bit_is_clear(PIND, PIND6))) return freeze;
    if (( bit_is_set (PIND, PIND5))&&(bit_is_clear(PIND, PIND6))) return regulate;
    return center;
```

```
  }else {
    if (( bit_is_clear (PIND, PIND3))&&(bit_is_clear(PIND, PIND4))) return freeze;
    if (( bit_is_set (PIND, PIND3))&&(bit_is_clear(PIND, PIND4))) return regulate;
    return center;
  }
}

// gibt ein Zeichen zurck , das den bergebenen Modus darstellt
// freeze : 'F',      regulate 'R',      center 'C'
char ModeToC( reg_mode m) {
  if (m==freeze) return 'F';
  if (m==regulate) return 'R';
  if (m==center) return 'C';
  return 'F';
}

// gibt ein Zeichen zurck , das den bergebenen Center-Modus darstellt
char CModeToC(center_mode m) {
  if (m==cm_center) return 'C';
  if (m==cm_zero) return '0';
  if (m==cm_high) return 'H';
  return 'C';
}

//gibt den bergebenen Wert auf dem LCD-Display aus
void printDD(diode_direction d) {
  if (d==up) lcd_putstr("up");
  if (d==down) lcd_putstr("down");
  if (d==left) lcd_putstr(" left");
  if (d==right) lcd_putstr("right");
}

trigger_mode getTriggerMode(void) {
  if ( bit_is_set (PINA, PINA7)) return internal;
  return external;
}

// gibt 'true' aus, wenn die bergebene Variable >0, sonst 'false' auf dem LCD-Display aus
void printbool(unsigned char value) {
  if (value==0) lcd_putstr("false"); else lcd_putstr("true");
}

#define otherchannel(x) ((x==0)?1:0)

// Setzen der LED-Kreuze
void SetLED(ledstate state1x, ledstate state1y, ledstate state2x, ledstate state2y) {
  outp(0xFF, DDRC);
  PORTC=0x00;
  switch (state1x) {
    case over:   break;
    case ok:     sbi(PORTC, PC2); break;
    case under:  sbi(PORTC, PC3); break;
  }
  switch (state2x) {
    case over:   break;
    case ok:     sbi(PORTC, PC0); break;
    case under:  sbi(PORTC, PC1); break;
  }
  switch (state1y) {
    case over:   break;
    case ok:     sbi(PORTC, PC6); break;
    case under:  sbi(PORTC, PC7); break;
  }
  switch (state2y) {
    case over:   break;
    case ok:     sbi(PORTC, PC4); break;
    case under:  sbi(PORTC, PC5); break;
  }
}

//Setzen der CS/-Leitung fr den AD und die DA Wandler (Auswahl eines Bausteins)
// ber 74HC139 (2zu4-Dekoder)
void SetAdress(dataconverter ic) {
  uint16_t i;
  switch (ic) {
    case AD:  cbi(PORTB, PB0); cbi(PORTB, PB1); break;
    case DA1: cbi(PORTB, PB0); sbi(PORTB, PB1); break;
    case DA2: sbi(PORTB, PB0); cbi(PORTB, PB1); break;
    case none: sbi(PORTB, PB0); sbi(PORTB, PB1); break;
  }
  for (i=0; i<2000; i++); //Verzoegerung
}

//Ausgeben von <data> und gleichzeitiges Lesen an SPI-Schnittstelle
unsigned char SPIread_write(unsigned char data){
  SPDR=data;                  //Daten an SChnittstelle senden
  while(!(SPSR & (1<<SPIF))); //warten bis Transmission erfolgt
  return inp(SPDR);           //empfangenes Byte zurcklesen
}

//Ausgeben eines Bytes ber die SPI-Scnittstelle
void SPIwrite(unsigned char data) {
  SPDR=data;                  //Daten an SChnittstelle senden
  while(!(SPSR & (1<<SPIF))); //warten bis Transmission erfolgt
}

//Ausgeben von 0 und gleichzeitiges Lesen an der SPI-Schnittstelle
unsigned char SPIread(){
  SPDR=0;                     //0 senden
  while(!(SPSR & (1<<SPIF))); //warten bis Transmission erfolgt
  return inp(SPDR);           //empfangenes Byte zurcklesen
}

//setzen der vier DA-Kanle (LTC1446)
void SetDA(uint16_t value1, uint16_t value2, uint16_t value3, uint16_t value4) {
  SetAdress(DA1); //Adresse setzen
  SPIwrite((unsigned char)((0x0FF0 & value1)>>4)); //Daten uebertragen ...
```

```
  SPIwrite((unsigned char)(((0x000F & value1)<<4)|((0x0F00 & value2)>>8)));
  SPIwrite((unsigned char)(0x00FF & value2));

  SetAdress(DA2); //ADresse des zweiten DA setzen
  SPIwrite((unsigned char)((0x0FF0 & value3)>>4)); // Daten ausgeben
  SPIwrite((unsigned char)(((0x000F & value3)<<4)|((0x0F00 & value4)>>8)));
  SPIwrite((unsigned char)(0x00FF & value4));

  SetAdress(none); //kein Device adessieren
}

/* Auslesen der Kanals <channel> (in [0..7]) vom AD-Wandler LTC1294 */
unsigned int ReadAD(unsigned char channel) {
  unsigned char send1, send2, send3;
  unsigned int recv1, recv2, recv3;
  unsigned int res;
  unsigned char channelcode[8]={8, 12, 9, 13, 10, 14, 11, 15};

  SetAdress(AD);
  // Programmierung, wie fr 68HC11 in Datenblatt beschrieben (AD-Wandler Datenblatt S. 15)
  //      Schnittstelle des AD-Wandlers im datenblatt ab Seite 10 "Applications Information"
  send1 = channelcode[channel] + 16; // 0 0 0 START1 4-bit-Adresse
  send2 = 224; // UNI1 MSBF1 PS1 0 0 0 0 0
  send3 = 0; //Dummy-Datenwort 0

  //send1=0xFF; send2=0xFF; send3=0xFF; //Test-Ausgabe, simuliert Testschaltung Fig. 23 aus Datenblatt zu LTC1294
  recv1 = SPIread_write(send1);
  recv2 = SPIread_write(send2);
  recv3 = SPIread_write(send3);

  //send1=0xFF; send2=0xFF; send3=0xFF; //Test-Ausgabe, simuliert Testschaltung Fig. 23 aus Datenblatt zu LTC1294
  /*recv1 = SPIread_write(channelcode[channel] + 16); // 0 0 0 START1 4-bit-Adresse
  recv2 = SPIread_write(224);                          // UNI1 MSBF1 PS1 0 0 0 0 0
  recv3 = SPIread_write(0); */                           //Dummy-Datenwort 0

  res=(recv2<<8)+recv3;

  SetAdress(none);
  return res;
}
```

A.3 Hauptprogramm (stabilise1.c)

```
/*
  Laserstahl-Positions-Steuerung (2 unabhngige Regelkanle)

  (c) 2003 by Jan Krieger
  jan@jkrieger.de
  www.jkrieger.de

  Zielsystem: Atmel ATmega32 ohne externes RAM, Quarz: 16MHz
  Compiler:   gcc 3.3.1 fr AVR
*/

#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/wdt.h>
#include "lcd.h"
#include "alignment.c"
#include <stdio.h>

#define multiplier 10000
// ein Faktor mit dem die ausgelesenen SPannungen multipliziert werden.
// er bewirkt eine Verschiebung des Kommas und somit ein hhere/niedrigere
// Genauigkeit.

unsigned char LEDTolerance=3;
//tolerance gibt an, um wieviel eine Differenz von 0 abweichen
//darf, um noch als mittig an den LEDs angezeigt zu werden. Das
//Maximum betrgt 2048 (=[2^12]/2), das Minimum 0.

unsigned char MinValuecount=10;
//MinValuecount gibt an, ber wieviele Werte mindestens gemittelt
//worden sen muss, bevor eine Nachregelung erfolgt:
//     10 bedeutet, dass erst 10 Werte aufgezeicvhnet werden, bevor die
//     Regelung die Ausgangsspannungen ndert!

unsigned char MeanFactor[2]={8,8};
//Faktor zur Berechnung des laufenden Mittelwerts [0..1]. Der alte Wert
//wird mit dem Faktor multipliziert. Dazu wird der neue Messwert, mit
//(1-<MeanFactor>) multipliziert, addiert.

#define Preset_center 2047
//<Preset_center> gibt an, auf welchen Wert die Ausgangskanle gestellt werden,
//wenn die Mittelwert/Preset-Schalter aktiviert sind.

unsigned char RegulationFactor[2]={2,2};
unsigned char RegulationFraction[2]={2,2};
signed char SignX[2]={-1, -1};
signed char SignY[2]={1, 1};
//<RegulationFactor> ist ein Anpassungsfaktor fr die (inkrementelle) Regelfunktion
//aus void regulation_out (...). Die Formel lautet:
//     out =+ (Delta_mean * <RegulationFactor> / <RegulationFraction>)

diode_direction DiodeDirection[2]={up,up};
// gibt die Ausrichtung der Dioden an (Erluterung, siehe regulation_input (), oder alignment.c

#define displaycountMax 30
#define displaycountMax_extern 2
/* im frei laufenden Betrieb (interner Trigger) wird <displaycountMax> Auslesevorgnge
   abgewartet, bis ein Wert fr die Srahlposition auf dem LCD ausgegeben wird. Dadurch
   wird bei ca 200-300Hz Regelfrequenz noch ein leserliches Bild erreicht.
```

```
    displaycountMax_extern macht dasselbe  fr  externe  Triggerung.  Hier  sollte  ein  kleiner
    Wert (~2)  fr  niedrige  Triggerfrequenzen  (~10 Hz)  eingestellt  werden.*/

unsigned char outofarea[2]={0, 0};
/* falls  sich  die  Regelung am Anschlag befindet (max./min. Spannung am Piezo), dann versuche
    bei  outofarea==1 durch Ausgeben der halben Maximalspannung das Problem zu lsen. */

center_mode centermode[2]={cm_center,cm_center};

//------------------------------------------------------------------------------------------------
/*
    Adresskonstanten zum Ablegen der Konfigurationsdaten im EEPROM
*/
#define A_MinValuecount 0
#define A_MeanFactor 1
#define A_RegulationFactor 3
#define A_RegulationFraction 5
#define A_SignX 7
#define A_SignY 9
#define A_DiodeDirection 11
#define A_LEDTolerance 13
#define A_outofarea 14
#define A_centermode 16


/*
   Variablen,  fr   fr  Regelung 1 bzw. 2 mit  nachgestellter  1 bzw 2

   Xmean, Ymean: laufender Mittelwert in X-  bzw.  Y-Richtung
   valuecount:   Anzahl der bisher  aufgezeichneten  Werte. Diese Variable wird
                 nur bis <MinValuecount> hochgezhlt und dazu benutzt die
                 Regelung zu sperren,  falls  zu wenig Werte vorliegen.
   Xout, Yout:    aktueller  Ausgabe-Wert an den DA-Kanlen in X-  bzw. Y-Richtung

   displaycount: ein  Zhler . Immer, wenn er <displaycountMax> erreicht, wird bei  interner
                 Triggerung ein  Wert auf dem LCD-Display angezeigt.
*/
int Xmean0, Ymean0;
unsigned int Xout0, Yout0;
int Xmean1, Ymean1;
int valuecount[2];
unsigned int Xout1, Yout1;
reg_mode mode[2];
trigger_mode TriggerMode;
FILE *LCD;
int displaycount0, displaycount1;

unsigned char editmode=0; //editmode>0 bedeutet Schaltung ist im Dateneingabemodus
                          // Editmode gibt an, welche Variable gerade  editiert  wird.
unsigned int buttoncount_UP=0, buttoncount_DOWN=0, buttoncount_MODE=0;

//   SaveConfig() & LoadConfig() speichern/laden die  Konfigurationsadten (Variablen) aus dem/ins EEPROM
void SaveConfig(void) {
  eeprom_wb(A_MinValuecount, MinValuecount);
  eeprom_wb(A_MeanFactor, MeanFactor[0]);
  eeprom_wb(A_MeanFactor+1, MeanFactor[1]);
  eeprom_wb(A_RegulationFactor, RegulationFactor[0]);
  eeprom_wb(A_RegulationFactor+1, RegulationFactor[1]);
  eeprom_wb(A_RegulationFraction, RegulationFraction[0]);
  eeprom_wb(A_RegulationFraction+1, RegulationFraction[1]);
  eeprom_wb(A_SignX, SignX[0]);
  eeprom_wb(A_SignX+1, SignX[1]);
  eeprom_wb(A_SignY, SignY[0]);
  eeprom_wb(A_SignY+1, SignY[1]);
  eeprom_wb(A_DiodeDirection, DiodeDirection[0]);
  eeprom_wb(A_DiodeDirection+1, DiodeDirection[1]);
  eeprom_wb(A_LEDTolerance, LEDTolerance);
  eeprom_wb(A_outofarea, outofarea[0]);
  eeprom_wb(A_outofarea+1, outofarea[1]);
  eeprom_wb(A_centermode, centermode[0]);
  eeprom_wb(A_centermode+1, centermode[1]);
}

void LoadConfig(void) {
  MinValuecount=eeprom_rb(A_MinValuecount ); if ((MinValuecount>250)||(MinValuecount<1)) MinValuecount=10;
  valuecount[0]=0;
  valuecount[1]=MinValuecount/2;
  MeanFactor[0]=eeprom_rb(A_MeanFactor ); if ((MeanFactor[0]>10)||(MeanFactor[0]<1)) MeanFactor[0]=8;
  MeanFactor[1]=eeprom_rb(A_MeanFactor+1 ); if ((MeanFactor[1]>10)||(MeanFactor[1]<1)) MeanFactor[1]=8;
  RegulationFactor[0]=eeprom_rb(A_RegulationFactor ); if ((RegulationFactor[0]>100)||(RegulationFactor[0]<1)) RegulationFactor[0]=2;
  RegulationFactor[1]=eeprom_rb(A_RegulationFactor+1 ); if ((RegulationFactor[1]>100)||(RegulationFactor[1]<1)) RegulationFactor[1]=2;
  RegulationFraction[0]=eeprom_rb(A_RegulationFraction ); if ((RegulationFraction[0]>100)||(RegulationFraction[0]<1)) RegulationFraction[0]=2;
  RegulationFraction[1]=eeprom_rb(A_RegulationFraction+1 ); if ((RegulationFraction[1]>100)||(RegulationFraction[1]<1)) RegulationFraction[1]=2;
  SignX[0]=eeprom_rb(A_SignX ); if ((SignX[0]!=-1)&&(SignX[0]!=1)) SignX[0]=-1;
  SignX[1]=eeprom_rb(A_SignX+1 ); if ((SignX[1]!=-1)&&(SignX[1]!=1)) SignX[1]=-1;
  SignY[0]=eeprom_rb(A_SignY ); if ((SignY[0]!=-1)&&(SignY[0]!=1)) SignY[0]=1;
  SignY[1]=eeprom_rb(A_SignY+1 ); if ((SignY[1]!=-1)&&(SignY[1]!=1)) SignY[1]=1;
  DiodeDirection[0]=eeprom_rb(A_DiodeDirection ); if ((unsigned char)DiodeDirection[0]>3) DiodeDirection[0]=up;
  DiodeDirection[1]=eeprom_rb(A_DiodeDirection+1 ); if ((unsigned char)DiodeDirection[1]>3) DiodeDirection[1]=up;
  LEDTolerance=eeprom_rb(A_LEDTolerance ); if ((LEDTolerance>100)||(LEDTolerance<1)) LEDTolerance=3;
  outofarea[0]=eeprom_rb(A_outofarea );  if  ((outofarea[0]!=0)&&(outofarea[0]!=1)) outofarea[0]=0;
  outofarea[1]=eeprom_rb(A_outofarea+1 ); if ((outofarea[1]!=0)&&(outofarea[1]!=1)) outofarea[1]=0;
  centermode[0]=eeprom_rb(A_centermode ); if ((unsigned char)centermode[0]>2) centermode[0]=cm_center;
  centermode[1]=eeprom_rb(A_centermode+1 ); if ((unsigned char)centermode[1]>2) centermode[1]=cm_center;
}

/* Erklrungen  zum Editiermodus:
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   Der Controller  schaltet  in den Editiermodus, wenn beide  (!!!)  Kanle  auf FREEZE stehen. Im CENTER-Modus
   werden die Werte von den Dioden angezeigt. Der Editiermodus erlaubt es die Regelparameter einzustellen .
   Die Daten werden dann im EEPROM gespeichert und beim nchsten Programmstart wieder geladen.

   Die Bedienung erfolgt  ber  drei  Tasten. Die ENTRY-MODE-Taste an PA6 schaltet zwischen den einzelnen Parametern
   hin und her (die  Parameter werden als Ring durchlaufen). Die beiden anderen Tasten UP (PB3) und DOWN (PB2) dienen
   zum ndern des entsprechenden Wertes.
```

```
  Die Abfrage der Tasten erfolgt folgendermaen:
  Der 16-bit-Timer1 luft durch und fragt einige male pro Sekunde die Tasten ab. Zu jeder Taste existiert eine
   Zhlervariable , die angibt wieviele Timer- berlufe die Taste bereits gedrckt ist . Wird die Taste losgelassen ,
  so wird diese Variable auf 0 zurckgesetzt . Die Dateneingabe erfolgt dann ber die Auswertung dieser Variablen.
  Wobei ein lngeres drcken einer Taste ein schnelles Hochzhlen des Wertes zur Folge hat.

*/


/*
  DIe folgenden Funktionen zeigen den Editor auf dem LCD an und fragt die Tasten UP/DOWN ab
  je nach Wert von emode wird der Editor zu einem Parameter angezeigt. Falls sich die Werte ndern,
  wird der neue Wert im EEPROM abgelegt.

  ACHTUNG: Bei nderungen den Maximalwert fr editmode im Interrupt-Handlert anpassen !!!

  unschnder Programmierstiel, um die Hauptlast vom RAM ins FLASH zu verlagern !!!
*/
#define button_firsthit 1  // gibt an, ab welchen Zhlerstand der Tastendruck erkannt wird
#define button_modulostart 6 // gibt an, ab wann der Schnelllaufmodus aktiviert wird
#define button_modulo 2 // gibt bei jeder wievielten Zahl ein Tastendruck erkannt wird

void editor_signed(signed char *value, signed char min, signed char max, const int adr) {
  if (buttoncount_UP==button_firsthit) {
    if (*value<max) {*value+=1; eeprom_wb(adr, *value); } else {*value=max; eeprom_wb(adr, *value); }
  } else {
    if (buttoncount_UP>button_modulostart) {
      if ((buttoncount_UP%button_modulo)==0){
        if (*value<max) { *value+=1; eeprom_wb(adr, *value); } else {*value=max; eeprom_wb(adr, *value);}
    }}
  }
  if (buttoncount_DOWN==button_firsthit) {
    if (*value>min) {*value-=1; eeprom_wb(adr, *value); } else {*value=min; eeprom_wb(adr, *value); }
  } else {
    if (buttoncount_DOWN>button_modulostart) {
      if ((buttoncount_DOWN%button_modulo)==0){
        if (*value>min) { *value-=1; eeprom_wb(adr, *value); } else { *value=min; eeprom_wb(adr, *value); }
    }}
  }
}

void editor_unsigned(unsigned char *value, unsigned char min, unsigned char max, const int adr) {
  if (buttoncount_UP==button_firsthit) {
    if (*value<max) {*value+=1; eeprom_wb(adr, *value); } else {*value=max; eeprom_wb(adr, *value); }
  } else {
    if (buttoncount_UP>button_modulostart) {
      if ((buttoncount_UP%button_modulo)==0) {
        if (*value<max) { *value+=1; eeprom_wb(adr, *value); } else {*value=max; eeprom_wb(adr, *value);}
    }}
  }
  if (buttoncount_DOWN==button_firsthit) {
    if (*value>min) {*value-=1; eeprom_wb(adr, *value); } else {*value=min; eeprom_wb(adr, *value); }
  } else {
    if (buttoncount_DOWN>button_modulostart) {
      if ((buttoncount_DOWN%button_modulo)==0){
        if (*value>min) { *value-=1; eeprom_wb(adr, *value); } else { *value=min; eeprom_wb(adr, *value); }
    }}
  }
}

void editor_sign(signed char *value, const int adr) {
  if (buttoncount_UP==button_firsthit) {
    *value=+1; eeprom_wb(adr, *value);
  }
  if (buttoncount_DOWN==button_firsthit) {
    *value=-1; eeprom_wb(adr, *value);
  }
}

void editor_bool(unsigned char *value, const int adr) {
  if (buttoncount_UP==button_firsthit) {
    *value=1;  eeprom_wb(adr, *value);
  }
  if (buttoncount_DOWN==button_firsthit) {
    *value=0;  eeprom_wb(adr, *value);
  }
}

void showeditor(unsigned char emode) {
  switch(emode) {
    case 1: lcd_goto(0x00); fputs("Samples:_____________", LCD);
            lcd_goto(0x40);  fprintf(LCD, "[1..250]:_%d_____", MinValuecount);
            editor_unsigned(&MinValuecount, 1, 250, A_MinValuecount);
            valuecount[0]=0;
            valuecount[1]=MinValuecount/2;
            break;
    case 2: lcd_goto(0x00); fputs("LEDTolerance:__________", LCD);
            lcd_goto(0x40);  fprintf(LCD, "[1..100]:_%d_____", LEDTolerance);
            editor_unsigned(&LEDTolerance, 1, 100, A_LEDTolerance);
            break;
    case 3: lcd_goto(0x00); fputs("CH1:_MeanFactor________", LCD);
            lcd_goto(0x40);  fprintf(LCD, "[0..10]:__%d_____", MeanFactor[0]);
            editor_unsigned(&MeanFactor[0], 1, 10, A_MeanFactor);
            break;
    case 4: lcd_goto(0x00); fputs("CH1:_RegFactor:________", LCD);
            lcd_goto(0x40);  fprintf(LCD, "[1..100]:_%d_____", RegulationFactor[0]);
            editor_unsigned(&RegulationFactor[0], 1, 100, A_RegulationFactor);
            break;
    case 5: lcd_goto(0x00); fputs("CH1:_RegFraction:_______", LCD);
            lcd_goto(0x40);  fprintf(LCD, "[1..100]:_%d_____", RegulationFraction[0]);
            editor_unsigned(&RegulationFraction[0], 1, 100, A_RegulationFraction);
            break;
    case 6: lcd_goto(0x00); fputs("CH1:_SignX:____________", LCD);
            lcd_goto(0x40);  fprintf(LCD, "[+,_-]:___%d_____", SignX[0]);
            editor_sign(&SignX[0], A_SignX);
            break;
```

```
    case 7: lcd_goto(0x00); fputs("CH1: SignY:             ", LCD);
            lcd_goto(0x40); fprintf (LCD, "[+,-]:   %d     ", SignY[0]);
            editor_sign (&SignY[0], A_SignY);
            break;
    case 8: lcd_goto(0x00); fputs("CH1: DiodeDir.:          ", LCD);
            lcd_goto(0x40); lcd_putstr("         "); printDD(DiodeDirection[0]);lcd_putstr("         ");
            editor_unsigned (&DiodeDirection [0], 0, 3, A_DiodeDirection);
            break;
    case 9: lcd_goto(0x00); fputs("CH1: Out of Area Korr.    ", LCD);
            lcd_goto(0x40); lcd_putstr ("         "); printbool(outofarea [0]); lcd_putstr ("         ");
            editor_bool(&outofarea [0], A_outofarea);
            break;
    case 10: lcd_goto(0x00); fputs("CH2: MeanFactor         ", LCD);
            lcd_goto(0x40); fprintf (LCD, "[0..10]:  %d     ", MeanFactor[1]);
            editor_unsigned(&MeanFactor[1], 1, 10, A_MeanFactor+1);
            break;
    case 11: lcd_goto(0x00); fputs("CH2: RegFactor:         ", LCD);
            lcd_goto(0x40); fprintf (LCD, "[1..100]: %d     ", RegulationFactor[1]);
            editor_unsigned(&RegulationFactor [1], 1, 100, A_RegulationFactor+1);
            break;
    case 12: lcd_goto(0x00); fputs("CH2: RegFraction:       ", LCD);
            lcd_goto(0x40); fprintf (LCD, "[1..100]: %d     ", RegulationFraction[1]);
            editor_unsigned(&RegulationFraction [1], 1, 100, A_RegulationFraction+1);
            break;
    case 13: lcd_goto(0x00); fputs("CH2: SignX:             ", LCD);
            lcd_goto(0x40); fprintf (LCD, "[+,-]:   %d     ", SignX[1]);
            editor_sign (&SignX[1], A_SignX+1);
            break;
    case 14: lcd_goto(0x00); fputs("CH2: SignY:             ", LCD);
            lcd_goto(0x40); fprintf (LCD, "[+,-]:   %d     ", SignY[1]);
            editor_sign (&SignY[1], A_SignY+1);
            break;
    case 15: lcd_goto(0x00); fputs("CH2: DiodeDir.:          ", LCD);
            lcd_goto(0x40); lcd_putstr("         "); printDD(DiodeDirection[1]);lcd_putstr("         ");
            editor_unsigned (&DiodeDirection [1], 0, 3, A_DiodeDirection+1);
            break;
    case 16: lcd_goto(0x00); fputs("CH2: Out of Area Korr.    ", LCD);
            lcd_goto(0x40); lcd_putstr ("         "); printbool(outofarea [1]); lcd_putstr ("         ");
            editor_bool(&outofarea [1], A_outofarea+1);
            break;
  }
}

// Reaktion auf Timer-Interrupt
SIGNAL (SIG_OUTPUT_COMPARE1A) {
  cli ();
  if (buttoncount_UP>0xFFFA) buttoncount_UP=1; //Rcksetzen der Zhler vor  berlauf
  if (buttoncount_DOWN>0xFFFA) buttoncount_DOWN=1;
  if (buttoncount_MODE>0xFFFA) buttoncount_MODE=1;

  if (button_UP()) buttoncount_UP+=1; else buttoncount_UP=0;
  if (button_DOWN()) buttoncount_DOWN+=1; else buttoncount_DOWN=0;
  if (button_MODE()) buttoncount_MODE+=1; else buttoncount_MODE=0;

  // Abfrage, ob Modus gewechselt werden soll. Fr jeden Moduswechsel ist ein eigener Klick erfordelrich , der mind.
  // zwei Timer-Interrupts anhlt:
  if (buttoncount_MODE==1) {
    editmode+=1;
  } else {
    if (buttoncount_MODE>8) {
      if (((buttoncount_DOWN%4)==0)) editmode+=1;
    }
  }
  if (editmode>16) editmode=1; //Falls in Editiermodus>13 geschlaten werden soll (dieser existeirt nicht)
                               // dann wird wieder in Modus 1 zurckgeschlatet .
  showeditor(editmode); //zeigt den Editor auf dem LCD an
  sei ();
  timer1_reset (); //Timer wieder fr  nchsten Interrupt vorbereiten (Startwert setzen)
}

/*
   regulation_input beinhaltet das Einlesen der Daten fr die Regelung
  und die Verarbeitung bis zur Mittelwertbildung und der Anzeige auf den LEDs

  Aufruf z.B. in der INT0-Interruptroutine

  channel: gibt die Regelung an. channel=[0..1]
*/
void regulation_input(unsigned char channel, int *Xmean, int *Ymean, int *valuecount) {
  /*
     A-D:            Messwerte der einzelnen Quadranten [0..4096]
     summe:          A+B+C+D zur Normierung der Abweichungen
     DeltaX, DeltaY: Abweichung von 0 in X-/Y-Richtung
  */
  long A, B, C, D;
  unsigned char channel1;
  long DeltaX, DeltaY, summe;


  // Regelung 1 -----------------------------------------------------------------
    // 1. Daten einlesen und Summen, sowie standardisierte Abweichungen berecnen

    if (*valuecount<=MinValuecount) {
      (*valuecount)++;
    }

    channel1 = channel*4;
    A = ReadAD(channel1); //Auslesen der vier Quadranten
    B = ReadAD(channel1+1);
    C = ReadAD(channel1+2);
    D = ReadAD(channel1+3);
    summe = A+B+C+D; //Summe zum Normieren auf Gesamtintensitt

/*
 mgliche Ausrichtungen der Qudrantendioden:
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
 diode_direction :          up              left            down            right

Dioden (von Vorne):         A | B           B | D           D | C           C | A
                            --+--           --+--           --+--           --+--
                            C | D           A | C           B | A           D | B

DeltaX:                   A+C-(B+D)       A+B-(C+D)       B+D-(A+C)       C+D-(A+B)
DeltaY:                   A+B-(C+D)       B+D-(A+C)       C+D-(A+B)       A+C-(B+D)

*/

    switch(DiodeDirection[channel]) {
      case up:
        DeltaX = (A+C-(B+D))*multiplier /summe;///summe1;
        DeltaY = (A+B-(C+D))*multiplier /summe;///summe1;
        break;
      case left:
        DeltaX = (A+B-(C+D))*multiplier/summe;///summe1;
        DeltaY = (B+D-(A+C))*multiplier/summe;///summe1;
        break;
      case down:
        DeltaX = (B+D-(A+C))*multiplier/summe;///summe1;
        DeltaY = (C+D-(A+B))*multiplier/summe;///summe1;
        break;
      case right:
        DeltaX = (C+D-(A+B))*multiplier/summe;///summe1;
        DeltaY = (A+C-(B+D))*multiplier/summe;///summe1;
        break;
    }

    //neue Mittelwerte berechnen ( gleitender  Mittelwert , siehe auch
    //Konstantenbeschreibung zu <MeanFactor>
    *Xmean *= MeanFactor[channel];
    *Ymean *= MeanFactor[channel];
    *Xmean += (DeltaX*(10-MeanFactor[channel]));
    *Ymean += (DeltaY*(10-MeanFactor[channel]));
    *Xmean /= 10;
    *Ymean /= 10;
}

/*
   reguation_output  beinhaltet  die  Umrechnung des Mittelwertes in eine
    Ausgabegre .


   Bei einem  berlauf  der Ausgabespannung nach oben, oder unten */
void regulation_output(unsigned char channel, int Xmean, int Ymean, unsigned int *Xout, unsigned int *Yout, int *valuecount, int *displaycount) {
  signed int dummyXout, dummyYout;

    dummyXout= (int)(*Xout);
    dummyYout= (int)(*Yout);

     // Falls  weniger  als <MinValueCount> Messwerte erfasst, dann keine Nachregelung.
     // nur valuecount  erhhen .
     if (*valuecount>=MinValuecount) {
       //Zentrieren-Schalter abfragen ,  falls  aktiviert  dann keine Regelung,
       //sondern <Preset_center> ausgeben
       if (mode[channel]!=center) {
         // 2. Regelung, wenn "Einfrieren-Schalter" nicht  bettigt .
         if (mode[channel]!=freeze) {
           dummyXout += SignX[channel]*(Xmean * RegulationFactor[channel] / RegulationFraction[channel]);
           dummyYout += SignY[channel]*(Ymean * RegulationFactor[channel] / RegulationFraction[channel]);
           *valuecount=0;
        } else {
           *valuecount=0;
           //nach der Rckkehr  aus dem freeze-Modus mssen wieder neue Werte aufgezeichnet werden
         }
       } else {
         // aktivierter  Mittelwert-Schalter:

         //nach dem deaktivieren  mssen  wieder  einige  Messwerte aufgenommen werden
         //bevor wieder  geregelt  wird.  ==> valuecount auf 0
         *valuecount=0;

         //Ausgabe von <Preset_center> an X-/Y-Kanal.
         dummyXout=Preset_center;
         dummyYout=Preset_center;
         //printf("preset");
       }
     }

     if ((*displaycount>=displaycountMax)||(TriggerMode!=internal)) {
       if (channel==0) {lcd_goto(0x00); } else { lcd_goto(0x40); }          //DeltaX und DeltaY auf LCD ausgeben
       fprintf(LCD, "%d_%c%c:_%3d_", channel, ModeToC(mode[channel]), (mode[channel]!=center)?'_':CModeToC(centermode[channel]), Xmean);
       fprintf(LCD, "%3d____", Ymean);
       *displaycount=0;
     } else {*displaycount=*displaycount+1;}


   //Umwandlung von dummyXout/dummyYout in einen signed int aus [0..4096], falls
   //dummyXout/dummyYout unter oder oberhalb der Grenzen liegt wird es auf den
   //jeweilig  nchsten  Grenzwert gesetzt .
   if (channel==0) {lcd_goto(14);}else{lcd_goto(0x4E);}
   if (dummyXout>4095) {
     *Xout=4095 ;
     lcd_putch('o');
     if (outofarea[channel]==1) *Xout=Preset_center;
   } else {
     if (dummyXout<0){
       *Xout=0;
       if (outofarea[channel]==1) *Xout=Preset_center;
       lcd_putch('u');
     } else {
       *Xout=(unsigned int)dummyXout;
       lcd_putch('_');
     }
   }
```

```
  if (channel==0) {lcd_goto(15);}else{lcd_goto(0x4F);}
  if (dummyYout>4095) {
    *Yout=4095 ;
    lcd_putch('o');
    if (outofarea[channel]==1) *Yout=Preset_center;
  } else {
    if (dummyYout<0){
      *Yout=0;
      lcd_putch('u');
      if (outofarea[channel]==1) *Yout=Preset_center;
    } else {
      *Yout=(unsigned int)dummyYout;
      lcd_putch('_');
    }
  }
}

/*
   stabilise  liest  die  Daten von beiden Kan"alen ein, berechnet  die  neuen Ausgaben und schickt diese  auf
  die  DA-Wandler
*/
void stabilise (void) {
  ledstate  ledX0=ok, ledX1=ok, ledY0=ok, ledY1=ok;

  mode[0]=getMode(0);
  mode[1]=getMode(1);

  if ((mode[0]!=freeze)||(mode[1]!=freeze)) {
    SetBusyLED(1);

    //Daten einlesen und Position berechnen
    regulation_input(0, &Xmean0, &Ymean0, &valuecount[0]);
    regulation_input(1, &Xmean1, &Ymean1, &valuecount[1]);

    //Regel-Ausgabe berechnen
    regulation_output(0, Xmean0, Ymean0, &Xout0, &Yout0, &valuecount[0], &displaycount0);
    regulation_output(1, Xmean1, Ymean1, &Xout1, &Yout1, &valuecount[1], &displaycount1);

    //LED-Kreuze setzen
    if (Xmean0>LEDTolerance) ledX0=over; if (Xmean0<-LEDTolerance) ledX0=under;
    if (Xmean1>LEDTolerance) ledX1=over; if (Xmean1<-LEDTolerance) ledX1=under;
    if (Ymean0>LEDTolerance) ledY0=over; if (Ymean0<-LEDTolerance) ledY0=under;
    if (Ymean1>LEDTolerance) ledY1=over; if (Ymean1<-LEDTolerance) ledY1=under;
    SetLED(ledX0, ledY0, ledX1, ledY1);


    // berprfung  der Phasenbeziehung der Kanle
    if (valuecount[0]==0) {
      valuecount[1]=MinValuecount/2;
    }

    SetBusyLED(0);
  }
  // festgelegte  Spannungen ausgeben.
  SetDA(Xout0, Yout0, Xout1, Yout1);
}

/*  Interrupt  0 wird vom Timing auf der Platine bei jedem Trigger-Impuls
    etwas  verzgert  ausgelst .Das Timing muss so eingestellt werden, dass
    zuerst  die Sample/Hold-Stufen getriggert werden, also den bis dahin
    gesampelten Wert halten (Sampling beginnt direkt beim Trigger-Impuls,
    der kurz vor dem Laser-Impuls kommen muss) und danach, etwas verzgert
    (um ein Einpendeln des Wertes zu ermglichen ), der Interrupt  ausgelst
    wird.
    In dieser  Interrupt-Routine wird die Regelung implementiert. Das Haupt-
    programm befindet sich nach der Hardware-Initialisierung in einer
    Endosschleife!
*/
SIGNAL (SIG_INTERRUPT0) {
  stabilise ();
}

//Haupt-Routine mit Endlosschleife
int main(void)
{

    InitHardware();  //Hardware  initialisieren
    /*
      Watchdog-Timer wird eingesetzt um zu verhindern, dass die Regelung hngen  bleibt . Die Zeitkonstante  ist  auf
      500 ms  eingestellt . D.h.  sptestens  500 ms nach einem Absturz wird ein  Reset  ausgefhrt .

      Das Rcksetzen des Watchdogs erfolgt in der folgenden Endlos-Schleife
    */
    wdt_enable(WDTO_500MS);
    disableINT0();
    timer1_disable();
    SetBusyLED(0);
    LCD=fdevopen(LCDwrite, NULL, 0);       //LCD-Ausgabe initialisieren
    lcd_init ();
    lcd_control (0,0,0);
    lcd_cls ();
    lcd_home();
    lcd_putstr(" Strahlstabilis .___");
    lcd_goto(0x40);
    lcd_putstr("(c)03_J.Krieger___");
    SetDA(2047, 2047, 2047, 2047);
    wdt_reset();

     //Variablen  initialisieren
    valuecount[0]=0;
    valuecount[1]=MinValuecount/2;
    displaycount0=0;
    displaycount1=0;
    Xmean0=0;
    Ymean0=0;
    Xmean1=0;
    Ymean1=0;
```

```
    Xout0=2047;
    Yout0=2047;
    Xout1=2047;
    Yout1=2047;
    mode[0]=center;
    mode[1]=center;
    TriggerMode=internal;
    editmode=0;
    wdt_reset();

    LoadConfig();
    wdt_reset();

    sei();

    for (;;) {   // forever

      // Watchdog-Reset
      wdt_reset();


      /*
        Einstellen des Trigger-Modus (intern/extern):
          - im externen Modus ist die Regelung Interrupt getrieben (INT0)
          - im internen Modus wird direkt im Anschluss an eine Berechnung die nchste durchgefhrt .
      */
      if ((getTriggerMode()==external)&&(TriggerMode!=external)) {
        enableINT0();
        TriggerMode=external;
      }
      wdt_reset();
      if ((getTriggerMode()==internal)&&(TriggerMode!=internal)) {
        disableINT0();
        TriggerMode=internal;
      }
      if (TriggerMode==internal) stabilise();
      wdt_reset();

      mode[0]=getMode(0);
      mode[1]=getMode(1);
      if ((mode[0]==freeze)&&(mode[1]==freeze) &&(editmode==0)) { //Umschalten in den Editiermodus, wenn beide Kanle auf FREEZE und Editiermodus noch nicht aktiv
            //timer2_disable();
            editmode=1;
            timer1_enable();
      }
      if ((mode[0]!=freeze)||(mode[1]!=freeze)) { timer1_disable(); editmode=0; } //sonst sicherstellen, dass Editiermodus nicht aktiviert ist
      wdt_reset(); //Watchdog zuruecksetzen

    }
}
```

Literatur

[APEX Microtechnology 2002] *APEX Microtechnology*: **"High Voltage Operational Amplifier PA140."** 2002
[http://eportal.apexmicrotech.com/mainsite/pdf/PA140U.pdf]

[Atmel 2003] *Atmel*: **"ATmega32: 8-bit AVR Microcontroller with 32K Bytes in-system Programmable Flash."** 2003
[http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf]

[Dreher 2003] *Dreher, M.*: **"Superradiant Amplification of Ultra-Short Laser Pulses."** (http://www.mpq.mpg.de/lpg/research/supraAmplification/sra.html, 01.12.2003)

[Laser 2000 2003a] *Laser 2000*: **"Si-Photodioden - Theorie."** 2003
[http://www.laser2000.de/pdf/Messtechnikkatalog-2002/Einzelseiten/08-Photodioden/01-Si-Photodioden_Theorie.pdf]

[Laser 2000 2003b] *Laser 2000*: **"Optische Positionsdetektoren - Theorie."** 2003
[http://www.laser2000.de/pdf/Messtechnikkatalog-2002/Einzelseiten/04-Positionsmessung/01-Positionsdet-Theorie.pdf]

[Linear Technologies 1992] *Linear Technologies*: **"LTC1293/LTC1294/LTC1296. Single Chip 12-Bit Data Acquisition System."** 1992
[http://www.linear.com/pdf/lt1293.pdf]

[Linear Technologies 1996] *Linear Technologies*: **"LTC1446/LTC1446L. Dual 12-Bit Rail-to-Rail Micropower DACs."** 1996
[http://www.linear.com/pdf/1446fa.pdf]

[Stöcker 2000] *Stöcker, H. (Hrsg.)*: **"Taschenbuch der Physik.",** Verlag Harri Deutsch, Thun Frankfurt/Main 2000[4]

[Texas Instruments 1999] *Texas Instruments*: **"TL081-084 JFET-Input Operational Amplifiers."** 1999
[http://www-s.ti.com/sc/ds/tl082a.pdf]

[Tietze, Schenk 2002] *Tietze, U., Schenk, C.*: **"Halbleiter-Schaltungstechnik.",** Springer, New York Berlin Heidelberg 2002[12]

Danksagung

Zum Schluss möchte ich allen Mitarbeitern in der Gruppe von Prof. Witte danken, die mich unterstützt haben: Insbesondere Michael Jost und Matthias Dreher für die Betreuung während meines Praktikums und den Elektronikern Anton Horn und Harald Haas für Ihre Geduld, Hilfe bei der Elektronik und die Befriedigung meines Bauteile-Hungers. Außerdem möchte ich Prof. Lindenstruth von der Universität Heidelberg danken, der es mir ermöglicht hat dieses Projekt als Praktikum für mein Physik-Studium anerkannt zu bekommen.