Diplomarbeit

Christoph Klaukin

Entwicklung und Realisierung einer mikroprozessor-basierten low-cost Showlasersteuerung mit Grafik-LCD und SD-Karte

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Christoph Klaukin

Entwicklung und Realisierung einer mikroprozessorbasierten low-cost Showlasersteuerung mit Grafik-LCD und SD-Karte

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.-Ing. Robert Fitz
Zweitgutachter : Prof. Dr.-Ing. Ulrich Vogelsang

Abgegeben am 07. Dezember 2007

Christoph Klaukin

Thema der Diplomarbeit

Entwicklung und Realisierung einer mikroprozessorbasierten low-cost Showlasersteuerung mit Grafik-LCD und SD-Karte

Stichworte

ILDA, Showlaser, SD-Karte, Dateisystem FAT, Mikroprozessor, SRAM, Grafikdisplay, 12-Bit-DAC

Kurzzusammenfassung

Entwickelt wird ein kostengünstiges Steuergerät für die Wiedergabe von Lasershows ohne feste Anbindung an einen PC. Die Dateien, in denen die Lasershows nach *ILDA* gespeichert sind, werden auf einer handelsüblichen SD-Karte abgelegt. Als Dateisystem ist FAT16/32 vorgesehen. So ist es möglich, den Inhalt der SD-Karte mit einem PC zu verwalten. Für die SD-Karte und das auf ihr befindliche Dateisystem werden in dieser Arbeit nur Lesezugriffe unterstützt. Das Ansprechen der SD-Karte und des Dateisystems erfolgt mit fester Block- bzw. Sektorgröße. Für die Bedienung sind ein Grafikdisplay und drei über Hardware entprellte Taster vorgesehen. Das Gerät verfügt über eine Schnittstelle (nach *ILDA*) zum Anschluss an einen Laserprojektor. Die symmetrisch übertragenen Steuersignale werden für die beiden Achsenkanäle mittels 12-Bit-DACs und für die verwendeten Farbkanäle mittels 4-Bit-DACs generiert.

Christoph Klaukin

Title of the paper

Development and implementation of a microprocessor-based low-cost laser show controlling system with graphic-LCD and SD Card

Keywords

ILDA, show laser, SD Card, file system FAT, microprocessor, SRAM, graphic display, 12-Bit-DAC

Abstract

A low-cost controlling system for playing laser shows without a fixed connection to a PC is developed. The files containing the laser shows after *ILDA* are saved on a standard SD Card. The used file system is FAT16/32. Therefore it is possible to manage the contents of the SD Card with a PC. In this Thesis the SD Card and its file system can only be read, write access is not supported. The connection to the SD Card including its file system is done by fixed block size and fixed sector size. For operation a graphic display and three on hardware debounced buttons are planned. The device has got an interface (after *ILDA*) to be connected to a laser projector. The symmetrically transferred control signals for the two axis channels are generated by means of 12-Bit-DACs and the used color channels by means of 4-Bit-DACs.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Gleichungsverzeichnis

1 Einleitung

In den letzten Jahren hat sich die Show- und Medientechnik besonders stark verändert, was u.a. mit der sich sehr schnell weiterentwickelnden Elektrotechnik bzw. Elektronik und den damit verbundenen neuen Möglichkeiten zusammenhängt. Des Weiteren verschmelzen die Grenzen zwischen Show- und Videotechnik zunehmend. Die Anzahl der oftmals fest installierten und immer aufwendiger werdenden Multimediashows steigt stetig von Jahr zu Jahr.
Insbesondere die Lasertechnik hat in den letzten Jahren mit der Serienproduktion von DPSS-Halbleiterlasern (engl.: Diode Pumped Solid State) die Vielfältigkeit von Lasershows außerordentlich erweitert, da sich diese Laser ohne mechanischen und mit nur geringem elektronischen Aufwand sowie hoher Frequenz in der Helligkeit (Intensität) steuern lassen. Bei der Verwendung von drei verschiedenfarbigen Lasern (rot, grün, blau) und einer entsprechenden Optik zur Überlagerung der einzelnen Laserstrahlen lässt sich praktisch jede beliebige Farbe mischen. Dieses ist jedoch von der Auflösung der steuernden DACs (engl.: Digital Analog Converter) für die Farbkanäle abhängig.

DPSS-Laser bieten zudem einen deutlich höheren Wirkungsgrad als die bisher verwendeten Festkörper- oder Gaslaser. Die Leistungsdichte der Lasersysteme nimmt zu, während der Aufwand zum Abführen der überschüssigen Verlustwärme nur noch durch kleine Lüfter erfolgen muss. Schwere Wasserkühlungen und ein Drehstromanschluss sind nicht mehr erforderlich. DPSS-Laser haben eine längere Lebensdauer als Festkörper- oder Gaslaser und müssen im Normalfall nie nachjustiert werden. Dadurch werden die Betriebskosten deutlich gesenkt.

1.1 Problemstellung

Lasershoweinlagen bilden bei vielen Veranstaltungen und Showevents den absoluten technischen Höhepunkt. Viele Zuschauer sind speziell durch diese Form des Lichtes und die damit verbundenen faszinierenden Möglichkeiten besonders beeindruckt und begeistert. Einige haben die Showlasertechnik aufgrund dieser besonderen Faszination zu ihrem Hobby gemacht. Doch leider sind auf dem Markt bisher nur sehr teure Geräte und Systeme zu finden. Wer sich dieses Hobby aussucht, hat allerdings an dem Selbstbau ohnehin mehr Freude als nur am fertig gekauften System. Die einzelnen Komponenten, die zum Aufbau eines Lasersystems benötigt werden, sind zu angemessenen Preisen erhältlich. Nur die eigentliche Steuerung erweist sich meistens als großes Problem: Professionelle, eigenständige und PC-unabhängige Steuerungen kosten mehrere Tausend Euro. Selbst gebaute Lösungen sind quasi immer an einen PC gekoppelte Platinen, die als Kernbauteile DACs mit einer Auflösung von acht Bit für die beiden Achsenkanäle enthalten und von einfachen PC-Programmen angesteuert werden. Zur Kommunikation wird entweder die parallele Schnittstelle oder ein USB-Anschluss verwendet. Ein PC oder Laptop ist dabei zum Ausgeben einer Lasershow immer erforderlich. Teilweise sind Farbdarstellungen mit diesen selbst gebauten Steuerungen und einfacheren Steuerungsprogrammen nach Bedarf auch möglich. Jedoch können die einzelnen Farben in der Regel nur an- bzw. abgeschaltet und nicht in ihrer Intensität eingestellt werden.

1.2 Lösung des Problems

Entwickelt werden soll ein Gerät, das nicht zu teuer ist, sich einfach bedienen lässt und trotzdem Lasershows mit guten Ergebnissen wiedergeben kann. Zudem soll dieses Gerät ohne PC-Anbindung Lasershows von einem in normalen Geschäften erhältlichen Speichermedium (SD-Karte) abspielen können.
Durch die sich rasch weiterentwickelnde Mikroelektronik sind moderne Mikroprozessoren und Speicherbausteine mit hohen Kapazitäten bereits für wenige Euro zu haben. Allerdings sind hochwertige DACs nach wie vor relativ teuer. Hierbei sollte jedoch nicht gespart werden, da die Qualität der DACs, die die Signale für den X- und den Y-Kanal umwandeln, sichtbaren Einfluss auf die Qualität der Laserausgabe hat.
Da rote Laserdioden und grüne DPSS-Laser mit kleinen Leistungen unter 10mW günstiger geworden sind, soll bei dieser Entwicklung auch die Fähigkeit der Farbmischung unterstützt werden. Zu diesem Zweck werden zwei DACs mit einer Genauigkeit von je vier Bit eingebaut. Da blaue DPSS-Laser nach wie vor sehr teuer sind, kommen diese für Hobbyanwendungen zur Zeit quasi nicht in Frage. Daher wird auf die Ausgabe des blauen Farbkanals komplett verzichtet.
Die Farbausgabe soll für die Verwendung an unterschiedlichen Lasersystemen einstellbar sein: Bei einem mehrfarbigen System kann die Farbausgabe aktiviert werden. Wird ein einfarbiges Lasersystem ohne Farbdarstellung benutzt, kann man die Farbausgabe deaktivieren.

1.3 Erfüllung verschiedener Normen und Standards

Da das Gerät mit möglichst vielen bereits vorhandenen Komponenten zusammenarbeiten soll, ist die Einhaltung von Normen und Standards unumgänglich. Zu berücksichtigen sind:

- Die Spezifikation der SD-Karte, die hier als Speicherkarte verwendet werden soll.
- Das Dateisystem FAT16/32 (engl.: File Allocation Table) auf der SD-Karte, damit diese von einem normalen PC mit einem handelsüblichen Kartenleser gelesen und beschrieben werden kann.
- Die Schnittstellenbelegung nach *ILDA* (engl.: International Laser Display Association), damit auch gekaufte Lasersysteme und einzelne Laserkomponenten einfach angesteuert werden können.
- Der Dateiaufbau nach *ILDA*, damit die Erstellung einer Lasershow nicht an ein spezielles Programm gebunden ist.

Alle Normen und Standards betreffen die Kommunikation mit einem PC (SD-Karte) oder dem angeschlossenen Lasersystem.
Sämtliche ausgewählten Bauteile im Inneren des Gerätes müssen zueinander passen oder angepasst werden, damit ein reibungsloser und zuverlässiger Betrieb sichergestellt werden kann.
Da Laserstrahlen eine Gefährdung für die Augen und ab einer gewissen Leistung auch für die Haut darstellen, muss dafür Sorge getragen werden, dass der Laserstrahl im Fehlerfall so schnell wie möglich abgeschaltet wird. Die Software enthält zu diesem Zweck verschiedene Schutzfunktionen durch Abfragen, die im Fehlerfall eine Abschaltung des Laserstrahls bewirken. Auch hierzu ist eine zuverlässige Elektronik unumgänglich.

2 Der Standard der *ILDA*

Damit die auf dem Markt erhältlichen Geräte unterschiedlicher Hersteller problemlos miteinander verbunden werden können, ist ein einheitlicher Standard für die Schnittstelle zwischen dem Steuergerät und dem eigentlichen Lasersystem erforderlich.
Die *ILDA* (engl.: International Laser Display Association) wurde im August 1986 gegründet [1]. Sie gibt den Standard der Schnittstelle zwischen Steuerung und Lasersystem sowie den Aufbau eines speziellen Dateiformates für das Speichern von Lasershows vor. Des Weiteren existieren Richtlinien bezüglich der Qualität und Sicherheit von Lasern und Lasersystemen.

2.1 Schnittstelle zwischen Steuerung und Laserprojektor

Die Verbindung erfolgt über geschirmte Leitungen und Kabel. Als Steckverbinder werden D-SUB-Stecker und -Buchsen mit 25 Polen verwendet. Steuergeräte verfügen über eine D-SUB-Buchse als Ausgang, Lasersysteme über einen D-SUB-Stecker als Eingang.
Da die analogen Steuersignale symmetrisch (differentiell) übertragen werden, können auch Leitungslängen von mehreren zehn Metern ohne auffällige Qualitätsverluste überbrückt werden.
Die Pinbelegung der Schnittstellensteckverbindung ist aus folgender Tabelle ersichtlich:

Funktion	Pin	Pin	Funktion
X +	1	14	X -
Y +	2	15	Y -
Intensität +	3	16	Intensität -
Interlock A	4	17	Interlock B
R +	5	18	R -
G +	6	19	G -
B +	7	20	B -
User-defined Signal 1 +	8	21	User-defined Signal 1 -
User-defined Signal 2 +	9	22	User-defined Signal 2 -
User-defined Signal 3 +	10	23	User-defined Signal 3 -
User-defined Signal 4 +	11	24	User-defined Signal 4 -
Rückmeldung	12	25	Masse
Shutter	13		

Tabelle 1: Pinbelegung der D-SUB-Steckverbinder nach *ILDA* [2]

Abbildung 1: D-SUB-Buchse und -Stecker (25-polige Version)

Die benutzerdefinierten Signale (User-defined Signals) werden nur selten und dann auch nur für Spezialanwendungen verwendet. In der Regel werden diese nicht benutzt und im Steuergerät über 75Ω-Widerstände mit Masse verbunden. Im Lasersystem bleiben diese Leitungen meistens offen.

Beispiele für Sonderanwendungen sind die Kontrolle des Strahldurchmessers, ein zweiter X-Kanal (für Stereoscanner; beide Scanner bekommen dieselben Y-Koordinaten, aber unterschiedliche X-Koordinaten) oder zusätzliche Farbkanäle wie z.B. ein dunkleres Blau, ein Gelb oder Cyan. Da diese genannten Sonderfunktionen in einem günstigen System nicht vorhanden sind, werden sie nicht genauer betrachtet.
Die Rückmeldeleitung liefert dem Steuergerät die Information, ob der Laserstrahl auch tatsächlich das Lasersystem verlässt oder ob im System ein Defekt vorliegt. Dieses Signal wird allerdings nur von den wenigsten Systemen verwendet bzw. unterstützt oder für eine andere Form der Rückmeldung genutzt.

2.1.1 Die Signale X+/- und Y+/-

Die Signale, die die Position des Laserstrahls festlegen, werden symmetrisch (differentiell) übertragen. Es handelt sich um bipolare, analoge Signale. Die differentielle Spannung beträgt maximal ±10V. Die Signale werden ohne Filter oder zusätzliche Verstärker direkt an die Treiber der beiden Scannerachsen geleitet. Allerdings verfügen nicht alle Treiber über differentielle Eingänge, so dass das Signal auf zwei verschiedene Arten zugeführt werden kann:

1. Nur die Leitungen X+ bzw. Y+ werden verwendet, während die Leitungen X- bzw. Y- offen bleiben oder mit Masse verbunden werden. Diese Möglichkeit eignet sich nur für kurze Leitungslängen, da hier keine symmetrische Übertragung mehr stattfindet und somit Störungen direkt sichtbar werden. Bei längeren Leitungen können die Störungen so stark sein, dass die eigentliche Lasershow gar nicht mehr erkannt wird!
2. Signale auf den Leitungen X- bzw. Y- werden mit Hilfe eines Operationsverstärkers invertiert und anschließend mit einem weiteren Operationsverstärker zu den Signalen X+ bzw. Y+ hinzuaddiert. Die Störungen, die sich innerhalb des langen Leitungsweges zu den einzelnen Signalen hinzuaddiert haben, fallen somit fast komplett weg. Allerdings ist die Amplitude der Signale doppelt so hoch. Daher muss noch ein zusätzlicher Verstärker mit dem Verstärkungsfaktor 0,5 eingesetzt werden. Diese Möglichkeit ist zwar mit einem großen Aufwand verbunden, jedoch treten dann auch bei größeren Leitungslängen kaum Störungen beim Empfänger auf. Diese Möglichkeit kann praktisch immer verwendet werden, da jede Lasersteuerung, deren Ausgang die Schnittstellendefinition nach *ILDA* verwendet, die Achsensignale X und Y in symmetrischer Form bereitstellt.

Die nachfolgende Tabelle zeigt die Zuordnung der projizierten Strahlposition zu den maximalen Leitungspegeln der Achsenkanäle aus der Sicht vom Laserprojektor zur Projektionsebene:

X+	X-	X_{diff}	Y+	Y-	Y_{diff}	Strahlposition
+5V	-5V	+10V	0V	0V	0V	rechts
-5V	+5V	-10V	0V	0V	0V	links
0V	0V	0V	0V	0V	0V	Mitte
0V	0V	0V	+5V	-5V	+10V	oben
0V	0V	0V	-5V	+5V	-10V	unten
+5V	-5V	+10V	+5V	-5V	+10V	rechts oben
+5V	-5V	+10V	-5V	+5V	-10V	rechts unten
-5V	+5V	-10V	+5V	-5V	+10V	links oben
-5V	+5V	-10V	-5V	+5V	-10V	links unten

Tabelle 2: Zuordnung der Strahlpositionen zu den Pegeln der X- und Y-Signale (nach [2])

2.1.2 Intensität

Das Intensitätssignal dient der Helligkeitsregelung des Laserstrahls. Das Signal wird ebenfalls symmetrisch übertragen, jedoch liegt der Pegel nur halb so hoch wie bei den X- und Y-Signalen, nämlich bei ±2,5V. Die differentielle Spannung (zwischen Intensität + und Intensität -) beträgt 0V bis +5V. Das Intensitätssignal ist somit ein unipolares Analogsignal.
Manche Lasersysteme, insbesondere einfarbige Lasersysteme mit analogem Blanking, verwenden allein das Intensitätssignal zum Steuern der Laserhelligkeit und keinen der drei Farbkanäle. In der Regel wird das Signal mit einer Genauigkeit von acht Bit ausgegeben, was 256 Helligkeitsabstufungen bedeutet.

2.1.3 Interlock A und B

Bei den Leitungen Interlock A und B handelt es sich nicht um Signale, sondern um eine Kabelbruchsicherung. Das Lasersystem versorgt eine der beiden Leitungen (hier Interlock A) mit +5V. Im Steuergerät sind die beiden Leitungen Interlock A und B direkt miteinander verbunden. Das Lasersystem erwartet auf der anderen Interlock-Leitung (hier Interlock B) „seine“ eigenen +5V. Fehlt die Spannung, ist die Verbindung zwischen Steuergerät und Lasersystem unterbrochen und der Laserstrahl wird sofort abgeschaltet bzw. durch einen Shutter unterbrochen.

2.1.4 Die Farbsignale R+/-, G+/- und B+/-

Die drei Farbanteile werden ebenfalls symmetrisch übertragen. Manche Steuergeräte schalten die Farben „nur“ an bzw. aus. Andere verwenden DACs mit einer Genauigkeit von bis zu acht Bit pro Farbe.
Es handelt sich, wie auch beim Intensitätssignal, um unipolare Analogsignale mit einem maximalen Pegel von ±2,5V gegen Masse. Die differentielle Spannung beträgt 0V bis +5V.
Die drei Farbkanäle werden für das sogenannte RGB-Blanking (Rot / Grün / Blau) verwendet. Jeder einzelne Farbkanal stellt die Intensität eines Farbanteils von 0% bis 100% ein. Manchmal wird die Intensität der Farbkanäle zusätzlich durch das Intensitätssignal beeinflusst. Dies hängt vom Steuergerät und vom Lasersystem ab und ist nicht erforderlich.

2.1.5 Shutter

Das Signal für den Shutter wird nicht differentiell übertragen. Dieses Signal ist ein digitales Signal und dient zum An- und Abschalten des Laserstrahls. Der Pegel beträgt entweder +5V oder 0V (gegen Masse). Bei +5V ist der Laserstrahl sichtbar, der Shutter ist offen. Bei 0V ist der Laserstrahl nicht sichtbar, der Shutter ist geschlossen. Der Shuttereingang des Lasersystems sollte über einen Pulldown-Widerstand mit Masse verbunden sein, damit sichergestellt werden kann, dass bei einer fehlerhaften Verbindung zwischen Lasersystem und Steuergerät kein unkontrollierter Laserstrahl aus dem Lasersystem austreten kann.
In modernen Lasersystemen, die als Laserquelle Halbleiterlaser (Diodenlaser oder DPSS-Laser) verwenden, ist ein mechanischer Shutter nicht mehr erforderlich, da die Halbleiterlaser ohne großen technischen Aufwand sehr schnell ein- und ausgeschaltet werden können.
Das Shuttersignal wird immer mit ausgegeben, da einige Lasersysteme das Shuttersignal als zusätzliche Sicherung verwenden: Der Laser wird nur dann freigegeben, wenn das Shuttersignal es erlaubt.

2.2 Aufbau einer *.ild-Datei

Neben der einheitlichen Schnittstelle definiert die *ILDA* auch ein einheitliches Dateiformat zum Speichern von Lasershows. Die Dateiendung ist *.ild.
Die Dateien unterstützen eine Auflösung von 16 Bit für den X- und den Y-Kanal sowie eine Auflösung von 8 Bit für jeden der drei Farbkanäle. Somit lassen sich theoretisch bis zu 16.777.216 Farben mischen. In der Praxis wird von dieser Farbtiefe jedoch kein Gebrauch gemacht.
Viele Hersteller von Lasershowsoftware verwenden eigene Dateiformate. Damit Lasershows untereinander ausgetauscht und im- bzw. exportiert werden können, ist ein zusätzliches, einheitliches Dateiformat unverzichtbar. Dieses Dateiformat enthält die einzelnen Frames (Bilder) einer Lasershow. Ein Frame wird durch einen Header mit einer Größe von 32 Bytes eingeleitet. Im Anschluss folgen die einzelnen Punkte eines Frames. Ein Punkt besteht aus sechs bzw. acht Bytes und enthält die einzelnen Koordinaten X, Y und oft auch Z sowie einen Status Code. Nach dem Ende eines Frames folgt der Header des nächsten Frames. Der letzte Frame enthält keinen Punkt mehr. Daher ist der Wert „Total Points“ dieses Frames auch null.

Byte(s)	Inhalt	Beschreibung
Frame Header		
1 - 4	„I“, „L“, „D“, „A“	Diese vier Bytes enthalten die ASCII-Zeichen *ILDA* und dienen der Identifizierung des Frame Headers.
5 - 7	0, 0, 0	Diese drei Bytes werden nicht benutzt, müssen aber immer 0 sein.
8	Format Code	Der Format Code gibt an, in welchem Format der aktuelle Frame aufgebaut ist: 0 = Frame liegt im 3D-Format vor. Ein Punkt besteht in der Datei aus acht Bytes und enthält die Koordinaten für X, Y und Z sowie den Status Code. 1 = Frame liegt im 2D-Format vor. Ein Punkt besteht in der Datei aus sechs Bytes und enthält die Koordinaten für X und Y sowie den Status Code. 2 = Es handelt sich nicht um einen Frame, sondern um eine Farbtabelle. Als Standard wird das 3D-Format mit dem Format Code 0 verwendet.
9 - 16	Frame Name	In diesen acht Bytes wird der Name des Frames gespeichert. Einige Programme erlauben dem Benutzer die eigene Eingabe einer Bezeichnung, andere führen automatisch eine Bezeichnung, die eine Nummer beinhaltet, ein.
17 - 24	Company Name, wird oft als Erweiterung des Frame Namens verwendet	In diesen acht Bytes wird der Name der Firma gespeichert. Oft werden diese acht Bytes auch als Erweiterung des Frame Namens verwendet. Somit stehen insgesamt 16 Bytes für eine Kennzeichnung des Frames zur Verfügung.
25 - 26	Total Points	Diese beiden Bytes enthalten die Anzahl der in dem Frame vorhandenen Punkte. Da in zwei Bytes 16 Bits gespeichert werden können, kann ein Frame aus bis zu 65535 Punkten bestehen. Enthalten beide Bytes eine Null, handelt es sich um den letzten Frame der Datei. Auf den Header folgen keine Koordinaten mehr.

Byte(s)	Inhalt	Beschreibung
27 - 28	Frame Number	Diese beiden Bytes enthalten die Nummer des Frames. Der Wert kann zwischen 0 und 65535 liegen. Die Framenummer wird fortlaufend durchgezählt. Wird beim Editieren ein Frame gelöscht, kopiert oder ein neuer eingefügt, wird die Nummer bei jedem einzelnen Frame aktualisiert.
29 - 30	Total Frames	Diese beiden Bytes enthalten die Anzahl der in der Datei vorhandenen Frames. Der Wert kann zwischen 1 und 65535 liegen. Es muss mindestens ein kompletter Frame in der Datei vorhanden sein.
31	Scanner Head	Es gibt Programme und Steuerungsgeräte, die parallel mehrere Lasersysteme mit unterschiedlichen Frames ansteuern können. Diese Systeme benötigen die Information, auf welchem Lasersystem der Frame ausgegeben werden soll. Der Wert kann zwischen 0 und 255 liegen, der typische Wert ist 0.
32	Future	Reserviert für zukünftige Erweiterungen, muss immer 0 sein!
Daten der Koordinaten		
33 - 34	X-Koordinate	Das erste Bytepaar eines Punktes enthält den Wert der X-Koordinate mit einer Auflösung von 16 Bit.
35 - 36	Y-Koordinate	Das zweite Bytepaar eines Punktes enthält den Wert der Y-Koordinate mit einer Auflösung von 16 Bit.
37 - 38	Z-Koordinate	Das dritte Bytepaar eines Punktes enthält den Wert der Z-Koordinate mit einer Auflösung von 16 Bit. Dieser Wert ist normalerweise immer 0. Dieses Bytepaar ist nur vorhanden, wenn der Format Code 0 (3D-Format) ist. Im 2D-Format besteht ein Punkt aus sechs Bytes.
39 - 40	Status Code	Das vierte und letzte Bytepaar enthält den Status Code. Im 2D-Format ist dieses Bytepaar das dritte Bytepaar des Punktes.
41 - N	Nächste X-Koordinate	Hinter dem Status Code eines Punktes folgt direkt die X-Koordinate des nächsten Punktes usw.
N+1	Nächster Frame Header	Hinter dem Status Code des letzten Punktes eines Frames folgt der Header des nächsten Frames.

Tabelle 3: Aufbau einer *.ild-Datei (nach [3])

Direkt hinter dem Status Code eines Punktes folgt die X-Koordinate des nächsten Bildpunktes. Ist der letzte Bildpunkt erreicht, folgt hinter dem Status Code des Punktes der Header des nächsten Frames mit der Signatur „ILDA“.
Der Status Code besteht aus zwei Bytes und beinhaltet die Farbinformationen sowie die Blankinginformation des aktuellen Punktes.

Bit(s)	Inhalt	Beschreibung
0 - 7	Color Number	Die untersten acht Bits enthalten die Farbnummer, mit dem der Punkt angezeigt werden soll. Der Wert liegt zwischen 0 und 255.
8 - 13	reserviert	Diese Bits sind reserviert und müssen immer den Wert 0 enthalten!

Bit(s)	Inhalt	Beschreibung
14	Blanking Bit	Dieses Bit gibt an, ob der Laserstrahl bei der Wiedergabe des Punktes sichtbar (Bit 14 = 0) oder nicht sichtbar (Bit 14 = 1) sein soll.
15	Last Point Bit	Dieses Bit ist nur bei dem letzten Punkt eines Frames auf eins gesetzt. Für alle anderen Punkte ist dieses Bit auf null zu setzen.

Tabelle 4: Status Code eines Punktes (nach [3])

Da praktisch alle einfacheren und auch viele der aufwendigeren Lasershows die Standard-Farbpalette des Herstellers *Pangolin Laser Software* verwenden, wird auf das Einlesen einer separaten Farbtabelle aus der *.ild-Datei verzichtet und auf deren Aufbau nicht weiter eingegangen.
Die Color Number verweist auf den Eintrag in einer Tabelle, in der die drei Farbanteile mit einer Auflösung von je acht Bit hinterlegt sind. Diese Farbtabelle kann theoretisch für jeden Frame andere Werte erhalten (vor jedem Frame kann eine neue Farbtabelle stehen), was jedoch in der Praxis absolut unüblich ist.

Die folgende Abbildung verdeutlicht beispielhaft den Aufbau einer *.ild-Datei in der Ansicht eines Hex-Editors:

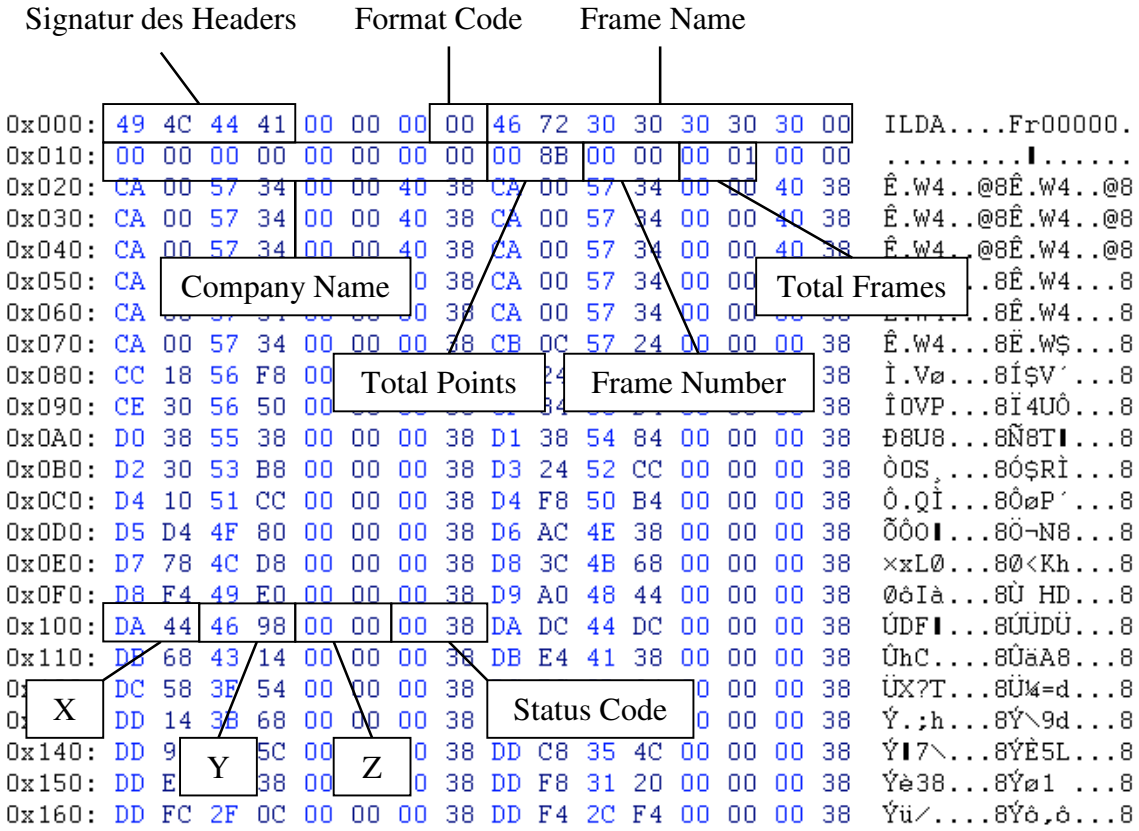


Abbildung 2: Beispiel der Datei *kreis.ild* (siehe CD-ROM)

In dieser Datei ist ein Frame gespeichert. Laut Format Code handelt es sich um einen Frame im 3D-Format. Es ist zu erkennen, dass die Z-Koordinate nicht verwendet wird und immer den Wert null aufweist.

3 SD-Karte

Heutzutage gibt es viele verschiedene Speicherkarten auf dem Markt. Die Auswahl, welche Speicherkarte für dieses Projekt verwendet werden soll, war jedoch relativ einfach: Die SD-Karte (Secure Digital Card) ist eine schnelle, kompakte und zuverlässige Speicherkarte, die bei sehr vielen Geräten des normalen Lebens (z.B. Handy, mp3-Player, Navigationsgerät usw.) Anwendung findet. Außerdem sind SD-Karten relativ preiswert, benötigen nur wenige Pins zum Anschluss an weitere Hardware und unterstützen das Dateisystem FAT.
Das Datenblatt der SD-Karte umfasst 113 Seiten [4]. Von der Fülle an Informationen werden allerdings nur einige Punkte benötigt. Diese werden in diesem Kapitel behandelt.

3.1 Anschluss der SD-Karte

Die SD-Karte wird über insgesamt neun Pins angeschlossen. Die Karte kann mittels eines kleinen Schiebers vor dem Schreibzugriff geschützt werden. Da dieser Schieber jedoch keine direkte elektronische Verbindung zu der karteninternen Elektronik hat, kann dieser Schreibschutz leicht umgangen werden. Der Kartenleser, in den die Karte eingelegt wird, muss die Position des Schiebers überprüfen. Dies geschieht meistens durch einen kleinen Kontakt. Ein Schreibzugriff kann praktisch jederzeit passieren und wird nur von entsprechender Software unterbunden. Da das im Rahmen dieses Projektes zu entwickelnde Gerät keinerlei Schreibzugriffe auf die SD-Karte ausführen soll, wird die Schreibschutzüberprüfung auch nicht mit in die Softwareentwicklung mit einbezogen.

Abbildung 3: SD-Karte

Da die SD-Karte zwei verschiedene Betriebsmodi unterstützt, haben einige Pins doppelte Funktionsbelegung, abhängig vom gewählten Modus. Wird die Karte im SD-Card-Modus betrieben, so werden die Daten über vier bidirektionale Datenleitungen parallel übertragen. Ein Byte benötigt somit nur zwei Taktimpulse der Clock-Leitung. Die maximal zulässige Taktfrequenz beträgt 25 MHz. Somit lassen sich bis zu 12,5MB pro Sekunde übertragen. Die Datenrichtung der vier Datenleitungen kann gewechselt werden, es handelt sich um den sogenannten SD-Bus.
Im SPI-Modus steht für jede Datenrichtung nur eine Leitung zur Verfügung. Da auch hier die maximal zulässige Taktfrequenz 25 MHz beträgt, lassen sich so nur bis zu 3,125MB pro Sekunde übertragen. Diese Werte sind jedoch nur theoretisch erreichbar, denn neben Nutzdaten werden auch Kommandos zum Steuern der Kommunikation übertragen!

Pin-Nummer	Funktion im SD-Card-Modus	Funktion im SPI-Modus
1	Card Detect / Data Line (Bit 3)	Chip Select
2	Command / Response	Data In (Host to Card)
3	GND	
4	Versorgungsspannung, typ. +3,3VDC	
5	Clock	
6	GND	
7	Data Line (Bit 0)	Data Out (Card to Host)
8	Data Line (Bit 1)	reserviert
9	Data Line (Bit 2)	reserviert

Tabelle 5: Pinbelegung der SD-Karte (nach [4])

Im SD-Card-Modus müssen die Leitungen der Pins 1, 2, 7, 8 und 9 mit je einem Pullup-Widerstand mit 10KΩ bis 100KΩ an die Betriebsspannung angeschlossen werden, damit der SD-Bus nie einen unbestimmten Zustand einnehmen kann. Dieser Zustand tritt auf, wenn keine Karte am Bus angeschlossen ist oder alle angeschlossenen Karten im hochohmigen Zustand (inaktiv) sind.
Im SPI-Modus (engl.: Serial Peripheral Interface) sollte Pin 7 mit einem Pulldown-Widerstand gegen Masse angeschlossen werden, damit kein unbestimmter Zustand auftritt, wenn keine SD-Karte angeschlossen ist. Dieses ist jedoch nicht vorgeschrieben.

3.2 Elektrische Eigenschaften

Die typische Betriebsspannung einer SD-Karte beträgt 3,3V. Auch die Datenleitungen und die Taktleitung arbeiten typischerweise mit einem Pegel von 0V bzw. 3,3V.
Die Betriebsspannung muss zwischen 2,0V und 3,6V liegen. Jedoch sind unterhalb von 2,7V keine Speicherzugriffe und kein normaler Betrieb möglich, sondern nur die Commands CMD0 (Reset), CMD15 (Karte wird inaktiv, nicht SPI-Modus), CMD55 (leitet ein ACMD (engl.: Application Command) ein) und ACMD41 (Karte sendet ihr OCR (engl.: Operating Condition Register)).
Beim Einschalten muss die Betriebsspannung innerhalb von 250ms von 0V auf die minimale Betriebsspannung von 2,0 V angestiegen sein (Power-up Time), damit der karteninterne Speichercontroller zuverlässig gestartet wird.
Wie hoch die Stromaufnahme ist, hängt vom jeweiligen Betriebsmodus (lesen oder schreiben) ab. Die Werte sind karten- und herstellerabhängig.
Die SD-Karten des Herstellers *SanDisk* mit Kapazitäten zwischen 16MB und 1GB benötigen 65mA beim Lesezugriff und 75mA beim Schreibzugriff.
Die Stromaufnahme kann außerdem dem CSD-Register (engl.: Card Specific Data, Beschreibung siehe unten) der SD-Karte entnommen werden: Für Lese- und Schreibzugriff beträgt die Stromaufnahme bei maximaler Betriebsspannung 80mA und bei minimaler Betriebsspannung 100mA.
Eingetragen werden können im CSD-Register bei minimaler Betriebsspannung folgende Werte: 500µA, 1mA, 5mA, 10mA, 25mA, 35mA, 60mA und 100mA.
Bei maximaler Betriebsspannung sind andere Werte möglich: 1mA, 5mA, 10mA, 25mA, 35mA, 45mA, 80mA und 200mA.
Die SD-Karte kann während des laufenden Betriebes in den Kartenschacht eingesteckt oder auch entfernt werden, da sie „hot plug“ unterstützt.
Die Spannungspegel der logischen Zustände werden im Kapitel 6 (Hardware) im Zusammenhang mit dem verwendeten Mikroprozessor erläutert.

3.3 Kommunikation mit der SD-Karte im SPI-Modus

Ein direkter Zugriff auf den Datenspeicher der SD-Karte ist nicht möglich. Daher existieren spezielle Befehle (Commands) für die Kommunikation mit dem karteninternen Speichercontroller, der den Zugriff auf den Datenspeicher übernimmt und die Daten entsprechend weiterleitet. Ebenso ist eine Fehlersicherung der Übertragung durch zwei verschiedene CRC integriert. Dabei sind festgelegte Reihenfolgen einzuhalten.

Wird die SD-Karte zum Beispiel durch Einlegen in den Kartenschacht an die Betriebsspannung angeschlossen, befindet sie sich automatisch im hier nicht erwünschten SD-Card-Modus. Um den Betriebsmodus der SD-Karte in den SPI-Modus zu wechseln, ist eine fest vorgegebene Sequenz nötig [4]:

1. Die Karte wird oder bleibt deaktiviert (Pin 1 liegt auf dem Betriebsspannungspegel).

2. Es müssen mindestens 74 Takte über die Clock-Leitung (Pin 5) an die SD-Karte gesendet werden, während Pin 2 (Data In) „high“ ist (Betriebsspannungspegel).

3. Das Command CMD0 (entspricht 0x40) wird mit gültiger Checksumme (nach CRC7, siehe 3.4) gesendet. Die Checksumme ist hier immer 0x95, da das CMD0 keine weiteren Daten akzeptiert. Die folgenden sechs Bytes werden an die SD-Karte gesendet: 0x40, 0x00, 0x00, 0x00, 0x00, 0x95. Im Anschluss befindet sich die SD-Karte im Idle-Zustand.

4. Ist die Rückmeldung ungleich eins, wird das CMD0 mit gelöschter Checksumme (0x00) so oft wiederholt gesendet, bis die Rückmeldung eins ist.

Jetzt arbeitet die SD-Karte im SPI-Modus. Allerdings muss vor einem Zugriff mit dem Command CMD1 der karteninterne Initialisierungsprozess gestartet werden.

3.3.1 Rückmeldetypen der SD-Karte

Die SD-Karte erzeugt als Antwort auf fast jedes Command eine Rückmeldung. Es existieren vier verschiedene Rückmeldetypen, von denen hier allerdings nur einer zur Anwendung kommt und daher näher beschrieben wird: Der Rückmeldetyp R1.

Der Rückmeldetyp R1 ist ein Byte lang und ist damit der kürzeste aller Rückmeldetypen. Außerdem ist er Bestandteil jedes anderen Rückmeldetyps. Die anderen Typen sind R1b, R2 und R3. Alle drei werden von Commands verwendet, die hier nicht benötigt werden.

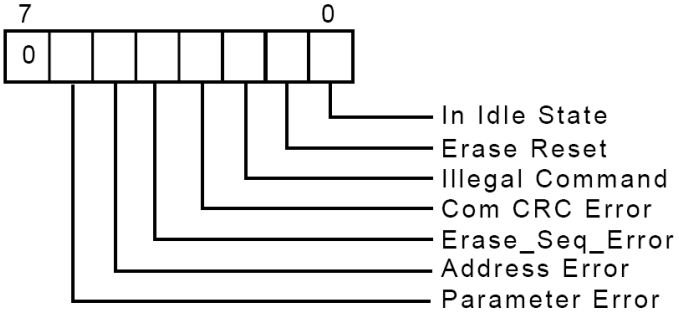


Abbildung 4: Rückmeldetyp R1 [4]

Dem Bit 7 ist keine Fehlerquelle zugeordnet, es ist immer null. Das Byte enthält sieben Statusbits (Bit 0 bis 6). Hat ein Bit den Wert eins, so ist an der zugeordneten Stelle ein Fehler aufgetreten. Nur wenn das komplette Byte den Wert null hat, ist das Command fehlerfrei übertragen und verstanden worden. Andernfalls muss die SD-Karte neu gestartet und in den Idle-Zustand versetzt werden oder einfach das Command erneut gesendet werden. Dieses ist abhängig von der zurückgegebenen Fehlermeldung.

3.3.2 Commands (Befehle)

Ein Command besteht immer aus sechs Bytes, die nacheinander ohne Pause gesendet werden. Nach dem Senden der meisten Commands gibt die SD-Karte eine Rückmeldung. Welchem Typ die Rückmeldung entspricht, hängt vom gesendeten Command ab.
Das erste Byte ist das eigentliche Command. Dieses setzt sich aus dem Offset 0x40 und dem Index des Commands zusammen. Ein Beispiel: Soll das CMD1 gesendet werden, so lautet das erste Byte 0x40 + 1 = 0x41. Die nächsten vier Bytes sind Parameter des Commands wie z.B. eine Adresse. Nicht jeder Command akzeptiert oder erwartet Parameter, die vier Bytes sind dann null. Das sechste und letzte Byte ist die Prüfsumme für die CRC7. Wird die CRC-Prüfung deaktiviert, muss das letzte Byte null sein.

Im SD-Card-Modus stehen 33 unterschiedliche Commands zur Verfügung. Von den Commands können im SPI-Modus nur 22 angewendet werden, die übrigen elf werden nicht unterstützt und liefern eine entsprechende Fehlerrückmeldung (Illegal Command, Bit 2 des Rückmeldetyps R1).
Neben den allgemeinen Commands (CMDx) stehen noch zwölf applikationsspezifische Commands (ACMDx) zur Verfügung, von denen elf im SPI-Modus verwendet werden können. Diese speziellen Commands müssen durch das Command CMD55 eingeleitet werden und behandeln Sicherheitseinstellungen und einige Sonderfunktionen, die hier nicht angewendet werden.

Fünf Commands werden für dieses Projekt benötigt. Alle verwenden den Rückmeldetyp R1.

- CMD0 = 0x40:
  Es löst den karteninternen Reset der SD-Karte aus und versetzt diese in den Idle-Zustand. Nach einem erfolgreichen CMD0 muss als nächstes das CMD1 gesendet werden. Wird anstelle des CMD1 ein beliebiges anderes Command gesendet, gibt die SD-Karte eine Fehlerrückmeldung (In Idle State, Bit 0 des Rückmeldetyps R1). Parameter werden nicht akzeptiert.

- CMD1 = 0x41:
  Es startet den Initialisierungsprozess der SD-Karte. Parameter werden nicht akzeptiert. Dieses Command muss als erstes nach einem CMD0 gesendet werden.

- CMD9 = 0x49:
  Es fordert von der SD-Karte das CSD-Register an. Parameter werden nicht akzeptiert. Nach der Rückmeldung wird das CSD-Register byteweise von der SD-Karte ausgegeben.

- CMD17 = 0x51:
  Es liest im „Block Read Mode“ einen kompletten Block von der SD-Karte. Die Blockgröße ist im CSD-Register eingetragen und beträgt meistens 512 Bytes. Das

CMD17 benötigt als Parameter die Startadresse des Blockes, der gelesen werden soll. Die Startadresse weist auf ein beliebiges Byte auf der SD-Karte.

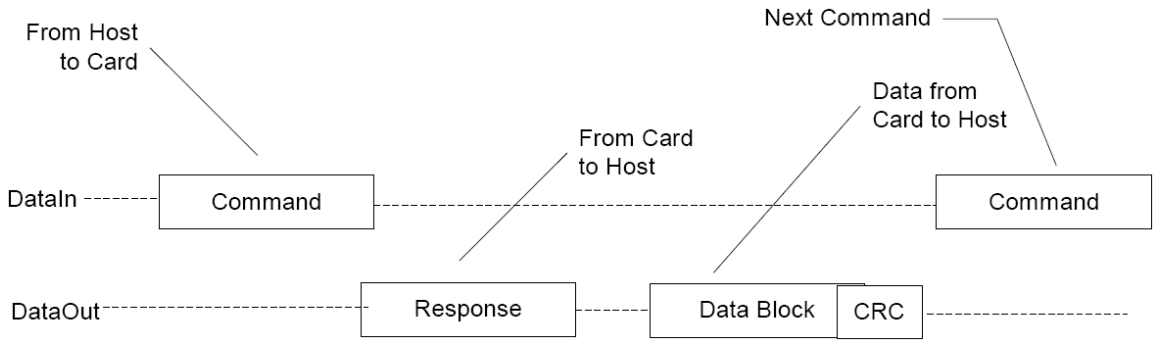


Abbildung 5: Auslesen eines kompletten Blockes von der SD-Karte [4]

Nach der Rückmeldung wird der adressierte Block byteweise von der SD-Karte ausgegeben. Im Anschluss werden immer zwei CRC-Bytes übertragen, unabhängig davon, ob die CRC-Prüfung aktiviert oder deaktiviert ist.

- CMD59 = 0x7B:
  Dieses Command aktiviert bzw. deaktiviert die CRC-Prüfung auf der SD-Karte. Die ersten vier Parameterbytes können beliebig gesetzt werden, nur die Bitposition 0 im vierten Parameterbyte ist von Bedeutung: Ist das Bit null, wird die CRC-Prüfung abgeschaltet. Ist das Bit eins, wird die CRC-Prüfung eingeschaltet.

3.3.3 CRC-Prüfung

Im SD-Card-Modus ist die CRC-Prüfung (engl.: Cyclic Redundancy Check) immer aktiv. Wird die Karte im SPI-Modus betrieben, ist die CRC-Prüfung standardmäßig deaktiviert.
Die CRC-Prüfung kann auch während des Betriebes aktiviert bzw. deaktiviert werden.

Die Commands, die Rückmeldungen vom Typ R3 sowie das CSD- und CID-Register (engl.: Card Identification Register) werden mit einer CRC7 gesichert. Das Generatorpolynom lautet:

$$g_{CRC7}(x) = x^7 + x^3 + 1$$

Gleichung 1: Generatorpolynom der SD-Karte für die CRC7 [4]

Die übertragenen Daten im „Block Read Mode“ bzw. „Block Write Mode“ werden mit einer CRC16 gesichert. Im SD-Card-Modus mit vier Datenleitungen verwendet jede Datenleitung eine separate CRC16. Das Generatorpolynom lautet:

$$g_{CRC16}(x) = x^{16} + x^{12} + x^5 + 1$$

Gleichung 2: Generatorpolynom der SD-Karte für die CRC16 [4]

Die CRC wird bei diesem Projekt nicht verwendet. Während der kompletten Entwicklungs- und Testphase sind mit abgeschalteter CRC nie Fehler aufgefallen, weder gravierende noch harmlose. Auch aus Gründen der Geschwindigkeit bleibt die CRC deaktiviert, da diese sonst laufend mit dem Mikroprozessor aufwendig berechnet werden müsste.

3.4 CSD-Register

Im CSD-Register (engl.: Card Specific Data) sind alle wichtigen Eigenschaften und Parameter der SD-Karte gespeichert. Bevor mit der SD-Karte erfolgreich gearbeitet werden kann, muss dieses Register ausgelesen und die für die Kommunikation wichtigen Einstellungen vorgenommen werden. Das CSD-Register ist ein 128 Bit großes Register und kann nur in kompletter Länge von der Karte abgerufen werden (CMD9). Die einzelnen Informationen belegen ein oder mehrere Bits und sind über die Grenzen zwischen den Bytes hinweg abgelegt, so dass einige Informationen aus mehreren Bytestücken zusammengesetzt werden müssen.
Die Bit- und Bytenummern in der folgenden Tabelle beginnen jeweils beim Zahlenwert null, wie es in der Digitaltechnik üblich ist.

Parameter	Größe in Bytes	Bitnummer(n) im CSD-Register	Bytenummer(n) im CSD-Register	Bitnummer(n) im Byte
Blockgröße beim Block Read Mode, Parameter *READ_BL_LEN*	4	80 - 83	10	0 - 3
Kartengröße, Parameter *C_SIZE*	12	62 - 73	7	6 - 7
			8	0 - 7
			9	0 - 1
Max. Stromaufnahme beim Lesen bei VDD min.	3	59 - 61	7	3 - 5
Max. Stromaufnahme beim Lesen bei VDD max.	3	56 - 58	7	0 - 2
Max. Stromaufnahme beim Schreiben bei VDD min.	3	53 - 55	6	5 - 7
Max. Stromaufnahme beim Schreiben bei VDD max.	3	50 - 52	6	2 - 5
Multiplikator, Parameter *C_SIZE_MULT*	3	47 - 49	5	7
			6	0 - 1
Dateisystemgruppe	1	15	1	7
Dateisystem	2	10 - 11	1	2 - 3

Tabelle 6: Ausschnitt aus dem CSD-Register [4]

Der Parameter Kartengröße (*C_SIZE*) enthält nicht die tatsächliche Kartengröße. Diese muss mit einer Formel aus den drei dem CSD-Register entnommenen Parametern *READ_BL_LEN*, *C_SIZE* und *C_SIZE_MULT* berechnet werden.
Die nachfolgende Formel zur Berechnung der Kartengröße in Bytes ist nicht an eine bestimmte Blockgröße gebunden und in der programmierten Software implementiert.

$$\text{Größe (in Bytes)} = C\_SIZE * 2^{(C\_SIZE\_MULT + 2)} * READ\_BL\_LEN$$

Gleichung 3: Berechnung der Kartengröße in Bytes (nach [4])

Das Bit der Dateisystemgruppe muss immer null sein und ist für Erweiterungen reserviert.
Das Dateisystem kann entweder ein festplattenähnliches System mit Partitionstabelle (0), ein diskettenähnliches DOS-FAT mit Bootsektor ohne Partitionstabelle (1), ein universelles Dateisystem (2) oder ein unbekanntes Dateisystem (3) sein.
Bei diesem Projekt wird das festplattenähnliche System mit Partitionstabelle (0) verwendet.

4 FAT-Dateisystem

Das Dateisystem FAT (engl.: File Allocation Table) ist ein weit verbreitetes Dateisystem, das nicht nur von den Betriebssystemen des Softwareherstellers *Microsoft®*, sondern auch von z.B. vielen unixbasierten Betriebssystemen unterstützt wird. Das Dateisystem FAT wurde 1980 eingeführt [5]. Seitdem wurde es in mehreren Schritten weiterentwickelt, um den Einsatz der immer schneller wachsenden Speicherkapazitäten zu ermöglichen.
Die offizielle Spezifikation des Dateisystems wird nicht benötigt, da alle relevanten Informationen, die für dessen Arbeitsweise und Verständnis erforderlich sind, aus mehreren Online-Dokumentationen zusammengetragen werden können ([5], [6], [7], [8]).
FAT ist ein sehr einfach strukturiertes Dateisystem. Es besteht aus der Volume ID (siehe 4.2), Zuordnungstabelle(n) (die eigentliche FAT) und dem Datenbereich. Der vierte wichtige Bestandteil ist das Root-Verzeichnis (Stammverzeichnis), das das erste Verzeichnis der Partition darstellt.
Es wird unterschieden zwischen primärer, erweiterter und externer Partition:

- Eine primäre Partition ist direkt im Master Boot Record (siehe 4.1) eingetragen und enthält keine weitere Partitionsunterteilung.
- Eine erweiterte Partition ist ebenfalls direkt im Master Boot Record eingetragen, jedoch bildet diese nur einen Rahmen für die externen Partitionen.
- Eine externe Partition kann nur innerhalb einer erweiterten Partition angelegt werden. Der Aufbau einer externen Partition entspricht der einer primären Partition.

Jede Partition enthält ein komplett eigenständiges Dateisystem, also separate Zuordnungstabellen (FAT) und ein eigenes Stammverzeichnis. Somit ist es möglich, auf einem Datenträger mehrere Partitionen mit verschiedenen Dateisystemen anzulegen.
Die Adressierung auf dem Datenträger erfolgt immer in Sektoren. Cluster stellen eine Gruppierung mehrerer aufeinander folgender Sektoren im Datenbereich einer Partition dar und sind rein virtuell. Die Adresse eines Clusters muss vor dem Zugriff in die Sektoradresse, bei der der Cluster beginnt, umgerechnet werden. Ein Sektor umfasst in der Regel 512 Bytes [5].

4.1 Master Boot Record (MBR)

Der Master Boot Record ist immer auf der ersten Spur (Adresse 0x00, 0x0000 oder 0x00000000) eines Datenträgers gespeichert, unabhängig davon, ob es sich um eine Festplatte oder eine Speicherkarte handelt. Es ist der erste Sektor, der von einem Datenträger nach dem Einschalten gelesen wird.

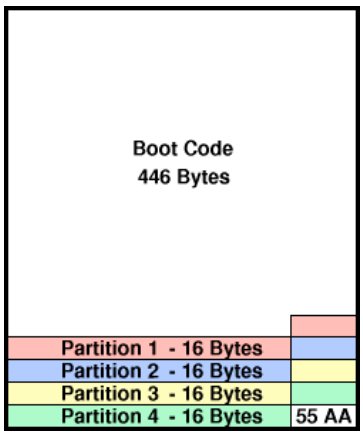


Abbildung 6: Aufbau des Master Boot Records [6]

Der MBR kann 446 Bytes ausführbaren Code enthalten, der direkt im Anschluss des BIOS (engl.: Basic Input Output System) eines PCs ausgeführt wird. Der Vorgang wird als Booten bezeichnet.
Hinter den 446 Bytes, die für ausführbaren (bootfähigen) Code reserviert sind, folgt eine Partitionstabelle mit vier Einträgen. Die letzten beiden Bytes des MBR enthalten die Zahlen 0x55 und 0xAA. Diese beiden Bytes werden vom BIOS zur Identifikation eines gültigen Bootsektors verwendet [5].
Jeder der vier Partitionseinträge ist 16 Bytes groß. Die entscheidenden Informationen sind der Typencode und die Adresse, an der sich die Volume ID der FAT-Partition befindet. Dieses ist die Startadresse der Partition auf dem Datenträger, sie zeigt auf einen Sektor.
Der Typencode ist im fünften Byte und die Startadresse im neunten bis zwölften Byte eines jeden Partitionseintrages gespeichert. Ist die Startadresse der Partition gefunden, muss die Volume ID der Partition ausgelesen und ausgewertet werden.
Der Typencode gibt Aufschluss darüber, um was für ein Dateisystem es sich handelt [7], [8]:

- 0x04 (Partition < 32MB) oder 0x06 (Partition ≥ 32MB): Es handelt sich um eine FAT16-Partition.
- 0x0B oder 0x0C (wie 0x0B, jedoch andere Adressierung durch das BIOS durch LBA (engl.: Logical Block Adressing)): Es handelt sich um eine FAT32-Partition.

4.2 Volume ID und Aufbau einer Partition

Die Volume ID (engl.: Volume Identification) ist die Bezeichnung des ersten Sektors einer Partition. Der grundlegende Aufbau einer Partition (primäre und externe Partition) ist bei FAT16 und FAT32 ähnlich. Der Volume ID folgen reservierte Sektoren, deren Anzahl vom Speichermedium abhängt und in der Volume ID eingetragen ist. Hinter den reservierten Sektoren befindet sich bei FAT16 in der Regel eine Zuordnungstabelle. FAT32 verwendet zur Sicherheit in der Regel noch eine zweite Zuordnungstabelle. Dabei ist der Inhalt bei beiden Zuordnungstabellen identisch, beide müssen bei einem Schreibzugriff aktualisiert werden. Die zweite Zuordnungstabelle dient der Sicherheit. Ist die erste beschädigt, können Dateien mit Hilfe der zweiten Zuordnungstabelle wiederhergestellt werden. Als Letztes folgt der Datenbereich, der in Cluster eingeteilt ist. Es ist üblich, dass am Ende des Datenbereiches einige Bytes oder Sektoren frei bleiben, die zusammen nicht mehr einen ganzen Cluster bilden können. Auf diesen Bereich kann das Dateisystem nicht zugreifen. Der daraus resultierende Speicherplatzverlust liegt im Bereich einiger Bytes bis zu wenigen Kilobytes und kann somit vernachlässigt werden.

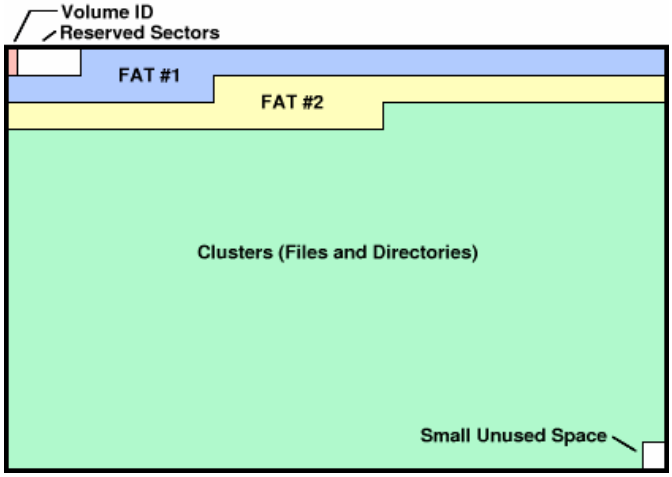


Abbildung 7: Aufbau einer Partition beim Dateisystem FAT32 [6]

Die Volume ID beinhaltet alle wichtigen Parameter der Partition. Aufbau und Inhalt der Volume ID sind bei FAT16 und FAT32 teilweise unterschiedlich.
Der Inhalt der ersten 36 Bytes (0x24) der Volume ID ist bei FAT16 und FAT32 identisch. Die wichtigen Informationen werden in folgender Tabelle zusammengefasst:

Position in der Volume ID	Größe (in Bytes)	Inhalt
0x0B	2	Bytes pro Sektor
0x0D	1	Sektoren pro Cluster
0x0E	2	Anzahl der reservierten Sektoren im Anschluss an die Volume ID (Volume ID wird mitgezählt)
0x10	1	Anzahl der Zuordnungstabellen FAT
0x11	2	Maximale Anzahl der Einträge im Stammverzeichnis
0x16	2	Sektoren pro Zuordnungstabelle
0x20	4	Gesamtanzahl der Sektoren in der Partition

Tabelle 7: Inhalt der ersten 36 Bytes der Volume ID (nach [5], [7], [8])

Ein Sektor könnte aus 512, 1024, 2048 oder 4096 Bytes bestehen. Es werden jedoch nur Sektoren mit einer Größe von 512 Bytes verwendet.
Ein Cluster kann 1, 2, 4, 8, 16, 32 oder 64 Sektoren beinhalten.
Die maximalen Einträge im Stammverzeichnis (0x11) werden nur von FAT16 verwendet, ebenso die Sektorenanzahl pro Zuordnungstabelle (0x16). Diese beiden Parameter sind bei FAT32 immer 0.

4.2.1 Fortsetzung der Volume ID bei FAT16

Ab dem 55. Byte (0x36) der Volume ID ist eine acht Zeichen lange Konstante abgelegt. Diese enthält die Bezeichnung des Dateisystems im Klartext und wird mit Leerzeichen aufgefüllt: „FAT16 “ [5]. Diese Zeichenkette wird bei diesem Projekt nicht verwendet.
Weitere Informationen werden für FAT16 nicht benötigt.

Der Startsektor der Zuordnungstabelle muss aus mehreren Parametern berechnet werden:

FAT-Startsektor = Startsektor der Volume ID + Anzahl reservierter Sektoren

Gleichung 4: Berechnung des Startsektors der Zuordnungstabelle bei FAT16 (nach [7])

Der Anfang des Root-Verzeichnisses (Stammverzeichnis) folgt bei FAT16 direkt auf die Zuordnungstabelle. Dieser Sektor muss ebenfalls berechnet werden:

Root-Verzeichnis-Startsektor = FAT-Startsektor + (Anzahl FAT * Sektoren pro FAT)

Gleichung 5: Berechnung des Startsektors des Root-Verzeichnisses bei FAT16 (nach [7])

Der Datenbereich folgt direkt auf das Root-Verzeichnis, dessen Länge in der Volume ID gespeichert ist (maximale Anzahl der Einträge im Stammverzeichnis).

Ein Eintrag in einem Verzeichnis besteht aus 32 Bytes. Daher muss die maximale Anzahl der Einträge im Stammverzeichnis mit 32 multipliziert und anschließend durch die Anzahl der Bytes pro Sektor dividiert werden. Da die Berechnung ausschließlich mit ganzen Zahlen durchgeführt wird, ist das Ergebnis ebenfalls ganzzahlig.

Datenbereich-Startsektor = Root-Verzeichnis-Startsektor + (maximale Anzahl der Einträge im Stammverzeichnis * 32 / Bytes pro Sektor)

Gleichung 6: Berechnung des Startsektors des Datenbereiches bei FAT16 (nach [7])

Der Datenbereich-Startsektor stellt den Beginn des Clusters mit der Nummer 2 dar. Cluster mit den Nummern 0 und 1 werden nicht akzeptiert, diese sind für Sonderanwendungen reserviert.

4.2.2 Fortsetzung der Volume ID bei FAT32

FAT32 verwendet ebenfalls einen Parameter für die Sektoren pro Zuordnungstabelle, jedoch ist dieser an der Position 0x24 gespeichert und nicht an der Position 0x16 [5], [8].
Das Root-Verzeichnis (Stammverzeichnis) muss bei FAT32 nicht unbedingt am Anfang des Datenbereiches stehen. Daher wird dessen Startcluster an der Position 0x2C in der Volume ID gespeichert [8].
Ab dem 83. Byte (0x52) der Volume ID ist eine acht Zeichen lange Konstante abgelegt [5]. Diese enthält die Bezeichnung des Dateisystems im Klartext und wird mit Leerzeichen aufgefüllt: „FAT32 “ [5]. Diese Zeichenkette wird bei diesem Projekt nicht verwendet.

Der Startsektor der Zuordnungstabelle muss aus mehreren Parametern berechnet werden:

FAT-Startsektor = Startsektor der Volume ID + Anzahl reservierter Sektoren

Gleichung 7: Berechnung des Startsektors der Zuordnungstabelle bei FAT32 (nach [8])

Der Datenbereich folgt direkt auf die letzte Zuordnungstabelle und stellt den Beginn des Clusters mit der Nummer 2 dar. Cluster mit den Nummern 0 und 1 werden nicht akzeptiert, diese sind für Sonderanwendungen reserviert.

Datenbereich-Startsektor = FAT-Startsektor + (Anzahl FAT * Sektoren pro FAT)

Gleichung 8: Berechnung des Startsektors des Datenbereiches bei FAT32 (nach [8])

Der Anfang des Root-Verzeichnisses folgt nicht automatisch im Anschluss an die Zuordnungstabelle. Seine Position muss nicht berechnet werden.

4.3 Verzeichnisinhalte

Ein Verzeichniseintrag besteht immer aus 32 Bytes [6]. In einem Sektor mit der Größe von 512 Bytes sind somit 16 Verzeichniseinträge abgelegt. Diese sind entweder belegt oder bleiben leer (null).
Die ersten acht Bytes eines jeden Eintrages enthalten den Dateinamen, die nächsten drei (ab 0x08) die Dateierweiterung. Der typische Punkt zwischen Dateiname und -erweiterung wird

nicht gespeichert. Ab der Byteposition 0x1A eines jeden Eintrages ist die zwei Byte lange Adresse des Anfangsclusters der Datei bzw. des Verzeichnisses abgelegt. Ab der Byteposition 0x1C ist die vier Byte lange Dateigröße abgelegt [6]. Die Dateigröße ist in Bytes angegeben.
Bei FAT32 befinden sich zwei zusätzliche Adressbytes an der Position 0x14. Diese stellen das dritte und vierte Byte der Clusternummer unter FAT32 dar. Unter FAT16 können diese beiden Bytes nicht genutzt werden.
Das zwölfte Byte enthält die Attribute des Eintrages. Das Attributbyte ist folgendermaßen strukturiert:

Attributbit	Funktion
0	Schreibschutz: Der Eintrag kann nur gelesen werden.
1	Versteckt: Der Eintrag ist nicht sichtbar.
2	Der Eintrag gehört zum Betriebssystem.
3	Der Eintrag ist die Bezeichnung der Partition.
4	Es handelt sich bei dem Eintrag um ein Unterverzeichnis.
5	Der Eintrag ist archiviert.
6	Nicht verwendet, wird auf null gesetzt.
7	

Tabelle 8: Strukturierung des Attributbytes eines Verzeichniseintrages (nach [6])

Eine Eigenschaft trifft auf den Verzeichniseintrag zu, wenn das zugehörige Attributbit gesetzt ist (eins). Sie trifft nicht zu, wenn das zugehörige Attributbit nicht gesetzt ist (null).
Der Eintrag eines Unterverzeichnisses hat immer das Attributbit 4 gesetzt und weist als Größe den Wert null auf.
Eine Datei kann ebenfalls eine Größe von null Byte haben, jedoch ist hier das Attributbit 4 gleich null.
In der Regel ist der erste Eintrag im Root-Verzeichnis die Bezeichnung der Partition. Diese ist bis zu elf Zeichen lang. In diesem Fall ist nur das Attributbit 3 gesetzt und die ersten elf Bytes des Eintrages bilden zusammengesetzt die Bezeichnung der Partition (Dateiname und Dateierweiterung).
Wird ein Eintrag gelöscht, wird das erste Zeichen des Dateinamens durch ein spezielles ASCII-Zeichen ersetzt. Der Rest des Eintrages bleibt unverändert.

Ein Verzeichnis kann auch länger als ein Sektor sein. In diesem Fall setzt sich das Verzeichnis im nächsten Sektor fort.
Handelt es sich um das Root-Verzeichnis von FAT16, so setzt sich das Verzeichnis über die hintereinanderliegenden Sektoren fort, bis die maximale Anzahl der Einträge im Stammverzeichnis erreicht ist.
Alle anderen Verzeichnisse belegen immer mindestens einen kompletten Cluster (auch das Root-Verzeichnis bei FAT32). Ist das Ende des Clusters erreicht, kann das Verzeichnis in einem anderen Cluster fortgesetzt werden, jedoch ist dieses selten im direkt anschließenden Cluster der Fall. Hierzu wird die Clusterkette benötigt.

4.4 Die Clusterkette

Die Zuordnungstabelle enthält so viele Einträge, wie es Cluster in der Partition gibt. Ein Eintrag ist bei FAT16 zwei Byte und bei FAT32 vier Byte lang. Ein Eintrag enthält die Nummer des Clusters, in dem die Datei oder das Verzeichnis fortgesetzt werden.
Ist der aktuelle Cluster der letzte Cluster der Datei oder des Verzeichnisses, ist an dieser Stelle eine entsprechende Markierung in der Zuordnungstabelle eingetragen.
In der folgenden Tabelle werden die Wertebereiche und deren Bedeutung genauer erklärt:

Funktion	Wert oder Wertebereich bei FAT16	Wert oder Wertebereich bei FAT32
Cluster ist leer	0x0000	0x00000000
Clusterzuordnung: Nummer des nächsten Clusters	0x0002 - 0xFFEF	0x00000002 - 0x0FFFFFFF
Letzter Cluster einer Datei	0xFFF8 - 0xFFFF	0xFFFFFFF8 - 0xFFFFFFFF

Tabelle 9: Einträge in der Zuordnungstabelle bei FAT16 und FAT32 (nach [6], [7], [8])

Um die Funktion der Zuordnungstabelle besser zu verstehen, wird diese anhand eines Beispieles erklärt:

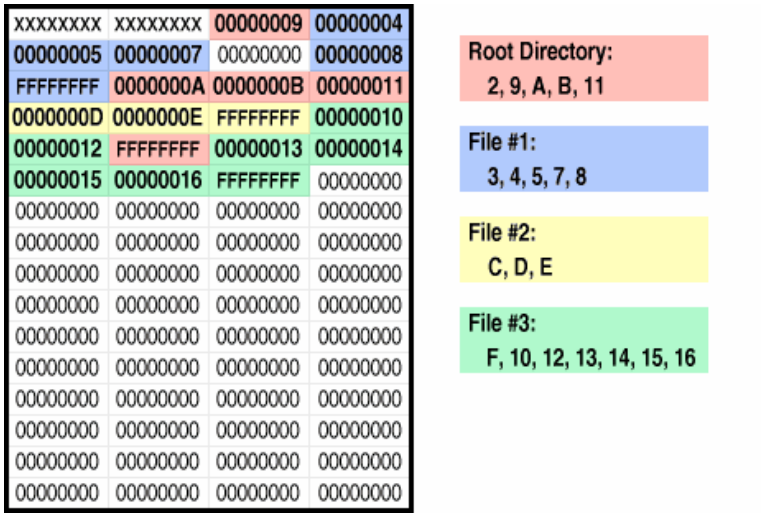


Abbildung 8: Beispiel einer Clusterkette bei FAT32 [6]

Dieses Beispiel zeigt das Root-Verzeichnis und drei Dateien bei FAT32.

- Das Root-Verzeichnis (rot) beginnt mit Cluster 0x00000002. Der Startcluster wird der Volume ID der Partition entnommen. Der Eintrag in der Zuordnungstabelle an der Stelle 0x00000002 verweist auf den Cluster 0x00000009. Hier setzt sich das Root-Verzeichnis fort. Der Eintrag in der Zuordnungstabelle an der Stelle 0x00000009 verweist auf den Cluster 0x0000000A usw. Das Root-Verzeichnis besteht somit aus den Clustern 0x00000002, 0x00000009, 0x0000000A, 0x0000000B und 0x00000011. Der Eintrag in der Zuordnungstabelle an der Stelle 0x00000011 zeigt durch den Wert 0xFFFFFFFF an, dass es sich um den letzten Cluster des Verzeichnisses handelt.
- Die erste Datei (blau) beginnt mit Cluster 0x00000003. Der Startcluster wird einem Eintrag in einem Verzeichnis (z.B. dem Root-Verzeichnis) entnommen. Der Eintrag in der Zuordnungstabelle an der Stelle 0x00000003 verweist auf den Cluster 0x00000004. Hier

setzt sich die Datei fort. Der Eintrag in der Zuordnungstabelle an der Stelle 0x00000004 verweist auf den Cluster 0x00000005 usw. Die Datei besteht somit aus den Clustern 0x00000003, 0x00000004, 0x00000005, 0x00000007 und 0x00000008. Der Eintrag in der Zuordnungstabelle an der Stelle 0x0000008 zeigt durch den Wert 0xFFFFFFFF an, dass es sich um den letzten Cluster der Datei handelt.

- Die zweite Datei (gelb) besteht aus den Clustern 0x0000000C, 0x0000000D und 0x0000000E. Der Eintrag in der Zuordnungstabelle an der Stelle 0x0000000E zeigt durch den Wert 0xFFFFFFFF an, dass es sich um den letzten Cluster der Datei handelt.
- Die dritte Datei (grün) besteht aus den Clustern 0x0000000F, 0x00000010, 0x00000012, 0x00000013, 0x00000014, 0x00000015 und 0x00000016. Der Eintrag in der Zuordnungstabelle an der Stelle 0x00000016 zeigt durch den Wert 0xFFFFFFFF an, dass es sich um den letzten Cluster der Datei handelt.
- Der Cluster 0x00000006 ist leer, ebenso die Cluster ab 0x00000017. Dieses wird durch deren Inhalt 0x00000000 in der Zuordnungstabelle markiert.

Die Sektoren innerhalb eines Clusters stehen immer direkt hintereinander. Nach dem letzten Sektor eines Clusters muss aus der Zuordnungstabelle der nächste Cluster ermittelt werden.

Der letzte Cluster muss nicht komplett gefüllt sein. Die freien Sektoren stehen jedoch für keine weiteren Anwendungen zur Verfügung. Wie weit der letzte Cluster belegt ist, muss der Dateigröße entnommen werden.

Es ist nicht möglich, innerhalb der Clusterkette zu dem vorherigen Cluster zurückzuspringen, da immer nur die Nummer des Folgeclusters gespeichert ist.

5 Aufbau des Lasersystems zu Testzwecken

Damit die zu entwickelnde Steuerung möglichst praxisnah getestet und optimiert werden kann, ist ein Lasersystem erforderlich. Da auch gebrauchte, fertig montierte Lasersysteme sehr viel Geld kosten, muss ein einfaches Lasersystem selbst gebaut werden.
Die meisten verwendeten Komponenten sind fertig aufgebaute und getestete Baugruppen:

- Schaltnetzteile für die Spannungsversorgung,
- Laserdiode,
- Laserscanner (im Aluminiumblock montiert) und passende Scannertreiber.

Der Aufbau ist nicht sehr kompliziert. Lediglich die Signalaufbereitung und die Ansteuerung der Laserdiode müssen selbst entwickelt werden.
Untergebracht wird das komplette System in einem stabilen 19“-Gehäuse aus 2mm dickem Aluminiumblech. Die Außenabmessungen des Gehäuses sind 435mm * 290mm * 134mm. Die optischen Komponenten, die Treiber der Scanner und die Platine mit der Signalaufbereitung werden auf einer 12mm dicken Aluminiumplatte montiert. Durch diese Aluminiumplatte wird sichergestellt, dass das Lasersystem nur einmal während der Aufbauphase justiert werden muss. Beim Justieren und Testen ist darauf zu achten, dass niemals in den Laserstrahl geblickt wird, weder direkt noch in dessen Reflektion!

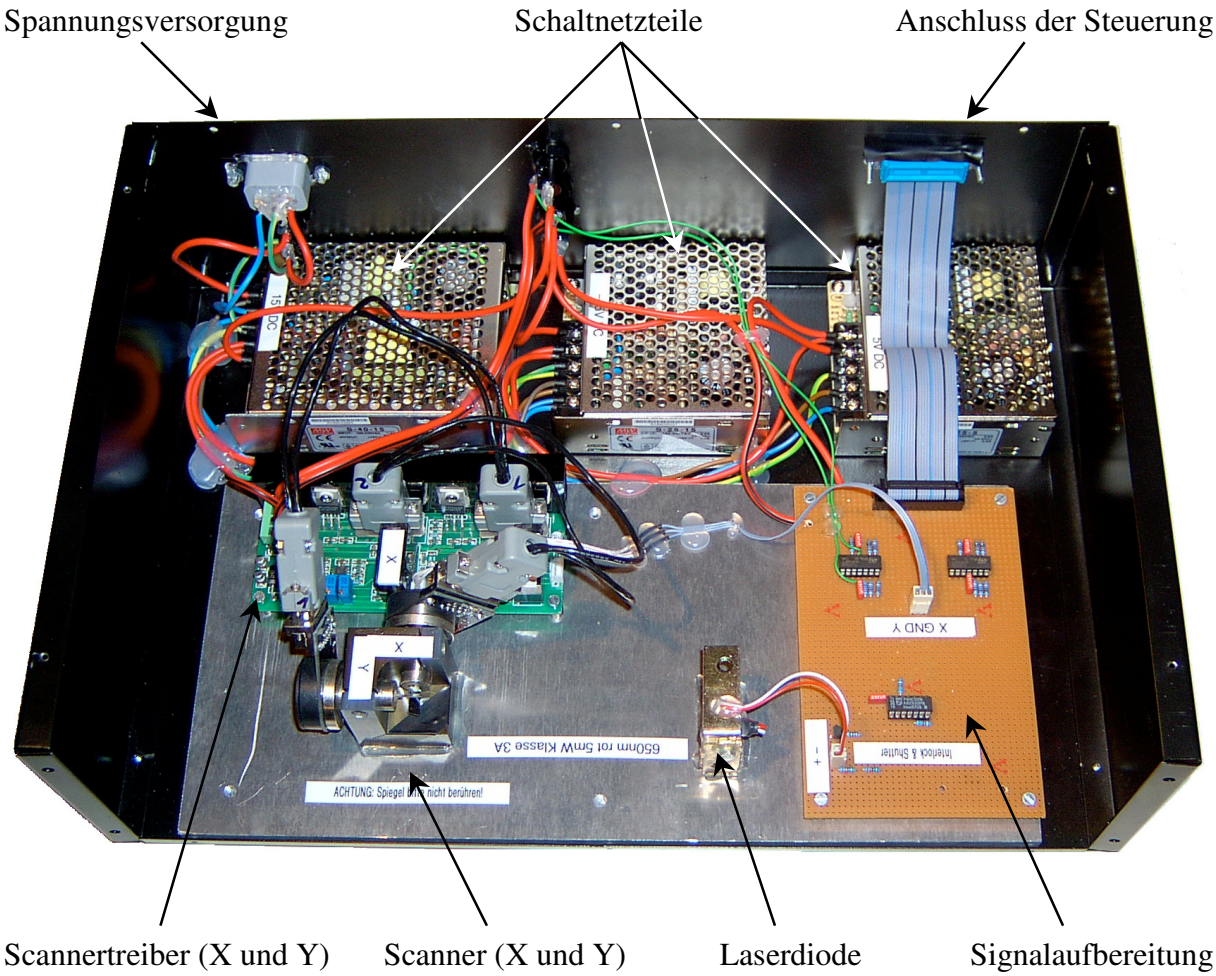


Abbildung 9: Übersicht des komplett aufgebauten Lasersystems

5.1 Spannungsversorgung

Für die Spannungsversorgung werden Schaltnetzteile gewählt, da sie einen deutlich höheren Wirkungsgrad als linear geregelte Netzteile haben. Vorteilhaft ist, dass bei diesen verwendeten Netzteilen keine Probleme durch Verlustwärme, die aufwendig abgeführt werden müsste, auftreten.
Es werden drei verschiedene Spannungen und somit drei Schaltnetzteile benötigt:

1. +15VDC mit mindestens 1,5A: Das verwendete 40W-Schaltnetzteil kann bis zu 2,8A liefern.
2. -15VDC mit mindestens 1A: Das verwendete 25W-Schaltnetzteil kann bis zu 1,7A liefern.
3. +5VDC mit mindestens 150mA: Das verwendete 25W-Schaltnetzteil kann bis zu 5A liefern.

Die Schaltnetzteile wurden bewusst großzügig dimensioniert, damit die Spannungsversorgung des Scanners und der Treiber nicht einbricht, denn bereits leichte Spannungsschwankungen können die Projektion stören.
Das Schaltnetzteil, das die +5VDC-Versorgung übernimmt, kann auch für deutlich stärkere Laserdioden verwendet werden.
Die Spannungsversorgung der Lasertreiber erfolgt durch Leitungen mit einem Leiterquerschnitt von 1,5mm^2, um Leitungsverluste zu minimieren.
Alle Signale werden über normale Schaltlitzen mit einem Querschnitt von 0,14mm^2 übertragen.

5.2 Laserdiode

Als Laserdiode kommt ein kleines Lasermodul der Laserklasse 3A mit einer Leistung von <5mW und einer Wellenlänge von 650nm (sichtbares Rot) zum Einsatz. Das Lasermodul beinhaltet die eigentliche Laserdiode, die Treiberelektronik (Strombegrenzung) und die Linse. Die Laserdiode und die Linse sind in einem kleinen Messinggehäuse eingebaut. Die Entfernung zwischen Laserdiode und Linse kann durch Drehen der vorderen Gehäusehälfte variiert und dadurch der Durchmesser des Laserstrahls justiert werden. Das Lasermodul wurde auf einen Strahldurchmesser ≤2mm eingestellt.
Die Laserdiode benötigt bei 3VDC einen Strom von 17mA.

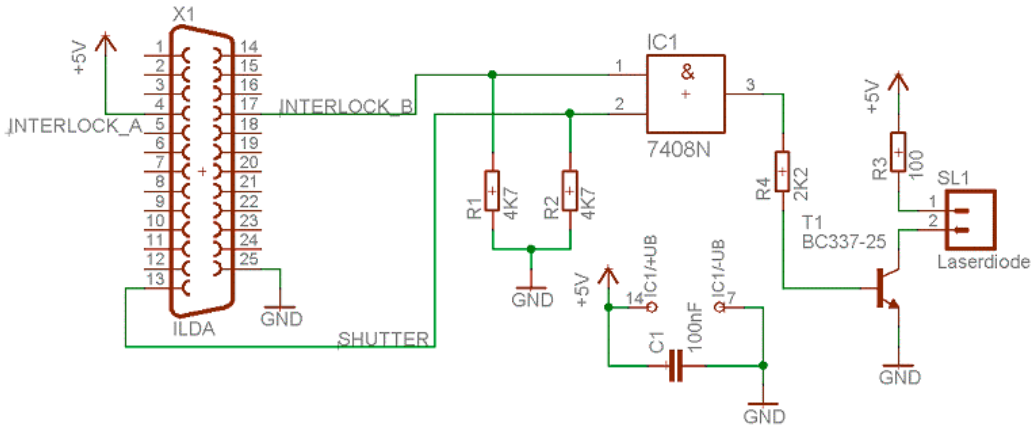


Abbildung 10: Schaltplan der Laserdiodenansteuerung

Die Widerstände R1 und R2 dienen als Pulldown-Widerstände dafür, dass der Laser ohne angeschlossene oder eingeschaltete Steuerung immer abgeschaltet ist.
Der Kondensator C1 ist ein typischer Siebkondensator, wie er in der digitalen Schaltungstechnik praktisch immer direkt neben jeder Spannungsversorgung eines ICs eingesetzt wird. Er soll die Spannungsversorgung des ICs störungsarm halten.

Der Vorwiderstand R3 der Laserdiode wird berechnet:

$$R3 = \frac{5V - U_{Laserdiode} - U_{CE}}{I_{Laserdiode}} = \frac{5V - 3V - 0{,}3V}{17mA} = \frac{1{,}7V}{17mA} = 100\Omega$$

Gleichung 9: Berechnung des Vorwiderstandes R3 der Laserdiode

Die Stromverstärkung des Transistors *BC337-25* beträgt B = 25 [9]. Der Basisstrom wird mit folgender Formel berechnet:

$$I_B = \frac{I_{Laserdiode}}{B} = \frac{17mA}{25} = 680\mu A$$

Gleichung 10: Berechnung des theoretischen Basisstromes des Transistors T1

Der Basisvorwiderstand R4 darf nicht größer sein als der im Folgenden berechnete Wert:

$$R4 = \frac{5V - U_{BE}}{I_B} = \frac{5V - 0{,}7V}{680\mu A} = \frac{4{,}3V}{680\mu A} = 6{,}323K\Omega$$

Gleichung 11: Berechnung des theoretischen Basisvorwiderstandes R4

Um das schnelle Schalten der Laserdiode sicherzustellen, wird ein kleinerer Wert für den Basisvorwiderstand R4 verwendet. Die Basis wird somit übersteuert. Für R4 wird der Widerstandswert 2,2KΩ gewählt. Der tatsächliche Basisstrom wird mit dem gewählten R4 neu berechnet:

$$I_B = \frac{5V - U_{BE}}{R4} = \frac{5V - 0{,}7V}{2{,}2K\Omega} = \frac{4{,}3V}{2{,}2K\Omega} = 1{,}95mA$$

Gleichung 12: Berechnung des tatsächlichen Basisstromes des Transistors T1

Der maximal zulässige Basisstrom beträgt 100mA, der Transistor wird trotz Übersteuerung im zulässigen Bereich betrieben [9].

5.3 Laserscanner und Treiber

Bei dem Laserscanner mit passender Treiberplatine handelt es sich um ein preiswertes, fertig gekauftes Modul. Die Treiber sind optimal auf die beiden Galvos abgestimmt. Galvos werden, mit einem Spiegel am Ende der Achse bestückt, zur Ablenkung der Laserstrahlen benutzt. In ihrem Inneren befindet sich ein kleiner, runder Magnet, der fest mit der Achse verbunden ist. Die Achse verläuft auf der Höhe des Magneten durch eine Spule. Die Spule wird von der externen Treiberplatine elektrisch angeregt und erzeugt ein polarisiertes

Magnetfeld. Daraufhin wird die Achse abgelenkt und kippt in eine Richtung. Wird die elektrische Anregung verändert, ändert sich das Magnetfeld und die Achse kippt zu einer neuen Position. Wird die elektrische Anregung abgeschaltet, wird die Achse durch eine kleine Feder in Mittelstellung gebracht.

Das verwendete Lasersystem ist ein Closed-Loop-System, d.h. in den Galvos sind Rückmeldeeinheiten eingebaut, die der Treiberelektronik die aktuelle Achsenstellung mitteilen. Die Treiberelektronik regelt Spannung und Strom für die Spule so aus, dass der Spiegel kurz vor dem Erreichen seiner gewünschten Endlage abgebremst wird. Dadurch wird die Endlage ohne größeres Überschwingen erreicht.

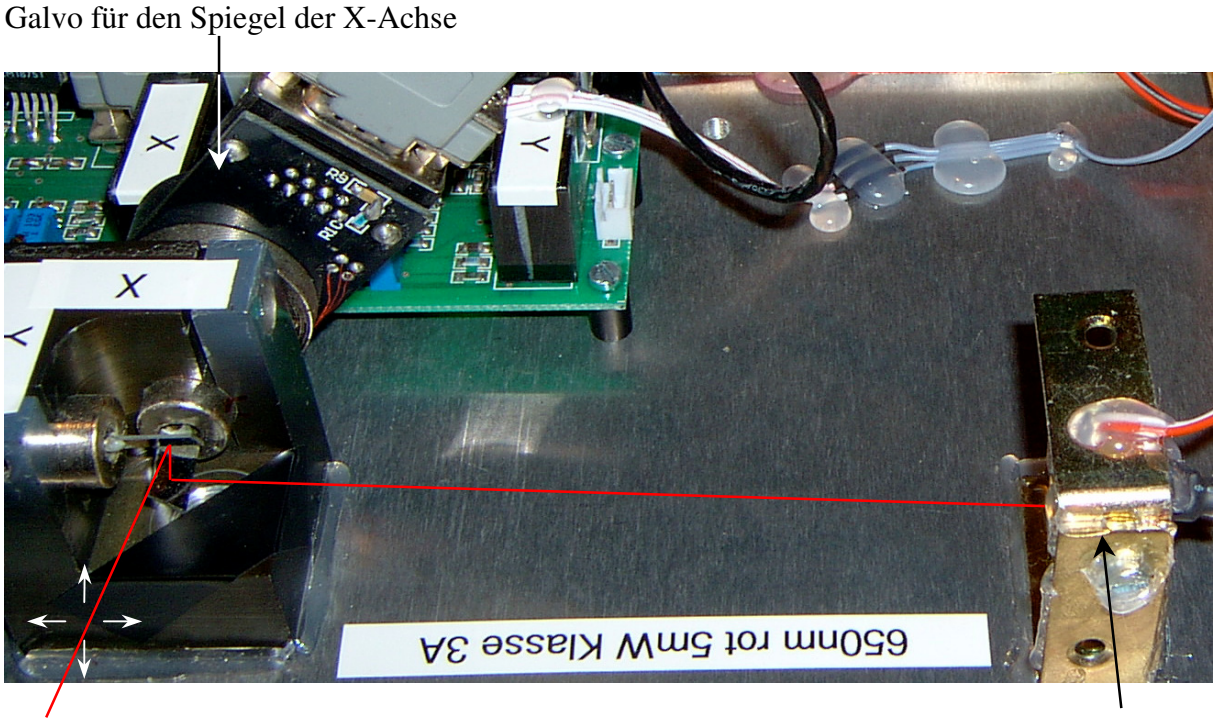


Abbildung 11: Verlauf des Laserstrahls im Lasersystem

Der Laserstrahl trifft zuerst auf den Spiegel, der die Ablenkung in X-Richtung übernimmt. Im Anschluss trifft der bereits abgelenkte Laserstrahl auf den zweiten Spiegel und wird über diesen in der Y-Achse abgelenkt.
Der verwendete Scanner erzielt bei einfachen Lasergrafiken gute Ergebnisse. Jedoch überfordern aufwendige Schriften und Figuren diesen Scanner. Sie sollten den deutlich teureren Systemen vorbehalten bleiben.

5.4 Signalaufbereitung für die beiden Scannerkanäle

Da es sich bei den Treibern für den Laserscanner um preiswerte Exemplare handelt, sind diese nicht mit differentiellen Signaleingängen ausgestattet.
Um trotzdem längere Leitungswege zwischen Steuerung und Lasersystem überbrücken zu können, wird eine Verstärkerschaltung mit differentiellem Eingang für jeden der beiden Scannerkanäle eingesetzt. Der Eingangswiderstand der Schaltung beträgt 10KΩ. Um einen Einfluss der Scannertreibereingänge auf die Addiererschaltung (N1C, R4, R5, R6)

auszuschließen, wird ein Spannungsfolger mit niedrigem Ausgangswiderstand nachgeschaltet (N1D).
Zur Spannungsversorgung werden die Operationsverstärker direkt an ±15V aus den Schaltnetzteilen angeschlossen. Dieses geschieht in der Simulation durch die beiden Spannungsquellen U1 und U2.
Da das Ausgangssignal der Addiererschaltung invertiert ist, muss eine weitere Invertierung eingesetzt werden, um den gewünschten Originalverlauf des Signals zu erhalten. Dieses kann entweder durch einen nachgeschalteten Inverter erfolgen oder die beiden Signale, die dem Addierer zugeführt werden, werden vorher invertiert.
Hier wird die zweite Möglichkeit angewendet:

- Das Signal in- wird nicht, wie es eigentlich erwartet würde, invertiert, sondern nur durch einen Spannungsfolger (N1A und R1, nichtinvertierender Verstärker mit dem Verstärkungsfaktor 1) aufbereitet.
- Das Signal in+ wird entgegen der Erwartung invertiert (N1B, R2 und R3). Die Verstärkung wird im Folgenden berechnet. Das negative Vorzeichen deutet die Invertierung an.

$$v_{N1B} = -\frac{R3}{R2} = -\frac{10K\Omega}{10K\Omega} = -1$$

Gleichung 13: Berechnung der Verstärkung der OP-Schaltung um N1B

Der Addierer (N1C, R4, R5, R6) summiert die beiden Zwischensignale out+ und out-. Damit die Verdopplung der Spannungshöhe vermieden wird, übernimmt der Addierer die Division seiner beiden Eingangsspannungen durch den Faktor 2. Berechnet wird beispielhaft der Zweig des Signals out-. Der Zweig für das Signal out+ wird nach derselben Formel (R5 statt R4) berechnet.

$$v_{N1C} = -\frac{R6}{R4} = -\frac{10K\Omega}{20K\Omega} = -0{,}5$$

Gleichung 14: Berechnung der Verstärkung des Addierers um N1C

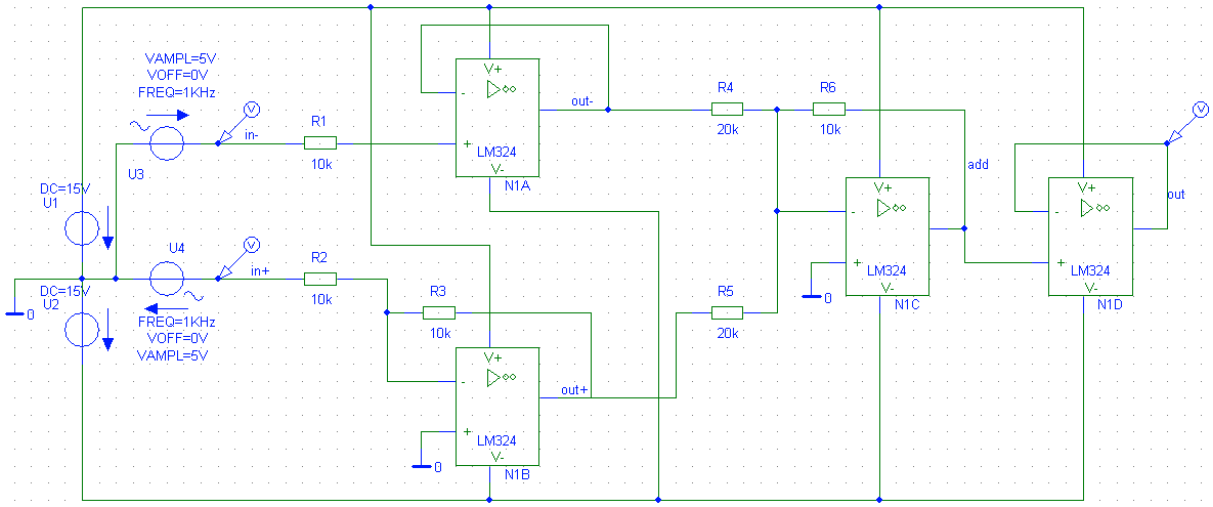


Abbildung 12: Schaltung der Eingangsstufe (PSpice)

Die Spannungsquellen U3 und U4 erzeugen eine sinusförmige Spannung mit einer Amplitude von ±5V bei einem Offset von 0V und der Frequenz von 1KHz.
Die Spannungsquelle U4 ist gedreht. Somit wird ein symmetrisches Eingangssignal simuliert.

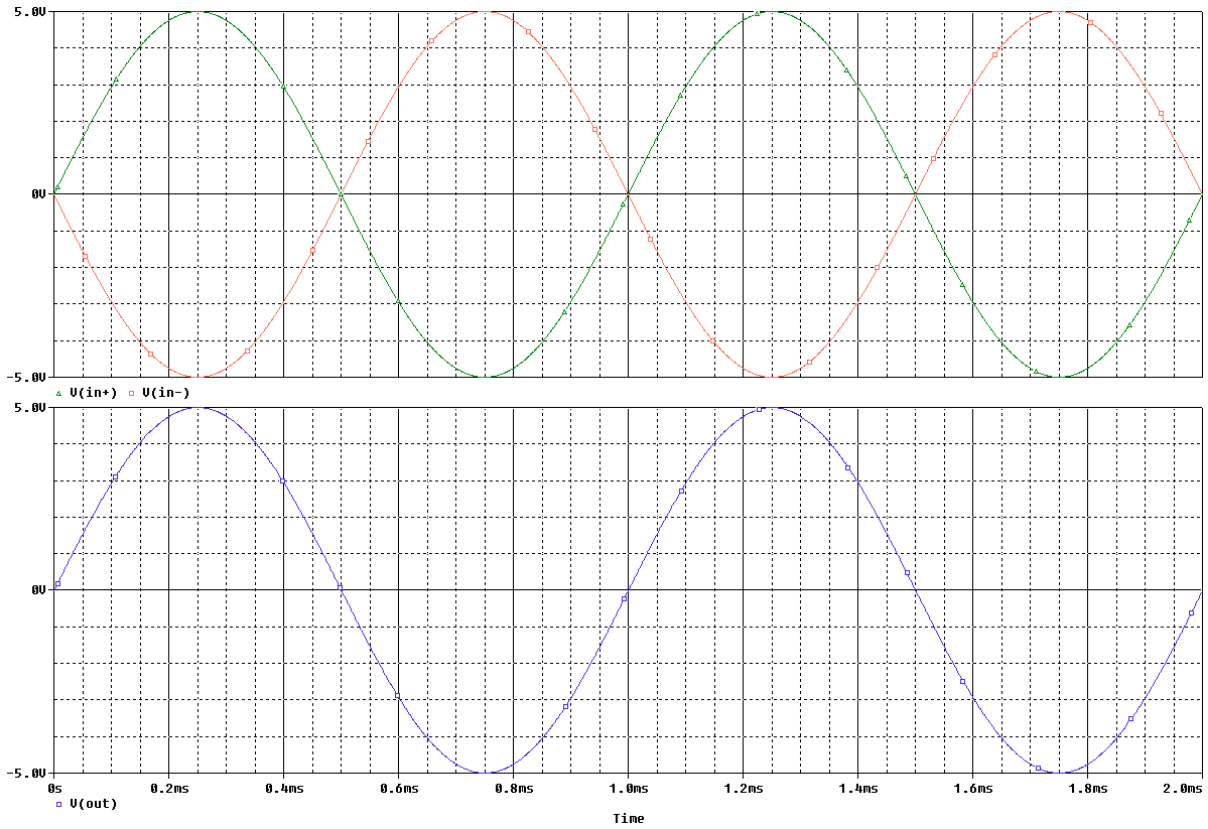


Abbildung 13: Simulationsergebnis der Eingangsstufe (PSpice)

Die Spannungsverläufe sind über einen Zeitraum von 2ms dargestellt. Bei einer Frequenz von 1KHz sind zwei komplette Perioden abgebildet.
Der obere Plot zeigt die symmetrische Eingangsspannung, bestehend aus in+ und in-. Der untere Plot zeigt das Ausgangssignal out. Dieses folgt exakt dem Verlauf des Eingangssignals in+.

Die simulierte Schaltung wird für jeden der beiden Achsenkanäle aufgebaut und eingesetzt. Als Operationsverstärker wird der *TL074* verwendet. Die Spannungsquellen U3 und U4 der Simulation entfallen, stattdessen werden die Signale X+/- oder Y+/- des 25poligen D-SUB-Steckers angeschlossen (Pin 1 und 14 bzw. 2 und 15). Das Ausgangssignal out wird direkt an einen der beiden Scannertreiber weitergeleitet.

6 Hardware

Da das zentrale Bauteil der Hardware ein programmierbarer Mikroprozessor ist, gehören die beiden Kapitel 6 (Hardware) und 7 (Software) eng zusammen. Teilweise werden in diesem Kapitel Funktionsblöcke erwähnt, die erst in Kapitel 7 ausführlicher erläutert werden (z.B. die Abläufe beim Ansteuern des Grafikdisplays). Würden diese beiden Kapitel in ihrer Reihenfolge vertauscht, so würde der reale Ablauf verfälscht: Zuerst werden grobe Züge der Hardware konstruiert, erst danach kann die Softwareentwicklung erfolgen.
In diesem Kapitel wird die Hardwareentwicklung erläutert, von den einzelnen besonderen Baugruppen bis hin zu den kompletten Schaltplänen und Platinenlayouts.

Die unten abgebildete Übersicht der Hardware zeigt die zentrale Lage des Mikroprozessors. Die Funktionsblöcke sind alle mit ihm verbunden und können ohne ihn nicht miteinander kommunizieren. Damit die komplette Hardware zuverlässig und sicher ihre Funktion erfüllen kann, ist es wichtig, dass die Bauteile miteinander kompatibel sind. Die digitalen Bauteile werden so gewählt, dass alle – mit Ausnahme der SD-Karte – mit einer Betriebsspannung von 5VDC arbeiten. Die einzelnen Funktionsblöcke werden im weiteren Verlauf des Kapitels beschrieben.

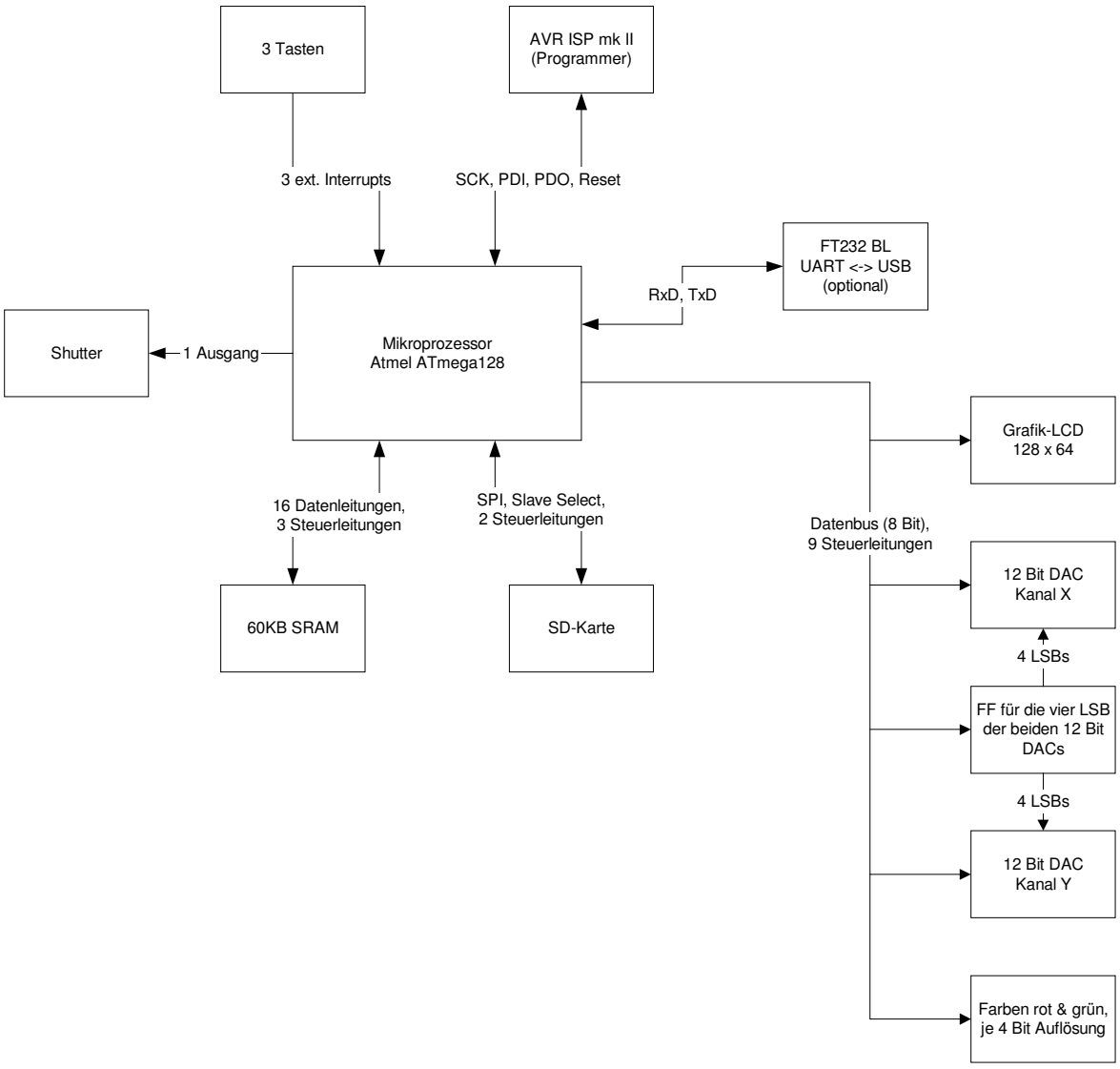


Abbildung 14: Übersicht über die einzelnen Funktionsgruppen (Blockschaltbild)

6.1 Prozessorauswahl

Die beiden wichtigsten Kriterien zur Auswahl eines für diese Aufgabenstellung geeigneten Mikroprozessors sind

1. günstiger Preis und
2. großer RAM (engl.: Random Access Memory), damit das Programm für ein optimales Zeitverhalten einen großen Puffer bei der Laserausgabe verwenden kann.

Alternativ kann auch ein Mikroprozessor mit Unterstützung für externen SRAM (engl.: Static Random Access Memory) gewählt werden.
Der Hersteller *Atmel* bietet weit verbreitete 8-Bit-Mikroprozessoren mit RISC-Technologie (engl.: Reduced Instruction Set Computing) in verschiedenen Größenordnungen und mit verschiedenem Funktionsumfang an. Diese Prozessoren sind preiswert und sehr zuverlässig. Sie können sowohl in C als auch in Assembler programmiert werden. Die Entwicklungsumgebung kann kostenfrei von der Homepage des Herstellers heruntergeladen werden (*AVR Studio*) [10], ein kostenfreier C-Compiler ist ebenfalls im Internet zu finden (*Win-AVR*) [11].
Die Programmierung des Prozessors erfolgt im auf der Platine eingebauten Zustand durch einen ISP (engl.: In System Programmer) über zwei Datenleitungen, Taktsignal und Reset.
Da der Mikroprozessor das zentrale Bauteil des gesamten Projektes ist und daher mit jeder Funktionsgruppe kommunizieren muss, sind folgende Schnittstellen unbedingt erforderlich:

- komplettes SPI und zwei weitere freie Pins (Eingänge) für den direkten Anschluss der SD-Karte
- Schnittstelle für externen SRAM-Baustein
- unidirektionaler paralleler Bus (8 Bit breit, Datenbus für das LCD und die DACs)
- neun freie Pins (Ausgänge) als Steuerleitungen für den Datenbus
- drei freie Interrupt-Eingänge (Taster)
- Programmierschnittstelle für den ISP
- UART-Schnittstelle (engl.: Universal Asynchronous Receiver Transmitter) für den Anschluss des USB-Bausteins *FT232 BL* (optional)
- jeweils mindestens einen Timer mit einer Genauigkeit von 8 Bit bzw. 16 Bit

Es stehen mehrere Prozessoren zur Auswahl, die diese Anforderungen erfüllen. Jedoch nur zwei erfüllen die Bedingung des niedrigen Preises: Der *ATmega64* und der *ATmega128*.

Die benötigten Ausstattungsmerkmale der beiden erwähnten Mikroprozessoren werden direkt miteinander verglichen:

	ATmega64	*ATmega128*
Programmspeicher (Flash)	64 KB	128 KB
RAM	4 KB	4 KB
Interner EEPROM	2 KB	4 KB
Maximale Taktfrequenz	16 MHz	16 MHz
Versorgungsspannung V_{CC}	4,5...5,5V	4,5...5,5V
Schnittstelle für externen SRAM	√	√
8 Bit I/O-Ports	6 + 6 Pins	6 + 5 Pins
U(S)ART	2	2
Externe Interrupts	8	8
Timer (8 Bit)	2	2
Timer (16 Bit)	2	2
SPI	1	1

Tabelle 10: *ATmega64* und *ATmega128* im Vergleich (nach [12], [13])

Beide Prozessoren sind bis auf den Pin 1 komplett pinkompatibel [12], [13]. Sie können direkt miteinander ausgetauscht werden. Pin 1 ist beim *ATmega64* ein normaler I/O-Pin, während dieser Pin beim *ATmega128* eine Sonderfunktion einnimmt und für keine andere Anwendung zur Verfügung steht. Der Prozessor kann in einen Kompatibilitätsmodus für den mittlerweile nicht mehr hergestellten Prozessor *ATmega103* versetzt werden. Dieser veraltete Prozessortyp verwendete einen zusätzlichen Pin zum Programmieren über den ISP.
Sowohl der *ATmega64* als auch der *ATmega128* unterstützen das On-Chip Debugging durch eine integrierte JTAG-Schnittstelle (engl.: Joint Test Action Group) nach IEEE1149.1 (engl.: Institute of Electrical and Electronics Engineers) [12], [13].
Zur Sicherheit wird für dieses Projekt der *ATmega128* ausgewählt, da er über mehr Programmspeicher (Flash) verfügt und dadurch bei der Softwareentwicklung keine Speicherplatzprobleme auftreten. Die komplette Bezeichnung des Prozessors lautet *ATmega128-16AC* [13].

6.2 Externer SRAM

Der Anschluss eines externen SRAM an den Mikroprozessor *ATmega128* ist in dessen Datenblatt genau vorgegeben [13]. Je nach gewünschter Größe verwendet der Adressbus zwischen 8 und 16 Adressleitungen.
Der Hersteller *Atmel* sieht bei mehr als acht Adressleitungen ein zusätzlich einzusetzendes Latch (zustandsgesteuertes D-Flipflop) vor, um die acht LSB (engl.: Least Significant Bit) der Adresse zu puffern. Nach dem Puffern werden beim nächsten Prozessortakt die acht unteren Adressleitungen des Mikroprozessors zum Datenbus, dessen Datenrichtung vom Mikroprozessor vorgegeben wird. Das Latch wird durch den Pin ALE (engl.: Adress Latch Enable) des Mikroprozessors gesteuert. Das Verhalten des SRAM (Schreiben bzw. Lesen) wird durch die beiden Pins –RD (engl.: Read) und –WR (engl.: Write) des Mikroprozessors gesteuert.

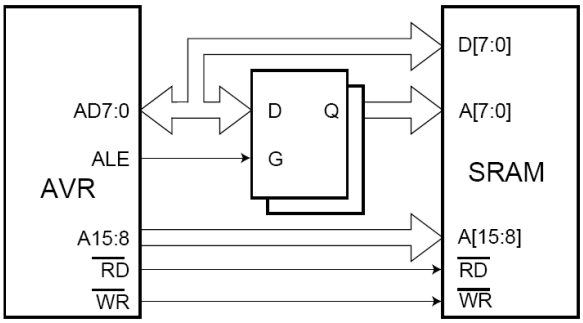


Abbildung 15: Anschluss vom externen SRAM an einen *ATmega128* [13]

Als Latch kommt ein *74VHC573* zum Einsatz. Als SRAM wird der *628512-55M* verwendet.
Um Leiterbahnkreuzungen und aufwendige Layerwechsel auf der Platine zu vermeiden, werden die acht MSB-Adressleitungen (engl.: Most Significant Bit) nicht „eins zu eins“ vom Mikroprozessor zum SRAM geführt, sondern wie folgt zugeordnet:

ATmega128	*628512-55M*	*ATmega128*	*628512-55M*
A8	A12	A12	A8
A9	A15	A13	A9
A10	A17	A14	A11
A11	A13	A15	A10

Tabelle 11: Verbindung der acht MSB-Adressleitungen zwischen Mikroprozessor und SRAM

Beim Platinenlayout können somit kürzere Leiterbahnen verwendet werden, was Störeinflüsse minimiert. Das Vertauschen von Adressleitungen hat keinen Einfluss auf die Zugriffszeit des SRAMs.
Die folgende Abbildung zeigt einen Ausschnitt aus dem Schaltplan des Laser-CPU-Boards (siehe 13.2), auf dem der Anschluss des SRAM-Bausteins und des Latches zu sehen ist. Der Mikroprozessor ist hier nicht mit abgebildet.

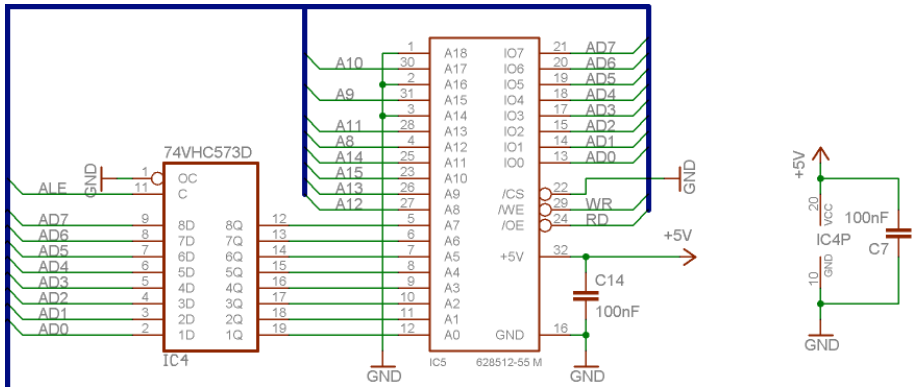


Abbildung 16: Anschluss des SRAM-Bausteins *628512-55M* (Pinbelegung: [14], [15])

Die Adressleitungen A14, A16 und A18 des SRAM werden nicht verwendet und müssen daher direkt mit Masse verbunden werden. Beim Fortlassen dieser Masseverbindungen würden durch die offenen Eingänge chipintern undefinierte Zustände auftreten, was Adressierungsfehler zur Folge haben könnte.
Die Kondensatoren C7 und C14 filtern Spannungsspitzen aus der Versorgungsspannung von IC4 und IC5.
Der Steuerausgang -RD des Mikroprozessors wird mit dem Steuereingang -OE (engl.: Output Enable) des SRAM-Bausteins und der Steuerausgang -WR des Mikroprozessors mit dem Steuereingang -WE (engl.: Write Enable) des SRAM-Bausteins verbunden.
Der verwendete SRAM hat zusätzlich einen Chip Select-Eingang. Dieser wird direkt mit Masse verbunden, der SRAM ist somit dauerhaft aktiv.
Die zeitlichen Signalwechsel und Abläufe beim Lese- und Schreibzugriff des Mikroprozessors auf den SRAM müssen gut aufeinander abgestimmt sein. Diese werden im Folgenden genauer untersucht.

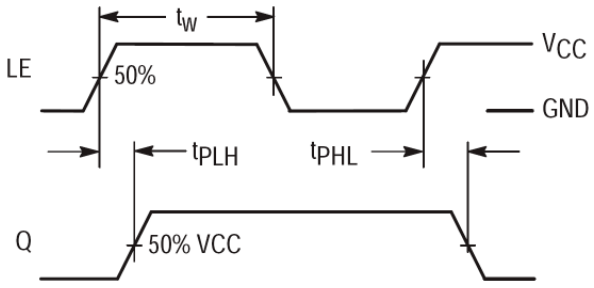


Abbildung 17: Propagation Delay des verwendeten Latches *74VHC573* [15]

Die maximale Verzögerung Propagation Delay (t_{PLH} bzw. t_{PHL}), bis das angelegte Eingangssignal nach gesetztem LE (engl.: Latch Enable) zum Latchausgang Q durchgeschaltet wird, beträgt 9,7ns [15]. Das LE-Signal entspricht dem ALE-Signal des Mikroprozessors.
Die minimale Toröffnungszeit Minimum Pulse Width (t_W) des LE-Signals beträgt 5ns [15].
Der Mikroprozessor arbeitet mit CLK_{CPU} = 16MHz, eine Taktperiode dauert T_{CPU} = 62,5ns.

6.2.1 Zeitlicher Ablauf des Lesezugriffes

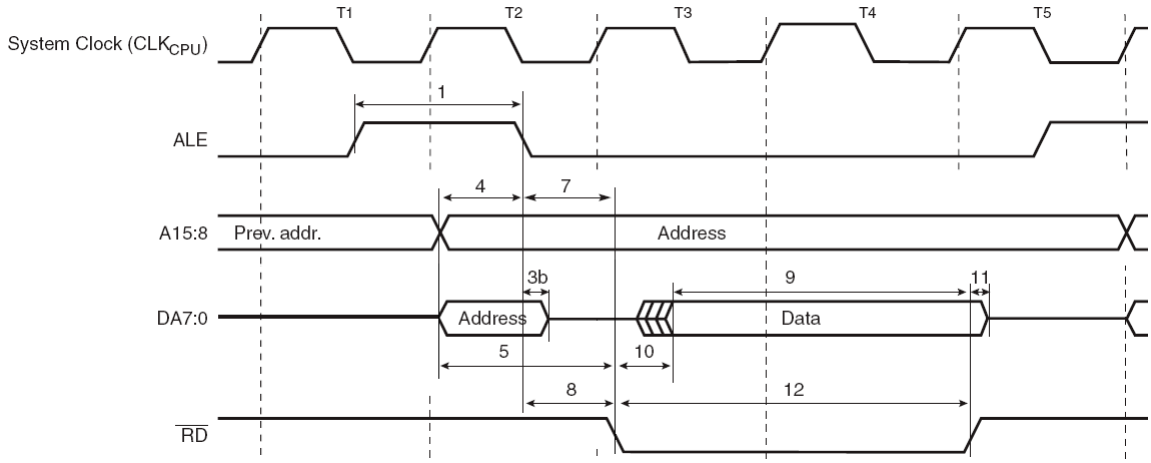


Abbildung 18: Timing beim Lesezugriff des *ATmega128* auf ext. SRAM ([13], bearb.)

Die Werte der verschiedenen Zeiten wurden den Datenblättern der ICs entnommen [13], [14], [15].
Der Mikroprozessor schaltet das Latch bei der fallenden Flanke der Taktperiode T1 transparent. Damit stehen die am Eingang angelegten Daten nach einer maximalen Durchlaufzeit von 8,8ns am Ausgang an.
Bei der steigenden Flanke der Taktperiode T2 legt der Mikroprozessor die Adresse an alle 16 Adressleitungen an.
Mit fallender Flanke in T2 hält das Latch mit dem Pegelwechsel des ALE-Signals von high nach low die unteren acht Adressbits. Der Prozessor gibt die unteren acht Adressbits noch mindestens 5ns lang aus (Zeitangabe 3b in Abbildung 18), um die minimale Haltezeit des Latches (1,5ns) nicht zu verletzen.
Bei der steigenden Taktflanke von T3 schaltet der Mikroprozessor -RD auf low und fordert dadurch vom SRAM Daten an. Spätestens 55ns (t_{AA}) nach dem Anlegen der Adresse an den SRAM stellt dieser sein Datenbyte an seinem Ausgang bereit.
Bei den eingesetzten Werten in der folgenden Berechnung der restlichen Wartezeit handelt es sich um die Werte, die den „Worst Case“ darstellen:

$$t_{restl.Wartezeit} = t_{AA,SRAM} + t_{PLH\,/\,PHL,Latch} - T_{CPU} = 55ns + 9{,}7ns - 62{,}5ns = 2{,}2ns$$

Gleichung 15: Berechnung der restlichen Wartezeit nach dem Aktivieren des Leseprozesses

Da die raminterne Zeitverzögerung vom aktivierten Eingang -RD bis zur Ausgabe der Daten bis zu 25ns betragen kann und der Mikroprozessor eine maximale Zeit von 75ns vorschreibt (Zeitangabe 10 in Abbildung 18), fällt die berechnete maximale Restwartezeit von 2,2ns nicht ins Gewicht.
Während der Taktperiode T4 ändern sich keine Signale. Diese Wartezeit wird benötigt, damit der SRAM die Daten sicher auf den Bus legt, bevor der Mikroprozessor diesen einliest.
Mit der steigenden Flanke der Taktperiode T5 liest der Mikroprozessor den Datenbus ein und schaltet das Signal -RD auf high.
Der nächste Zugriff auf den Datenbus erfolgt nach 62,5ns (T_{CPU}). Jedoch trennt der SRAM seinen Datenausgang bereits nach maximal 20ns vom Datenbus (Zeitangabe 11 in Abbildung 18) und versetzt ihn in den hochohmigen Zustand. Ein Buskonflikt tritt beim Lesevorgang nicht auf.

Bei fallender Flanke von T5 schaltet der Mikroprozessor das Latch transparent, damit wiederholt sich bereits die Taktperiode T1.
Für das Auslesen eines Datenbytes vom SRAM sind also vier Taktperioden erforderlich.

6.2.2 Zeitlicher Ablauf des Schreibzugriffes

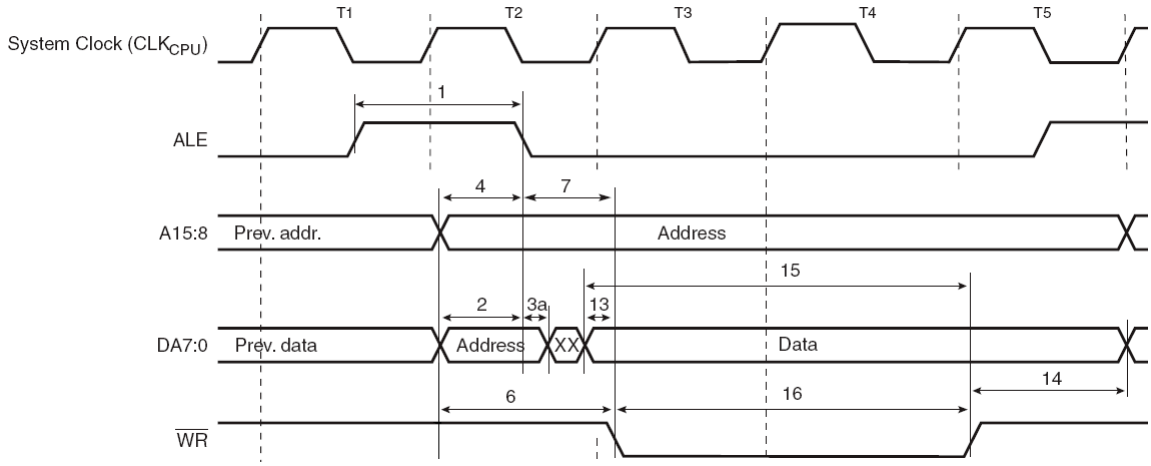


Abbildung 19: Timing beim Schreibzugriff des *ATmega128* auf ext. SRAM ([13], bearb.)

Die Werte der verschiedenen Zeiten wurden den Datenblättern der ICs entnommen [13], [14], [15].
Der Mikroprozessor schaltet das Latch bei der fallenden Flanke der Taktperiode T1 transparent. Damit stehen die am Eingang angelegten Daten nach einer maximalen Durchlaufzeit von 8,8ns am Ausgang an. Bei der steigenden Flanke der Taktperiode T2 legt der Mikroprozessor die Adresse an alle 16 Adressleitungen an.
Mit fallender Flanke in T2 hält das Latch mit dem Pegelwechsel des ALE-Signals von high nach low die unteren acht Adressbits. Der Prozessor gibt diese noch mindestens 5ns lang aus (Zeitangabe 3a in Abbildung 19), um die minimale Haltezeit des Latches (1,5ns) nicht zu verletzen. Im Anschluss – nach einem kurzzeitigen unbestimmten Zustand – legt der Mikroprozessor die zu speichernden Daten auf den Datenbus.
Bei der steigenden Taktflanke von T3 schaltet der Mikroprozessor -WR auf low und initiiert das Speichern der Daten im SRAM. Die Dauer des hierfür erforderlichen Low-Zustandes muss mindestens 40ns betragen (Zeitangabe 16 in Abbildung 19). Der Prozessor hält durch den zusätzlichen Wartetakt T4, der das zuverlässige Speichern ermöglicht, den Pin -WR für 115ns auf low.
Mit der steigenden Flanke der Taktperiode T5 schaltet das Signal -RD auf high.
Der Mikroprozessor hält die Daten noch mindestens 52,5ns auf dem Datenbus (Zeitangabe 14 in Abbildung 19). Ein Buskonflikt tritt beim Schreibvorgang nicht auf.
Bei fallender Flanke von T5 schaltet der Mikroprozessor das Latch transparent, damit wiederholt sich bereits die Taktperiode T1.
Für das Schreiben eines Datenbytes in den SRAM sind also vier Taktperioden erforderlich.

Zwischen den Lese- und Schreibvorgängen muss kein zusätzlicher Wartetakt eingehalten werden.

Bei einer Taktfrequenz von 16MHz und einer Dauer von vier Takten pro Bytetransfer ergibt sich ein maximaler Datendurchsatz von 3,8MB/s. Der SRAM arbeitet beim Lesen und Schreiben mit gleicher Geschwindigkeit.

6.3 Anschluss der SD-Karte

Die SD-Karte ist die einzige Komponente, die nicht mit einer Betriebsspannung von 5VDC betrieben werden darf.

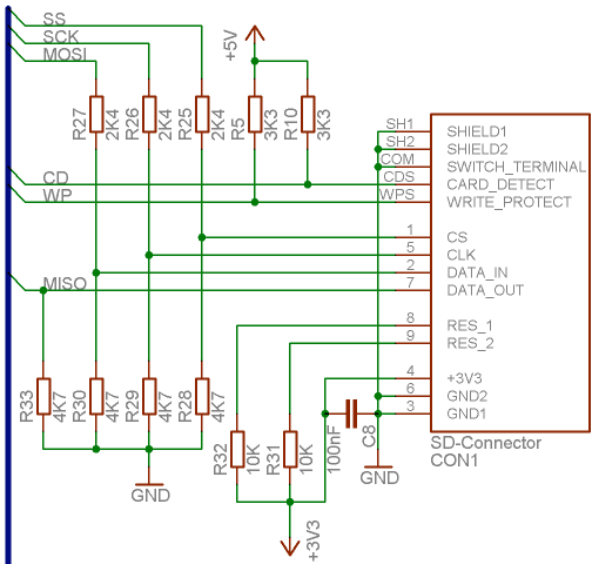


Abbildung 20: SD-Karten-Interface

Die Eingänge der SD-Karte können bei einer direkten Verbindung zum Mikroprozessor durch dessen High-Pegel von 5V zerstört werden.

$$U_{SD\_Eingang\,\max} = 3{,}3V + 0{,}3V = 3{,}6V$$

Gleichung 16: maximale Eingangsspannung an den Eingängen der SD-Karte (nach [4])

Der Pegel wird mit Hilfe einfacher Spannungsteiler in den zulässigen Bereich übertragen. Der Spannungsteiler wird am Beispiel der Signalleitung SCK (engl.: Serial Clock), die zum Eingang CLK (engl.: Clock) der SD-Karte geführt werden soll, wie folgt berechnet:

$$U_{CLK} = U_{SCK\,\max} * \frac{R29}{R26 + R29} = 5V * \frac{4{,}7K\Omega}{2{,}4K\Omega + 4{,}7K\Omega} = 5V * \frac{4{,}7K\Omega}{7{,}1K\Omega} = 3{,}31V$$

Gleichung 17: Berechnung des Spannungsteilers zur Pegelwandlung für die SD-Karte

Für die anderen beiden Widerstandspaare (R25 und R28, Signal Slave Select (CS) sowie R27 und R30, Signal MOSI (engl.: Master Out Slave In)) gilt diese Berechnung entsprechend.

Der einzige Ausgang der SD-Karte ist der Anschluss „Data out“. Dieser wird direkt zum Mikroprozessoreingang MISO (engl.: Master In Slave Out) geführt. Die Low-Pegel des SD-Karten-Ausganges und des Mikroprozessoreinganges arbeiten gut zusammen und erfordern keine Pegelkontrolle.
Der minimale High-Pegel, den die SD-Karte am Ausgang sendet, muss berechnet werden:

$$U_{DATA\_OUT\,\min} = 0{,}75 * V_{DD\_SD-Karte} = 0{,}75 * 3{,}3V = 2{,}5V$$

Gleichung 18: Berechnung des minimalen High-Pegels der SD-Karte am Ausgang (nach [4])

Die minimale Eingangsspannung, die an den Eingangspins des *ATmega128* bei einer Betriebsspannung (V_{CC}) von 5VDC noch als High-Pegel erkannt wird, wird aus folgender Grafik (bei Raumtemperatur, ~25°C) ermittelt:

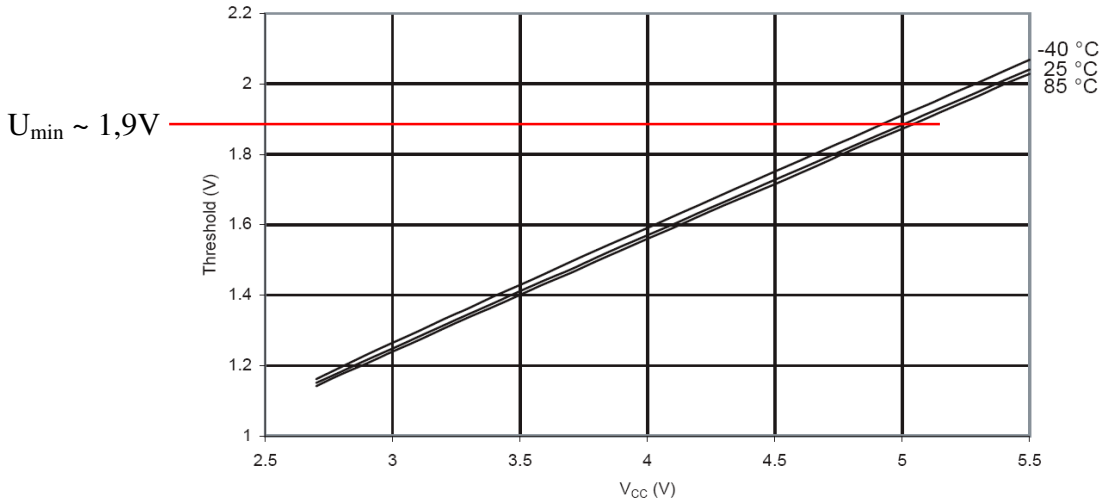


Abbildung 21: Minimaler High-Pegel am Eingang des *ATmega128* ([13], bearb.)

Der Ausgang der SD-Karte kann somit ohne zusätzlichen Schaltungsaufwand direkt an den Eingang des Mikroprozessors angeschlossen werden. Dieser erkennt immer die Zustände low und high. Ist keine SD-Karte eingelegt, kann ein undefinierter Zustand auftreten. Daher wird die MISO-Leitung über einen Pulldown-Widerstand mit Masse verbunden (R33, 4,7KΩ).
Die beiden reservierten Anschlüsse 8 und 9 der SD-Karte im SPI-Modus sind offene Eingänge und müssen daher jeweils mit einem Pullup-Widerstand (R31 und R32) an die Betriebsspannung der SD-Karte (3,3VDC) angeschlossen werden. Hier wird ein Wert von 10KΩ gewählt.
Die beiden Anschlüsse Card Detect (CD) und Write Protection (WP) sind keine Anschlüsse der SD-Karte. Es handelt sich um zwei kleine Schalterkontakte im SD-Karten-Sockel. Ist eine SD-Karte eingelegt, wird der Kontakt Card Detect geschlossen. Bei eingestelltem Schreibschutz der eingelegten Karte wird der Kontakt Write Protection ebenfalls geschlossen.

Die beiden Kontakte sind direkt an zwei Eingänge des Mikroprozessors angeschlossen. Da bei offenen Kontakten undefinierte Zustände auftreten könnten, werden die beiden Schaltkontakte über zwei Pulldown-Widerstände (R5 und R10, je 3,3KΩ) mit der Betriebsspannung (5VDC) verbunden. Im offenen Zustand erhält der Mikroprozessor einen High-Pegel und im geschlossenen einen Low-Pegel.

Der Kondensator C8 filtert Spannungsspitzen aus der Versorgungsspannung der SD-Karte.

Die Übertragung per SPI geschieht byteweise. Es wird der *SPI Mode 0* verwendet:
- MSB wird zuerst übertragen
- Bitübernahme bei steigender CLK-Flanke (Sample on leading edge)
- Bitänderung bei fallender Flanke (Setup on falling edge)
- acht Datenbits

Die Parameter gelten sowohl in der Richtung vom Mikroprozessor zur SD-Karte (MOSI) als auch von der SD-Karte zum Mikroprozessor (MISO).

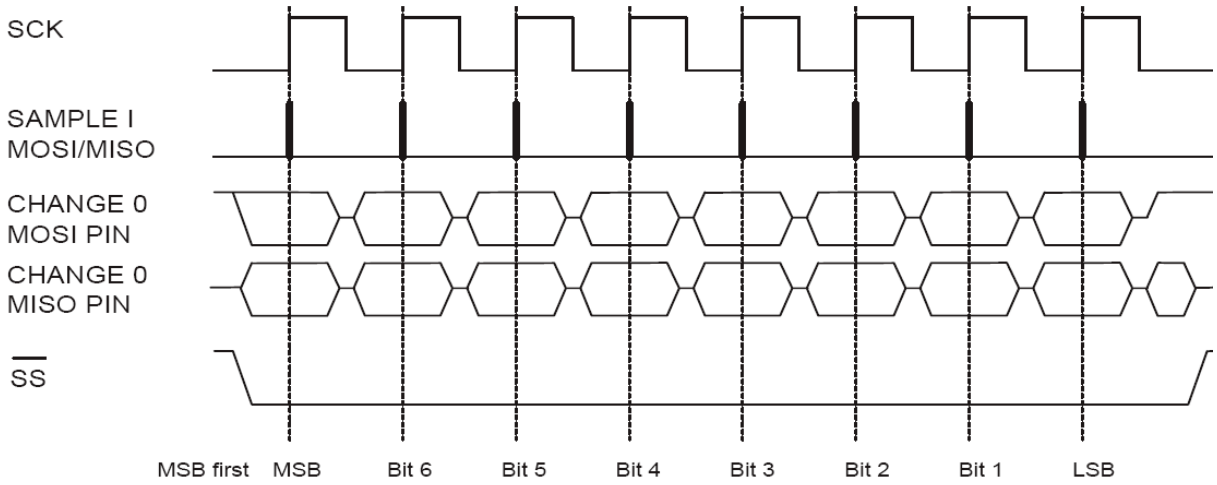


Abbildung 22: Aufbau der Übertragung im *SPI Mode 0* ([13], bearb.)

6.4 Der USB-Baustein *FT232 BL* (optional)

Für eine spätere Erweiterung wird ein USB-Port im Design der Leiterplatte integriert. Die Wahl des verwendeten ICs fällt auf den *FT232 BL* des Herstellers *FTDI Chip*, da dieser einen seriellen Port für den Mikroprozessor bereitstellt und somit nur zwei Pins des Mikroprozessors benötigt.
Die Bitrate kann sowohl bei dem *FT232 BL* als auch bei dem *ATmega128* sehr präzise eingestellt werden. Der *ATmega128* unterstützt Bitraten bis zu 4Mbit/s, der *FT232 BL* unterstützt Bitraten bis zu 3Mbit/s.
Beide verwenden acht Datenbits, ein oder zwei Stoppbit(s) und eine gerade oder ungerade Parität [13], [16]. Die Parität kann auch abgeschaltet werden.

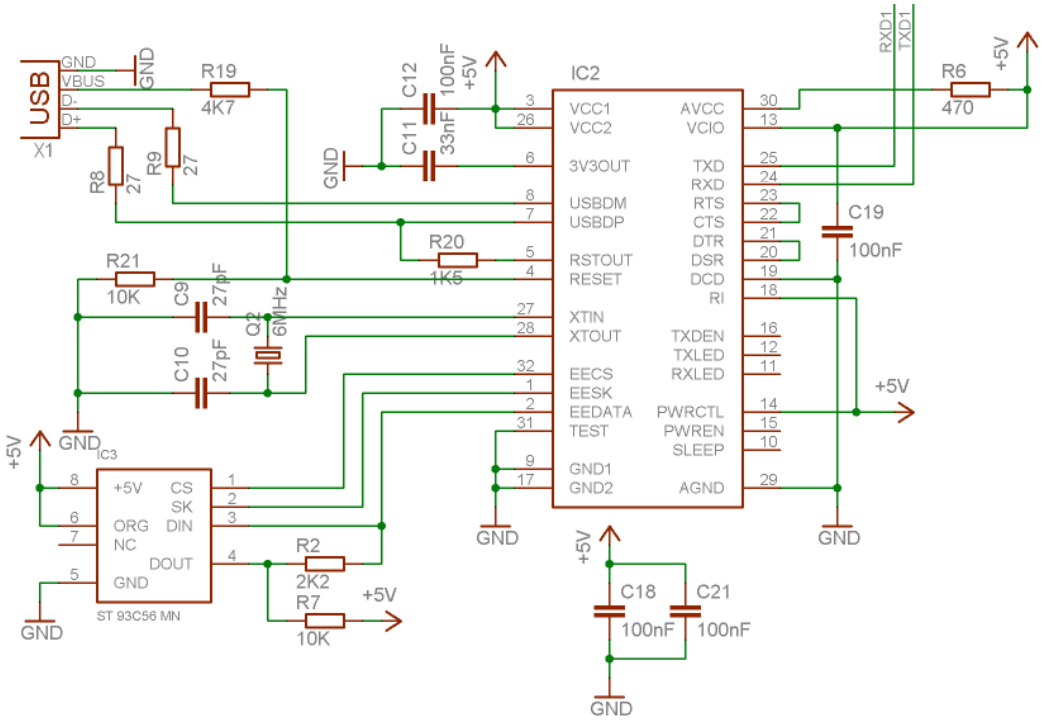


Abbildung 23: Schaltungsaufbau um den USB-Chip *FT232 BL* (nach [16])

Die beiden Leitungen TxD1 (engl.: Transmit Data) und RxD1 (engl.: Receive Data) führen zu den beiden gleichnamigen Pins des Mikroprozessors. Sie stellen den seriellen Port sowie die einzige Verbindung zwischen dem *FT232 BL* und dem Mikroprozessor dar.
Der komplette Schaltungsaufbau wird exakt nach den Vorgaben des Datenblattes [16] erstellt. Damit das USB-Gerät sich beim Anschluss an einen PC mit einem einfach zuzuordnenden Gerätenamen meldet, wird ein zusätzlicher EEPROM (engl.: Electrically Erasable Programmable Read-Only Memory) (IC3, R2, R7) mit eingesetzt. Der *FT232 BL* unterstützt die seriellen EEPROMs der Familie *93Cx6* und erkennt diese während seiner Initialisierungsphase nach einem Reset automatisch [16]. In diesem EEPROM speichert der *FT232 BL* zusätzliche Informationen (z.B. den Gerätenamen).

Zum Konfigurieren des *FT232 BL* muss dieser auf der fertig bestückten Platine an einen PC angeschlossen werden. Der PC erkennt den Chip automatisch. Nach der Installation des Gerätetreibers *FTD2XXST* wird das Konfigurationsprogramm *MProg 3.0a* gestartet. Beide können kostenlos von der Homepage des Herstellers *FTDI Chip* heruntergeladen werden [17], [18].

6.5 Grafikdisplay

Das verwendete Grafikdisplay hat eine Auflösung von 128 * 64 Pixel. Durch die integrierte Hintergrundbeleuchtung ist das Display auch bei schwachem Umgebungslicht gut ablesbar. Die Hintergrundbeleuchtung wird über einen Widerstand (R3, 75Ω) direkt an die Versorgungsspannung (5VDC) angeschlossen und ist somit während des gesamten Betriebes eingeschaltet. Der Wert des Widerstandes wurde durch Testen der Helligkeit der Hintergrundbeleuchtung im Zusammenhang mit der Kontrasteinstellung ermittelt.

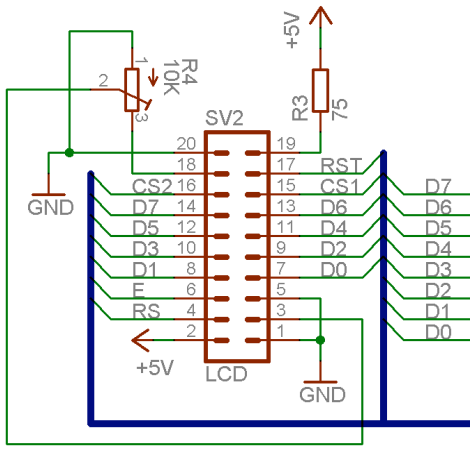


Abbildung 24: Beschaltung des Grafikdisplays (nach [19])

Das Display benötigt nur eine Spannungsversorgung von 5VDC und kann direkt an den Mikroprozessor angeschlossen werden. Die zum Betrieb benötigte negative Spannung wird auf der Platine des Displays selbst erzeugt, hierfür sind keine zusätzlichen Bauteile erforderlich. Die negative Spannung wird über den Pin 18 nach außen geführt [19]. Hier muss ein Potentiometer (R4, 10KΩ) als Spannungsteiler zum Einstellen des Kontrastes gegen Masse angeschlossen werden. Die eingestellte Spannung wird in den Pin 3 des LCDs zurückgeführt.

Das Display benötigt acht Datenleitungen, eine Enable-Leitung (E, positive Flanke zur Datenübernahme) und eine weitere Leitung (RS), die angibt, ob die anstehenden Daten als Zeichen auf dem Display ausgegeben werden sollen oder ob es sich um einen Steuerbefehl für das Display handelt. Das Display ist in zwei gleiche Hälften eingeteilt, die jeweils 64 * 64 Pixel groß sind. Jede der beiden Hälften hat eine eigene Chip Select-Leitung, mit der ausgewählt werden kann, welche der beiden Displayhälften die ankommenden Daten und Steuersignale verarbeiten soll. Diese Leitungen heißen CS1 und CS2. Des Weiteren besteht die Möglichkeit, beide Hälften zeitgleich anzusprechen.
Es können nicht nur Daten zum LCD gesendet, sondern auch aus ihm gelesen werden. Da hiervon kein Gebrauch gemacht wird, ist die Steuerleitung dieser Funktion (R/W, engl.: Read / Write, Pin 5) fest mit Masse verbunden. Somit ist die Datenrichtung vom Mikroprozessor zum LCD dauerhaft eingestellt.
Die Reaktionszeiten des LCD auf die Steuersignale liegen im Bereich von mehreren Hundert Mikrosekunden bis zu einer Millisekunde.
Die Signalabläufe, die zum Ansteuern des Displays unbedingt eingehalten werden müssen, werden im Kaptitel 7.4.2 beschrieben.

6.6 Tasterentprellung

Die drei Eingabetaster müssen unbedingt entprellt werden, damit diese nicht mehrere Impulse (Flanken) und damit mehrere Interrupts auslösen. Zwei Möglichkeiten stehen zur Verfügung: Hardwareentprellung oder Softwareentprellung. Die Softwareentprellung erfordert deutlich mehr Zeit während des realen Betriebes (z.B. Warteschleifen mit mehreren Millisekunden Wartezeit), was bei der vorliegenden Anwendung zu starken Zeitproblemen führen würde. Dagegen ist die Hardwareentprellung mit wenigen Bauteilen einfach zu realisieren und benötigt keinen zusätzlichen Zeitaufwand, da die Signale direkt flankengesteuerte Interrupts auslösen. Der Prozessor muss sich nur um das Abarbeiten der entsprechenden Interrupt Service Routinen kümmern. Durch eine solche Hardwareentprellung kommen auch deutlich weniger Störimpulse, die durch das Prellen entstehen können, in den empfindlichen Prozessor. Dieser könnte bei ungünstigen Störimpulsen stehen bleiben und müsste neu gestartet werden.

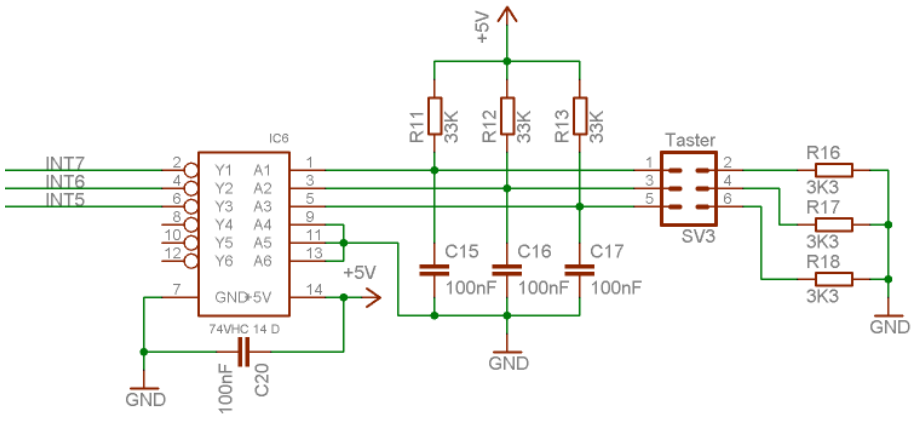


Abbildung 25: Hardwareentprellung für die drei Taster (nach [20])

Als Inverter wird der *74VHC14* verwendet, da dieser über Schmitt-Trigger-Eingänge verfügt. Diese sind für die Zustandserkennung der langsamen Auf- und Endladekurven der Kondensatoren unbedingt erforderlich.
Der Kondensator C20 filtert Spannungsspitzen aus der Versorgungsspannung des ICs heraus.

Alle drei Taster verwenden denselben Inverter-IC, der sechs Invertierungen zur Verfügung stellt. Die drei Ausgänge des ICs werden direkt mit den gewählten Interrupt-Eingängen des Mikroprozessors verbunden.

Jede der drei Tasten benötigt drei weitere Bauteile [20]:

- Taste „↑ / +“: R13, R18, C17
- Taste „↓ / -“: R11, R16, C15
- Taste „Enter“: R12, R17, C16

6.7 Schnittstelle zum Laserprojektor

Die beiden Achsenkanäle werden mit 12-Bit-DACs realisiert, damit die projizierten Grafiken möglichst gut abgebildet werden können. Für die Farbmischung werden mit Hilfe eines acht Bit breiten taktflankengesteuerten D-Flipflops (im Folgenden als „Flipflop“ bezeichnet), zweier Operationsverstärker und einiger Widerstände zwei einfache 4-Bit-DACs aufgebaut. Diese Auflösung genügt den Ansprüchen der Farbmischung voll und ganz. Für die symmetrische Signalübertragung werden weitere Operationsverstärker eingesetzt.
Die beiden Interlock-Leitungen werden direkt miteinander verbunden und erfordern keine weitere Erläuterung.
Das Shuttersignal wird ebenfalls mit einem Operationsverstärker (IC10B, siehe Abbildung 65 in Kapitel 13.2), der als Spannungsfolger mit der Verstärkung 1 arbeitet, entkoppelt, damit kein Prozessorpin direkt über den Steckverbinder von außen erreicht werden kann.
Alle weiteren Pins der 25-poligen D-SUB-Buchse werden mit 75Ω-Widerständen direkt an Masse angeschlossen.

6.7.1 12-Bit-DACs für die Strahlpositionierung

Der *AD7545 AKN* ist ein einfach anzusteuernder DAC mit einer Auflösung von 12 Bit. Er benötigt für den normalen Betrieb nur wenige weitere Bauteile [23].
Der Einsatz dieses DACs wird anhand der X-Achse (X-Kanal) verdeutlicht, die Y-Achse (Y-Kanal) ist identisch aufgebaut:

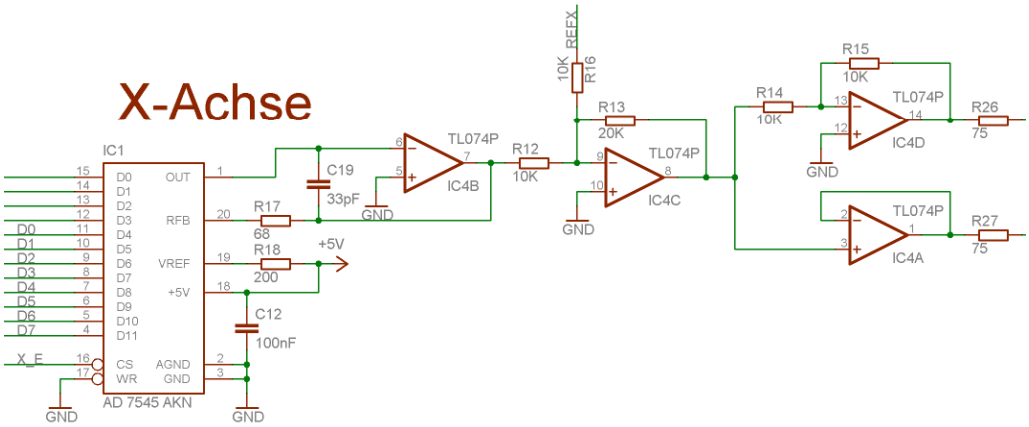


Abbildung 26: DAC-Schaltung der X-Achse mit 12-Bit-DAC

Die analoge Beschaltung mit R17, R18, C19 und IC4B ist dem Datenblatt entnommen [23].

Die Eingänge D4 bis D11 des IC1 sind direkt an den Datenbus angeschlossen. Die unteren vier Bits werden von einem vorgeschalteten Flipflop gepuffert (IC3, nicht mit abgebildet, siehe Abbildung 65 in Kapitel 13.2). X- und Y-Achse teilen sich ein Flipflop, da jeder der beiden Kanäle vier Bits benötigt. Als Flipflop wird ein *74HC574* verwendet.
Der Pin 16 des IC1 ist direkt an den Mikroprozessor angeschlossen (Signal X-DAC Enable).
Am Ausgang des IC4B liegt bei dem digitalen Eingangssignal (12 Bit) 0x000 eine Spannung von 0V an.
Die Ausgangsspannung bei dem digitalen Eingangssignal (12 Bit) 0xFFF wird wie folgt berechnet:

$$U_{IC4B\_Pin7} = -5V * \left(\frac{4095}{4096}\right) \sim -5V$$

Gleichung 19: Berechnung der Ausgangsspannung des DAC für 0xFFF (nach [23])

Der Kondensator C19 glättet Störspitzen und stabilisiert das analoge Ausgangssignal.
Der Kondensator C12 filtert Spannungsspitzen aus der Versorgungsspannung des ICs heraus.
Die auf den Ausgang des IC4B folgende Schaltung um IC4A, IC4C, IC4D, R12, R13, R14, R15 und R16 erzeugt aus dem unsymmetrischen und unipolaren Ausgangssignal ein differentielles und bipolares Signal. Die beiden Widerstände R26 und R27 werden zum Anpassen an die Leitungswiderstände der Verbindungsleitungen zum Lasersystem benötigt.

Der analoge Schaltungsteil wird mit PSpice simuliert. Die Schaltung der Simulation wird nicht erneut abgedruckt, da sie direkt der in Abbildung 26 dargestellten Schaltung entspricht. Die Simulation ergibt folgendes Ergebnis:

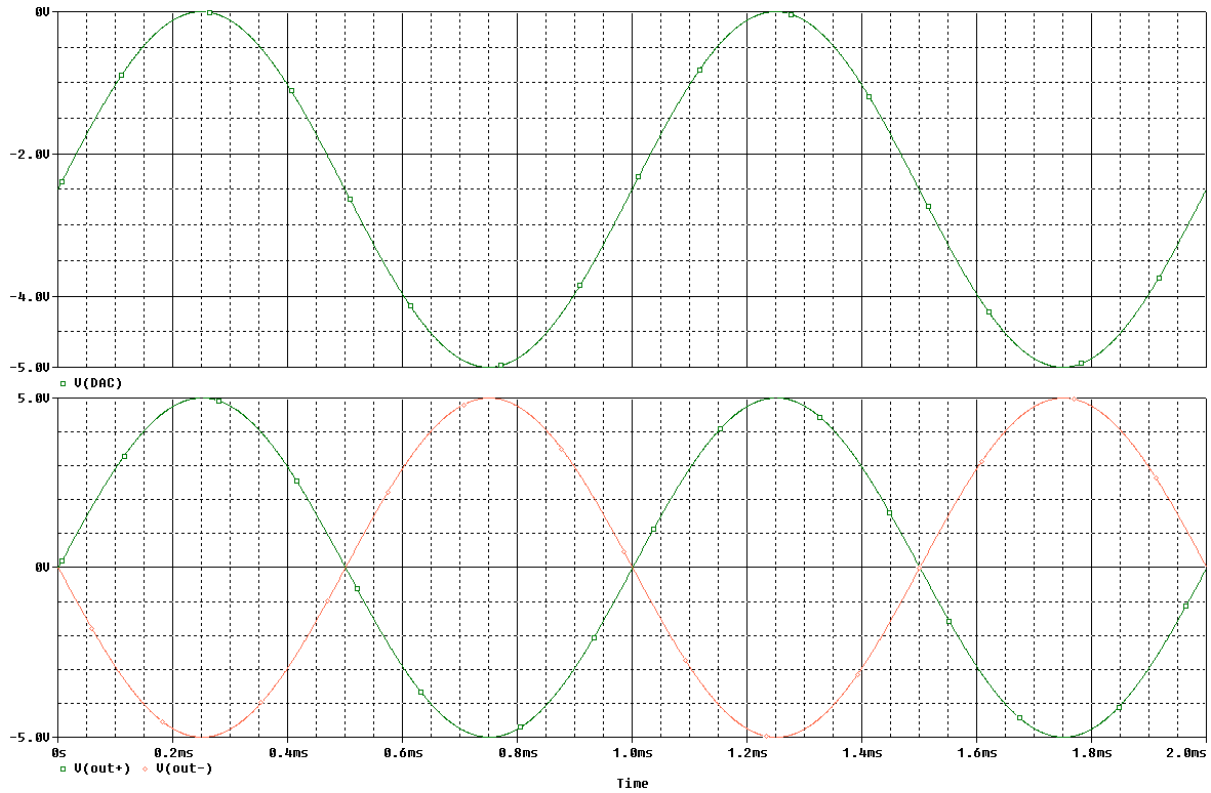


Abbildung 27: Simulationsergebnis der Symmetrierung der X-Achse (PSpice)

Die Spannungsverläufe sind über einen Zeitraum von 2ms dargestellt. Bei einer Frequenz von 1KHz sind zwei komplette Perioden abgebildet.
Der obere Plot zeigt die Ausgangsspannung des Operationsverstärkers IC4B. Diese Spannung wird zusammen mit der Referenzspannung für den X-Kanal (REFX) auf eine Addiererschaltung (IC4C, R12, R13, R16) geführt und um 2,5V angehoben. Außerdem verstärkt der Addierer das Signal um den Faktor –2. (Das Vorzeichen ist negativ, da der Addierer invertierende Wirkung hat.)

$$v_{IC4C} = -\frac{R13}{R12} = -\frac{20K\Omega}{10K\Omega} = -2$$

Gleichung 20: Berechnung der Verstärkung des Addierers IC4C

IC4A dient als Ausgangstreiber für das Signal X-. Dabei handelt es sich um einen Spannungsfolger mit der Verstärkung 1.
IC4D dient als Ausgangstreiber für das Signal X+. Da dieses Signal noch dieselbe Phase wie das Signal X- besitzt, muss es noch invertiert werden. Dazu kommt ein invertierender Verstärker mit einer Verstärkung von -1 zum Einsatz.
Der untere Plot der Simulation zeigt die beiden Ausgangsspannungen V(out+) = X+ und V(out-) = X-.
Die Referenzspannung REFX wird über einen separaten Festspannungsregler (IC6, *78L05Z*), einen einfachen Spannungsteiler (R31 und R32, je 10KΩ) und einen Operationsverstärker (IC10A), der als Spannungsfolger mit der Verstärkung 1 eingesetzt wird, generiert:

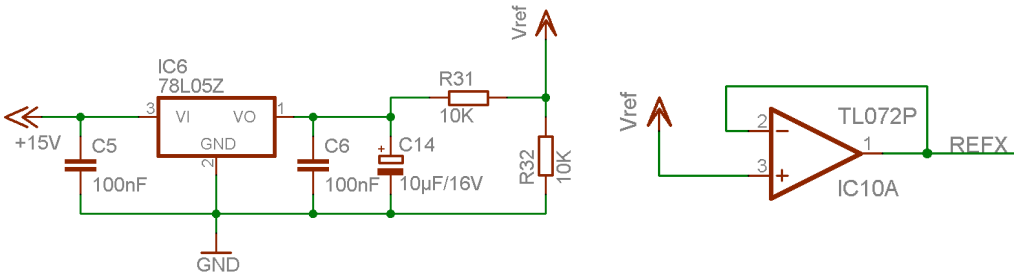


Abbildung 28: Schaltung für die Referenzspannungsquelle REFX

Die Kondensatoren C5 und C6 sollen ein eventuell auftretendes Schwingen des Festspannungsreglers unterdrücken.
Für eine komplette Unabhängigkeit hat die Y-Achse einen eigenen, separaten Operationsverstärker (IC9A) zum Entkoppeln ihrer Referenzspannung REFY.

6.7.2 4-Bit-DACs für die Farbmischung

DACs mit einer Genauigkeit von vier Bit sind im Handel schwer zu bekommen. Daher wird ein solcher DAC (beispielhaft für den roten Farbkanal) mit einem Flipflop (IC8, zum Speichern des Wertes), einem Operationsverstärker (IC7B) und fünf Widerständen (R1, R2, R3, R4, R5) realisiert. Der Aufbau (inkl. der Operationsverstärker für die anschließende Symmetrierung des Signals, IC7C, IC7D, R30, R33) wird in Abbildung 29 auf der folgenden Seite gezeigt.

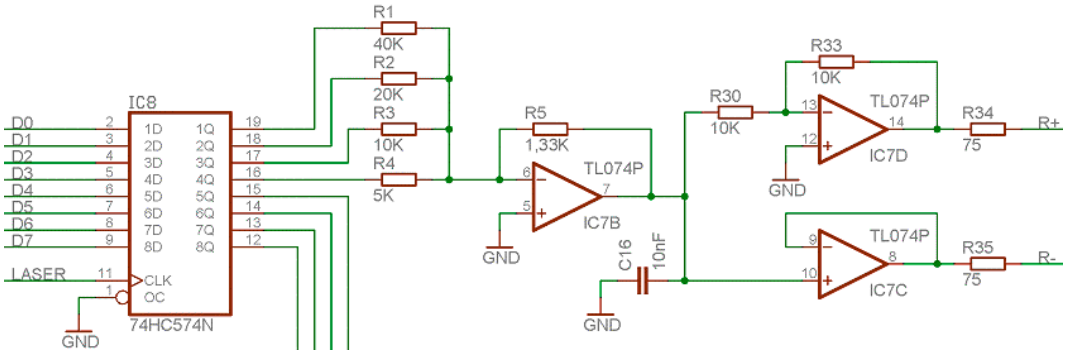


Abbildung 29: DAC-Schaltung des roten Farbkanals mit 4-Bit-DAC

Als Flipflop wird ein *74HC574* verwendet. Dessen Ausgangspegel sind entweder 0V (low) oder 5V (high). Die vier Bits werden über verschiedene Widerstände und dadurch mit verschiedenen Verstärkungen mit Hilfe eines Operationsverstärkers in Addiererschaltung überlagert. Als Widerstände für die vier Bits werden gewählt: R1 = 40KΩ, R2 = 20KΩ, R3 = 10KΩ und R4 = 5KΩ. Der Widerstand R1 ist für das LSB zuständig, während der Widerstand R4 für das MSB zuständig ist.
Die allgemeine Formel zum Berechnen der Ausgangsspannung eines Addierers lautet:

$$U_a = -R5 * \left(\frac{U_{E1}}{R1} + \frac{U_{E2}}{R2} + \frac{U_{E3}}{R3} + \frac{U_{E4}}{R4} \right)$$

Gleichung 21: Allgemeine Berechnung der Ausgangsspannung eines Addierers

Die Formel kann vereinfacht werden: U_{Ex} ist hier immer +5V, U_a ist hier –2,5V (negatives Vorzeichen, da der Addierer invertierende Wirkung hat). Die Werte werden in Gleichung 21 eingesetzt:

$$-2{,}5V = -R5 * \left(\frac{5V}{40K\Omega} + \frac{5V}{20K\Omega} + \frac{5V}{10K\Omega} + \frac{5V}{5K\Omega} \right)$$

Gleichung 22: Berechnung des Widerstandes R5 für den 4-Bit-DAC

Für R5 errechnet sich ein Wert von 1,333KΩ. In der E96-Reihe ist ein Widerstandswert von 1,33KΩ enthalten. Damit ist der 4-Bit-DAC komplett dimensioniert. Als Widerstände kommen für R1, R2, R3, R4 und R5 Metallschicht-Widerstände mit einer Toleranz von 1% zum Einsatz.
Da das Ausgangssignal R+ zwischen 0V und 2,5V bzw. das Ausgangssignal R- zwischen 0V und -2,5V liegen soll, muss hier kein zusätzlicher Offset hinzuaddiert werden.

Der Kondensator C16 wird zur Stabilisierung des analogen Signals und zum Glätten von Glitches im analogen Signal benötigt. Glitches können bei einem zeitgleichen Zustandswechsel von mehr als einem Ausgangsbit des Flipflops auftreten.

Die auf den Ausgang des IC4B folgende Schaltung um IC4A, IC4C, IC4D, R12, R13, R14, R15 und R16 erzeugt aus dem unsymmetrischen und unipolaren Ausgangssignal ein differentielles, unipolares Signal.
IC7C dient als Ausgangstreiber für das Signal R-. Dabei handelt es sich um einen Spannungsfolger mit der Verstärkung 1.

IC7D dient als Ausgangstreiber für das Signal R+. Da dieses Signal noch dieselbe Phase wie das Signal X- besitzt, muss es noch invertiert werden. Dazu kommt ein invertierender Verstärker mit einer Verstärkung von -1 zum Einsatz.

Die beiden Widerstände R26 und R27 werden zum Anpassen an die Leitungswiderstände der Verbindungsleitungen zum Lasersystem benötigt.

Die Dimensionierung wird mit PSpice kontrolliert. Die Schaltung der Simulation wird nicht erneut abgedruckt, da sie direkt der in Abbildung 29 dargestellten Schaltung entspricht. Die Simulation ergibt folgendes Ergebnis:

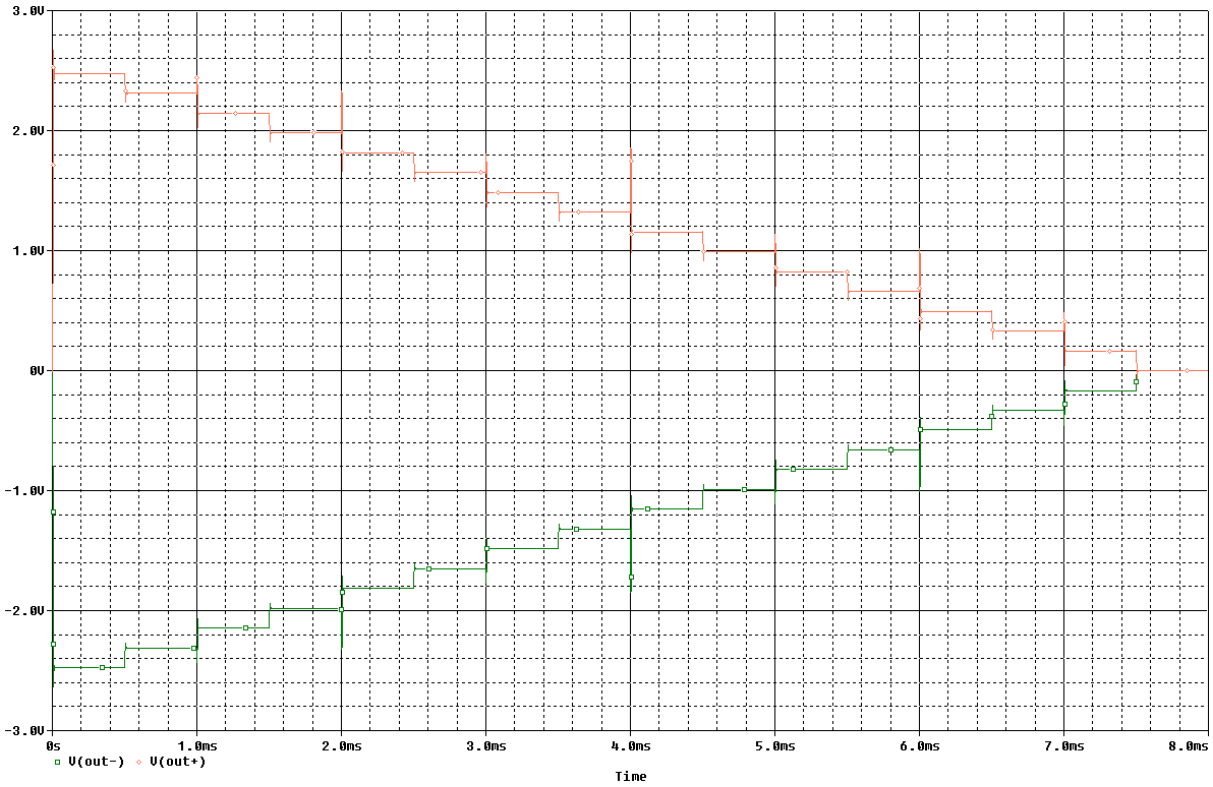


Abbildung 30: Simulationsergebnis der Symmetrierung des roten Farbkanals (PSpice)

Das Simulationsergebnis zeigt die beiden Ausgangsspannungen: V(out+) entspricht R+ und V(out-) entspricht R-. Es sind Glitches erkennbar, die jedoch bei Messungen am fertigen Gerät nicht erkannt werden können (siehe Abbildung 58 in Kapitel 9.2).
Wird der digitale Eingangswert (4 Bit) 0x0 angelegt, führen sowohl R+ als auch R- 0V. Die differentielle Spannung beträgt ebenfalls 0V.
Wird der digitale Eingangswert (4 Bit) 0xF angelegt, führt R+ einen Spannungspegel von +2,5V und R- einen Spannungspegel von –2,5V (beide Spannungen gegen Masse gemessen). Die differentielle Spannung beträgt +5V.
Bei jeder „Treppenstufe“ ändern sich die Signale R+ bzw. R- um ca. ±166,7mV. Die differentielle Spannung ändert sich damit pro „Treppenstufe“ um ±333,4mV.

Die Schaltung des grünen Farbkanals ist mit der des roten Farbkanals identisch. Der einzige Unterschied besteht in der Bitzuordnung am gemeinsamen Flipflop: Der rote Farbkanal verwendet die unteren und der grüne Farbkanal die oberen vier Bits. Somit wird ermöglicht, beiden Farbkanälen im selben Prozessortakt über den parallelen Datenbus einen neuen Datenwert zu übertragen.

6.8 Portbelegung des Mikroprozessors

Jeder I/O-Pin des Mikroprozessors verfügt über eine oder mehrere Sonderfunktion(en), die anstelle der normalen I/O-Funktion des Pins genutzt werden können.
Die übrigen elf Pins werden für die Versorgungsspannung, den externen Quarz zur Taktgenerierung und den externen Reset benötigt. Die Beschaltung dieser Pins wird direkt dem Datenblatt entnommen [13].
Die Auswahl der für die Kommunikation mit den einzelnen Baugruppen verwendeten Pins hängt von den erforderlichen Sonderfunktionen ab.
Drei der I/O-Pins werden nicht benötigt, diese bleiben offen.

Port	Pin	Pinfunktion	Peripherie
A[0..7]	44..51	AD0..7	Externer SRAM (A0..7)
B0	10	Output	SD-Karte Slave Select
B1	11	Output	SD-Karte SCK
B2	12	MOSI	SD-Karte MOSI
B3	13	MISO	SD-Karte MISO
B4	14	Input	SD-Karte Card Detect
B5	15	Input	SD-Karte Write Protection
B[6..7]	16..17	I/O	frei
C0	35	A8 Output	Externer SRAM A12
C1	36	A9 Output	Externer SRAM A15
C2	37	A10 Output	Externer SRAM A17
C3	38	A11 Output	Externer SRAM A13
C4	39	A12 Output	Externer SRAM A8
C5	40	A13 Output	Externer SRAM A9
C6	41	A14 Output	Externer SRAM A11
C7	42	A15 Output	Externer SRAM A10
D0	25	Output	LCD CS2
D1	26	Output	LCD CS1
D[2..3]	27..28	RxD1, TxD1	USB, FT232BL (optional)
D4	29	Output	DAC MSB-FF enable
D5	30	Output	Y-DAC enable
D6	31	Output	X-DAC enable
D7	32	Output	4-Bit-DACs enable (Farben)
E[0..1]	2..3	PDO, PDI	ISP
E2	4	Output	LCD E
E3	5	Output	LCD RS
E4	6	Output	LCD RST
E[5..7]	7..9	Input / IRQ	Taster (Interrupt 5..7)
F[0..7]	54..61	Output	Datenbus für LCD & DACs
G[0..1]	33..34	RD, WR	Externer SRAM
G2	43	ALE	Externer SRAM
G3	18	Output	Shutter
G4	19	I/O	frei

Tabelle 12: Portbelegung des Mikroprozessors (Port, Pin und Funktion nach [13])

6.9 Spannungsversorgung

Es werden vier verschiedene Gleichspannungen benötigt:
1. +5V (für den Betrieb des Mikroprozessors, des SRAM, des Latches, der drei Taster, des Grafikdisplays, der DACs und der optionalen USB-Schnittstelle)
2. +3,3V (für den Betrieb der SD-Karte)
3. +15V (für den Betrieb der Operationsverstärker der Schnittstelle zum Lasersystem nach *ILDA*)
4. -15V (für den Betrieb der Operationsverstärker der Schnittstelle zum Lasersystem nach *ILDA*)

Zur Spannungsversorgung des Gerätes soll ein handelsübliches Steckernetzteil verwendet werden, das anstelle der vier erforderlichen Einzelspannungen eine Spannung von ca. 8,5 bis 9VDC bereitstellt.

Die folgende Beschreibung bezieht sich auf den Schaltplan der Spannungsversorgung in Abbildung 69 im Anhang (Kapitel 13.2).

Die Diode D1 schützt das Gerät vor Beschädigungen, die durch ein Verpolen der Eingangsspannung entstehen können.
Für die erste Gleichspannung von +5V wird der Festspannungsregler *7805* (IC1) verwendet. Der Kondensator C1 dient zum Puffern der Eingangsspannung, während der Kondensator C2 als Puffer dieser ersten Versorgungsspannung (+5V) dient.
Für die zweite Gleichspannung von +3,3V wird der Festspannungsregler *LM3940 IT3,3* (IC2) verwendet. Als Puffer dieser zweiten Versorgungsspannung (+3,3V) dient der Kondensator C3.
Die Kondensatoren C4, C5, C6 und C7 sollen ein eventuell auftretendes Schwingen der Festspannungsregler unterdrücken.
Die verwendeten Operationsverstärker benötigen eine bipolare Versorgungsspannung mit ±15VDC. Diese beiden Spannungen werden von dem DC/DC-Wandler *AM2D-0515DZ* (DC1) erzeugt. Dieser IC benötigt eine Eingangsspannung von 5VDC und stellt an seinem Ausgang die Spannungen +15VDC bzw. –15VDC zur Verfügung [21]. Die Operationsverstärker benötigen keine hohen Ströme, so dass der von dem DC/DC-Wandler zur Verfügung gestellte Maximalstrom von 67mA [21] vollkommen ausreichend ist.

6.10 Testen der Hardware vor dem Platinenlayout

Die komplette Hardware wird zum Testen und Experimentieren vorerst auf zwei Lochrasterplatinen aufgebaut, damit alle Funktionsblöcke zur Sicherstellung der gewünschten Funktionen richtig dimensioniert werden können. Parallel dazu entstehen programmierte Ansteuerfunktionen für die Hardware, mit denen die Hardware in Betrieb genommen und das Zusammenspielen der Hardwarefunktionsgruppen untersucht werden kann. Viele dieser Funktionen sind Grundlage der Software.
Während des Aufbauens und ausgiebigen Untersuchens der Testplatinen entsteht auch nach und nach der Schaltplan. Nachdem alle Tests erfolgreich abgeschlossen sind, können dann daraus die Platinenlayouts erstellt werden. Bis auf die USB-Baugruppe werden alle Funktionen getestet.

6.11 Platzierung der Funktionsgruppen auf den Platinen

Die zur Verfügung stehende „Light Edition“ der Layoutsoftware *Eagle 4.16r2*, die kostenfrei von der Homepage des Herstellers *Cadsoft Computer GmbH* heruntergeladen werden kann [22], erlaubt nur eine maximale Platinengröße von 100mm * 80mm mit maximal zwei Leiterbahnebenen (Top und Bottom). Auf einer normalen Europlatine (160mm * 100mm) könnte die Schaltung komplett untergebracht werden. Eine Programmversion, die solche Abmessungen ermöglicht, wäre aber in der Anschaffung zu teuer.
Da die komplette Schaltung durch die Größenbeschränkung nicht auf einer einzelnen Platine untergebracht werden kann, muss überlegt werden, wie viele Platinen sinnvoll sind, welche Baugruppen auf welcher Platine untergebracht werden und wie die Platinen miteinander zu verbinden sind.
Die Versorgungsspannung und die Signale werden durch Flachbandkabel und Pfostensteckverbindungen zwischen den Platinen übertragen. Auch die drei Taster, das Display und die 25-polige D-SUB-Buchse werden mit Flachbandkabeln und Pfostensteckverbindungen an die Platinen angeschlossen.
Bei der Platzierung der Funktionsgruppen auf den Platinen müssen keine besonderen schaltungsrelevanten Regeln beachtet werden.
Sinnvollerweise werden drei Platinen verwendet.
Die Schaltpläne, Platinenlayouts und Stücklisten befinden sich im Anhang (Kapitel 13.2 und 13.3.2).

6.11.1 Laser-CPU-Board

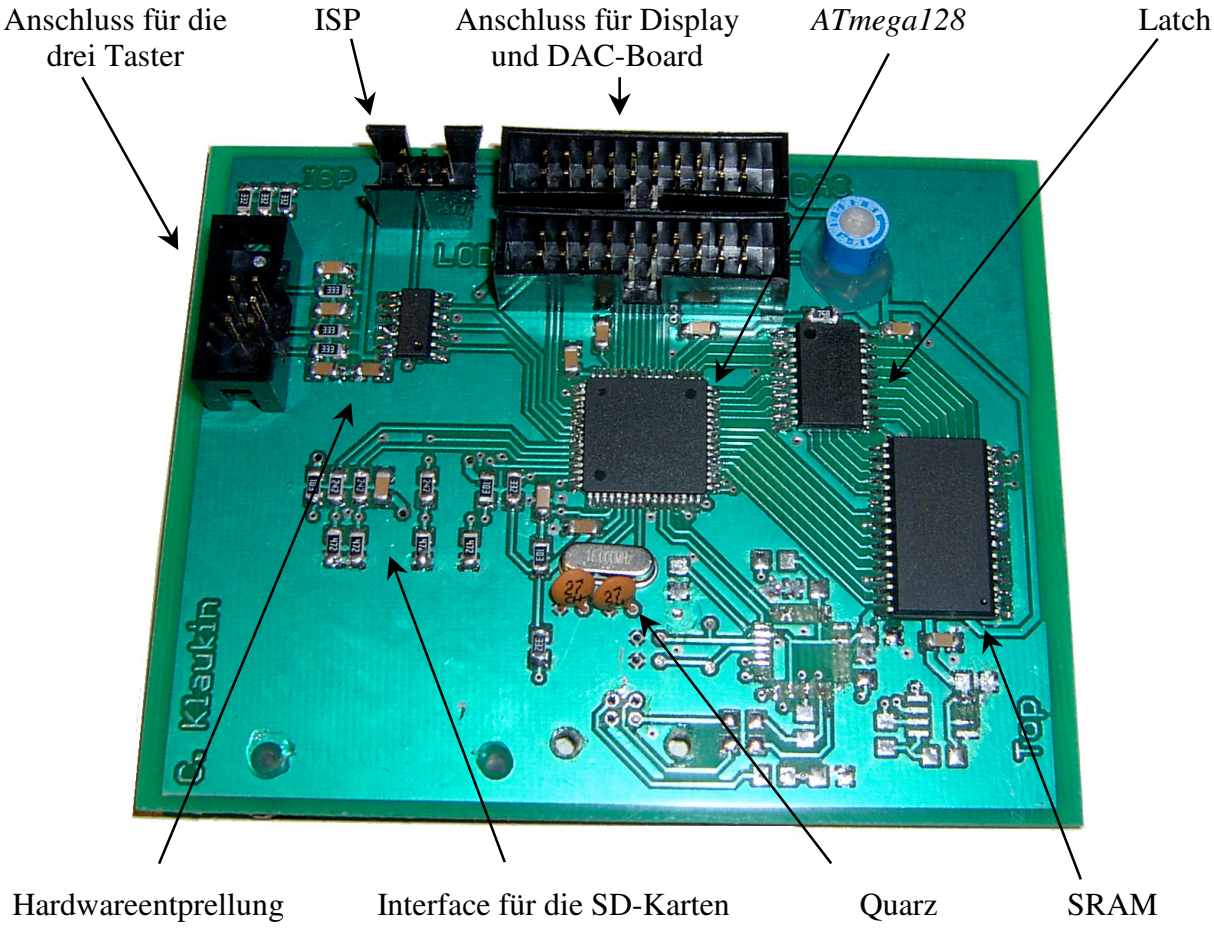


Abbildung 31: Laser-CPU-Board, fertig bestückte Platine (obere Seite, Top)

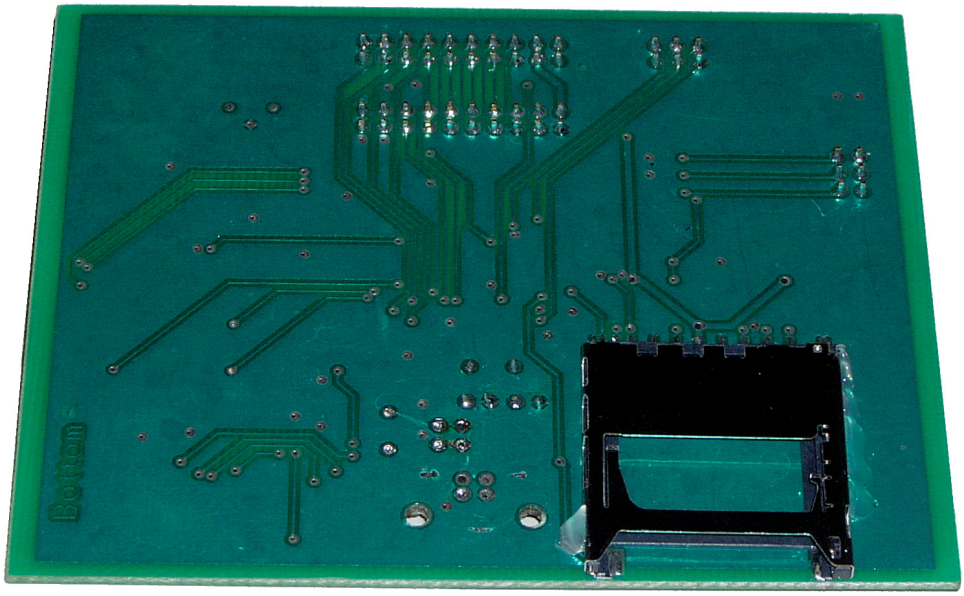


Abbildung 32: Laser-CPU-Board, fertig bestückte Platine (untere Seite, Bottom)

Auf der ersten Platine, dem Laser-CPU-Board, wird der Mikroprozessor untergebracht, ebenso der SRAM (inkl. Latch) und der Sockel für die SD-Karte. Auch die optionale USB-Schnittstelle findet hier ihren Platz. Da diese Baugruppen mit schnellen Übertragungsfrequenzen und einem hohen Datendurchsatz miteinander kommunizieren, muss der Abstand zwischen ihnen und damit die Länge der benötigten Leiterbahnen möglichst gering sein.
Diese Platine wird fast komplett mit Bauteilen im SMD-Gehäuse (engl.: Surface Mounted Devices) bestückt.
Die Anschlüsse für den ISP und für die drei Taster (inkl. der Hardwareentprellung) befinden sich ebenfalls mit auf dieser Platine.
Zwei 20-polige Pfostenstecker sind für den Anschluss des Displays und für den Anschluss an die zweite Platine vorgesehen.
Das Laser-CPU-Board hat keine direkte Anbindung an die dritte Platine (Spannungsversorgung). Sie erhält die benötigten Versorgungsspannungen über die 20-polige Verbindung zum DAC-Board, welches über eine zehnpolige Verbindung alle vier Spannungen von der dritten Platine erhält.
Die USB-Schnittstelle wird nicht mit bestückt. Die Lötpads zwischen SRAM und Quarz sind daher offen zu sehen.

6.11.2 DAC-Board

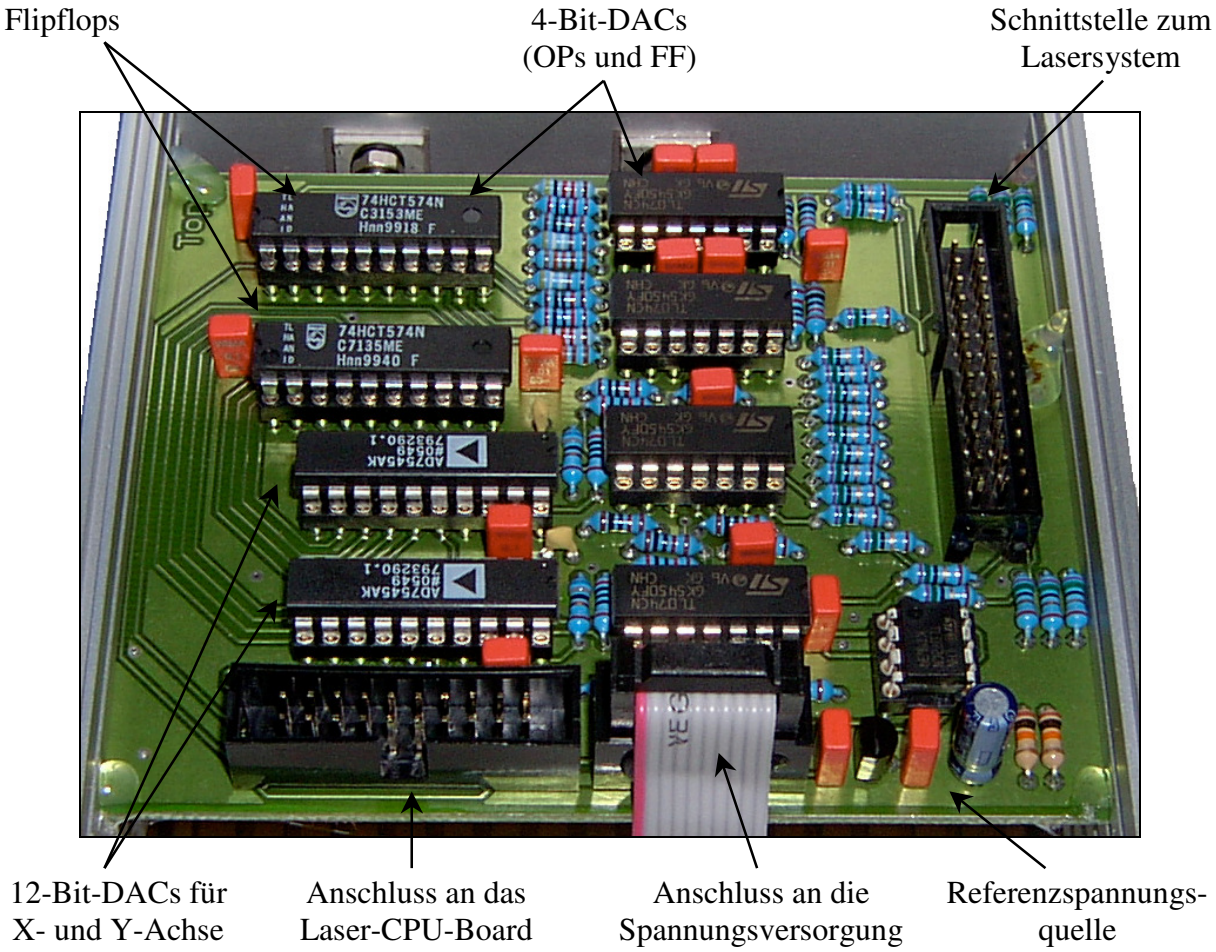


Abbildung 33: DAC-Board, fertig bestückte Platine (im Gehäuse eingebaut)

Auf der zweiten Platine, dem DAC-Board, sind sämtliche DACs untergebracht. Des Weiteren sind hier die Operationsverstärker und die 75Ω-Widerstände für die Schnittstelle zum Lasersystem (nach *ILDA*) untergebracht.
Ein 20-poliger Pfostenstecker ist für die Kommunikation mit der ersten Platine vorgesehen. Ein zehnpoliger Pfostenstecker stellt die Verbindung zu der dritten Platine her. Ein 26-poliger Pfostenstecker dient dem Anschluss der 25-poligen D-SUB-Buchse an die Schnittstelle zum Lasersystem, wobei der 26. Pin nicht verwendet wird.

6.11.3 Spannungsversorgung

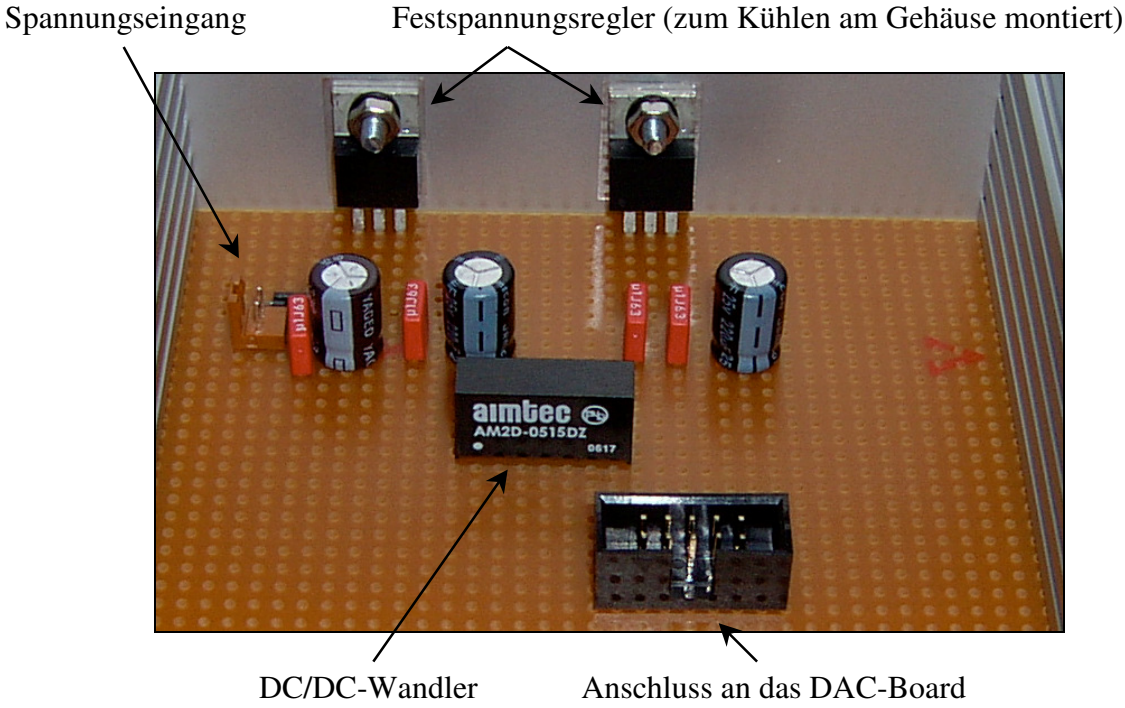


Abbildung 34: Spannungsversorgung (auf einer Lochrasterplatine, im Gehäuse montiert)

Auf der dritten Platine sind die Spannungsregler, die Pufferkondensatoren sowie die Verpolungsschutzdiode untergebracht. Ein zweipoliger Stecker stellt eine Verbindung zu einer Niederspannungsbuchse, über die ein Steckernetzteil seine Spannung einspeist, her.
Ein zehnpoliger Pfostenstecker dient der Verbindung mit der zweiten Platine.

Die Spannungsversorgung wird aus Kostengründen nicht auf einer professionell geätzten Platine aufgebaut, sondern auf einer handelsüblichen Lochrasterplatine.

7 Software

In diesem Kapitel werden die komplette Struktur, das Konzept sowie wichtige Hintergrundinformationen für das Verständnis der Software erläutert. Beispielhaft werden einige der komplexen Funktionen zudem mit Flussdiagrammen grafisch veranschaulicht.
Der komplette Quellcode umfasst 3661 Programm- und Kommentarzeilen. Er liegt dieser Diplomarbeit auf CD-ROM bei (siehe 13.4).

7.1 Modulare Struktur und Funktionsgruppen

Zur Realisierung werden sämtliche Teilaufgaben in kompakten Funktionen programmiert und nach ihrer Zugehörigkeit in einzelnen Quellcodedateien abgelegt. Durch dieses Konzept reduziert sich der Programmieraufwand deutlich, da viele Funktionen somit nur einmal programmiert werden müssen. Anschließend kann an beliebig vielen Stellen im Programm durch einen klar verständlichen und kurzen Aufruf auf diese Funktionen zugegriffen werden. Die Lesbarkeit des Quellcodes wird dadurch ebenfalls erleichtert, der Aufwand an zusätzlichen Kommentaren reduziert sich.
Die Unterteilung in Funktionsgruppen bedeutet hauptsächlich das sinnvolle Abspeichern der einzelnen Funktionen in die verschiedenen Quellcodedateien. Diese sind teilweise nach Hardwareteilen benannt, andere beinhalten z.B. das Menü und erhalten dadurch ihren Namen.
Alle Module werden so allgemein wie möglich programmiert, damit diese an verschiedenen Stellen für unterschiedliche Parameter eingesetzt werden können. Somit benötigen einige Funktionen neben einem Wert auch noch andere Parameter (z.B. Positionskoordinaten oder Sektoradressen). Durch diesen Ansatz wird eine hohe Flexibilität und eine einfache Erweiterung der Software ermöglicht.
In der folgenden Beschreibung der Software werden Variablen- und Funktionsnamen in quellcodetypischer Schrift dargestellt. Die hierfür gewählte Groß- und Kleinschreibung entspricht der im Quellcode verwendeten Schreibweise.

7.2 Struktur der *main.c*

Die *main.c* ist die Hauptdatei, in der das Hauptprogramm, das den genauen Ablauf vorgibt und steuert, gespeichert ist. In dieser Datei werden auch alle verwendeten Quellcodedateien und die Definitionsdatei *konst.h* eingebunden. Direkt in der Quellcodedatei *main.c* wird nur die globale Variable `*verz` definiert. Alle anderen Definitionen befinden sich aus Gründen der Übersichtlichkeit in separat angelegten Definitionsdateien.
Das Programm kann nur durch eine Unterbrechung der Spannungsversorgung beendet werden. Eine weitere Bedingung für ein Programmende ist nicht sinnvoll und daher auch nicht vorgesehen.
Die sechs wesentlichen Bestandteile des Hauptprogrammes sind

- die Einrichtung und Aktivierung des externen SRAMs,
- die Initialisierung der Hardware durch Aufrufen der einzelnen Initialisierungsfunktionen,
- die Speicherreservierung für die globalen Variablen `laser_puffer`, `v_id` und `verz` sowie für das Zeigerfeld `verz_inh[]`,
- das Laden der gespeicherten Parameterwerte aus dem prozessorinternen EEPROM,
- das Untersuchen der eingelegten SD-Karte sowie
- die Verwaltung der Menüanzeigen, wobei diese letzten beiden Punkte sich in der Hauptschleife befinden. Sie werden während des Betriebes je nach Bedarf wiederholt.

Das folgende Flussdiagramm zeigt den Ablauf des Hauptprogrammes:

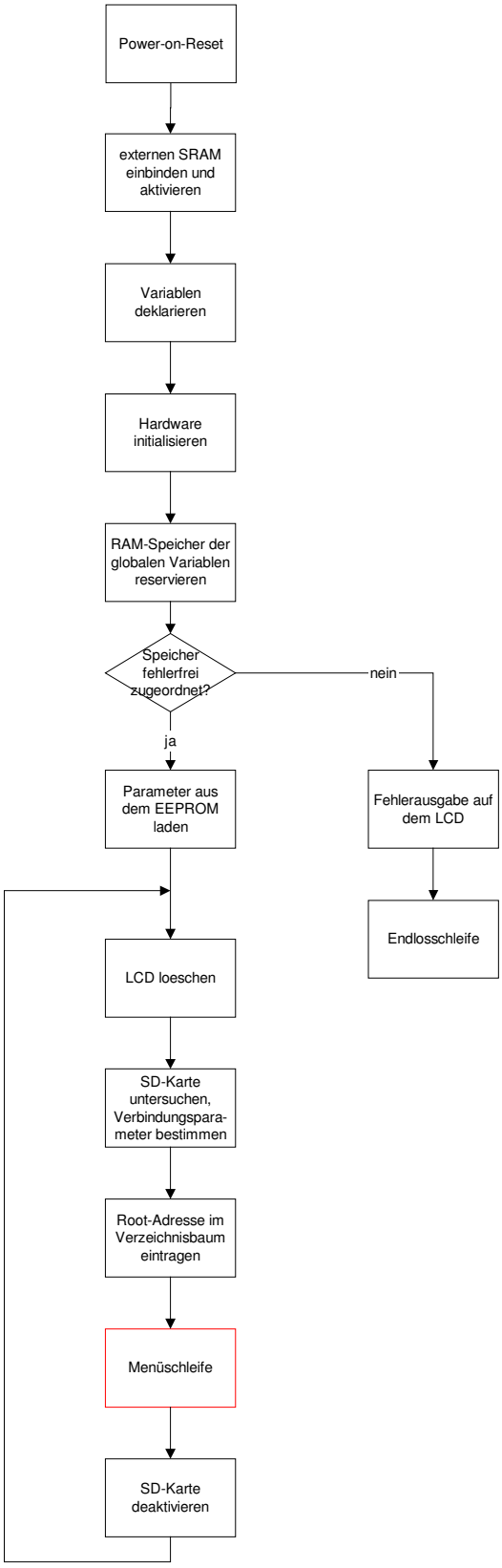


Abbildung 35: Flussdiagramm des Hauptprogrammes (*main.c*)

Der rot markierte Block enthält die Schleife, in der die Menüausgabe verwaltet wird. Aufgrund ihrer Komplexität wird sie in einem separaten Flussdiagramm dargestellt:

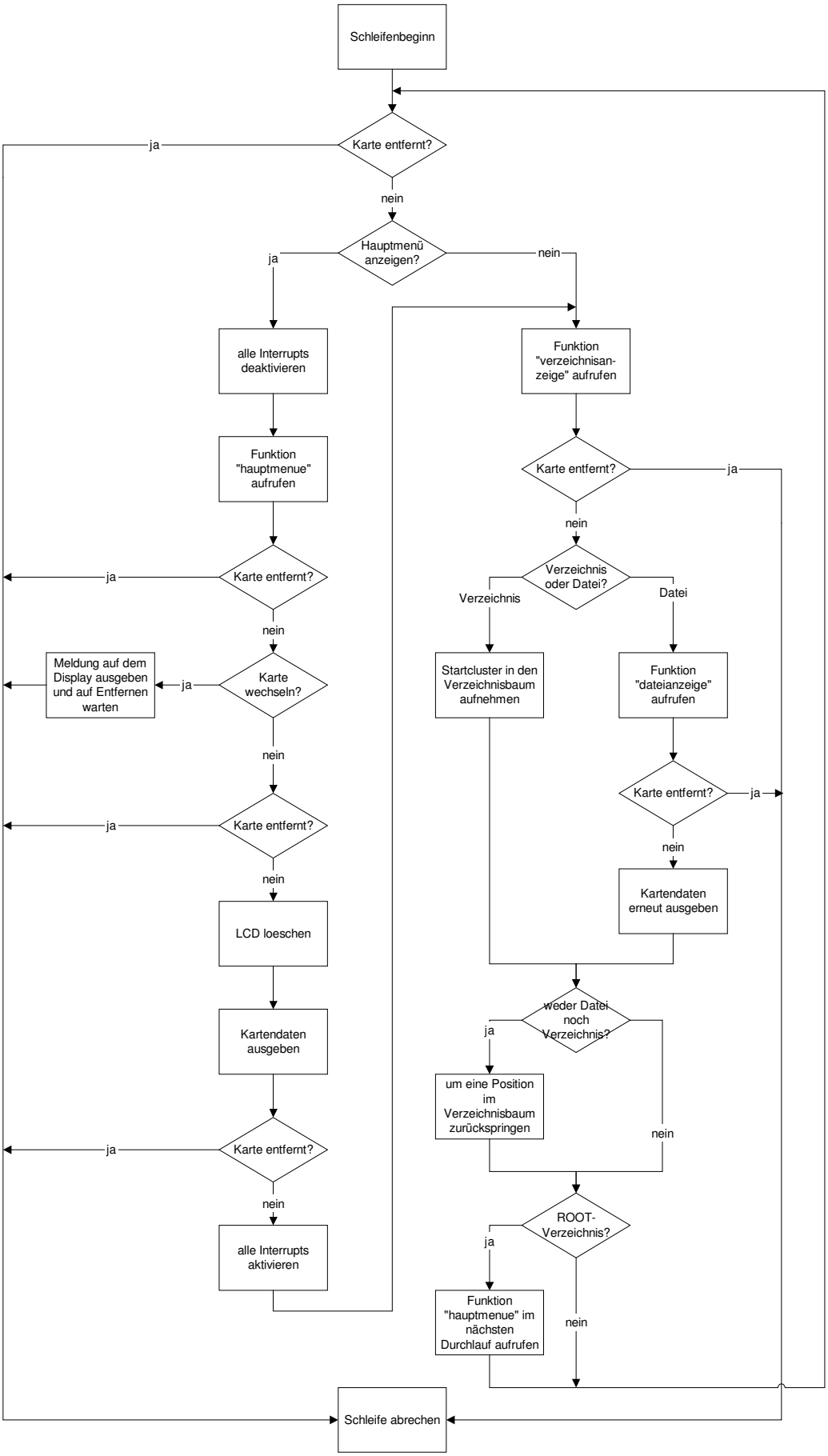


Abbildung 36: Flussdiagramm der Menüschleife (*main.c*)

Diese Menüschleife wird nur verlassen, wenn die SD-Karte entfernt wird.
In der Hauptschleife wird eine neu eingelegte SD-Karte untersucht und deren Verbindungs- sowie Dateisystemparameter ermittelt. Anschließend wird die Menüschleife gestartet. Hier wird bei Bedarf das Hauptmenü aufgerufen und im Anschluss bei Bedarf die Kartenentnahme vorbereitet. Außerdem wird hier die Verzeichnisausgabe aufgerufen. Der Verzeichnisbaum wird entsprechend aktualisiert. Wird eine Datei ausgewählt, wird die Funktion `dateiausgabe` aufgerufen (siehe 7.4.8).

7.3 Definierte Datentypen und Konstanten in der *konst.h*

Aufgrund der großen Anzahl an Definitionen werden diese in drei Unterkapitel aufgeteilt: Definierte Konstanten, definierte Datentypen und globale Variablen.

7.3.1 Definierte Konstanten

In der Definitionsdatei *konst.h* sind folgende allgemeine Definitionen vorhanden:
- `NEIN` = 0
- `JA` = 1

Für das Arbeiten mit dem Dateisystem FAT werden spezielle Konstanten benötigt:
- `FAT32` = 2
- `FAT16` = 3

Für das Handshake-Verfahren bei der Puffervariablenverwaltung während der Laserausgabe werden zwei weitere Konstanten benötigt:
- `STATUS_FERTIG_GELESEN` = 4
- `STATUS_LEER` = 5

Die Tasteninterrupts arbeiten mit derselben globalen Variablen (`taste`, siehe 7.4.3). Jede der drei Tasten setzt diese Variable über die zugehörige Interrupt Service Routine (ISR) auf einen konstanten Wert:
- `RAUF` = 6
- `ENTER` = 7
- `RUNTER` = 8

Die Farbausgabe des Lasers benötigt drei Konstanten für die Einstellung, wie ein reines Weiß ausgegeben werden soll. Da kein blauer Farbkanal vorhanden ist, gibt es nur drei Möglichkeiten: rot, grün oder gelb (rot + grün). Diese konstanten Werte werden auch für die Parameterspeicherung im prozessorinternen EEPROM verwendet:
- `EEPROM_FARBE_GRUEN` = 0
- `EEPROM_FARBE_ROT` = 1
- `EEPROM_FARBE_GELB` = 2

Für weitere Funktionen werden noch sechs andere Konstanten definiert:
- `DATEIFILTER` = „ILD“ (für das Dateifilter der Funktion `verzeichnisanzeige`)
- `ZEILENLAENGE` = 21 (für die Displayfunktionen)
- `MAX_VERZ` = 100 (maximale Anzahl gültiger Einträge in einem Verzeichnis)
- `ANZAHL_VERZEICHNISSE` = 10 (maximale Verzeichnistiefe des Verzeichnisbaumes)

- MAX_SEK_PUFFER = 60 (pufferbare Sektoren für die Laserausgabe)
- EEPROMGROESSE = 4095 (Größe-1 des prozessorinternen EEPROMs in Bytes)

7.3.2 Definierte Datentypen

Sieben spezielle Datentypen werden definiert.
Bei den einfachsten Datentypen handelt es sich um Unionen, mit deren Hilfe Variablen vom Typ `long` byteweise oder wortweise (entspricht dem Datentyp `short`) oder vom Typ `short` byteweise angesprochen werden können. Diese beiden Datentypen werden als `long_char` (belegt vier Bytes) bzw. `short_char` (belegt zwei Bytes) definiert.

Für einen einfacheren Umgang mit der Volume ID einer Partition des Dateisystems FAT wird eine Struktur definiert. Diese enthält elf Strukturelemente, von denen einige die beiden oben genannten Datentypen verwenden. Sie wird unter dem Namen `volume_id` definiert.

Der Aufbau eines Verzeichniseintrages wird in einer Struktur abgebildet (`verzeichnisinhalt`) und belegt 32 Bytes.

Um die Pixel aus einer *ILDA*-Datei besser zu verarbeiten, wird der Datentyp `ilda_aufbau` als Struktur definiert. Da nur das 3D-Format (siehe 2.2) unterstützt wird, enthält diese Struktur vier Strukturelemente, die alle vom Datentyp `short_char` sind. Somit belegt die Struktur acht Bytes.

Für den EEPROM-Zugriff wird eine „kleine“ Struktur definiert, die nur vier Variablen für die vier benötigten Parameter enthält.

Der letzte zu definierende Datentyp ist die Union `ausgabe_puffer`. Die Größe ist variabel und hängt direkt von der bereits definierten Konstanten `MAX_SEK_PUFFER` ab.
Die Union erlaubt einen byteweisen, sektor- oder pixelbasierten Zugriff. Hierzu sind drei Unionselemente notwendig:
- Das Feld `byte` ist ein eindimensionales Feld und besitzt `MAX_SEK_PUFFER` * 512 Einträge.
- Das Feld `block` ist ein zweidimensionales Feld und besteht aus `MAX_SEK_PUFFER` Sektoren, wobei jeder Sektor aus 512 Bytes besteht. Somit ist der Einsatz der bereits vorhandenen Lesefunktion möglich, die als Parameter einen Zeiger auf ein 512 Byte großes `unsigned char`-Feld benötigt (siehe 7.4.6).
- Das Feld `pixel` ist ein eindimensionales Feld und besitzt 64 * `MAX_SEK_PUFFER` Einträge des Datentyps `ilda_aufbau`.

Würde `MAX_SEK_PUFFER` auf 1 gesetzt werden, würde eine Variable dieses Datentyps 512 Bytes Speicher belegen. Da `MAX_SEK_PUFFER` hier allerdings auf den Wert 60 gesetzt wird, belegt eine Variable dieses Datentyps 30720 Bytes Speicher.

7.3.3 Globale Variablen

Da das vorliegende Programm sehr komplex ist und viele größere Variablen über Zeiger angesprochen werden müssen, werden mehrere globale Variablen verwendet, um die Parameterlisten der Funktionen nicht unnötig in die Länge zu ziehen. Für den Zugriff sind Zeiger erforderlich, weil der externe RAM ausschließlich für den Heap zur Verfügung steht.
Die meisten der globalen Variablen dienen der Kommunikation mit den Interrupt Service Routinen.

Globale Variablen, die über Zeiger angesprochen werden, sind:
- `*v_id`, Datentyp `volume_id` (enthält die relevanten Informationen der Partition, die aus deren Volume ID gelesen bzw. aus einigen ihrer Werte berechnet werden, siehe 7.4.6)
- Feld `*verz_inh`, Datentyp `verzeichnisinhalt` (enthält die gültigen Einträge des aktuell angezeigten Verzeichnisses, siehe 7.4.7)
- `*laser_puffer`, Datentyp `ausgabe_puffer` (wird für das Puffern während der Laserausgabe benötigt, siehe 7.4.8 und 7.4.9)

Drei weitere globale Variablen werden nicht als Zeiger definiert:
- Feld `verzeichnisbaum` (enthält die Startadressen der Verzeichnisse, siehe 7.4.7)
- `parameter`, Datentyp `eeprom` (enthält die vier veränderbaren Parameter, die im internen EEPROM gespeichert werden, siehe 7.4.1 und 7.4.4)
- `karte_entfernt` (wird gesetzt, wenn erkannt wird, dass die Karte nicht mehr im Sockel ist)

Für die drei Interrupt Service Routinen (ISR) der Taster wird eine globale Variable benötigt:
- `taste` (siehe 7.4.1 und 7.4.3)

Neun Variablen werden für die Kommunikation mit der ISR des verwendeten 16-Bit-Timers benötigt (siehe 7.4.8 und 7.4.9). Einige dienen als Zähler innerhalb der ISR, deren Zählerstände bei folgenden Aufrufen der ISR weiter verwendet werden können. Die benötigten Variablen sind:
- `statusfeld` (enthält den Speicherstatus der einzelnen Blöcke der Variablen `*laser_puffer`)
- `anzahl_pixel` (Pixel im aktuellen Frame)
- `pixelzaehler` (Zähler der ausgegebenen Pixel)
- `neuer_frame`
- `block` (enthält die aktuelle Blocknummer innerhalb des Laserpuffers)
- `pixel_im_sektor_zaehler` (zählt die Pixelnummer im aktuellen Sektor)
- `abbruch_taste` (wird gesetzt, wenn die Ausgabe durch eine Taste abgebrochen werden soll)
- `timer_interrupt_temp` (wird für die Farbausgabe benötigt)
- `tempfarbe` (wird für die Farbausgabe benötigt)

7.4 Einzelne Quellcodedateien

Im Folgenden werden die Inhalte der einzelnen Quellcodedateien genauer erklärt. Die einzelnen Funktionen werden aufgezählt, ihre Aufgabe und die Vorgehensweise erläutert.

7.4.1 Das Menü (*menue.c*)

Die Funktionen für die Anzeige der verschiedenen Menüs sind in einer Datei zusammengefasst. Die Datei enthält die folgenden fünf Funktionen:
- `startbildschirm`
  Die Funktion deaktiviert alle Interrupts, löscht den aktuellen Inhalt des Grafikdisplays und gibt den Begrüßungstext aus. Im Anschluss wartet die Funktion in einer `while`-Schleife (Warteschleife), bis eine der drei Tasten gedrückt wird.
  Die Darstellung auf dem Grafikdisplay wird in der Abbildung 37 gezeigt.

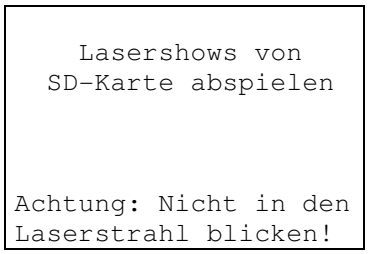


Abbildung 37: LCD-Ausgabe der Funktion `startbildschirm` (*menue.c*)

- `kartendaten_ausgeben`
  Die Funktion ruft die Funktion `kartenname_auslesen` (siehe 7.4.7) auf und gibt den ermittelten Kartennamen auf dem Display aus, ebenso die Kopfzeile der Dateitabelle. Diese Ausgabe belegt die obersten beiden Zeilen des Displays. Die Funktion wird nur von der Hauptfunktion aufgerufen, die im Anschluss ebenfalls die `verzeichnisanzeige` aufruft (siehe unten).
  Die Striche in der folgenden Abbildung deuten auf die Platzierung des Kartennamens hin:

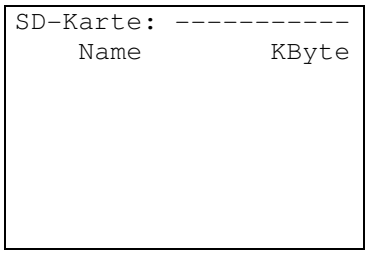


Abbildung 38: LCD-Ausgabe der Funktion `kartendaten_ausgeben` (*menue.c*)

- `verzeichnisanzeige`
  Die Funktion erhält als Übergabeparameter den Startsektor des Clusters, in dem das anzuzeigende Verzeichnis beginnt. Durch den Aufruf der Funktion `verzeichnis_puffern` wird das Verzeichnis von der SD-Karte gelesen. Die Anzahl der Ordner und der gültigen, gefilterten Dateien wird ermittelt. Die Dateien und Ordner werden sortiert und ausgegeben.
  Die ersten beiden Zeilen in der nachfolgenden Abbildung werden vorher von der Funktion `kartendaten_ausgeben` (siehe oben) ausgegeben. Diese werden für die vollständige Darstellung der Verzeichnisanzeige mit in diese Abbildung aufgenommen:

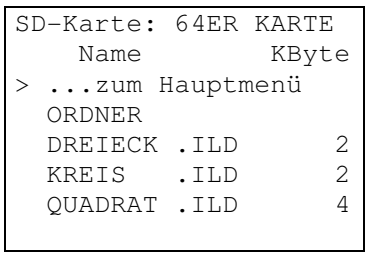


Abbildung 39: LCD-Ausgabe der Funktion `verzeichnisanzeige` (*menue.c*)

Da diese Funktion sehr komplex ist, wird auf eine detaillierte Erläuterung verzichtet und stattdessen das Flussdiagramm auf der nächsten Seite zur Veranschaulichung verwendet:

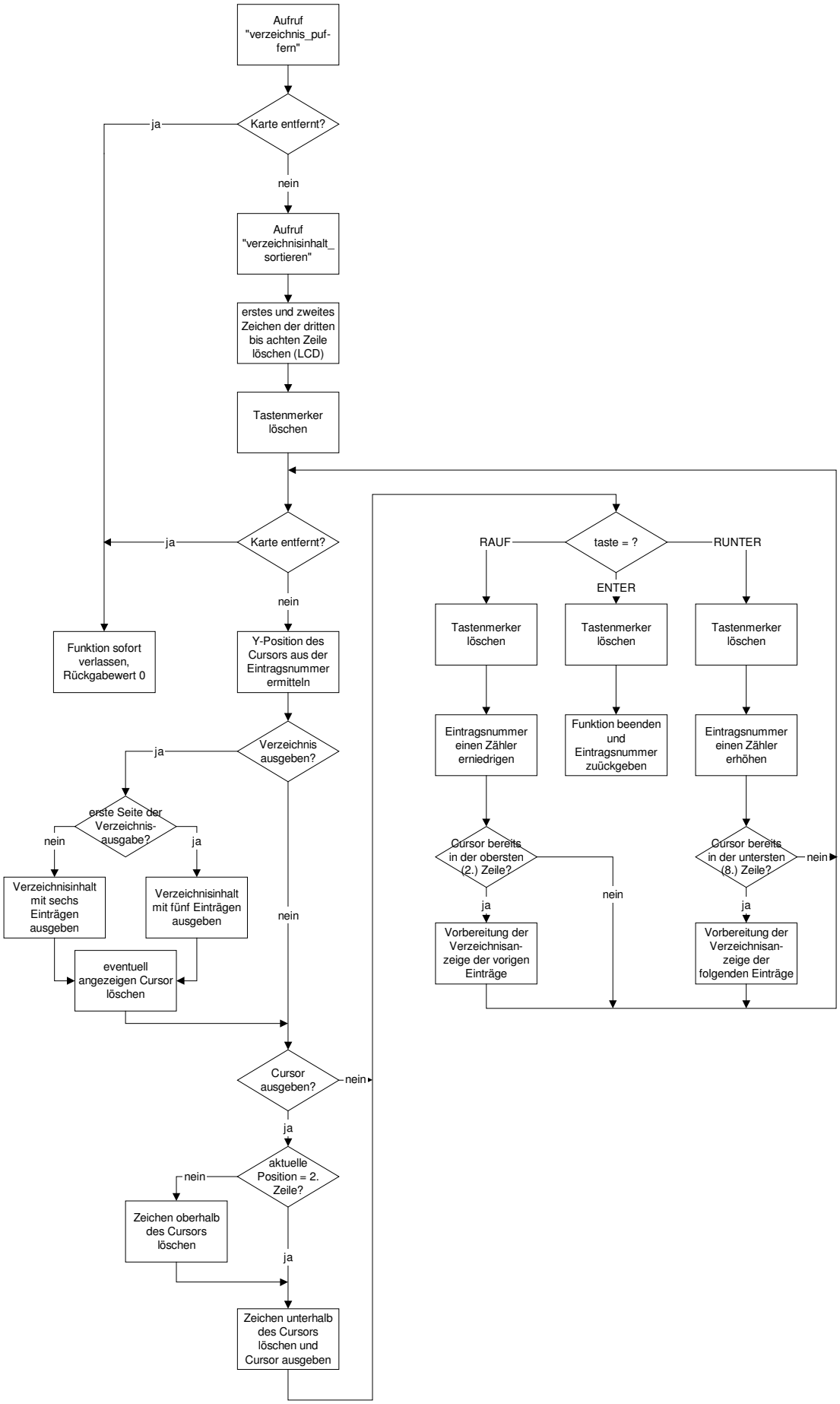


Abbildung 40: Flussdiagramm der Funktion `verzeichnisanzeige` (*menue.c*)

- `dateianzeige`
  Diese Funktion ist 241 Zeilen lang und kann aufgrund ihrer Komplexität nicht in einem Flussdiagramm dargestellt werden, da dieses entweder unleserlich werden würde oder unübersichtlich über mehrere Seiten verteilt wäre.
  Die Dateianzeige benötigt einen Lesepuffer zum Puffern eines kompletten Sektors von der SD-Karte. An dieser Stelle wird als Vorbereitung einer eventuell folgenden Laserausgabe der Ausgabepuffer `*laser_puffer` durch das Aufrufen der Funktion `laserpuffer_loeschen` (siehe 7.4.8) gelöscht.

  Ab hier beginnt eine `while`-Schleife, die sämtliche weiteren Funktionsteile beinhaltet:
  1. Die Zeigerposition wird auf die sechste Zeile festgelegt und der Inhalt des Grafikdisplays gelöscht.
  2. Der Dateiname, die Dateierweiterung sowie die Dateigröße in Kilobyte werden auf das Display geschrieben.
  3. Der Startsektor, an dessen Position die Datei beginnt, wird berechnet. Im Anschluss wird der Startsektor von der SD-Karte ausgelesen und gepuffert.
  4. Wenn die Anzahl der Punkte im ersten Frame größer als null ist, der Frame im 3D-Format vorliegt und die Frame Signatur *ILDA* der Inhalt der ersten vier Bytes ist, werden der Frame Name und der Company Name zusammenhängend auf dem Grafikdisplay ausgegeben. Die eingestellte Auflösung und die gewählte Scangeschwindigkeit erscheinen ebenfalls auf dem Grafikdisplay.
  5. Das Ausgabemenü wird angezeigt und der Tastenmerker gelöscht.

  Als Beispiel wird die Datei *dreieck.ild* (siehe CD-ROM) für die Darstellung auf dem Grafikdisplay gezeigt:

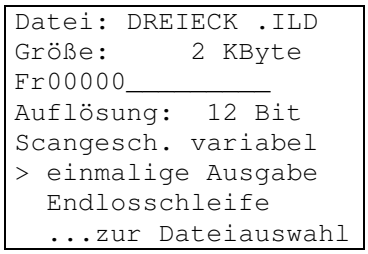


Abbildung 41: LCD-Ausgabe (Beispiel) der Funktion `dateianzeige` (*menue.c*)

  6. An dieser Stelle beginnt eine weitere `while`-Schleife, in der auf Tastendruck die Position des Cursors verändert wird (Rauf bzw. Runter) und der durch den Cursor ausgewählte Menüpunkt ausgeführt wird (Enter):
     - Befindet sich der Cursor beim Drücken der Enter-Taste in der sechsten Displayzeile, wird der angezeigte Displayinhalt gelöscht und der Name der aktuellen Datei ausgegeben. Im nächsten Schritt wird die Funktion `dateiausgabe` (siehe 7.4.8) für die einmalige Ausgabe der Datei aufgerufen. Nach dem Ende der Ausgabe wird die Unterschleife verlassen, die vorige Darstellung des Grafikdisplays wiederhergestellt und erneut mit der Cursorpositionierung begonnen. (Die Ablaufschritte 1 bis 6 wiederholen sich bis zur Auswahl der 8. Displayzeile.)
     - Befindet sich der Cursor beim Drücken der Enter-Taste in der siebten Displayzeile, wird der angezeigte Displayinhalt gelöscht und der Name der aktuellen Datei ausgegeben. Im nächsten Schritt wird die Funktion `dateiausgabe` (siehe 7.4.8) für die Ausgabe der Datei in einer Endlosschleife aufgerufen. Nach dem Ende der Ausgabe durch Abbrechen per Tastendruck wird die Unterschleife verlassen, die vorige Darstellung des Grafikdisplays wiederhergestellt und erneut mit der

Cursorpositionierung begonnen. (Die Ablaufschritte 1 bis 6 wiederholen sich bis zur Auswahl der 8. Displayzeile.)
- Befindet sich der Cursor beim Drücken der Enter-Taste in der achten Displayzeile, werden beide offenen Schleifen verlassen, es wird zum Hauptmenü (Verzeichnisanzeige) zurückgekehrt.

Wenn die Datei jedoch nicht die gültige Dateisignatur aufweist, der erste Frame leer ist oder nicht im 3D-Format vorliegt, wird folgende Meldung auf dem Display ausgegeben:

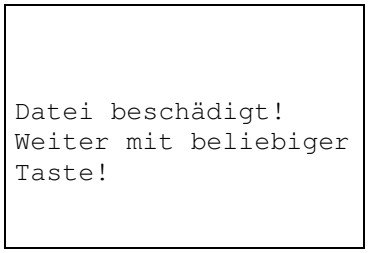


Abbildung 42: LCD-Ausgabe im Fehlerfall der Funktion `dateianzeige` (*menue.c*)

Die Meldung muss durch das Drücken einer beliebigen Taste quittiert werden. Im Anschluss wird die Funktion beendet und der Programmablauf kehrt zum Hauptmenü (Verzeichnisanzeige) zurück.

Aus Sicherheitsgründen wird an mehreren entscheidenden Stellen innerhalb dieser Funktion abgefragt, ob die SD-Karte eventuell entfernt wurde. Ist dieses der Fall, wird die Funktion mit einer entsprechenden Meldung auf dem schnellsten Wege verlassen.

- `hauptmenue`
  Diese Funktion ist 282 Zeilen lang und kann aufgrund ihrer Komplexität nicht in einem Flussdiagramm dargestellt werden, da dieses entweder unleserlich werden würde oder unübersichtlich über mehrere Seiten verteilt wäre.
  Im Hauptmenü stehen drei Punkte zur Auswahl. Es erscheint auf dem Grafikdisplay folgendermaßen:

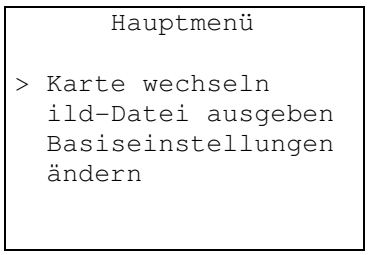


Abbildung 43: Darstellung des Hauptmenüs auf dem Grafikdisplay (*menue.c*)

Die dritte, vierte und fünfte Zeile können ausgewählt werden:
- Wird `Karte wechseln` ausgewählt, wird die SD-Karte deaktiviert und auf dem Display in der letzten Zeile die folgende Mitteilung ausgegeben: `bitte Karte entnehmen`.
- Wird `ild-Datei ausgeben` ausgewählt, wird die Funktion `verzeichnisanzeige` aufgerufen.
- Wird `Basiseinstellungen ändern` ausgewählt, wird ein anderes Menü angezeigt. Dieses wird im Anschluss an dessen Darstellung in Abbildung 44 erläutert.

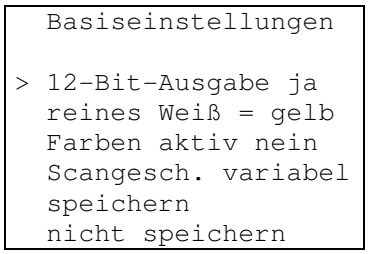


Abbildung 44: LCD-Ausgabe der Basiseinstellungen (Funktion `hauptmenue`, *menue.c*)

Der Cursor kann in diesem Menü innerhalb der dritten bis achten Zeile bewegt werden. Wird die Enter-Taste gedrückt, wechselt der Menüpunkt seinen Wert (dritte bis sechste Zeile) oder das Menü wird verlassen (siebte und achte Zeile). Nach dem Verlassen dieses Menüs wird das Hauptmenü erneut aufgebaut.

- Die Ausgabe kann entweder mit einer Genauigkeit von zwölf Bit oder mit acht Bit erfolgen. Der Parameter `12-Bit-Ausgabe` ist entweder `ja` oder `nein`.
- Da kein blauer Farbkanal eingebaut ist, kann ausgewählt werden, wie ein in einer *ILDA*-Datei gespeicherter weißer Punkt eingefärbt werden soll. Der Parameter `reines Weiß` ist entweder `rot`, `gelb` oder `grün`.
- Die Farbwiedergabe wird manuell ein- oder ausgeschaltet. Der Parameter `Farben aktiv` ist entweder `ja` oder `nein`.
- Die auszugebenden Pixel pro Sekunde werden entweder anhand der Anzahl der in den Frames vorhandenen Punkte berechnet oder auf einen festen Wert gesetzt. Der Parameter `Scangesch.` ist entweder `fest` oder `variabel`.

- Wird der Menüpunkt `speichern` ausgewählt, werden die vier Parameter im prozessorinternen EEPROM mit der Funktion `parameter_speichern` gespeichert (siehe 7.4.4).
- Wird der Menüpunkt `nicht speichern` ausgewählt, werden die vier Parameter erneut aus dem prozessorinternen EEPROM mit der Funktion `parameter_laden` geladen (siehe 7.4.4) und somit die eventuell vorgenommenen Änderungen verworfen.

7.4.2 Das Grafikdisplay (*lcd.c* und *zeichensatz.h*)

Da das Grafikdisplay über keinen eigenen Zeichensatz verfügt, muss dieser manuell erstellt werden. Die Datei *zeichensatz.h* enthält die 90 implementierten Zeichen in einem zweidimensionalen `unsigned char`-Feld. Die Zeichen sind:

- 0 – 9
- A – Z
- a – z
- ! “ # $ % & ‘ ( ) * + , - . / \ : ; < = > ? _
- Leerzeichen
- ä, ö, ü, ß

Jedes Zeichen besteht aus fünf Bytes. Unterhalb eines Zeichens bleibt eine Pixelreihe frei. Somit ist ein Zeichen sieben Pixel hoch und fünf Pixel breit. Abbildung 45 auf der folgenden Seite zeigt den Aufbau des Buchstabens M:

Bitnummer	dez. Wert	Bytenummer				
		0	1	2	3	4
0 (LSB)	1	■				■
1	2	■	■		■	■
2	4	■		■		■
3	8	■		■		■
4	16	■				■
5	32	■				■
6	64	■				■
7 (MSB)	128					

Abbildung 45: Beispiel des programmierten Zeichensatzes für das Grafikdisplay

Für das M werden die fünf Dezimalzahlen 127, 2, 12, 2, 127 manuell ermittelt und in das zweidimensionale Feld in der Datei *zeichensatz.h* eingetragen. Sämtliche anderen Zeichen werden nach derselben Vorgehensweise erstellt.
Die Umsetzung des Zeichensatzes geschieht in der Funktion `zeichen_schreiben`, die im weiteren Verlauf dieses Kapitels erläutert wird.

Für das Ansteuern des Grafikdisplays müssen eine bestimmte Signalreihenfolge und die vorgegebenen minimalen Wartezeiten unbedingt eingehalten werden, da das Display ansonsten unzuverlässig arbeiten könnte.
Die Signalverläufe des Ansteuervorganges werden im Folgenden gezeigt und erläutert:

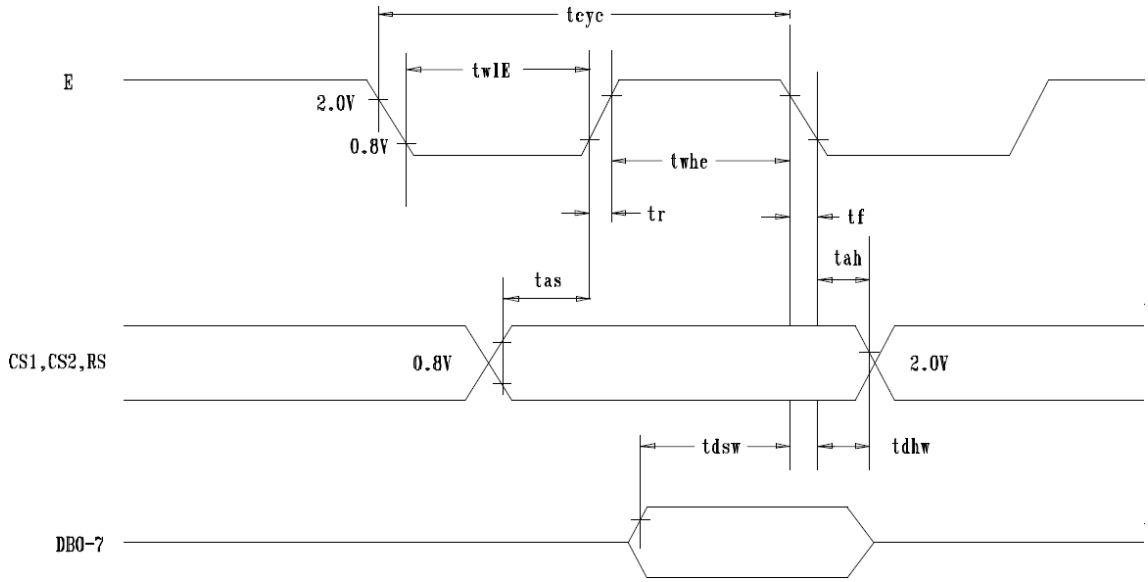


Abbildung 46: Timing des Grafikdisplays beim Schreibzugriff [19, bearb.]

Da jeder Zugriff auf das Display ein Schreibzugriff ist, wird der Lesezugriff hier nicht behandelt. Die entscheidenden Zeiten sind in der nachfolgenden Tabelle aufgeführt. Diese werden von den einzelnen Funktionen entsprechend umgesetzt und immer eingehalten. (Die Zeiten, die nicht aufgeführt sind, sind nicht relevant und müssen daher nicht weiter berücksichtigt werden.)

Bezeichnung	min. Wartezeit [19]	verwendete Wartezeit
Adress setup time t_{as}	140ns	~250ns, `wait_viertel_usec`
Adress hold time t_{ah}	10ns	>1 Prozessortakt ≥ 62,5ns
E high level witdh t_{whE}	450ns	~500ns, `wait_halbe_usec`
E low level witdh t_{wlE}	450ns	~500ns, `wait_halbe_usec`
Data setup time t_{ds}	200ns	~250ns, `wait_viertel_usec`
Data hold time t_{dh}	10ns	>1 Prozessortakt ≥ 62,5ns

Tabelle 13: vorgegebene und verwendete Wartezeiten bei der LCD-Ansteuerung [nach 19]

Die beiden Zeiten t_{ds} und t_{dh} sind in der Abbildung 46 nicht dargestellt.

Die Quellcodedatei *lcd.c* enthält 15 Funktionen, von denen zehn direkte Treiberfunktionen sind. Die anderen fünf greifen auf die Treiberfunktionen zu, um Daten an das Display zu senden:

- `LCD_on` [nach 19]
  Die Funktion schaltet das Display ein. Alle Änderungen sind im eingeschalteten Zustand direkt sichtbar. Beide Displayhälften werden gleichzeitig angesprochen (CS1 = CS2 = 1). Das Signal RS wird auf 0 gesetzt, da das gesendete Byte als Befehl interpretiert werden muss. Das zu sendende Byte ist 0x3F.

- `LCD_off` [nach 19]
  Die Funktion schaltet das Display aus. Alle Änderungen sind im ausgeschalteten Zustand nicht sichtbar. Beide Displayhälften werden gleichzeitig angesprochen (CS1 = CS2 = 1). Das Signal RS wird auf 0 gesetzt, da das gesendete Byte als Befehl interpretiert werden muss. Das zu sendende Byte ist 0x3E.

- `set_x_CS1` [nach 19]
  Die Funktion gibt der ersten (linken) Displayhälfte die Cursorposition in X-Richtung. Zulässige Werte sind 0 bis 63. Wird ein Wert ≥64 eingegeben, wird dieser korrigiert. Da hier nur die erste (linke) Displayhälfte angesteuert werden soll, ist CS1 = 1 und CS2 = 0. Das Signal RS wird auf 0 gesetzt, da das gesendete Byte als Befehl interpretiert werden muss. Das zu sendende Byte hat den Wert: 0x40 + X-Koordinate.

- `set_x_CS2` [nach 19]
  Die Funktion gibt der zweiten (rechten) Displayhälfte die Cursorposition in X-Richtung. Zulässige Werte sind 0 bis 63. Wird ein Wert ≥64 eingegeben, wird dieser korrigiert. Da hier nur die zweite (rechte) Displayhälfte angesteuert werden soll, ist CS1 = 0 und CS2 = 1. Das Signal RS wird auf 0 gesetzt, da das gesendete Byte als Befehl interpretiert werden muss. Das zu sendende Byte hat den Wert: 0x40 + X-Koordinate.

- `set_y_CS1` [nach 19]
  Die Funktion gibt der ersten (linken) Displayhälfte die Cursorposition in Y-Richtung. Zulässige Werte sind 0 bis 7. Wird ein Wert ≥8 eingegeben, wird dieser korrigiert. Da hier nur die erste (linke) Displayhälfte angesteuert werden soll, ist CS1 = 1 und CS2 = 0. Das Signal RS wird auf 0 gesetzt, da das gesendete Byte als Befehl interpretiert werden muss. Das zu sendende Byte hat den Wert: 0xB8 + Y-Koordinate.

- `set_y_CS2` [nach 19]
  Die Funktion gibt der zweiten (rechten) Displayhälfte die Cursorposition in Y-Richtung. Zulässige Werte sind 0 bis 7. Wird ein Wert ≥8 eingegeben, wird dieser korrigiert. Da hier

nur die zweite (rechte) Displayhälfte angesteuert werden soll, ist CS1 = 0 und CS2 = 1. Das Signal RS wird auf 0 gesetzt, da das gesendete Byte als Befehl interpretiert werden muss. Das zu sendende Byte hat den Wert: 0xB8 + Y-Koordinate.

- `LCD_byte_schreiben_CS1` [nach 19]
  Die Funktion gibt ein Byte auf der ersten (linken) Displayhälfte an der aktuellen Cursorposition aus. Der Wertebereich des Bytes ist nicht beschränkt. Da hier nur die erste (linke) Displayhälfte angesteuert werden soll, ist CS1 = 1 und CS2 = 0. Das Signal RS wird auf 1 gesetzt, da das gesendete Byte als Datenbyte interpretiert werden soll.

- `LCD_byte_schreiben_CS2` [nach 19]
  Die Funktion gibt ein Byte auf der zweiten (rechten) Displayhälfte an der aktuellen Cursorposition aus. Der Wertebereich des Bytes ist nicht beschränkt. Da hier nur die zweite (rechte) Displayhälfte angesteuert werden soll, ist CS1 = 0 und CS2 = 1. Das Signal RS wird auf 1 gesetzt, da das gesendete Byte als Datenbyte interpretiert werden soll.

- `LCD_loeschen`
  Eine displayinterne Funktion zum Löschen des kompletten Speichers wurde vom Hersteller leider nicht implementiert [19]. Daher muss jede einzelne Cursorposition mit einer Null überschrieben werden. Um Zeit zu sparen, werden die Cursor auf beiden Displayhälften parallel angesteuert. Hierdurch wird der langsame Löschvorgang beschleunigt. Nur das Senden des Datenbytes erfolgt einzeln für jede Displayhälfte.
  Es wird zeilenweise gelöscht: In der ersten Zeile werden die einzelnen Cursorpositionen aufsteigend adressiert und überschrieben. Ist das Zeilenende beider Displayhälften erreicht, springt der Cursor auf die erste Cursorposition der zweiten Zeile. Der Löschvorgang aller Zeilen setzt sich entsprechend fort.

- `init_LCD`
  Die Funktion initialisiert die Portpins für den Datenbus und für die Steuersignale am Mikroprozessor als Ausgänge. Der RST-Anschluss des Grafikdisplays wird einmalig vom Mikroprozessor auf high gesetzt. Das Grafikdisplay führt seine interne Initialisierung durch und steht nach einer Wartezeit von 10ms zur Verfügung[19]. Das Display wird mit der Funktion `LCD_off` abgeschaltet. Die Cursor werden auf beiden Displayhälften auf die erste Position in der ersten Zeile gesetzt. Beide Displayhälften werden gleichzeitig angesprochen (CS1 = CS2 = 1). Das Signal RS wird auf 0 gesetzt, da das gesendete Byte als Befehl interpretiert werden muss. Das zu sendende Byte ist 0xC0 [19].
  Das Display wird mit der Funktion `LCD_loeschen` gelöscht und im Anschluss mit der Funktion `LCD_on` wieder eingeschaltet.

- `zeichen_schreiben`
  Die Funktion „übersetzt“ ein übergebenes Zeichen in den Zeichensatz (siehe oben, *zeichensatz.h*). Neben dem auszugebenden Zeichen erwartet die Funktion die Position, an der das Zeichen platziert werden soll. Die Funktion unterteilt das Display in acht Zeilen und 22 Spalten mit einer Breite von sechs Pixeln. Die Angabe der Position wird hierdurch erleichtert.
  Zeichen, die im Zeichensatz nicht vorhanden sind, werden durch „_“ ersetzt. Das Ausgeben des einzelnen Zeichens geschieht byteweise.
  Hinter jedem ausgegebenen Zeichen setzt die Funktion eine ein Pixel breite Leerspalte ein, damit die Schrift auf dem Display besser zu lesen ist.
  Die Funktion greift auf die bereits beschriebenen Funktionen `set_x_CS1`, `set_x_CS2`, `set_y_CS1`, `set_y_CS2`, `LCD_byte_schreiben_CS1` und `LCD_byte_schreiben_CS2` zu.

- `string_schreiben`
  Die Funktion gibt einen übergebenen String zeichenweise über die Funktion `zeichen_schreiben` auf dem Display aus. Die Ausgabe wird entweder durch das Erreichen des Stringendezeichens `\0` oder durch Erreichen der zu übergebenden Ausgabelänge beendet.
  Die Startposition der Ausgabe muss ebenfalls übergeben werden. Die Y-Koordinate wird direkt an die Funktion `zeichen_schreiben` weitergeleitet, während die X-Koordinate bei jedem Zeichen um eine Position verschoben wird.

- `zahl_2_schreiben`
  Die Funktion zerlegt die übergebene `unsigned char`-Zahl in Zehner und Einer. Anschließend werden die beiden einzelnen Komponenten zu einem String zusammengefügt und mit der Funktion `string_schreiben` auf dem Display ausgegeben.
  Die Startposition der Ausgabe muss ebenfalls übergeben werden. Diese wird direkt an die Funktion `string_schreiben` weitergeleitet.
  Die Ausgabe einer führenden Null, die bei einem Zehneranteil = 0 eingesetzt werden kann, muss aktiviert oder deaktiviert werden.
  Die Zahl wird immer rechtsbündig ausgegeben.

- `zahl_3_schreiben`
  Die Funktion zerlegt die übergebene `unsigned char`-Zahl in Hunderter, Zehner und Einer. Anschließend werden die drei einzelnen Komponenten zu einem String zusammengefügt und mit der Funktion `string_schreiben` auf dem Display ausgegeben.
  Die Startposition der Ausgabe muss ebenfalls übergeben werden. Diese wird direkt an die Funktion `string_schreiben` weitergeleitet.
  Die Ausgabe führender Nullen muss aktiviert oder deaktiviert werden.
  Die Zahl wird immer rechtsbündig ausgegeben.

- `zahl_6_schreiben`
  Die Funktion zerlegt die übergebene `long`-Zahl in ihre einzelnen Komponenten. Anschließend werden die sechs Komponenten zu einem String zusammengefügt und mit der Funktion `string_schreiben` auf dem Display ausgegeben. Die Startposition der Ausgabe muss ebenfalls übergeben werden. Diese wird direkt an die Funktion `string_schreiben` weitergeleitet.
  Führende Nullen sind nicht vorgesehen, da diese Funktion nur zum Anzeigen von Dateigrößen verwendet wird und führende Nullen dabei überflüssig sind.
  Die Zahl wird immer rechtsbündig ausgegeben.

7.4.3 Die externen Interrupts (*interrupt.c*)

Für den Anschluss der drei Taster werden I/O-Pins verwendet, die als Sonderfunktion Interrupts auslösen können. Eine Flankenerkennung ist unbedingt erforderlich, damit ein einzelner Tastendruck zuverlässig als solcher erkannt wird und nicht als mehrfacher Tastendruck. Da die externen Interrupts so initialisiert werden können, dass nur bei einer positiven Flanke ein Interrupt ausgelöst wird, kann durch deren Verwendung auf eine aufwendige Flankenerkennung durch die Software verzichtet werden.

Sämtliche Interrupts können durch ein einziges Bit im Systemregister SREG aktiviert oder deaktiviert werden. Das ist sehr praktisch, wenn es darum geht, dass an bestimmten Stellen im Programm nicht auf Interrupts reagiert werden können soll, da nicht alle Interrupts einzeln aktiviert bzw. deaktiviert werden müssen.

Die Datei beinhaltet fünf Funktionen:

- `interrupt_aktivieren`
  Die Funktion schaltet alle aktivierten Interrupts durch das Setzen des I-Flags im Systemregister SREG ein.

- `interrupt_deaktivieren`
  Die Funktion schaltet alle aktivierten Interrupts durch das Löschen des I-Flags im Systemregister SREG aus.

- `init_ext_int`
  Die externen Interrupts müssen initialisiert werden. Die drei verwendeten externen Interrupts (INT5, INT6, INT7) werden aktiviert und sollen bei einer positiven Flanke an den entsprechenden Eingangspins ausgelöst werden.

- `tasteninterrupt_aktivieren`
  Die drei externen Interrupts werden aktiviert. Das I-Flag im Systemregister SREG bleibt hiervon unberührt. Solange dieses nicht gesetzt ist, hat das Aktivieren der externen Interrupts keine Auswirkungen auf Hard- und Software.

- `tasteninterrupt_deaktivieren`
  Die drei externen Interrupts werden deaktiviert. Das I-Flag im Systemregister SREG bleibt hiervon unberührt. Solange dieses nicht gesetzt ist, hat das Deaktivieren der externen Interrupts keine Auswirkungen auf Hard- und Software.

In jeder der drei ISR werden die Interrupt-Flags der externen Interrupts gelöscht. Aus Sicherheitsgründen wird bei jedem Tastendruck das Shuttersignal auf 0 gesetzt (Laser aus). Diese Zeile ist in jeder der drei ISR enthalten, damit der Laser beim Abbruch der Laserausgabe durch das Drücken eines beliebigen Tasters sofort abgeschaltet wird und nicht erst nach dem Ablauf der ISR und dem Beenden der mit der Ausgabe beschäftigten Funktionen.
Die drei Interrupt Service Routinen sind ebenfalls in der Quellcodedatei *interrupt.c* gespeichert:

- `ISR (INT5_vect)`
  Der globalen Variablen `taste` wird der Wert der Konstanten `RAUF` zugewiesen.

- `ISR (INT6_vect)`
  Der globalen Variablen `taste` wird der Wert der Konstanten `ENTER` zugewiesen.

- `ISR (INT7_vect)`
  Der globalen Variablen `taste` wird der Wert der Konstanten `RUNTER` zugewiesen.

7.4.4 Der interne EEPROM (*eeprom.c*)

Der interne EEPROM im Mikroprozessor kann die vier benötigten Bytes ohne Platzprobleme speichern. Ein externer EEPROM würde nur größeren Platinenplatzbedarf, gesteigerten Programmieraufwand für die Ansteuerfunktionen sowie zusätzliche Kosten mit sich bringen.
Der interne EEPROM benötigt keine besondere Initialisierung. Daher ist eine solche Funktion auch nicht programmiert.
Der EEPROM wird nur für das Speichern der vier einstellbaren Parameter des Gerätes benötigt, damit Änderungen auch nach einer Unterbrechung der Spannungsversorgung erhalten bleiben und nicht alle Einstellungen neu vorgenommen werden müssen.

Die Datei beinhaltet die vier Funktionen:

- `eeprom_lesen`
  Der EEPROM hat eine Größe von >256 Bytes. Daher reicht der Zahlenraum von acht Bit zur Adressierung nicht aus und es wird eine 16 Bit breite Variable verwendet. Die Adresse wird also in die oberen und die unteren acht Bits zerlegt. Anschließend wird geprüft, ob zur Zeit kein Schreibzugriff auf den EEPROM stattfindet. Ist diese Bedingung erfüllt, so wird die Adresse angelegt, der gespeicherte Wert ausgelesen und an das aufrufende Programm zurückgegeben.

- `eeprom_schreiben`
  Mit dieser Funktion kann ein Byte an einer frei adressierbaren Stelle im EEPROM gespeichert werden.
  Der Schreibzugriff auf den EEPROM unterliegt einer strengen Auflage: Nachdem die Zieladresse und der abzuspeichernde Wert in die entsprechenden Register geschrieben wurden, muss der generelle Schreibzugriff auf den EEPROM durch das Setzen des Bits EEMWE (engl.: EEPROM Master Write Enable) freigegeben werden. Innerhalb von vier weiteren Taktzyklen muss das Schreiben durch das Setzen des Bits EEWE (engl.: EEPROM Write Enable) im Register EECR (engl.: EEPROM Control Register) gestartet werden. Andernfalls löscht sich das Bit EEMWE von selbst, der Schreibzugriff ist nicht mehr möglich. Während des kompletten Schreibzugriffes müssen folglich auch sämtliche Interrupts deaktiviert werden. Da dies ein sehr zeitkritischer Zugriff ist, wird die bereits vorhandene Funktion `eeprom_write_byte` aus der fertig mit dem Compiler gelieferten Bibliothek *eeprom.h* verwendet. Diese wird in die Funktion `eeprom_schreiben` eingebaut, damit sie zu den Namen der restlichen Funktionen passt.

- `parameter_laden`
  Die vier gespeicherten Parameter werden einzeln aus dem EEPROM geladen und direkt auf Gültigkeit überprüft. Liegt der geladene Wert außerhalb des zulässigen Wertebereiches, wird der jeweilige Standardwert an die fehlerhafte Stelle des EEPROMs geschrieben und derselbe Standardwert für den Parameter verwendet.

- `parameter_speichern`
  Die vier Parameter werden einzeln in den EEPROM geschrieben. Hierzu wird die bereits oben beschriebene Funktion `eeprom_schreiben` verwendet.

7.4.5 Die Wartefunktionen (*warten.c*)

Alle programmierten Wartefunktionen verwenden denselben Timer, d.h. sie können nicht parallel ausgeführt werden. Das ist aber nicht störend, da hier innerhalb der ISR ohnehin keine Wartefunktionen verwendet werden und ein anderer Parallelbetrieb nicht möglich ist. Zur Anwendung kommt der acht Bit breite Timer 0.

Die Datei beinhaltet die fünf Funktionen:

- `init_timer`
  Der Timer muss initialisiert werden. Er wird angehalten, alle Timer-Interrupts werden abgeschaltet, das Output Compare Register sowie der aktuelle Zählerstand werden gelöscht.

- `wait`
  Die Funktion erwartet als Übergabewert eine ganzzahlige Wartezeit ≤15 in Millisekunden. Wird ein größerer Wert übergeben, wird dieser korrigiert, da ein größerer Zeitfaktor nicht

in den acht Bit breiten Zahlenraum passt. Das Output Compare Register wird mit der 16-fachen Wartezeit beschrieben, um einer Auflösung von einer Millisekunde gerecht zu werden.
Anschließend wird der Zähler mit einem Taktteiler von 1024 gestartet. Erreicht der Zählerstand den Wert des Output Compare Registers, so wird der Zähler gestoppt. Die Wartezeit ist abgelaufen, die Funktion wird nach dem Löschen der Timer-Statusbits beendet.
Die sich ergebende Wartezeit wird wie folgt für den Wert 1 berechnet:

$$t = \frac{1024 * 16 * Wert}{f_{CPU}} = \frac{1024 * 16 * 1}{16MHz} = 1{,}024ms \sim 1ms$$

Gleichung 23: Kontrolle des Timers für die Wartefunktion `wait` (*warten.c*)

Es können andere Werte eingesetzt werden (1 bis 15). Die Zeiten werden nicht exakt erreicht, was sich jedoch nicht negativ bemerkbar macht.

- `wait_halbe_usec`
  Für das Ansteuern des Grafikdisplays wird eine kürzere Wartezeit benötigt (siehe 7.4.2). (Es könnte zwar eine ganze Millisekunde gewartet werden, jedoch würde die Displaydarstellung dadurch deutlich gebremst.)
  Das Output Compare Register erhält den festen Wert 8. Der Zähler wird ohne Taktteiler gestartet und arbeitet direkt mit der Prozessorfrequenz f_{CPU}.
  Die sich ergebende Wartezeit wird wie folgt berechnet:

$$t = \frac{8}{f_{CPU}} = \frac{8}{16MHz} = 500ns = 0{,}5\mu s$$

Gleichung 24: Kontrolle des Timers für eine halbe Mikrosekunde (*warten.c*)

Die Zeit wird durch das Starten und Anhalten des Timers um einige Prozessortakte verlängert, was sich jedoch nicht negativ bemerkbar macht.

- `wait_viertel_usec`
  Das Grafikdisplay benötigt eine zweite spezielle Wartezeit von ca. 250ns (siehe 7.4.2). (Es könnte zwar auch hier eine halbe oder eine ganze Millisekunde gewartet werden, jedoch würde die Displaydarstellung dadurch nochmals deutlich gebremst.)
  Das Output Compare Register erhält den festen Wert 4. Der Zähler wird ohne Taktteiler gestartet und arbeitet direkt mit der Prozessorfrequenz f_{CPU}.
  Die sich ergebende Wartezeit wird wie folgt berechnet:

$$t = \frac{4}{f_{CPU}} = \frac{4}{16MHz} = 250ns = 0{,}25\mu s$$

Gleichung 25: Kontrolle des Timers für eine viertel Mikrosekunde (*warten.c*)

Die Zeit wird durch das Starten und Anhalten des Timers um einige Prozessortakte verlängert, was sich jedoch nicht negativ bemerkbar macht.

7.4.6 Die SD-Karte (*sd.c* und *sd.h*)

Die Datei *sd.h* enthält definierte Konstanten für die fünf benötigten Commands und die Länge eines Commands (siehe 3.3.2). Die Größe (in Byte) eines Sektors und die des CSD-Registers sind ebenfalls als Konstanten definiert.
Drei spezielle Datentypen werden definiert:

- Der Aufbau der Struktur `acht_punkt_drei` entspricht dem Aufbau eines Verzeichniseintrages des Dateisystems FAT. Lange Dateinamen werden nicht unterstützt, da diese auf dem verwendeten Grafikdisplay nicht dargestellt werden können.
  Da jeder Verzeichniseintrag 32 Byte lang ist, belegt eine Variable dieses Strukturtyps 32 Bytes im RAM.
- Die Union `eintrag_im_verzeichnis` ermöglicht byteweisen Zugriff auf einen Verzeichniseintrag. Hierzu werden eine Variable vom Datentyp `acht_punkt_drei` und ein `unsigned char`-Feld mit 32 Feldelementen in die Union eingetragen.
- Ein Sektor, in dem ein Verzeichnis gespeichert ist, enthält 16 Verzeichniseinträge. Damit zum Lesen eines Sektors (Blockes) von der SD-Karte keine separate Funktion geschrieben werden muss, werden in die Union `verzeichnissektor` ein Feld mit 16 Feldelementen des Datentyps `eintrag_im_verzeichnis` und ein `unsigned char`-Feld mit 512 Feldelementen eingetragen. Somit kann die Funktion `sd_block_lesen` (siehe unten) für das Lesen eines Verzeichnissektors verwendet werden.

Die Quellcodedatei *sd.c* enthält folgende 14 Funktionen:

- `init_spi`
  Die beiden Pins für die beiden Schalter Card Detect und Write Protection, die in dem Sockel für die SD-Karte integriert sind, werden als Eingänge initialisiert. Der Pin für das Signal CS wird als Ausgang initialisiert und auf high gesetzt (SD-Karte nicht aktiv).
  Der SPI des Mikroprozessors wird aktiviert und der Master Mode eingeschaltet. Das MSB wird zuerst übertragen. Bei fallender Taktflanke des SCK-Signals wird ein neues Datenbit angelegt, bei steigender Taktflanke wird das Datenbit gelesen (siehe 3.3).
  Die maximal mögliche Geschwindigkeit wird eingestellt, der SPI arbeitet mit halber Prozessorfrequenz (8MHz).

- `karte_aktivieren`
  Die Funktion setzt das CS-Signal auf low und aktiviert somit die SD-Karte.

- `karte_deaktivieren`
  Die Funktion setzt das CS-Signal auf high und deaktiviert somit die SD-Karte.

- `sd_karte_erkennen`
  Die Funktion fragt den Zustand des Eingangspins ab, an dem der Schalter Card Detect angeschlossen ist. Wenn eine SD-Karte eingelegt ist, ist der Pin low und die Funktion wird mit dem konstanten Rückgabewert `JA` beendet. Wenn keine SD-Karte eingelegt ist, ist der Pin high und die Funktion wird mit dem konstanten Rückgabewert `NEIN` beendet.

- `sd_karte_byte_senden`
  Die Funktion sendet ein Byte über den SPI an die SD-Karte und wartet, bis das Flag im Statusregister des SPI, welches den abgeschlossenen Sendevorgang anzeigt, gesetzt wird. Die Funktion gibt beim Beenden den Inhalt des Datenregisters des SPI zurück.

- `sd_karte_byte_empfangen`
  Die Funktion sendet ein leeres Byte (0xFF) an die SD-Karte und gibt das parallel dazu empfangene Byte zurück.

- `sd_karte_reset` [nach 4]
  Nach dem Einlegen der SD-Karte muss diese vom Mikroprozessor ein Reset erhalten. Hierzu sind mindestens 74 Takte des SCK-Signals bei deaktivierter SD-Karte (CS ist high) erforderlich. Zum Ausgeben von 80 Takten wird zehnmal die Funktion `sd_karte_byte_empfangen` aufgerufen und somit das Byte 0xFF an die SD-Karte gesendet.

- `sd_karte_command_senden_r1`
  Die Funktion sendet ein übergebenes, aus sechs Bytes bestehendes Command an die SD-Karte. Das Command wird byteweise mit sechsfachem Aufruf der Funktion `sd_karte_byte_senden` an die SD-Karte gesendet. Nach dem letzten gesendeten Byte wird solange die Funktion `sd_karte_byte_empfangen` aufgerufen und deren empfangenes Byte ausgewertet, bis dieses nicht mehr 0xFF ist.
  Das zuletzt empfangene Byte wird zurückgegeben.

- `sd_karte_GO_IDLE_STATE`
  Die Funktion sendet das Command CMD0 mit Hilfe der Funktion `sd_karte_command_senden_r1` an die SD-Karte. Da beim ersten Zugriff nach einem Reset die CRC-Prüfung aktiv ist, muss die gültige Checksumme mit übertragen werden. Diese ist für das Command CMD0 (mit vier leeren Datenbytes) 0x95 [4].
  Nachdem das CMD0 einmal gesendet wurde, wird geprüft, ob in dem Antwortbyte der SD-Karte das Idle-Bit gesetzt ist. Ist dieses nicht der Fall, wird das CMD0 ohne Checksumme so lange wiederholt gesendet, bis das Idle-Bit in dem Antwortbyte der SD-Karte gesetzt ist (0x80). Die SD-Karte befindet sich von diesem Zeitpunkt an im Idle-Zustand.

- `sd_csd_einlesen`
  Die Funktion liest das komplette CSD-Register von der SD-Karte. Eingeleitet wird der Vorgang durch das Senden des Commands CMD9 an die SD-Karte über die Funktion `sd_karte_command_senden_r1`.
  Das Startbyte muss abgewartet werden. Dazu werden solange Bytes von der SD-Karte empfangen und ausgewertet, bis der Wert 0xFE erkannt wird. Die nächsten 16 Bytes enthalten das CSD-Register. Anschließend werden noch zwei weitere Bytes empfangen. Diese enthalten die von der SD-Karte generierte und mit übertragene Checksumme. Diese beiden Bytes werden verworfen. Sie müssen aber von der SD-Karte abgerufen werden, um den nächsten Kartenzugriff nicht zu blockieren.

- `sd_karten_groesse`
  Die Funktion berechnet die Anzahl der auf der SD-Karte vorhandenen Sektoren (Blöcke) nach der Gleichung 3 in Kapitel 3.4. Die benötigten Werte werden dem CSD-Register, das vorher mit der Funktion `sd_csd_einlesen` ausgelesen werden muss, entnommen (siehe Tabelle 6 in Kapitel 3.4). Die berechnete Sektoranzahl (Blockanzahl) wird zurückgegeben.
  Außerdem kontrolliert die Funktion die Größe der Blöcke. Ist diese nicht 512 Byte groß, wird die Funktion beendet und der Wert 0 als Fehlermeldung zurückgegeben.

- `sd_dateisystem`
  Die Funktion prüft, ob das CSD-Register auf ein festplattenähnliches System mit Partitionstabelle hinweist (siehe 3.4). Nur in diesem Fall wird der konstante Wert `JA` zurückgegeben. Weist das CSD-Register auf ein anderes oder gar kein Dateisystem hin, wird der konstante Wert `NEIN` zurückgegeben.

- sd_block_lesen
  Die Funktion liest einen Block von der SD-Karte. Eingeleitet wird der Vorgang durch das Senden des Commands CMD17 über die Funktion sd_karte_command_senden_r1 an die SD-Karte. Das Command benötigt die Adresse des ersten Bytes, das von der SD-Karte gelesen werden soll. Da die Funktion eine Blocknummer als Übergabeparameter erhält, muss diese mit 512 multipliziert oder deren Bits um neun Positionen nach links verschoben werden. Die Funktion sd_karte_command_senden_r1 wird in einer Schleife so oft aufgerufen, bis deren Rückgabewert 0 ist.
  In einer Schleife werden solange Bytes von der SD-Karte empfangen und ausgewertet, bis das Startbyte 0xFE erkannt wird. Der Block wird byteweise mit der Funktion sd_karte_byte_empfangen geladen und in dem 512 Byte großen Feld gespeichert, das der Funktion übergeben werden muss.
  Anschließend werden noch zwei weitere Bytes empfangen. Diese enthalten die von der SD-Karte generierte und mit übertragene Checksumme. Diese beiden Bytes werden verworfen. Sie müssen aber von der SD-Karte abgerufen werden, um den nächsten Kartenzugriff nicht zu blockieren.

- sd_karte_testen_und_root_start_ermitteln
  Die Funktion übernimmt den Start der Kommunikation mit der SD-Karte. Sie kontrolliert, ob eine SD-Karte eingelegt wurde bzw. wartet bei Bedarf auf das Einlegen.
  Die SD-Karte wird gestartet:
  1. Ein Reset der SD-Karte erfolgt durch die Funktion sd_karte_reset.
  2. Die SD-Karte wird mit der Funktion sd_karte_GO_IDLE_STATE in den Idle-Zustand gebracht.
  3. Der Initialisierungsprozess der SD-Karte wird durch bei Bedarf wiederholtes Senden des CMD1 über die Funktion sd_karte_command_senden_r1 an die SD-Karte gestartet.
  4. Es erfolgt das Einlesen des CSD-Registers durch die Funktion sd_csd_einlesen.
  5. Die Größe der SD-Karte (Anzahl der enthaltenen Blöcke) wird mit der Funktion sd_karten_groesse berechnet.
  6. Das Vorhandensein eines FAT-Dateisystems wird mit der Funktion sd_dateisystem kontrolliert.

  Ist kein FAT-Dateisystem vorhanden, wird die SD-Karte abgelehnt und muss durch eine andere ersetzt werden.
  Ist ein FAT-Dateisystem vorhanden, wird der Master Boot Record, beginnend ab der Kartenadresse 0 mit der Funktion sd_block_lesen eingelesen und im 512 Byte großen lesepuffer gespeichert. Das 451. Byte enthält die Information, um welche FAT-Version bzw. um welche FAT-Variante es sich handelt (siehe 4.1). Die globale Strukturvariable v_id erhält die numerische Konstante FAT16 oder FAT32. Ist keine gültige FAT-Version bzw. FAT-Variante enthalten, wird die SD-Karte abgelehnt und muss durch eine andere ersetzt werden.
  Die Bytes 454 bis 457 enthalten die 32 Bit lange Adresse der Volume ID der ersten Partition. Diese Adresse wird mit der Funktion sd_block_lesen eingelesen und im 512 Byte großen lesepuffer gespeichert. Die benötigten Werte werden der Volume ID entnommen und in die globale Strukturvariable v_id kopiert (siehe 4.2).

  Handelt es sich um eine FAT16-Partition, müssen drei Parameter für das Arbeiten mit dem Dateisystem berechnet und in der globalen Strukturvariablen v_id gespeichert werden:
  - Der Startsektor, in dem die erste Dateizuordnungstabelle FAT beginnt, wird nach Gleichung 4 in Kapitel 4.2.1 berechnet.

- Der Startsektor, in dem das Root-Verzeichnis beginnt, wird nach Gleichung 5 in Kapitel 4.2.1 berechnet.
- Der Startsektor, in dem der Datenbereich der Partition beginnt, wird nach Gleichung 6 in Kapitel 4.2.1 berechnet.

Handelt es sich um eine FAT32-Partition, müssen zwei Parameter für das Arbeiten mit dem Dateisystem berechnet und in der globalen Strukturvariablen `v_id` gespeichert werden:
- Der Startsektor, in dem die erste Dateizuordnungstabelle FAT beginnt, wird nach Gleichung 7 in Kapitel 4.2.2 berechnet.
- Der Startsektor, in dem der Datenbereich der Partition beginnt, wird nach Gleichung 8 in Kapitel 4.2.2 berechnet.

Der Startsektor, in dem das Root-Verzeichnis beginnt, muss nicht berechnet werden. Dieser ist als Parameter in der Volume ID der Partition enthalten und wird direkt in die globale Strukturvariable `v_id` kopiert.

7.4.7 Das Dateisystem FAT (*fat.c*)

Die Funktionen in der Quellcodedatei *fat.c* greifen oft auf die Funktion `sd_block_lesen` aus der Quellcodedatei *sd.c* zu (siehe 7.4.6). Ein Block entspricht dabei genau einem Sektor, beide sind 512 Byte groß.

Die Quellcodedatei enthält folgende neun Funktionen:
- `verzeichnisinhalt_loeschen`
  Die Funktion löscht den kompletten Inhalt der globalen Variablen `verz_inh[]`, indem jedes einzelne Strukturelement der einzelnen Feldelemente mit einer 0 überschrieben wird.

- `dateifilter`
  Die Funktion vergleicht zwei drei Zeichen lange Strings miteinander. Sind beide Strings identisch, wird die Konstante `JA` zurückgegeben, ansonsten die Konstante `NEIN`. Die Funktion wird zum Auswerten der Dateiendung benötigt und von der Funktion `verzeichnis_puffern` (siehe unten) aufgerufen.

- `naechster_cluster`
  Die Funktion ermittelt den nächsten Cluster in der Clusterkette (siehe 4.4). Die Clusternummer ist bei FAT16 zwei Byte und bei FAT32 vier Byte lang. Daher werden FAT16 und FAT32 gesondert behandelt:
  - FAT16:
    Die Sektornummer und die Bytenummer des nächsten Clusters in der Dateizuordnungstabelle FAT werden berechnet. Ein Sektor der Dateizuordnungstabelle enthält 256 Clustereinträge, die jeweils zwei Byte groß sind.
    Für die Berechnung der `sektornummer` und der `bytenummer` werden die beiden folgenden Formeln implementiert:

```
sektornummer = clusternummer / 256
```

Gleichung 26: Berechnung der `sektornummer` in der FAT (FAT16)

```
bytenummer = (clusternummer % 256) * 2
```

Gleichung 27: Berechnung der `bytenummer` innerhalb des Sektors (FAT16)

Die Sektornummer wird zu der Adresse, bei der die Dateizuordnungstabelle beginnt, hinzuaddiert. Anschließend wird der so adressierte Sektor von der SD-Karte gelesen und gepuffert.
In der ermittelten Byteposition ist das untere Byte der nächsten Clusternummer gespeichert. Das folgende Byte enthält das obere Byte dieser Clusternummer. Zusammengesetzt ergeben diese die fortsetzende Clusternummer. Ist dieser Wert 0x0000 oder ≥0xFFF8, sind das Verzeichnis bzw. die Datei zu Ende (siehe 4.4) und es wird eine Null zurückgegeben. Andernfalls wird die Nummer des soeben ermittelten Clusters zurückgegeben.

- FAT32:
  Die Sektornummer und die Bytenummer des nächsten Clusters in der Dateizuordnungstabelle FAT werden berechnet. Ein Sektor der Dateizuordnungstabelle enthält 128 Clustereinträge, die jeweils vier Byte groß sind.
  Für die Berechnung der `sektornummer` und der `bytenummer` werden die beiden folgenden Formeln implementiert:

```
sektornummer = clusternummer / 128
```

Gleichung 28: Berechnung der `sektornummer` in der FAT (FAT32)

```
bytenummer = (clusternummer % 128) * 4
```

Gleichung 29: Berechnung der `bytenummer` innerhalb des Sektors (FAT32)

  Die Sektornummer wird zu der Adresse, bei der die Dateizuordnungstabelle beginnt, hinzuaddiert. Anschließend wird der so adressierte Sektor von der SD-Karte gelesen und gepuffert.
  In der ermittelten Byteposition ist das unterste Byte der nächsten Clusternummer gespeichert. Die folgenden drei Bytes enthalten aufsteigend die oberen Bytes dieser Clusternummer. Zusammengesetzt ergeben diese die fortsetzende Clusternummer. Ist dieser Wert 0x00000000 oder ≥0x0FFFFFFF, sind das Verzeichnis bzw. die Datei zu Ende (siehe 4.4) und es wird eine Null zurückgegeben. Andernfalls wird die Nummer des soeben ermittelten Clusters zurückgegeben.

- `eintrag_gueltig`
  Die Funktion überprüft jedes der acht Zeichen des übergebenen Strings, ob es sich dabei um eines der folgenden, erlaubten Zeichen handelt:
  - 0 – 9
  - A – Z
  - Leerzeichen
  - _
  - ~ (Nur dann, wenn dieses an der vorletzten Stelle im String steht.)

  Alle anderen Zeichen sind nicht zugelassen. Sind alle acht Zeichen des Strings zulässig, wird die Funktion mit dem konstanten Rückgabewert `JA` beendet, andernfalls mit dem konstanten Rückgabewert `NEIN`.

- `verzeichnis_puffern`
  Die Funktion liest ein Verzeichnis sektorweise von der SD-Karte aus und puffert die darin enthaltenen Verzeichniseinträge. Dabei werden nur gültige, nicht gelöschte Dateien und Ordner gespeichert. Alle anderen Dateien und gelöschten Ordner werden ignoriert.

Die Dateiendung wird der Konstanten `DATEIFILTER` in der *konst.h* zugewiesen (siehe 7.3.1).
Die Funktion untersucht jeden Verzeichniseintrag für sich. Alle folgenden Bedingungen müssen erfüllt sein, um einen Eintrag zu puffern:

- Attribut ≠ 2 (versteckt), ≠ 8 (Laufwerksname) und ≠ 15 (Systemdatei)
- Startcluster ≥2
- Funktion `eintrag_gueltig` liefert den Rückgabewert `JA` (siehe oben)
- Erster Buchstabe ≠ 229 (dezimal)
- Datei: Funktion `dateifilter` liefert den Rückgabewert `JA` (siehe oben)
  Ordner: Attribut = 16

Ist eine der Bedingungen nicht erfüllt, wird dieser Eintrag ignoriert und der nächste Eintrag untersucht.

Der Dateiname bzw. Ordnername und der zugehörige Startcluster werden in das Feld `verz_inh[]` gespeichert. Im Anschluss wird untersucht, ob das Attribut ungleich 16 ist. Ist das der Fall, handelt es sich um eine Datei und die Dateiendung sowie die Dateigröße werden ebenfalls in das Feld `verz_inh[]` gespeichert.

Es können bis zu 100 Einträge gepuffert werden. Ist das Ende des Verzeichnisses erreicht oder sind 100 gültige Einträge gespeichert, wird die Funktion verlassen.

- `verzeichnisinhalt_sortieren`
  Die Funktion sortiert den gepufferten Verzeichnisinhalt so, dass als erstes die Ordner und im Anschluss die Dateien aufgelistet sind. Hierzu wird ein temporäres Zeigerfeld eingesetzt, um nur die Zeiger der Einträge zu kopieren und nicht den kompletten Inhalt.
  Im Anschluss werden Ordner und Dateien separat nach aufsteigendem Alphabet sortiert.

  Der Sortieralgorithmus wird beispielhaft anhand der Dateien beschrieben. (Der Sortieralgorithmus für das Sortieren der Ordner ist identisch aufgebaut und wird nicht erneut erläutert.)
  Ist kein oder nur ein Eintrag vorhanden, wird der Sortiervorgang übersprungen.
  Der erste und der zweite Eintrag werden zeichenweise verglichen, bis sich eine Abweichung ergibt. Die Abweichung kann auch an letzter Stelle liegen. Ist der Zahlenwert an der gefundenen Stelle des ersten Eintrages größer als der des zweiten Eintrages, werden sie miteinander vertauscht. Ist der Zahlenwert an der gefundenen Stelle des ersten Eintrages kleiner als der des zweiten Eintrages, werden sie nicht vertauscht.
  Der Sortiervorgang fährt mit dem zweiten und dritten Eintrag, dem dritten und vierten Eintrag usw. fort. Ist der letzte Eintrag erreicht, wird der Sortieralgorithmus erneut gestartet.
  Jeder Reihenfolgentausch wird mitgezählt. Der Zähler wird beim erneuten Starten des Algorithmus auf 0 gesetzt. Das Sortieren ist beendet, wenn dieser Zähler am Ende eines Sortierdurchlaufes immer noch den Wert 0 hat.

  Sind Ordner und Dateien sortiert, werden die Zeiger von dem temporären Zeigerfeld mit der neuen Reihenfolge in das ursprüngliche Zeigerfeld `verz_inh[]` zurückkopiert und das temporäre Zeigerfeld gelöscht.

- `verzeichnisinhalt_anzeigen`
  Die Funktion gibt den Verzeichnisinhalt abschnittsweise auf dem Grafikdisplay aus. Übergeben werden müssen die Nummer des ersten anzuzeigenden Eintrages, die Anzahl der insgesamt vorhandenen Einträge und die Anzahl der anzuzeigenden Einträge.

Ist das auszugebende Verzeichnis das Root-Verzeichnis und soll vom ersten Eintrag an ausgegeben werden, lautet die erste Zeile `...zum Hauptmenü` (siehe Abbildung 39 in Kapitel 7.4.1). Ist das auszugebende Verzeichnis ein anderes Verzeichnis und soll vom ersten Eintrag an ausgegeben werden, lautet die erste Zeile `...Ordner verlassen`.
In diesen beiden Fällen werden insgesamt maximal fünf Verzeichniseinträge angezeigt, ansonsten sechs. Sind keine oder zu wenig Einträge vorhanden, um alle Zeilen des Grafikdisplays zu füllen, werden die restlichen Zeilen mit Leerzeichen überschrieben.

- `kartenname_auslesen`
  Die Funktion liest den Namen der SD-Karte aus dem Root-Verzeichnis aus, wobei dieses eintragsweise untersucht wird. Wenn das vierte Attributbit eines Verzeichniseintrages gesetzt ist (0x08), handelt es sich bei Dateiname und –erweiterung um den elf Zeichen langen Namen der SD-Karte (Laufwerksname). Für die Ausgabe ist die Funktion `kartendaten_ausgeben` zuständig (siehe 7.4.1).

- `startsektor_berechnen`
  Die Funktion errechnet anhand der Clusternummer den Startsektor, bei dem der Cluster beginnt. Hierzu wird folgende Formel implementiert:

  Sektornummer = ((Clusternummer - 2) * Sektoren pro Cluster) + Root-Adresse

  Gleichung 30: Berechnung des Startsektors eines Clusters

  FAT16 verwendet als Root-Adresse das Strukturelement `start_data_fat16` der globalen Strukturvariablen `v_id`, während FAT32 das Strukturelement `root_dir_adresse` verwendet.
  Die berechnete Sektornummer wird zurückgegeben.
  Die Cluster 0 und 1 sind reserviert. Wird einer dieser beiden Werte an die Funktion übergeben, bricht diese ab und gibt als Fehlermeldung den Wert 0 zurück.

7.4.8 Die DACs (*DAC.c*)

Die Quellcodedatei *DAC.c* enthält acht Funktionen, von denen vier speziell für Messungen zu programmieren sind. Diese werden lediglich für die Testphase benötigt und im normalen Betrieb nie angesprochen bzw. aufgerufen (siehe 9.1 und 9.2).

- `init_DAC`
  Die Ports, die für die Steuersignale und den Datenbus Verwendung finden, werden als Ausgänge eingestellt. Der Datenbus und das Shuttersignal erhalten den Wert 0. Die übrigen Steuerleitungen werden auf 1 gesetzt, da die verwendeten 12-Bit-DACs und die beiden Flipflops bei fallender Flanke den Wert vom Datenbus übernehmen.

- `rot_dreieck`
  Die Funktion steuert den 4-Bit-DAC an, der das analoge Signal für den roten Farbkanal generiert. Sie erzeugt an dessen Ausgang eine dreieckförmige Spannung, die sich über den kompletten Werte- und Spannungsbereich des Kanals erstreckt.

- `gruen_dreieck`
  Die Funktion steuert den 4-Bit-DAC an, der das analoge Signal für den grünen Farbkanal generiert. Sie erzeugt an dessen Ausgang eine dreieckförmige Spannung, die sich über den kompletten Werte- und Spannungsbereich des Kanals erstreckt.

- `y_dreieck`
  Die Funktion steuert den 12-Bit-DAC an, der das analoge Signal für den Kanal der Y-Achse generiert. Außerdem steuert sie das gemeinsame Flipflop der beiden Achsenkanäle an, um die vier LSB des 12-Bit-DACs verändern zu können. Die Funktion erzeugt an dem Ausgang des DAC eine dreieckförmige Spannung, die sich über den kompletten Werte- und Spannungsbereich des Kanals erstreckt.

- `x_dreieck`
  Die Funktion steuert den 12-Bit-DAC an, der das analoge Signal für den Kanal der X-Achse generiert. Außerdem steuert sie das gemeinsame Flipflop der beiden Achsenkanäle an, um die vier LSB des 12-Bit-DACs verändern zu können. Die Funktion erzeugt an dem Ausgang des DAC eine dreieckförmige Spannung, die sich über den kompletten Werte- und Spannungsbereich des Kanals erstreckt.

- `laserpuffer_loeschen`
  Der Speicher, auf den mit Hilfe des globalen Zeigers `*laser_puffer` zugegriffen wird, wird komplett mit Nullen überschrieben. Der Aufruf dieses Löschvorganges erfolgt vor jeder gestarteten Laserausgabe zum Schutz gegen eventuell fehlgeleitete Laserstrahlen.
  Das Löschen geschieht byteweise.

- `ausgabe`
  Die Funktion ist für das Lesen der auszugebenden *ILDA*-Datei von der SD-Karte während der Showausgabe zuständig und arbeitet mit der ISR des Timers zusammen. Um Konflikte und falsche Ausgaben sowie das Überschreiben noch nicht ausgegebener Sektoren des Puffers zu verhindern, kommunizieren beide über ein einfaches Handshake-Verfahren. Neben dem in Blöcke aufgeteilten Ausgabepuffer `*laser_puffer` existiert ein `statusfeld`, das den Speicherstatus jeden Blockes enthält. Die Funktion `ausgabe` speichert einen gelesenen Sektor in einen freien Block des Ausgabepuffers `*laser_puffer` und setzt dessen Eintrag im `statusfeld` auf `STATUS_FERTIG_GELESEN`. Der Block kann solange nicht erneut beschrieben werden, bis die ISR des Timers diesen Block ausgegeben und anschließend den Eintrag im `statusfeld` auf `STATUS_LEER` gesetzt hat (siehe 7.4.9).
  Die Funktion übernimmt das Starten und das Anhalten des Timers vor bzw. nach der Ausgabe (siehe 7.4.9).
  Wird die SD-Karte entfernt oder die Ausgabe durch das Drücken einer Taste abgebrochen, wird der Laser abgeschaltet und die Funktion mit einer entsprechenden Rückmeldung verlassen.
  In der folgenden Abbildung wird die Meldung, die während einer laufenden Ausgabe auf dem Grafikdisplay zu sehen ist, für die gewählte Ausgabe in einer Schleife dargestellt (Dateibeispiel: *dreieck.ild*). Die erste Zeile wird von der aufrufenden Funktion `dateianzeige` auf das Display geschrieben (siehe 7.4.1):

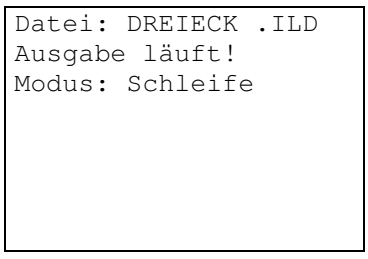


Abbildung 47: LCD-Ausgabe der Funktion `ausgabe` (*DAC.c*)

Da diese Funktion sehr komplex ist, wird auf eine detaillierte Erläuterung verzichtet und stattdessen ein Flussdiagramm zur Veranschaulichung verwendet:

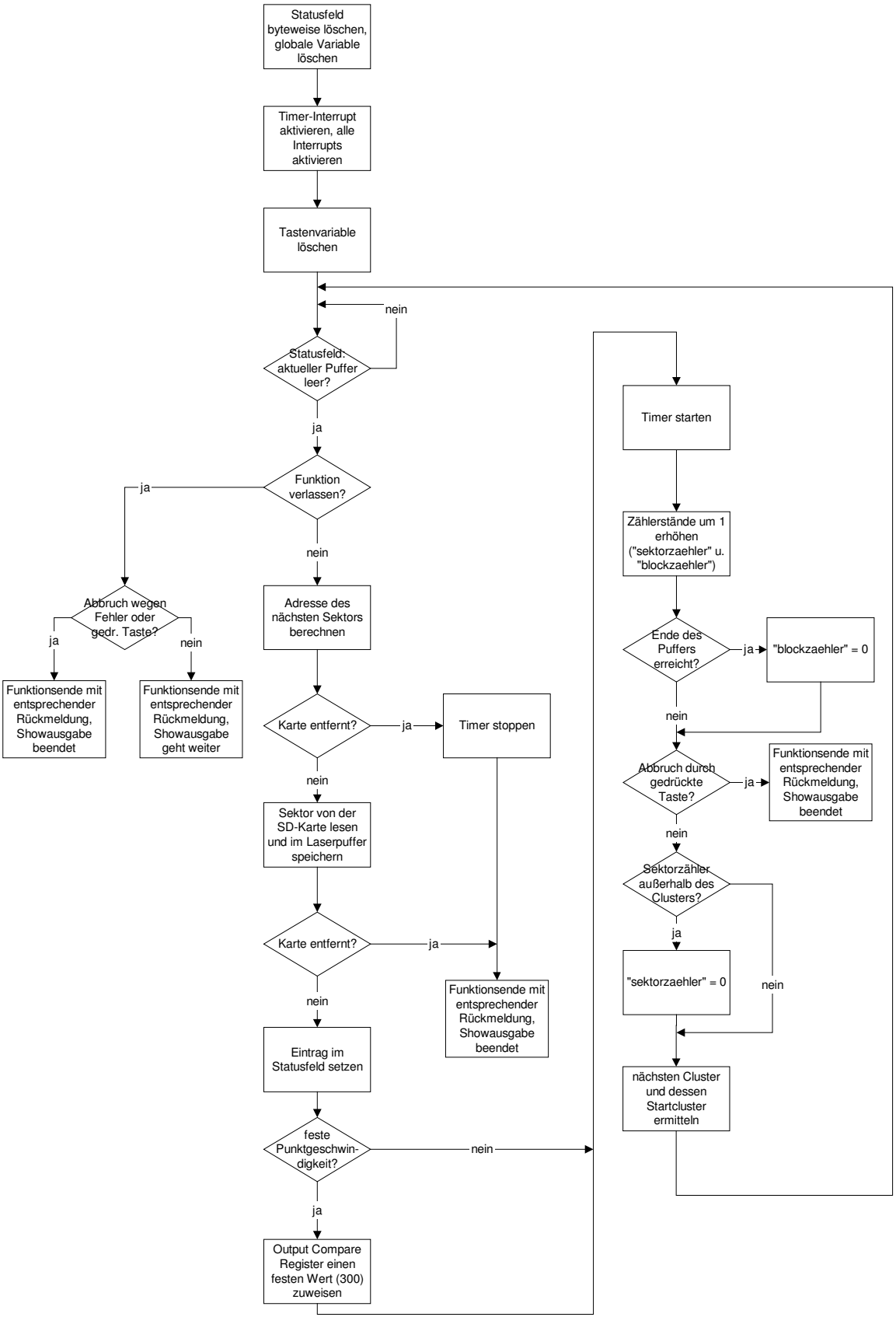


Abbildung 48: Flussdiagramm der Funktion `ausgabe` (*DAC.c*)

- `dateiausgabe`
  Die Funktion wird von der Funktion `dateianzeige` aufgerufen (siehe 7.4.1), wenn die gewählte Datei auf den Laser ausgegeben werden soll. Dabei stehen zwei verschiedene Ausgabemodi zur Auswahl:
  - Bei der einmaligen Ausgabe wird eine entsprechende Mitteilung auf dem Grafikdisplay ausgegeben und die Funktion `ausgabe` (siehe oben) ein einziges Mal aufgerufen. Nach deren Ende wird die Funktion `dateiausgabe` beendet, der Programmablauf kehrt in die Funktion `dateianzeige` zurück. Die Ausgabe kann jederzeit durch das Drücken einer Taste beendet werden. Wird die SD-Karte entfernt, wird die Ausgabe aus Sicherheitsgründen ebenfalls sofort beendet und der Laser abgeschaltet.
  - Bei der wiederholten Ausgabe in einer Schleife wird eine entsprechende Mitteilung auf dem Grafikdisplay ausgegeben und die Funktion `ausgabe` (siehe oben) in einer Schleife kontinuierlich wiederholt aufgerufen. Die Ausgabe kann jederzeit durch das Drücken einer Taste beendet werden. Nach einem solchen Abbruch wird die Funktion `dateiausgabe` beendet, der Programmablauf kehrt in die Funktion `dateianzeige` zurück. Wird die SD-Karte entfernt, wird die Ausgabe ebenfalls sofort beendet und der Laser abgeschaltet.

7.4.9 Der Timer für die Laserausgabe (*timer16.c* und *farbpalette.h*)

In der *farbpalette.h* ist die Standard-Farbpalette des Herstellers *Pangolin Laser Software*, die für dieses Projekt auf zwei Farben mit je vier Bit Auflösung umgestellt wurde, in einem 64 Elemente umfassenden `unsigned char`-Feld enthalten. Dieses Feld wird von der in der Quellcodedatei *timer16.c* enthaltenen ISR des verwendeten 16-Bit-Timers benutzt.

Die Quellcodedatei *timer16.c* enthält vier Funktionen:

- `init_ausgabe_timer`
  Der Timer muss initialisiert werden. Er wird angehalten, das Output Compare Register A sowie der aktuelle Zählerstand werden gelöscht. Ebenfalls werden das Input Capture Register und alle Interrupt-Flags des Timers gelöscht. Der Timer wird immer mit einem Taktteiler betrieben, der die Prozessorfrequenz durch acht teilt (2MHz).
  Das Output Compare Register A erhält einen Standardwert von 300 Takten (dezimal). Mit diesem Wert wird die Haltezeit pro Punkt (mit dem Taktteiler acht) wie folgt berechnet:

$$t_{fest} = \frac{8}{16MHz} * 300Takte = 150\mu s$$

Gleichung 31: Berechnung der Haltezeit pro Punkt bei fester Punktgeschwindigkeit

  Die Standardzeit wird verwendet, wenn die Lasershow mit fester Punktgeschwindigkeit ausgegeben werden soll.

- `timer_interrupt_aktivieren`
  Der Timer Compare Interrupt wird aktiviert. Das I-Flag im Systemregister SREG bleibt hiervon unberührt. Solange dieses nicht gesetzt ist, hat das Aktivieren des Timer Compare Interrupts keine Auswirkungen auf Hard- und Software.

- `timer_interrupt_deaktivieren`
  Der Timer Compare Interrupt wird deaktiviert. Das I-Flag im Systemregister SREG bleibt hiervon unberührt. Solange dieses nicht gesetzt ist, hat das Deaktivieren des Timer

Compare Interrupts keine Auswirkungen auf Hard- und Software.

- `timer_compare_berechnen`
  Die Funktion berechnet den Compare-Wert für den 16-Bit-Timer 1 so, dass 25 Frames pro Sekunde mit zwei bis 700 Punkten pro Frame ausgegeben werden können. Sind mehr als 700 Punkte enthalten, wird anstelle eines berechneten Wertes ein Standardwert von 100 Takten in das Output Compare Register A geschrieben. Kürzere Zeiten sind für das verwendete Lasersystem nicht zu empfehlen und werden nicht weiter getestet.

$$Takte\_pro\_Frame = \frac{Taktteiler}{Frames\_pro\_sekunde * 16MHz} = \frac{8}{25 * 16MHz} = 80000$$

Gleichung 32: Berechnung der Takte pro Frame (*timer16.c*)

Die berechnete Anzahl wird als Zahlenkonstante in die folgende Gleichung, die im Mikroprozessor implementiert wird, eingesetzt. Mit dieser Gleichung wird der Wert für das Output Compare Register A (OCRA) berechnet und im Anschluss von der Funktion zurückgegeben.
Als Beispiel wird der Wert für ein Frame mit 100 Punkten berechnet:

$$OCRA = \frac{80000}{Punkte\_pro\_Frame} = \frac{80000}{100} = 800$$

Gleichung 33: Berechnung des Wertes für das Output Compare Register A (*timer16.c*)

Beispielhaft wird nun der berechnete Wert, der in das Output Compare Register A geladen wird, in die Zeit, die für einen Punkt benötigt wird, umgerechnet. Eingesetzt werden der Taktteiler acht und der beispielhafte Inhalt des OCRA aus Gleichung 33:

$$t_{100\_Punkte} = \frac{8}{16MHz} * 800 = 400\mu s$$

Gleichung 34: Beispielberechnung der Haltezeit pro Punkt (*timer16.c*)

400µs pro Punkt bedeuten bei einem Frame mit 100 Punkten eine Frameausgabezeit von 40ms. Somit können pro Sekunde 25 solcher Frames projiziert werden.

Des Weiteren ist die Interrupt Service Routine, die beim Compare Match des Timers ausgelöst wird, in dieser Quellcodedatei gespeichert:

- `ISR (TIMER1_COMPA_vect)`
  Die Interrupt Service Routine arbeitet direkt mit der Funktion `ausgabe` zusammen, welche das sektorweise Lesen der ausgewählten Datei von der SD-Karte übernimmt (siehe 7.4.8). In dieser ISR werden die Sektoren untersucht und die Lasershow mit fester oder mit der durch die Funktion `timer_compare_berechnen` berechneten Punktgeschwindigkeit ausgegeben.
  Die ISR arbeitet ausschließlich mit globalen Variablen, um die Geschwindigkeit durch die Verwaltung lokaler Variablen nicht zu behindern. Sämtliche Zugriffe auf die Sektoren erfolgen pixelweise. Ein Pixel besteht aus acht Bytes, dadurch ist einfacher Zugriff auf die drei Koordinaten sowie den Status Code möglich (Datentyp `ilda_aufbau`, siehe 7.3.2).
  Da die ISR sehr komplex ist, wird auf eine detaillierte Erläuterung verzichtet und stattdessen ein Flussdiagramm zur Veranschaulichung verwendet:

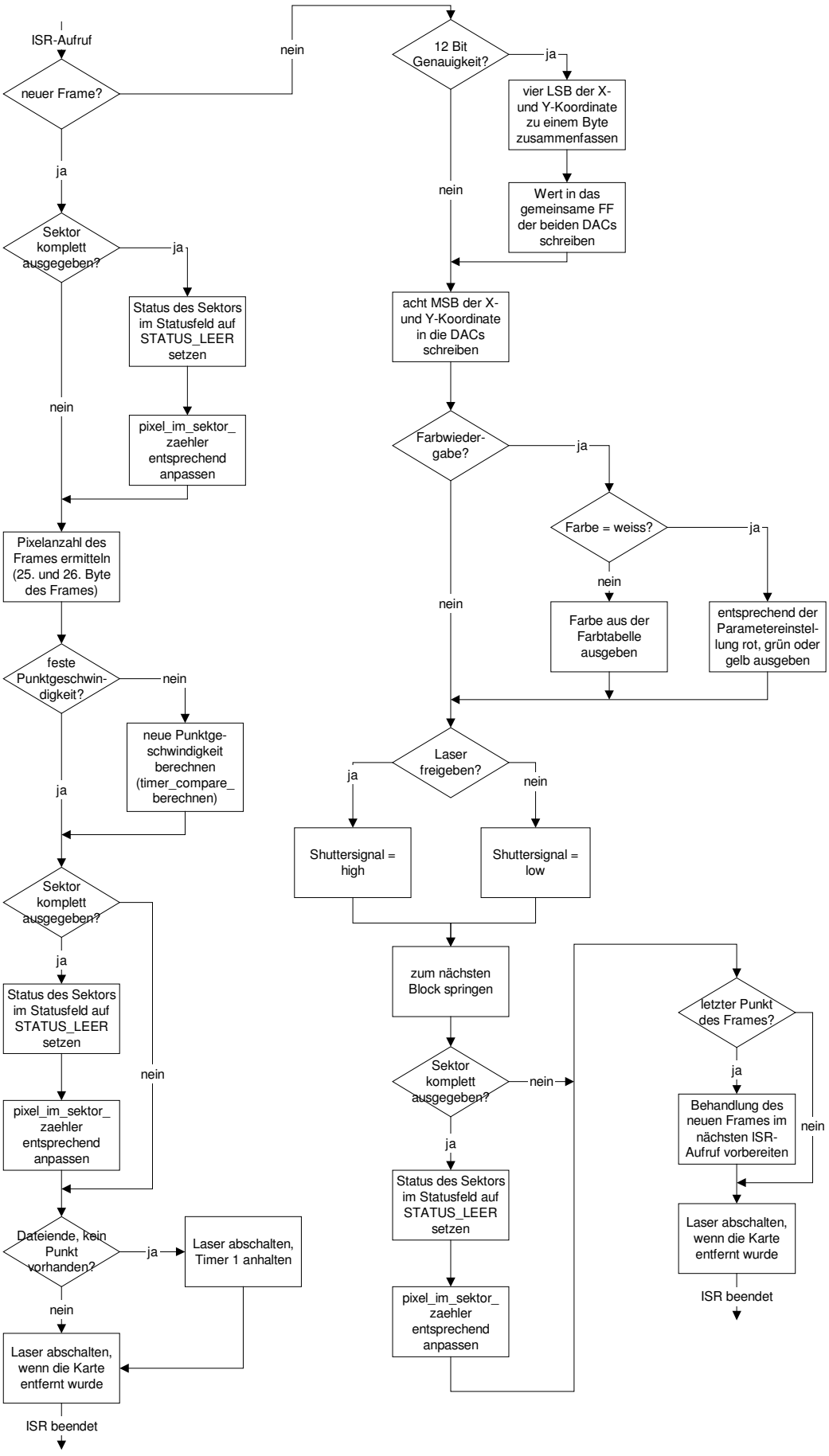


Abbildung 49: Flussdiagramm der Interrupt Service Routine des 16-Bit-Timers (*timer16.c*)

7.5 Das kompilierte Programm

Der selbst geschriebene Quellcode (ohne die Dateien *io.h*, *interrupt.h*, *stdlib.h*, und *eeprom.h*) hat eine Länge von 3661 Programm- und Kommentarzeilen. Er belegt in dem Mikroprozessor folgenden Speicherplatz:

Speicherart	belegter Speicher (in Bytes)	Speicherplatzbelegung	
		ATmega64	*ATmega128*
Programmspeicher (Flash)	14.178	21,6%	10,8%
Int. RAM	1.546	37,7%	37,7%
Int. EEPROM	4	0,2%	0,1%
Ext. SRAM (60KB)	ca. 35.000	ca. 53,3%	ca. 53,3%

Tabelle 14: Speicherplatzbelegung im Mikroprozessor

Die Belegung des externen SRAMs kann nur ungefähr ermittelt werden. Dieser SRAM wird für das Arbeiten mit der SD-Karte benötigt, vor allem zum Puffern gelesener Sektoren (60 Sektoren * 512 Bytes), damit diese für die Laserausgabe schnell zur Verfügung stehen und die Laserprojektion nicht ins Stocken gerät. Ca. 5 KB werden von anderen Variablen belegt. Der restliche SRAM stellt sicher, dass dem Mikroprozessor auch bei längerem Betrieb keine Fehler (durch das Fragmentieren des SRAMs) unterlaufen oder dieser stehen bleibt.

Damit die Gerätefunktion der Showlasersteuerung erfüllt wird, könnte laut Tabelle 14 auch der *ATmega64* eingesetzt werden, da dieser (bis auf Pin 1) pinkompatibel zum verwendeten *ATmega128* ist und daher praktischerweise auch kein neues Platinenlayout voraussetzt (siehe 6.1). Kleinere Prozessoren des Herstellers können jedoch nicht verwendet werden, da diese nicht über die erforderliche Anzahl an I/O-Pins verfügen.

8 Aufbau des Prototypen

Für den Aufbau des Prototypen wird ein unbearbeitetes Aluminiumgehäuse verwendet. Das gewählte Gehäuse besteht aus einem oberen und einem unteren Halbschalenprofil sowie zwei Deckelplatten. In den Innenseiten der beiden Halbschalen befinden sich Rillen, um Platinen mit einer Breite von 100mm und einer Dicke von bis zu 1,5mm ohne Schrauben einfach im Gehäuse montieren zu können. Die Tiefe der beiden Halbschalen beträgt ebenfalls 100mm. Da die entwickelten Platinen nur 80mm tief sind, sind noch 20mm für die zu verlegenden Flachbandkabel und den 25-poligen D-SUB-Stecker frei.
Das Gehäuse muss bearbeitet werden: Die benötigten Öffnungen werden ausgesägt bzw. gebohrt. In den vorderen Gehäusedeckel werden zwei Löcher mit einem Durchmesser von 3mm zum Befestigen der beiden Festspannungsregler gebohrt (siehe Abbildung 34 in Kapitel 6.11.3). Die untere Halbschale wird nicht bearbeitet.

8.1 Gehäuseoberteil mit dem Grafikdisplay und den drei Tastern

In dem Gehäuseoberteil (obere der beiden Halbschalen) finden das Display und die drei Taster Platz. Hierbei handelt es sich um die Elemente, die zum Bedienen des Gerätes erforderlich sind.
Für das Grafikdisplay wird eine rechteckige Öffnung mit den Abmessungen 90mm * 54mm passgenau gesägt. Für die Befestigung mit M2,5-Schrauben werden vier Löcher gebohrt. Das Display wird unter Verwendung einiger Unterlegscheiben und Muttern gerade in das Gehäuse eingesetzt und fixiert.
Die verwendeten Taster erfordern je ein Loch mit einem Durchmesser von 8mm.

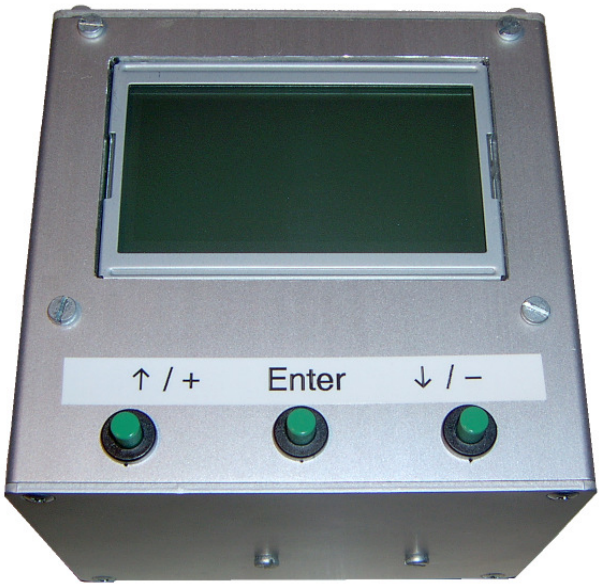


Abbildung 50: fertig gebautes Gerät, Ansicht auf die Oberseite

An das Grafikdisplay wird ein 20-poliges Flachbandkabel angelötet, welchem am anderen Ende ein (ebenfalls 20-poliger) Pfostenstecker aufgecrimpt wird. An die drei Taster werden je zwei benachbarte Adern eines sechspoligen Flachbandkabels angelötet. Dem sechspoligen Flachbandkabel wird ein zehnpoliger Pfostenstecker so aufgecrimpt, dass die Pins 1, 2, 9 und 10 des Pfostensteckers offen bleiben. Diese werden nicht benutzt.
Damit ist die Gehäuseoberseite komplett bearbeitet.

8.2 Hintere Gehäuseseite mit Anschlüssen und SD-Kartenschacht

In der hinteren Deckelplatte finden die Niederspannungsbuchse und die 25-polige D-SUB-Buchse Platz. Des Weiteren wird ein rechteckiger Durchbruch für das Einlegen einer SD-Karte in den auf dem Laser-CPU-Board enthaltenen Sockel für SD-Karten benötigt.
Für die D-SUB-Buchse wird eine rechteckige, passgenaue Öffnung mit den Abmessungen 39mm * 12,5mm gesägt. Für die Befestigung mit M2-Schrauben werden zwei Löcher gebohrt.
Für die Niederspannungsbuchse wird ein Loch mit einem Durchmesser von 11mm gebohrt. Diese Buchse wird auf der Rückseite mit einem passenden Federring und einer passenden Mutter befestigt.
Der rechteckige Durchbruch muss genau vor dem Sockel für die SD-Karte ausgesägt werden. Die Abmessungen dieses Loches sind 25mm * 4mm. Somit passt eine SD-Karte, deren Abmessungen (Breite * Höhe) 24mm * 2mm sind, gut durch die Öffnung.

Abbildung 51: fertig gebautes Gerät, Ansicht auf die Rückseite

Als D-SUB-Buchse wird eine Variante mit Crimpanschluss auf der Innenseite gewählt, damit die Verbindung mit dem Flachbandkabel und dem Pfostenstecker einfach und stabil ist. Bei dem Pfostenstecker auf der anderen Seite des Flachbandkabels bleibt der 26. Pin frei.
Damit ist der hintere Gehäusedeckel komplett bearbeitet.

Der SD-Kartenschacht wird so gebaut, dass die SD-Karte einfach eingelegt und wieder entnommen werden kann.
Wird eine SD-Karte ohne Einrasten in den Schacht eingelegt, steht die SD-Karte ca. 6,5mm aus dem Schacht heraus und kann leicht entnommen werden (Abbildung 52, Mitte).
Wird die SD-Karte durch Drücken eingerastet, steht die SD-Karte ca. 2,5mm aus dem Schacht heraus (Abbildung 52, rechts). Eine Entnahme ohne vorherige Entriegelung durch wiederholtes Drücken ist somit nur schwer möglich.

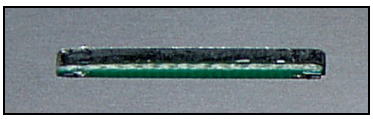
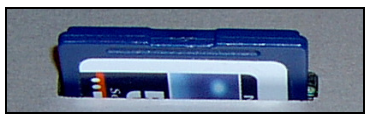
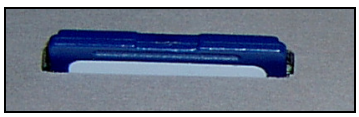
Kartenschacht leer, keine Karte eingelegt	Karte eingelegt, nicht eingerastet (kein Datenzugriff)	Karte eingelegt und eingerastet (Datenzugriff)
		

Abbildung 52: Schacht der SD-Karte

8.3 Einsetzen der Platinen

Die fertigen Komponenten werden zusammengesetzt. Die untere Gehäusehälfte ist mit dem vorderen Gehäusedeckel verschraubt. Die Platine der Spannungsversorgung ist in die untere Gehäusehälfte eingeschoben und durch die beiden Festspannungsregler fest mit dem vorderen Gehäusedeckel verbunden (siehe Abbildung 34 in Kapitel 6.11.3). Das Grafikdisplay, die drei Taster sowie die 25-polige D-SUB-Buchse und die Niederspannungsbuchse sind bereits montiert.
Als nächstes wird das DAC-Board oberhalb der Platine für die Spannungsversorgung so tief wie möglich in die untere Gehäusehälfte geschoben. Die beiden Platinen werden über ein zehnpoliges Flachbandkabel (mit beidseitig aufgecrimpten Pfostensteckern) miteinander verbunden.
Die obere Gehäusehälfte wird aufgesteckt und mit dem vorderen Gehäusedeckel verschraubt. Das Laser-CPU-Board wird über ein 20-poliges Flachbandkabel (mit beidseitig aufgecrimpten Pfostensteckern) mit dem DAC-Board verbunden. Die beiden Pfostenstecker des Grafikdisplays und der Taster werden ebenfalls aufgesteckt. Anschließend wird das Laser-CPU-Board - mit den beiden 20-poligen Pfostensteckern voraus - in die obere Gehäusehälfte gesteckt. Diese Platine wird nicht tief in das Gehäuse geschoben, sondern berührt bei komplett geschlossenem Gehäuse als einzige Platine den hinteren Gehäusedeckel, damit der Sockel der SD-Karte erreichbar ist.
Anschließend wird der hintere Gehäusedeckel aufgesetzt und festgeschraubt.

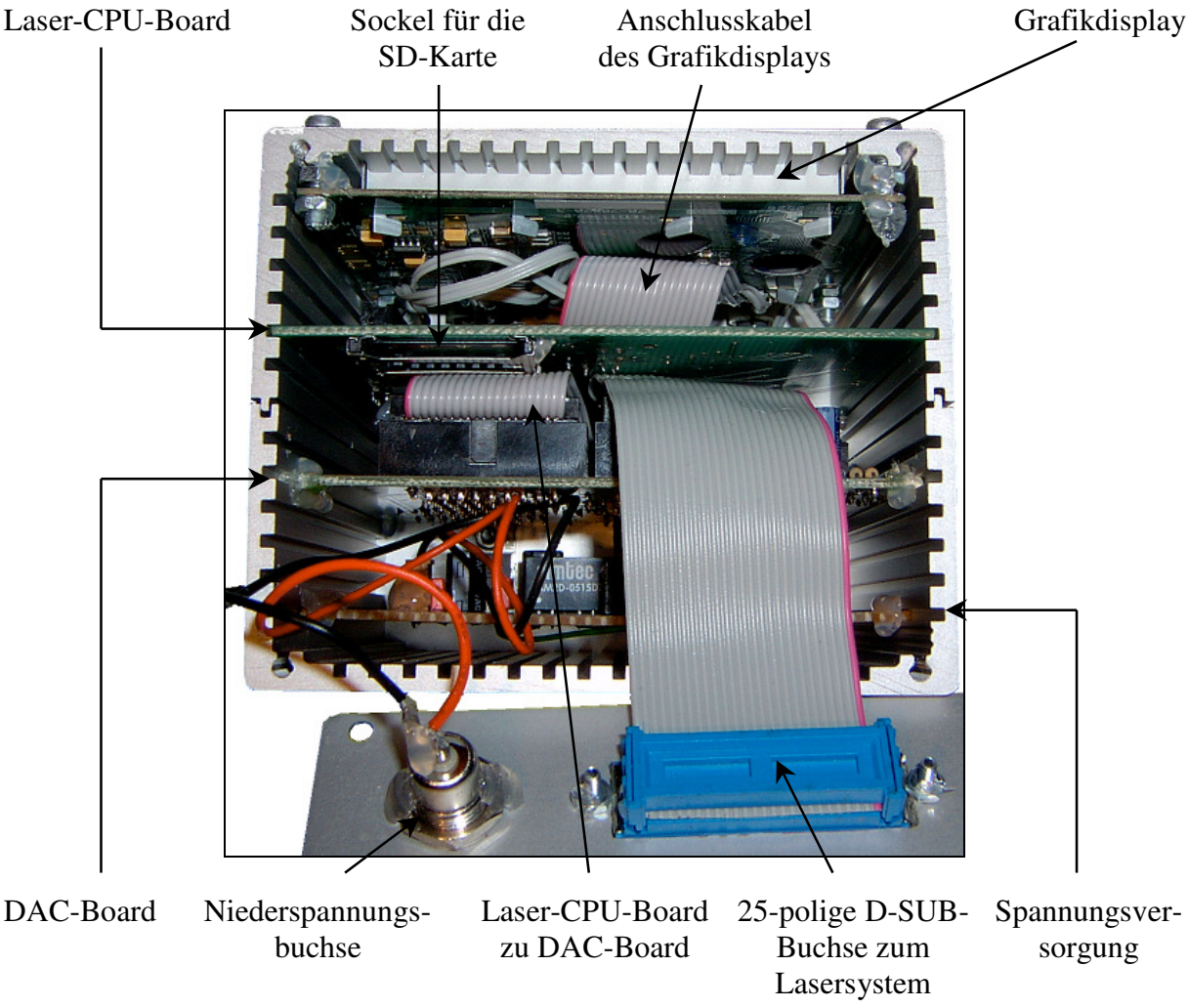


Abbildung 53: seitliche Sicht in das fertig bestückte Gehäuse

9 Messungen am fertigen Prototypen

Die Ausgangssignale der Schnittstelle zum Lasersystem werden am fertig aufgebauten und programmierten Gerät mit einem Oszilloskop gemessen. Die Signalverläufe werden untersucht und mit der vorgegebenen Schnittstellenspezifikation nach *ILDA* verglichen.
Zu diesem Zweck werden spezielle Funktionen geschrieben, die ein kontinuierliches Dreiecksignal an die zu untersuchenden Kanäle über deren gesamten Spannungsbereich ausgeben. Diese Funktionen befinden sich in der Quellcodedatei *DAC.c* und werden in Kapitel 7.4.8 genauer erläutert.
Als Oszilloskop wird das PC-gestützte Oszilloskop *PCS500* von *Velleman®* verwendet.

9.1 X- und Y-Kanal

Da für die beiden Achsenkanäle hochauflösende 12-Bit-DACs verwendet werden, sind auf den Oszilloskopbildern der beiden Achsenkanäle keine „Treppenstufen“ erkennbar.

Das folgende Oszilloskopbild zeigt die beiden Signale X+ und X-:

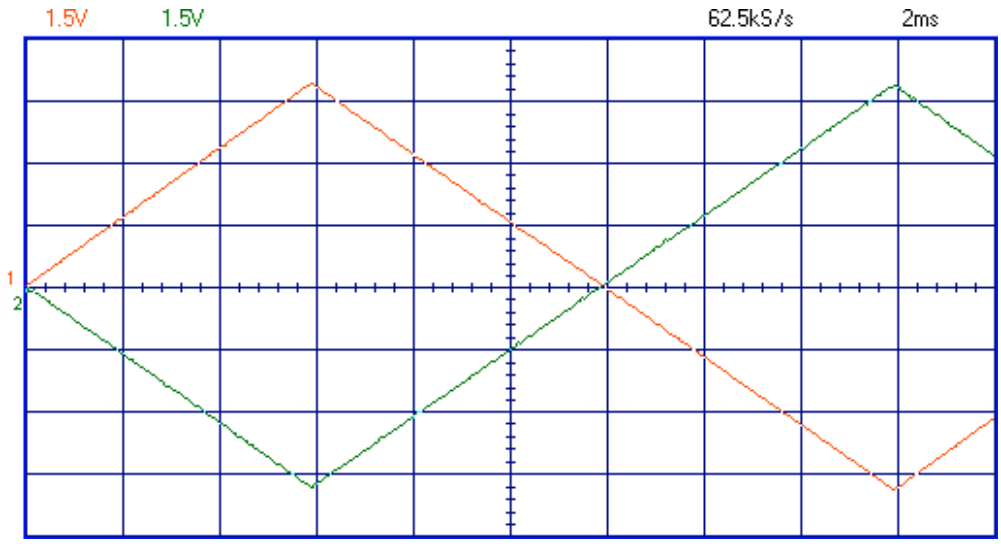


Abbildung 54: Dreieckspannungen am X-Kanal

Die rote Kurve (Kanal 1 am Oszilloskop) stellt das Signal X+ dar, die grüne Kurve (Kanal 2 am Oszilloskop) stellt das Signal X- dar. Beide verwenden dieselbe Genauigkeit von 1,5V pro Division. Dargestellt werden auf der Zeitachse 2ms pro Division. Die beiden Signale werden über eine Dauer von 20ms abgebildet.
Die beiden Ausgangssignale werden gegen Masse gemessen. Die differentielle Spannung wird nicht abgebildet. Sie ist das Ergebnis der folgenden Formel:

$$U_{diff\_X} = U_{X+} - U_{X-}$$

Gleichung 35: Formel zur Berechnung der differentiellen Spannung am X-Kanal

Die Schnittstelle nach *ILDA* soll am X-Kanal zwei bipolare Spannungen (X+ und X-) zwischen +5V und -5V ausgeben. Die differentielle Spannung zwischen den beiden Ausgangssignalen liegt somit zwischen +10V und -10V.

Das Oszilloskopbild (Abbildung 54) wird ausgewertet:

1. Beim Maximum der roten Kurve liegt am Signalausgang X+ eine Spannung von +5V. Die Spannung am Signalausgang X- beträgt -5V (Minimum der grünen Kurve), die differentielle Spannung +10V. Der Laserstrahl wird an die linke äußere Kante der Projektionsfläche gelenkt.
2. Beim Minimum der roten Kurve liegt am Signalausgang X+ eine Spannung von -5V. Die Spannung am Signalausgang X- beträgt +5V (Maximum der grünen Kurve), die differentielle Spannung -10V. Der Laserstrahl wird an die rechte äußere Kante der Projektionsfläche gelenkt.
3. Bei jedem Nulldurchgang der roten und der grünen Kurven liegen an den Signalausgängen X+ und X- 0V an. Die differentielle Spannung beträgt ebenfalls 0V. Der Laserstrahl wird in die Mitte der Projektionsfläche gelenkt.

Die Spannungsbereiche der X-Achse der Schnittstelle nach *ILDA* werden somit eingehalten.

Das folgende Oszilloskopbild zeigt die beiden Signale Y+ und Y-:

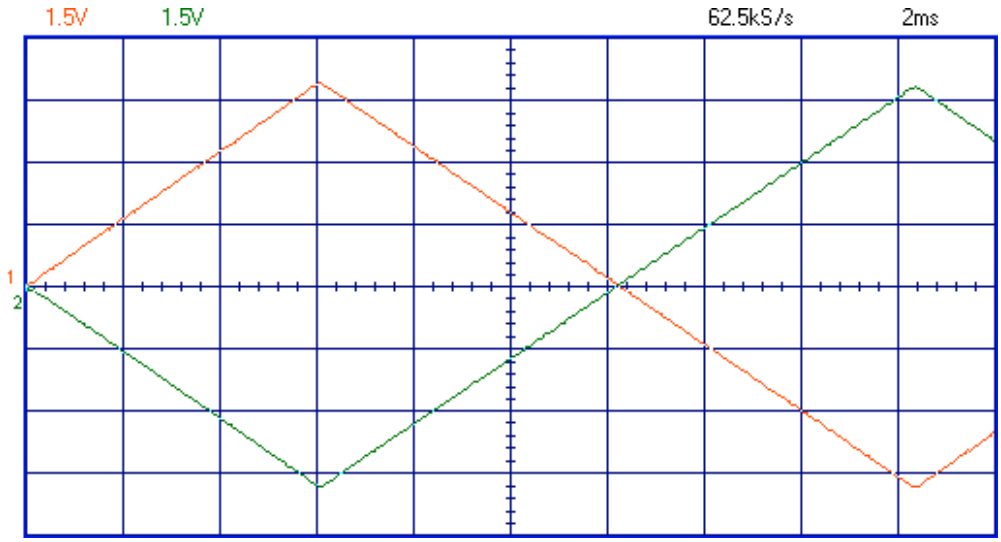


Abbildung 55: Dreieckspannungen am Y-Kanal

Die rote Kurve (Kanal 1 am Oszilloskop) stellt das Signal Y+ dar, die grüne Kurve (Kanal 2 am Oszilloskop) stellt das Signal Y- dar. Beide verwenden dieselbe Genauigkeit von 1,5V pro Division. Dargestellt werden auf der Zeitachse 2ms pro Division. Die beiden Signale werden über eine Dauer von 20ms abgebildet.
Die beiden Ausgangssignale werden gegen Masse gemessen. Die differentielle Spannung wird nicht abgebildet. Sie ist das Ergebnis der folgenden Formel:

$$U_{diff\_Y} = U_{Y+} - U_{Y-}$$

Gleichung 36: Formel zur Berechnung der differentiellen Spannung am Y-Kanal

Die Schnittstelle nach *ILDA* soll am Y-Kanal zwei bipolare Spannungen (Y+ und Y-) zwischen +5V und -5V ausgeben. Die differentielle Spannung zwischen den beiden Ausgangssignalen liegt somit zwischen +10V und -10V.

Das Oszilloskopbild (Abbildung 55) wird ausgewertet:

1. Beim Maximum der roten Kurve liegt am Signalausgang Y+ eine Spannung von +5V. Die Spannung am Signalausgang Y- beträgt -5V (Minimum der grünen Kurve), die differentielle Spannung +10V. Der Laserstrahl wird an die obere Kante der Projektionsfläche gelenkt.
2. Beim Minimum der roten Kurve liegt am Signalausgang Y+ eine Spannung von -5V. Die Spannung am Signalausgang Y- beträgt +5V (Maximum der grünen Kurve), die differentielle Spannung -10V. Der Laserstrahl wird an die untere Kante der Projektionsfläche gelenkt.
3. Bei jedem Nulldurchgang der roten und der grünen Kurven liegen an den Signalausgängen Y+ und Y- 0V an. Die differentielle Spannung beträgt ebenfalls 0V. Der Laserstrahl wird in die Mitte der Projektionsfläche gelenkt.

Die Spannungsbereiche der Y-Achse der Schnittstelle nach *ILDA* werden somit eingehalten.

Zum Veranschaulichen der beiden Achsen wird das Signal X+ an Kanal 1 des Oszilloskopes und das Signal Y+ an Kanal 2 des Oszilloskopes angeschlossen. Die Signale X- und Y- werden offen gelassen. Das Oszilloskop wird in den XY-Betrieb umgeschaltet. Die Lasershowdatei *quadrat.ild* (siehe CD-ROM), die ein großes Quadrat beinhaltet, wird mit einer Auflösung von 12 Bit pro Achse ausgegeben:

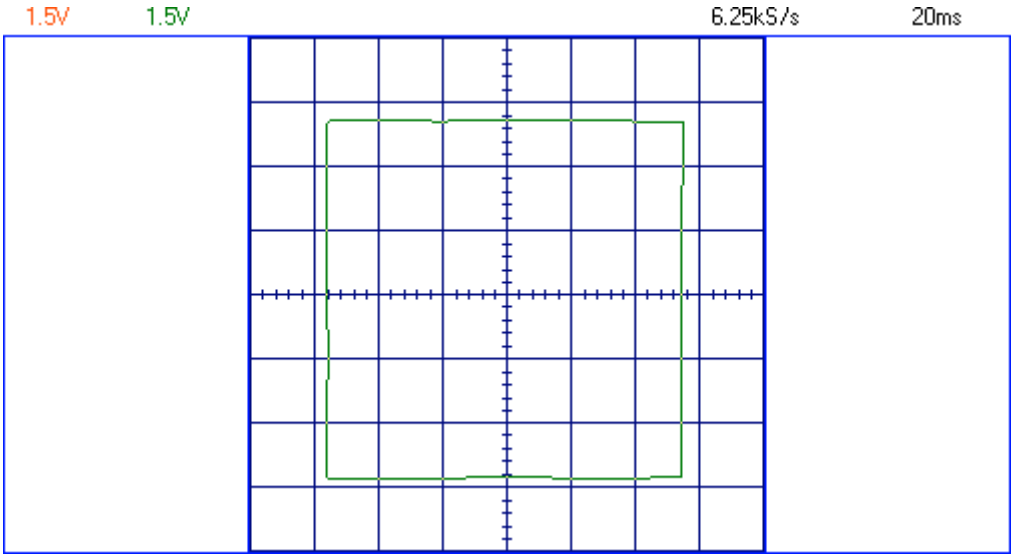


Abbildung 56: maximale Projektionsgröße (Oszilloskop im XY-Betrieb)

Wird anstelle des Oszilloskopes ein Lasersystem angeschlossen, kann die im Oszilloskop grün dargestellte Figur auf der Projektionsfläche betrachtet werden.

9.2 Roter und grüner Farbkanal

Für die Farbausgabe kommen zwei selbst gebaute 4-Bit-DACs zum Einsatz. In den Signalen sind „Treppenstufen“, bedingt durch die geringe Auflösung der DACs, deutlich zu erkennen.

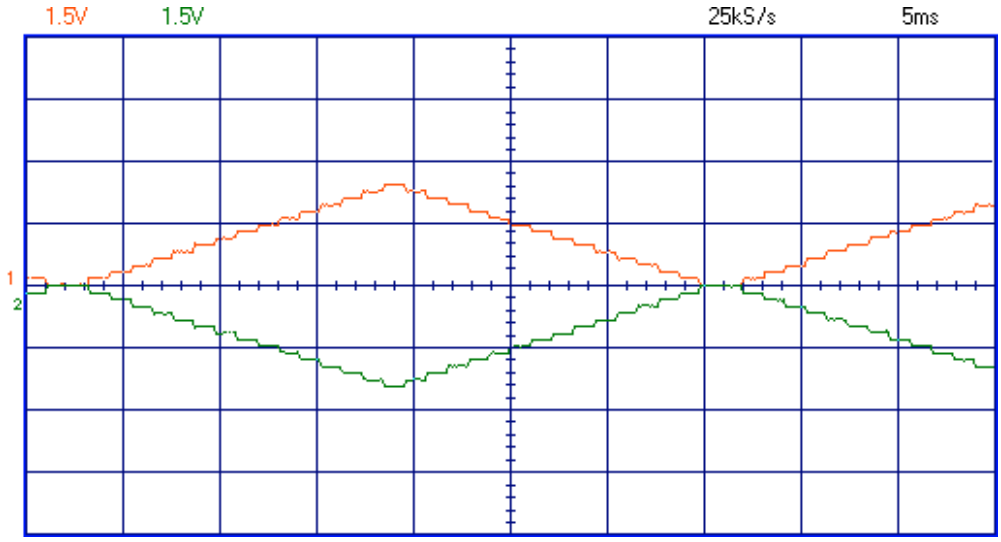


Abbildung 57: Dreieckspannungen am roten Farbkanal

Die rote Kurve (Kanal 1 am Oszilloskop) stellt das Signal R+ dar, die grüne Kurve (Kanal 2 am Oszilloskop) stellt das Signal R- dar. Beide verwenden dieselbe Genauigkeit von 1,5V pro Division. Dargestellt werden auf der Zeitachse 5ms pro Division. Die beiden Signale werden über eine Dauer von 50ms abgebildet.
Die beiden Ausgangssignale werden gegen Masse gemessen. Die differentielle Spannung wird nicht abgebildet. Sie ist das Ergebnis der folgenden Formel:

$$U_{diff\_R} = U_{R+} - U_{R-}$$

Gleichung 37: Formel zur Berechnung der differentiellen Spannung am roten Farbkanal

Die Schnittstelle nach *ILDA* soll am roten Farbkanal zwei unipolare Spannungen ausgeben: Die Spannung des Signals R+ soll zwischen 0V und +2,5V liegen, die Spannung des Signals R- zwischen 0V und -2,5V. Die differentielle Spannung zwischen den beiden Ausgangssignalen liegt somit zwischen 0V und +5V.

Das Oszilloskopbild (Abbildung 57) wird ausgewertet:

1. Beim Maximum der roten Kurve liegt am Signalausgang R+ eine Spannung von +2,5V. Die Spannung am Signalausgang R- beträgt –2,5V (Minimum der grünen Kurve), die differentielle Spannung +5V. Der rote Farbanteil des Lasers wird mit maximaler Helligkeit dargestellt.
2. Bei jedem Minimum der roten Kurve liegt am Signalausgang R+ eine Spannung von 0V. Die Spannung am Signalausgang R- beträgt 0V (Maximum der grünen Kurve). Die differentielle Spannung beträgt ebenfalls 0V. Der rote Farbanteil des Lasers wird komplett abgeschaltet.

Die Spannungsbereiche des roten Farbkanals der Schnittstelle nach *ILDA* werden somit eingehalten.

Die „Treppenstufen“ des roten Farbkanals werden anhand des Signals R+ untersucht:

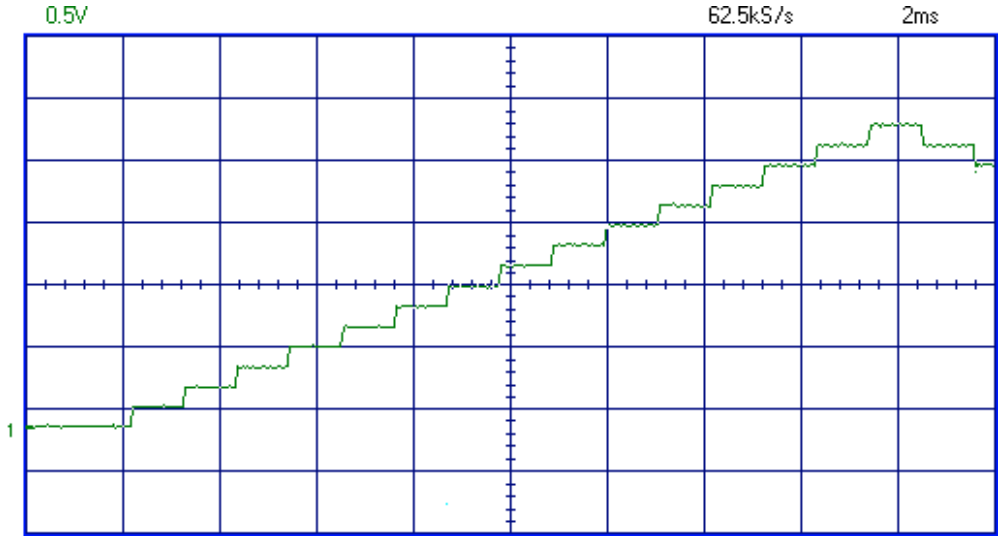


Abbildung 58: Detailansicht des roten Farbkanals (nur R+, verschobene 0V-Linie)

Die Höhe einer jeden „Treppenstufe“ beträgt ca. 165mV. Der Verlauf des im Oszilloskopbild dargestellten Signals und die Höhe der „Treppenstufen“ stimmen mit den Simulationsergebnissen aus Kapitel 6.7.2 überein.

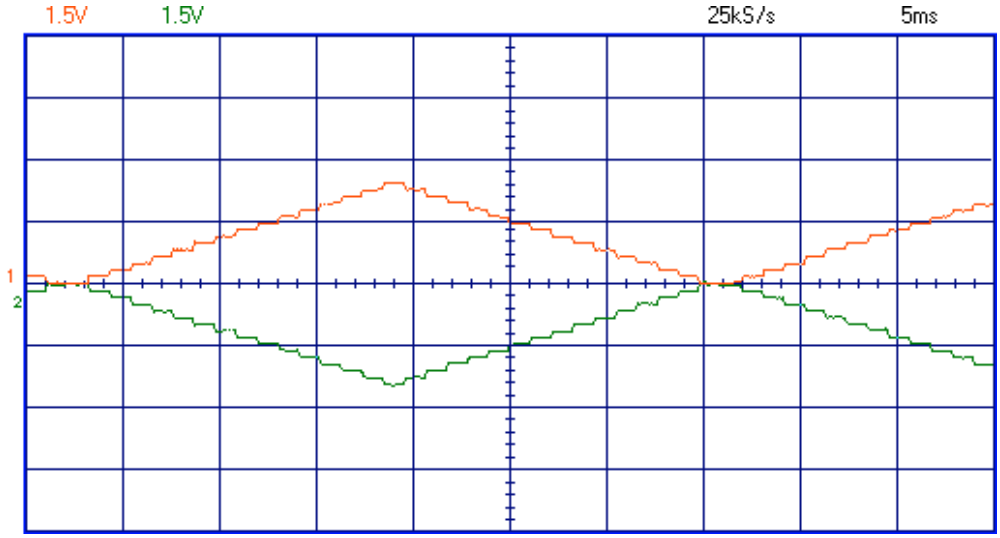


Abbildung 59: Dreieckspannungen am grünen Farbkanal

Die rote Kurve (Kanal 1 am Oszilloskop) stellt das Signal G+ dar, die grüne Kurve (Kanal 2 am Oszilloskop) stellt das Signal G- dar. Beide verwenden dieselbe Genauigkeit von 1,5V pro Division. Dargestellt werden auf der Zeitachse 5ms pro Division. Die beiden Signale werden über eine Dauer von 50ms abgebildet.
Die beiden Ausgangssignale werden gegen Masse gemessen. Die differentielle Spannung wird nicht abgebildet. Sie ist das Ergebnis der folgenden Formel:

$$U_{diff\_G} = U_{G+} - U_{G-}$$

Gleichung 38: Formel zur Berechnung der differentiellen Spannung am grünen Farbkanal

Die Schnittstelle nach *ILDA* soll am grünen Farbkanal zwei unipolare Spannungen ausgeben: Die Spannung des Signals G+ soll zwischen 0V und +2,5V liegen, die Spannung des Signals G- zwischen 0V und -2,5V. Die differentielle Spannung zwischen den beiden Ausgangssignalen liegt somit zwischen 0V und +5V.

Das Oszilloskopbild (Abbildung 59) wird ausgewertet:

1. Beim Maximum der roten Kurve liegt am Signalausgang G+ eine Spannung von +2,5V. Die Spannung am Signalausgang G- beträgt –2,5V (Minimum der grünen Kurve), die differentielle Spannung +5V. Der grüne Farbanteil des Lasers wird mit maximaler Helligkeit dargestellt.
2. Bei jedem Minimum der roten Kurve liegt am Signalausgang G+ eine Spannung von 0V. Die Spannung am Signalausgang G- beträgt 0V (Maximum der grünen Kurve). Die differentielle Spannung beträgt ebenfalls 0V. Der grüne Farbanteil des Lasers wird komplett abgeschaltet.

Die Spannungsbereiche des grünen Farbkanals der Schnittstelle nach *ILDA* werden somit eingehalten. Die „Treppenstufen“ werden nicht erneut untersucht (vgl. Abb. 58).

9.3 Das Shuttersignal

Das Shuttersignal ist ein TTL-Signal (Transistor-Transistor-Logik). Zum Testen wird keine spezielle Funktion benötigt. Stattdessen kommt eine zum Testzeitpunkt geschriebene Schleife, die mit einer Wartezeit von 1ms das Signal am entsprechenden Ausgangspin des Mikroprozessors an- und abschaltet.

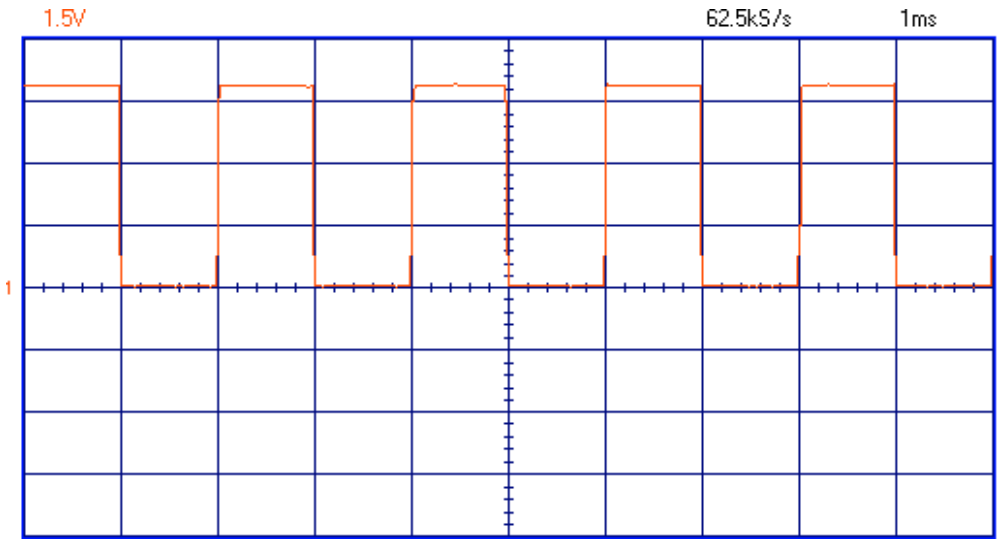


Abbildung 60: Rechteckspannung am Shutterausgang

Die rote Kurve (Kanal 1 am Oszilloskop) stellt das Shuttersignal dar. Die Darstellung verwendet eine Genauigkeit von 1,5V pro Division. Mit der Einstellung von 1ms pro Division können fünf komplette Perioden abgebildet werden. Die Schnittstelle nach *ILDA* soll am Shutterausgang eine unipolare, digitale Spannung ausgeben: Ist die Spannung 0V, wird der Laserstrahl abgeschaltet. Bei +5V wird der Laser eingeschaltet.
Das Oszilloskopbild in Abbildung 60 zeigt, dass das Shuttersignal die Vorgaben der Schnittstelle nach *ILDA* einhält.

10 Abschließende Testphase der fertigen Showlasersteuerung

Nach Fertigstellung der Hard- und Software folgte eine ausführliche Testphase, in der die Zuverlässigkeit des Gerätes und die Benutzerfreundlichkeit auf die Probe gestellt wurden.

10.1 Testen mit verschiedenen SD-Karten

Das Gerät wurde mit neun verschiedenen SD-Karten getestet. Für diesen Test wurden die SD-Karten nicht neu formatiert. Beliebige Dateien und Ordner wurden auf die SD-Karten kopiert, auch die drei Dateien *kreis.ild, dreieck.ild* und *quadrat.ild* (siehe CD-ROM). Derselbe Inhalt wurde zusätzlich in einen neu erstellten Ordner auf der SD-Karte kopiert.

Nr.	Hersteller	Speicherplatz
1	*HP*	32MB
2	*SimpleTech*	64MB
3	*Panasonic*	128MB
4	*SanDisk*	256MB
5	*extrememory*	256MB
6	*Medion*	512MB
7	*Transcend*	1GB
8	*Toshiba*	1GB
9	*Panasonic*	2GB

Tabelle 15: SD-Karten zum Testen des Gerätes

Nur die Karte mit der Nummer 8 (*Toshiba*, 1GB) ließ sich nicht ansprechen. Diese Karte reagierte auch nach mehrfachem Neueinlegen nicht auf das CMD0. Die anderen acht Karten funktionierten ohne Probleme. Von jeder der funktionierenden SD-Karten wurde für eine Dauer von fünf Minuten die Datei *dreieck.ild* auf den Laser ausgegeben. Diese Laserausgaben haben bei keiner dieser Karten „geruckelt“.

10.2 Ausgabe einer Lasershow im Langzeittest

Die Datei *dreieck.ild* (siehe Abbildung 72 in Kapitel 13.5) wurde auf eine SD-Karte des Herstellers *Medion* (512MB, siehe Kartennummer 6 in Tabelle 15) kopiert und 60 Minuten lang mit einer Auflösung von 12 Bit in einer Endlosschleife ausgegeben. Das Dreieck wurde dauerhaft mit spitzen Ecken und geraden Linien ohne Verzerrungen dargestellt.
Die Ausgabe wurde gestoppt, die Auflösungsgenauigkeit auf 8 Bit umgestellt. Mit dieser Einstellung wurde dieselbe Datei weitere 60 Minuten in einer Endlosschleife ausgegeben. Das Dreieck wurde mit spitzen Ecken, jedoch mit leichten Verzerrungen bei den geraden Linien dargestellt. Die auftretenden Verzerrungen lassen sich mit der geringen Auflösung von 8 Bit pro Achse erklären.
In einem zweiten Langzeittest wurde die Datei *dreieck.ild* über 240 Minuten lang mit einer Auflösung von 12 Bit ausgegeben. Wieder wurde das Dreieck während der gesamten Dauer mit spitzen Ecken und geraden Linien ohne Verzerrungen projiziert.
Ein dritter Test wurde durchgeführt: Nachdem das Gerät eine Stunde lang ohne eingelegte Karte im Leerlauf betrieben wurde, erfolgte das Einlegen der SD-Karte und das Ausgeben der

Datei *dreieck.ild* über einen Zeitraum von 30 Minuten. Das Dreieck wurde ebenfalls mit spitzen Ecken und geraden Linien ohne Verzerrungen dargestellt.
Bei einem vierten und letzten Test wurde die Datei *dreieck.ild* ausgegeben. Während der laufenden Ausgabe wurde die SD-Karte entfernt. Dieser Test wurde mehr als 30-mal wiederholt. Die Ausgabe brach unmittelbar nach jedem Entfernen der SD-Karte ab, der Laser wurde sofort abgeschaltet.
Festgestellt wurde bei diesen vier durchgeführten Tests, dass das Gerät über längere Zeiträume mit einfachen Lasergrafiken zuverlässig und sicher arbeitet.

Auch ein schnelles Wechseln der auszugebenden Laserdateien wurde getestet: Die Datei *dreieck.ild* wurde mit einer Auflösung von 12 Bit in einer Endlosschleife ausgegeben. Nach einigen Sekunden wurde die Ausgabe gestoppt und das Ausgabemenü verlassen. In der Verzeichnisanzeige wurde im Anschluss die Datei *kreis.ild* ausgewählt und ebenfalls in einer Endlosschleife ausgegeben. Nach dem Stoppen dieser Ausgabe wurden Ausgabemenü und Verzeichnisanzeige wieder verlassen. Im Hauptmenü wurde der Punkt „Basiseinstellungen ändern“ ausgewählt. Hier wurde die Auflösung auf 8 Bit umgeschaltet. Anschließend wurde die Datei *quadrat.ild* ausgewählt und in einer Endlosschleife ausgegeben.
Festgestellt wurde, dass die Bedienung des Gerätes einfach und benutzerfreundlich ist, sie läßt schnelle Dateiwechsel und Parameteränderungen zu.

Da das vorhandene Lasersystem keine Farbausgabe unterstützt (siehe Kapitel 5), wurde diese nicht getestet, sondern nur gemessen (siehe Kapitel 9.2).

11 Fazit

Eine ausführliche Literatur- und Quellenrecherche sorgte für ein tiefgreifendes Verständnis der Themenbereiche „SD-Karte“, „FAT-Dateisystem“ und „*ILDA*-Richtlinien“. Durch meine gründliche und intensive Planungsphase, in der die Bauteile für die Showlasersteuerung ausgesucht, verglichen und deren Kompatibilität kritisch untersucht wurden, konnten Fehler und Probleme von vornherein minimiert werden. Auch während des Aufbaus stellten sich keine Schwierigkeiten ein.

Das einzige Problem, welches auch nicht gelöst werden konnte, betraf die integrierte USB-Schnittstelle mit dem *FT232 BL* des Herstellers *FTDI Chip*. Daher ist die USB-Schnittstelle für eine optionale zukünftige Erweiterung des Gerätes zwar in dem Platinenlayout integriert, jedoch kann diese von der programmierten Software noch nicht angesprochen werden. Diese müsste entsprechend erweitert werden.
Leider wurde der *FT232 BL* von keinem der beim Testen verwendeten PCs erkannt, weder beim Anschluss an einen USB 1.1-Port noch an einen USB 2.0-Port. Mit dem kostenlos herunterladbaren Programm *USB Device Viewer* [24] von *Microsoft®* wurde festgestellt, dass jeder Verbindungsparameter des Chips - entgegen allen Erwartungen - eine Null enthielt.
Auch ein separater Aufbau, der nur aus der in Abbildung 23 in Kapitel 6.4 gezeigten Schaltung bestand, zeigte trotz neu gekaufter Bauteile kein anderes Resultat. Daraufhin wurde der *FT232 BL* nochmals erneuert, doch das Resultat blieb unverändert. Der Chip konnte während der Entwicklung des Projektes nicht in Betrieb genommen werden, obwohl die verwendete Schaltung exakt den Vorgaben des Datenblattes entsprach. Auch die Suche im Internet verwies immer wieder auf den genauen Nachbau der Schaltungen aus dem Datenblatt des *FT232 BL*. Aufgrund dieser Probleme wurde die USB-Schnittstelle auf der Platine nicht mit bestückt und von der weiteren Entwicklung ausgenommen.

Das Laser-CPU-Board ist, bis auf wenige Bauteile, komplett mit SMD-Bauteilen bestückt, während das DAC-Board nicht ein einziges SMD-Bauteil beherbergt. Diese Vorgehensweise habe ich gewählt, um den Einsatz verschiedener Bauformen anzuwenden. Außerdem waren nicht von allen verwendeten Bauteilen SMD-Versionen erhältlich.
Das DAC-Board enthält die Schnittstelle zum Lasersystem und ist daher direkt von außen „erreichbar“. Durch einen Anschlussfehler oder durch ein defektes Lasergerät könnten Bauteile auf dieser Platine beschädigt oder zerstört werden. Daher sind für sämtliche ICs auf dieser Platine IC-Fassungen vorgesehen, damit die betroffenen ICs bei einem Defekt leicht ausgetauscht werden können.
Das Laser-CPU-Board enthält die Bauteile, die mit hohen Übertragungsgeschwindigkeiten arbeiten. Die Leiterbahnen zwischen den ICs sollten daher möglichst kurz sein. Hier bieten sich Bauteile im SMD-Gehäuse geradezu an.
Als richtige Entscheidung hat sich herausgestellt, die Platinen für das Laser-CPU-Board und das DAC-Board professionell von einer Firma ätzen zu lassen. Selbst belichtete und geätzte Platinen wären besonders bei SMD-Bauformen und den damit verbundenen extrem dünnen Leiterbahnen und deren Abständen unzuverlässig.

Interessant war es, ein Gerät zu entwickeln, das sich in eine Umgebung mit genau festgelegten Standards einpassen muss (FAT, SD-Karte, Schnittstelle und Dateiformat nach *ILDA*). Um so erfreulicher ist das Endergebnis, wenn das eigenständig entwickelte und selbst gebaute Gerät mit anderen Geräten (z.B. PC, Laptop) zusammenarbeitet und seine Aufgabe ohne festgestellte Störungen erfüllt.

Das Kriterium „low-cost“ als Ziel der Diplomarbeit wird erfüllt: Die Kosten für die komplette Showlasersteuerung liegen (inkl. Gehäuse und professionell geätzten Platinen) unter 350€, wobei sämtliche Bauteile und das Gehäuse zu Endverbraucherpreisen eingekauft wurden.
Einzig und allein zum Verwalten des Dateninhalts der SD-Karte wird ein PC benötigt. Da dieser für den weiteren Betrieb der Showlasersteuerung nicht erforderlich ist, wird auch das Projektziel der PC-Unabhängigkeit erfüllt.
Das Gerät ist kompakt und stabil gebaut. Durch die hardwareentprellten Taster, das hintergrundbeleuchtete Grafikdisplay und eine einfache Menüstruktur ist die Bedienung sehr benutzerfreundlich. Einfache Lasershows können mit guten Ergebnissen präsentiert werden.

Der Nachbau allerdings sollte aus Sicherheitsgründen ausschließlich Experten vorbehalten bleiben, da die verwendeten Technologien sehr komplex sind und gute Fachkenntnisse erfordern. Des Weiteren müssen bei dem Umgang mit Laserstrahlen Sicherheitsbestimmungen unbedingt eingehalten werden.

Die entwickelte Showlasersteuerung hat allerdings noch keine Marktreife erlangt. Der Aufbau des Gerätes mit dem Gehäuse ist fertigungstechnisch noch nicht optimiert (z.B. Aufbau auf einer größeren Platine, andere Gehäuseform, angepasste Taster usw.). Ebenfalls müssten noch diverse weitere Tests durchgeführt werden, wie z.B. Tests zur Elektromagnetischen Verträglichkeit (EMV), da das Gerät auf keinen Fall durch andere Geräte beeinflusst werden darf. Aber auch die Software sollte durch weitere Testreihen auf Stör- und Fehlersicherheit überprüft werden. Zudem müsste das Gerät noch zahlreiche Zulassungsverfahren durchlaufen.

12 Quellenverzeichnis

[1] International Laser Display Association: Our mission. Erschienen als Online-Dokumentation unter http://www.laserist.org/mission.htm, Stand vom 24.10.2007.

[2] Michael, L.: Standards – ISP-BD25. Erschienen als Online-Dokumentation unter http://www.laserfx.com/Backstage.LaserFX.com/Standards/ILDA-DB25.pdf, Stand vom 11.03.2007.

[3] Heminover, S. (Aura Technologies, Inc.); Murphy, P. (Pangolin Laser Software); Plughoff, F.; Plughoff, K.: *ILDA* Image Data Transfer Format. Erschienen als Online-Dokumentation unter http://www.laserfx.com/Backstage.LaserFX.com/Standards/ILDAframeFormat.pdf, Stand vom 11.03.2007.

[4] SanDisk Corporation: SanDisk Secure Digital Card – Product Manual. Erschienen als Online-Dokumentation unter www.cs.ucr.edu/~amitra/sdcard/ProdManualSDCardv1.9.pdf, Stand von 12.2003.

[5] Wikipedia: File Allocation Table. Erschienen als Online-Dokumentation unter http://de.wikipedia.org/wiki/File_Allocation_Table, Stand vom 09.09.2007.

[6] Stoffregen, P.: Understanding FAT32 Filesystems. Erschienen als Online-Dokumentation unter http://www.pjrc.com/tech/8051/die/fat32.html, Stand vom 24.02.2005.

[7] Dobiash, J.: FAT16 Structure Information. Erschienen als Online-Dokumentation unter http://home.teleport.com/~brainy/fat16.htm, Stand vom 17.06.1999.

[8] Dobiash, J.: FAT32 Structure Information. Erschienen als Online-Dokumentation unter http://home.teleport.com/~brainy/fat32.htm, Stand vom 04.12.2005.

[9] Vishay Semiconductors: BC337. Erschienen als Datenblatt unter http://www.reichelt.de/?;ACTION=7;LA=6;OPEN=1;INDEX=0;FILENAME=A100%252FBC337_BC338_VIS.pdf;SID=315-Y8x6wQAR8AAGuRVwM9d8e167c1f16cd8afd5d910b5a31c5f9, Stand vom 08.02.2002.

[10] Atmel Corporation: AVR Studio 4. Erschienen als Software unter http://www.atmel.com/dyn/resources/prod_documents/aStudio413SP1b557.exe, Stand von 09.2007.

[11] C-Compiler WinAVR. Erschienen als open-source Software unter http://garr.dl.sourceforge.net/sourceforge/winavr/WinAVR-20070525-install.exe, Stand vom 25.05.2007.

[12] Atmel Corporation: ATmega64. Erschienen als Datenblatt unter http://www.atmel.com/dyn/resources/prod_documents/doc2490.pdf, Stand von 08.2007.

[13] Atmel Corporation: ATmega128. Erschienen als Datenblatt unter http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf, Stand von 03.2006.

[14] Hitachi Semiconductor: HM628400B Series 4M SRAM (512-kword x 8-bit). Erschienen als Datenblatt unter http://www.tranzistoare.ro/datasheets/90/177156_DS.pdf, Stand vom 24.08.1999.

[15] ON Semiconductor: MC74VHC573 Octal D-Type Latch with 3-State Output. Erschienen als Datenblatt unter http://www.tranzistoare.ro/datasheets2/20/208656_1.pdf, Stand von 04.2000.

[16] FTDI Chip: FT232BL USB UART. Erschienen als Datenblatt unter http://www.ftdichip.com/Documents/DataSheets/DS_FT232BL.pdf, Stand von 12.2005.

[17] FTDI Chip: D2XX Drivers. Erschienen als USB-Gerätetreiber unter http://www.ftdichip.com/Drivers/CDM/CDM%202.02.04.exe, Version 2.02.04 vom 03.07.2007.

[18] FTDI Chip: MProg 3.0a. Erschienen als Software unter http://www.ftdichip.com/Resources/Utilities/MProg3.0_Setup.exe, Stand vom 26.07.2007.

[19] Displaytech Ltd: LCD Module 64128A Series. Erschienen als Datenblatt unter http://www.reichelt.de/?;ACTION=7;LA=6;OPEN=1;INDEX=0;FILENAME=A500%252FLCD64128A%2523DIS.pdf;SID=25GowYIn8AAAIAAD45zMc5d097f0d98893274364929f594716d12, Stand vom 04.01.2007.

[20] Vohburger, M: Tasten entprellen, eine kleine Einführung. Erschienen als Online-Dokumentation unter http://www.loetstelle.net/praxis/entprellen/entprellen.php, Stand vom 09.04.2007.

[21] Aimtec: Series AM2DZ, 2Watt DC-DC-Converter. Erschienen als Datenblatt unter http://www.aimtec.com/HighResolution/AM2D-Z.PDF, Stand vom 23.04.2007.

[22] Cadsoft Computer GmbH: Eagle Layout Editor. Erschienen als Software unter ftp://ftp.cadsoft.de/eagle/program/4.16r2/eagle-win-ger-4.16r2.exe, Stand vom 13.12.2006.

[23] Analog Devices: CMOS 12 Bit Buffered Multiplying DAC AD7545. Erschienen als Datenblatt unter http://www.analog.com/UploadedFiles/Data_Sheets/AD7545.pdf, Stand von 05.2000.

[24] Microsoft® Corporation: USB Device Viewer. Erschienen als Software unter http://www.ftdichip.com/Resources/Utilities/usbview.zip, Stand vom 26.07.2007.

13 Anhang

13.1 Verzeichnis fachspezifischer Abkürzungen

ACMD	Application Command
AD0...7	Adress- und Datenleitung 0...7
ALE	Adress Latch Enable
ASCII	American Standard Code for Information Interchange
AVR	Prozessorfamilie des Herstellers *Atmel*
BIOS	Basic Input Output System
CD(S)	Card Detect (Switch)
CID	Card Identification Register
CLK	Clock
CMD	Command
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CS	Chip Select
CSD	Card Specific Data
DAC	Digital Analog Converter
DOS-FAT	Dateisystem unter dem Betriebssystem DOS (*Microsoft®*)
DPSS	Diode Pumped Solid State
D-SUB-Stecker	D (shaped) Subminiature Stecker
E	Enable (Signal)
EECR	EEPROM Control Register
EEMWE	EEPROM Master Write Enable
EEPROM	Electrically Erasable Programmable Read-Only Memory
EEWE	EEPROM Write Enable
EMV	Elektromagnetische Verträglichkeit
FAT	File Allocation Table
FF	Flipflop
GND	Ground, Masse
IC	Integrated Circuit
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
ILDA	*International Laser Display Association*
I/O	Input / Output
IRQ	Interrupt
ISP	In System Programmer
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LBA	Logical Block Adressing
LCD	Liquid Crystal Display
LE	Latch Enable
LSB	Least Significant Bit
MBR	Master Boot Record
MISO	Master In Slave Out
MOSI	Master Out Slave In
MSB	Most Significant Bit
OCR	Operating Conditions Register
-OE	Output Enable

OP	Operationsverstärker
OCRA	Output Compare Register A
PDI	Programm Data Input
PDO	Programm Data Output
RAM	Random Access Memory
-RD	Read
REFX / REFY	Referenzspannungen für den X- bzw. Y-Kanal
RGB	Rot / Grün / Blau
RISC	Reduced Instruction Set Computing
RS	Data oder Instruction Code (Signal vom verwendeten Grafikdisplay)
RST	Reset
R/W	Read / Write
RxD	Receive Data
SCK	Serial Clock
SD Card	Secure Digital Card
SMD	Surface Mounted Devices
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SREG	Bezeichnung eines Systemregisters der Mikroprozessorfamilie AVR
t_{ah}	Adress hold time (LCD)
t_{as}	Adress setup time (LCD)
t_{dh}	Data hold time (LCD)
t_{ds}	Data setup time (LCD)
t_{PHL}	Propagation Delay, high to low (ext. SRAM)
t_{PLH}	Propagation Delay, low to high (ext. SRAM)
t_{whE}	E high level witdh
t_{wlE}	E low level witdh
TTL	Transistor-Transistor-Logik
TxD	Transmit Data
UART	Universal Asynchronous Receiver Transmitter
USART	Universal Synchronous/Asynchronous Receiver Transmitter
USB	Universal Serial Bus
Vcc	Voltage of the common collector, positive Versorgungsspannung eines ICs
VDC	Gleichspannung
VDD	V drains, positive Versorgungsspannung eines ICs
-WE	Write Enable
WP(S)	Write Protection (Switch)
-WR	Write

13.2 Schaltpläne und Platinenlayouts

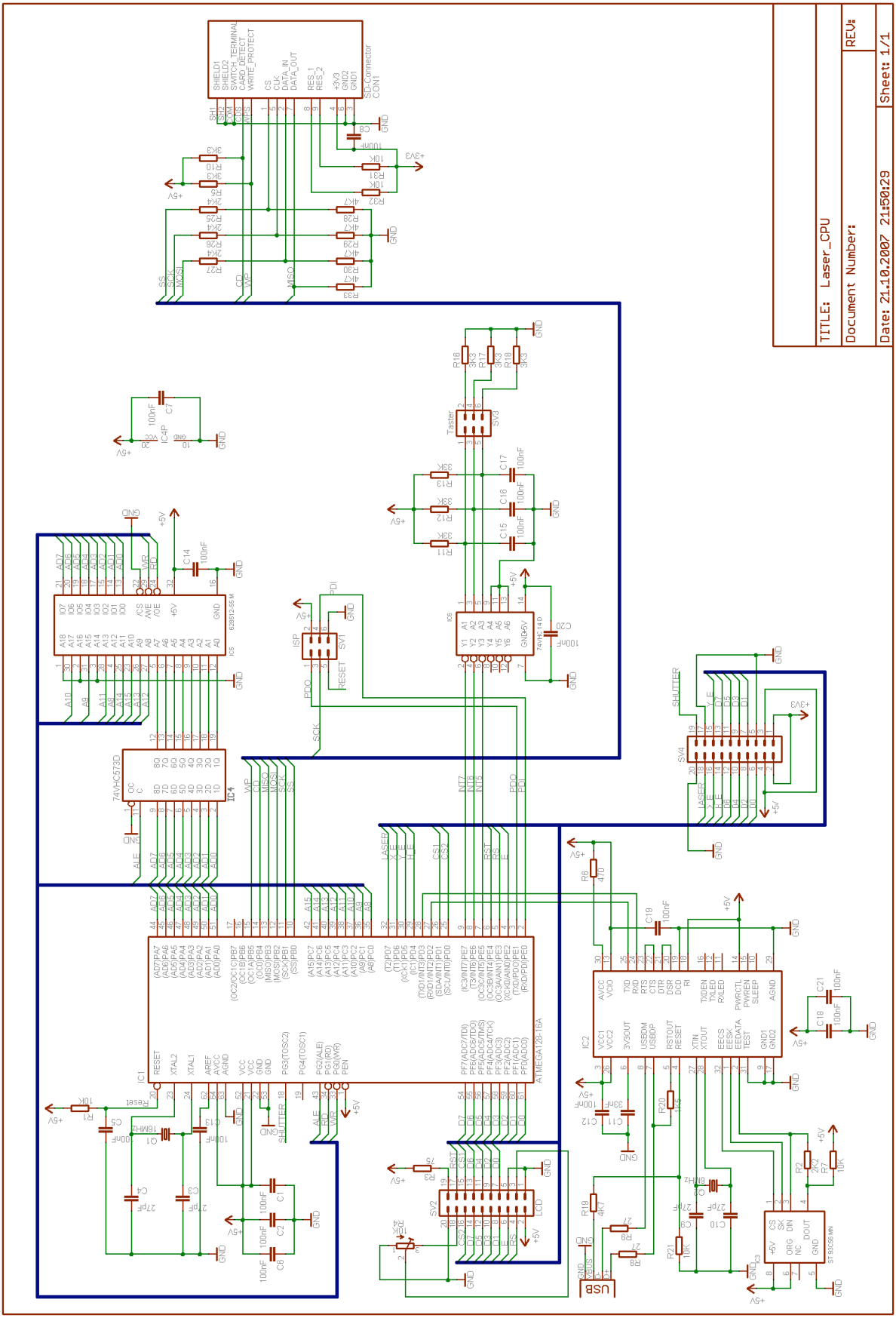


Abbildung 61: Schaltplan des Laser-CPU-Boards

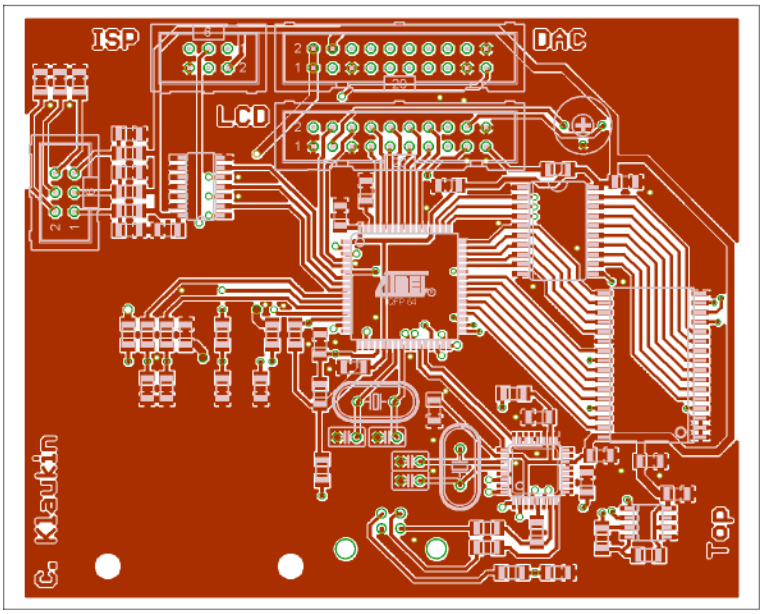


Abbildung 62: Bestückungsseite des Laser-CPU-Boards in Originalgröße

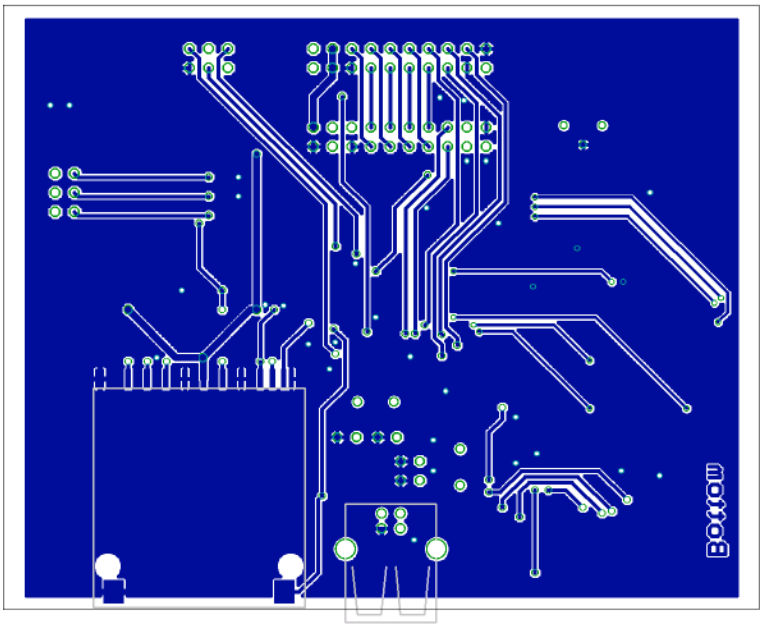


Abbildung 63: Lötseite des Laser-CPU-Boards in Originalgröße

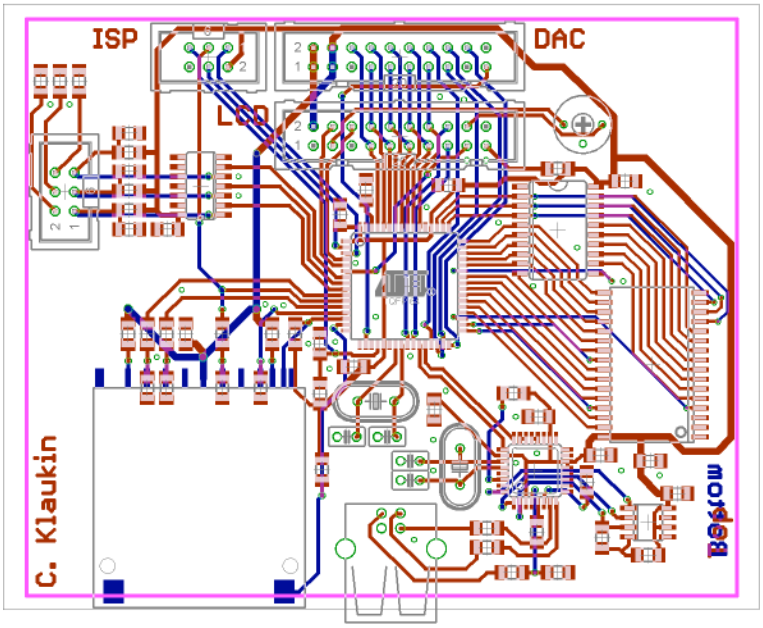


Abbildung 64: Layer des Laser-CPU-Boards übereinander positioniert (Originalgröße)

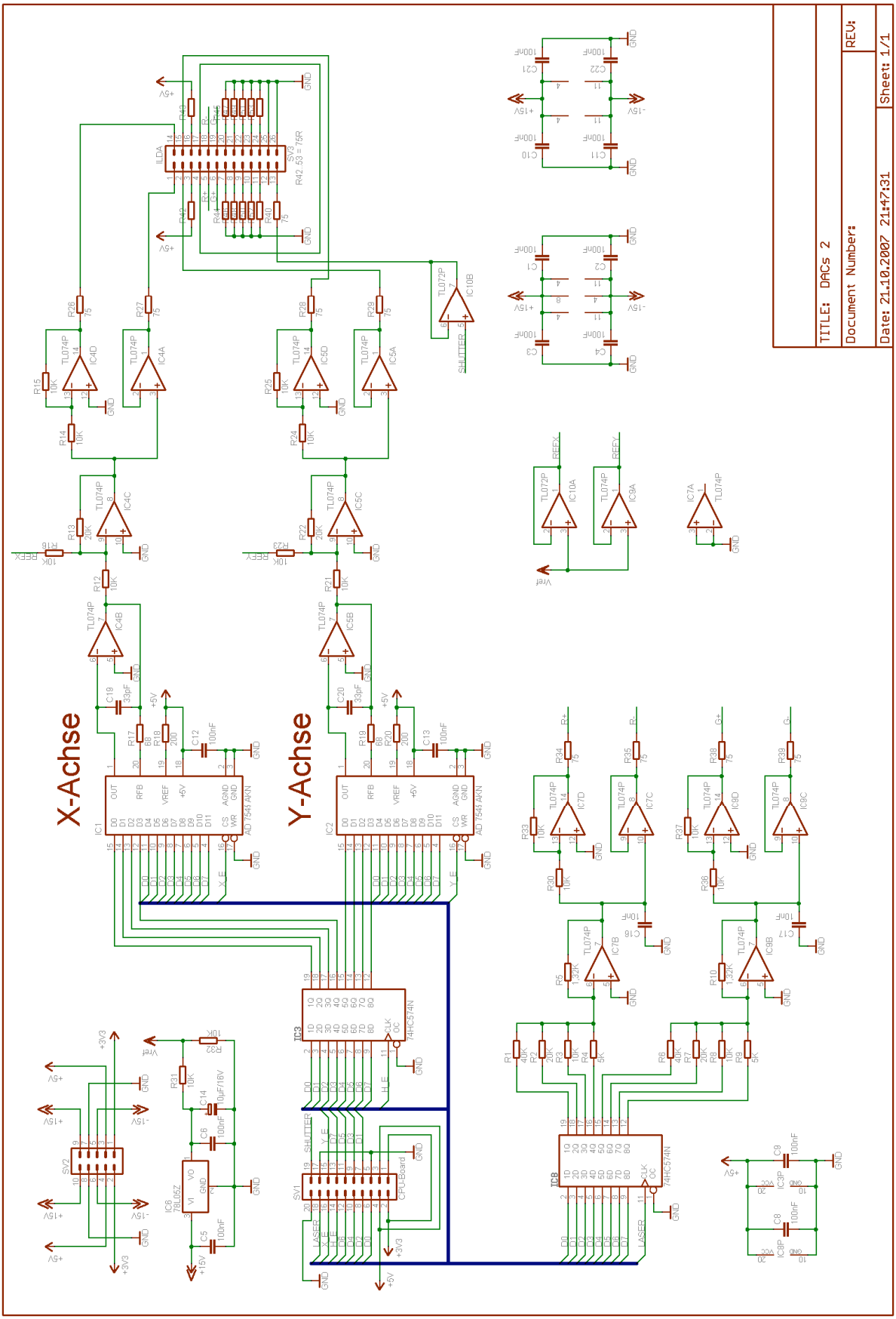


Abbildung 65: Schaltplan des DAC-Boards

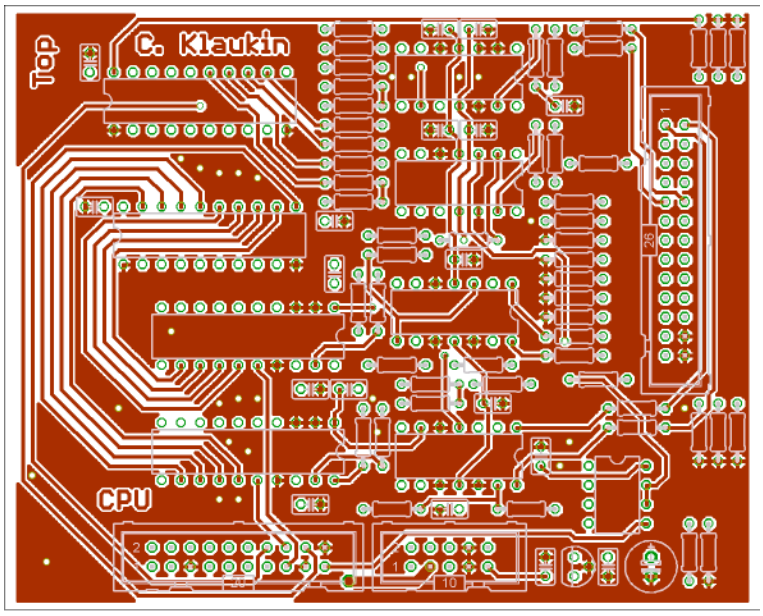


Abbildung 66: Bestückungsseite des DAC-Boards in Originalgröße

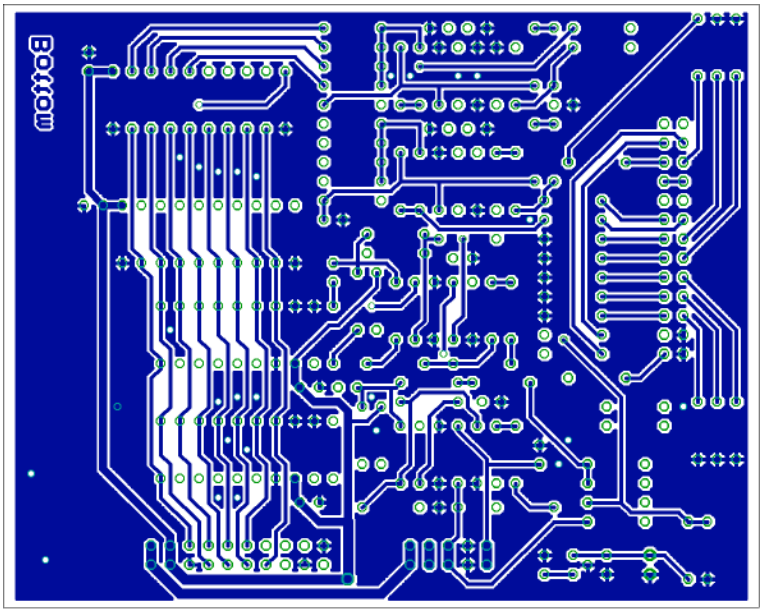


Abbildung 67: Lötseite des DAC-Boards in Originalgröße

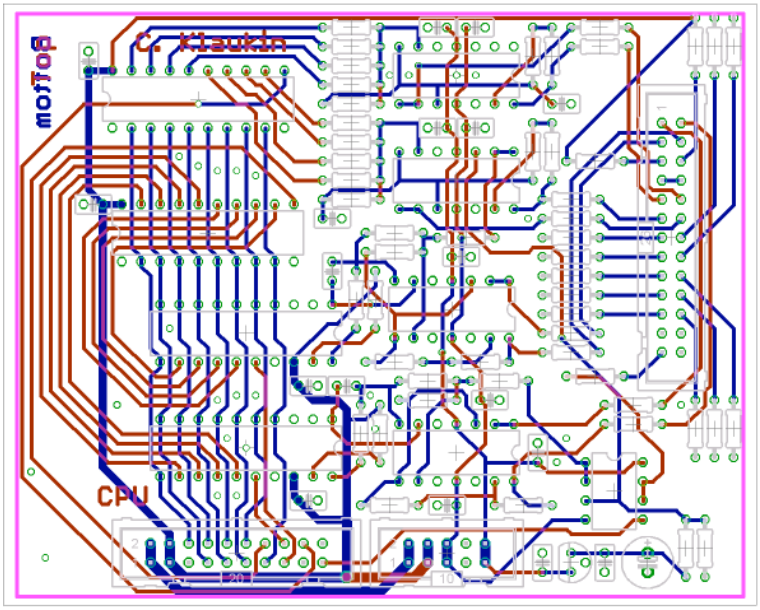


Abbildung 68: Layer des DAC-Boards übereinander positioniert (Originalgröße)

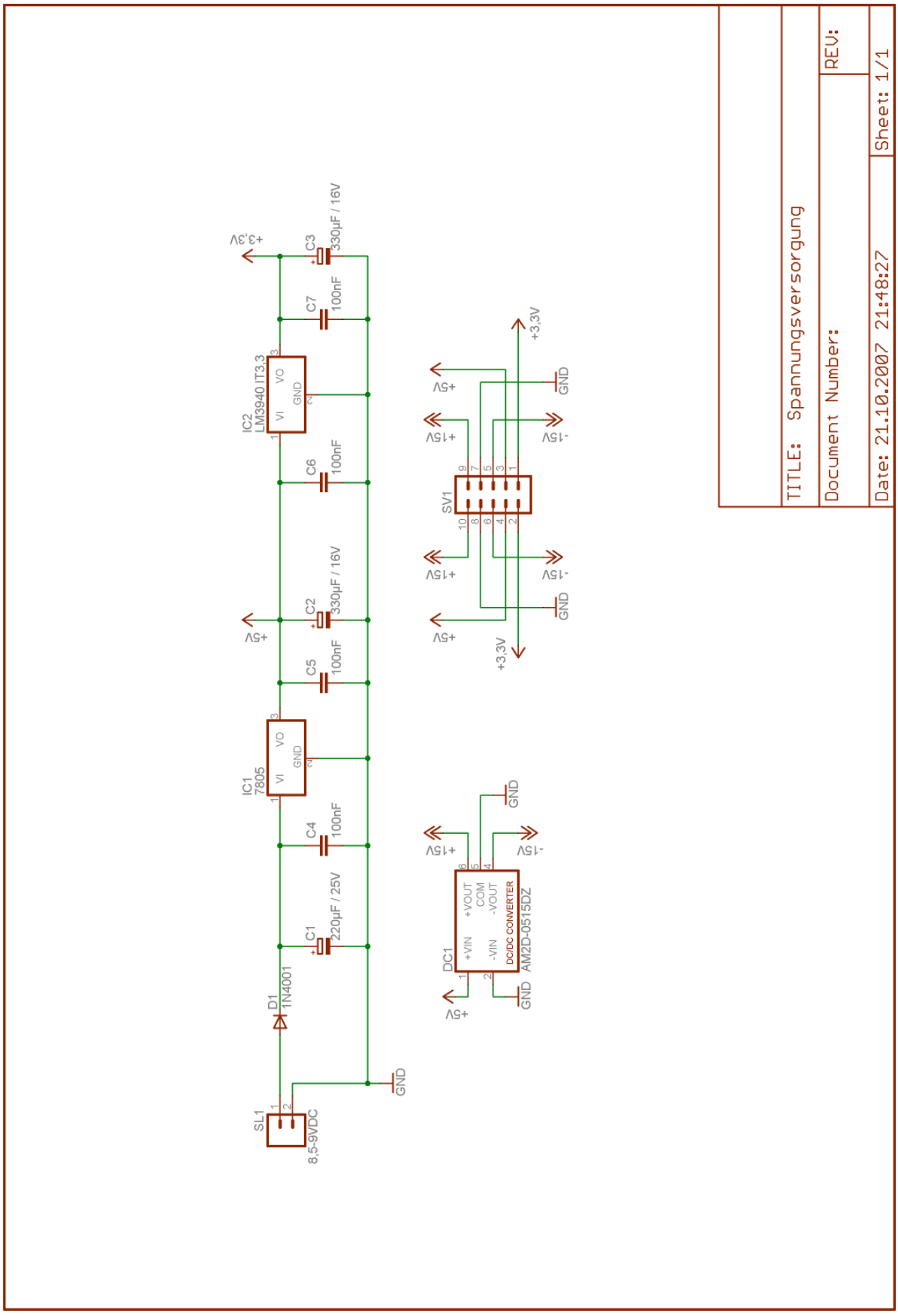


Abbildung 69: Schaltplan der Spannungsversorgung

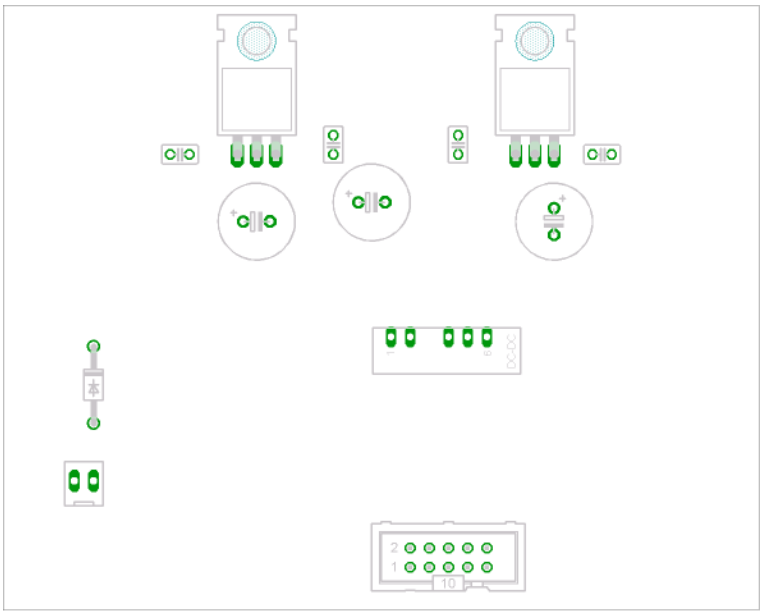


Abbildung 70: Bestückungsseite der Spannungsversorgung in Originalgröße

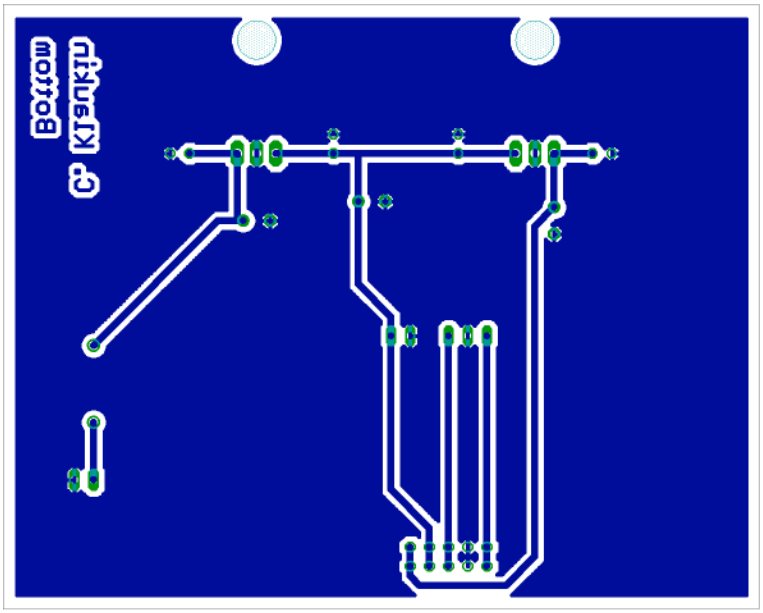


Abbildung 71: Lötseite der Spannungsversorgung in Originalgröße

13.3 Stücklisten

In diesem Kapitel werden sämtliche Stücklisten für die selbst gebauten und bestückten Platinen aufgeführt.

13.3.1 Stückliste für das Testlasersystem

13.3.1.1 Stückliste für die Signalanpassung der Scannertreiber

Aufgeführt sind die Bauteile für einen Kanal. Für zwei Kanäle wird die doppelte Anzahl benötigt.
Die beiden Kondensatoren C1 und C2 sind im mit PSpice erstellten Schaltplan nicht mit abgebildet. Diese werden in unmittelbarer Nähe der Spannungsversorgungspins von IC1 zwischen +15VDC und Masse bzw. zwischen -15VDC und Masse geschaltet.
Die Signalversorgung erfolgt über den Stecker X1, der in der Stückliste der Laserdiodenansteuerung mit aufgeführt ist.

Bezeichnung in dem Schaltplan von PSpice	Bauteil
R1, R2, R3, R6	Kohleschicht-Widerstand, 10KΩ, 5%, 0,25W
R4, R5	Kohleschicht-Widerstand, 20KΩ, 5%, 0,25W
C1, C2	Folienkondensator, 100nF, 20%, Rastermaß 2,5mm
IC1 (in PSpice: N1A - N1D)	*TL074*, Gehäuse DIP-14

Tabelle 16: Stückliste für die Signalanpassung eines Scannertreibers

13.3.1.2 Stückliste für die Laserdiodenansteuerung

Bezeichnung im Schaltplan	Bauteil
R1, R2	Kohleschicht-Widerstand, 4,7KΩ, 5%, 0,25W
R3	Kohleschicht-Widerstand, 100Ω, 5%, 0,25W
R4	Kohleschicht-Widerstand, 2,2KΩ, 5%, 0,25W
C1	Folienkondensator, 100nF, 20%, Rastermaß 2,5mm
IC1	*74HC08*, Gehäuse DIP-14
T1	*BC337-25*, Gehäuse TO-92
SL1	Stiftleiste, 2-polig, gerade, einreihig, Rastermaß 2,54mm
X1	D-SUB-Stecker, 25-polig, für Flachbandkabel

Tabelle 17: Stückliste für die Laserdiodenansteuerung

13.3.2 Stücklisten für die Lasersteuerung

- 1x Grafik-LCD *Displaytech Ltd 64128A-3LP*
- 3x Taster, Schließkontakt, zum Einbauen
- 1x D-SUB-Buchse, 25-polig, für Flachbandkabel
- 1x Einbaubuchse für Niederspannung, Metall, ohne Schaltkontakt, Stift-∅ 2,5mm
- 3x Pfostenstecker, 10-polig, für Flachbandkabel, zweireihig, Rastermaß 2,54mm
- 3x Pfostenstecker, 20-polig, für Flachbandkabel, zweireihig, Rastermaß 2,54mm
- 1x Pfostenstecker, 26-polig, für Flachbandkabel, zweireihig, Rastermaß 2,54mm
- Flachbandkabel
- 1x Aluminiumgehäuse, 2mm Blechstärke, Größe z.B. 105mm * 105mm * 85mm

13.3.2.1 Stückliste für das Laser-CPU-Board

Die Bauteile, deren Bezeichnung in Klammern gesetzt ist, werden für die USB-Schnittstelle benötigt und aufgrund der beschriebenen Problematik bei der Realisierung nicht mit bestückt.

Bezeichnung im Schaltplan	Bauteil
R1, (R7), (R21), R31, R32	SMD-Widerstand, 10KΩ, 5%, 0,25W, Bauform 1206
(R2)	SMD-Widerstand, 2,2KΩ, 5%, 0,25W, Bauform 1206
R3	SMD-Widerstand, 75Ω, 5%, 0,25W, Bauform 1206
R4	Miniatur-Potentiometer, liegend, 10KΩ, ∅ 6,6mm
R5, R10, R16, R17, R18	SMD-Widerstand, 3,3KΩ, 5%, 0,25W, Bauform 1206
(R6)	SMD-Widerstand, 470Ω, 5%, 0,25W, Bauform 1206
(R8), (R9)	SMD-Widerstand, 27Ω, 5%, 0,25W, Bauform 1206
R11, R12, R13	SMD-Widerstand, 33KΩ, 5%, 0,25W, Bauform 1206
(R19), R28, R29, R30, R33	SMD-Widerstand, 4,7Ω, 5%, 0,25W, Bauform 1206
(R29)	SMD-Widerstand, 1,5KΩ, 5%, 0,25W, Bauform 1206
R25, R26, R27	SMD-Widerstand, 2,4KΩ, 5%, 0,25W, Bauform 1206
C1, C2, C5, C6, C7, C8, (C12), C13, C14, C15, C16, C17, (C18), (C19), C20, (C21)	SMD-Vielschicht-Kondensator, 100nF, 10%, Bauform 1206
C3, C4, (C9), (C10)	Keramik-Kondensator, 27pF, Rastermaß 2,5mm
(C11)	SMD-Vielschicht-Kondensator, 33nF, 10%, Bauform 1206
Q1	Standardquarz, 16,0000MHz ±30ppm, Gehäuse HC49U-S
(Q2)	Standardquarz, 6,0000MHz ±30ppm, Gehäuse HC49U-S
IC1	*ATmega128-16AC*, Gehäuse TQFP-64
(IC2)	*FT232 BL*, Gehäuse LQFP-32
(IC3)	*ST93C56MN*, Gehäuse SO-8
IC4	*74VHC573*, Gehäuse SO-20W
IC5	*628512-55M*, Gehäuse SO-32
IC6	*74VHC14*, Gehäuse SO-14
CON1	SD-Connector *FPS 009-2405-0*
SV1, SV3	Wannenstecker 6-polig, gerade, zweireihig, Rastermaß 2,54mm
SV2, SV4	Wannenstecker 20-polig, gerade, zweireihig, Rastermaß 2,54mm
X1	USB-Buchse, Typ B, liegend, gewinkelt, Printmontage

Tabelle 18: Stückliste für das Laser-CPU-Board

13.3.2.2 Stückliste für das DAC-Board

Bezeichnung im Schaltplan	Bauteil
R1, R6	Metallschicht-Widerstand, 40KΩ, 1%, 0,6W
R2, R7, R13, R22	Metallschicht-Widerstand, 20KΩ, 1%, 0,6W
R3, R8, R12, R14, R15, R16, R21, R23, R24, R25, R30, R31, R32, R33, R36, R37	Metallschicht-Widerstand, 10KΩ, 1%, 0,6W
R4, R9	Metallschicht-Widerstand, 5KΩ, 1%, 0,6W
R5, R10	Metallschicht-Widerstand, 1,33KΩ, 1%, 0,6W
R17, R19	Metallschicht-Widerstand, 68Ω, 1%, 0,6W
R18, R20	Metallschicht-Widerstand, 200Ω, 1%, 0,6W
R26, R27, R28, R29, R34, R35, R38, R39, R40, R42, R43, R44, R45, R46, R47, R48, R49, R50, R51, R52, R53	Metallschicht-Widerstand, 75Ω, 1%, 0,6W
C1, C2, C3, C4, C5, C6, C8, C9, C10, C11, C12, C13, C16, C17, C21, C22	Folienkondensator, 100nF, 20%, Rastermaß 2,5mm
C14	Elektrolytkondensator, 10µF, 16V, 20%, Rastermaß 2,5mm
C19, C20	Keramik-Kondensator, 33pF, Rastermaß 2,5mm
IC1, IC2	*AD7545AKN*, Gehäuse DIP-20
IC3, IC8	*74HC574*, Gehäuse DIP-20
IC4, IC5, IC7, IC9	*TL074*, Gehäuse DIP-14
IC6	*78L05*, Gehäuse TO-92
IC10	*TL072*, Gehäuse DIP-8
SV1	Wannenstecker 20-polig, gerade, zweireihig, Rastermaß 2,54mm
SV2	Wannenstecker 10-polig, gerade, zweireihig, Rastermaß 2,54mm
SV3	Wannenstecker 26-polig, gerade, zweireihig, Rastermaß 2,54mm

Tabelle 19: Stückliste für das DAC-Board

13.3.2.3 Stückliste für die Spannungsversorgung

Bezeichnung im Schaltplan	Bauteil
C1	Elektrolytkondensator, 220µF, 25V, 20%, Rastermaß 3,5mm
C2, C3	Elektrolytkondensator, 330µF, 16V, 20%, Rastermaß 3,5mm
C4, C5, C6, C7	Folienkondensator, 100nF, 20%, Rastermaß 2,5mm
D1	*1N4001*
IC1	*7805*, Gehäuse TO-220
IC2	*LM3940 IT3,3*, Gehäuse TO-220
DC1	DC/DC-Wandler *AM2D-0515DZ*
SL1	Stiftleiste, 2-polig, gerade, einreihig, Rastermaß 2,54mm
SV1	Wannenstecker 10-polig, gerade, zweireihig, Rastermaß 2,54mm

Tabelle 20: Stückliste für die Spannungsversorgung

13.4 Quellcode und Inhalt der CD-ROM

Der Quellcode umfasst insgesamt 3661 Programmzeilen. Würden diese hier abgedruckt, wäre die Diplomarbeit um ca. 70 Seiten länger. Aus diesem Grunde befindet sich der komplette Quellcode auf der dieser Diplomarbeit beigelegten CD-ROM.

Inhalt der CD-ROM:

- Quellcode
  - DAC.c
  - eeprom.c
  - farbpalette.h
  - fat.c
  - interrupt.c
  - konst.h
  - LCD.c
  - main.c
  - menue.c
  - sd.c
  - sd.h
  - timer16.c
  - warten.c
  - zeichensatz.c
- Diplomarbeit.pdf
- dreieck.ild
- kreis.ild
- quadrat.ild

Die CD-ROM befindet sich auf der letzten Seite dieser Diplomarbeit (Seite 122).

13.5 Fotografien ausgewählter Lasergrafiken

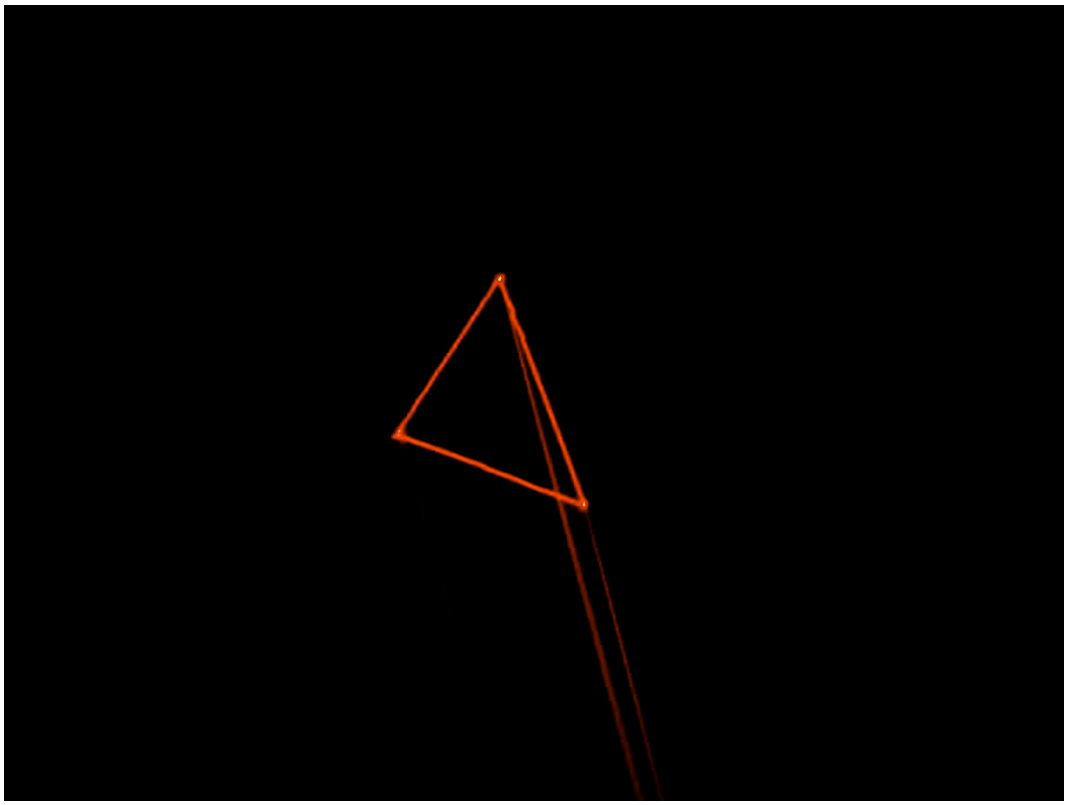


Abbildung 72: Dreieck (Sicht vom Laser zur Leinwand)

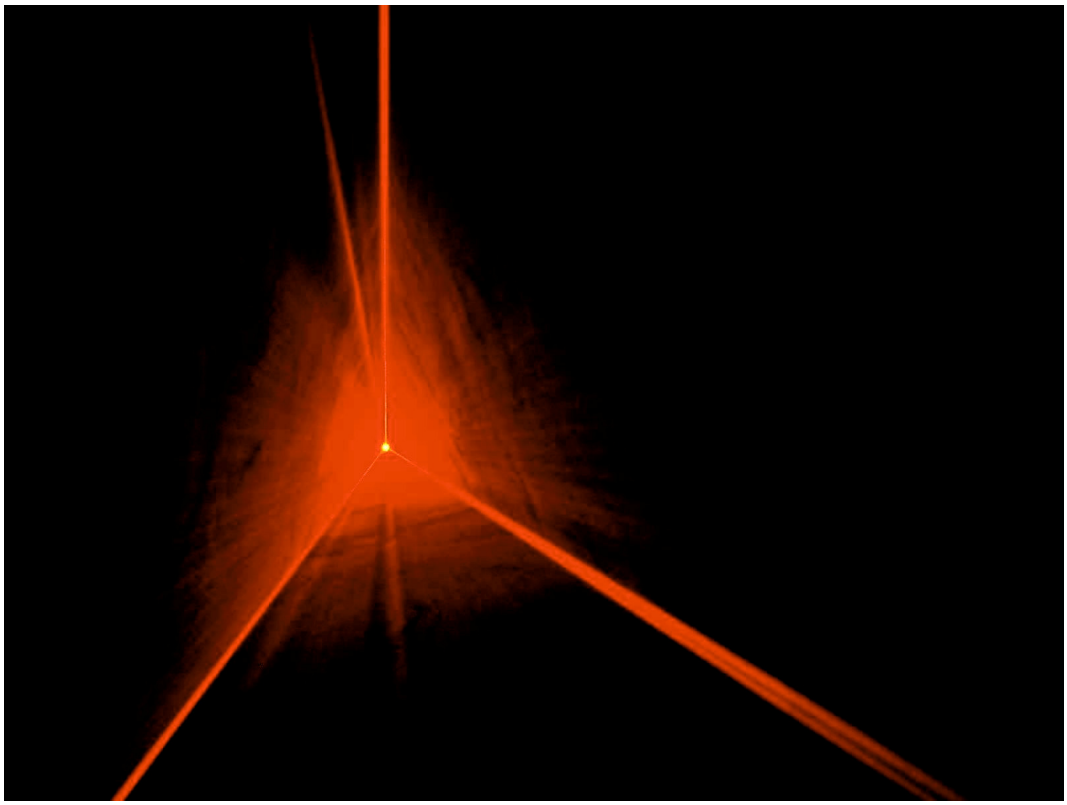


Abbildung 73: Dreieck (Sicht von der Leinwand zum Laser)

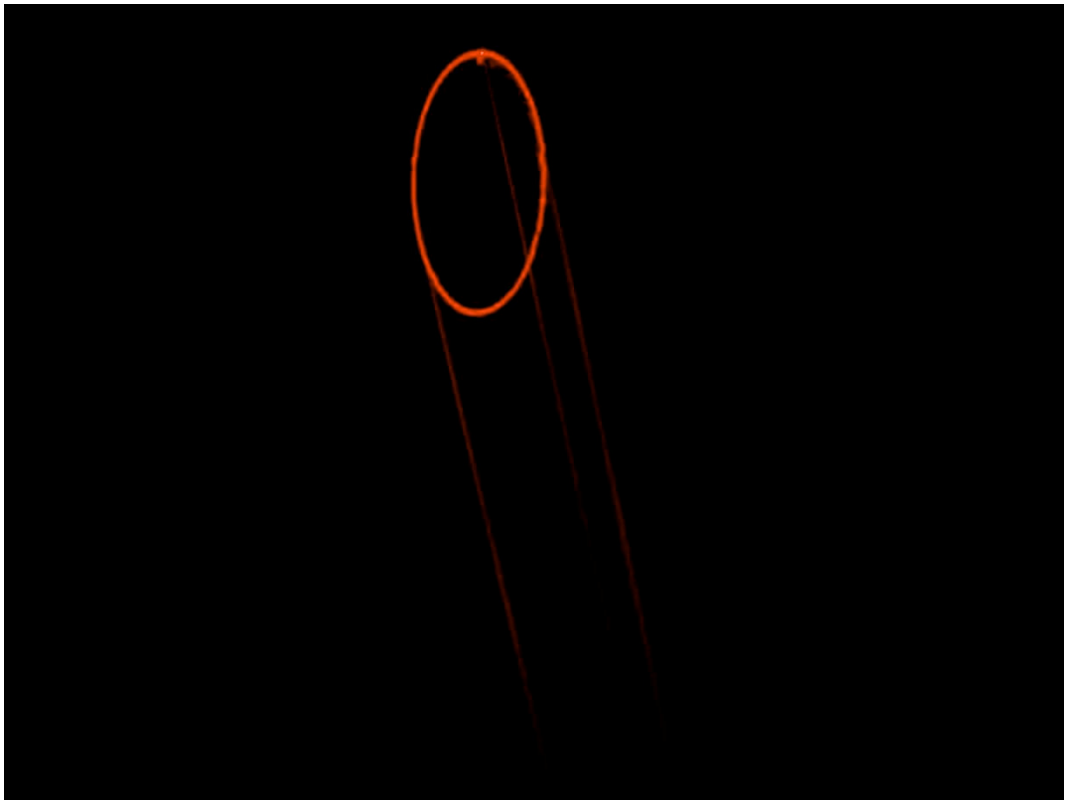


Abbildung 74: Oval (Sicht vom Laser zur Leinwand)

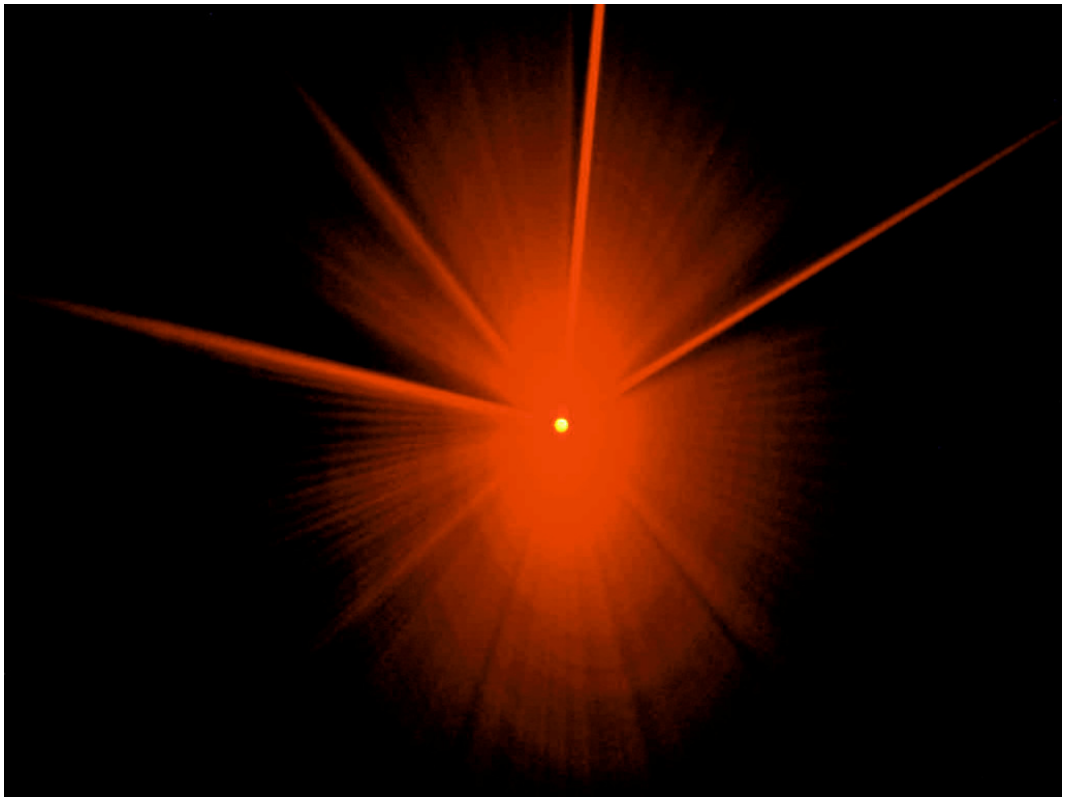


Abbildung 75: Oval (Sicht von der Leinwand zum Laser)

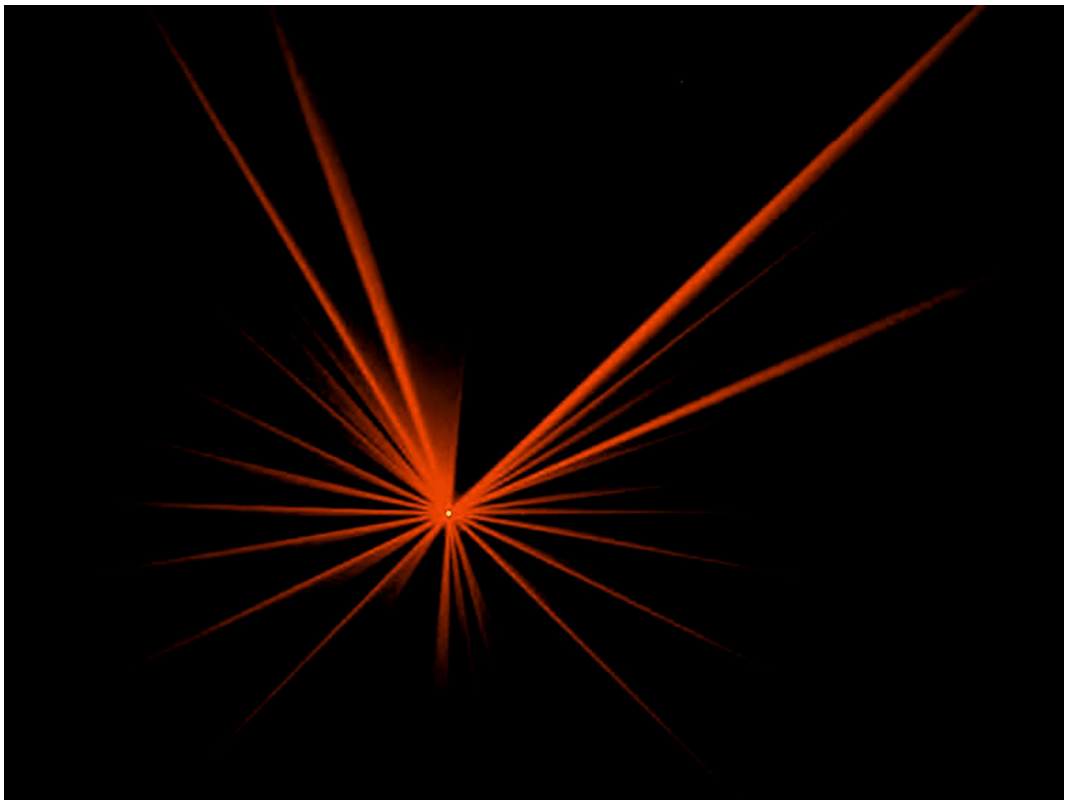


Abbildung 76: Strahlenmuster (Sicht von der Leinwand zum Laser)

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, den 07.12.2007

(Christoph Klaukin)