Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

# DIPLOMARBEIT

# Entwicklung einer universell einsetzbaren Steuereinheit für Lasermaterialbearbeitungsplätze

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs

am Institut für
Industrielle Elektronik und Materialwissenschaften
der Technischen Universität Wien

Betreuer: Dipl.-Ing. Dr.techn. L. Musiejovsky
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. J. Nicolics

Vorstand: Univ.Prof. Dr. M. Vellekoop

eingereicht von
Klaus Steinhammer
3292 Gaming, Birkenweg 4
Matr. Nr.: 9325037

Gaming, im Mai 2003 .....

Im Rahmen dieser Arbeit wurde kein kommerzielles Interesse verfolgt. Die für das vorliegende Ziel entwickelten Programme werden daher unter der **GNU General Public License** bzw. der **GNU Library General Public License** veröffentlicht.

Die in diesem Dokument erwähnten Software- und Hardware-Bezeichnungen sind in den meisten Fällen auch eingetragene Warenzeichen und unterliegen als solche den gesetzlichen Bestimmungen.

Dieses Dokument wurde mit LaTeX $2_\varepsilon$ gesetzt.

Diese Arbeit ist meiner Familie gewidmet,
die mir dieses Studium erst ermöglichte.

# Vorwort

Sie halten nun ein Schriftstück in der Hand, das zwar in kurzer Zeit zu Papier gebracht wurde, dessen technischer Hintergrund jedoch über mehrere Jahre gereift ist. Die in dieser Arbeit vorgestellte Steuereinheit ist als offenes System konzipiert, und darum finden sich Anwendungsmöglichkeiten, die wir bei der Entwicklung nicht bedacht haben und die auch von der hier dokumentierten Arbeit profitieren können. Aus diesem Grund und mit der Hoffnung dass sich jemand findet, der die Weiterentwicklung dieses Systems übernimmt und vorantreibt, wurde sie unter der GNU-GPL veröffentlicht. Ich würde mich freuen, wenn ich in einiger Zeit irgendwo einen Materialbearbeitungsplatz vorfinde, bei dessen Steuereinheit einige der Ideen, die hier vorgestellt werden, implementiert sind.

Ohne die Hilfe und die Unterstützung vieler Personen wäre diese Arbeit mit ziemlicher Sicherheit nicht zustande gekommen. Darum möchte ich mich hier bedanken für die Unterstützung, die ich empfangen habe. Ganz besonders möchte ich mich bei meinen Teamkollegen Stefan Großberger bedanken, der mit mir dieses Projekt bewältigt hat. Weiterer Dank gebührt unseren Betreuern Dipl.-Ing. Dr.techn. Laszlo Musiejovsky und Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Johann Nicolics, die immer mit Rat und Tat beiseite standen. Auch bei den vielen freiwilligen Korrekturlesern, die teilweise diese Arbeit während ihrer Osterfeiertage überarbeitet haben, möchte ich mich herzlich bedanken.

Ganz besonders möchte ich mich bei meiner Familie bedanken.

Klaus Steinhammer,

Gaming, am 18. Mai 2003

It's kind of fun to do the impossible. – Walt Disney

# Inhaltsverzeichnis

**Inhaltsverzeichnis** **i**

**1 Einleitung** **1**

**2 Prinzipielle Funktionsweise eines Lasers** **4**

2.1 Verstärkung einer elektromagnetischen Welle . . . . . 4

2.2 Aufbau eines optischen Resonators . . . . . 6

2.3 Güteschalter oder Q-Switch . . . . . 8

2.4 $CO_2$-Laser . . . . . 9

2.5 Nd:YAG-Laser . . . . . 10

2.6 Halbleiterlaser . . . . . 10

**3 Materialbearbeitung mittels Laserlichts** **11**

3.1 Bohren . . . . . 13

3.2 Schneiden . . . . . 14

3.3 Beschriften . . . . . 15

3.4 Zusammenfassende Bemerkungen . . . . . 17

**4 Echtzeit-Systeme** **18**

4.1 Vordergrund-/Hintergrundsysteme . . . . . 18

4.2 Multitasking-Systeme . . . . . 19

4.2.1 Task . . . . . 20

4.2.2 Scheduler . . . . . 21

4.3 Non-Preemtive-Kernel . . . . . 21

4.4 Preemtive-Kernel . . . . . 23

4.4.1 Reentrant-Funktionen . . . . . 24

4.4.2 Mutal Exclusion . . . . . 24

4.4.3 Semaphore . . . . . 26
4.4.4 Message Queue . . . . . 27
4.4.5 Clock Tick . . . . . 27
4.5 Zusammenfassende Bemerkungen . . . . . 28

**5 Betriebssystem des Steuerrechners 29**
5.1 Linux . . . . . 29
5.2 Realtime-Linux . . . . . 29
5.2.1 Struktur von Realtime-Linux . . . . . 30
5.2.2 Der Modulare Aufbau von Realtime-Linux . . . . . 30
5.2.3 Applikationsinterface . . . . . 31
5.2.4 Beispiel einer Applikation unter Realtime-Linux . . . . 34

**6 Entwicklung der Hard- und Softwarekomponenten 40**
6.1 IO-Karte . . . . . 41
6.1.1 Funktionen der Karte . . . . . 41
6.1.2 Der Schaltplan . . . . . 46
6.2 Handsteuerbox . . . . . 58
6.2.1 Funktionen der Handsteuerbox . . . . . 58
6.2.2 Schaltplan . . . . . 58
6.3 Low-Level-Treiber als “Laser-Server” . . . . . 64
6.3.1 Q-Switch-Task . . . . . 64
6.3.2 Guard-Task . . . . . 65
6.3.3 Feeder-Task . . . . . 65
6.3.4 $I^2C$-Task . . . . . 66
6.3.5 Galvo-Task . . . . . 66
6.3.6 Adjust-Task . . . . . 67
6.3.7 Die FIFO-Handler . . . . . 68
6.3.8 Aufteilung des Sourcecodes . . . . . 69
6.4 High-Level-Treiber als “Laser-Library” . . . . . 70

**7 Entwickelte Anwenderprogramme 75**
7.1 Das HLL-Datenformat . . . . . 75
7.2 Erstellen einer HLL-Datei . . . . . 78
7.3 Abarbeiten einer HLL-Datei . . . . . 80

**8 Zusammenfassung** **81**

**Literaturverzeichnis** **83**

**A Hardware** **I**

A.1 IO-Karte . . . . . I

A.1.1 Schaltplan . . . . . II

A.1.2 Bestückungsplan . . . . . III

A.1.3 Liste der verwendeten Bauteile . . . . . IV

A.1.4 Layouts . . . . . VI

A.2 Handsteuerbox . . . . . VIII

A.2.1 Schaltplan . . . . . IX

A.2.2 Bestückungsplan . . . . . X

A.2.3 Liste der verwendeten Bauteile . . . . . XI

A.2.4 Layouts . . . . . XII

**B EPLD Sourcecode** **XIV**

B.1 IO-Karte . . . . . XIV

B.1.1 Top Level Design . . . . . XV

B.1.2 clkgen.gdf . . . . . XVI

B.1.3 keyboard.gdf . . . . . XVII

B.1.4 CHIPSEL.tdf . . . . . XVIII

B.1.5 transmit_byte.tdf . . . . . XVIII

B.1.6 transmit_bit.tdf . . . . . XXII

B.1.7 keycontrol.tdf . . . . . XXIII

B.2 Handsteuergerät . . . . . XXVI

B.2.1 Top Level Design . . . . . XXVI

B.2.2 display.tdf . . . . . XXVII

B.2.3 display_out.tdf . . . . . XXXIII

B.2.4 display_busy.tdf . . . . . XXXIV

B.2.5 i2c.tdf . . . . . XXXV

B.2.6 i2c_start_stop.tdf . . . . . XXXVI

B.2.7 i2c_bit_pos.tdf . . . . . XXXVII

B.2.8 i2c_transmit_byte.tdf . . . . . XXXVII

**C Bilddokumentation** **XLI**

C.1 Materialbearbeitung . . . . . XLI

C.2 Hardware . . . . . XLVII

**D GNU General Public License** **XLIX**

D.1 Preamble . . . . . XLIX

D.2 GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION . . . . . L

D.3 NO WARRANTY . . . . . LIV

D.4 How to Apply These Terms to Your New Programs . . . . . LV

# Kapitel 1

# Einleitung

Die Mikromaterialbearbeitung mittels Laser ist heute aus einer Vielzahl von Anwendungsmöglichkeiten nicht mehr wegzudenken. Die vielen Vorteile der Lasermaterialbearbeitung, wie die hohe Bearbeitungsgeschwindigkeit, die geringe thermische Belastung oder die erreichbare Genauigkeit, um nur einige zu nennen, übertreffen in einigen Bereichen die Vorteile mancher konventionellen Materialbearbeitungsmethoden.

Ziel dieser Diplomarbeit ist es, für die Lasermaterialbearbeitung eine Steuereinheit zu entwickeln, die universell einsetzbar ist und eine Vielzahl von Bearbeitungsmöglichkeiten bietet. Es werden Steuerfunktionen realisiert, die auf dem Gebiet der Lasermaterialbearbeitung in dieser Form bisher nicht verfügbar waren.

Zum Beispiel wird eine "Laserdrucker"-Funktion implementiert, die einen Postscript-Netzwerkdrucker simuliert und so eine Dateneingabe auf einfachstem Weg, dem der Druck-Funktion, aus fast jeder Software unter den heute gebräuchlichen Betriebssystemen zulässt. Diese vereinheitlichte Form der Datenübergabe steht im Gegensatz zu den vielen verschiedenen Datenformaten, die kommerzielle Systeme zum Datenaustausch verwenden.

Wichtig ist weiters die einfache Übertragbarkeit des Steuersystems auf andere, am Institut vorhandenen Lasermaterialbearbeitungsplätze mit ähnlichem Aufbau. Dem Anwender werden so unterschiedliche Systeme mit einer gemeinsamen Benutzerumgebung zur Verfügung gestellt, die er ohne große Einarbeitungszeit benutzen kann.

Die im Rahmen dieser Diplomarbeit entwickelte Steuereinheit wurde bis heute auf insgesamt drei verschiedenen Strahlablenkungseinheiten erfolgreich eingesetzt, wobei jede auf einem eigenen Ansteuerungsprinzip beruht. Sie wurde für einen Lasermaterialbearbeitungsplatz verwendet, der eine starre Strahlführung aufweist und das Werkstück mittels Koordinatentisch positioniert, der mit Schrittmotoren betrieben wird. Der Vorteil dieser Methode liegt in der Möglichkeit, bei der Bearbeitung Schutzgas einzusetzen. Derselbe Bearbeitungsplatz kann auf ein Spiegelablenksystem umgerüstet werden, um

so eine höhere Bearbeitungsgeschwindigkeit zu erzielen. Die Ansteuerung dieses Spiegelablenkungssystems erfolgt durch eine mitgelieferte PC-Steckkarte. Ein weiterer Lasermaterialbearbeitungsplatz ist ebenfalls mit einem Spiegelablenksystem ausgerüstet, das jedoch zur Ansteuerung einen parallelen 16-Bit-Bus mit Strobe-Signalen benötigt.

Das Umrüsten der Steuereinheit für deren Einsatz in unterschiedlichen Lasermaterialbearbeitungsplätzen beschränkt sich, neben der Änderung des Hardwareaufbaus, auf die Anpassung des Low-Level-Treibers.

Die grafische Benutzerschnittstelle ist einfach und funktionell aufgebaut, die Dateneingabe erfolgt mittels einer ASCII-Datei. Diese kann von Hand, mittels eines Vektorzeichenprogramms oder durch einen Netzwerkdrucker erstellt werden. Die beim Druck fehlenden Materialdaten werden nachträglich eingetragen oder aus einer selbst aufgebauten Bibliothek vor der Abarbeitung ausgelesen.

Eine Handsteuerbox mit Anzeige ermöglicht eine örtliche Trennung zwischen Bildschirmplatz und der Bearbeitungsmaschine. Damit kann der Programmiervorgang von der bei der Bearbeitung auftretenden Lärm- und Schmutzbelastung räumlich getrennt durchgeführt werden.

Die Steuerung des gesamten Systems wird von einem handelsüblichen Personal-Computer übernommen. Um den externen Hardwareaufwand so gering wie möglich zu halten, werden die im PC vorhandenen Schnittstellen genutzt. Dies erleichtert die Portierung auf andere Lasermaterialbearbeitungsplätze, da auf standardisierte Schnittstellen aufgesetzt werden kann und nicht immer eine auf das Lasersystem abgestimmte Hardware konstruiert werden muss.

Die Funktionen dieser sonst zusätzlichen Hardware werden durch Software realisiert. Um schnell genug auf auftretende Ereignisse mit einem Personal-Computer reagieren zu können, ist der Einsatz eines echtzeitfähigen Betriebssystems erforderlich. Dieses garantiert im Gegensatz zu einem normalen Betriebssystem die Einhaltung gewisser zeitlicher Vorgaben, unabhängig von der Auslastung des Prozessors. Es wird hier Linux mit der Echtzeiterweiterung RT-Linux verwendet, weil dieses frei erhältlich ist und alle unsere Anforderungen erfüllt.

Einen groben Überblick über das erarbeitete System gibt die Abbildung 1.1. Im Zentrum steht hier der Steuerrechner, der die Laserleistung über das Netzteil der Pumpquelle steuert, die Strahlimpulse über den Q-Switch vorgibt und die Geometriedaten an die Strahlablenkung ausgibt. Weiters ist der Steuerrechner an die Netzwerkumgebung angeschlossen um die anderen Rechnersysteme durch die Netzwerkdruckerfunktion in den Produktionszyklus einzubinden.

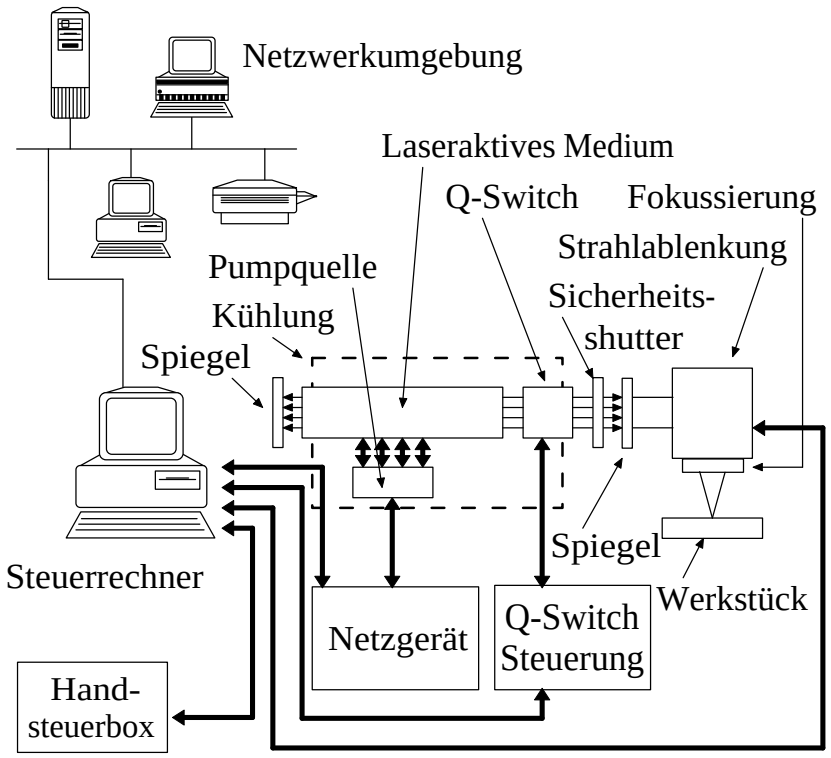



Abbildung 1.1: Aufbau des Lasersystems

Das gesamte Projekt setzt sich aus mehreren Teilen zusammen. Die vorliegende Diplomarbeit behandelt den Hardwareaufbau am Beispiel der vorhandenen Laserquelle, die Ansteuerung der Hardware, bis hin zu einer Softwareschnittstelle, auf die die Anwenderprogramme aufsetzen. Die Anwenderprogramme selbst, welche die Bahnsteuerung und Datenaufbereitung enthalten, werden hier nur einführend erwähnt. Sie werden in einer Arbeit meines Teamkollegen Stefan Großberger [9] beschrieben.

# Kapitel 2

# Prinzipielle Funktionsweise eines Lasers

Das Wort Laser bedeutet "Lichtverstärkung durch stimmulierte Aussendung von Strahlung". Die prinzipielle Realisierung dieses physikalischen Grundprinzips soll im Folgenden kurz erklärt werden.

## 2.1 Verstärkung einer elektromagnetischen Welle

Betrachtet man den Aufbau eines Atoms, wird man feststellen, dass verschiedene energetische Anregungszustände existieren. Beim einfachsten Fall, der des Wasserstoffatoms, kann das Elektron nicht nur den Grundzustand, sondern auch eine Reihe von angeregten Zuständen einnehmen. Diese sind dadurch charakterisiert, dass das Elektron eine Umlaufbahn einnimmt, die weiter vom Kern entfernt liegt. In diesem Zustand besitzt es eine höhere potentielle Energie als im Grundzustand. Da jedes System immer den Zustand geringster innerer Energie anstrebt, wird das Elektron nach einer kurzer Zeit, der Lebensdauer $t$ des Zustandes, wieder den ursprünglichen Orbit um den Kern einnehmen.

Trifft eine elektromagnetische Welle mit einer gewissen Energiedichte und Frequenz auf das Material, wird dieser Welle Energie entnommen. Diese Energie wird dafür aufgewendet, Teile des Materials in einen höheren Energiezustand zu versetzten. Diesen Vorgang nennt man Absorption (siehe Abbildung 2.1). Beim umgekehrten Vorgang, der Emission, kann zweierlei passieren: Bei der stimulierten Emission durchläuft eine elektromagnetische Welle mit bestimmter Frequenz das Material, die den Übergang zum niedrigeren Energieniveau einleitet. Die Energie im Material wird abgebaut und der Welle zugeführt, was bedeutet, dass die Energiedichte der Welle steigt. Bei der spontanen Emission hingegen trifft keine Welle innerhalb der Lebensdauer $t$

auf die Materie. Der Übergang zum niedrigeren Energieniveau wird spontan eingeleitet, und die Energiedifferenz wird als ungerichtete elektromagnetische Welle abgegeben. Dieser Prozess ist rein statistischer Natur.

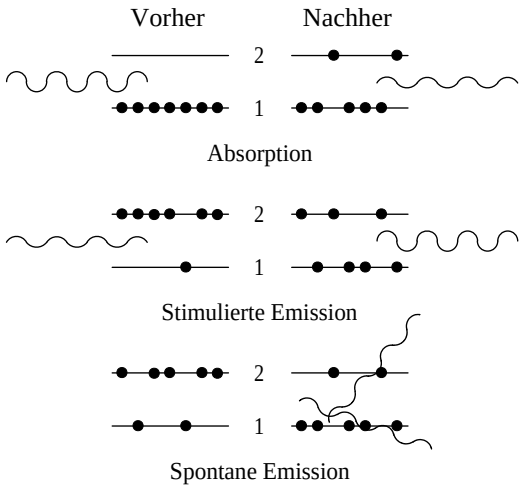



Abbildung 2.1: Wechselwirkung einer Welle mit dem laseraktiven Material

Für die Lasertechnik ist die Absorption und die stimulierte Emission von vorrangigem Interesse. Da diese Prozesse gleichzeitig ablaufen und man im Endeffekt eine Verstärkung der Welle erreichen will, ist es erforderlich, das obere Energieniveau stärker besetzt zu halten als das untere. Dies ist bei einem 2-Niveausystem nicht zu erreichen, da bei steigender Besetzung des höheren Niveaus immer weniger Energie absorbiert wird, bis bei einer Gleichbesetzung der Energiezustände das Material für diese Wellenlänge transparent ist.

Abhilfe schafft hier ein 3-Niveau-System (Abbildung 2.2).

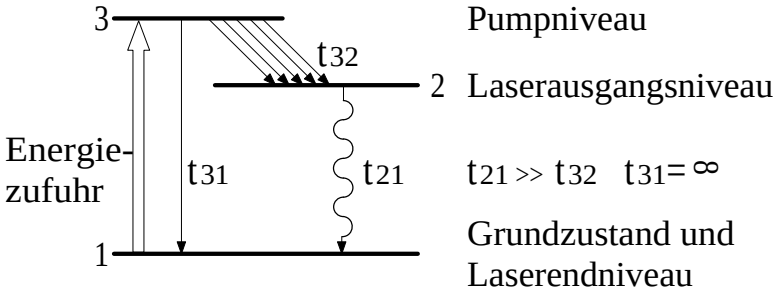



Abbildung 2.2: 3-Niveau-System

Material, das sich im Zustand 1 befindet, geht aufgrund der Anregung in den Zustand 3 über. Die Lebensdauer $t_{32}$ soll nun im Verhältnis zu $t_{21}$ sehr kurz und im wesentlichen durch Übergänge in den Zustand 2 charakterisiert sein. Dann wird Zustand 3 nur eine geringe Besetzungsdichte aufweisen. Da die Lebensdauer $t_{21}$ sehr groß sein soll, wird es bei hoher Energiezufuhr von außen

(pumpen) gelingen, den Zustand 2 mit dem Umweg über den Zwischenzustand 3 relativ zu 1 überzubesetzen. Allerdings muss bei einem 3-Niveaulaser immerhin noch mehr als 50% der Materie in einen angeregten Zustand versetzt werden.

Bei einem 4-Niveau-System ist auch dies nicht mehr nötig (Abbildung 2.3).

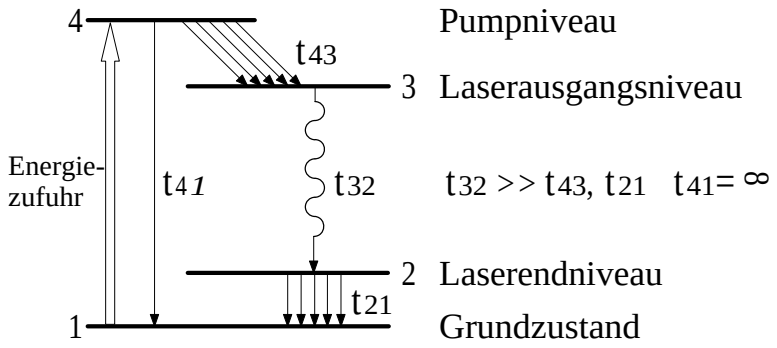



Abbildung 2.3: 4-Niveau-System

Es wird hier ein weiteres Niveau eingeführt, welches sich nahe über dem Grundniveau befindet und dessen Lebensdauer ebenfalls sehr kurz sein soll. Dann sammelt sich wiederum die angeregte Materie in Zustand 3, welcher aber, bedingt durch die kurze Lebensdauer des Zustandes 4, leicht eine relative Überbesetzung gegenüber Zustand 4 erreicht.

Eine detailliertere Betrachtung dieses Themas findet man in [1].

## 2.2 Aufbau eines optischen Resonators

Wird nun Material mit dieser Eigenschaft innerhalb eines optischen Resonators platziert, entsteht ein Laser. Der optische Resonator besteht im allgemeinen aus zwei hochreflektierenden Spiegeln. Diese stehen so zueinander, dass das zwischen ihnen reflektierte Licht einen möglichst langen Weg im laseraktiven Medium zurücklegt. Auf diese Weise wird die Verstärkungseigenschaft des Materials optimal ausgenutzt. Die Energie wird dem Resonator über eine Pumpquelle zugeführt. Diese ist auf das laseraktiven Medium abgestimmt, Festkörperlaser werden von Lichtquellen, Gaslaser über Hochspannungsentladungen gepumpt.

Stehende Wellen können sich innerhalb dieses Resonators nur dann ausbilden, wenn die Resonatorlänge gleich einem ganzzahligen Vielfachen der halben Wellenlänge $\lambda$ ist ($L = q * \frac{\lambda}{2n}$, dabei sei n der Brechungsindex des laseraktiven Materials). Man nennt diese stehenden Wellen die longitudinalen Eigenschwingungen oder Moden des optischen Resonators und bezeichnet sie mit der Modenzahl q (siehe Abbildung 2.4).

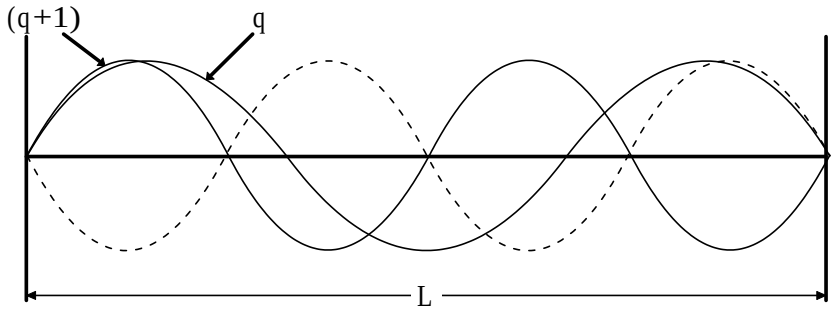



Abbildung 2.4: Modenbildung im Resonator

Um die Verluste im Resonator gering zu halten, wird meist ein oder beide Spiegel als Hohlspiegel ausgeführt. Dabei muss sich der Krümmungsmittelpunkt oder der Spiegel selbst zwischen dem anderen Spiegel und seinem Krümmungsmittelpunkt befinden.

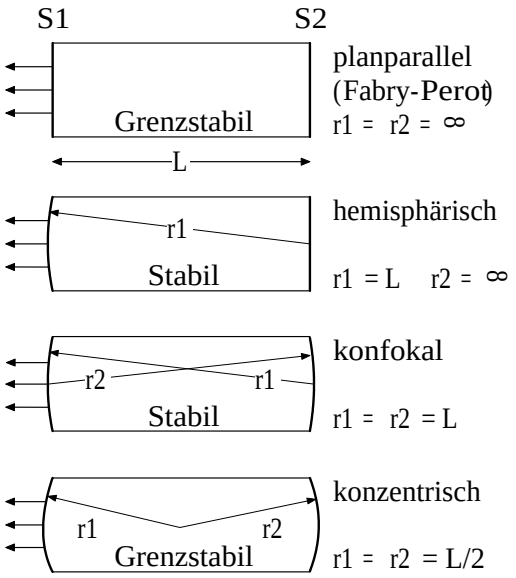



Abbildung 2.5: einige Spiegelanordnungen am Resonator

Abbildung 2.5 zeigt die beiden gerade noch stabilen Anordnungen, die planparallele Fabry-Perot-Anordung und die konzentrische Anordnung und die besonders stabil und leicht zu justierende konfokale und hemisphärische Anordnung.

Um Laserlicht aus dieser Anordnung auskoppeln zu können, wird einer der beiden Spiegel geringfügig durchlässig ausgeführt.

## 2.3 Güteschalter oder Q-Switch

Eine ganze Reihe von Lasern lässt sich nur gepulst betreiben, weil das laseraktive Medium die im CW Betrieb[1] auftretenden Temperaturen nicht schnell genug an die Umgebung abführen kann. Weiters gibt es einige Anwendungen, bei denen es notwendig ist, den Laser in einer Impulsbetriebsart zu betreiben. Dies können zum Beispiel ultrakurze Impulse für die Nachrichtenübertragung oder aber hochenergetische Impulse zur Materialbearbeitung sein.

Eine ganze Reihe von Verfahren zur Erzeugung kurzer Laserimpulse beruht auf der gezielten Beeinflussung der Resonatorverluste, auch Güte Q (von quality) des Resonators genannt. Dabei werden die Verluste im Resonator künstlich auf einem hohen Niveau gehalten. Da sich das Medium aufladen kann ohne Energie abzugeben, wird sich eine Überbesetzung aufbauen, die wesentlich höher ist als im CW Betrieb. Nach einiger Zeit des Aufladens wird dann die Resonatorgüte wieder sprunghaft erhöht, und die gespeicherte Energie wird in Form eines Laserimpulses abgegeben. Der Vorteil dieser Methode, abgesehen von ihrer Einfachheit, ist, dass die Impulsstärke und der Impulszeitpunkt relativ einfach über Energiezufuhr beziehungsweise Zeitpunkt der Resonatorgüteveränderung zu beeinflussen sind. Eine Vorrichtung, die die Resonatorgüte beeinflusst, nennt man einen Q-Switch.

Die Anzahl der verschiedenen Methoden um die Resonatorgüte zu beeinflussen ist groß. Man kann unter passiven, aktiven mechanischen und aktiven optischen Methoden unterscheiden.

Eine passive Methode nutzt die Eigenschaften sättigbarer Absorber (Farbstoffe). Diese absorbieren im Grundzustand den Laserstrahl und verhindern somit das Anschwingen des Resonators. Ab der Absorption einer gewissen Energiemenge wird der Absorber plötzlich transparent, und die inzwischen aufgebaute Überbesetzung im laseraktiven Medium setzt einen Laserimpuls frei. Der große Vorteil dieser Methode liegt in der einfachen Handhabung und dem relativ günstigen Anschaffungspreis. Ein Nachteil jedoch ist, dass ein gewisser Zeit- und Amplitudenjitter auftritt. Auch die genaue Auslösezeit ist nicht beeinflussbar.

Eine ebenfalls einfache Methode ist es, einen der zwei Spiegel des Resonators um eine zum Lichtweg senkrecht stehende Achse rotieren zu lassen. Auch hier kann sich nur innerhalb einer kurzen Zeit, dann, wenn die Spiegel parallel stehen, ein Laserpuls ausbilden (siehe Abbildung 2.6). Die Energiezufuhr kann hier mit dem Drehwinkel des Spiegels synchronisiert werden, um dem laseraktiven Medium Zeit zum Abkühlen zu geben. Der Nachteil dieser Methode ist, dass aufgrund der Massenträgheit des rotierenden Spiegels die Impulsfrequenz nicht sprungartig geändert werden kann.

Die aktiven optischen Methoden beruhen auf elektro-optischen, magneto-optischen oder akusto-optischen Effekten. Aufgrund dieser Effekte lassen sich

[1] CW bedeutet continous wave oder Dauerstrich-Betrieb

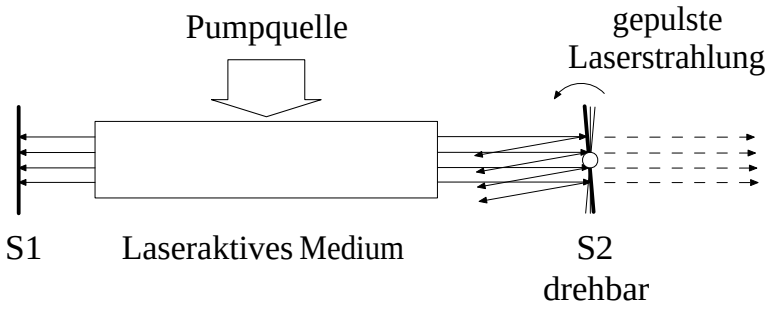



Abbildung 2.6: Q-Switch mit Drehspiegel

die optischen Eigenschaften des Q-Switch Materials beeinflussen. Meist wird beim Anlegen eines Signals die Gitterstruktur in der Art verändert, dass die normalerweise ungehindert durchlaufenden Wellen am Gitter gebeugt werden und somit den Resonator verlassen. Die Resonatorgüte sinkt dadurch. Da dies mit sehr hoher Frequenz erfolgen kann, erhält man wesentlich kürzere Impulse als bei mechanischen Methoden. Auch die Synchronisation zum Augenblick maximaler Überbesetzung lässt sich besser durchführen (siehe Abbildung 2.7).

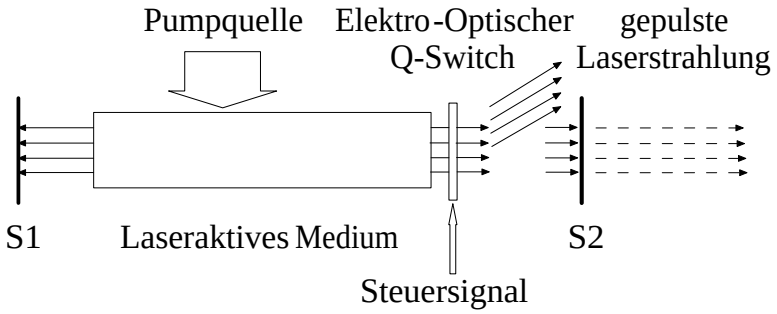



Abbildung 2.7: elektro-optischer Q-Switch

Den Vergleich einer Bearbeitung mit und ohne Q-Switch zeigen die Abbildungen C.1 und C.2 im Anhang C.1.

## 2.4 $CO_2$-Laser

In der Materialbearbeitung wird meist der $CO_2$-Laser eingesetzt. Das laseraktive Medium ist hier ein molekulares Gas. Ein Molekül besitzt zusätzlich zu den energetischen Anregungszuständen auch Schwingungs- und Rotationsanregungszustände, welche Strahlung im mittleren Infrarotbereich (10,6 $\mu$m) emittieren. Da die Energie, die zur Anregung der Schwingungsniveaus notwendig ist, um einiges geringer ist als die der elektronischen Niveaus, weist der $CO_2$-Laser einen wesentlich höheren Wirkungsgrad auf als Systeme mit

anderen laseraktiven Medien. Gepumpt wird der $CO_2$-Laser indirekt mittels Hochspannung. Im elektrischen Feld werden Elektronen beschleunigt, die bei einem Zusammenstoß ihre kinetische Energie an das Gas abgeben.

## 2.5 Nd:YAG-Laser

Weiters zu erwähnen ist der Neodym-Yttrium-Aluminium-Granat-Laser, bekannt unter den Namen Nd:YAG-Laser. Er ist ein Vertreter der Festkörperlaser. Der Vorteil dieses Materials ist, dass es bereits mit normalen Lampen mit vergleichsweise wenig Leistung zur stimmulierten Emission gebracht werden kann. Weiters ist für die Steuerung der Ausgangsleistung keine Aufwendige Schaltungstechnik notwendig. Sie kann mittels des Lampenstroms der Pumpbeleuchtung gesteuert werden. Der Laser kann im CW- oder im Pulsbetrieb betrieben werden und emittiert Laserlicht mit einer Wellenlänge von 1,06 $\mu$m. Der Nachteil dieses Materials liegt in der Schwierigkeit, große Kristalle zu ziehen, welche die gewünschte optische Homogenität aufweisen.

## 2.6 Halbleiterlaser

Halbleiterlaser spielen in der Materialbearbeitung nur eine untergeordnete Rolle, sie werden wegen ihrer geringen Größe und der Möglichkeit, den Laserstrahl mit extrem hohen Frequenzen zu modulieren, in der Nachrichtentechnik eingesetzt.

# Kapitel 3

# Materialbearbeitung mittels Laserlichts

Wird Laserlicht mittels einer Optik auf eine Werkstückoberfläche fokussiert, dann wird dieses dort teilweise absorbiert und es entsteht Wärme. Diese lokal begrenzte, durch die Leistungsdichte des Laserstrahles steuerbare Erwärmung wird als Prozesswärme für verschiedene Bearbeitungsvorgänge benutzt.

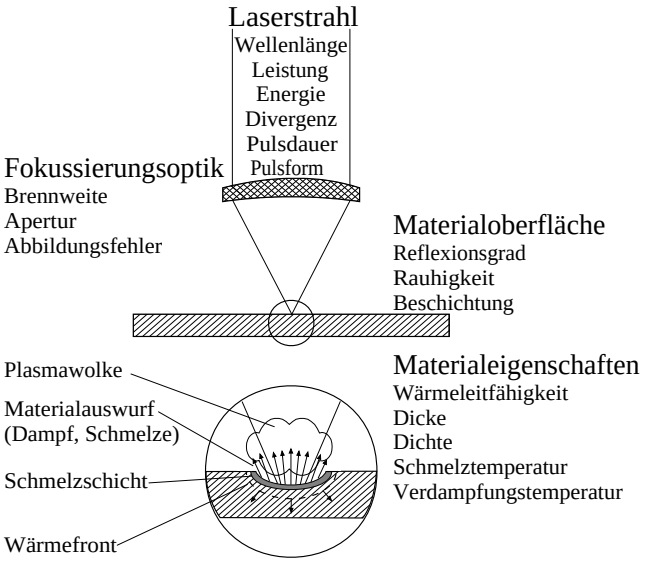



Abbildung 3.1: Wirkung von fokussiertem Laserlicht auf Materialoberflächen

Trifft das Laserlicht nur mit geringer Leistungsdichte auf ein Material, so reicht die Energie nicht aus, um das Material zum Schmelzen zu bringen und es wird lediglich erwärmt. Dies wird für das Transformationshärten von Metallen ausgenutzt. Es lassen sich ausreichend hohe Aufheiz- und Abkühlgeschwindigkeiten realisieren, um z.B. eine Stahloberfläche zu härten. Aufgrund des geringen Fokusdurchmessers kann dies örtlich begrenzt geschehen, ohne

das umgebende Metall in seinen mechanischen Eigenschaften zu beeinflussen (Abbildung 3.2).

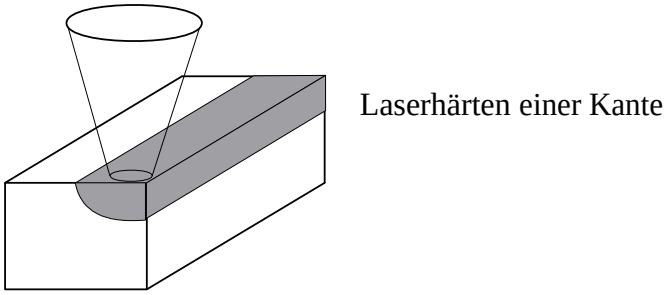



Abbildung 3.2: Härten mittels Laserstrahls

Wird die Leistungsdichte erhöht, so wird an der Oberfläche von Metallen die Schmelztemperatur überschritten. Diesen Tempereaturbereich wird man beim Schweißen, Löten und Legieren finden. Die Schmelztiefe wird dabei primär von der Leistungsdichte bestimmt. Die Einwirkdauer muss dabei so bemessen werden, dass nur wenig oder gar kein Material abgetragen wird (Abbildung 3.3).

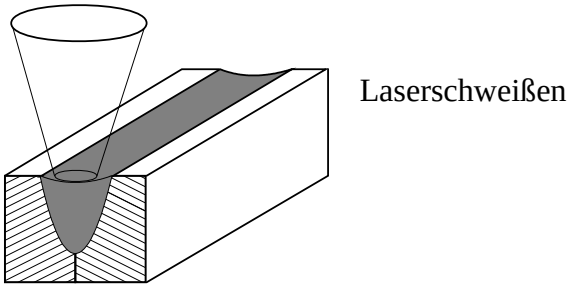



Abbildung 3.3: Schweißen mittels Laserstrahls

Ist die zugeführte Energie zu groß, wird die Schmelze in Dampf übergehen und sich vom Werkstück lösen. Es entsteht ein Loch im Material, es wird gebohrt oder bei einer Relativbewegung zwischen Laserstrahl und Werkstück geschnitten (Abbildung 3.4).

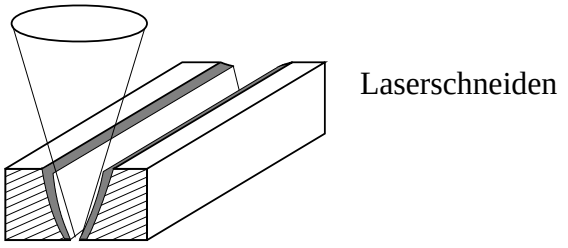



Abbildung 3.4: Schneiden mittels Laserstrahls

Dabei lassen sich vier Bereiche mit unterschiedlichen Wechselwirkungsprozessen unterscheiden:

- Schwellenbereich: Ab einer gewissen Leistungsdichte wird die Siedetemperatur an der Oberfläche erreicht und das Material beginnt abzudampfen. Es bildet sich eine Verdampfungszone, die von Schmelze umgeben ist. Charakteristisch am Schwellenbereich ist eine kontinuierliche Abdampfung des Materials, wobei die Wärmeleitung eine wesentliche Rolle spielt.

- Bereich mit Flüssigkeitsaustreibung: Wird die Intensität weiter erhöht, so führt dies zu einer Überhitzung der Schmelze und flüssiges Material wird explosionsartig aus dem Schmelzebereich herausschleudert. Diese Materialtröpfchen lagern sich in der Nähe des Bearbeitungsbereichs ab, was zu unregelmäßigen Strukturen führt und die Qualität der Bearbeitung beeinträchtigt. Da jedoch nicht mehr das gesamte Material verdampft werden muss, kann die Bearbeitungsgeschwindigkeit bei gleicher Leistungsdichte gesteigert werden.

- Bereich der anormalen Absorption: In diesem Bereich tritt die Plasmawolke in Wechselwirkung mit dem Laserstrahl. Dabei wird die Energie nicht mehr unmittelbahr von der Materialoberfläche, sondern vom Plasma absorbiert und in Wärmeenergie umgeformt, die dann an das Material abgegeben wird. Da das Absorptionsvermögen von Plasma größer ist als das des Materials, steigt die Energieaufnahme sprungartig an, dementsprechend erhöht sich die Verdampfungsrate, die Größe der Schmelzzone reduziert sich, und es wird kein flüssiges Material mehr herausgeschleudert.

- Abschirmbereich: Bei noch höherer Intensität wird die Energie vollständig in der Plasmawolke absorbiert und an das Material über Strahlung und Wärmekontakt abgegeben. Die Ausdehnung der Bearbeitungsstelle hängt jetzt nicht nur von der Fokussierung, sondern auch von der Ausdehnung der Plasmawolke ab, die wesentlich größer als der Fokusbereich sein kann.

Eine genauere Beschreibung dieser Vorgänge findet man in [2].

## 3.1 Bohren

Bohrlöcher mit einem Durchmesser unter einem Millimeter werden meist mittels Laserlichtes hergestellt. Man unterscheidet zwischen:

- Feinstbohrtechnik: Dabei werden Löcher mit einem Durchmesser von 20 $\mu$m bis 250 $\mu$m bis zu einer Tiefe von 1,5mm mit einem einzigen Laserpuls hergestellt (siehe Abbildung C.11).

- Laser-Schlagbohren: Löcher mit 0,1 bis 1mm Durchmesser werden durch eine Serie von Pulsen in das Material bis zu einer Tiefe von 6mm getrieben (Abbildung C.9).
- Laserfräsen: Es wird die Kontur des Loches in das Material geschnitten. Der Vorteil dabei ist, dass das Loch nicht unbedingt Zylindrisch sein muss. Die Gestalt der Grundfläche kann in sehr weiten Grenzen variiert werden und das Loch kann auch trichterförmig ausgeführt werden.

Als Beispiel möchte ich die Herstellung von Drahtziehformen anführen (siehe Abbildung 3.5). Diese werden aus Diamant hergestellt, welcher aufgrund seiner hohen Härte mechanisch schwierig zu bearbeiten ist. Erst wird der Eingangskegel mittels Laserfräsens hergestellt. Dabei wird Material mit einer Spiralbewegung des Laserstrahls abgetragen. Danach wird durch Laser-Schlagbohren die Hauptbohrung erzeugt. Da an die Toleranzen des Ausgangskegels keine so großen Anforderungen gestellt werden, wird dieser mechanisch gebohrt.

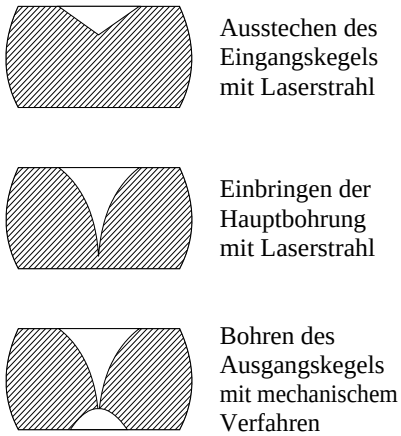



Abbildung 3.5: Herstellung einer Drahtzieform aus Diamant

## 3.2 Schneiden

Laserschneiden wird meist bei Materialien eingesetzt, bei denen eine konventionelle Bearbeitung schwierig durchzuführen ist. Dieser Fall kann zum Beispiel aufgrund der Härte eines Werkstoffes oder durch Vorgabe einer komplexen Schnittführung eintreten. Bei einer Laserstrahlsteuerung, die mit mindestens zwei Achsen arbeitet, kann fast jede Kontur programmiert und abgearbeitet werden. Bei einigen Werkstoffen wie Keramik- oder Siliziumscheiben genügt es, die Oberfläche einzuritzen und anschließend das Werkstück zu brechen (siehe Abbildungen C.10 und C.11). Es gibt auch Vorrichtungen, in denen die Laseroptik in eine Düse integriert wird, durch die kolinear

zum Laserstrahl ein Gas ausströmen kann (Abbildung 3.6). Damit kann man einerseits Schutzgas verwenden, das es ermöglicht, brennbare oder leicht oxydierende Materialien (siehe Abbildung C.8) mit hoher Qualität zu bearbeiten, oder aber ein Reaktionsgas, das bei den Temperaturen im Brennpunkt mit dem zu bearbeitenden Material eine chemische Reaktion eingeht und mit der entstehenden Prozesswärme den Schneidevorgang unterstützt. Weiters wird mit diesem Gas das Schmelzgut im Bearbeitungspunkt ausgeblasen, was wiederum die Leistungsfähigkeit und die erzielbare Schnitttiefe erhöht.

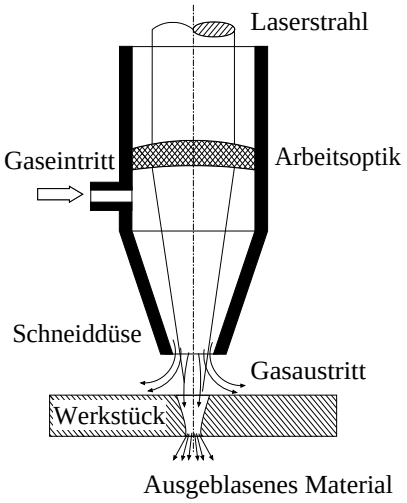



Abbildung 3.6: Laserschneiden mit Schutzgas

Ein weiteres Einsatzgebiet ist das Trimmen von Widerständen in der Dünn- und Dickschichttechnik. Dort werden Widerstände mittels Abscheiden eines Films aus der Gasphase beziehungsweise durch Drucken einer Widerstandspaste auf ein Substrat aufgetragen. Der Widerstandswert wird grob mit den Abmessungen des Wiederstandes festgelegt. Häufig ist jedoch ein Feinabgleich erforderlich, um den gewünschten Widerstandswert hinreichend genau zu erreichen. Dabei wird, während der Widerstandswert gemessen wird, mit dem Laser so viel der Widerstandsschicht entfernt, bis der gewünschte Wert erreicht ist (Abbildung 3.7).

## 3.3 Beschriften

Beim Beschriften wird, ähnlich wie beim Schneiden, Material abgetragen. Es gibt mehrere Verfahren, um Markierungen aufzubringen:

- Lasergravierverfahren: Ähnlich wie bei einem Stiftplotter wandert der Laserstrahl über das Material und hinterlässt dort eine sichtbare Zeichnung. Geführt wird der Strahl dabei über 2 Ablenkspiegel, deren Drehachse um 90 Grad versetz ist und somit eine Ablenkung in der XY

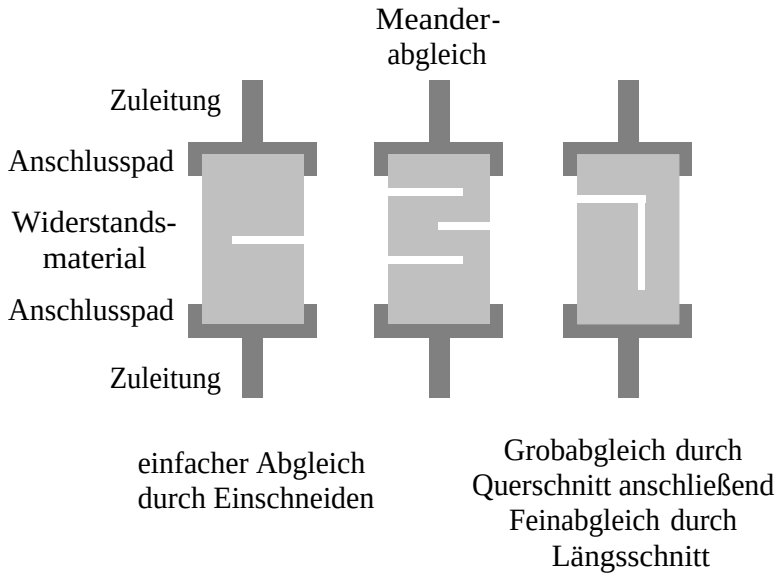



Abbildung 3.7: Widerstandstrimmen mittels Laserstrahls

Ebene zulässt. Ein Beispiel ist im Anhang (Abbildung C.3) zu finden. Es zeigt ein graviertes Gitter im Größenvergleich mit einen Millimetermaßstab.

- Matrixverfahren: Das Matrixverfahren ist ähnlich dem Lasergravierverfahren, jedoch wird der Laser nicht im CW-Modus betrieben, sondern das Schriftbild wird wie bei einem Matrixdrucker aus einzelnen Punkten aufgebaut, die so dicht aneinander liegen, dass das menschliche Auge sie nicht mehr als einzelne Punkte sieht, sondern vielmehr zu einer Fläche mit unterschiedlichen Helligkeitswerten zusammensetzt. Eine Vergrößerung einer so bearbeiteten Fläche ist im Anhang (Abbildung C.4) zu sehen.

- Scanningverfahren: Der Laserstrahl wird von einem rotierenden Spiegel in einer Achse abgelenkt und überstreicht so eine Zeile, wobei er je nach Schriftbild abwechselnd hell- oder dunkelgetastet wird. Für den Vorschub in die nächste Zeile wird entweder das Werkstück bewegt (Beschriftung auf einem Fließband), oder ein zweiter Spiegel sorgt für die nötige Ablenkung des Laserstrahls. In den Abbildungen C.5und C.6 im Anhang wird die Bearbeitung eines Aufklebers nach dem Scanningverfahren illustriert.

- Markierungen: Zum Markieren wird meist nur ein einzelner Punkt mit dem Laser erzeugt. Dieses Verfahren wird meist in der Qualitätssicherung zur Markierung von Ausschussteilen eingesetzt.

Um ein besonders gutes Schriftbild zu erhalten wird meist ein Verbund aus zwei verschiedenfärbigen Materialien bearbeitet oder ein Trägermaterial mit

einer Farbschicht verwendet. Dann wird die obere Schicht so abgetragen, dass die darunterliegende Schicht zum Vorschein kommt und zum Schriftbild beiträgt.

## 3.4 Zusammenfassende Bemerkungen

Die Bearbeitung mittels Laserlichtes wird also durch folgende Eigenschaften charakterisiert:

Aufgrund der gut beherrschbaren Strahlführung des Laserlichts ist die Laserbearbeitung in die Klasse der Präzisionsbearbeitungsverfahren einzuordnen. Die erzielbaren Toleranzen sind klein, die Wiederholgenauigkeit groß.

Eine Besonderheit der Lasermaterialbearbeitung ist die Möglichkeit der dynamische Erzeugung von Geometriedaten während einer Bearbeitung. Damit ist es möglich die Geometrie individuell an das Werkstück anzupassen oder ohne Mehraufwand Beschriftungen mit sich ändernden Text zu erzeugen.

Wegen der geringen thermischen Beeinflussung außerhalb der Bearbeitungszone entsteht kein Verzug des Werkstücks. Dieses muss nicht aufwendig fixiert werden, da keinerlei Bearbeitungskräfte ausgeübt werden. Es ist nicht erforderlich, dass die zu bearbeitende Stelle mechanisch zugänglich ist, es muss lediglich eine Sichtverbindung bestehen. Laser haben zwar hohe Anschaffungskosten, jedoch fallen im Betrieb nur geringe Kosten für Abnutzung und dergleichen an.

Weitere Informationen über das Thema Materialbearbeitung mittels Laserstrahlung finden sich in [2].

# Kapitel 4

# Echtzeit-Systeme

Ein System wird zu einem Echtzeit-System, wenn nicht nur logische, sondern auch zeitliche Vorgaben eingehalten werden müssen, die vorgegebene Aufgabe zu erfüllen.

Man unterscheidet 2 Arten vom Echtzeit-Systemen:

- Hard-Realtime-System: Hier muss eine strikte Einhaltung des zeitlichen Rahmens gewährleistet sein. Jeder Programmabschnitt muss in der festgelegten Zeit abgearbeitet werden, egal welcher Systemlast- oder Ausnahmefall vorliegt.
- Soft-Realtime-System: Bei Soft-Realtime-Systemen wird der vorgegebene zeitliche Rahmen in der Regel eingehalten, in einigen Fällen kann die Bearbeitung länger dauern. Es dürfen sich keine negativen Auswirkungen auf das Gesamtsystem durch dieses Verhalten ergeben, ansonsten liegt ein Hard-Realtime-System vor.

Meist findet man in einen System eine Kombination aus beiden.

## 4.1 Vordergrund-/Hintergrundsysteme

Kleine Systeme mit niedriger Komplexität werden meist als Vordergrund-/Hintergrundsysteme ausgeführt. Das Programm besteht dabei aus einer Endlosschleife (Hintergrund), in der die nicht zeitkritischen Aufgaben ausgeführt werden. Asynchrone, zeitkritische Ereignisse lösen (Hardware-)Interrupts aus, die das Hauptprogramm unterbrechen und sofort abgearbeitet werden (Abbildung 4.1). Nach Bearbeitung der Interruptserviceroutine (ISR) wird die Ausführung des Hauptprogramms fortgesetzt. Der Nachteil dieser Methode ist, dass in der Zeit, in der die Interrupt-Routine ausgeführt wird, die restlichen Aufgaben vernachlässigt werden. Die Reaktionszeit des Systems wird also von der Ausführungszeit der zeitintensivsten

Interruptroutine bestimmt. Da die Anzahl der zur Abarbeitung benötigten Taktzyklen eines typischen Codes nicht konstant ist, ist der zeitliche Ablauf eines solchen Systems nichtdeterministisch. Weiters ändert sich das Zeitverhalten eines solchen Systems mit jeder Softwareänderung.

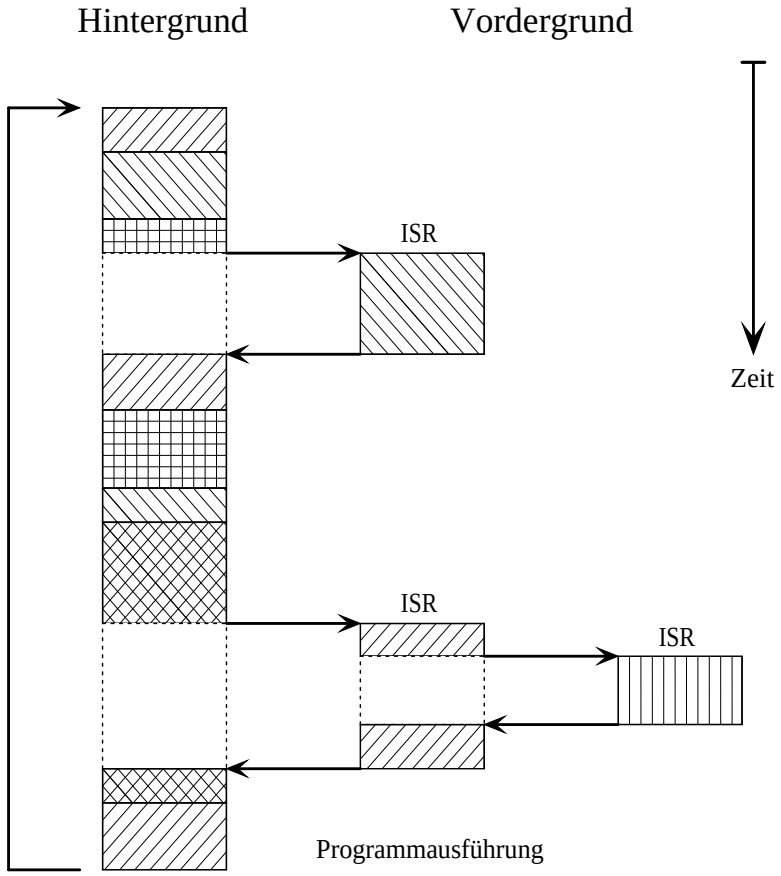



Abbildung 4.1: Vordergrund-/Hintergrundsystem

## 4.2 Multitasking-Systeme

Ein Task[1] ist ein einfaches Programm, das so agiert, als benutze es die CPU für sich alleine. Multitasking ist ein System, um mehrere Tasks scheinbar parallel auf einer CPU ablaufen zu lassen. Der Entwicklungsprozess eines Realtime-Systems beinhaltet, die Funktionen des Systems in einzelne Tasks aufzuteilen, die dann in einer Multitaskingumgebung ablaufen. Um nun die Illusion einer parallelen Abarbeitung zu erzeugen, werden Zeitabschnitte eingeführt, in denen je ein anderer Task das System für sich beanspruchen darf. Die Länge der einzelnen Zeitabschnitte und deren Verteilung an einzelne Tasks unterscheidet die verschiedenen Implementierungen von Multitasking

[1] oder Thread

Systemen. Des weiteren beinhaltet ein Multitasking-System Mechanismen zur Kommunikation der Tasks untereinander und zur Verwaltung der Resourcen des Systems (Speicher, Ein/Ausgabeeinheiten, ...).

### 4.2.1 Task

In einem Multitasking-System wird jedem Task eine Priorität zugeordnet. Mittels dieser Prioritätenliste wird entschieden, welcher Task für den nächsten Zeitabschnitt das System übernehmen darf und ob er einen anderen Task in seiner Ausführung unterbrechen kann. Weiters wird jedem Task einer von fünf Zuständen zugewiesen (Abbildung 4.2):

- DORMANT[2]

  Der Task befindet sich zwar im Speicher, ist jedoch noch nicht dem Multitasking-System zugeführt worden.

- READY

  Der Task ist bereit, jedoch ist seine Priorität zu gering, um abgearbeitet zu werden.

- RUNNING

  Der Task hat die Kontrolle über das System.

- WAITING

  Der Task wartet auf das Auftreten eines Ereignisses (Beendigung einer I/O-Operation, Verstreichen einer Zeitspanne,...).

- ISR

  Der Task wurde von einem Ereignis unterbrochen, welches einen Interrupt ausgelöst hat.

Verwaltet werden die einzelnen Tasks vom Kernel des Echtzeit-Systems. Die fundamentale Funktion des Kernels ist der "Context Switch", das Umschalten zwischen den Tasks. Der Umschaltvorgang läuft in mehreren Stufen ab: Als erstes werden die CPU-Register in den Stack[3] des aktuellen Tasks kopiert, und dieser gesichert. Dann wird entweder anhand der Prioritäten oder durch Auftreten eines Ereignisses der nachfolgende Task ermittelt. Als nächstes werden der gesicherte Stack und die Register vom nachfolgenden Task in den Arbeitsbereich kopiert, um letztendlich die Kontrolle über das System an diesen abzugeben.

[2] oder schlafend

[3] Speicherbereich für CPU-Register und lokale Variablen

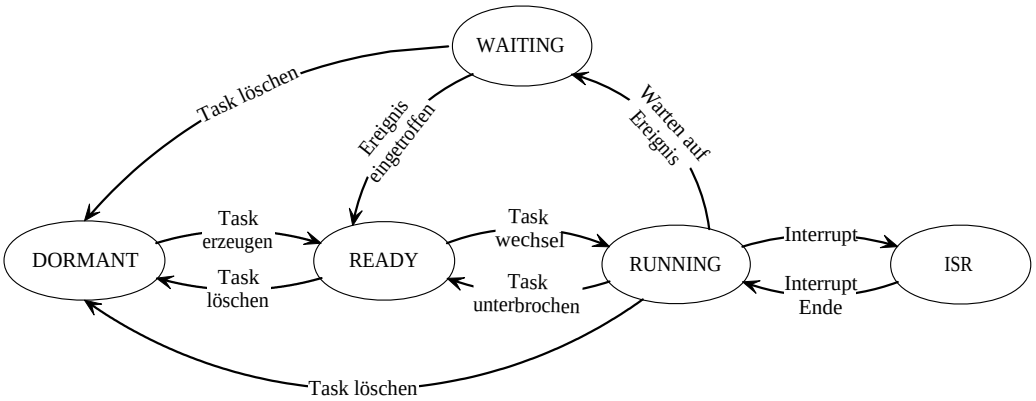



Abbildung 4.2: Task-Zustände

### 4.2.2 Scheduler

Der Scheduler ist der Teil des Kernels, der ermittelt, welcher Task als nächstes ablaufen soll. Die meisten Echtzeit-Kernel basieren auf Vergabe von Prioritäten. Die Priorität jedes Tasks ist abhängig von seiner Wichtigkeit. Sie muss bei einigen Systemen zur Programmerstellung fixiert werden, oder kann bei anderen Systemen zur Laufzeit geändert, und somit an aktuelle Ereignisse angepasst werden. Die Kontrolle über das System wird bei prioritätsbasierenden Systemen immer an den Task mit der höchsten Priorität fallen. Wann dieser Task jedoch die Kontrolle erhält, ist von der verwendeten Kernelart abhängig.

Es gibt 2 Arten von Echtzeit-Kernels: “preemtive[4]” und “non-preemtive”, welche im Folgenden kurz erleutert werden.

## 4.3 Non-Preemtive-Kernel

Beim Non-Preemtive-Kernel ist es notwendig, dass jeder Task die Kontrolle des Systems an den Kernel von sich aus zurückgibt. Um nun die Illusion der parallelen Abarbeitung aufrecht zu erhalten, muss dieses häufig geschehen, da erst die Aufgabe der Systemkontrolle einen Context Switch einleitet. Asynchrone Ereignisse werden jedoch von einer Interrupt Service Routine (ISR) abgearbeitet. Eine ISR kann einen Task mit hoher Priorität in den “READY”-Status versetzen, nach Beendigung der Bearbeitung wird jedoch wieder der unterbrochene Task fortgesetzt. Wird nun innerhalb der Interruptroutine die Priorität dieses Tasks auf einen niedrigeren Wert geändert, wird dies erst beim nächsten Context Switch berücksichtigt. Meist wird dieses Verhalten auch “Cooperative Multitasking” genannt.

[4]oder präventiv

Einer der Vorteile des Non-Preemtive-Kernel ist, dass auf Task-Ebene Non-Reentrant Funktionen (siehe Abschnitt 4.4.1) aufgerufen werden können, da sichergestellt ist, dass kein anderer Task dieselbe Funktion gleichzeitig benutzt. Die Verzögerung auf Task-Ebene ist geringer als im Vordergrund-/Hintergrundsystem, da nicht die komplette Schleife des Hintergrundsystems durchlaufen werden muss. Sie ist gleichzusetzen mit der Bearbeitungszeit für den zeitintensivsten Task. Weiters ist es nicht notwendig, gemeinsam benutzte Datenbereiche gegen Schreibzugriffe von anderen Tasks zu sichern. Dies gilt allerdings nur auf Task-Ebene. Zugriffe auf andere Resourcen, die zeitintensiver sind und deshalb die Kontrolle über das System zeitweilig nicht notwendig ist[5], sind sehrwohl gegen die Störung durch andere Tasks zu sichern.

Der typische Ablauf eines Non-Preemptive-Kernel ist in Abbildung 4.3 dargestellt.

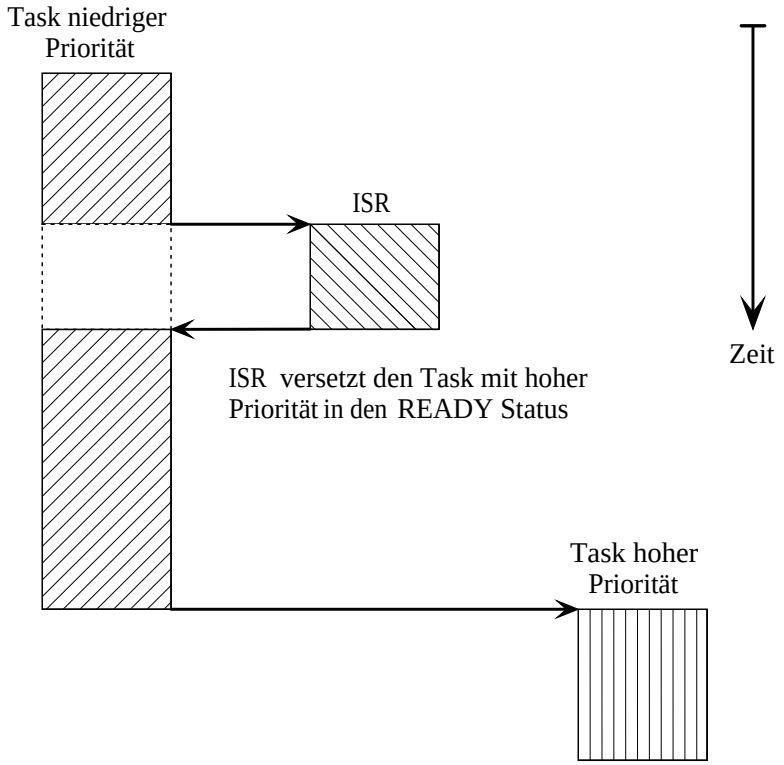



Abbildung 4.3: Non-Preemptive-Kernel

Ein Task läuft ab, wird aber durch ein asynchrones Ereignis unterbrochen. Dieses Ereignis löst einen Interrupt aus. Sind Interrupts gerade nicht unterbunden (etwa um sensible Operationen zu schützen), so wird eine ISR aufgerufen. Diese bearbeitet das Ereignis und setzt einen höher priorisierten Task in den READY-Status. Danach wird ein "Return From Interrupt"-Befehl

[5]Zugriffe auf Drucker, externe I/O Operationen, ...

aufgerufen und die CPU bearbeitet den unterbrochenen Task weiter. Dieser gibt nach Beendigung seiner Berechnungen die Kontrolle des Systems an den Scheduler zurück. Dann wird der Task gestartet, der sich im READY-Status befindet und die höchste Priorität besitzt.

Die wichtigste Einschränkung der Non-Preemtive Methode ist die Reaktionszeit in der Task-Ebene. Wird ein Task mit hoher Priorität aktiviert, muss dieser so lange mit der Ausführung warten, bis der gerade ablaufende Task die Kontrolle übergibt. Ebenso wie beim Vordergrund-/Hintergrundsystem ist dadurch der zeitliche Ablauf des Systems nichtdeterministisch.

## 4.4 Preemtive-Kernel

Erst beim Preemtiv-Kernel ist dieses Problem der Nichtvorhersagbarkeit des Zeitverhaltens gelöst. Der Unterschied zum Non-Preemtive-Kernel ist, dass nach Aktivieren eines höher priorisierten Tasks in der Task-Ebene sofort zu diesem Task gewechselt wird. Es wird nicht gewartet, bis der aktuelle Task die Kontrolle freiwillig abgibt, es wird ihm die Kontrolle entzogen.

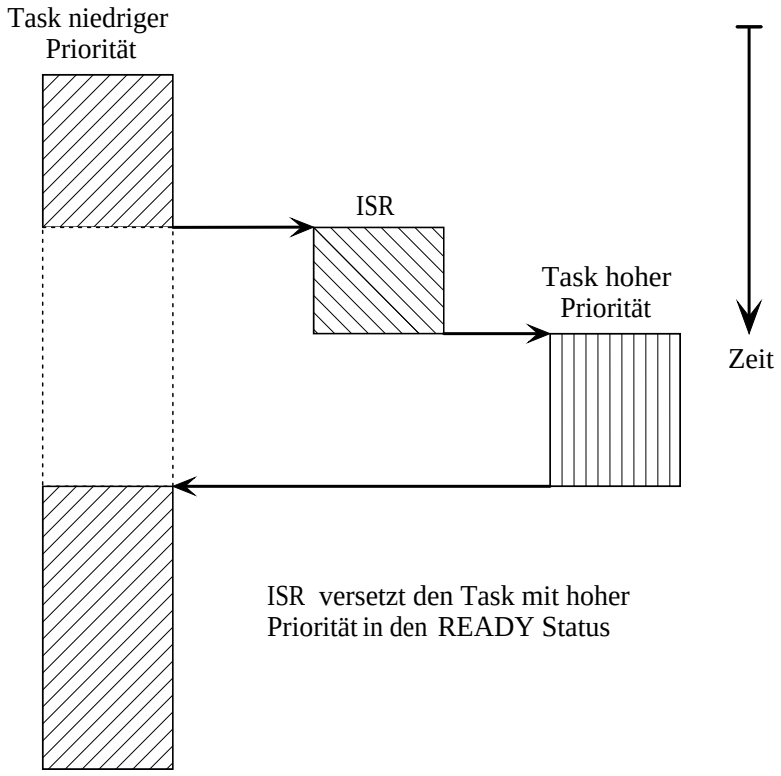



Abbildung 4.4: Preemptive-Kernel

Wird ein Task in der ISR aktiviert, dann kehrt das System nach der Beendigung der ISR nicht zum unterbrochenen Task zurück, sondern es wird der

am höchsten priorisierte Task gestartet (siehe Abbildung 4.4). Damit wird mit einem Preemtiv-Kernel der Ablauf deterministisch und die Reaktionszeit in der Task-Ebene minimiert.

Eine Einschränkung ist, dass beim Preemtiv-Kernel nur Reentrant-Funktionen benutzt werden sollten, beziehungsweise müssen Vorkehrungen getroffen werden, um exklusiven Zugang zu Non-Reentrant-Funktionen sicherzustellen, da hier ein Task jederzeit von einem anderen unterbrochen werden kann.

### 4.4.1 Reentrant-Funktionen

Reentrant-Funktionen können von mehreren Tasks gleichzeitig verwendet werden, ohne sich gegenseitig zu beeinflussen. Eine Reentrant-Funktion kann jederzeit unterbrochen und ohne Datenverlust später fortgesetzt werden. Um dieses zu ermöglichen, müssen entweder lokale Variablen (CPU Register oder Variablen am Stack) verwendet, oder bei globalen Variablen diese geschützt werden.

Um dies zu verdeutlichen, soll diese einfache Non-Reentrant-Funktion durchdacht werden (siehe auch Abbildung 4.5):

```
    char temp;

    void VariablenTausch (char a, char b)
    {
1       temp = a;
2       a = b;
3       b = temp;
    }
```

Angenommen, diese Funktion wird von Task 1 mit den Werten a=11, b=12 aufgerufen und während der Abarbeitung in Zeile 2 von einem Task 2 mit höherer Priorität unterbrochen. Wenn dieser dieselbe Funktion mit den Parametern a=21, b=22 aufruft, wird a zu 22 und b zu 21 und temp zu 21. Nun wird Task 2 beendet und Task 1 fortgesetzt. Die Variable a wird korrekt auf 12 gesetzt, b wird jedoch nicht auf 11, sondern auf 21 gesetzt, da temp global definiert ist und somit von Task 2 benutzt wurde.

### 4.4.2 Mutal Exclusion

Die einfachste Weise zwei Tasks miteinander kommunizieren zu lassen, ist mittels eines gemeinsamen Speicherbereichs. Es ist jedoch darauf zu achten, dass andere Tasks diesen Speicherbereich nicht verändern dürfen wenn ein Task darauf zugreift. Dieses gegenseitige Blockieren einer Operation nennt

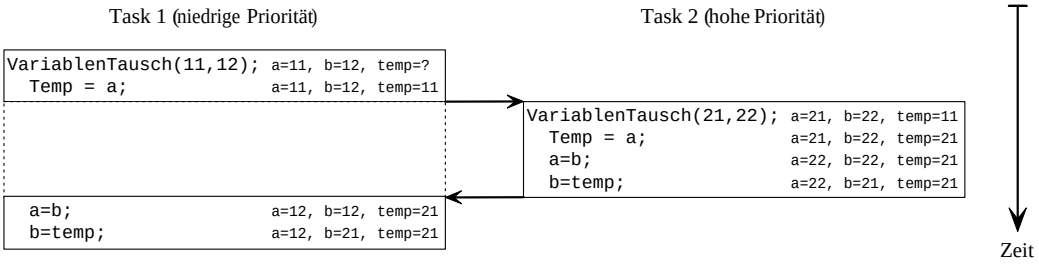



Abbildung 4.5: Reentrant-Funktion

man auch “Mutal Exclusion”, den Programmbereich, der die Operationen durchführt, auch “Critical Section”.

Im allgemeinen gibt es folgende Methoden um dieses zu erreichen:

- Sperren von Interruptaufrufen

  Im allgemeinen ist es nicht von Vorteil, Interruptanforderungen zu sperren. Um die Interrupt Reaktionszeit nicht allzusehr zu verschlechtern, sind die Abschnitte, in denen die Interrupts gesperrt sind, kurz und selten zu halten.

- Verwenden von “Test and Set”-Operationen

  Bei den meisten CPUs gibt es sogenannte “Test and Set”-Operationen. Dies sind Assembleranweisungen, die eine Zustandsabfrage und eine Zuweisungsoperation in einem Maschinenschritt ausführen. Dies hat den Vorteil, dass eine solche Anweisung nicht unterbrochen werden kann. Sollte ein solcher Befehl nicht existieren, kann dies umgangen werden, indem die Interrupts gesperrt, dann die Abfrage- und Zuweisungsoperation ausgeführt und die Interrupts sofort wieder freigegeben werden. Mit einer “Test and Set”-Operation ist es möglich, vor einem Speicherzugriff eine Überprüfung durchzuführen, ob auf den Speicherbereich zugegriffen werden darf, und wenn ja, den Zugriff auf diesen für andere Tasks mit der selben Operation zu sperren.

- Sperren des Schedulers

  Wird der gemeinsame Speicherbereich nicht in einer ISR genutzt, kann in der Zeit des Zugriffes der Scheduler gesperrt werden. Dies verschlechtert jedoch wieder die Reaktionszeit auf Task-Ebene, jedoch nicht auf Interrupt-Ebene. Um diese Funktionalität im Preemtive-Kernel bereitzustellen, muss nach der ISR an den unterbrochenen Task zurückgekehrt werden, ganz gleich, ob ein höher priorisierter Task an der Reihe ist.

- Verwendung von Semaphoren (siehe Abschnitt 4.4.3)

  Semaphore werden von den meisten Multitasking-Kernels zur Verfügung gestellt. Sie finden nicht nur Verwendung zur Mutal Exclusion, sonder werden auch zur Signalisierung eines Ereignisses und zur Synchronisation zwischen zwei Tasks eingesetzt.

  Es sind zwei Arten von Semaphoren möglich: Zählende Semaphoren und binäre Semaphoren.

### 4.4.3 Semaphore

Ein Semaphor besteht aus einem Zähler und einer Warteschlange. Der Zähler kann nur positive Werte inklusive Null annehmen. Wird von einem Task ein Semaphor angefordert, wird erst der Zähler abgefragt. Ist dieser Null, wird der Task in die Warteschlange eingetragen und der Scheduler aufgerufen, der den nächsten Task, der sich nicht in der Warteschlange befindet, aktiviert. Ist der Zähler größer Null, wird dieser dekrementiert und die Bearbeitung fortgesetzt. Beim Verlassen der Critical Section wird der Semaphor schließlich wieder inkrementiert und die Warteschlange nach höher priorisierten Tasks durchsucht, die dann das System übernehmen. Befindet sich kein Task mit höherer Priorität in der Warteschlange, wird mit der Abarbeitung des aktuellen Tasks fortgefahren.

Der Unterschied zwischen binärem und zählendem Semaphor besteht lediglich in der Initialisierung des Zählers. Dieser wird beim binären Semaphor auf 1 initialisiert. Die Semaphoraufrufe müssen immer paarweise erfolgen. Werden mehrere verschiedene Semaphore auf einmal angefordert, muss dies immer und in allen Bereichen in der gleichen Reihenfolge erfolgen, die beim Verlassen der Critical Section umzukehren ist.

Wird dies nicht beachtet, kann es zu gegenseitiger Stilllegung einzelner Tasks kommen, dem sogenannten “Deadlock”. Müssen zum Beispiel, um auf einen gemeinsamen Speicherbereich zugreifen zu dürfen, zwei Semaphore ($a$ und $b$) angefordert werden und dies geschieht in zwei Tasks in unterschiedlicher Reihenfolge, kann es passieren, dass der niederpriorisierte Task den Semaphor $a$ anfordert, danach von dem höher priorisierten unterbrochen wird, der den Semaphor $b$ als erstes anfordert. Versucht nun dieser den Semaphor $a$ anzufordern, wird er in die Warteschlange dieses Semaphores eingereiht und der Task mit niedrigerer Priorität fortgesetzt. Dieser versucht den Semaphor $b$ anzufordern und wird ebenfalls in eine Warteschlange eingereiht. Nun hat jeder Task einen Semaphor für sich beansprucht und wartet in einer Warteschlange auf die Freigabe des jeweils anderen. Es ist dann für beide Tasks nicht möglich, ihre Abarbeitung fortzusetzen.

### 4.4.4 Message Queue

Anders als bei Semaphoren, die zur Synchronisation von Tasks dienen, werden Messagequeues zur Interprozesskommunikation eingesetzt. Meist werden diese folgendermaßen implementiert: Die Queue wird als FIFO-Buffer (First In First Out) realisiert und der Zugriff darauf mittels Semaphoren synchronisiert. In diesem Buffer darf jeder Task Nachrichten ablegen, die dann vom Besitzertask abgeholt werden. Der Besitzer der Queue pollt nun in seiner Hauptschleife, ob neue Nachrichten vorhanden sind, reagiert darauf und gibt die Systemkontrolle am Ende seiner Hauptschleife an das System ab. Sind keine Nachrichten vorhanden, wird die Hauptschleife ohne nachrichtenbezogene Aktivität durchlaufen. Da dies unnötige Rechenzeit benötigt, werden auch andere Ansätze verfolgt. Dabei kann sich der Task bei Aufruf einer Nachricht in eine Warteschlange eintragen lassen, die erst wieder verlassen wird, wenn eine Nachricht eingetroffen ist.

Der Nachteil ist, dass der Task seine anderen Aufgaben in der Zwischenzeit nicht ausführen kann. Es hat also jede der beiden Verfahren ihre Nachteile.

### 4.4.5 Clock Tick

Der periodisch auftretende Clock-Tick-Interrupt erlaubt es dem Kernel, einen Task um ein ganzzahliges Vielfaches dieser Periode zu verzögern und Zeitüberschreitungen beim Nichtauftreten von Ereignissen, die eine Systemreaktion hervorrufen sollen, zu erkennen. Die Periodendauer seines Auftretens wird applikationsspezifisch gewählt und liegt meist im Bereich von 10 bis 200 ms. Je kleiner dieser Wert gewählt wird, desto höher ist die Systemlast, die durch die Abarbeitung dieses Interrupts hervorgerufen wird. Wird die Periodendauer hingegen zu groß gewählt, werden dadurch wachsende Ungenauigkeiten in der gezielten Taskverzögerung hervorgerufen, da immer ein ganzzahliges Vielfaches der Interruptperiodendauer als Verzögerungszeitspanne gewählt werden muss. Die Auflösung der Zeitverzögerung ist dabei ein “Tick”, eine Periodendauer. Über die Zeitspanne der Verzögerung können so nur sehr ungenaue Angaben gemacht werden (siehe Abbildung 4.6). Sie hängt in der Regel vom verwendeten Kernel, beziehungsweise vom verwendeten Scheduler und von der Taskpriorität ab, sogar die Codelänge der einzelnen Tasks hat Einfluss auf den Ausführungszeitpunkt.

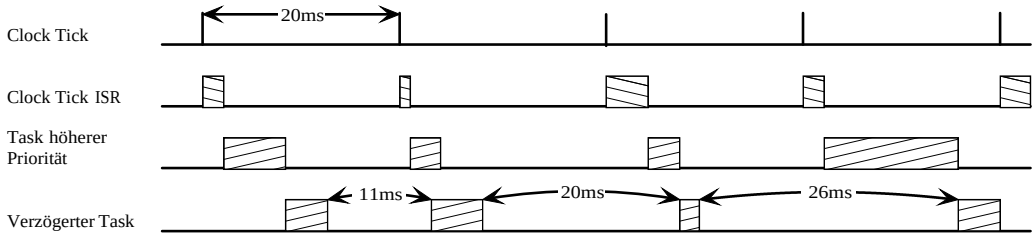



Abbildung 4.6: Verzögerung eines einzelnen Ticks

## 4.5 Zusammenfassende Bemerkungen

Der große Unterschied vom Multitasking-System zum Vordergrund-/Hintergrundsystem liegt in der Verwendung eines Kernels. Er stellt Funktionen zur Verfügung, die das Erstellen der Software wesentlich vereinfachen. Natürlich ist mit dem System des Multitaskings ein gewisser Mehraufwand in der Abarbeitungszeit und des Resourcenverbrauchs verbunden, der sich jedoch ab einer gewissen Systemgröße und -komplexität leicht rechtfertigen lässt. Ebenso zu erwähnen sind die Kosten eines solchen Softwarepakets. Diese können einmalige Zahlungen oder auf Stückzahl bezogene Abgaben an den Hersteller sein. Weiters darf man die Kosten für den gesteigerten Resourcenverbrauch (ROM, RAM, Prozessorleistung) nicht vernachlässigen.

Als weiterführende Literatur kann das Buch “MicroC/OS-II: The Real-Time Kernel” von J. Labrosse empfohlen werden [4].

# Kapitel 5

# Betriebssystem des Steuerrechners

## 5.1 Linux

Linux ist ein dem Unix ähnliches, freies Betriebssystem, ursprünglich entwickelt von Linus Torwalds. Dieser veröffentlichte den Quellcode seiner Software unter der GNU General Public License. Damit ist es für jeden Softwareentwickler möglich, sich an der Entwicklung von Linux zu beteiligen. Die einzige zu erfüllende Voraussetzung ist, dass er die Änderungen am Linux Code wieder der Allgemeinheit zuführt. Dies ist der hauptsächliche Gedanke, der hinter der Idee freier Software steht und der das rasche Wachstum von Linux und anderen freien Softwarepaketen ermöglicht.

Linux ist aufgrund seiner Vielfältigkeit und Skalierbarkeit weltweit sehr beliebt. Die Anwendungen reichen vom Bürocomputer über den Internet-Server bis in die Welt der Embedded Systems[1]. Zu diesem Zweck wurde Linux auf zahlreiche Hardwareplattformen portiert, deren Rechenleistung fast das gesamte vorhandene Spektrum abdecken.

## 5.2 Realtime-Linux

Realtime-Linux ist eine Variante des Linux-Kernels, die es ermöglicht, Hard-Realtime-Aufgaben innerhalb einer Linux-Umgebung zu bearbeiten. Dies ist für die Verwendung von Linux in Embedded-Systemen von entscheidender Bedeutung, denn Linux an sich ist nicht echtzeitfähig und die Reaktionszeit des Standard-Linux somit nicht vorhersehbar, da diese vom aktuellen Systemlastfall abhängt. Reaktionszeiten von einigen Millisekunden sind keine

[1] Ein Embedded-System ist ein System, das einen Rechner beinhaltet, der in das System integriert ist, und nicht als eigenständiger Computer zu erkennen ist. (z.B. ein Steuerrechner in einer Waschmaschine).

Seltenheit. Mit RT-Linux können Reaktionszeiten unter 15 Mikrosekunden (auf einer Intel-Plattform) garantiert werden, unabhängig davon welcher Systemlastfall vorliegt. Erreicht wird dies durch die spezielle Art und Weise, wie das Echtzeitsystem an das Linux-System angebunden ist. Der große Vorteil von RT-Linux ist, dass die Standard-Linux Entwicklungsumgebung auf Echtzeitprobleme angewandt werden kann. Die Echtzeitroutinen kommunizieren mittels FIFO-Buffer oder Shared Memory mit normalen Linux-Programmen. Dies hat den Vorteil, dass der Echtzeitteil möglichst unkompliziert gehalten wird und die nicht zeitkritische Aufbereitung und Verarbeitung der Daten durch einfach zu entwickelnde Linux-Software erfolgt. Diese kann auf alle Resoucen, die das Linux-System bereitstellt, in gewohnter Art und Weise zugreifen. Einer der Haupt-Designstrategien von Realtime-Linux ist, dass soviel wie möglich im Standard-Linux und so wenig wie nur unbedingt nötig im Realtime-Linux implementiert werden soll.

### 5.2.1 Struktur von Realtime-Linux

Realtime-Linux ist im Grunde ein eigenständiges Mini-Betriebssystem, in dessen Ruhe-Task (dieser Task wird ausgeführt, wenn gerade kein anderer Task das System beansprucht) das Standard-Linux abgearbeitet wird. Linux besitzt hier die niedrigste Priorität. Das und die Tatsache, dass das Mini-Betriebssystem auf Reaktionsgeschwindigkeit getrimmt ist, garantieren die Hard-Realtime-Fähigkeit des Systems. Außerdem ist dem Linux-System kein Zugriff auf die Sperrung von Interrupt-Anforderungen der Hardware gestattet. Möchte der Linux-Kernel Interrupts sperren, wird dies vom RT-Kern abgefangen, notiert und nicht an die Hardware weitergegeben. Wird nun in der Zwischenzeit ein Interrupt ausgelöst, wird dieser vom Realtime-Kern behandelt und erst nach erfolgter Interruptaktivierung durch den Linux-Kernel an diesen weitergegeben.

### 5.2.2 Der Modulare Aufbau von Realtime-Linux

Realtime-Linux ist modular aufgebaut (Abbildung 5.1). Die Module werden vom Standard-Linux als Kernel-Module geladen und installiert. Dies hat den Vorteil, dass nur die benötigten Module eingebunden werden. Sollte ein Modul nicht den Anforderungen des zu lösenden Problems genügen, kann dieses leicht durch ein anderes ersetzt werden. So gibt es zum Beispiel mehrere verschiedene Scheduler mit unterschiedlichen Strategien.

Abbildung 5.2 zeigt die im Moment implementierten Module von Realtime-Linux.

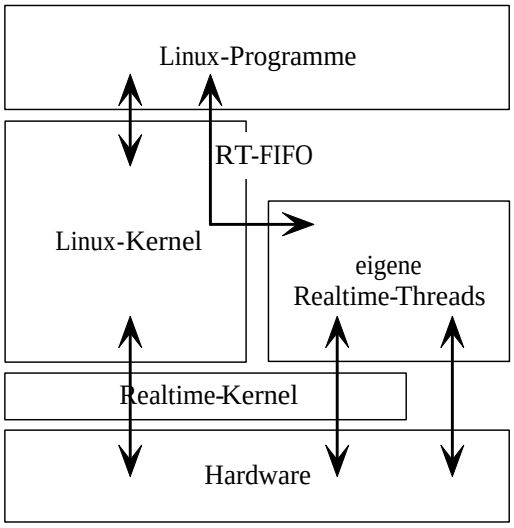



Abbildung 5.1: Funktionsblöcke im Realtime-Linux

### 5.2.3 Applikationsinterface

Realtime-Linux verfügt seit der Version 2 über POSIX[2] kompatible Systemaufrufe. In den Schriften von POSIX wird der Begriff "Thread" anstatt von Task verwendet. Diese Bezeichnung wurde in der Originaldokumentation von Realtime-Linux übernommen, daher wird sie auch hier verwendet. Dies mag für Leser mit nicht so tiefem Einblick in die Materie etwas verwirrend sein, vermeidet aber Inkonsistenz zu der englischsprachigen Originaldokumentation. Deren genaueres Studium ist bei der Verwendung von Realtime-Linux unbedingt notwendig, da hier nur ein Überblick in den Funktionsumfang geboten werden soll.

Im Folgenden werden Systemaufrufe aus dem Applikationsinterface von Realtime-Linux auszugrzeise dargestellt und deren Funktion jeweils in einem anschließenden Absatz erörtert.

**int pthread_create(pthread_t *thread, pthread_attr_t *attr,**
**void *(*start_routine)(void *), void *arg);**

kreiert einen Echtzeit-Thread mit der Threadstruktur $*thread$. Die Threadstruktur $*thread$ wird vom Programm bereitgestellt und muss während der Lebenszeit des Tasks statisch sein. Es werden Attribute an den Thread übergeben, die mit *pthread_attr_init* erzeugt werden. Wird anstatt der Attribute der NULL Pointer übergeben, werden Standardattribute verwendet. $*start_routine$ beschreibt den Einstiegspunkt des neuen Threads. Außerdem kann einen Pointer $*arg$ an den neuen Thread übergeben werden.

[2]Abteilung der IEEE, die Abkürzung steht für Portable Operation Systems Interface. Diese Gruppe beschäftigt sich mit der Standardisierung im Bereich der Echtzeisysteme

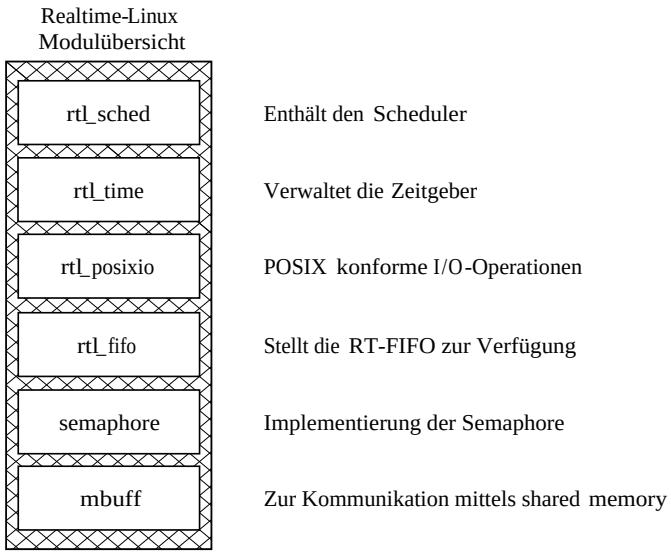



Abbildung 5.2: Module von Realtime-Linux

Im Normalfall wird 0 zurückgegeben, im Fehlerfall gibt der Rückgabewert genauere Auskunft über den Fehler.

**int pthread_attr_init(pthread_attr_t *attr);**
**int pthread_attr_destroy(pthread_attr_t *attr);**

*pthread_attr_init* initialisiert $*attr$, welches die Attribute eines Thread enthält. Das resultierende Objekt $*attr$ kann durch Veränderung einzelner Attribute an die Gegebenheiten angepasst werden und wird zum Erstellen eines Threads benötigt. Dieselbe Struktur kann zum Kreieren mehrerer Threads verwendet werden.

Mit *pthread_attr_destroy* wird das durch *pthread_attr_init* erzeugte Objekt gelöscht.

**int pthread_delete_np(pthread_t thread);**

löscht einen mit *pthread_create* erzeugten Thread aus dem System. *thread* ist die Threadstruktur, die beim Initialisieren übergeben wurde.

**int pthread_make_periodic_np(pthread_t thread,**
**hrtime_t start_time, hrtime_t period);**

versetzt den Thread *thread* in den READY-Status. Der Thread wird zu der mit *start_time* festgelegten Zeit gestartet und ab diesem Zeitpunkt periodisch mit der Periodendauer *period* aufgerufen. Die beiden Zeiten werden in Nanosekunden angegeben, die resultierende Auflösung des Systems hängt jedoch von der verwendeten Hardwarezeitbasis ab.

**int pthread_wait_np(void);**

versetzt den aktuellen Thread in den WAITING-Status. Die Systemkontrolle wird an den Scheduler abgegeben, der den nächsten Thread startet. Der unterbrochene Thread wird zu gegebener Zeit vom Scheduler wieder aktiviert.

**int pthread_suspend_np(pthread_t thread);**
**int pthread_wakeup_np(pthread_t thread);**

*pthread_suspend_np* unterbricht die Abarbeitung des angegebenen Threads bis zum Aufruf von *pthread_wakeup_np*. Soll der aktuelle Thread mittels Aufruf von $pthread_suspend_np(pthread_self())$ unterbrochen werden, muss als nächste Anweisung *pthread_wait_np* folgen, um die Kontrolle an den Scheduler zu übergeben. Der unterbrochene Thread kann dann von einem anderen Thread oder Händler[3] mittels *pthread_wakeup_np* aktiviert werden.

**int rtf_create(unsigned int fifo, int size);**

erzeugt einen RT-FIFO[4]. Diese kann zur Kommunikation mit einem Linux-Programm verwendet werden. RT-FIFO sind unter Linux als character device implementiert (/dev/rtf*). Das bedeutet, man kann auf der Linux-Seite den RT-FIFO behandeln wie eine Datei. Der RT-FIFO wird mit *open* geöffnet, mit *read* oder *write* wird darauf zugegriffen und mittels *close* wird sie geschlossen. Die Speichergröße des zu erzeugenden RT-FIFO wird mittels *size* festgelegt. $fifo$ muss eine systemweit eindeutige Zahl sein, deren Wert kleiner als RTF_NO sein muss.

**int rtf_create_handler(unsigned int fifo, int (* handler)());**

Um einen RT-FIFO nicht immer auf das Vorhandensein neuer Daten pollen zu müssen, kann mittels $rtf_creat_handler$ eine Routine eingerichtet werden, die aufgerufen wird, wenn ein Linux-Programm in den RT-FIFO schreibt oder aus ihm Daten liest.

**int rtf_destroy(unsigned int fifo);**

entfernt den mit $fifo$ bezeichnete RT-FIFO aus dem System. Alle Handler-Routinen werden von der RT-FIFO entfernt, und der Wert $fifo$ steht dann zur weiteren Verwendung mit $rtf_create$ zu Verfügung.

[3]ist eine Routine, die vom Kernel aufgerufen wird, wenn Daten an den FIFO-Buffer geschickt werden. Man erspart sich das periodische Abfragen, ob Daten vorhanden sind.
[4]Realtime-FIFO-Buffer

**int rtf_get(unsigned int fifo, char * buf, int count);**
**int rtf_put(unsigned int fifo, char * buf, int count);**

*rtf_get* liest ein Datenpaket von einen mit $fifo$ bestimmten RT-FIFO der Länge *count* in den Speicherbereich $*buf$. Mit *rtf_put* werden Daten in den RT-FIFO geschrieben. Diese Aufrufe werden nur von der Realtime-Seite des RT-FIFO benutzt. Auf der Linux-Seite sind die Kommandos *read* bzw. *write* zu verwenden, wenn auf den RT-FIFO zugegriffen werden soll.

Im Erfolgsfall wird die Anzahl der übertragenen Zeichen zurückgegeben, im Fehlerfall hingegen ein negativer Wert, der Auskunft über die Fehlerursache gibt.

Die nachfolgende Grafik gibt Auskunft über die Threadzustände bei Verwendung der oben angeführten Funktionen:

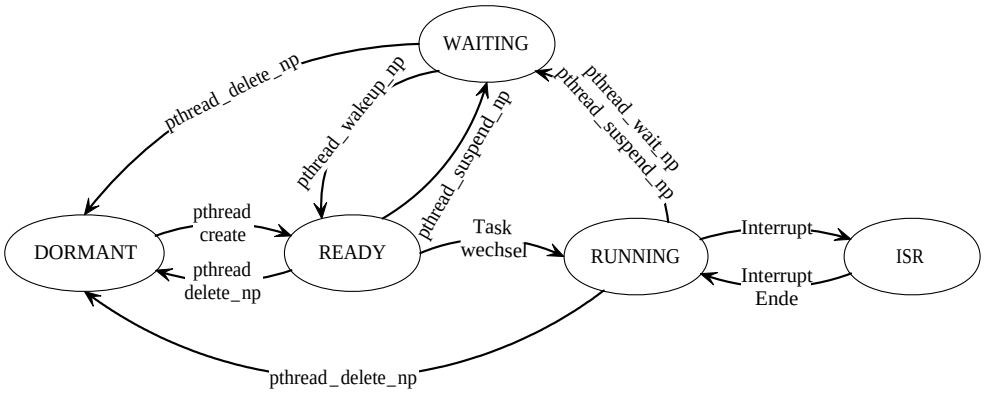



Abbildung 5.3: Taskzustandswechsel im Realtime-Linux

### 5.2.4 Beispiel einer Applikation unter Realtime-Linux

Folgendes Beispiel soll verdeutlichen, wie gering der Aufwand ist, um eine Echtzeitanwendung zu erstellen.

Folgende Aufgabenstellung soll für das Beispiel gelten: Es befindet sich eine Ein-/Ausgabe-Karte im Computer. Diese ist unter der Adresse 0x03A0 im Adressbereich des ISA-Busses anzusprechen.

Sie besitzt 4 Register:

**0x03A0** Port A

**0x03A1** Port B

**0x03A2** Port C

**0x03A3** Konfigurations- und Statusregister

Wird in das Statusregister der Wert 0x81 geschrieben, so ist Port A in Ausgaberichtung geschaltet. Port B und C werden in diesem Beispiel nicht verwendet, bleiben daher unberücksichtigt. Mittels Bit 0 des Ports A lässt sich eine Leuchtdiode ein- und ausschalten. Es soll ein Programm erstellt werden, das diese Leuchtdiode mit 1 Herz blinken lässt. Diese Vorgabe soll unter allen Umständen eingehalten werden. Das Blinken soll beliebig ein- und ausgeschaltet werden.

Wir lösen diese Aufgabe im Folgendem mittels eines Realtime-Programms.

Als erstes werden ein paar notwendige Include-Dateien eingebunden und Definitionen festgelegt. Damit die Realtime-Routinen als Kernel-Modul eingebunden werden, müssen sowohl die Realtime- als auch die Kernel-Includes geladen werden. Danach folgt die Festlegung bezüglich der verwendeten Hardware, der verwendeten RT-FIFO und diverser Thread-Parameter.

```
#include "/usr/src/linux/include/linux/modversions.h"
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/version.h>
#include <linux/errno.h>
#include <linux/malloc.h>

#include <rtl_fifo.h>

#define PORT_CONFIG         0x81
#define CONFIG_PORT_ADDR    0x03A3
#define OUTPUT_PORT_ADDR    0x03A0
#define LED_ON              0x01
#define LED_OFF             0x00

#define FIFO_NUM    0
#define FIFO_SIZE   1000

#define MY_THREAD_SPEED         2           /* 2 times per second */
#define MY_THREAD_PERIOD (NS_PER_SECOND/MY_THREAD_SPEED)
#define MY_THREAD_STACK     4000
#define MY_THREAD_PRIORITY  4

struct my_thread_struct
{
    pthread_t thread;
    int running;
};

struct my_thread_struct my_thread;
```

In der Variable *running* in der Struktur *my_thread* wird festgehalten, ob die Leuchtdiode blinken soll oder nicht. Die Variable *thread* wird für interne Zwecke von RT-Linux verwendet.

```
/* *************** Initialisierung ********************** */

int init_module(void)
{
    pthread_attr_t attr;
    struct sched_param sched_param;

    my_thread.running = 0;

    /* init hardware */

    outb (PORT_CONFIG,CONFIG_PORT_ADDR);

    /* calculate attributes */

    pthread_attr_init (&attr);
    sched_param.sched_priority = MY_THREAD_PRIORITY;
    pthread_attr_setschedparam (&attr, &sched_param);

    /* create thread and run it */

    if (pthread_create (&(my_thread.thread), &attr,
                        my_thread_routine, (void *)1) < 0)
        return 1;

    /* create the rt-fifo */

    rtf_destroy(FIFO_NUM);
    if (rtf_create(FIFO_NUM, FIFO_SIZE))
        return 1;                           // report an error

    /* attach handler to fifo */

    if (rtf_create_handler(FIFO_NUM, &fifo_handler))
        return 1;                           // report an error

    return 0;
}
```

Nach dem Laden des Moduls wird zuerst die Funktion *init_module* aufgerufen. Dort wird unser Thread in das Echtzeitsystem eingebunden, gestartet

und der RT-FIFO zur Kommunikation zum Linux-System initialisiert. Weiters wird ein Händler für den RT-FIFO eingebunden, der aufgerufen wird, sobald der RT-FIFO Daten zur Verarbeitung enthält. Die Verwendung dieses Mechanismus erspart das ständige Pollen auf Vorhandensein von Daten in den RT-FIFO.

```
/* ***************** fifo handler ********************** */

int fifo_handler(unsigned int fifo)
{
    unsigned char data;
    int err;

    while ((err = rtf_get(FIFO_NUM, &data, sizeof(data)))
                == sizeof(data))
    {
        switch (data)
        {
            case '0':
                my_thread.running = 0;
                break;
            case '1':
                my_thread.running = 1;
                break;
            default:
                break;
        }
    }
    if (err != 0)
    {
        return -EINVAL;
    }
    return 0;
}
```

Anschließend wird der Echtzeit-Thread abgearbeitet, der als erstes einen Timer startet, um den Thread in periodischen Abständen zu aktivieren. In einer Endlosschleife wird dann auf das Eintreten des ersten Timer-Ablaufs gewartet. Tritt dieses Timer-Ereignis auf, wird der Scheduler diesen Thread aufrufen, falls nicht die Bearbeitung eines höher priorisierten anliegt. Er wird dann aufgrund der Variable *running* entweder die Leuchtdiode einschalten, ausschalten oder einfach nichts tun. Danach wird aufgrund der Endlosschleife auf den nächsten Timer-Ablauf gewartet.

```
/* ***************** thread function ********************** */

void *my_thread_routine(void *call_data)
{
    pthread_make_periodic_np(my_thread.thread,
                 gethrtime(), MY_THREAD_PERIOD);

    while (1)
    {
        pthread_wait_np();
        switch (my_thread.running)
        {
            case 1:
                outb (LED_ON,OUTPUT_PORT_ADDR);
                my_thread.running = 2;
                break;
            case 2:
                outb (LED_OFF,OUTPUT_PORT_ADDR);
                my_thread.running = 1;
                break;
            default:
                break;
        }
    }
}
```

Wird nun dieser Modul wieder aus dem Realtime-Kontext entfernt, wird die Funktion *cleanup_module* aufgerufen, die den Thread aus der Liste der aktiven Threads entfernt, die Hardware in einen sicheren Betriebszustand versetzt und den RT-FIFO zur anderweitigen Verwendung freigibt.

```
/* ***************** Beenden ********************** */

void cleanup_module(void)
{
    pthread_delete_np(guardthread.thread);

    outb (LED_OFF,OUTPUT_PORT_ADDR);

    rtf_destroy(FIFO_NUM);
}
```

Nach dem Kompilieren kann das Echtzeitmodul mittels “insmod filename.o” in das System geladen werden. Mit den Befehlen “echo 1 > /dev/rtf0” beziehungsweise “echo 0 > /dev/rtf0” kann das Blinken der Leuchtdiode gesteuert

werden. Mit “rmmod filename” kann das Echtzeitmodul aus dem System entfernt werden.

Tiefere Einblicke in RT-Linux kann man aus [5] und [6] beziehen.

# Kapitel 6

# Entwicklung der Hard- und Softwarekomponenten

Da das von uns verwendete System über so viele Funktionen verfügt, reichen die Standard-Anschlüsse eines Personal-Computers nicht aus, um die gesamte Laserhardware anzusteuern. Darum musste eine zusätzliche IO-Karte entwickelt werden. In diesem Kapitel wird nun diese Karte vorgestellt und deren Funktionsweise erläutert.

Bild 6.1 zeigt das Blockschaltbild der entwickelten Hardware.

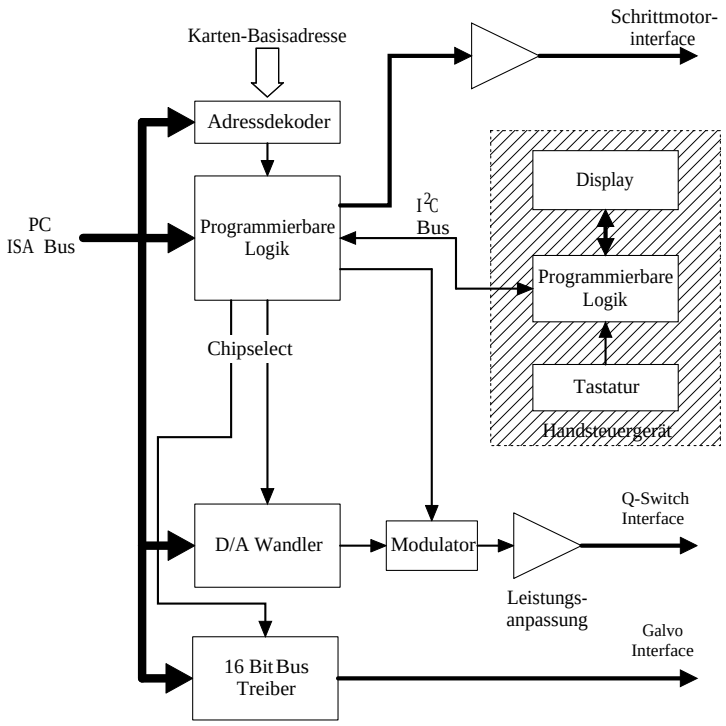



Abbildung 6.1: Blockschaltbild der entwickelten Hardware

## 6.1 IO-Karte

### 6.1.1 Funktionen der Karte

Folgende Funktionen wurden bei der Entwicklung der Karte implementiert:

- Generierung der Impulse für die Ansteuerung des Q-Switch
- Ein 16-Bit Bus zur Ansteuerung einer zwei-achsigen Spiegelablenkeinheit
- Ausgänge zur Ansteuerung einer drei-achsigen Schrittmotorsteuerung
- Schnittstelle zur Handsteuerbox
- Galvanische Trennung einer RS-232-Schnittstelle
- Ausführung als ISA[1]-Steckkarte im Inneren des PC-Gehäuses
- Flexible Adressierung im IO-Bereich des ISA-Busses

Bild 6.2 zeigt die Lage der Stecker an der IO-Karte.

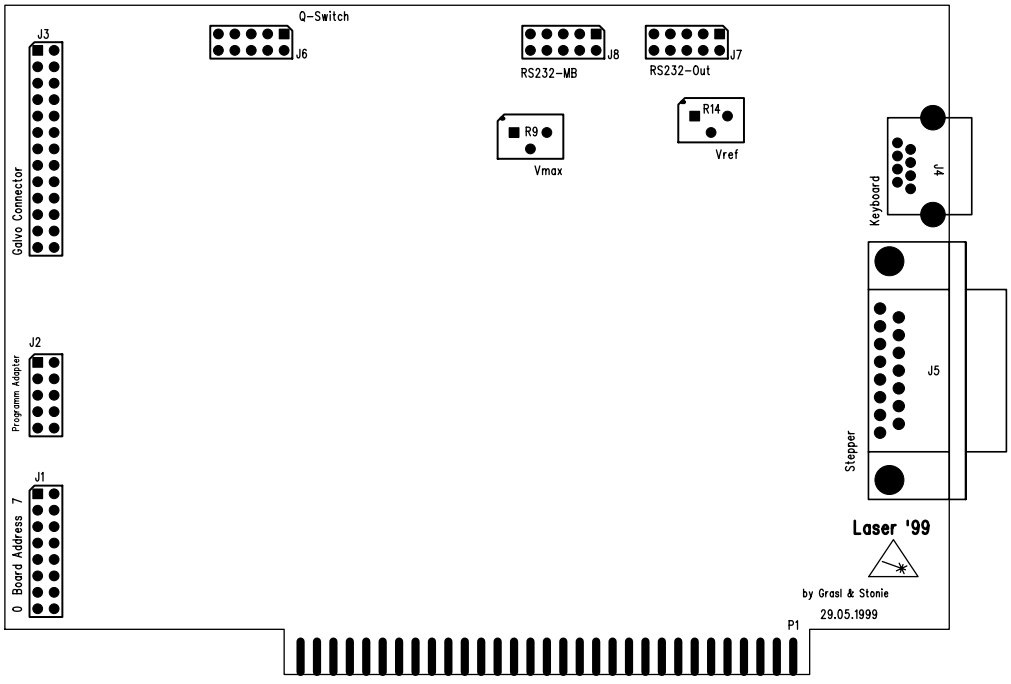



Abbildung 6.2: Lage der Stecker und Regler

Die Belegung der Stecker ist wie folgt:

[1]Industrie Standard Architecture

| Nr. | Name | Pin | Belegung |
|---|---|---|---|
| J1 | Board Address | 1 | A11 |
| | | 3 | A10 |
| | | 5 | A09 |
| | | 7 | A08 |
| | | 9 | A07 |
| | | 11 | A06 |
| | | 13 | A05 |
| | | 15 | A04 |
| | | 2,4,6,8 | GND |
| | | 10,12,14,16 | GND |
| J2 | Altera Programm Adapter | 1 | TCK |
| | | 2 | GND |
| | | 3 | TDO |
| | | 4 | Vcc |
| | | 5 | TMS |
| | | 6 | NC |
| | | 7 | NC |
| | | 8 | NC |
| | | 9 | TDI |
| | | 10 | GND |
| J4 | Keyboard Connector | 1 | NC |
| | | 2 | Vcc |
| | | 3 | SCL |
| | | 4 | GND |
| | | 5 | SDA |
| | | 6 | GND |
| | | 7 | NC |
| | | 8 | -12V |
| | | 9 | NC |
| | | 10 | NC |
| J6 | Q-Switch Connector | 1 | GND |
| | | 2 | NC |
| | | 3 | NC |
| | | 4 | NC |
| | | 5 | CLK |
| | | 6 | Killer Pulse |
| | | 7 | RF Fault |
| | | 8 | PA High Temperature Fault |
| | | 9 | High Temperature Fault |
| | | 10 | NC |

| Nr. | Name | Pin | Belegung |
|---|---|---|---|
| J5 | Stepper Connector | 1 | Step X Collector |
| | | 2 | Step X Emitter |
| | | 3 | Direction X Collector |
| | | 4 | Direction X Emitter |
| | | 5 | Step Y Collector |
| | | 6 | Step Y Emitter |
| | | 7 | NC |
| | | 8 | NC |
| | | 9 | NC |
| | | 10 | Direction Y Collector |
| | | 11 | Direction Y Emitter |
| | | 12 | Step Z Collector |
| | | 13 | Step Z Emitter |
| | | 14 | Direction Z Collector |
| | | 15 | Direction Z Emitter |
| J3 | Galvo Connector | 1 | GND |
| | | 2 | GND |
| | | 3 | Strobe X |
| | | 4 | NC |
| | | 5 | D14 |
| | | 6 | D15 |
| | | 7 | D12 |
| | | 8 | D13 |
| | | 9 | D10 |
| | | 10 | D11 |
| | | 11 | D8 |
| | | 12 | D9 |
| | | 13 | GND |
| | | 14 | GND |
| | | 15 | Strobe Y |
| | | 16 | NC |
| | | 17 | D6 |
| | | 18 | D7 |
| | | 19 | D4 |
| | | 20 | D5 |
| | | 21 | D2 |
| | | 22 | D3 |
| | | 23 | D0 |
| | | 24 | D1 |
| | | 25 | NC |
| | | 26 | NC |

| Nr. | Name | Pin | Belegung |
|---|---|---|---|
| J7 | external RS232 | 1 | NC |
| | | 2 | RXD |
| | | 3 | TXT |
| | | 4 | NC |
| | | 5 | GND |
| | | 6 | DSR |
| | | 7 | NC |
| | | 8 | CTS |
| | | 9 | NC |
| | | 10 | NC |
| J8 | internal RS232 | 1 | NC |
| | | 2 | RXD |
| | | 3 | TXT |
| | | 4 | connected to Pins 6 and 8 |
| | | 5 | GND |
| | | 6 | connected to Pins 4 and 8 |
| | | 7 | NC |
| | | 8 | connected to Pins 4 and 6 |
| | | 9 | NC |
| | | 10 | NC |

Folgende Register wurden implementiert, um die verschiedenen Funktionen anzusprechen:

| Offset | R/W | Name | Bit | Funktion |
|---|---|---|---|---|
| BASE+0 | W | Q_SW_CTRL | 0 | Q-Switch Killer-Puls Unterdrückung |
| BASE+0 | R | B_STATUS | 0 | Q-Switch RF Fault |
| | | | 1 | Q-Switch PA High Temperature Fault |
| | | | 2 | Q-Switch High Temperature Fault |
| | | | 3 | $I^2C$-Busy |
| | | | 4 | $I^2C$-Error |
| BASE+1 | W | Q_SW_VOLT | 7-0 | Amplitude des Q-Switch CLK Signals |
| BASE+2 | W | GALVO_LSB | 7-0 | Niederwertiges Byte für Galvosteuerung am Stecker J3 |
| BASE+3 | W | GALVO_MSB | 7-0 | Höherwertiges Byte für Galvosteuerung am Stecker J3 |
| BASE+4 | W | Q_SW_HIGH_LSB | 7-0 | Niederwertiges Byte der High-Phase für Q-Switch Frequenzerzeugung |
| BASE+5 | W | Q_SW_HIGH_MSB | 7-0 | Höherwertiges Byte der High-Phase für Q-Switch Frequenzerzeugung |

| Offset | R/W | Name | Bit | Funktion |
|---|---|---|---|---|
| BASE+6 | W | STEPPER_CTRL | 0 | Step X |
| | | | 1 | Dir X |
| | | | 2 | Step Y |
| | | | 3 | Dir Y |
| | | | 4 | Step Z |
| | | | 5 | Dir Z |
| | | | 6 | Galvo Strobe X |
| | | | 7 | Galvo Strobe Y |
| BASE+7 | W | I2C_ADDRESS | 7-1 | 7 Bit Adresse für $I^2C$-Datenverkehr |
| | | | 0 | Richtungsangabe für $I^2C$-Bus<br>1: lesen, 0: schreiben |
| BASE+8 | R/W | I2C_DATA | 7-0 | Datenregister für $I^2C$-Datenverkehr |
| BASE+9 | W | Q_SW_LOW_LSB | 7-0 | Niederwertiges Byte der Low-Phase für Q-Switch Frequenzerzeugung |
| BASE+10 | W | Q_SW_LOW_MSB | 7-0 | Höherwertiges Byte der Low-Phase für Q-Switch Frequenzerzeugung |

Wird ein Wert in ein Register geschrieben, wirkt sich dieser sofort auf die Ausgänge der Karte aus. Lediglich die $I^2C$-Übertragung wird erst beim Beschreiben des Adressregisters aktiv. Will man eine $I^2C$-Übertragung initialisieren, muss man folgendermaßen vorgehen:

- erst prüfen, ob das $I^2C$-Busy-Bit gelöscht ist, da man sonst in eine bereits laufende Übertragung störend eingreifen könnte
- bei einem Schreibzugriff die Daten in das Datenregister laden
- die 7-Bit-Adresse um eine Stelle nach links schieben und das Bit 0 entsprechend der Übertragungsrichtung setzen
- die so gewonnenen 8 Bits in das $I^2C$-Datenregister schreiben, um die Übertragung zu starten
- bei einem Lesezugriff warten, bis das $I^2C$-Busy-Bit im Statusregister zurückgesetzt wird. Danach das $I^2C$-Error-Bit im Statusregister abfragen. Wenn es nicht gesetzt ist, befinden sich die empfangenen Daten im $I^2C$-Datenregister.
- bei einem Schreibzugriff wird die Übertragung selbsttätig abgeschlossen, es kann mit dem Programmablauf fortgefahren werden.

Ist im Statusregister das $I^2C$-Busy-Bit gesetzt, darf nicht auf die $I^2C$-Register zugegriffen werden, da sonst Fehler in der Übertragung auftreten können.

### 6.1.2 Der Schaltplan

Der Schaltplan der IO-Karte (Abbildung A.1.1) lässt folgende Bausteine erkennen:

- Die Anbindung an den ISA-Bus
- Die Adressdekodierung
- Den EPLD[2] zur Erzeugung der einzelnen Chipselect- und Q-Switch-Signale
- Den 16-Bit-Bus für die Strahlablenkung
- Den Verstärker für das Q-Switch-Signal
- Das Interface zu den Schrittmotoren und der Handsteuerbox
- Die Schaltung zur galvanischen Trennung einer RS-232-Schnittstelle

#### Der ISA-Bus

Der ISA-Bus ist ein 16-Bit-Bus bestehend aus 16 Datenleitungen, 20 Adressleitungen und einigen Steuersignalen. Der Bustakt beträgt 7 MHz, es können also maximal 14 Megabytes Daten pro Sekunde übertragen werden. Das ist für heutige Verhältnisse nicht sehr viel, andere Bussysteme können einige 100 MB Daten pro Sekunde bewältigen. Im Hinblick auf den Mehraufwand der bei der Verwendung eines anderen Bussystems einerseits und der im gegebenen Fall benötigten niedrigen Datenrate andererseits, ist hier der unkomplizierte ISA Bus eindeutig den anderen nach technischen Daten überlegenen Bussystemen vorzuziehen.

Im einzelnen werden folgende Signale verwendet:

| Name | Beschreibung |
|---|---|
| DB<15:0> | Datenbus 16 Bit breite |
| AB<19:0> | Adressbus 20 Bit breite, damit kann bis zu 1 Megabyte adressiert werden, es werden die unteren 12 Bits verwendet |
| /IORD | IO-Read Signal, low aktiv, die steigende Flanke markiert den Zeitpunkt der Datenübernahme |
| /IOWR | IO-Write Signal, low aktiv, die steigende Flanke markiert den Zeitpunkt der Datenübernahme |
| /AEN | Addresslatch Enable, low aktiv, so lange dieses Signal aktiv ist, liegen am Adressbus gültige Daten an |
| Reset | Reset Signal, dient zum Zurücksetzen der Hardware in einen definierten Zustand |

[2]Eraseable Programmable Logic Device, ein auf Flash-Speicher basierender programmierbarer Logik-Baustein

Die Übertragung der Daten erfolgt nach dem Timingdiagramm aus Abbildung 6.3

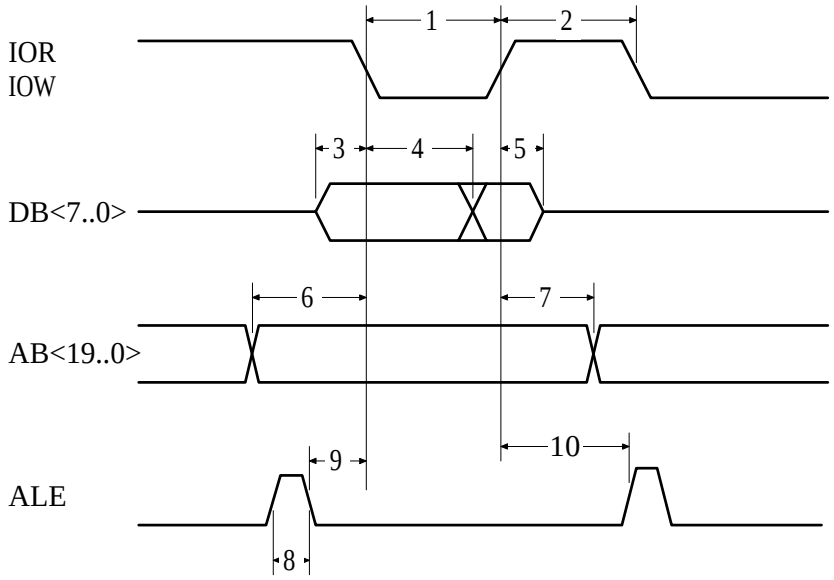



Abbildung 6.3: Schreib/Lesezugriff am ISA Bus

Die zur Abbildung 6.3 gehörende Parametertabelle:

| Ref. | Beschreibung | Min | Max |
|---|---|---|---|
| 1 | IOx asserted | 165 ns | |
| 2 | IOx deasserted | 170 ns | |
| 3 | Write data setup | 22 ns | |
| 4 | Read data access | | 110 ns |
| 5 | Read data hold | 0 ns | |
| 5 | Write data hold | 25 ns | |
| 6 | AB Setup to IOx¡ | 102 ns | |
| 7 | AB hold | 53 ns | |
| 8 | ALE Pulswidth | 61 ns | |
| 9 | ALE deasserted to IOx asserted | 47 ns | |
| 10 | ALE asserted from IOx deasserted | 46 ns | |

### Adressdekodierung

Der IO-Bereich des ISA-Busses erstreckt sich von 0x000 bis 0xfff. Diesen Adressbereich müssen sich alle ISA-Karten teilen. Unsere Karte benötigt einen Bereich mit 16 Adressen, der exklusiv zur Verfügung gestellt werden muss. Eingestellt wird dieser an der Steckerleiste J1. Diese ist so ausgelegt, dass mittels handelsüblicher Jumper[3] der gesamte Adressbereich in Blöcken von je 16 Bytes überstrichen werden kann. Der so festgelegte Bereich wird bei Aktivierung des ALE-Signals, welches anzeigt, ob eine gültige Adresse am Bus anliegt, mit dieser Adresse verglichen. Wird eine Übereinstimmung festgestellt, wird der Ausgang des Adressdecoders aktiviert.

### EPLD

Herzstück der Platine ist der EPLD "EPM7128" der Firma Altera. Er besteht aus 128 Makrozellen, die unter anderem je ein Flipflop enthalten und einer umfangreichen Verknüpfungsmatrix, welche die einzelnen Makrozellen miteinander verbindet. Die spezielle Kombination dieser Elemente ist in der Lage, die unterschiedlichsten logischen Verknüpfungen aufzubauen. Die Konfiguration der Verbindungsmatrix und den Makrozellen wird beim Programmiervorgang im chipinternen Flash-Speicher abgelegt und bestimmt so das Betriebsverhalten des Bausteins. Für eine genauere Beschreibung dieses Bausteins möchte ich auf das Datenblatt [8] verweisen.

Zum Programmieren dieses Bausteins stellt Altera eine eigene Entwicklungsumgebung kostenlos zur Verfügung. In ihr sind ein grafischer und ein textorientierter Editor, ein Compiler, der Simulator und die Programmiersoftware vereint.

Öffnet man die oberste Ebene der Design-Files (siehe Abbildung 6.4), sieht man ähnlich wie bei einem Schaltplan die Struktur des Designs. Als erstes möchte ich mich dem "chipsel"-Block zuwenden. Hier werden mit Hilfe des externen Adressdecoders die Chipselect-Signale gebildet. Dabei erzeugt der externe Komparator ein Signal, das aktiviert wird, wenn die Adressbits 4-11 jenen mit dem Jumperblock eingestellten Bitmuster entsprechen. Die unteren 4 Bits des Adressbusses werden innerhalb dieses "chipsel"-Blocks mit dem /IORD- und /IOWR-Signal vom ISA Bus verknüpft, und so die benötigten high aktiven Chipselect-Signale generiert. Der Quellcode für den "chipsel"-Block befindet sich im Anhang Kapitel B.1.4.

Darunter befindet sich ein 8-fach-D-Flipflop, das als Register für die Ausgänge der Schrittmotoransteuerung genutzt wird. Liegt die richtige Adresse am ISA-Bus an und wird das /IOWR-Signal aktiv, generiert das chipsel-Modul einen Impuls, auf Grund dessen die Daten, die am Datenbus

[3]= Kurzschlussbrücke

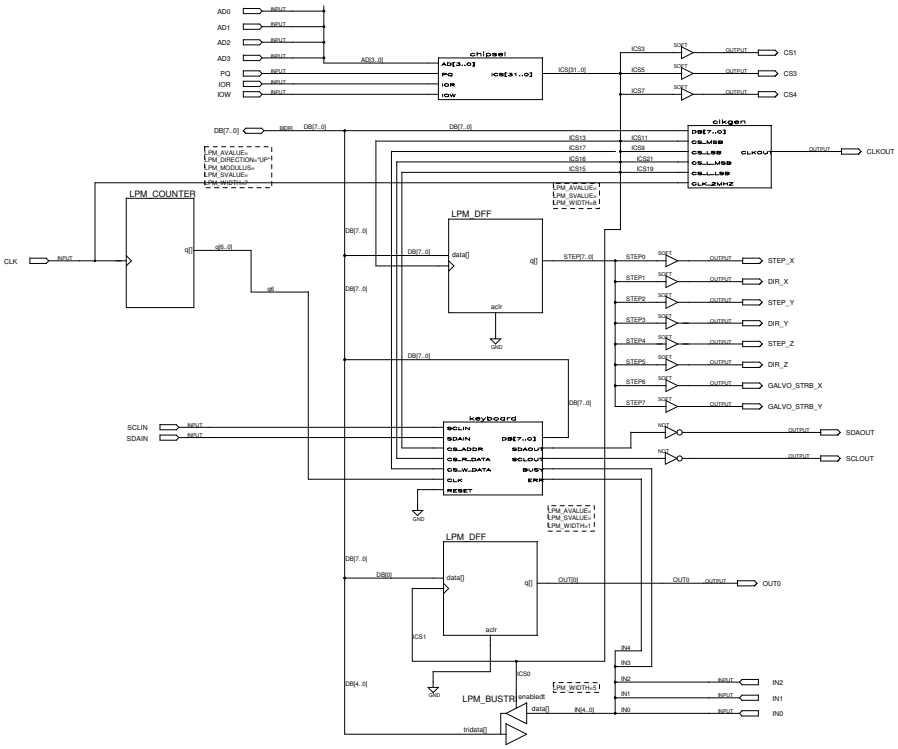



Abbildung 6.4: EPLD, oberste Designebene

anliegen, in das Flipflop übernommen werden und ab diesen Zeitpunkt an den Chipausgängen zur Verfügung stehen.

Rechts daneben befindet sich der “clkgen”-Block (siehe Abbildung 6.5). Hier wird das Taktsignal für den Q-Switch generiert. Der verwendete Q-Switch benötigt ein Taktsignal, das zwischen 100 Hz und 100 kHz liegt und dessen Puls-Pausenverhältnis von 0 bis 100 % einstellbar ist.

Realisiert wird diese Funktion mittels eines ladbaren 15 Bit Zählers, der abwärts zählt und bei einem Überlauf ein Toggle-Flipflop umschaltet (siehe Abbildung 6.6). Das Ausgangssignal setzt sich also aus zwei Zählperioden zusammen. Die im Schaltplan oberen zwei 8-Bit-Flipflops speichern den Zählwert, der der Länge der Low Phase des Ausgangssignals entspricht, die unteren zwei 8-Bit-Flipflops speichern den Zählwert, der der Dauer der High Phase entspricht. Das Umschalten zwischen diesen beiden Zählwerten übernimmt ein Bus-Multiplexer, der je nach Ausgangssignal des T-Flipflops die oberen oder unteren Zählwertspeicher mit dem Ladewerteingang des Zähler verbindet. Hat der Zähler Null erreicht, ereignet sich beim nächsten Takt ein Unterlauf und das oberste Zählerbit wird gesetzt.

Dies hat zweierlei Folgen: Erstens schaltet das T-Flipflop um und zweitens wird der Ladeeingang des Zählers aktiviert, was ihn dazu veranlasst, beim nächsten Taktzyklus den Zählerwert neu zu laden, wobei das oberste Zähler-

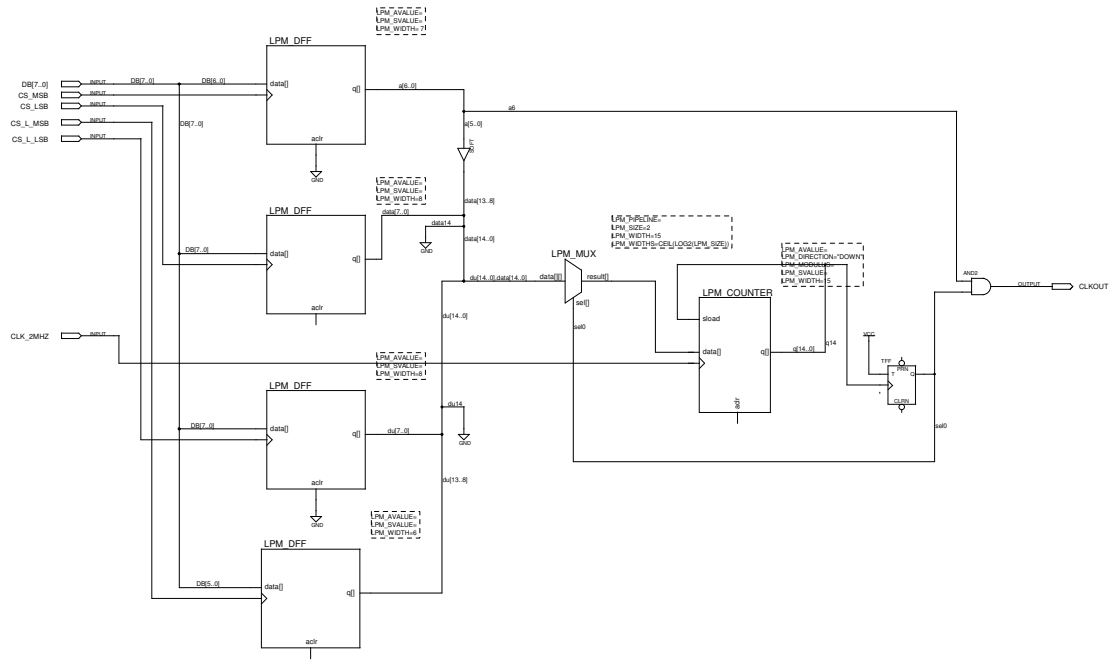



Abbildung 6.5: EPLD, Signalgenerator für die Q-Switch Ansteuerung (“clkgen”-Modul)

bit wieder zurückgesetzt wird.

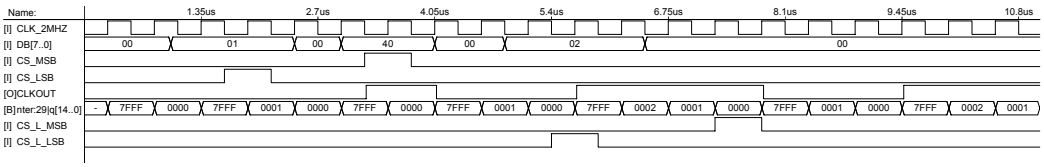



Abbildung 6.6: EPLD, Simulation des “clkgen”-Moduls

Am Ausgang der Schaltung befindet sich ein Und-Gatter, mit dessen Hilfe der Ausgang permanent auf Low gesetzt werden kann. Dies hat den Vorteil, dass der Q-Switch auch als Shutter, der den Laserstrahl mit geringer Verzögerungszeit schaltet, eingesetzt werden kann.

Um mit der Handsteuerbox kommunizieren zu können, wurde das “keyboard”-Modul entwickelt (Abbildung 6.7).

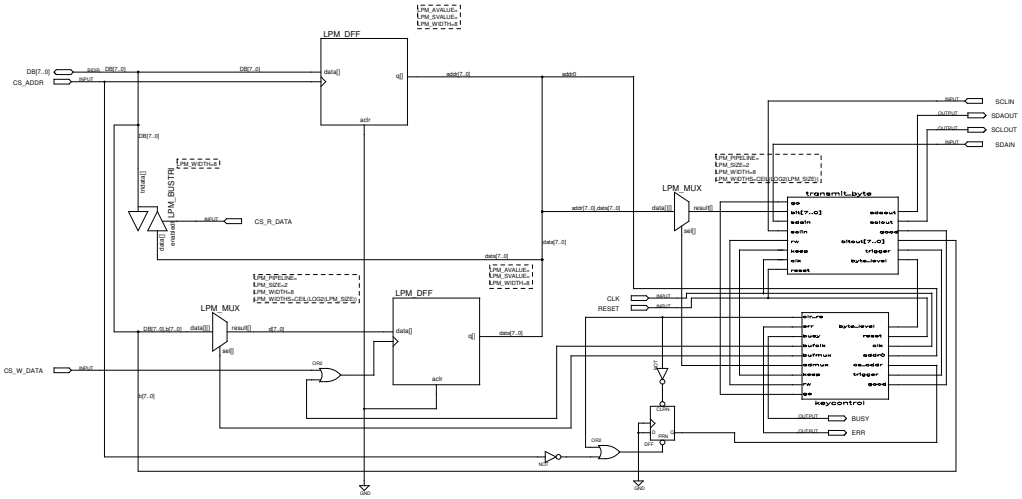



Abbildung 6.7: EPLD, Schnittstellenmodul zur Handsteuerbox (“keyboard”-Modul)

Es stellt eine $I^2C$-Schnittstelle zur Verfügung. Der $I^2C$-Bus ist eine bidirektionale synchrone serielle Schnittstelle, die 2 Leitungen verwendet - eine bidirektionale Datenleitung und eine unidirektionale Taktleitung. Bei der $I^2C$-Kommunikation gibt es immer genau einen Busmaster, der den Takt vorgibt und mehrere Slaves, die vom Master abgefragt werden können. Ein Slave kann von sich aus keinen Busverkehr initialisieren, dies kann nur der Master. Jeder Slave hat eine eindeutige Adresse am Bus, die ihm hardwaremäßig zugeteilt ist.

Ein $I^2C$-Datenrahmen hat folgendes Format (siehe Abbildung 6.8):

Wenn der Master mit der Übertragung beginnt, generiert er erst eine Startbedingung. Diese, wie auch die Stoppbedingung, ist dadurch gekennzeichnet, dass die Datenleitung ihren Pegel ändert, während die Taktleitung High-Pegel führt. Normalerweise dürfte die Datenleitung nur dann ihren Zustand ändern, wenn die Taktleitung auf Low liegt. Danach werden, wie in der Abbildung 6.8 gezeigt, 7 Adressbits und ein Schreib/Lesebit hinausgeschoben. Jetzt generiert der Empfänger ein Acknowledge-Bit, wobei der Takt vom Master vorgegeben wird. Bleibt dieses aus, wird die Übertragung mit einem Fehler abgebrochen. Je nachdem, ob schreiben oder lesen, wird die Richtung der Datenleitung umgekehrt, und es werden weitere 8 Bits verschoben und ein Acknowledge-Bit vom Empfänger erzeugt. Dieser Vorgang der Datenübertragung kann sich nun öfter wiederholen, bis die Stoppbedingung generiert und damit der Rahmen beendet wird. Für eine genauere Beschreibung verweise ich auf die $I^2C$-Dokumentation von Philips Semiconducters [7] und der

Einführung in [3].

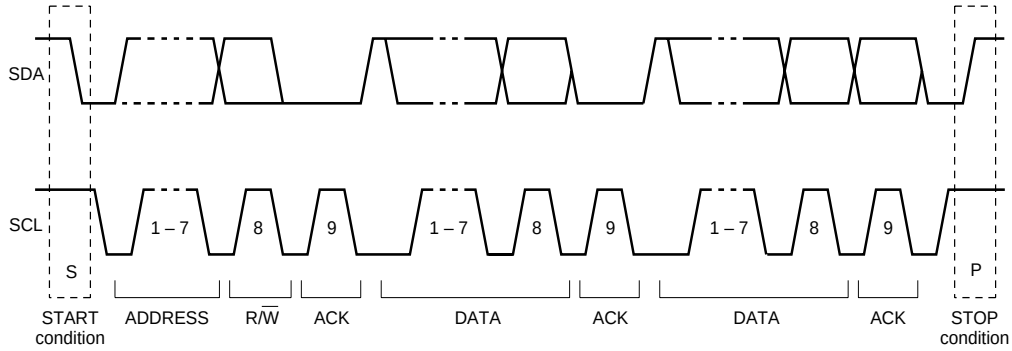



Abbildung 6.8: I$^2$C-Busrahmen

Genau dieses Bitmuster wird im Schnittstellenmodul ("keyboard") erzeugt. Mit Hilfe mehrere State-Maschinen[4], die in 3 Ebenen aufgebaut sind, können Daten gesendet und empfangen werden. Die unterste Ebene verschiebt ein Bit (siehe Abbildung 6.9) über den Bus.

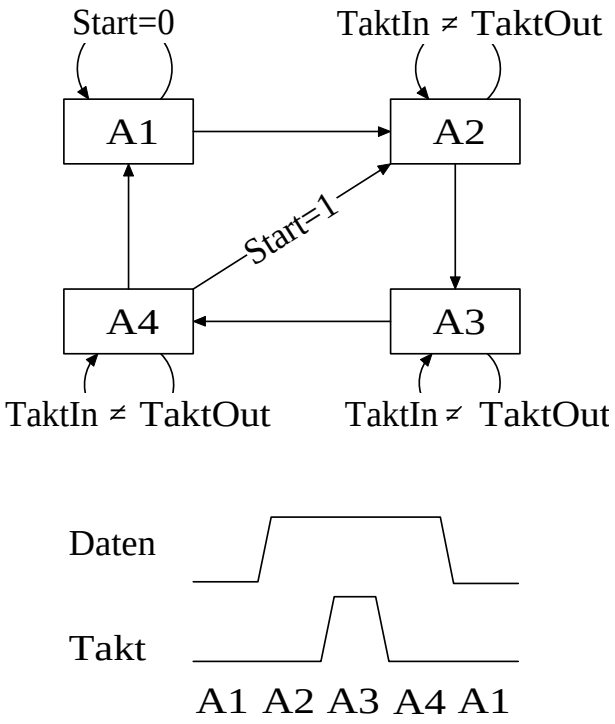



Abbildung 6.9: EPLD, State-Maschine zur Übertragung eines Bits

[4]Eine State-Maschine, oder Zustandsautomat, ist eine taktgesteuerte Zustandsmaschine, die aufgrund der Eingangssignale ihren inneren Zustand nach festgelegten Regeln und in Abhängigkeit ihres aktuellen Zustandes zu bestimmten Zeiten ändert und dann abhängig vom jeweiligen Zustand Ausgangssignale generiert.

Die darüberliegende Ebene “transmit_byte” verwendet nun dieses Modul, um ein Byte auf den Bus zu schieben. Die State-Maschine ist in Abbildung 6.10 dargestellt, die Simulation des Modules in Abbildung 6.11.

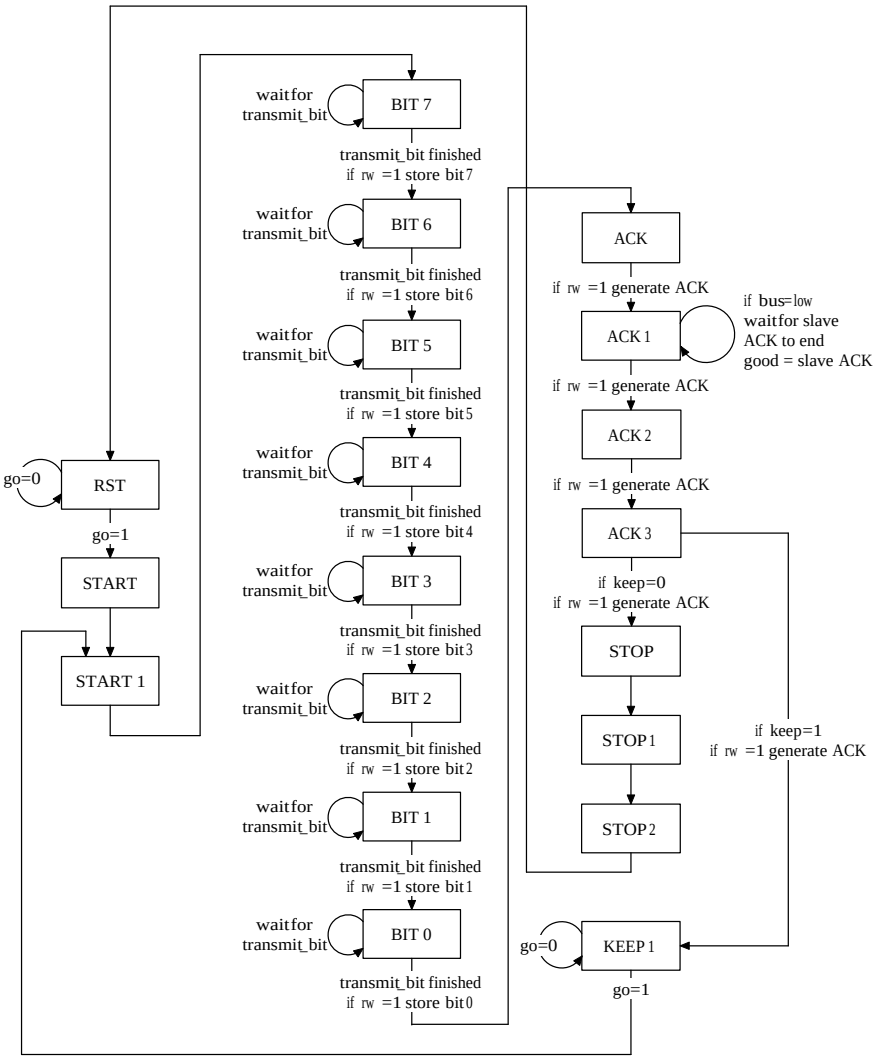



Abbildung 6.10: EPLD, State-Maschine zur Übertragung eines Bytes

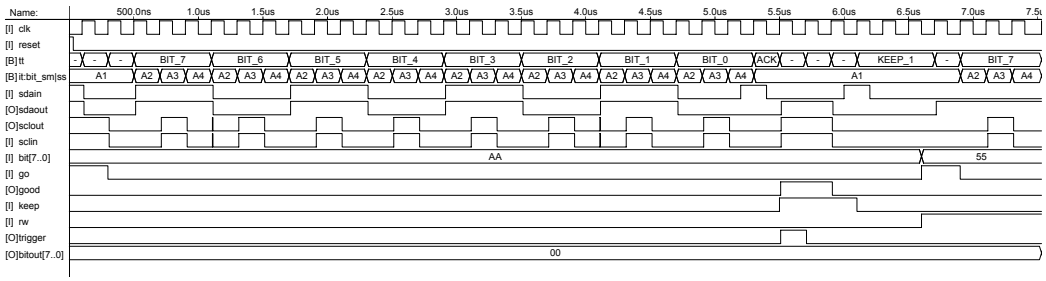



Abbildung 6.11: EPLD, Simulation der Übertragung eines Bytes

Gesteuert werden diese State-Maschinen von dem Modul “Keycontrol” (Abbildung 6.12).

Erfolgt ein Schreibzugriff auf das Adressregister, wird die Datenübertragung gestartet. Es wird erst die Adresse auf den Bus geschoben. Während dies geschieht, wird auf das Acknowledge-Bit am Bus gewartet. Trifft dieses nicht zur vorgesehenen Zeit ein, wird das Error-Bit im Status-Register gesetzt und wieder in den Ruhezustand zurückgekehrt. Wird das Acknowledge-Bit wie vorgesehen empfangen, so wird unterschieden, ob gelesen oder geschrieben werden soll.

Beim Lesen werden 8 Taktzyklen erzeugt und die Daten am Bus in einen Speicher geschrieben. Anschließend wird ein Acknowledge-Bit an den Slave gesendet, gefolgt von der Stoppbedingung. Abschließend werden die Daten im Speicher in das Datenregister verschoben und das Busy-Bit zurückgesetzt.

Beim Schreiben hingegen wird das Daten-Byte aus dem Datenregister in den Speicher der State-Maschine geladen. Diese Daten werden entsprechend dem Protokoll auf den Bus geschoben, dann das Acknowledge des Slaves ausgewertet, und das Error-Bit im Status Register dementsprechend gesetzt. Zum Abschluss wird eine Stoppbedingung generiert und das Busy-Bit zurückgesetzt. Dieser Vorgang wird in der Simulation, gezeigt in Abbildung 6.13, dargestellt.

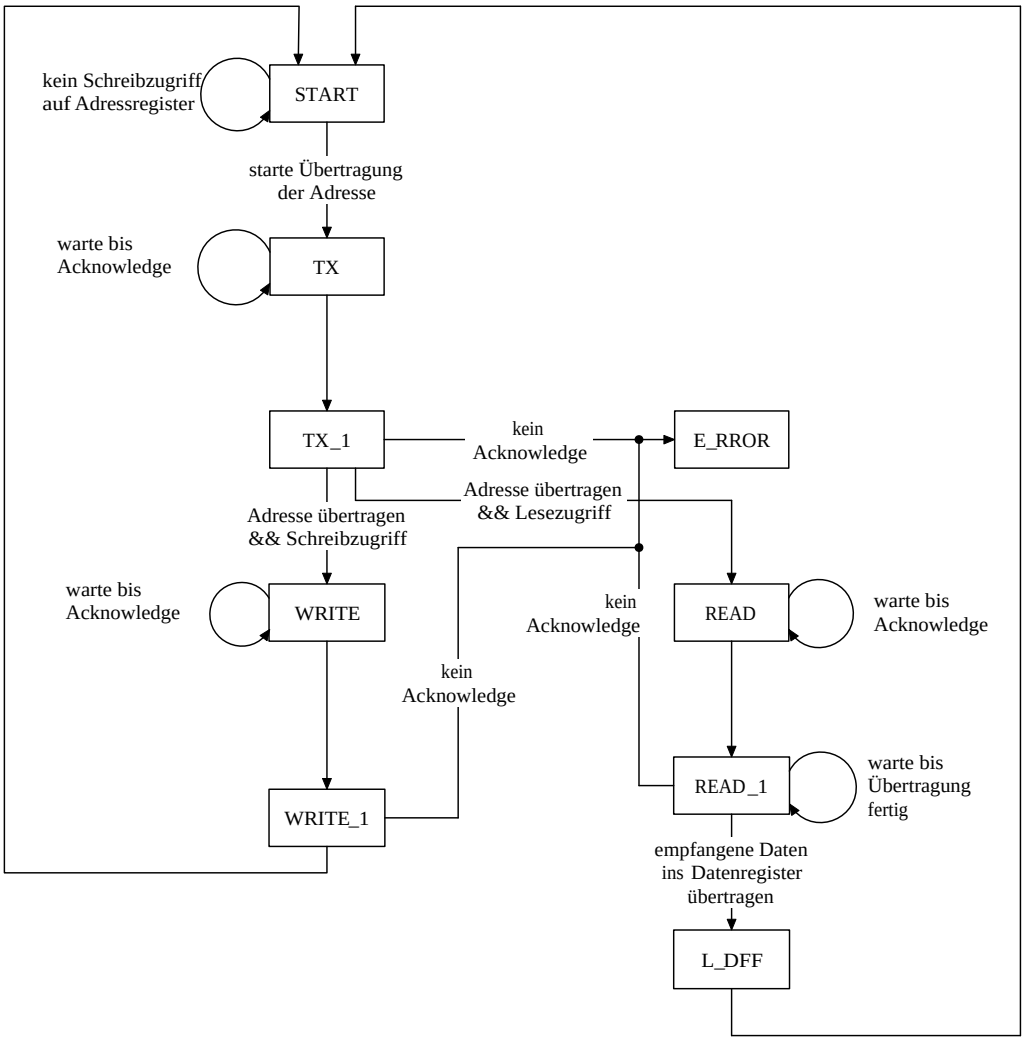



Abbildung 6.12: EPLD, State-Maschine zur Durchführung einer Übertragung

Den Abschluss bildet die Schnittstelle zu den Q-Switch Statusleitungen. Ein einzelnes Flipflop speichert den Zustand des als Killerpuls-Freigabe benutzten Ausgangs. Die Funktion ist gleich wie die der oben erwähnten Schrittmotoranbindung. Die Statusleitungen, die Auskunft über den Betriebszustand des Q-Switch geben, sind mittels eines Tri-State Buffers an den Datenbus angekoppelt. Wird die Enable-Leitung dieses Buffers von dem chipsel Modul aktiviert, schaltet dieser Buffer die Signale vom Q-Switch auf den Datenbus, der sich, da gelesen wird, zu der Zeit im hochohmigen Zustand befindet. Bei Deaktivierung der Enable-Leitung wird der Bufferausgang in den hochohmigen Zustand versetzt, bevor der Datenbus denselben verlässt.

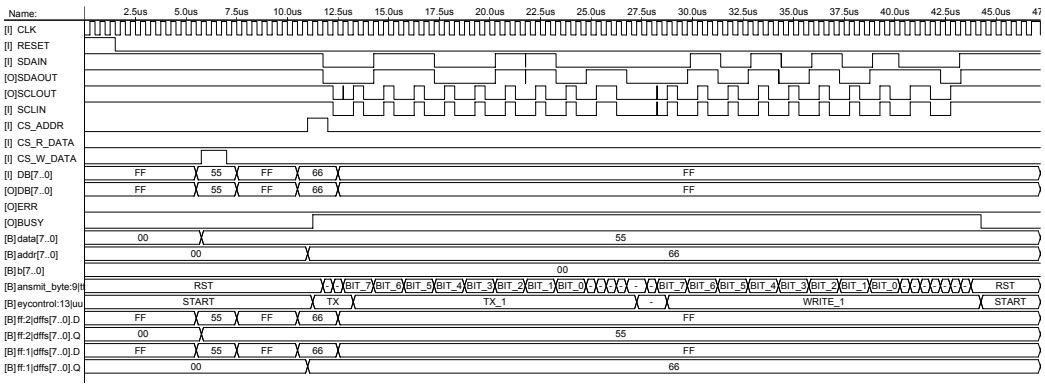



Abbildung 6.13: EPLD, Simulation einer Übertragung am I²C-Bus

### Galvo-Bus

Der Galvo-Bus ist ein 16-Bit-Bus mit getrennten Stobe-Leitungen für den X- und Y-Galvo. Die beiden Busbuffer sind jeweils 8 Bit breit und verfügen über ein low aktives Chipselect. Wird dieses aktiv, übernehmen die Buffer das am Eingang anliegende Datenwort, speichern es in Flipflops ab und geben es am Ausgang aus. Die zum Galvobus dazugehörigen Strobe-Signale werden im EPLD gespeichert.

### Q-Switch Signalanpassung

Der von uns verwendete Q-Switch Treiber verfügt über einen 50-Ohm-Eingang und hat einen Eingangsamplitudenbereich von 0 bis 10 Volt. Darum muss das vom EPLD kommende Signal verstärkt werden. Mit den DA-Wandler, der einen externen Datenbuffer besitzt, wird die Amplitude des Ausgangssignals eingestellt und mit den nachgeschalteten Operationsverstärkerstufen soweit verstärkt, dass eine am Ausgang angebrachte 50-Ohm-Last angesteuert werden kann. Da der DA-Wandler ein differenzielles Ausgangssignal liefert, wird dieses durch die um den OPV2-A aufgebaute Verstärkerstufe in eine auf Masse bezogene Spannung transformiert. Diese wird mit dem Transistor Q1 mit dem vom EPLD kommenden Q-Switch-CLK-Signal moduliert, anschließend durch einen Impedanzwandler stabilisiert und mit OPV1-A auf 10 Volt begrenzt. Zuletzt wird mit den um OPV1-B aufgebauten Leistungsverstärker das Signal verstärkt. Bei der Inbetriebnahme der Schaltung wird erst ein Spannungsmessgerät an den Emitter von Transistor Q6 angeschlossen und die gegen Masse gemessene Spannung mit dem Potentiometer R14 auf 10 Volt justiert. Danach stellt man per Software die maximale Amplitude des DA-Wandlers ein und gleicht mit einem Oszilloskop, das an Pin 5 vom Stecker J6 angeschlossen ist und gegen Masse misst, die Amplitude des Rechtecksignals auf 10 Volt ab. Damit ist die Justierung der Schaltung abgeschlossen.

### Stepper-Control, RS232

Als Vorsichtsmaßnahme wurden die Signale zur Schrittmotorsteuerung und eine RS-232 Schnittstelle des Rechners mittels Optokopplers galvanisch abgetrennt. Als Besonderheit ist hier zu erwähnen, dass die Optokoppler der RS-232 Schnittstelle direkt von derselben versorgt werden. Dies sollte bei der geringen Stromaufnahme dieses Optokopplertyps kein Problem darstellen.

### Handsteuerbox-Schnittstelle

Da die Hansteuerbox-Schnittstelle bidirektional über zwei Leitungen betrieben wird und somit sich die Übertragungsrichtung der Daten laufend ändert, wurde die Schnittstelle mit Open-Collector-Transistorschaltungen realisiert. Die Leitung wird dabei von einem Pullup-Widerstand auf High gehalten, während mittels Durchschaltens des Transistors die Low-Phasen realisiert werden. Dieses System verhindert zuverlässig Kurzschlüsse auf den Leitungen.

## 6.2 Handsteuerbox

### 6.2.1 Funktionen der Handsteuerbox

Die Handsteuerbox wurde entwickelt, um dem Betreiber des Lasersystems die Möglichkeit einzuräumen, den Steuerrechner vom Laser örtlich getrennt aufzustellen. Sie soll weder Tastatur noch Bildschirm des Steuerrechners ersetzen, sondern als ein weiteres Bedienelement angesehen werden.

Die Handsteuerbox besitzt ein Display mit 16*2 Zeichen Anzeigekapazität, zwei Leuchtdioden und 14 Tasten. Die Kommunikation zum Steuerrechner wird über eine $I^2C$-Schnittstelle abgewickelt.
Da das verwendete Display den von Hitachi entwickelten Kontroller "HD44780" benutzt, der einen gewissen Standard in der Displayansteuerung gesetzt hat, und dieser auch bei anderen Displays eingesetzt wird, kann das hier verwendete Display ohne Änderungen an der Elektronik gegen ein größeres mit demselben Kontroller ausgetauscht werden.

Aus diesem Grund belegt das Display einen Großteil des $I^2C$-Adressraumes:

| Adresse | R/W | Funktion |
|---|---|---|
| 0x00 - 0x26 | W | Adressbereich der oberen Reihe des Displays |
| 0x27 | W | Adresse des Leuchtdiodenregisters. |
| 0x28 - 0x4F | W | Adressbereich der unteren Reihe des Displays |
| 0x20 | R | Einlesen der Tasten 0-7 |
| 0x21 | R | Einlesen der Tasten 8-13 |

### 6.2.2 Schaltplan

Aus dem Schaltplan des Handsteuergerätes (Abbildung A.2.1) ist ersichtlich, dass die gesamte Funktionalität des Gerätes im EPLD untergebracht wurde. Neben der obligatorischen Open-Collector-Anbindung an den $I^2C$-Bus ist im Schaltplan praktisch nur der EPLD erwähnenswert.

#### EPLD

Betrachtet man hier die oberste Ebene des Design Files (Abbildung 6.14), wird ähnlich wie beim Schaltplan ein einziges Modul hervorstechen. Im "display"-Modul liegt die gesamte Funktionalität der Handsteuerbox. Zwei State-Maschinen sind für den Datentransport per $I^2C$-Bus zuständig, eine dritte steuert den Zugriff auf das Display.

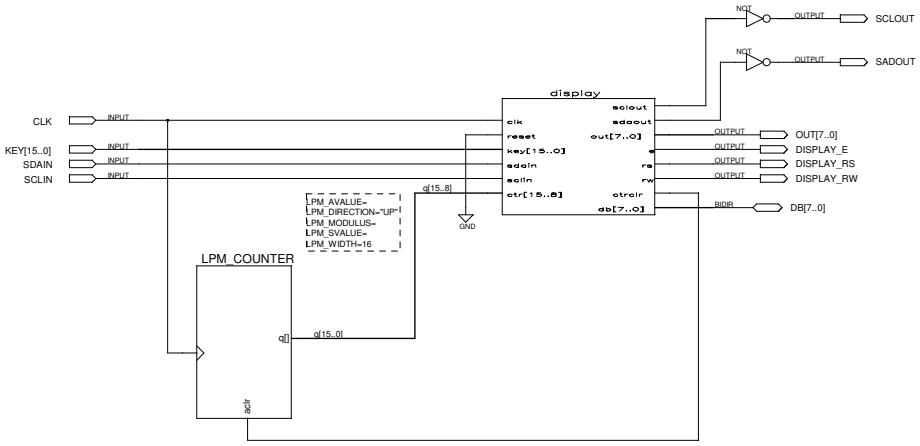



Abbildung 6.14: Oberste Designebene des EPLD im Handsteuergerät

Betrachten wir das I$^2$C-Modul, ist es von Vorteil von der untersten Ebene auszugehen. Dort findet sich das Modul "i2c_bit_pos" (Abbildung 6.15). Aufgabe dieses Moduls ist es, steigende und fallende Flanken auf der Daten- oder Taktleitung anzuzeigen.

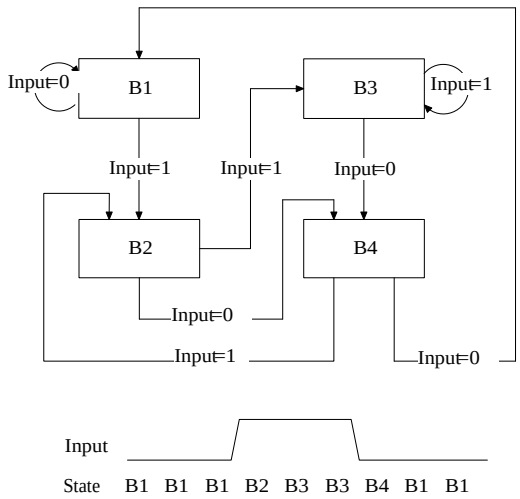



Abbildung 6.15: EPLD: "i$^2$c_bit_pos"-State-Maschine zur Erkennung der Signalflanken im Datenstrom

Diese Information wird vom Modul "i2c_start_stop" genutzt, um Start- und Stoppbedingung zu erkennen (Abbildung 6.16).

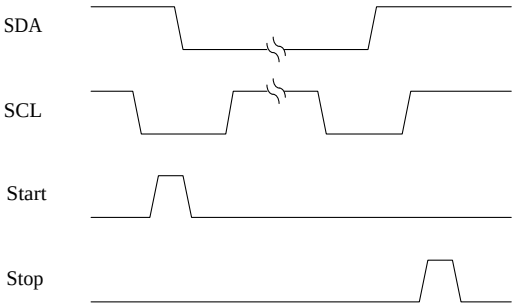



Abbildung 6.16: EPLD: “i$^2$c_start_stop”-Funktion zur Erkennung der Start- und Stopbedingungen im Datenstrom

Die State-Maschine in “i$^2$c_transmit_byte” ist zuständig für das Schreiben oder Lesen von 8 Bit und anschließender Generierung eines Acknowledge-Bits am Bus (Abbildung 6.17).

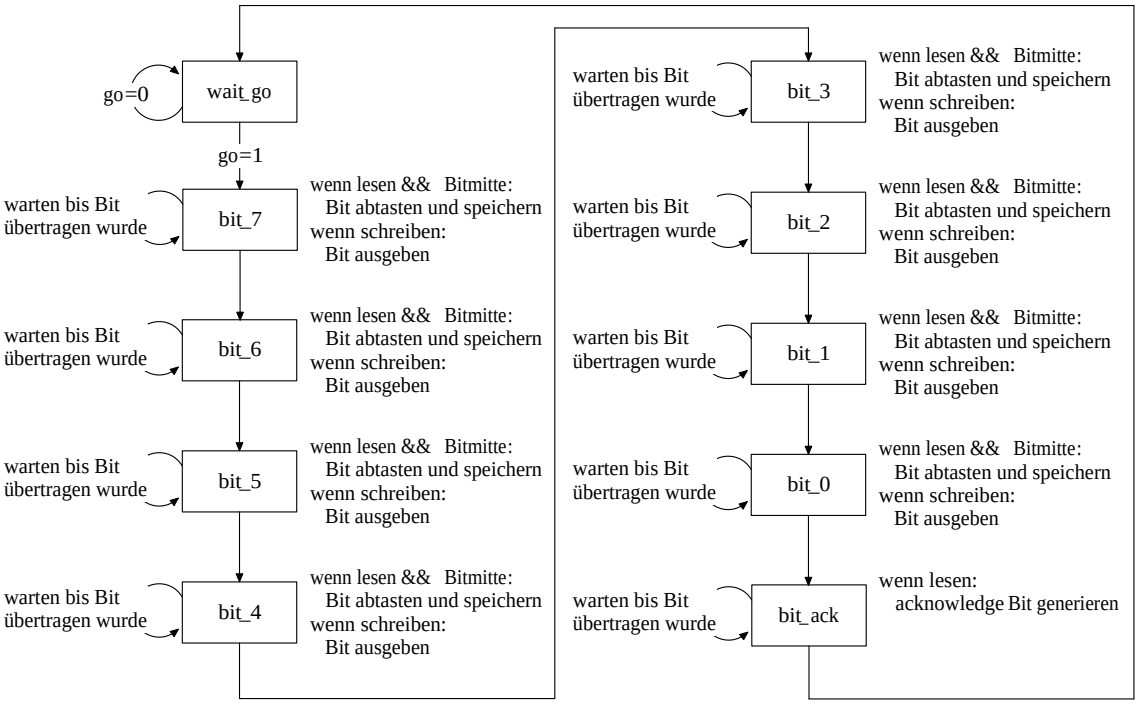



Abbildung 6.17: EPLD: “i$^2$c_transmit_byte”-State-Maschine, zuständig für die Übertragung der I$^2$C-Datenbits

Beim Lesen vom Bus wird mittels der Ausgabe von “i$^2$c_bit_pos” der Abtastzeitpunkt ermittelt, bei jedem Bit zu dieser Zeit abgetastet und der Abtastwert in einem Register gesichert. Beim Schreiben wird auf die von “i$^2$c_bit_pos” ermittelte Position im Datenstrom synchronisiert und die Daten Bit für Bit auf den Bus geschoben.

Gesteuert wird der I$^2$C-Buszugriff vom “i$^2$c”-Modul (Abbildung 6.18).

Dieses stellt die richtige Abfolge von Startbedingung, Übertragung und Stoppbedingung sicher. Dieses Modul alleine ist in der Lage, den I²C-Datenverkehr abzuwickeln. Es stellt die Funktionen des Busses für die darauf zugreifenden Module transparent dar, das heißt, es meldet einkommende Daten, signalisiert die Beendigung der Übertragung und liefert die Daten als 8-Bit-Wort der nächsten Instanz.

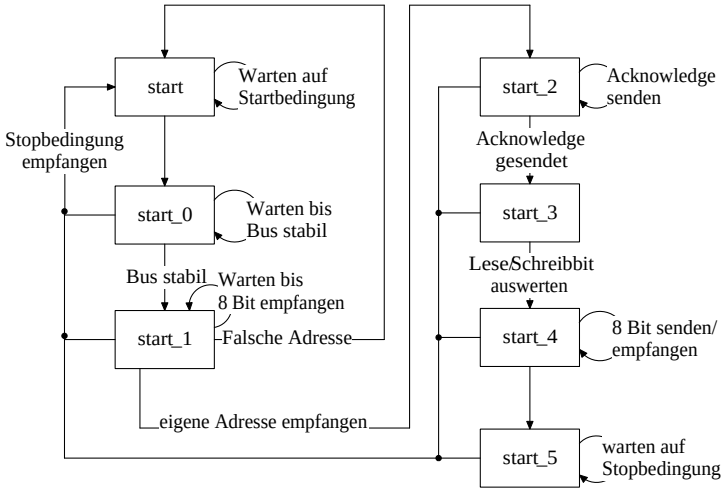



Abbildung 6.18: EPLD: "i²c"-State-Maschine, steuert den Kommunikationsablauf am I²C-Bus

Der Zugriff auf das Display ist nicht ganz so kompliziert aufgebaut. Es gibt nur 2 Funktionen, um mit dem Display zu kommunizieren. An der untersten Ebene ist eine Funktion angesiedelt, welche die Bereitschaft des Displays abwartet, neue Daten aufzunehmen (Abbildung 6.19).

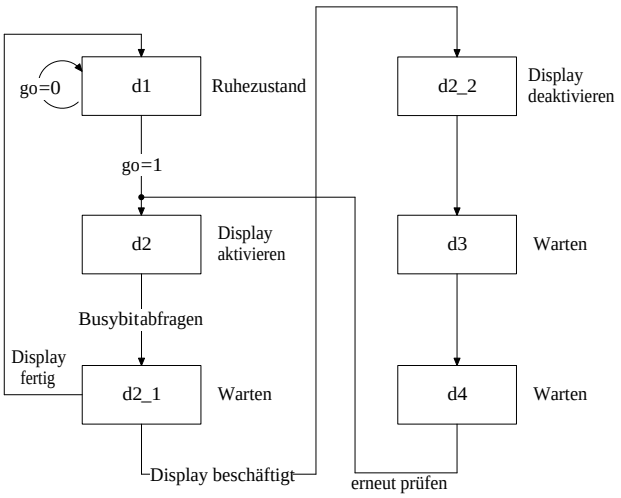



Abbildung 6.19: EPLD: "display_busy"-State-Maschine, erkennt, ob auf das Display zugegriffen werden darf

Ist das Display bereit für neue Daten, werden diese von der “display_out”-State-Maschine übertragen (Abbildung 6.20).

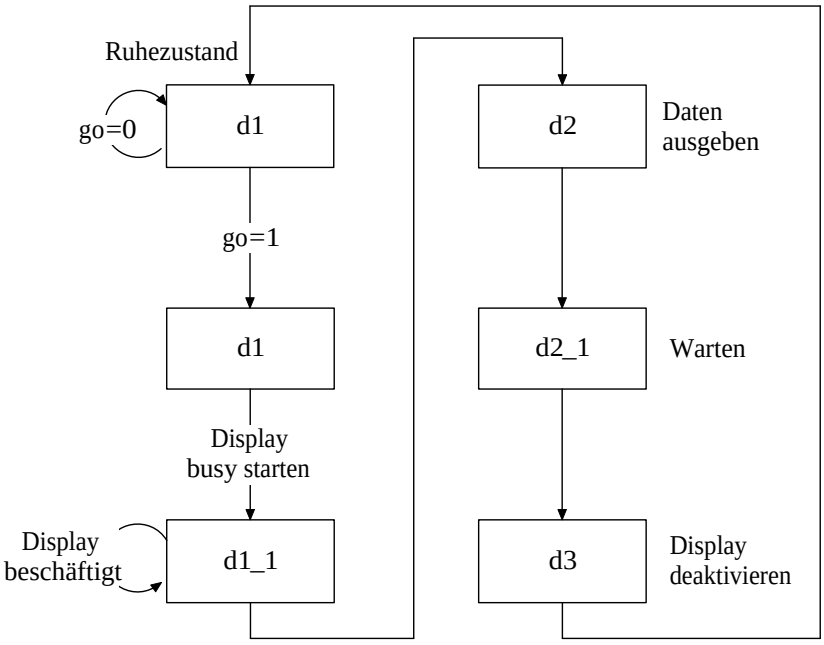



Abbildung 6.20: EPLD: “display_out”-State-Maschine zur Übertragung von Daten an das Display

Im Modul “display” werden schließlich die Funktionen von I$^2$C-Bus und Display vereint (Abbildung 6.21).

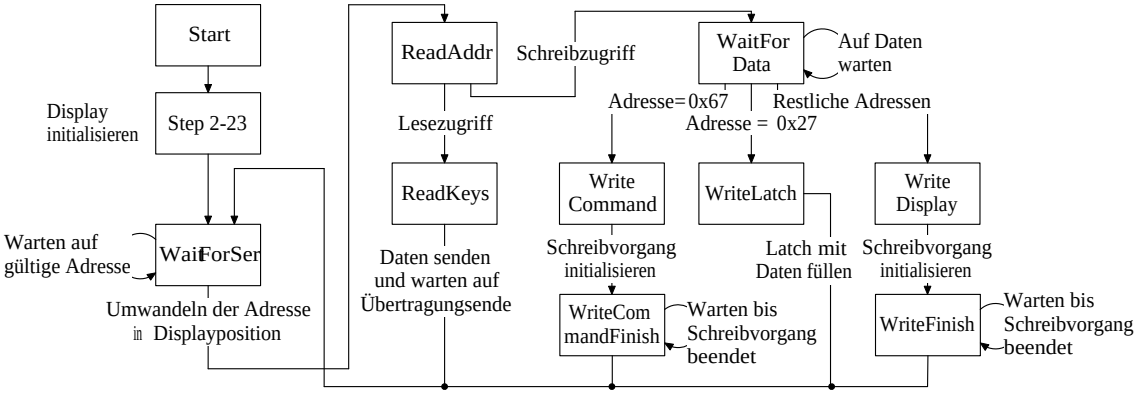



Abbildung 6.21: EPLD: “display”-State-Maschine koordiniert I$^2$C-Bus und Displayaktivitäten

Nach der Initialisierung des Displays wird der I$^2$C-Bus auf eine gültige Displayadresse überwacht. Trifft eine solche ein, wird das Schreib/Lese-Bit extrahiert, und die Adresse für das Display umgerechnet. Würde man diese Umrechnung unterlassen, würde die Handsteuerbox den gesamten I$^2$C-Adressraum belegen, ohne an Funktionalität dazugewonnen zu haben. Wie

in der Übersicht der Handsteuerbox kurz angedeutet, sind einige Adressen im Adressraum für Sonderfunktionen reserviert. Es gibt einen 2-Bit-Speicher, der verwendet wird, um die Leuchtdioden anzusteuern. Seine Registeradresse ist in der Tabelle am Anfang dieses Kapitels angegeben. Zwei weitere Register sind für das Einlesen der Tastatur zuständig. Es kann hier jede beliebige Adresse im Adressbereich verwendet werden. Da es sonst keine lesbaren Register gibt, wird nur das untere Adressbit verwendet, um jeweils 8 Tasten auszuwählen.

## 6.3 Low-Level-Treiber als "Laser-Server"

Der Laser-Server ist eine Sammlung von Realtime-Threads, die gemeinsam das Laserbearbeitungszentrum steuern. Für jede der folgenden Tätigkeiten wurde der Übersicht halber ein eigener Thread kreiert:

- Q-Switch Operationen
- Systemüberwachung
- Materialvorschub mittels Schrittmotors
- Kommunikation mit der Handsteuerbox
- Galvobewegung
- Teach-In Funktion

Für die Kommunikation mit der Benutzeroberfläche werden fünf RT-FIFO verwendet. Zwei RT-FIFO (je eine für Lesen und Schreiben) werden zur Übermittlung der Kontrollkommandos verwendet, zwei für die Statusinformationen und der letzte RT-FIFO übermittelt die Bahninformationen für die Ablenkeinheit. Hier ist kein Rückkanal vorgesehen, da eine Bestätigungsnachricht für die Bahninformationen, die den Großteil des Datenaufkommens ausmachen, sinnlos ist, und nur die Prozessorbelastung unnötig steigern würde.

### 6.3.1 Q-Switch-Task

Der Q-Switch-Task ist der am einfachsten strukturierte Thread. Er besteht aus einer Endlosschleife, die periodisch abgearbeitet wird. Dort wird der Q-Switch-Sollwert berechnet, und anschließend ausgegeben. Da die Killerpulsunterdrückung bei unserem Q-Switch den ersten Puls nach dem Öffnen nicht so unterdrücken kann wie erwartet, wird der Q-Switch erst mit etwas kleinerer Amplitude angesteuert, die sich nach einigen Durchlaufzyklen des Threads auf den gewünschten Wert erhöht und so eine "Soft Start"-Funktion durchführt. Nach jedem Zyklus wird die Systemkontrolle an den Scheduler abgegeben, und bis zur nächsten Abarbeitungsperiode gewartet.

Um diesen, wie auch die anderen Threads ordnungsgemäß zu starten und zu beenden, sind die Funktionen $Init \ldots Task$ und $Delete \ldots Task$ vorhanden. Sie werden jeweils beim Einfügen und Entfernen des Laser-Servers aufgerufen.

### 6.3.2 Guard-Task

Der Guard-Task soll zur Systemüberwachung dienen. Er überprüft periodisch alle systemkritischen Komponenten, und falls ein Grenzwert überschritten wird, setzt er die Anlage in einen sicheren Betriebszustand zurück. Unsere Laserquelle überwacht sich selbst, daher hat der Guard-Task keinerlei Funktion. Er wurde trotzdem nicht aus dem System entfernt, um nicht bei der Portierung auf andere Systeme darauf zu vergessen.

### 6.3.3 Feeder-Task

Zur Erzeugung von Etiketten wurde ein Gerät entwickelt, das mittels eines Schrittmotors eine auf einer Rolle befindliche Folie am Bearbeitungsfeld des Lasers abspult. So können mehrere Aufkleber ohne Einwirkung des Bedienpersonals gefertigt werden. Die Ansteuerung dieses Schrittmotors wird in diesem Thread durchgeführt. Der Schrittmotorsteuerung wird ein Richtungs- und Taktsignal vorgegeben, das diese in ein Schrittmuster umwandelt, verstärkt und an den Motor ausgibt (siehe Abbildung 6.22).

Es wird bei jedem Durchlauf der Schleife der Pegel des Taktsignals geändert, und das Richtungssignal der geforderten Drehrichtung entsprechend ausgegeben. Die Schrittgeschwindigkeit, und somit die Vorschubgeschwindigkeit des Schrittmotors, beträgt also die doppelte Periodendauer des Feeder-Tasks.

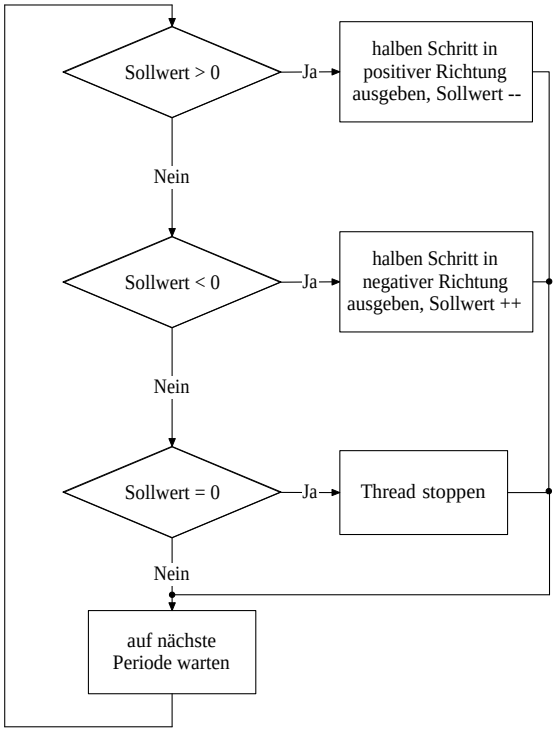



Abbildung 6.22: Feeder-Task, Ablaufdiagramm

### 6.3.4 I²C-Task

Im I²C-Task wird die Kommunikation mit dem Handsteuergerät durchgeführt. Eine State-Maschine ermittelt bei jedem Durchlauf den Zustand der Tastatur, setzt den Zustand der Leuchtdioden und überträgt ein Zeichen aus dem Bildschirmbuffer des Displays an die richtige Position an der Handsteuergerätanzeige (siehe Abbildung 6.23). Im Falle einer fehlerhaften Übertragung wird dieses erneut übertragen.

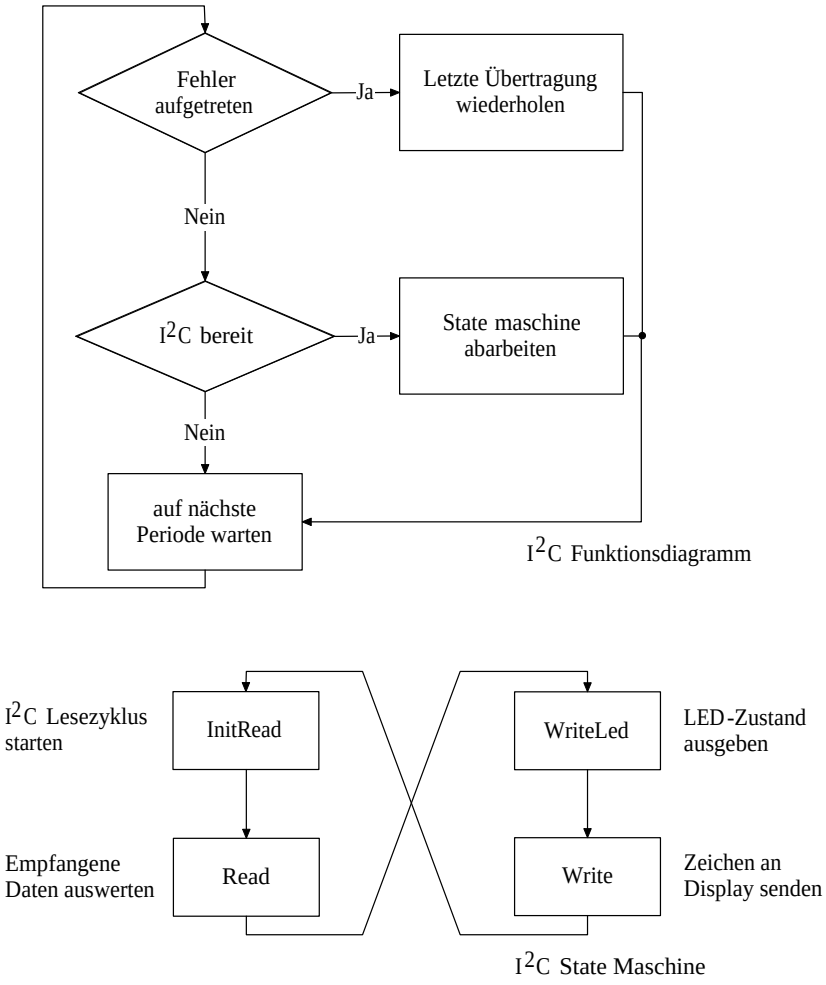



Abbildung 6.23: I²C-Task, Ablaufdiagramm und State Maschine

### 6.3.5 Galvo-Task

Im Galvo-Task wird die Ablenkung des Laserstrahls bearbeitet. Bei der von uns verwendeten Methode, der Ablenkung mittels elektromagnetisch gestellter Spiegel, ist die mögliche Ablenkgeschwindigkeit hoch genug, dass "Wobbeln" implementiert werden konnte. Dabei wird der Vorschubbewegung des

Laserstrahls eine Kreisbahn überlagert (siehe Abbildung 6.24). Dies bringt den Vorteil eines gleichmäßigeren Abtrags bei flächenhafter Bearbeitung, da das Material gleichmäßig erwärmt wird.

Ohne Mit

Wobbelbewegung
(schematische Darstellung)

Abbildung 6.24: Bearbeitung ohne und mit Wobbelbewegung

Aufgrund des hohen Systemressourcenbedarfs des Galvo-Tasks[5] wird dieser nur während des Bearbeitungsvorgangs mit einer niedrigen Periodendauer abgearbeitet, um bei hoher Punktedichte eine entsprechende Vorschubgeschwindigkeit zu erzielen. Läuft gerade keine Bearbeitung, wird auf eine Stillstandsperiodendauer umgeschaltet, um Rechenleistung für andere Systemaufgaben zur Verfügung zu stellen. Würde der Galvo-Task vollständig abgeschaltet werden, könnten die Galvos nicht mehr positioniert werden.

In der Endlosschleife des Galvo-Tasks werden die eintreffenden Daten aus dem RT-FIFO ausgelesen und dementsprechende Bewegungen der Galovs durchgeführt. Da der Q-Switch als Shutter eingesetzt wird, geschieht das Öffnen und Schließen des Q-Switches im Galvo-Task, um so automatisch eine Synchronisation zum Vorschub zu erreichen. Sind die Daten des RT-FIFO ausgewertet, werden diese eventuell mit Wobbeldaten überlagert und anschließend an die Galvos ausgegeben.

### 6.3.6 Adjust-Task

Der Adjust-Task wird für die Teach-In Funktionalität benötigt. Es wird hier ermöglicht, die Strahlablenkung mittels der Tasten des Handsteuergeräts zu betätigen und einzelne Positionsdaten an ein Linux Programm zu übergeben. Dieses kann daraus ein Bearbeitungsprogramm zusammenstellen, das abgespeichert und danach abgearbeitet werden kann. Es sind aber auch andere Verwendungsmöglichkeiten für solch eine Funktion gegeben[6]. Der Laser ist aus Sicherheitsgründen bei diesem Vorgang nicht in Betrieb.

Es sind zwei verschiedene Funktionen implementiert: Der Kalibrier-Modus und der Adjust-Modus. Der Unterschied ist gering, es werden in beiden Fällen die Daten der Strahlablenkung an das Linux-Programm übergeben.

[5] Die Periodendauer kann kleiner als 100 Mikrosekunden sein. Sie ist nur von der Rechenleistung des Steuerrechners beschränkt

[6] Weiterführende Angaben finden sich in der Arbeit meines Kollegen Stefan Großberger

Der wesentliche Unterschied ist die Ausgabe am Display der Handsteuerbox und die Verwendung der Daten im Linux-Programm. Es sind Funktionen für das Anfahren zweier Referenzpunkte implementiert, ein Scanmodus, welcher während der Teach-In Funktion benutzt wird und ein Kalibriermodus, der benutzt wird, um bei einem zuvor gravierten Referenzbild markante Punkte anzufahren und so die optischen Komponenten des Strahlengangs zu kalibrieren.

### 6.3.7 Die FIFO-Handler

In dem Handler der Control- und Status-FIFO-Buffer wurde die Koordination der einzelnen Module und die Datenübergabe von und zu den verschiedenen Linux-Programmen verwirklicht. Wie auch in den RT-FIFO des Galvo Tasks wird die Kommunikation zwischen Linux-Programm und FIFO-Handler über Messages verrichtet, die aus je einem Kommando- und einem Datenwort bestehen. Diese Kommandos sind in der Datei *lasercontrol.h* definiert. Es existiert ein Handler für den Status-FIFO-Buffer, der hauptsächlich Statusinformationen über die aktuell laufende Bearbeitung überträgt, und einer für den Kommando-FIFO-Buffer, der Daten an die anderen Module zuweist, beziehungsweise diese steuert.

### 6.3.8 Aufteilung des Sourcecodes

Folgende Tabelle gibt Auskunft darüber, wo welche Funktion im Sourcecode zu finden ist:

| Funktion | File |
|---|---|
| void *AdjustTask(void *call_data) | adjusttask.c |
| void cleanup_module(void) | Laser_server_ipc.c |
| int controlpipe_handler(unsigned int fifo) | Laser_server_ipc.c |
| void DeleteAdjustTask(void) | adjusttask.c |
| void DeleteFeederTask(void) | feedertask.c |
| void DeleteGalvo(void) | galvo.c |
| void DeleteGalvoTask(void) | galvotask.c |
| void DeleteGuardTask(void) | guardtask.c |
| void DeleteI2CTask(void) | i2ctask.c |
| void DeleteQSwitchTask(void) | qswitchtask.c |
| void *FeederTask(void *call_data) | feedertask.c |
| void Galvo(int x,int y) | galvo.c |
| void *GalvoTask(void *call_data) | galvotask.c |
| void GalvoTaskSuspend(void) | galvotask.c |
| void *GuardTask(void *call_data) | guardtask.c |
| void *I2CTask(void *t) | i2ctask.c |
| int InitAdjustTask(void) | adjusttask.c |
| int InitFeederTask(void) | feedertask.c |
| void InitGalvo(void) | galvo.c |
| int InitGalvoTask(void) | galvotask.c |
| int InitGuardTask(void) | guardtask.c |
| int InitI2CTask(void) | i2ctask.c |
| void InitKeyboard(void) | i2ctask.c |
| int InitQSwitchTask(void) | qswitchtask.c |
| int init_module(void) | Laser_server_ipc.c |
| int intcos(int phi) | Laser_server_ipc.c |
| void *QSwitchTask(void *call_data) | qswitchtask.c |
| int statuspipe_handler(unsigned int fifo) | Laser_server_ipc.c |

## 6.4 High-Level-Treiber als “Laser-Library”

Die Laser-Library wurde erstellt, um eine einheitliche Schnittstelle für Linux-Programme, die mit dem Realtime-Modulen kommunizieren müssen, herzustellen. Die Bibliothek ist als Objekt in C++ erstellt, da sie meist in Programmen mit grafischer Oberfläche eingebunden wird, welche ebenfalls in C++ geschrieben sind.

Folgende Funktionen werden zur Verfügung gestellt:

**LaserComm::LaserComm(const bool status, int (*errorcallback)(const char *msg));**

Initialisiert die Kommunikation mit der Hardware. Ist *status* 1, werden die Status-FIFO-Buffer geöffnet, anderenfalls werden die serielle Schnittstelle zum Laser, die Kontroll-FIFO-Buffer und der Galvo-FIFO-Buffer geöffnet. Wird die Laser-Library zur Status-Kommunikation eingesetzt, dürfen nur die $ST_\ldots$-Befehle benutzt werden. Wird anderenfalls 0 als *status* übergeben, dürfen diese Befehle nicht benutzt werden. Wird mittels $*errorcallback$ ein Zeiger auf eine Funktion übergeben, so wird diese benutzt, um Fehlermeldungen auszugeben. Wird dies nicht gewünscht, dann muss $NULL$ übergeben werden. Die Fehlermeldungen werden dann auf *stderr* ausgegeben.

**LaserComm:: LaserComm();**

Diese Funktion wird automatisch aufgerufen, wenn die Bibliothek beim Programmende geschlossen wird. Es werden alle RT-FIFO und die serielle Schnittstelle geschlossen.

**void LaserComm::KillerPulse(const bool enable);**

Damit wird die Killerpulsunterdrückung des Q-Switch ein- und ausgeschaltet.

**void LaserComm::ChangeWobbel(const unsigned int i);**

Es wird ein neuer Wobbelkreisdurchmesser festgesetzt. Der Wert $i$ entspricht dem Durchmesser in hundertstel Millimeterschritten.

**void LaserComm::ChangeQSWVoltage(const unsigned char i);**

Ändert die Amplitude des Q-Switch Taktsignals. Der Bereich der Variablen $i$ reicht von 0 bis 255, was 0% bis 100% entspricht.

**void LaserComm::ChangeQSWFreq(const unsigned int i);**

Ändert die Frequenz des Q-Switch Taksignals. Der Wert $i$ entspricht der Q-Switch Frequenz in Herz.

**void LaserComm::ChangeQSWOnTime(const unsigned int i);**

Beeinflusst die Zeitspanne, in welcher der Q-Switch durchschaltet. Dem Aufruf dieser Funktion muss ein $ChangeQSWFrequ$ Aufruf folgen. $i$ gibt die Zeit in hundertstel Sekunden an. Die Sinnhaftigkeit in Verbindung mit der Q-Switch Frequenz wird nicht überprüft.

**void LaserComm::SetLampCurrent(const unsigned char i);**

Sendet einen neuen Lampenstromwert an den Laser. Gültige Werte für $i$ sind 0 bis 255, was 0% bis 100% entspricht.

**unsigned char LaserComm::GetLampCurrent();**

Liest den aktuell eingestellten Lampenstrom vom Laser aus. Es werden Werte zwischen 0 und 255 zurückgegeben.

**bool LaserComm::PowerOn();**

Aktiviert den Laser.

**bool LaserComm::LaserOn();**

Deaktiviert den Laser.

**void LaserComm::OpenShutter();**

Öffnet den mechanischen Shutter des Lasers. Dieser wird nur aus Sicherheitsgründen verwendet. Während der Bearbeitung sollte der Q-Switch als Shutter eingesetzt werden, um den Laserstrahl dunkelzutasten.

**void LaserComm::CloseShutter();**

Schließt den mechanischen Shutter des Lasers.

**void LaserComm::FeederAdvance(const int count);**

Aktiviert den Vorschub des Etikettenspenders. *count* gibt die Schrittanzahl des Motors an. Diese kann je nach Bauweise des Spenders auch negative Werte annehmen, um die Etikettenrolle aufzuspulen.

**void LaserComm::StopGalvotask();**

Wartet, bis die Bearbeitung durch den Galvo-Task abgeschlossen ist, und versetzt diesen dann in den Ruhezustand.

**void LaserComm::ResetError();**

Setzt den vom Guard-Task entdeckten Fehler zurück.

**void LaserComm::StartTask(const int freq);**

Versetzt den Galvo-Task in den Arbeitszustand mit der Frequenz $freq$. $freq$ ist in Herz anzugeben.

**void LaserComm::SendSync(const int _data);**

Beim Abarbeiten dieser Anweisung wird vom Galvo-Task eine Synchronisationsnachricht mit $_data$ im Datenfeld an den Status-FIFO-Buffer geschickt.

**void LaserComm::OutX(const int x);**
**void LaserComm::OutY(const int y);**
**void LaserComm::OutXY(const int x,const int y);**
**void LaserComm::OutYX(const int y,const int x);**

Leitet eine Bewegung der Strahlablenkung in den entsprechenden Achsen ein. Die neue Position wird absolut durch $x$ und $y$ beschrieben. Der Wertebereich hängt von der verwendeten Strahlablenkungseinheit ab. Der Unterschied zwischen $OutXY$ und $OutYX$ ist die Reihenfolge der Achsenbewegungen.

**void LaserComm::DelayTask(const unsigned int delay);**

Verzögert die darauffolgenden Galvobefehle um *delay**Periodendauer der mit $StartTask$ festgelegten Frequenz.

**int LaserComm::WaitForTask();**

Veranlasst die Übertragung einer Message an den Status-FIFO-Buffer. Das Ende der Bearbeitung wird so der Software mitgeteilt, die den Status-FIFO-Buffer benutzt.

**int LaserComm::FinishTask();**

Selbe Funktionalität wie $WaitForTask$.

**void LaserComm::QSwitch(const bool open);**

Öffnet oder schließt den als Shutter benutzten Q-Switch.

**void LaserComm::SetCorrection(int to);**

Dieser Befehl selektiert eine Korrekturtabelle im Laser-Server. Die Korrekturtabelle wird aufgrund der Verzerrungen in der Optik benötigt und muss beim Objektivwechsel angepasst werden. Eine eingehende Erläuterung der Korrekturtabelle findet sich in S.Großberger, Diplomarbeit TU-Wien [9].

**unsigned char LaserComm::ST_GetLampCurrent();**

Liest den aktuell eingestellten Lampenstrom per Status-FIFO-Buffer.

**unsigned char LaserComm::ST_GetQSwStatus();**

Ermittelt den Q-Switch Status. Das Byte enthält folgende Informationen:

| Bit | Bedeutung |
|---|---|
| 0 | Q_SW RF Fault |
| 1 | Q_SW PA High Temperature Fault |
| 2 | Q_SW High Temperature Fault |
| 3 | I2C Busy |
| 4 | I2C Error |

**void LaserComm::ST_Adjust(int &minX, int &maxX, int &minY, int &maxY, int &scale);**

Startet den Adjust-Task mit den angegebenen Parameter. Die Prozedur, welche hiermit durchgeführt werden kann, wird in einer separaten Arbeit von S.Großberger, Diplomarbeit TU-Wien [9] beschrieben.

**void LaserComm::ST_StopAdjust();**

Versetzt den Adjust-Task in den Ruhezustand.

**void LaserComm::ST_Calibrate(int &x, int &y);**

Startet den Adjust-Task im Kalibriermodus. Eine nähere Beschreibung der Prozedur, die hier durchgeführt werden kann, wird von Stefan Großberger in seiner Arbeit [9] gegeben.

**void LaserComm::ST_EMERGENCY_STOP();**

Bricht die Bearbeitung sofort ab und bringt das System in einen sicheren Zustand.

**bool LaserComm::ST_PollInPipe(struct my_data_struct *msg);**

Überprüft ob eine Message im Status-FIFO-Buffer vorhanden ist und kopiert diese, wenn vorhanden, in die $*msg$ Struktur. Beim Erfolg ist der Rückgabewert True, sonst ist er False.

**void LaserComm::lerror(const char *function_name, const char *format, ...);**

Gibt einen im $printf$ Format übergebenen Text aus. Diese Funktion ist dem Programmierer nicht zugänglich.

**int LaserComm::Receive(const int fd, void *buf, const int buflen, const int timeout_ms);**

Dient zur Kommunikation mit der seriellen Schnittstelle. Diese Funktion ist dem Programmierer nicht zugänglich.

**void LaserComm::Transmit(const int fd, const char *format, ...);**

Dient zur Kommunikation mit der seriellen Schnittstelle. Diese Funktion ist dem Programmierer nicht zugänglich.

# Kapitel 7

# Entwickelte Anwenderprogramme

Die Punkte, die in diesem Kapitel besprochen werden, sollen nur zur Übersicht dienen. Eine tiefergehende Diskussion ist in der Arbeit von Stefan Großberger [9] zu finden.

## 7.1 Das HLL-Datenformat

Um den Bearbeitungsvorgang für die Laserbearbeitung zu beschreiben, wurde im Rahmen dieser Arbeit das HLL[1]-Datenformat entwickelt. Es enthält nicht nur die Geometriedaten, sondern auch materialbezogene Daten wie Laserleistung, Vorschubgeschwindigkeit und dergleichen.

Das HLL-Format wurde in zwei Bereiche aufgeteilt: Einen Parameterbereich, in dem die materialbezogenen Daten beschrieben werden, und einen Geometriedatenbereich. Diese beiden Bereiche werden im folgenden kurz dargestellt.

[1] HLL steht hier für Hyper Laser Language

### Der Parameterbereich

Es können bis zu 255 verschiedene Parametersätze definiert werden.

Im allgemeinen müssen die Parameter am Beginn der Datei definiert werden. Einzige Ausnahme ist, wenn die Parameter in einer Datei abgelegt werden, die nur Parameter enthält. Dann muss das HLL-File keine Parameterdefinitionen enthalten. Sie werden beim Abarbeiten aus dieser externen Datei geladen.

Folgende Parameterformate werden unterstützt:

```
P(a) = QFREQ (b)         QVOLT (d) QSOFT (e) LAMP (f) SPEED (g) DELAY (i);
P(a) = QFREQ (b)         QVOLT (d) QSOFT (e) LAMP (f) SPEED (g) DELAY (i)   WOBBEL (j);
P(a) = QFREQ (b) QON (c) QVOLT (d) QSOFT (e) LAMP (f) SPEED (g) DELAY (h,i) WOBBEL (j);
P(a) = QFREQ (b) QON (c) QVOLT (d) QSOFT (e) LAMP (f) SPEED (g) DELAY (i)   WOBBEL (j);
```

| # | Bedeutung | Einheit | Wertebereich |
|---|---|---|---|
| a | Parameter Nummer | - | 0 - 254 |
| b | Q-Switch Frequenz | Hz | 0, 100 - 50000 |
| c | Q-Switch On Time | $\mu$s | 0 - $\frac{1.000.000}{2*Q_FREQU}$ |
| d | Q-Switch Amplitude | $\frac{d}{255} * 100$ % | 0 - 255 |
| e | Q-Switch Killerpuls-<br>unterdrückung | - | 0, 1 |
| f | Laser Lampenstrom | $\frac{f}{255} * 100$ % | 60 - 255 |
| g | Vorschubgeschwindigkeit | $\frac{1}{100} \frac{mm}{sec}$ | 100 - 25000 |
| h | Verzögerung zwischen<br>Galvobewegung und<br>Q-Switch öffnen | ms | 0 - 255 |
| i | Verzögerung zwischen<br>Q-Switch öffnen<br>und Galvobewegung | ms | 0 - 255 |
| j | Wobbel Durchmesser | $\frac{1}{100}$ mm | 0 - |

### Die Geometriedaten

Die Geometrie wird mittels folgender Befehle beschrieben:

```
START AT (x,y);
REFERENCE AT (x,y);
LINE TO (x,y) P (p);
CIRCLE CENTER (x,y) RADIUS (rx,ry) ANGLE (a) ROT (r) P (p);
BOX WITH (x,y) ROT (r) P (p);
TEXT WITH (x,y) ROT (r) LP (l) FAMILY (f) WEIGHT (w) ITALIC (i) P (p);
```

| # | Bedeutung | Einheit | Wertebereich |
|---|---|---|---|
| a | Öffnungswinkel | $\frac{1}{100}^0$ | ±0 - 36000 |
| f | Font Familie | - | als Text angeben |
| i | Schrägstellung des Textes | - | 0, 1 |
| l | Offset des Textfeldes | - | siehe Arbeit von |
| | in Bezug auf die Eingabedatei | | Stefan Großberger |
| p | zugehöriger Parametersatz | - | muss vorher definiert werden |
| r | Rotation der Figur | $\frac{1}{100}^0$ | 0 - 36000 |
| rx | Radius in X Richtung | $\frac{1}{100}$ mm | |
| ry | Radius in Y Richtung | $\frac{1}{100}$ mm | |
| w | fette Schriftart | - | 0, 1 |
| x | absolute X Koordinate | $\frac{1}{100}$ mm | hängt von der |
| | | | Strahlablenkung ab |
| y | absolute Y Koordinate | $\frac{1}{100}$ mm | hängt von der |
| | | | Strahlablenkung ab |
| | | | (0,0) ist normalerweise |
| | | | der Mittelpunkt |

## 7.2 Erstellen einer HLL-Datei

Es gibt im derzeitigen Entwicklungsstadium vier Möglichkeiten, ein HLL-File zu erstellen:

- Direkteingabe in ASCII-Editor: Dies funktioniert bei kurzen Dateien mit einfacher Geometrie, wird jedoch schnell unübersichtlich, besonders wenn Kreise und Ellipsen vorkommen.
- LPaint: Dies ist ein im Zuge dieser Arbeit entwickeltes Vektor-Zeichenprogramm. Damit wird aus den oben angeführten Objekten eine Bearbeitungsdatei zusammengestellt. Dabei können alle Parameter in Textfeldern editiert werden, um sich ein pixelgenaues Einfügen von Punkten zu ersparen (siehe Abbildung 7.1 und Abbildung C.7).

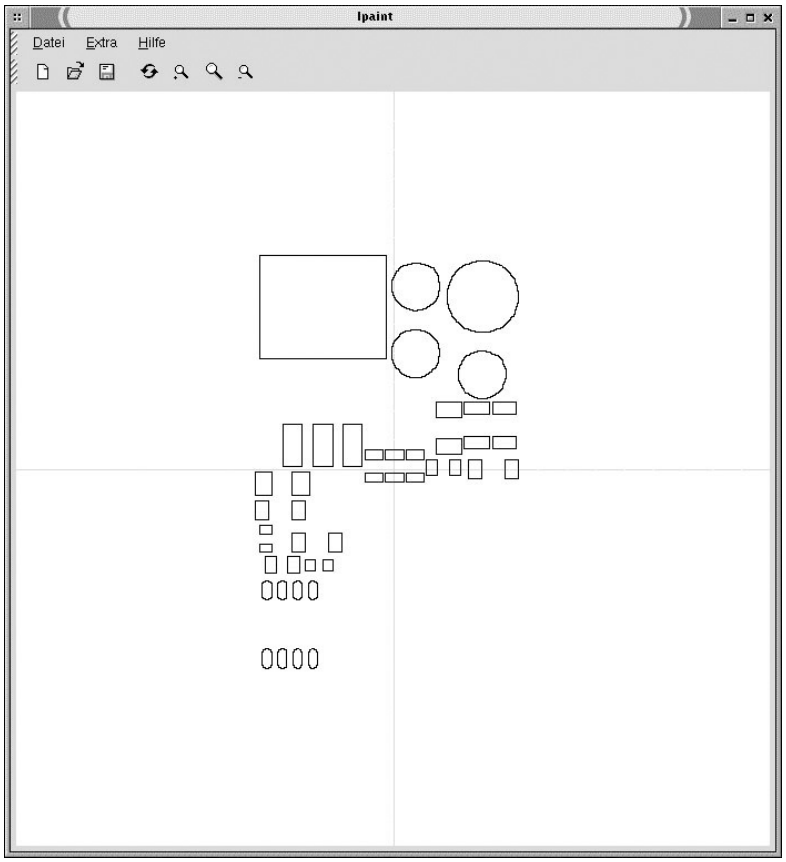



Abbildung 7.1: Lpaint: Darstellung einer Datei für eine Druckschablone

- Simulation eines Postscript-Netzwerkdruckers: Diese ist die eleganteste der hier vorgestellten Methoden. Es wird ein postscript[2]-fähiger Netzwerkdrucker simuliert, auf dem jede Software zugreifen kann, die fähig ist zu drucken.

[2]Postscript ist eine von Adobe entwickelte Seitenbeschreibungssprache die unter anderem bei Laserdrucker eingesetzt wird.

Es wurden drei verschiedene Verfahren entwickelt, um die Postscript-Daten in eine HLL-Datei umzuwandeln:

  - Scan-Verfahren: Die Bearbeitungsfläche wird Zeile für Zeile in HLL-umgewandelt. Die Abarbeitung erfolgt ebenso (siehe Scanningverfahren im Kapitel 3.3).
  - Outline-Verfahren: Es wird nur die äußere Kontur umfahren. Es entsteht keine originalgetreue Abbildung der Postscriptdaten. Der hauptsächliche Einsatz ist in der Herstellung von Etiketten gegeben.
  - Vektor-Füll-Verfahren: Hier werden alle geschlossenen Flächen nach einem intelligenten Algorithmus abgearbeitet. Ziel ist die Minimierung der Absetzvorgänge, also die Figur in einem Zug durchzufahren.

- Konvertierung einer Pixelgrafik: Dies funktioniert nach dem Matrixverfahren (Abschnitt 3.3). Ein Beispiel hiervon zeigt die Abbildung 7.2.

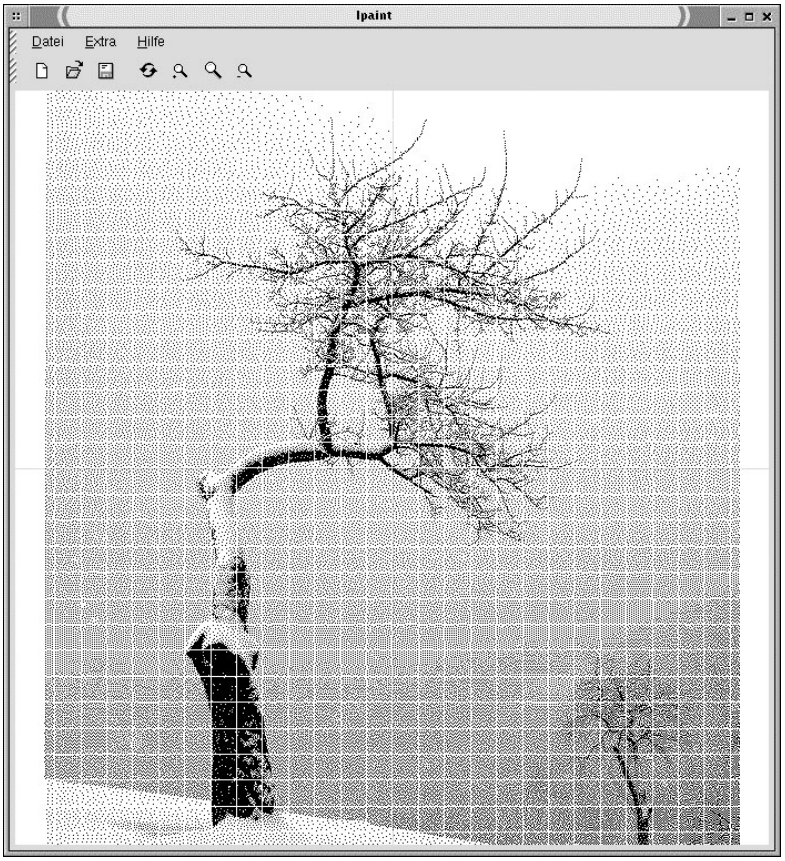



Abbildung 7.2: Lpaint: Darstellung einer konvertierter Pixelgrafik

## 7.3 Abarbeiten einer HLL-Datei

Eine HLL-Datei wird durch das Programm "LWork" abgearbeitet. LWork besitzt keine grafische Benutzerschnittstelle, sondern wird durch die folgende Kommandozeile aufgerufen:

```
lwork [-hvT] [-p play_n_times] [-P parameterfile] [-f feederadvance]
      [-w feeder_when] [-x offsetX] [-y offsetY] [-s scale]
      [-r rotation] [-t textfile] [-i lineincrement] filename
```

Die Parameter haben dabei folgende Bedeutung:

| # | Bedeutung | Einheit | Wertebereich/Beschreibung |
|---|---|---|---|
| -h | Hilfe | - | gibt obigen Programmaufruf aus |
| -v | Verbose | - | es wird genauer Mitprotokolliert |
| -T | Testmode | - | der Laserstrahl bleibt während der Bearbeitung deaktiviert |
| -p | Play_n_times | 1 | Anzahl der Wiederholungen |
| -P | Parameterfile | - | Name der Datei die die Paramter enthält. nur angeben wenn keine Parameter im HLL-File vorhanden sind |
| -f | Feederadvance | Schritte | Schrittanzahl die der Etikettengröße entspricht |
| -w | Feeder_when | - | 1, 2, 3<br>1: Vorschub vor der Bearbeitung<br>2: $\frac{1}{2}$ Vorschub vor,$\frac{1}{2}$ Vorschub danach<br>3: Vorschub nach der Bearbeitung |
| -x | offsetX | $\frac{1}{100}$ mm | wird zu jeder X Koordinate addiert |
| -y | offsetY | $\frac{1}{100}$ mm | wird zu jeder Y Koordinate addiert |
| -s | scale | 1 | 0-1 ergibt 0% bis 100%<br>es sind aber auch Werte >1 möglich |
| -r | rotation | $\frac{1}{100}^0$ | ±0 - 36000 |
| -t | textfile | - | wird für Textfelder verwendet |
| -i | lineincrement | - | wird ebenfalls für Textfelder verwendet |
| | filename | - | Name der HLL-Datei |

# Kapitel 8

# Zusammenfassung

Die Mikromaterialbearbeitung mit Laser hat sich in zahlreichen Bereichen der Werkstoffbearbeitung ihren festen Platz errungen. Ausschlaggebend für die erfolgreiche Verbreitung sind die hervorragende Eignung zur numerischen Steuerbarkeit, die hohe Geschwindigkeit und die Genauigkeit der Bearbeitungsvorgänge. Durch die Verfügbarkeit von ausreichend starken Lasern mit unterschiedlichen Wellenlängen stehen den Anwender verschiedenartige Lasermaterialbearbeitungsplätze für eine große Vielfalt von Fertigungsvorgängen zur Auswahl, welche je nach Einsatzzweck unterschiedlichen Anforderungsprofilen entsprechen. Ziel der vorliegenden Diplomarbeit war es, eine universell einsetzbare Steuereinheit für die am Institut vorhandenen Lasermaterialbearbeitungsplätze zu konzipieren, zu entwickeln, aufzubauen und zu testen.

Realisiert wurde die Steuereinheit mit einem Personal Computer, wobei durch Einsatz des Echtzeitbetriebssystems Realtime-Linux die zum Steuern eines solchen Systems notwendige Hardware auf ein Minimum reduziert wurde. Es ist keine zusätzliche Hardware notwendig, die eigene Intelligenz besitzt. Der gesamte Bearbeitungsablauf wird, bei der Dateneingabe beginnend, bis zur Ausgabe der einzelnen Impulse für die Strahlablenkung, vom PC gesteuert.

Da der PC mit zu wenig Schnittstellen ausgerüstet war, und einige von uns verwendeten Komponenten spezielle Ansteuersignale benötigten, wurde eine ISA-Bus-Steckkarte entwickelt, die alle notwendigen Signale zur Ansteuerung des Lasers, des Q-Switches und verschiedenster Strahlablenkungseinheiten bereit stellt.

Zur örtlichen Trennung des Bearbeitungsplatzes und des Bildschirmarbeitsplatzes wurde eine Handsteuerbox mit Display entwickelt. Diese ermöglicht eine einfache Steuerung des Systems ohne den Bildschirm und der Tastatur des Steuerrechners.

Zur Datenübernahme wird ein postscriptfähiger Netzwerkdrucker emuliert, der mit mehreren verschiedenen Konvertierungsalgorithmen die Übernahme

der Daten von anderen Rechnern vereinfacht und den Arbeitsablauf dabei verkürzt. Die so entstandene ASCII-Datei ist vom Menschen lesbar und kann nachträglich mit einem Texteditor editiert werden. Die materialabhängigen Laserparameter werden dabei erst bei der Bearbeitung zugefügt. Ebenso einfach können die Bearbeitungsdaten mittels eines speziell entwickelten Vektorzeichenprogramms eingegeben werden.

Dieses System wurde bereits zweimal Portiert. Einmal, zum Betrieb mit einer starren Strahlführung welches die Verwendung einer Fokussieroptik mit Schutzgasdüse ermöglicht. Das Werkstück wird dabei mittels Schrittmotoren bewegt. Ein zweites Mal wurde ein anderes Lasersystem damit ausgerüstet, dass nun dem Anwender die gleiche, einfache Benutzerumgebung bieten kann. Der Datenaustausch zwischen diesen Systemen ist problemlos möglich, einzig die Maschinenparameter müssen angeglichen werden.

Dem Betreiber wie auch dem Anwender wird durch diese Arbeit ein offenes System zur Verfügung gestellt, das modular aufgebaut ist, und somit einfach und effizient an verschiedenste Geräte und Anwendungsfälle angepasst werden kann.

# Literaturverzeichnis


[1] Prof. Dr. D. Bimberg,*1985, Laser in Industrie und Technik.*
expert verlag, Sindelfingen.

[2] W.Brunner, K.Junge,*1982, Wissensspeicher Lasertechnik.*
VEB Fachbuchverlag, Leipzig.

[3] S. Furchtbar, W. Hackländer,*1995, $I^2C$-Bus angewandt.*
Elektor Verlag, Aachen.

[4] J. Labrosse,*1999, MicroC/OS-II: The Real-Time Kernel.*
R&D Books, Lawrence.

[5] M. Barabanov, V.Yodaiken,*1996, Real-Time Linux.*
erschienen im Internet, http://www.fsmlabs.com.

[6] M. Sherer,*2002, Writing Applications with RTLinux.*
erschienen im Internet, http://www.fsmlabs.com.

[7] Phillips Semiconductors,*2000, The $I^2C$-Bus specification.*
erschienen im Internet, http://www.semiconductors.philips.com/buses/i2c.

[8] Altera Semiconductors,
*2002, Max 7000 Programmable Logic Device Family.*
erschienen im Internet, http://www.altera.com/products/devices/max7k.

[9] Stefan Großberger, Matr. Nr.: 9226534, *Diplomarbeit in Arbeit.*
Inst. für Industrielle Elektronik und Materialwissenschaften, Wien.

# Anhang A

# Hardware

## A.1 IO-Karte

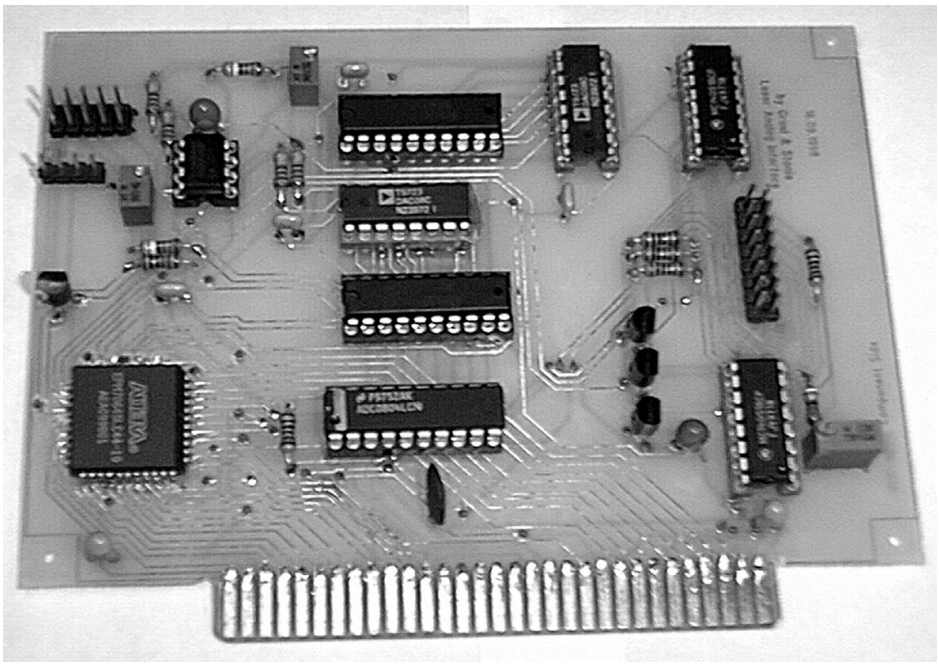
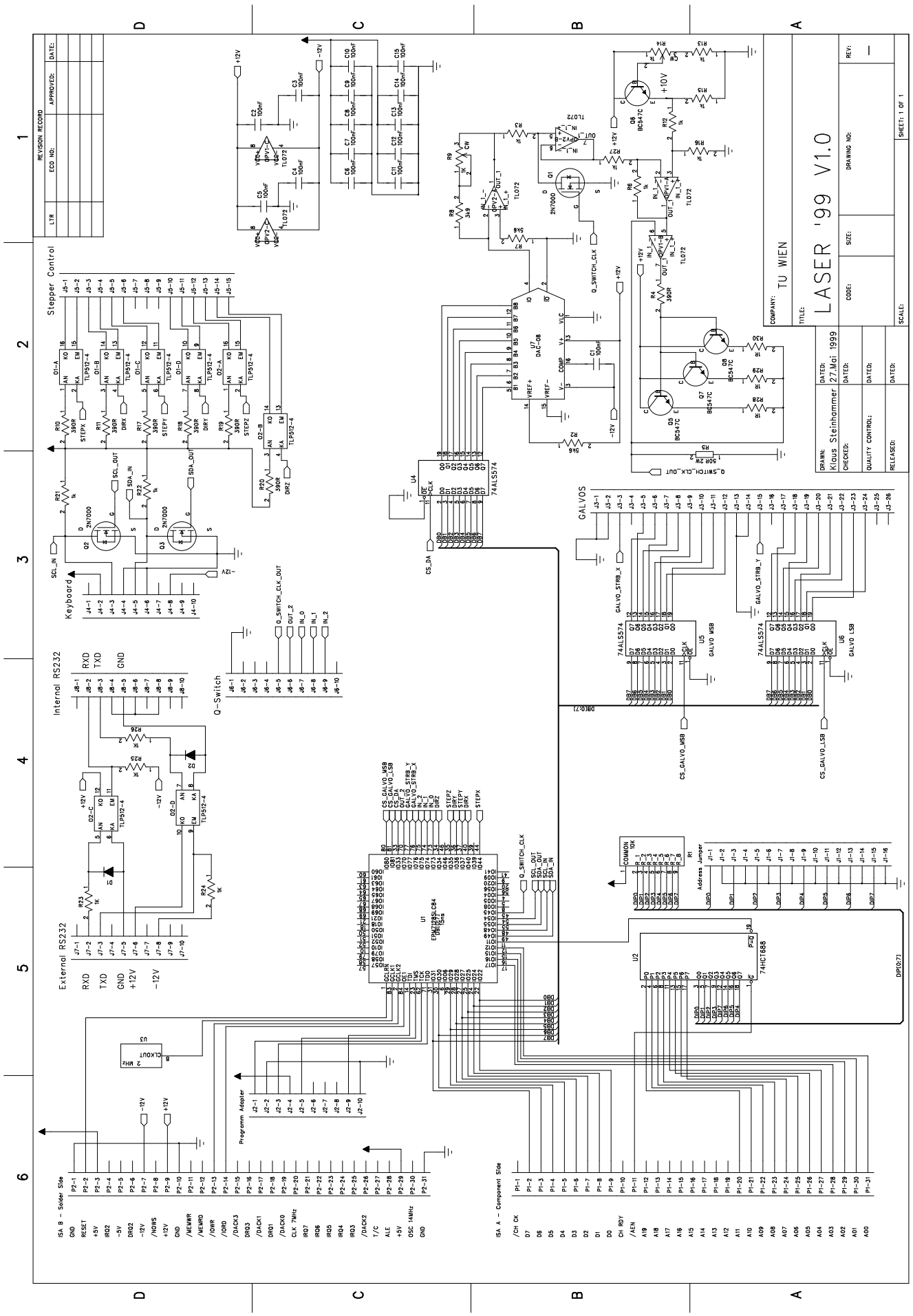


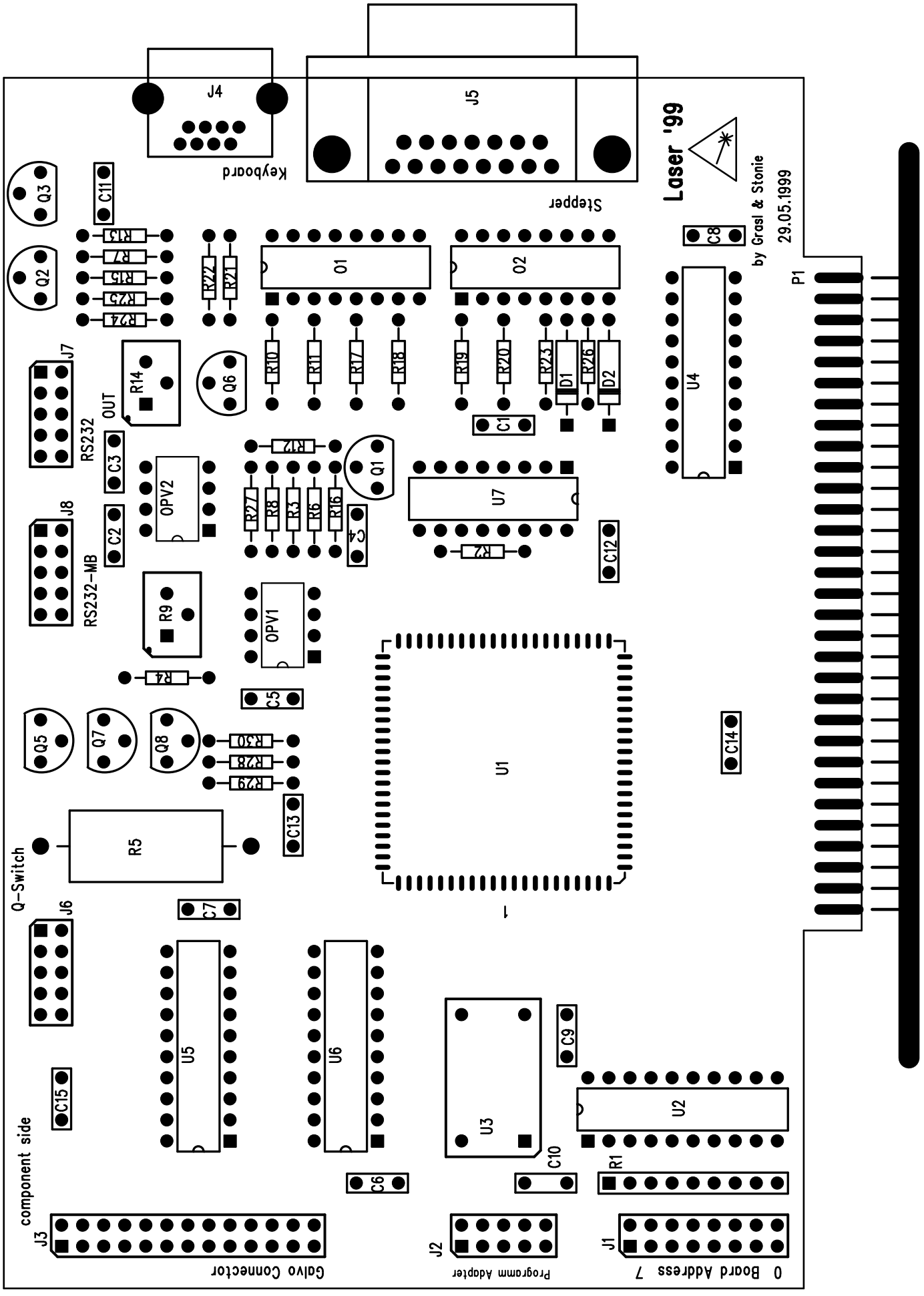
Abbildung A.1: IO-Karte, Bestückungsseite

## A.1.1 Schaltplan




IO-Karte, Schaltplan

### A.1.2 Bestückungsplan



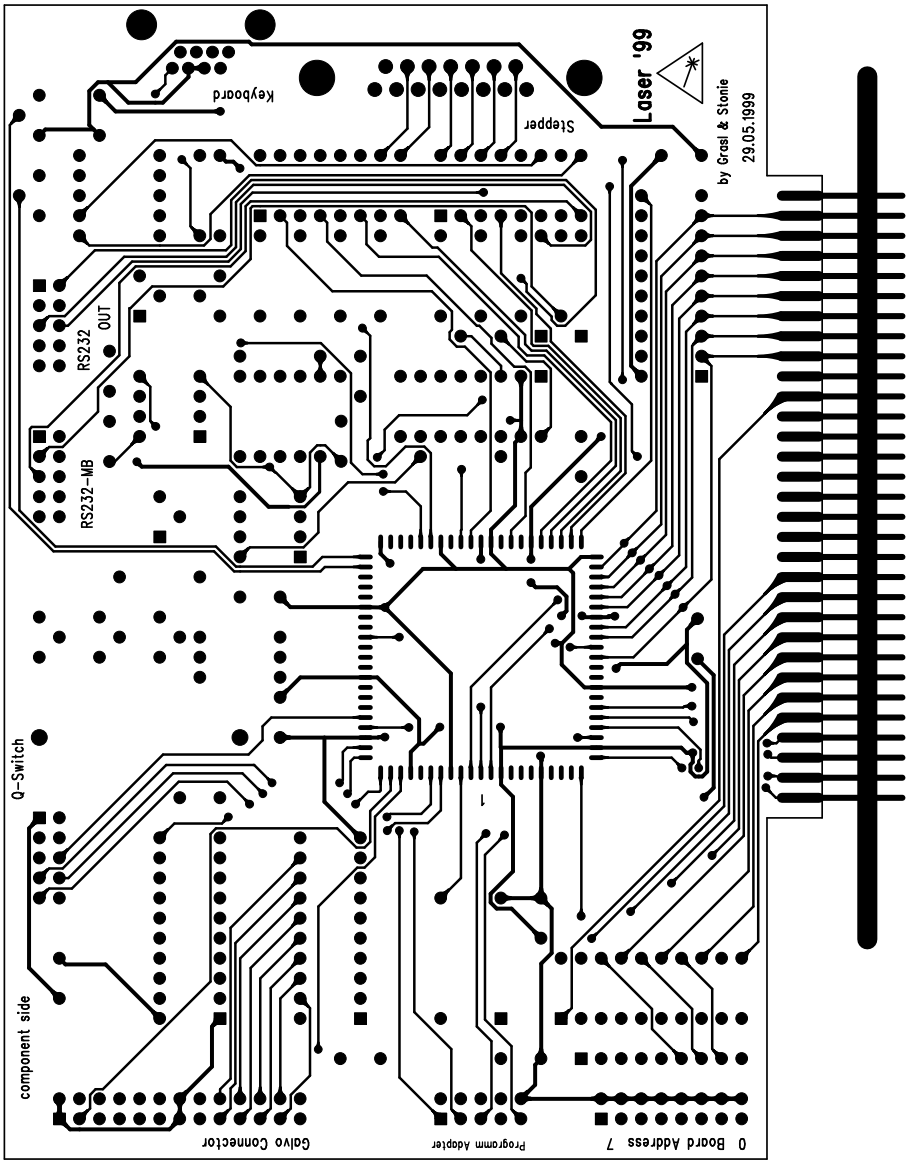

IO-Karte, Bestückungsplan

### A.1.3 Liste der verwendeten Bauteile

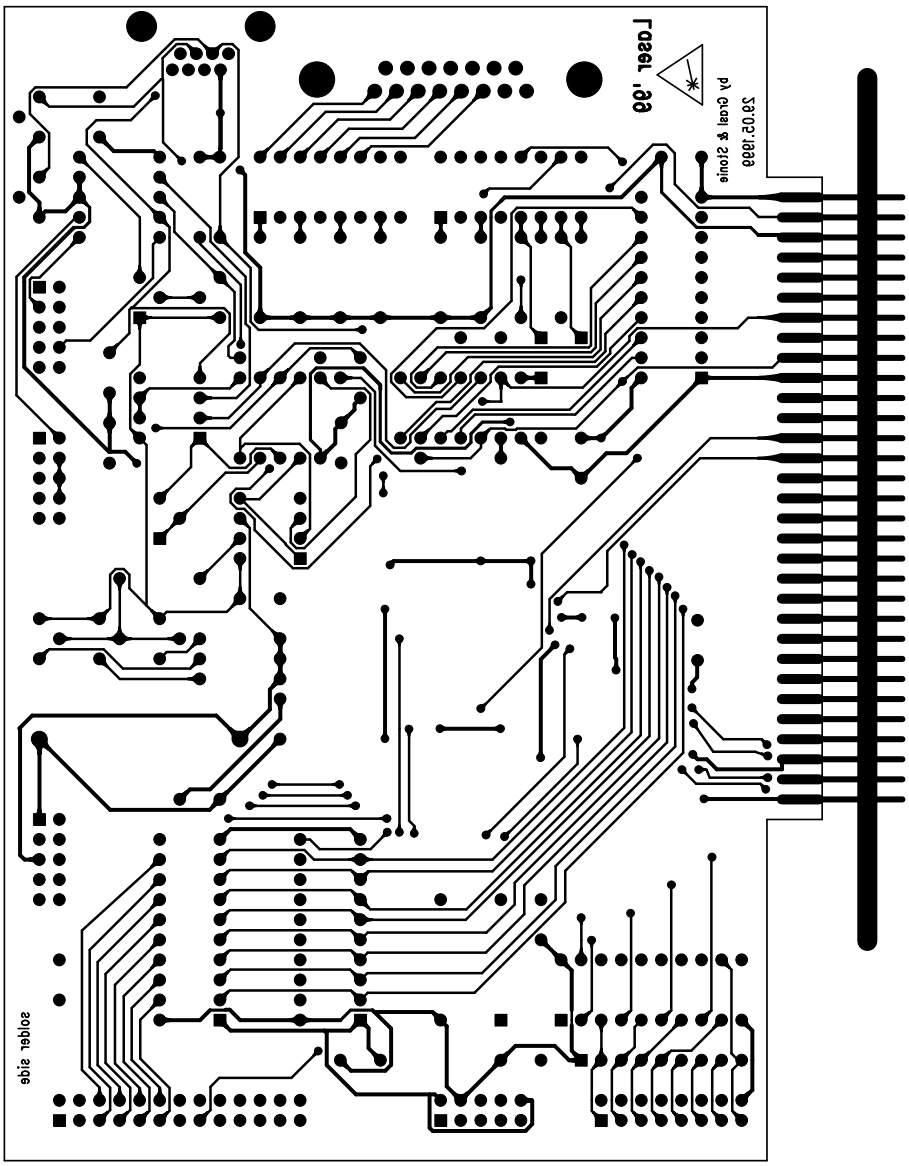
| No. | Referenz | Name | Wert | Beschreibung |
|---|---|---|---|---|
| 1 | C1 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 2 | C2 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 3 | C3 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 4 | C4 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 5 | C5 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 6 | C6 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 7 | C7 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 8 | C8 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 9 | C9 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 10 | C10 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 11 | C11 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 12 | C12 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 13 | C13 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 14 | C14 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 15 | C15 | CAP-CX12-B | 100nF | FOLIENKONDENSATOR MKS2 |
| 16 | D1 | DIODE | | DIODE 1N4184 |
| 17 | D2 | DIODE | | DIODE 1N4184 |
| 18 | J1 | HEADER16 | | STIFTLEISTE RM 2,54 2*8 PINS |
| 19 | J2 | HEADER10 | | STIFTLEISTE RM 2,54 2*5 PINS |
| 20 | J3 | HEADER26 | | STIFTLEISTE RM 2,54 2*13 PINS |
| 21 | J4 | CON-RJ45 | | RJ45 BUCHSE FÜR PRINTMONTAGE |
| 22 | J5 | CON-DB15HF | | SUB-D BUCHSE 15 POL. FÜR PRINTMONTAGE |
| 23 | J6 | HEADER10 | | STIFTLEISTE RM 2,54 2*5 PINS |
| 24 | J7 | HEADER10 | | STIFTLEISTE RM 2,54 2*5 PINS |
| 25 | J8 | HEADER10 | | STIFTLEISTE RM 2,54 2*5 PINS |
| 26 | O1 | TLP512-4 | | QUAD OPTOKOPPLER |
| 27 | O2 | TLP512-4 | | QUAD OPTOKOPPLER |
| 28 | OPV1 | TL072 | | DUAL OPV |
| 29 | OPV2 | TL072 | | DUAL OPV |
| 30 | P1 | EISAA | | ISA EDGE CONNECTOR |
| 31 | P2 | EISAB | | ISA EDGE CONNECTOR |
| 32 | Q1 | 2N7000 | | N-CHANNEL,ENHANCEMENT MODE, TMOS FET |
| 33 | Q2 | 2N7000 | | N-CHANNEL,ENHANCEMENT MODE, TMOS FET |
| 34 | Q3 | 2N7000 | | N-CHANNEL,ENHANCEMENT MODE, TMOS FET |
| 35 | Q5 | BC547C | | AMPLIFIER NPN SILICON TRANSISTOR |
| 36 | Q6 | BC547C | | AMPLIFIER NPN SILICON TRANSISTOR |
| 37 | Q7 | BC547C | | AMPLIFIER NPN SILICON TRANSISTOR |
| 38 | Q8 | BC547C | | AMPLIFIER NPN SILICON TRANSISTOR |

| No. | Referenz | Name | Wert | Beschreibung |
|---|---|---|---|---|
| 39 | R1 | R8_ARRAY | 10K | RES 8 fach |
| 40 | R2 | R1/8W | 5K6 | RES BODY:060 CENTERS:400 |
| 41 | R3 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 42 | R4 | R1/8W | 390R | RES BODY:060 CENTERS:400 |
| 43 | R5 | RES-2W | 50R | RES BODY:340 CENTERS:1000 |
| 44 | R6 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 45 | R7 | R1/8W | 5K6 | RES BODY:060 CENTERS:400 |
| 46 | R8 | R1/8W | 3K9 | RES BODY:060 CENTERS:400 |
| 47 | R9 | VRES | 1K | VARIABLE RESISTOR (POT) |
| 48 | R10 | R1/8W | 390R | RES BODY:060 CENTERS:400 |
| 49 | R11 | R1/8W | 390R | RES BODY:060 CENTERS:400 |
| 50 | R12 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 51 | R13 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 52 | R14 | VRES | 1K | VARIABLE RESISTOR (POT) |
| 53 | R15 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 54 | R16 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 55 | R17 | R1/8W | 390R | RES BODY:060 CENTERS:400 |
| 56 | R18 | R1/8W | 390R | RES BODY:060 CENTERS:400 |
| 57 | R19 | R1/8W | 390R | RES BODY:060 CENTERS:400 |
| 58 | R20 | R1/8W | 390R | RES BODY:060 CENTERS:400 |
| 59 | R21 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 60 | R22 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 61 | R23 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 62 | R24 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 63 | R25 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 64 | R26 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 65 | R27 | R1/8W | 1K | RES BODY:060 CENTERS:400 |
| 66 | R28 | R1/8W | 1R | RES BODY:060 CENTERS:400 |
| 67 | R29 | R1/8W | 1R | RES BODY:060 CENTERS:400 |
| 68 | R30 | R1/8W | 1R | RES BODY:060 CENTERS:400 |
| 69 | U1 | EPM7128SLC84 | | FPGA ALTERA |
| 70 | U2 | 74HCT688 | | 8-BIT IDENTITY COMPARATORS |
| 71 | U3 | OSCDIP14 | 2 MHz | QUARZ OSZILLATOR DIL 14 |
| 72 | U4 | 74ALS574 | | OCTAL D-TYPE EDGE TRIGGERED FLIP-FLOP |
| 73 | U5 | 74ALS574 | | OCTAL D-TYPE EDGE TRIGGERED FLIP-FLOP |
| 74 | U6 | 74ALS574 | | OCTAL D-TYPE EDGE TRIGGERED FLIP-FLOP |
| 75 | U7 | DAC-08 | | 8-BIT HIGH SPEED MULTIPLYING D-TO-A CONVERTER |

### A.1.4 Layouts




IO-Karte, Layout Bauteilseite




IO-Karte, Layout Lötseite

## A.2 Handsteuerbox

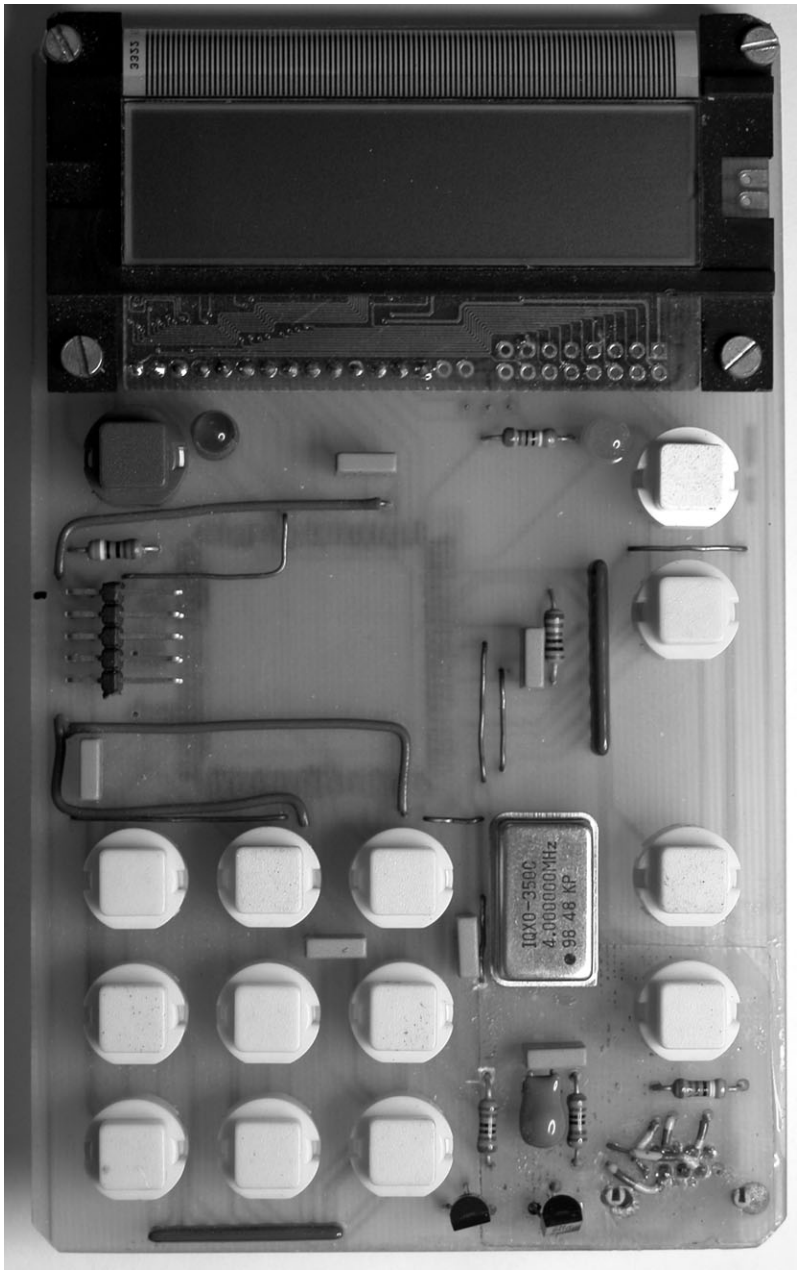
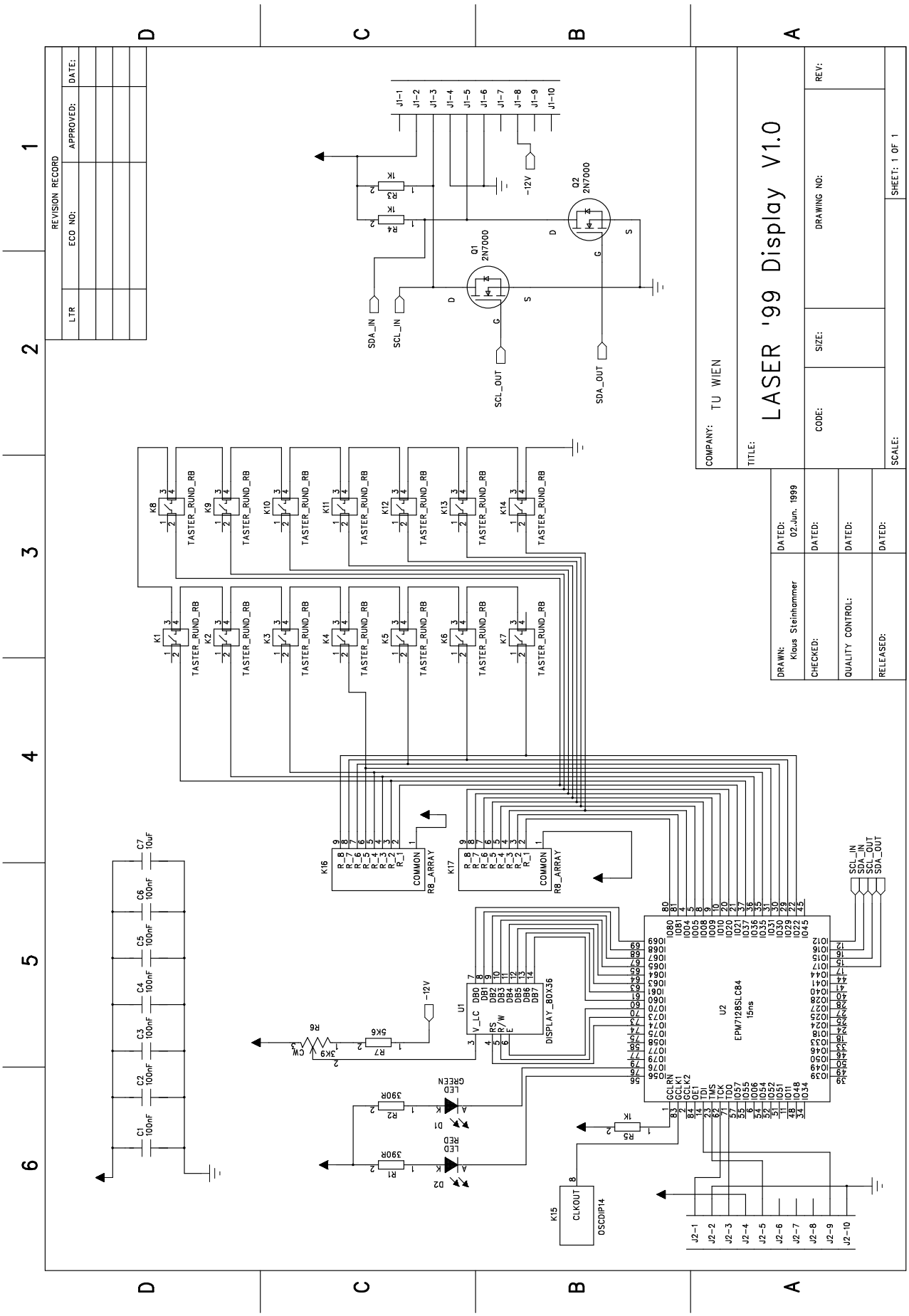


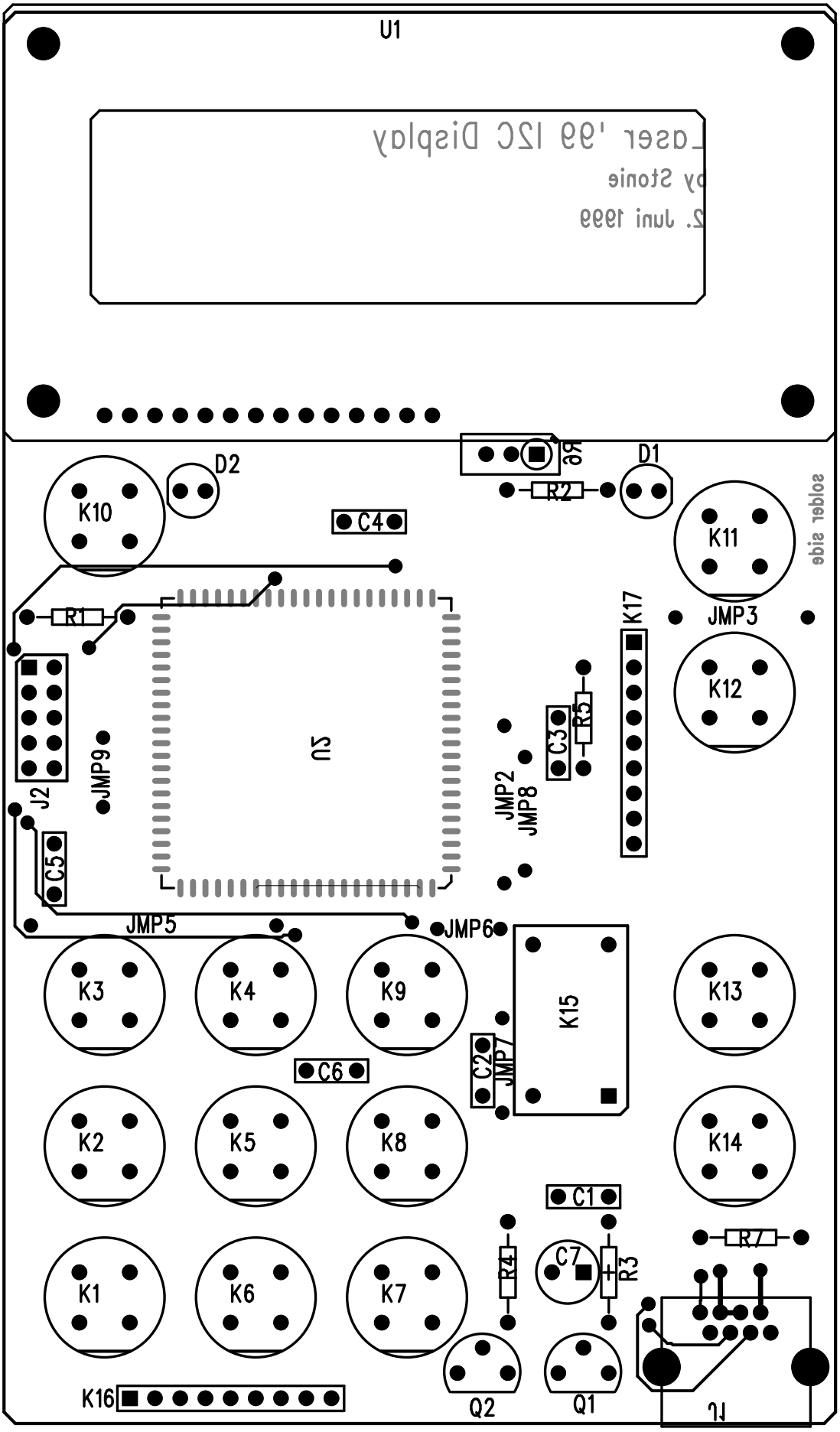

Abbildung A.2: Handsteuerbox, Bestückungsseite

### A.2.1 Schaltplan




Handsteuerbox, Schaltplan

### A.2.2 Bestückungsplan

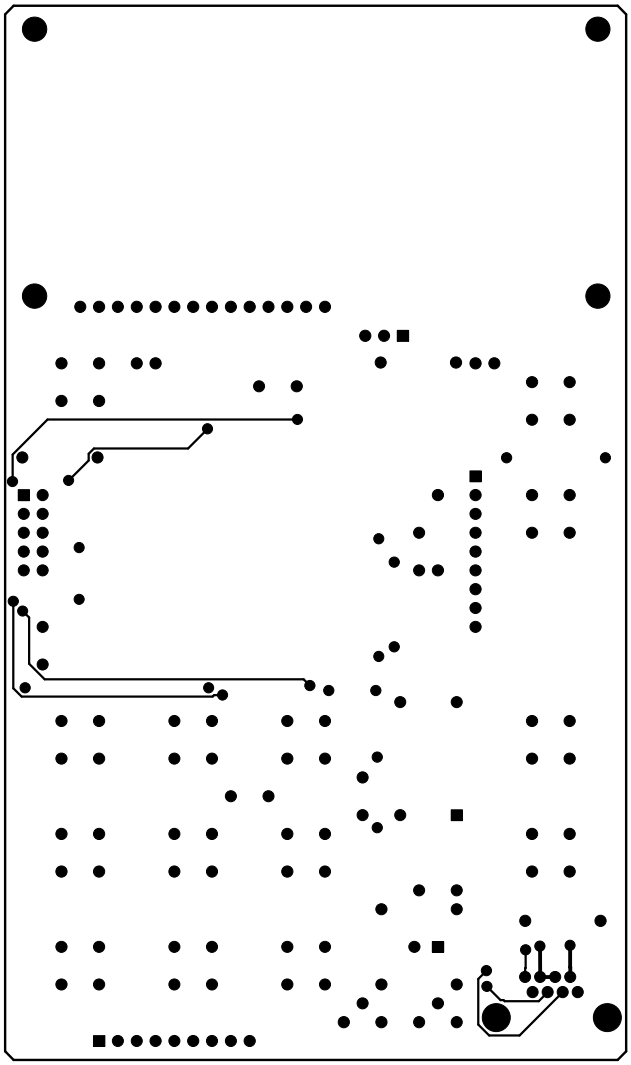



Handsteuerbox, Bestückungsplan

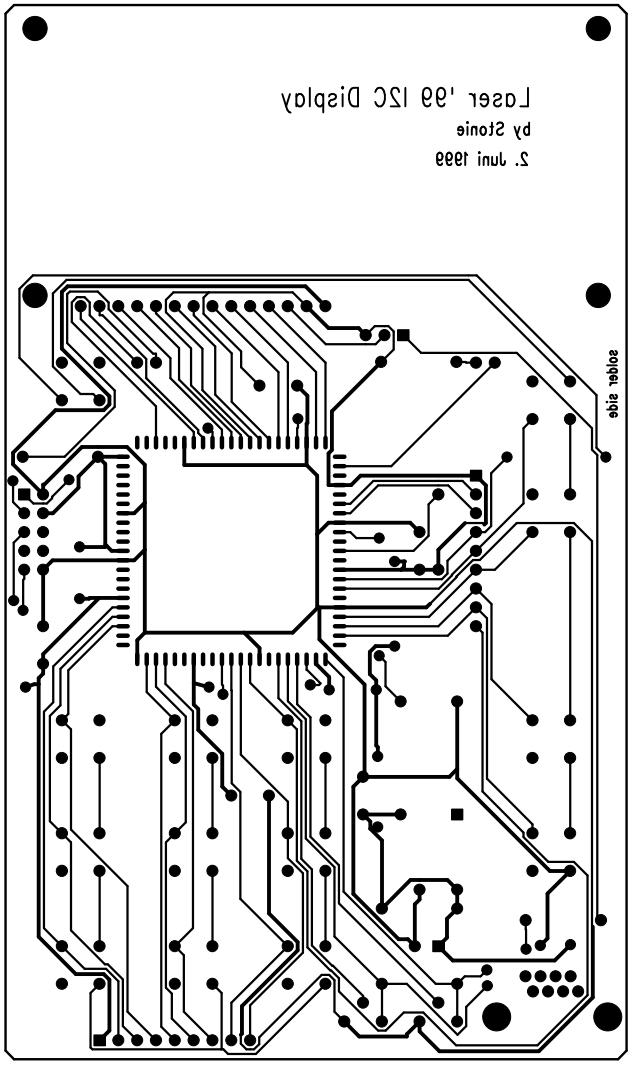
### A.2.3 Liste der verwendeten Bauteile

| No. | Referenz | Name | Wert | Beschreibung |
|---|---|---|---|---|
| 1 | C1 | CAP-CK06 | 100nF | FOLIENKONDENSATOR MKS2 |
| 2 | C2 | CAP-CK06 | 100nF | FOLIENKONDENSATOR MKS2 |
| 3 | C3 | CAP-CK06 | 100nF | FOLIENKONDENSATOR MKS2 |
| 4 | C4 | CAP-CK06 | 100nF | FOLIENKONDENSATOR MKS2 |
| 5 | C5 | CAP-CK06 | 100nF | FOLIENKONDENSATOR MKS2 |
| 6 | C6 | CAP-CK06 | 100nF | FOLIENKONDENSATOR MKS2 |
| 7 | C7 | CAP-CK06 | 10uF | FOLIENKONDENSATOR MKS2 |
| 8 | D1 | LED | | LIGHT EMITTING DIODE |
| 9 | D2 | LED | | LIGHT EMITTING DIODE |
| 10 | J1 | CON-RJ45 | | RJ45 BUCHSE FÜR PRINTMONTAGE |
| 11 | J2 | HEADER10 | | STIFTLEISTE RM 2,54 2*5 PINS |
| 12 | K1 | TASTER_RUND_RB | | TASTER |
| 13 | K2 | TASTER_RUND_RB | | TASTER |
| 14 | K3 | TASTER_RUND_RB | | TASTER |
| 15 | K4 | TASTER_RUND_RB | | TASTER |
| 16 | K5 | TASTER_RUND_RB | | TASTER |
| 17 | K6 | TASTER_RUND_RB | | TASTER |
| 18 | K7 | TASTER_RUND_RB | | TASTER |
| 19 | K8 | TASTER_RUND_RB | | TASTER |
| 20 | K9 | TASTER_RUND_RB | | TASTER |
| 21 | K10 | TASTER_RUND_RB | | TASTER |
| 22 | K11 | TASTER_RUND_RB | | TASTER |
| 23 | K12 | TASTER_RUND_RB | | TASTER |
| 24 | K13 | TASTER_RUND_RB | | TASTER |
| 25 | K14 | TASTER_RUND_RB | | TASTER |
| 26 | K15 | OSCDIP14 | 4 MHz | QUARZ OSZILLATOR DIL 14 |
| 27 | K16 | R8_ARRAY | 10K | RES 8 FACH |
| 28 | K17 | R8_ARRAY | 10K | RES 8 FACH |
| 29 | Q1 | 2N7000 | | N-CHANNEL,ENHANCEMENT MODE, TMOS FET |
| 30 | Q2 | 2N7000 | | N-CHANNEL,ENHANCEMENT MODE, TMOS FET |
| 31 | R1 | RES-1/8W | 390R | RES BODY:060 CENTERS:400 |
| 32 | R2 | RES-1/8W | 390R | RES BODY:060 CENTERS:400 |
| 33 | R3 | RES-1/8W | 1K | RES BODY:060 CENTERS:400 |
| 34 | R4 | RES-1/8W | 1K | RES BODY:060 CENTERS:400 |
| 35 | R5 | RES-1/8W | 1K | RES BODY:060 CENTERS:400 |
| 36 | R6 | VRES-TOP-ADJ_2 | 3K9 | VARIABLE RESISTOR (TOP ADJUST TYPE) |
| 37 | R7 | RES-1/8W | 5K6 | RES BODY:060 CENTERS:400 |
| 38 | U1 | DISPLAY_80x36 | | LCD DISPLAY, HD44780 KOMPATIBEL |
| 39 | U2 | EPM7128SLC84 | | FPGA Altera |

### A.2.4 Layouts



Display, Layout Bauteilseite




Display, Layout Lötseite

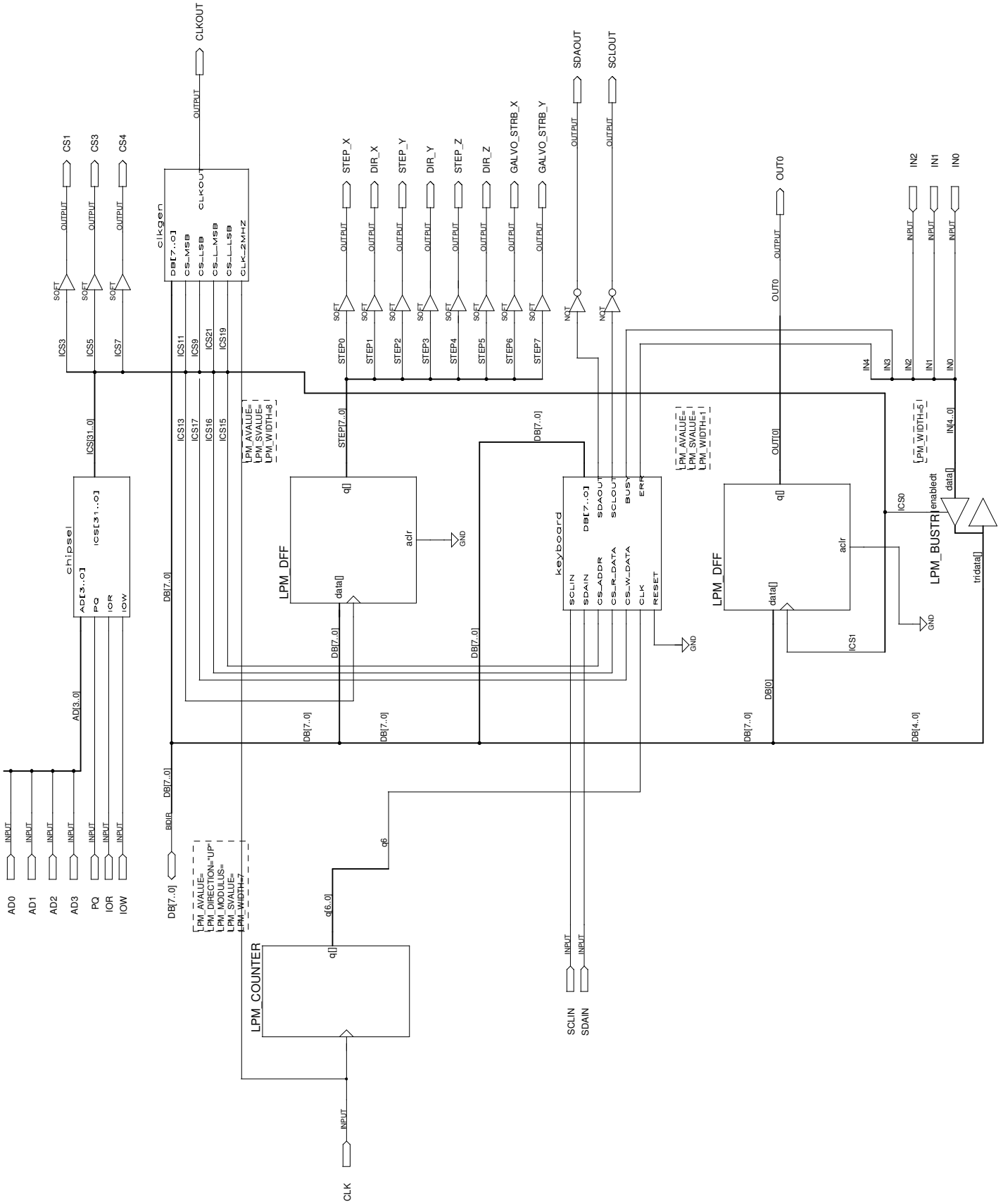
# Anhang B

# EPLD Sourcecode

## B.1 IO-Karte

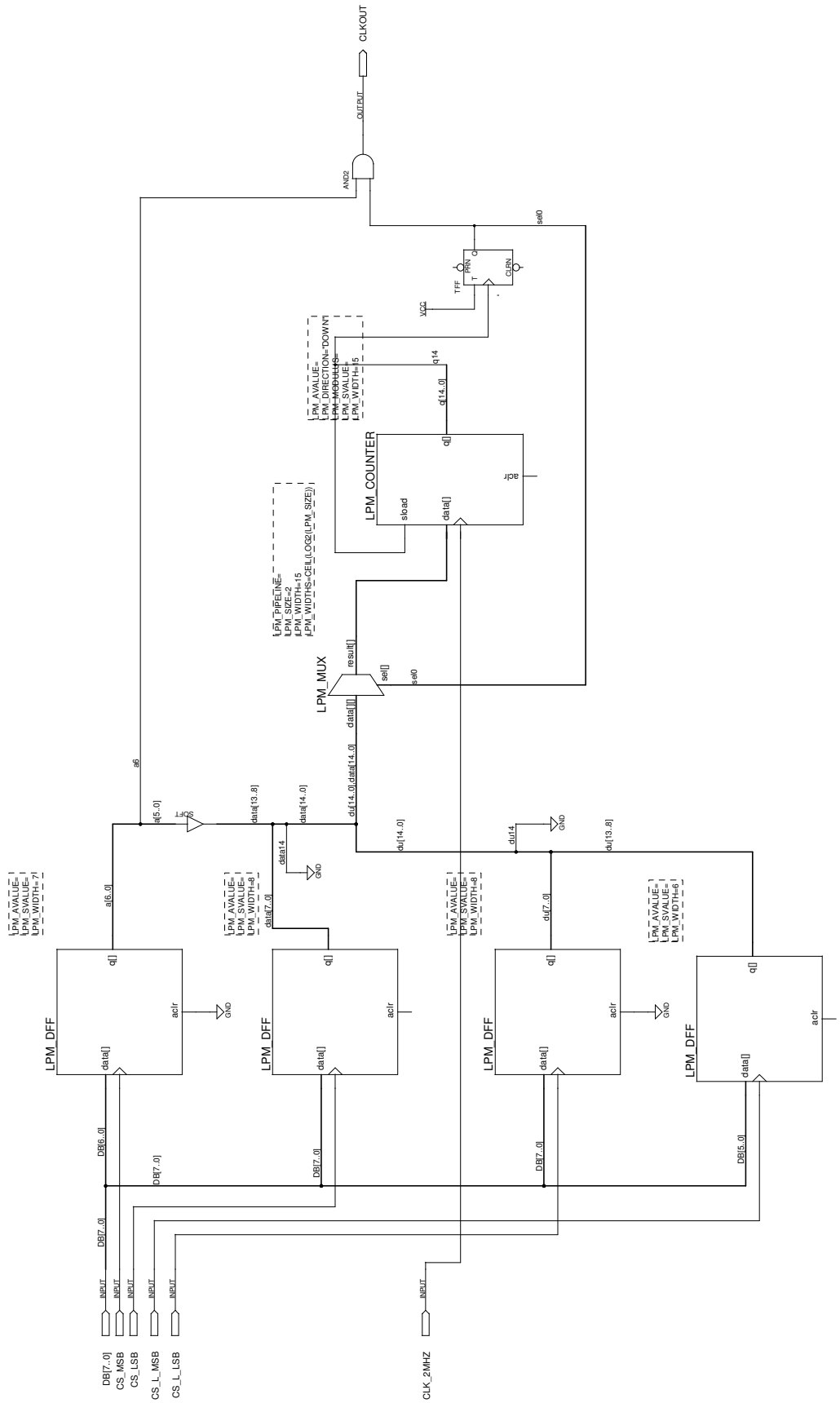
Abbildung B.1: Altera EPLDs

### B.1.1 Top Level Design



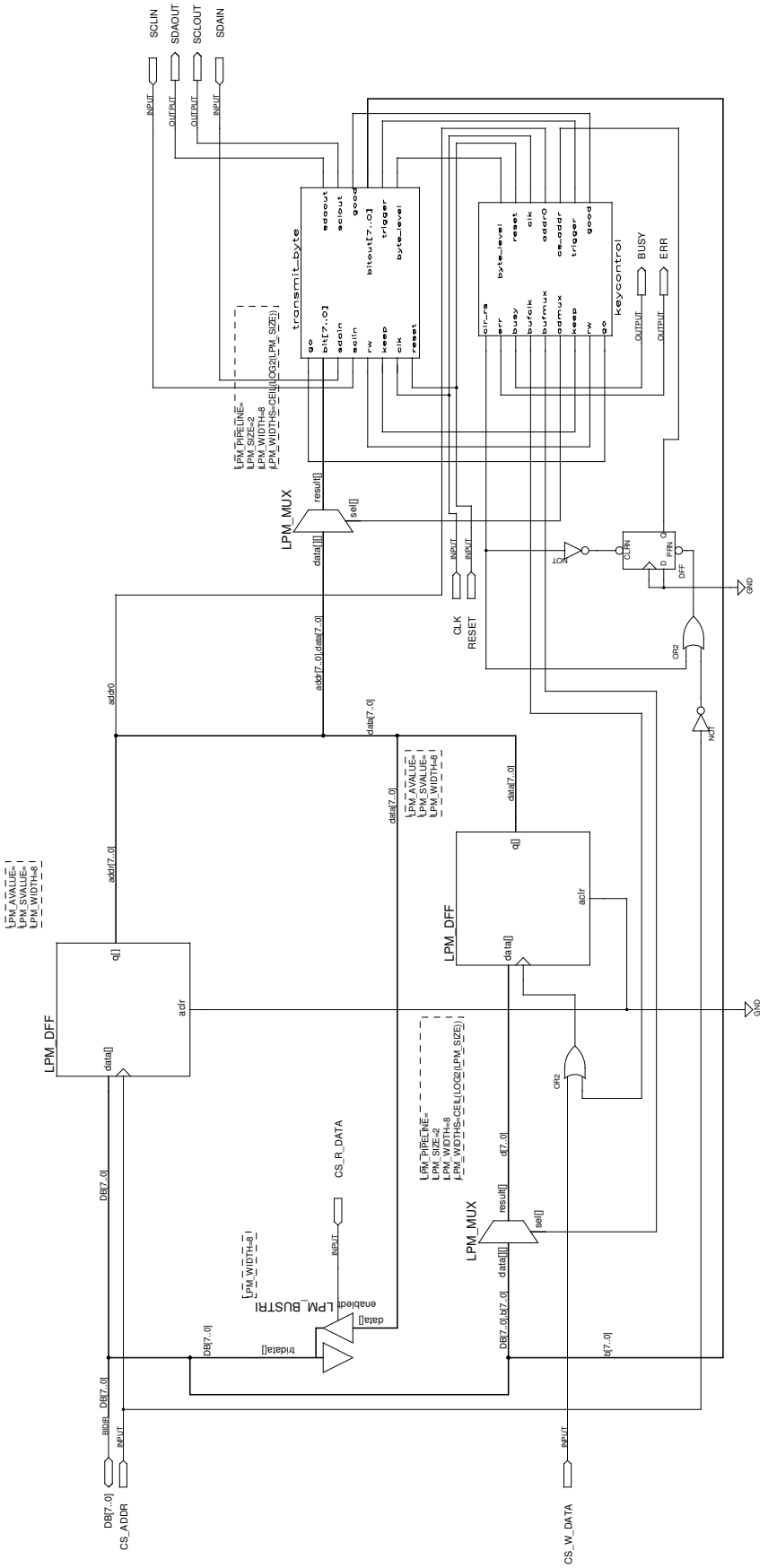

EPLD IO-Karte, Top Level Design

## B.1.2 clkgen.gdf




EPLD IO-Karte, “clkgen” Modul

### B.1.3 keyboard.gdf




EPLD IO-Karte, “keyboard” Modul

## B.1.4 CHIPSEL.tdf

```
SUBDESIGN CHIPSEL
(
    AD[3..0],PQ,IOR,IOW     : INPUT = VCC;
    ICS[31..0]              : OUTPUT;
)
BEGIN
 ICS0  = !AD0 & !AD1 & !AD2 & !AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx0 READ : INPUT LATCH
 ICS1  = !AD0 & !AD1 & !AD2 & !AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx0 WRITE: OUTPUT LATCH
!ICS2  =  AD0 & !AD1 & !AD2 & !AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx1 READ : NC
!ICS3  =  AD0 & !AD1 & !AD2 & !AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx1 WRITE: D/A CONVERTER
!ICS4  = !AD0 &  AD1 & !AD2 & !AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx2 READ : NC
!ICS5  = !AD0 &  AD1 & !AD2 & !AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx2 WRITE: GALVO LSB
!ICS6  =  AD0 &  AD1 & !AD2 & !AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx3 READ : NC
!ICS7  =  AD0 &  AD1 & !AD2 & !AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx3 WRITE: GALVO MSB

 ICS8  = !AD0 & !AD1 &  AD2 & !AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx4 READ : NC
 ICS9  = !AD0 & !AD1 &  AD2 & !AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx4 WRITE: CLKGEN LSB
 ICS10 =  AD0 & !AD1 &  AD2 & !AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx5 READ : NC
 ICS11 =  AD0 & !AD1 &  AD2 & !AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx5 WRITE: CLKGEN MSB
 ICS12 = !AD0 &  AD1 &  AD2 & !AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx6 READ : NC
 ICS13 = !AD0 &  AD1 &  AD2 & !AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx6 WRITE: STEPPER
 ICS14 =  AD0 &  AD1 &  AD2 & !AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx7 READ : NC
 ICS15 =  AD0 &  AD1 &  AD2 & !AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx7 WRITE: KEYBOARD ADDR

 ICS16 = !AD0 & !AD1 & !AD2 &  AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx8 READ : KEYBOARD DATA
 ICS17 = !AD0 & !AD1 & !AD2 &  AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx8 WRITE: KEYBOARD DATA
 ICS18 =  AD0 & !AD1 & !AD2 &  AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxx9 READ : NC
 ICS19 =  AD0 & !AD1 & !AD2 &  AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxx9 WRITE: CLK_DUTY_CYCLE
 ICS20 = !AD0 &  AD1 & !AD2 &  AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxxA READ : NC
 ICS21 = !AD0 &  AD1 & !AD2 &  AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxxA WRITE: NC
 ICS22 =  AD0 &  AD1 & !AD2 &  AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxxB READ : NC
 ICS23 =  AD0 &  AD1 & !AD2 &  AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxxB WRITE: NC

 ICS24 = !AD0 & !AD1 &  AD2 &  AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxxC READ : NC
 ICS25 = !AD0 & !AD1 &  AD2 &  AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxxC WRITE: NC
 ICS26 =  AD0 & !AD1 &  AD2 &  AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxxD READ : NC
 ICS27 =  AD0 & !AD1 &  AD2 &  AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxxD WRITE: NC
 ICS28 = !AD0 &  AD1 &  AD2 &  AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxxE READ : NC
 ICS29 = !AD0 &  AD1 &  AD2 &  AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxxE WRITE: NC
 ICS30 =  AD0 &  AD1 &  AD2 &  AD3 & !PQ & !IOR &  IOW; -- ADRESS 0xxxF READ : NC
 ICS31 =  AD0 &  AD1 &  AD2 &  AD3 & !PQ &  IOR & !IOW; -- ADRESS 0xxxF WRITE: NC

END;
```

## B.1.5 transmit_byte.tdf

```
FUNCTION transmit_bit (t_start, t_bit, t_sclin, t_clk, t_reset) RETURNS (bit_sdaout, bit_sclout, MACHINE bit_level);


SUBDESIGN transmit_byte
(
    go, bit[7..0], sdain, sclin, rw, keep, clk, reset   : INPUT;
    sdaout, sclout, good, bitout[7..0], trigger         : OUTPUT;
    byte_level                                          : MACHINE OUTPUT;
)
VARIABLE
    tt  : MACHINE WITH STATES (RST,START,START_1,BIT_7,BIT_6,BIT_5,
        BIT_4,BIT_3,BIT_2,BIT_1,BIT_0,ACK,ACK_1,ACK_2,ACK_3,STOP,STOP_1,STOP_2,KEEP_1);
    bit_sm : transmit_bit;
    bit_start, bit_bit : node;
    storage[7..0] : dff;

begin
    byte_level = tt;
    bit_sm.(t_start, t_bit, t_sclin, t_clk, t_reset) = (bit_start, bit_bit, sclin, clk, reset);
    CASE tt IS
        WHEN RST=>
            if go == VCC then tt = START;
            else tt = RST;
            end if;
            sdaout = VCC;
            sclout = VCC;
            bit_bit = GND;
            bit_start = GND;
            good = GND;
            trigger = GND;
            storage[7..0].d = H"00";
        WHEN START=>
            tt = START_1;
            sdaout = GND;
            sclout = VCC;
            bit_bit = GND;
            bit_start = GND;
            good = GND;
            trigger = GND;
```

```
    storage[7..0].d = H"00";
WHEN START_1=>
    tt = BIT_7;
    if rw == GND then
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[7];
    bit_start = VCC;
    good = GND;
    trigger = GND;
    storage[7..0].d = H"00";
WHEN BIT_7=>
    if bit_sm.bit_level == A4 then
        tt = BIT_6;
        bit_start = VCC;
        storage7.d = storage7.q;
    elsif bit_sm.bit_level == A3 and rw == VCC then
        tt = BIT_7;
        bit_start = GND;
        storage7.d = sdain;
    else
        tt = BIT_7;
        bit_start = GND;
        storage7.d = storage7.q;
    end if;
    if rw == GND then
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    storage[6..0].d = storage[6..0].q;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[7];
    good = GND;
    trigger = GND;

WHEN BIT_6=>
    if bit_sm.bit_level == A4 then
        tt = BIT_5;
        bit_start = VCC;
        storage6.d = storage6.q;
    elsif bit_sm.bit_level == A3 and rw == VCC then
        tt = BIT_6;
        bit_start = GND;
        storage6.d = sdain;
    else
        tt = BIT_6;
        bit_start = GND;
        storage6.d = storage6.q;
    end if;
    if rw == GND then
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    storage7.d = storage7.q;
    storage[5..0].d = storage[5..0].q;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[6];
    good = GND;
    trigger = GND;

WHEN BIT_5=>
    if bit_sm.bit_level == A4 then
        tt = BIT_4;
        bit_start = VCC;
        storage5.d = storage5.q;
    elsif bit_sm.bit_level == A3 and rw == VCC then
        tt = BIT_5;
        bit_start = GND;
        storage5.d = sdain;
    else
        tt = BIT_5;
        bit_start = GND;
        storage5.d = storage5.q;
    end if;
    if rw == GND then
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    storage[7..6].d = storage[7..6].q;
    storage[4..0].d = storage[4..0].q;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[5];
    good = GND;
    trigger = GND;

WHEN BIT_4=>
```

```
    if bit_sm.bit_level == A4 then
        tt = BIT_3;
        bit_start = VCC;
        storage4.d = storage4.q;
    elsif bit_sm.bit_level == A3 and rw == VCC then
        tt = BIT_4;
        bit_start = GND;
        storage4.d = sdain;
    else
        tt = BIT_4;
        bit_start = GND;
        storage4.d = storage4.q;
    end if;
    if rw == GND then
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    storage[7..5].d = storage[7..5].q;
    storage[3..0].d = storage[3..0].q;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[4];
    good = GND;
    trigger = GND;

WHEN BIT_3=>
    if bit_sm.bit_level == A4 then
        tt = BIT_2;
        bit_start = VCC;
        storage3.d = storage3.q;
    elsif bit_sm.bit_level == A3 and rw == VCC then
        tt = BIT_3;
        bit_start = GND;
        storage3.d = sdain;
    else
        tt = BIT_3;
        bit_start = GND;
        storage3.d = storage3.q;
    end if;
    if rw == GND then
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    storage[7..4].d = storage[7..4].q;
    storage[2..0].d = storage[2..0].q;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[3];
    good = GND;
    trigger = GND;

WHEN BIT_2=>
    if bit_sm.bit_level == A4 then
        tt = BIT_1;
        bit_start = VCC;
        storage2.d = storage2.q;
    elsif bit_sm.bit_level == A3 and rw == VCC then
        tt = BIT_2;
        bit_start = GND;
        storage2.d = sdain;
    else
        tt = BIT_2;
        bit_start = GND;
        storage2.d = storage2.q;
    end if;
    if rw == GND then
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    storage[7..3].d = storage[7..3].q;
    storage[1..0].d = storage[1..0].q;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[2];
    good = GND;
    trigger = GND;

WHEN BIT_1=>
    if bit_sm.bit_level == A4 then
        tt = BIT_0;
        bit_start = VCC;
        storage1.d = storage1.q;
    elsif bit_sm.bit_level == A3 and rw == VCC then
        tt = BIT_1;
        bit_start = GND;
        storage1.d = sdain;
    else
        tt = BIT_1;
        bit_start = GND;
        storage1.d = storage1.q;
    end if;
    if rw == GND then
```

```
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    storage[7..2].d = storage[7..2].q;
    storage0.d = storage0.q;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[1];
    good = GND;
    trigger = GND;

WHEN BIT_0=>
    if bit_sm.bit_level == A4 then
        tt = ACK;
        storage0.d = storage0.q;
    elsif bit_sm.bit_level == A3 and rw == VCC then
        tt = BIT_0;
        storage0.d = sdain;
    else
        tt = BIT_0;
        storage0.d = storage0.q;
    end if;
    if rw == GND then
        sdaout = bit_sm.bit_sdaout;
    else
        sdaout = VCC;
    end if;
    storage[7..1].d = storage[7..1].q;
    sclout = bit_sm.bit_sclout;
    bit_bit = bit[0];
    bit_start = GND;
    good = GND;
    trigger = GND;

WHEN ACK=>
    tt = ACK_1;
    if rw == VCC then
        sdaout = GND;
    else
        sdaout = VCC;
    end if;
    trigger = GND;
    sclout = GND;
    bit_bit = GND;
    bit_start = GND;
    good = GND;
    storage[7..0].d = storage[7..0].q;
WHEN ACK_1=>
    if sclin == VCC then
        tt = ACK_2;
    else
        tt = ACK_1;
    end if;
    if rw == VCC then
        sdaout = GND;
    else
        sdaout = VCC;
    end if;
    sclout = VCC;
    bit_bit = GND;
    bit_start = GND;
    good = !sdain;
    trigger = VCC;
    storage[7..0].d = storage[7..0].q;
WHEN ACK_2=>
    tt = ACK_3;
    if rw == VCC then
        sdaout = GND;
    else
        sdaout = VCC;
    end if;
    sclout = VCC;
    bit_bit = GND;
    bit_start = GND;
    good = !sdain;
    trigger = GND;
    storage[7..0].d = storage[7..0].q;
WHEN ACK_3=>
    if keep == VCC then
        tt = KEEP_1;
    else
        tt = STOP;
    end if;
    if rw == VCC then
        sdaout = GND;
    else
        sdaout = VCC;
    end if;
    sclout = GND;
    bit_bit = GND;
    bit_start = GND;
    good = GND;
```

```
            trigger = GND;
            storage[7..0].d = storage[7..0].q;
        WHEN STOP=>
            tt = STOP_1;
            sdaout = GND;
            sclout = GND;
            bit_bit = GND;
            bit_start = GND;
            good = GND;
            trigger = GND;
            storage[7..0].d = H"00";
        WHEN STOP_1=>
            tt = STOP_2;
            sdaout = GND;
            sclout = VCC;
            bit_bit = GND;
            bit_start = GND;
            good = GND;
            trigger = GND;
            storage[7..0].d = H"00";
        WHEN STOP_2=>
            tt = RST;
            sdaout = VCC;
            sclout = VCC;
            bit_bit = GND;
            bit_start = GND;
            good = GND;
            trigger = GND;
            storage[7..0].d = H"00";

        WHEN KEEP_1=>
            if go == VCC then tt = START_1;
            else tt = KEEP_1;
            end if;
            sdaout = GND;
            sclout = GND;
            bit_bit = GND;
            bit_start = GND;
            good = GND;
            trigger = GND;
            storage[7..0].d = H"00";

        WHEN OTHERS=>
            tt = RST;
            sdaout = VCC;
            sclout = VCC;
            bit_bit = GND;
            bit_start = GND;
            good = GND;
            trigger = GND;
            storage[7..0].d = H"00";
    END CASE;

    bitout[7..0] = storage[7..0].q;

    tt.(clk, reset) = (clk, reset);
    storage[7..0].clrn=!reset;
    storage[7..0].clk= clk;
END;
```

## B.1.6 transmit_bit.tdf

```
SUBDESIGN transmit_bit
(
    t_start, t_bit, t_sclin, t_clk, t_reset : INPUT;
    bit_sdaout, bit_sclout                  : OUTPUT;
    bit_level                               : MACHINE OUTPUT;
)
VARIABLE
    ss  : MACHINE WITH STATES (A1,A2,A3,A4);

%   SDA     __/--------\__  %
%   SCL     _____/--\_____  %
%           A1 A2 A3 A4 A1                              %

begin
    bit_level = ss;
    CASE ss IS
        WHEN A1=>
            if t_start == VCC then ss = A2;
            else ss = A1;
            end if;
            bit_sdaout = GND;
            bit_sclout = GND;
        WHEN A2=>
            if t_sclin == VCC then ss = A2;
            else ss = A3;
            end if;
```

```
            bit_sdaout = t_bit;
            bit_sclout = GND;
        WHEN A3=>
            if t_sclin == GND then ss = A3;
            else ss = A4;
            end if;
            bit_sdaout = t_bit;
            bit_sclout = VCC;
        WHEN A4=>
            if t_sclin == VCC then ss = A4;
            elsif t_start == VCC then ss = A2;
            else ss = A1;
            end if;
            bit_sdaout = t_bit;
            bit_sclout = GND;
        WHEN OTHERS=>
            ss = A1;
            bit_sdaout = GND;
            bit_sclout = GND;
    END CASE;

    ss.(clk, reset) = (t_clk, t_reset);
END;
```

## B.1.7 keycontrol.tdf

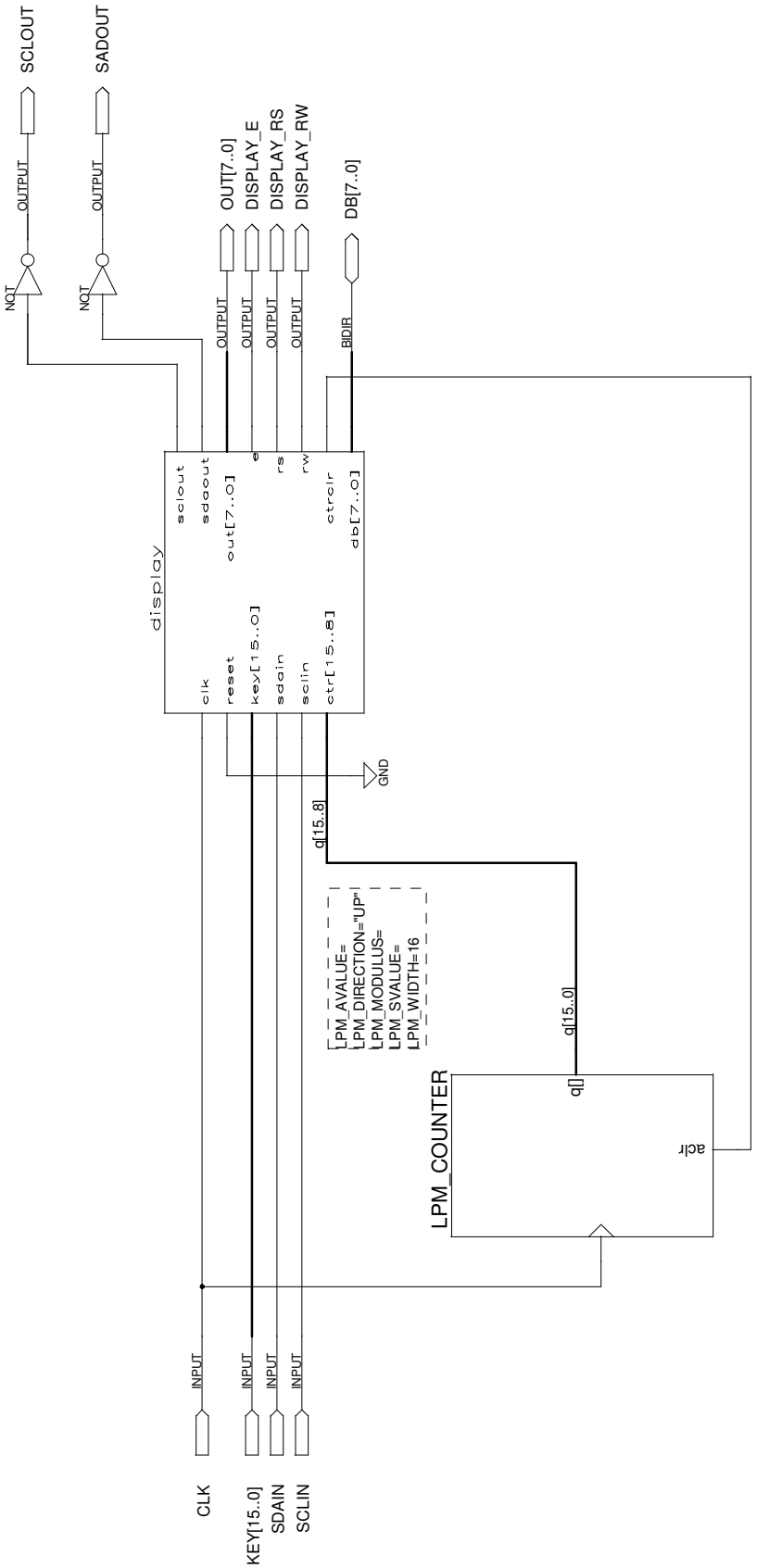
```
SUBDESIGN keycontrol
(
    good, trigger, cs_addr, addr0, clk ,reset               : INPUT;
    go, rw, keep, admux, bufmux, bufclk, busy, err, clr_rs  : OUTPUT;
    byte_level                                              : MACHINE INPUT;
)
VARIABLE
    uu  : MACHINE WITH STATES (START,E_RROR,TX,TX_1,READ,READ_1,L_DFF,WRITE,WRITE_1);

begin
    CASE uu IS
        WHEN START=>
            if cs_addr == VCC then
                uu = TX;
            else
                uu = START;
            end if;
            err = GND;
            busy = GND;

            bufclk = GND;
            bufmux = VCC;
            admux = VCC;

            keep = GND;
            rw = GND;
            go = GND;
            clr_rs = GND;

        WHEN TX=>
            if byte_level == BIT_7 then
                uu = TX_1;
            else
                uu = TX;
            end if;
            err = GND;
            busy = VCC;

            bufclk = GND;
            bufmux = VCC;
            admux = VCC;

            keep = VCC;
            rw = GND;
            go = VCC;
            clr_rs = VCC;

        WHEN TX_1=>
            if trigger == VCC and good == GND then
                uu =E_RROR;
            elsif byte_level == KEEP_1 and addr0 == GND then
                uu = WRITE;
            elsif byte_level == KEEP_1 and addr0 == VCC then
                uu = READ;
            else
                uu = TX_1;
            end if;
            err = GND;
            busy = VCC;

            bufclk = GND;
            bufmux = VCC;
```

```
    admux = VCC;

    keep = VCC;
    rw = GND;
    go = GND;
    clr_rs = GND;

WHEN READ=>
    if byte_level == BIT_7 then
        uu = READ_1;
    else
        uu = READ;
    end if;
    err = GND;
    busy = VCC;

    bufclk = GND;
    bufmux = VCC;
    admux = VCC;

    keep = GND;
    rw = VCC;
    go = VCC;
    clr_rs = GND;

WHEN READ_1=>
    if trigger == VCC and good == GND then
        uu = E_RROR;
    elsif byte_level == ACK_2 then
        uu = L_DFF;
    else
        uu = READ_1;
    end if;
    err = GND;
    busy = VCC;

    bufclk = GND;
    bufmux = GND;
    admux = VCC;

    keep = GND;
    rw = VCC;
    go = GND;
    clr_rs = GND;

WHEN L_DFF=>
    uu = START;
    err = GND;
    busy = VCC;

    bufclk = VCC;
    bufmux = GND;
    admux = GND;

    keep = GND;
    rw = GND;
    go = GND;
    clr_rs = GND;

WHEN WRITE=>
    if byte_level == BIT_7 then
        uu = WRITE_1;
    else
        uu = WRITE;
    end if;
    err = GND;
    busy = VCC;

    bufclk = GND;
    bufmux = VCC;
    admux = GND;

    keep = GND;
    rw = GND;
    go = VCC;
    clr_rs = GND;

WHEN WRITE_1=>
    if trigger == VCC and good == GND then
        uu = E_RROR;
    elsif byte_level == RST then
        uu = START;
    else
        uu = WRITE_1;
    end if;
    err = GND;
    busy = VCC;

    bufclk = GND;
    bufmux = VCC;
    admux = GND;
```

```
            keep = GND;
            rw = GND;
            go = GND;
            clr_rs = GND;

        WHEN E_RROR=>
            if cs_addr == VCC then
                uu = TX;
            else
                uu = E_RROR;
            end if;
            err = VCC;
            busy = GND;

            bufclk = GND;
            bufmux = VCC;
            admux = VCC;

            keep = GND;
            rw = GND;
            go = GND;
            clr_rs = GND;

        WHEN OTHERS=>
            uu = E_RROR;
            err = GND;
            busy = GND;
            bufclk = GND;
            bufmux = VCC;
            admux = VCC;
            keep = GND;
            rw = GND;
            go = GND;
            clr_rs = GND;
    END CASE;

    uu.(clk, reset) = (clk, reset);
END;
```

## B.2 Handsteuergerät

### B.2.1 Top Level Design




EPLD Display, Top Level Design

## B.2.2 display.tdf

```
FUNCTION display_out (clk, reset, data[7..0],go,rs_in,db7_in) RETURNS (db[7..0],e,rs_out,rw,finish);
FUNCTION i2c (clk,reset,sclin,sdain,datain[7..0]) RETURNS (trigger,addr,dataout[7..0],sclout,sdaout);

SUBDESIGN display
(
    clk, reset, key[15..0], sdain, sclin, ctr[15..8]    : INPUT;
    sclout, sdaout, out[7..0], e, rs, rw, ctrclr                : OUTPUT;
    db[7..0]                                                    : BIDIR;
)

VARIABLE
    ee: MACHINE WITH STATES (Start,WaitForSer,ReadAddr,WaitForData,WriteDisplay,WriteFinish,
        WriteCommand,CommandFinish,WriteLatch,ReadKeys,Step2,Step3,Step4,Step5,Step6,Step7,
        Step8,Step9,Step10,Step11,Step12,Step13,Step14,Step15,Step16,Step17,Step18,Step19,
        Step20,Step21,Step22,Step23);

    outff[7..0]:dff;
    storage[8..0] : dff;
    serial : i2c;
    dp: display_out;
    keysel,init,idb[7..0],ie:NODE;
    keyff[7..0]: dff;

BEGIN
    dp.(clk, reset) = (clk,reset);
    serial.(clk,reset,sclin,sdain) = (clk,reset,sclin,sdain);

    sclout = serial.sclout;
    sdaout = serial.sdaout;
    serial.datain[7..0] = keyff[7..0].q;

    db7 = tri(idb7,(!init or !dp.rw));
    db6 = tri(idb6,(!init or !dp.rw));
    db5 = tri(idb5,(!init or !dp.rw));
    db4 = tri(idb4,(!init or !dp.rw));
    db3 = tri(idb3,(!init or !dp.rw));
    db2 = tri(idb2,(!init or !dp.rw));
    db1 = tri(idb1,(!init or !dp.rw));
    db0 = tri(idb0,(!init or !dp.rw));
    if init == VCC then
        e  = dp.e;
        rs = dp.rs_out;
        rw = dp.rw;
    else
        e  = ie;
        rs = GND;
        rw = GND;
    end if;
    dp.db7_in = db7;

    if keysel == GND then
        keyff[7..0].d = key[7..0];
    else
        keyff[7..0].d = key[15..8];
    end if;

    out[7..0] = outff[7..0].q;

    CASE ee IS

        WHEN Start=>
            idb[7..0] = H"38";
            ie = GND;
            init = GND;
            ctrclr = VCC;
            ee = Step2;
--          ee = WaitForSer;                -- just for simulation
            dp.go = GND;
            dp.rs_in = GND;
            dp.data[7..0]=H"00";
            outff[7..0].d = H"00";
            storage[8..0].d = H"000";
            keysel = GND;

        WHEN Step2=>                        -- wait for 15ms after power on
            if ctr[15..8] == H"F0" then
                ee = Step3;
            else
                ee = Step2;
            end if;
            idb[7..0] = H"38";
            ie = GND;
            init = GND;
            ctrclr = GND;
            dp.go = GND;
            dp.rs_in = GND;
            dp.data[7..0]=H"00";
            outff[7..0].d = H"00";
            storage[8..0].d = H"000";
```

```
    keysel = GND;

WHEN Step3=>
    idb[7..0] = H"38";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step4;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step4=>
    idb[7..0] = H"38";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step5;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step5=>                -- wait for 4 ms
    idb[7..0] = H"38";
    ie  = GND;
    init = GND;
    ctrclr = GND;
    if ctr[15..8] == H"40" then
        ee = Step6;
    else
        ee = Step5;
    end if;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step6=>
    idb[7..0] = H"38";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step7;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step7=>
    idb[7..0] = H"38";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step8;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step8=>                    -- wait for 100 us
    idb[7..0] = H"38";
    ie  = GND;
    init = GND;
    ctrclr = GND;
    if ctr[15..8] == H"02" then
        ee = Step9;
    else
        ee = Step8;
    end if;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step9=>
    idb[7..0] = H"38";
    ie  = VCC;
    init = GND;
```

```
    ctrclr = VCC;
    ee = Step10;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step10=>
    idb[7..0] = H"38";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step11;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step11=>                   -- wait for 50 us
    idb[7..0] = H"38";
    ie  = GND;
    init = GND;
    ctrclr = GND;
    if ctr[15..8] == H"01" then
        ee = Step12;
    else
        ee = Step11;
    end if;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step12=>
    idb[7..0] = H"38";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step13;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step13=>
    idb[7..0] = H"38";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step14;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step14=>                   -- wait for 50 us
    idb[7..0] = H"38";
    ie  = GND;
    init = GND;
    ctrclr = GND;
    if ctr[15..8] == H"01" then
        ee = Step15;
    else
        ee = Step14;
    end if;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step15=>
    idb[7..0] = H"06";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step16;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
```

```
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step16=>
    idb[7..0] = H"06";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step17;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step17=>                   -- wait for 50 us
    idb[7..0] = H"06";
    ie  = GND;
    init = GND;
    ctrclr = GND;
    if ctr[15..8] == H"01" then
        ee = Step18;
    else
        ee = Step17;
    end if;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step18=>
    idb[7..0] = H"0E";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step19;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step19=>
    idb[7..0] = H"0E";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step20;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step20=>                   -- wait for 50 us
    idb[7..0] = H"0E";
    ie  = GND;
    init = GND;
    ctrclr = GND;
    if ctr[15..8] == H"01" then
        ee = Step21;
    else
        ee = Step20;
    end if;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step21=>
    idb[7..0] = H"01";
    ie  = VCC;
    init = GND;
    ctrclr = VCC;
    ee = Step22;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step22=>
    idb[7..0] = H"01";
    ie  = VCC;
```

```
    init = GND;
    ctrclr = VCC;
    ee = Step23;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN Step23=>                   -- wait for 50 us
    idb[7..0] = H"01";
    ie  = GND;
    init = GND;
    ctrclr = GND;
    if ctr[15..8] == H"01" then
        ee = WaitForSer;
    else
        ee = Step23;
    end if;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = H"00";
    storage[8..0].d = H"000";
    keysel = GND;

WHEN WaitForSer=>
    if serial.addr == VCC then
        ee = ReadAddr;
    else
        ee = WaitForSer;
    end if;
    storage8.d=serial.dataout0;
    if serial.dataout[7..4] == H"5" or serial.dataout[7..5] == H"3" or serial.dataout7 == VCC then
        storage7.d=VCC;                 -- 28h - 7fh
        storage6.d=VCC;
        storage5.d=(serial.dataout7 and  serial.dataout4);
        storage4.d=(serial.dataout6 and  serial.dataout5 and  serial.dataout4) or (serial.dataout7 and  !serial.dataout4);
        storage3.d=!serial.dataout4;
        storage[2..0].d = serial.dataout[3..1];
    else                            -- 00h - 27h
        storage7.d = VCC;
        storage6.d = GND;
        storage[5..0].d = serial.dataout[6..1];
    end if;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = outff[7..0].q;
    keysel = serial.dataout1;
    idb[7..0] = dp.db[7..0];
    ie  = GND;
    init = VCC;
    ctrclr = VCC;

WHEN ReadAddr=>
    if storage8.q == GND and storage[7..0].q != H"E7" and storage[7..0].q != H"A7" then
        dp.go = VCC;                    -- valid display address
    else
        dp.go = GND;
    end if;
    if storage8.q == GND then           -- command
        ee = WaitForData;
        keysel = GND;
    else
        ee = ReadKeys;                  -- read keys
        keysel = storage0.q;
    end if;
    dp.rs_in = GND;
    dp.data[7..0]=storage[7..0].q;
    storage[8..0].d = storage[8..0].q;
    outff[7..0].d = outff[7..0].q;
    idb[7..0] = dp.db[7..0];
    ie  = GND;
    init = VCC;
    ctrclr = VCC;

WHEN WaitForData=>
    if serial.trigger == GND then
        ee = WaitForData;
        storage[8..0].d = storage[8..0].q;
    elsif storage[7..0].q == H"A7" then --  write to latch
        ee = WriteLatch;
        storage[7..0].d = serial.dataout[7..0];
        storage8.d = storage8.q;
    elsif storage[7..0].q == H"E7" then --  write command to display
        ee = WriteCommand;
        storage[7..0].d = serial.dataout[7..0];
        storage8.d = storage8.q;
    else
        ee = WriteDisplay;              --  write data to display
```

```
        storage[7..0].d = serial.dataout[7..0];
        storage8.d = storage8.q;
    end if;
    keysel = GND;
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=storage[7..0].q;
    outff[7..0].d = outff[7..0].q;
    idb[7..0] = dp.db[7..0];
    ie  = GND;
    init = VCC;
    ctrclr = VCC;

WHEN WriteDisplay=>
    dp.go = VCC;
    dp.rs_in = VCC;
    dp.data[7..0]=storage[7..0].q;
    outff[7..0].d = outff[7..0].q;
    storage[8..0].d = storage[8..0].q;
    keysel = GND;
    ee = WriteFinish;
    idb[7..0] = dp.db[7..0];
    ie  = GND;
    init = VCC;
    ctrclr = VCC;

WHEN WriteFinish=>
    dp.go = GND;
    dp.rs_in = VCC;
    dp.data[7..0]=storage[7..0].q;
    outff[7..0].d = outff[7..0].q;
    storage[8..0].d = storage[8..0].q;
    keysel = GND;
    if dp.finish == GND then
        ee = WriteFinish;
    else
        ee =WaitForSer;
    end if;
    idb[7..0] = dp.db[7..0];
    ie  = GND;
    init = VCC;
    ctrclr = VCC;

WHEN WriteCommand=>
    dp.go = VCC;
    dp.rs_in = GND;
    dp.data[7..0]=storage[7..0].q;
    outff[7..0].d = outff[7..0].q;
    storage[8..0].d = storage[8..0].q;
    keysel = GND;
    ee = CommandFinish;
    idb[7..0] = dp.db[7..0];
    ie  = GND;
    init = VCC;
    ctrclr = VCC;

WHEN CommandFinish=>
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=storage[7..0].q;
    outff[7..0].d = outff[7..0].q;
    storage[8..0].d = storage[8..0].q;
    keysel = GND;
    if dp.finish == GND then
        ee = CommandFinish;
    else
        ee =WaitForSer;
    end if;
    idb[7..0] = dp.db[7..0];
    ie  = GND;
    init = VCC;
    ctrclr = VCC;

WHEN WriteLatch=>
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = storage[7..0].q;
    storage[8..0].d = storage[8..0].q;
    keysel = GND;
    ee = WaitForSer;
    idb[7..0] = dp.db[7..0];
    ie  = GND;
    init = VCC;
    ctrclr = VCC;

WHEN ReadKeys=>
    dp.go = GND;
    dp.rs_in = GND;
    dp.data[7..0]=H"00";
    outff[7..0].d = outff[7..0].q;
    storage[8..0].d = storage[8..0].q;
```

```
            keysel = storage0.q;
            if serial.trigger == GND then
                ee = ReadKeys;
            else
                ee = WaitForSer;
            end if;
            idb[7..0] = dp.db[7..0];
            ie  = GND;
            init = VCC;
            ctrclr = VCC;

        WHEN OTHERS=>
            dp.go = GND;
            dp.rs_in = GND;
            dp.data[7..0]=H"00";
            outff[7..0].d = outff[7..0].q;
            storage[8..0].d = H"00";
            keysel = GND;
            ee = WaitForSer;
            idb[7..0] = dp.db[7..0];
            ie  = GND;
            init = VCC;
            ctrclr = VCC;

    END CASE;
    keyff[7..0].(clk,clrn) = (clk, !reset);
    outff[7..0].(clk,clrn) = (clk, !reset);
    storage[8..0].(clk,clrn) = (clk, !reset);
    ee.(clk, reset) = (clk, reset);
END;
```

## B.2.3 display_out.tdf

```
FUNCTION display_busy (clk, reset, go, data7) RETURNS (e, rs_out, rw, busy);

SUBDESIGN display_out
(
    clk, reset, data[7..0],go,rs_in,db7_in  : INPUT;
    db[7..0],e,rs_out,rw,finish             : OUTPUT;
)

VARIABLE
    dd: MACHINE WITH STATES (d1,d1_1,d1_2,d2,d2_1,d3);
    storage : dff;
    busy : display_busy;
    igo:NODE;

BEGIN
    busy.(clk,reset,go, data7) = (clk,reset,igo,db7_in);
    CASE dd IS
        WHEN d1=>
            if go == VCC then
                dd = d1_1;
            else
                dd = d1;
            end if;
            e = GND;
            rw = GND;
            finish = GND;
            rs_out = rs_in;
            db[7..0] = data[7..0];
            storage.d = rs_in;
            igo = GND;
        WHEN d1_1=>
            dd = d1_2;
            e = busy.e;
            rw = busy.rw;
            rs_out = busy.rs_out;
            finish = GND;
            db[7..0] = data[7..0];
            storage.d = storage.q;
            igo = VCC;
        WHEN d1_2=>
            if busy.busy == GND then
                dd = d2;
            else
                dd = d1_2;
            end if;
            e = busy.e;
            rw = busy.rw;
            rs_out = busy.rs_out;
            finish = GND;
            db[7..0] = data[7..0];
            storage.d = storage.q;
            igo = GND;
        WHEN d2=>
            dd = d2_1;
            e = VCC;
```

```
            rw = GND;
            finish = GND;
            rs_out = storage;
            db[7..0] = data[7..0];
            storage.d = storage.q;
        WHEN d2_1=>
            dd = d3;
            e = VCC;
            rw = GND;
            finish = GND;
            rs_out = storage;
            db[7..0] = data[7..0];
            storage.d = storage.q;
        WHEN d3=>
            dd = d1;
            e = GND;
            rw = GND;
            finish = VCC;
            rs_out = storage;
            db[7..0] = data[7..0];
            storage.d = GND;

        WHEN OTHERS=>
            dd = d1;
            e = GND;
            rw = GND;
            finish = GND;
            rs_out = GND;
            db[7..0] = H"00";
            storage.d = GND;
    END CASE;
    storage.(clk,clrn) = (clk,!reset);
    dd.(clk, reset) = (clk, reset);
END;
```

## B.2.4 display_busy.tdf

```
SUBDESIGN display_busy
(
    clk, reset,go,data7 : INPUT;
    e,rs_out,rw,busy    : OUTPUT;
)

VARIABLE
    dd: MACHINE WITH STATES (d1,d2,d2_1,d2_2,d3,d4);

BEGIN
    CASE dd IS
        WHEN d1=>
            if go == VCC then
                dd = d2;
            else
                dd = d1;
            end if;
            e = GND;
            rw = VCC;
            busy = GND;
            rs_out = GND;
        WHEN d2=>
            dd = d2_1;
            e = VCC;
            rw = VCC;
            busy = VCC;
            rs_out = GND;
        WHEN d2_1=>
            dd = d2_2;
            e = VCC;
            rw = VCC;
            busy = VCC;
            rs_out = GND;
        WHEN d2_2=>
            if data7 == VCC then        -- still busy
                dd = d3;
            else
                dd = d1;
            end if;
            e = VCC;
            rw = VCC;
            busy = VCC;
            rs_out = GND;
        WHEN d3=>
            dd = d4;
            e = GND;
            rw = VCC;
            busy = VCC;
            rs_out = GND;
        WHEN d4=>
            dd = d2;
```

```
            e = GND;
            rw = VCC;
            busy = VCC;
            rs_out = GND;

        WHEN OTHERS=>
            dd = d1;
            e = GND;
            rw = GND;
            busy = GND;
            rs_out = GND;
    END CASE;
    dd.(clk, reset) = (clk, reset);
END;
```

## B.2.5 i2c.tdf

```
FUNCTION i2c_start_stop (clk, reset, sclin,sdain) RETURNS (start,stop);
FUNCTION i2c_transmit_byte (clk, reset, sclin, sdain, rw, bytein[7..0], go)
        RETURNS (sclout, sdaout, byteout[7..0], trigger,MACHINE bytestate);


SUBDESIGN i2c
(
    clk,reset,sclin,sdain,datain[7..0]          : INPUT;
    trigger,addr,dataout[7..0],sclout,sdaout    : OUTPUT;
)

VARIABLE
    startstop : i2c_start_stop;
    transmit : i2c_transmit_byte;
    ff : MACHINE WITH STATES (start,start0,start1,start2,start3,start4,start5);
    igo : NODE;
    irw : dff;


BEGIN
    startstop.(clk,reset,sclin,sdain) = (clk,reset,sclin,sdain);
    transmit.(clk,reset,sclin,sdain,rw,bytein[7..0],go) =
            (clk,reset,sclin,sdain,irw,datain[7..0],igo);
    sdaout = transmit.sdaout;
    sclout = transmit.sclout;
    CASE ff IS
        WHEN start=>
            if startstop.start == VCC then
                ff = start0;
            else
                ff =start;
            end if;
            irw.d = VCC;
            igo = GND;
            trigger = GND;
            addr = GND;
        WHEN start0=>
            if sdain == GND and sclin == GND then
                ff = start1;
            elsif startstop.stop == VCC then
                ff = start;
            else
                ff = start0;
            end if;
            irw.d = irw.q;
            igo = GND;
            trigger = GND;
            addr = GND;
        WHEN start1=>
            if transmit.bytestate == BIT_4 and (dataout[7..5] == B"101" or dataout[7..5] == B"110" or dataout[7..5] == B"111") then
                ff = start;
                igo = GND;
            elsif startstop.stop == VCC then
                ff = start;
                igo = GND;
            else
                if transmit.trigger == VCC then
                    ff = start2;
                    igo = GND;
                else
                    ff = start1;
                    igo = VCC;
                end if;
            end if;
            irw.d = irw.q;
            trigger = GND;
            addr = GND;
        WHEN start2=>
            if transmit.trigger == GND then
                ff = start3;
            elsif startstop.stop == VCC then
```

```
            ff = start;
        else
            ff = start2;
        end if;
        irw.d = irw.q;
        igo = GND;
        trigger = GND;
        addr = GND;
    WHEN start3=>
        if startstop.stop == VCC then
            ff = start;
        elsif dataout0 == GND then          -- read
            irw.d = VCC;
        else                                -- write
            irw.d = GND;
        end if;
        ff = start4;
        igo = GND;
        trigger = GND;
        addr = VCC;
    WHEN start4=>
        if transmit.trigger == VCC then
            ff = start5;
        elsif startstop.stop == VCC then
            ff = start;
        else
            ff = start4;
        end if;
        irw.d = irw.q;
        igo = VCC;
        trigger = GND;
        addr = GND;
    WHEN start5=>
        if startstop.stop == VCC then
            ff = start;
        else
            ff = start5;
        end if;
        irw.d = irw.q;
        igo = GND;
        trigger = VCC;
        addr = GND;

    WHEN OTHERS=>
        ff = start;
        irw.d = VCC;
        igo = GND;
        trigger = GND;
        addr = GND;
END CASE;

dataout[7..0] = transmit.byteout[7..0];
ff.(clk,reset) = (clk,reset);
irw.(clk,clrn) = (clk,!reset);
END;
```

## B.2.6 i2c_start_stop.tdf

```
FUNCTION i2c_bit_pos (clk, reset, sclin) RETURNS (MACHINE bitstate);

SUBDESIGN i2c_start_stop
(
    clk, reset, sclin, sdain    : INPUT;
    start, stop                 : OUTPUT;
)

% sdaint ----------\_________________/-------------   %
% bb     B3 B3 B3 B3 B4 B1 B1  B1 B1 B1 B2 B3 B3 B3    %
% sclin  ________/--------\________/--------\________   %
% bb     B1 B1 B1 B2 B3 B3 B4 B1 B1 B2 B3 B3 B4 B1 B1   %
% start  ------------__-----------------------------   %
% stop   -------------------------------__------------   %

VARIABLE
    sda : i2c_bit_pos;
    scl : i2c_bit_pos;

BEGIN

    sda.(clk,reset,sclin) = (clk,reset,sdain);
    scl.(clk,reset,sclin) = (clk,reset,sclin);

    if scl.bitstate == B3 then
        if sda.bitstate == B4 then
            start = VCC;
            stop = GND;
        elsif sda.bitstate == B2 then
            start = GND;
```

```
            stop = VCC;
        else
            start = GND;
            stop = GND;
        end if;
    else
        start = GND;
        stop = GND;
    end if;
END;
```

## B.2.7 i2c_bit_pos.tdf

```
SUBDESIGN i2c_bit_pos
(
    clk, reset, sclin   : INPUT;
    bitstate            : MACHINE OUTPUT;
)

% sclin ________/--------\________  %
% bb    B1 B1 B1 B2 B3 B3 B4 B1 B1  %

VARIABLE
    bb: MACHINE WITH STATES (B1,B2,B3,B4);
BEGIN
    bitstate = bb;
    CASE bb IS
        WHEN B1=>
            if sclin == VCC then
                bb = B2;
            else
                bb = B1;
            end if;
        WHEN B2=>
            if SCLIN == VCC then
                bb = B3;
            else
                bb = B4;
            end if;
        WHEN B3=>
            if sclin == VCC then
                bb = B3;
            else
                bb = B4;
            end if;
        WHEN B4=>
            if sclin == VCC then
                bb = B2;
            else
                bb = B1;
            end if;
    END CASE;
    bb.(clk, reset) = (clk, reset);
END;
```

## B.2.8 i2c_transmit_byte.tdf

```
FUNCTION i2c_bit_pos (clk, reset, sclin) RETURNS (MACHINE bitstate);

SUBDESIGN i2c_transmit_byte
(
    clk, reset, sclin, sdain, rw, bytein[7..0], go  : INPUT;
    sclout, sdaout, byteout[7..0], trigger          : OUTPUT;
    bytestate                                       : MACHINE OUTPUT;
)
VARIABLE
    aa: MACHINE WITH STATES (wait_go,bit_7,bit_6,
            bit_5,bit_4,bit_3,bit_2,bit_1,bit_0,bit_ack);
    cc: i2c_bit_pos;
    ireset : NODE;
    storage[7..0]:dff;

BEGIN
    bytestate = aa;
    cc.(clk,reset,sclin) = (clk,ireset,sclin);

    CASE aa IS
        WHEN wait_go=>
            if go == VCC then
                aa = bit_7;
            else
                aa = wait_go;
            end if;
```

```
    sclout = VCC;
    sdaout = VCC;
    ireset = VCC;
    storage[7..0].d = storage[7..0].q;
    trigger = GND;
WHEN bit_7=>
    if cc.bitstate == B4 then
        aa = bit_6;
    else
        aa = bit_7;
    end if;
    if cc.bitstate == B2 then
        if rw == VCC then
            storage7.d = sdain;
            storage[6..0].d=B"0000000";
        else
            storage[7..0].d=B"00000000";
        end if;
    else
        storage[7..0].d = storage[7..0].q;
    end if;
    if rw == GND then
        sdaout = bytein7;
    else
        sdaout = VCC;
    end if;
    trigger = GND;
    sclout = VCC;
    ireset = GND;
WHEN bit_6=>
    if cc.bitstate == B4 then
        aa = bit_5;
    else
        aa = bit_6;
    end if;
    if cc.bitstate == B2 and rw == VCC then
        storage6.d = sdain;
    else
        storage6.d = storage6.q;
    end if;
    if rw == GND then
        sdaout = bytein6;
    else
        sdaout = VCC;
    end if;
    storage7.d=storage7.q;
    storage[5..0].d=storage[5..0].q;
    trigger = GND;
    sclout = VCC;
    ireset = GND;
WHEN bit_5=>
    if cc.bitstate == B4 then
        aa = bit_4;
    else
        aa = bit_5;
    end if;
    if cc.bitstate == B2 and rw == VCC then
        storage5.d = sdain;
    else
        storage5.d = storage5.q;
    end if;
    if rw == GND then
        sdaout = bytein5;
    else
        sdaout = VCC;
    end if;
    storage[7..6].d=storage[7..6].q;
    storage[4..0].d=storage[4..0].q;
    trigger = GND;
    sclout = VCC;
    ireset = GND;
WHEN bit_4=>
    if cc.bitstate == B4 then
        aa = bit_3;
    else
        aa = bit_4;
    end if;
    if cc.bitstate == B2 and rw == VCC then
        storage4.d = sdain;
    else
        storage4.d = storage4.q;
    end if;
    if rw == GND then
        sdaout = bytein4;
    else
        sdaout = VCC;
    end if;
    storage[7..5].d=storage[7..5].q;
    storage[3..0].d=storage[3..0].q;
    trigger = GND;
    sclout = VCC;
    ireset = GND;
```

```
WHEN bit_3=>
    if cc.bitstate == B4 then
        aa = bit_2;
    else
        aa = bit_3;
    end if;
    if cc.bitstate == B2 and rw == VCC then
        storage3.d = sdain;
    else
        storage3.d = storage3.q;
    end if;
    if rw == GND then
        sdaout = bytein3;
    else
        sdaout = VCC;
    end if;
    storage[7..4].d=storage[7..4].q;
    storage[2..0].d=storage[2..0].q;
    trigger = GND;
    sclout = VCC;
    ireset = GND;
WHEN bit_2=>
    if cc.bitstate == B4 then
        aa = bit_1;
    else
        aa = bit_2;
    end if;
    if cc.bitstate == B2 and rw == VCC then
        storage2.d = sdain;
    else
        storage2.d = storage2.q;
    end if;
    if rw == GND then
        sdaout = bytein2;
    else
        sdaout = VCC;
    end if;
    storage[7..3].d=storage[7..3].q;
    storage[1..0].d=storage[1..0].q;
    trigger = GND;
    sclout = VCC;
    ireset = GND;
WHEN bit_1=>
    if cc.bitstate == B4 then
        aa = bit_0;
    else
        aa = bit_1;
    end if;
    if cc.bitstate == B2 and rw == VCC then
        storage1.d = sdain;
    else
        storage1.d = storage1.q;
    end if;
    if rw == GND then
        sdaout = bytein1;
    else
        sdaout = VCC;
    end if;
    storage[7..2].d=storage[7..2].q;
    storage0.d=storage0.q;
    trigger = GND;
    sclout = VCC;
    ireset = GND;
WHEN bit_0=>
    if cc.bitstate == B4 and go == VCC then
        aa = bit_ack;
    elsif cc.bitstate == B4 and go == GND then
        aa = wait_go;
    else
        aa = bit_0;
    end if;
    if cc.bitstate == B2 and rw == VCC then
        storage0.d = sdain;
    else
        storage0.d = storage0.q;
    end if;
    if rw == GND then
        sdaout = bytein0;
    else
        sdaout = VCC;
    end if;
    storage[7..1].d=storage[7..1].q;
    trigger = GND;
    sclout = VCC;
    ireset = GND;
WHEN bit_ack=>
    if cc.bitstate == B4 then
        aa = wait_go;
        sdaout = VCC;
    else
        aa = bit_ack;
        if rw == VCC then
```

```
                sdaout = GND;
            else
                sdaout = VCC;
            end if;
        end if;
        storage[7..0].d=storage[7..0].q;
        trigger = VCC;
        sclout = VCC;
        ireset = GND;
    WHEN OTHERS=>
        aa = wait_go;
        sclout = VCC;
        sdaout = VCC;
END case;
byteout[7..0]=storage[7..0];
aa.(clk, reset) = (clk, reset);
storage[7..0].(clk,clrn) = (clk,!reset);
END;
```

# Anhang C

# Bilddokumentation

## C.1 Materialbearbeitung

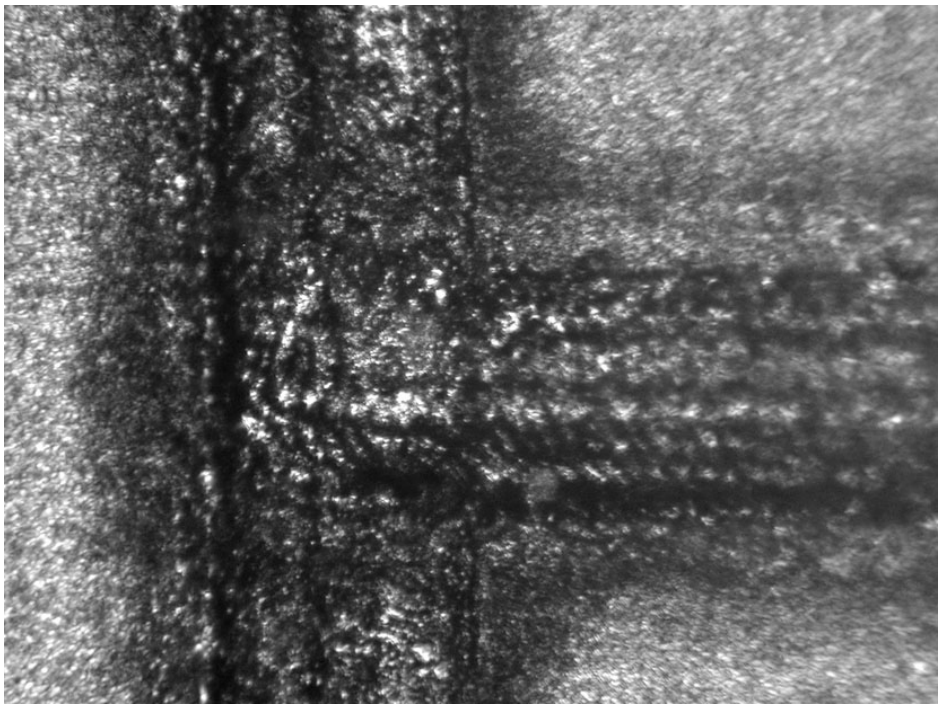


Abbildung C.1: Linie im CW Modus

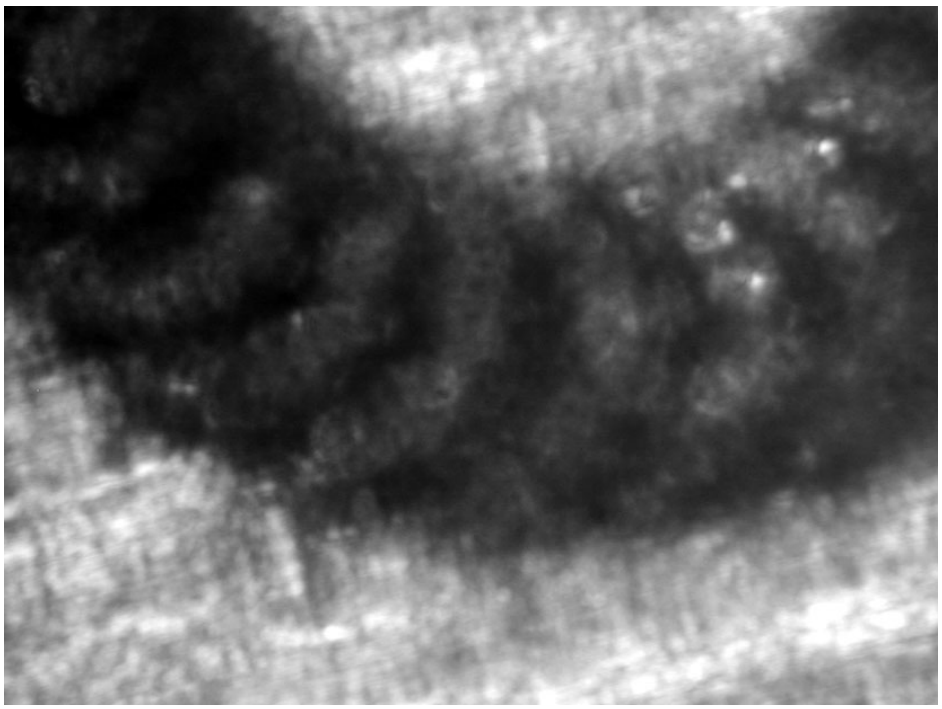


Abbildung C.2: Linie mit aktivierten Q-Switch

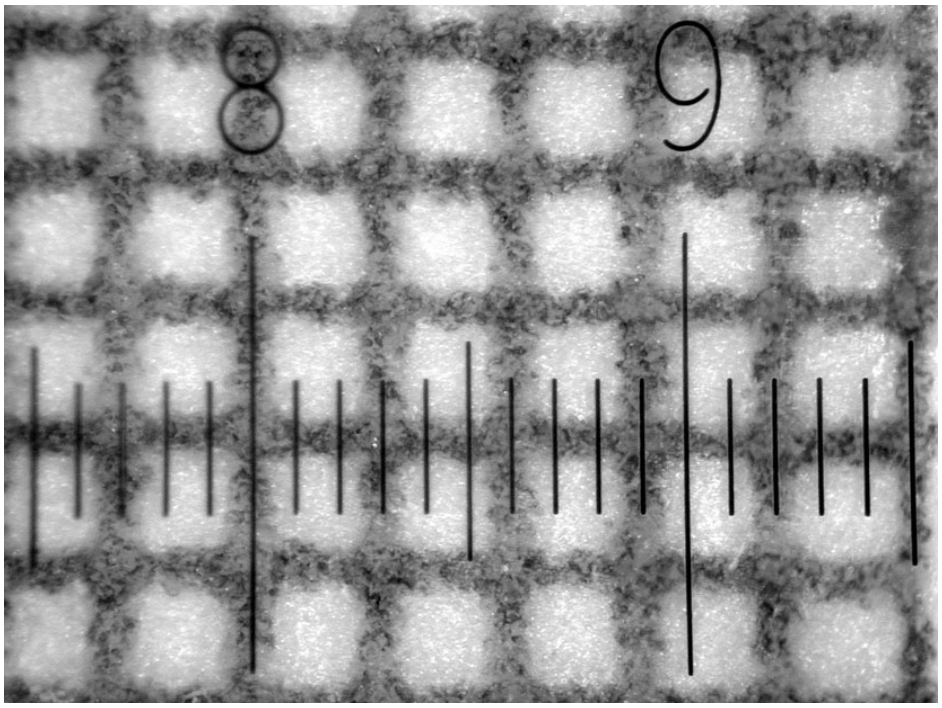



Abbildung C.3: Ein mittels Laserstrahl markiertes Keramikplättchen mit Millimetermaßstab

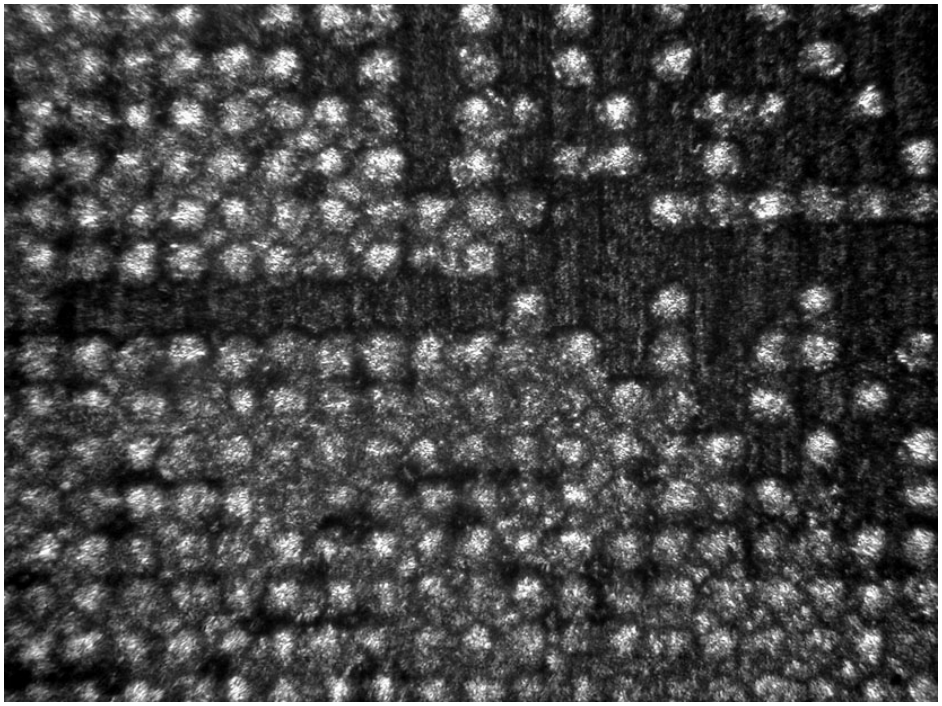


Abbildung C.4: Ein im Matrixverfahren abgetragene Fläche auf eloxiertem Aluminium

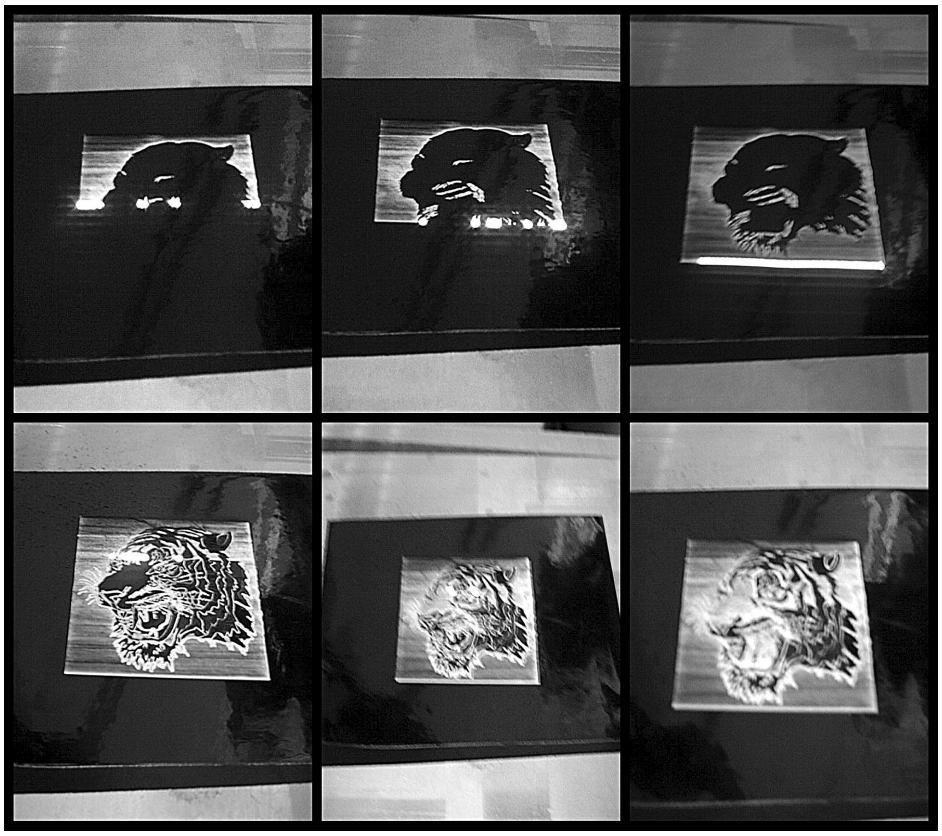


Abbildung C.5: Beschriftungsvorgang einer Folie aus zweilagigem Kunststoff

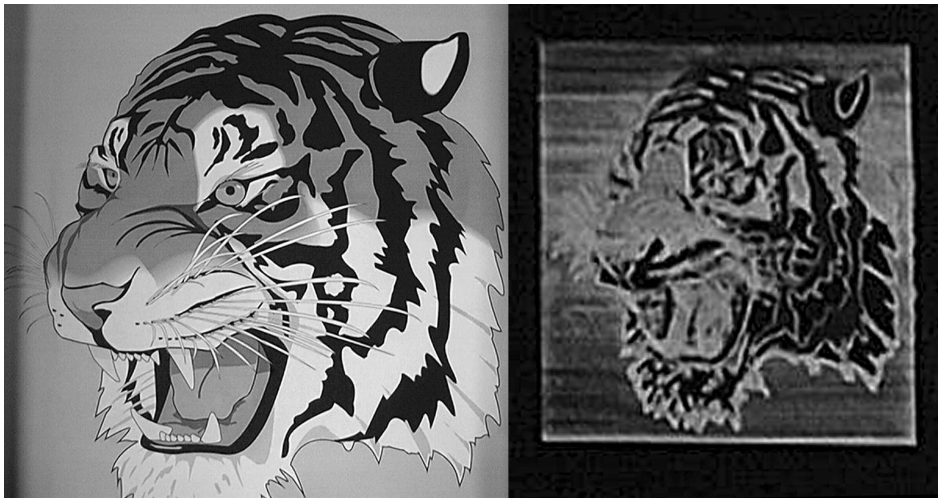


Abbildung C.6: Ausdruck auf Laserdrucker und Bild auf Folie

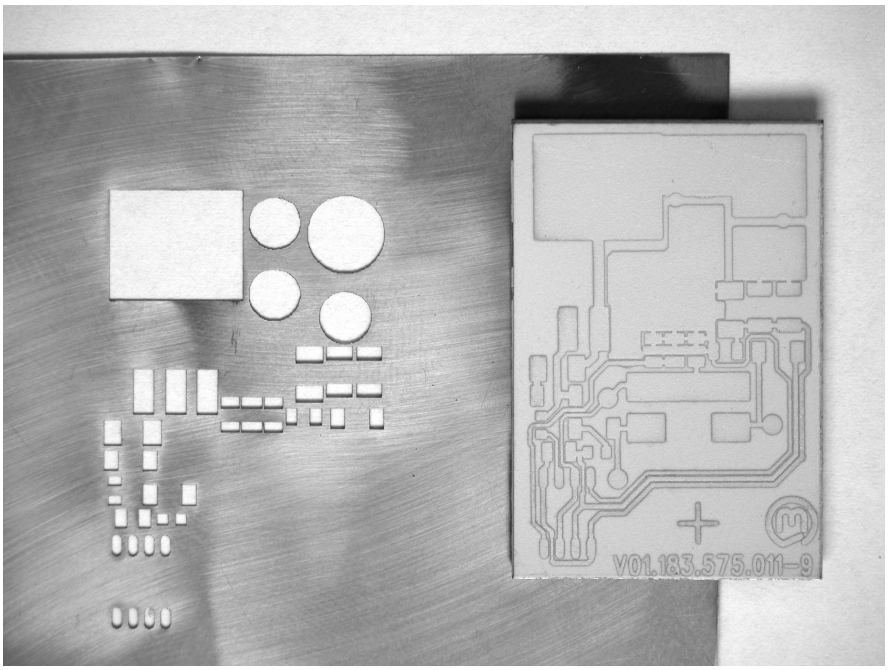



Abbildung C.7: Druckschablone für die Aufbringung von Lotpaste auf den Kermikträger(rechts)

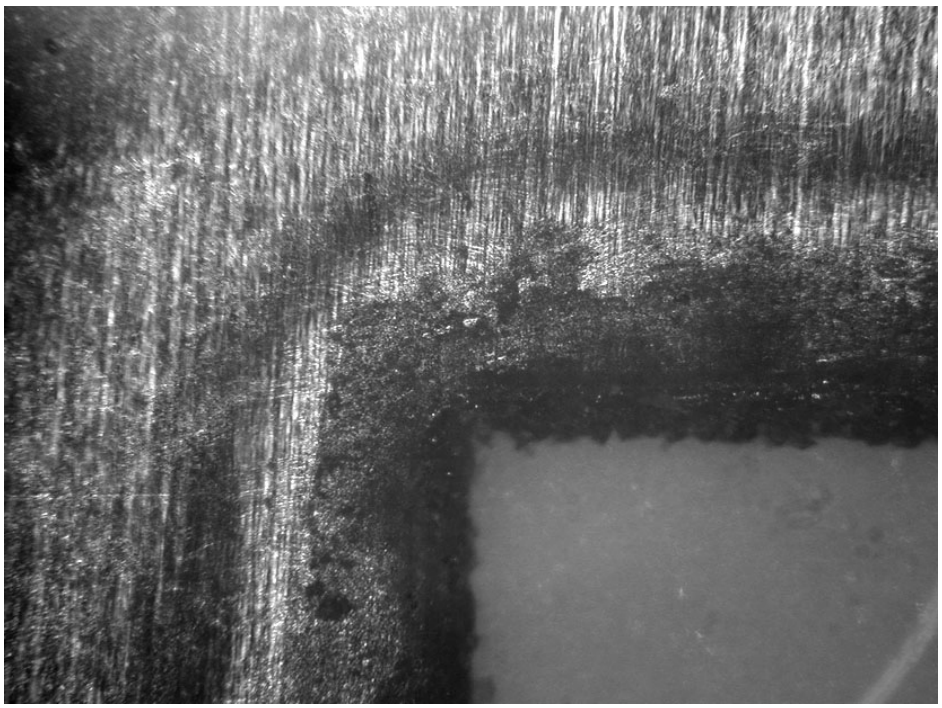


Abbildung C.8: Verfärbungen beim Laserschneiden ohne Schutzgas

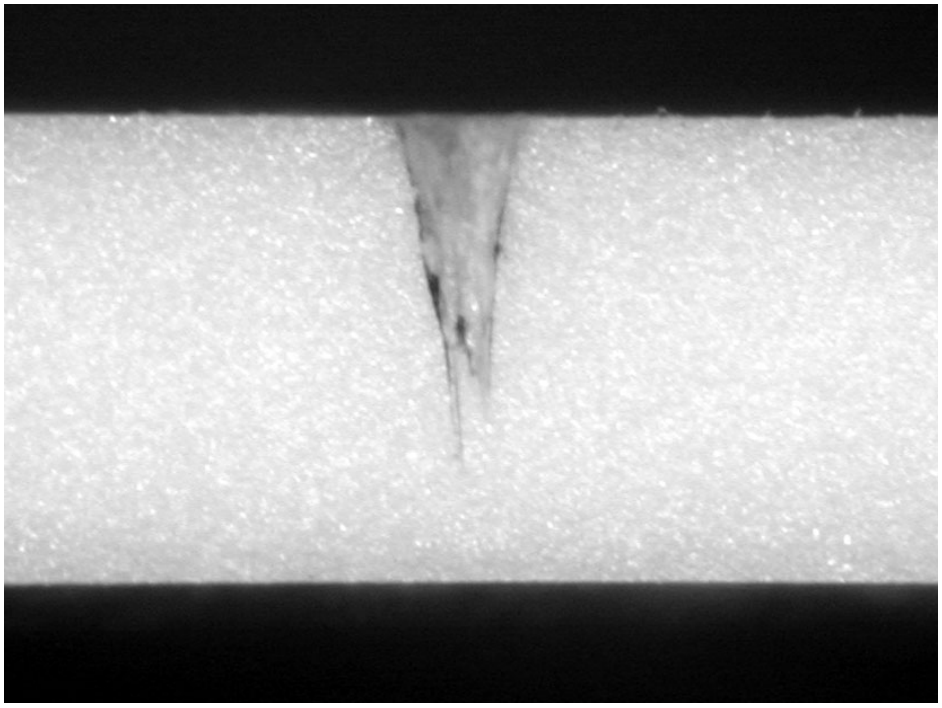


Abbildung C.9: Laserschlagbohren in Keramik

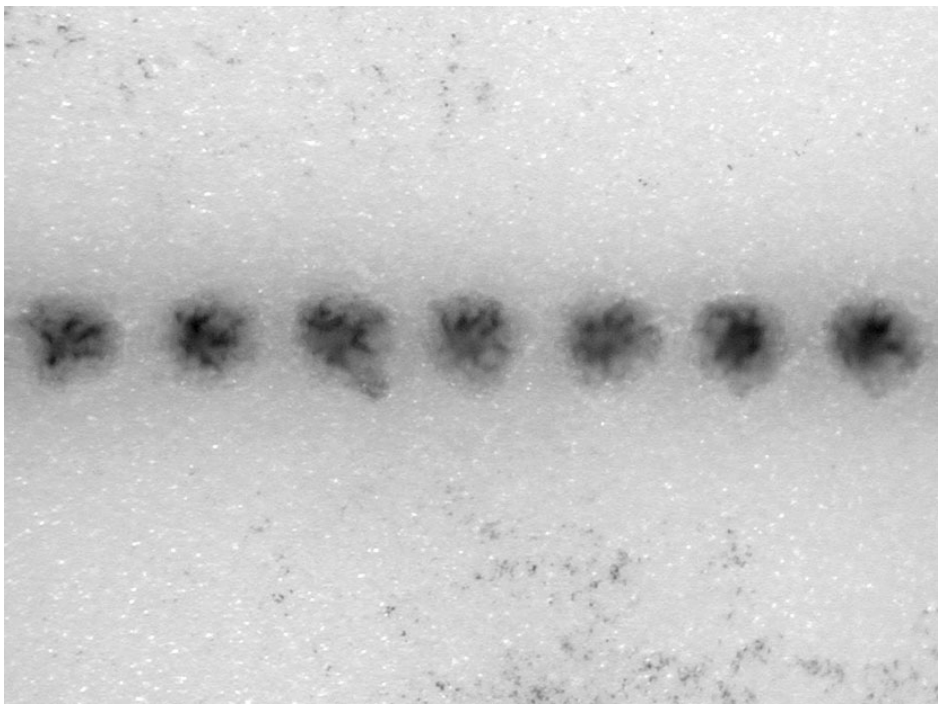


Abbildung C.10: Perforierte Keramik, Löcher in Feinstbohrtechnik hergestellt

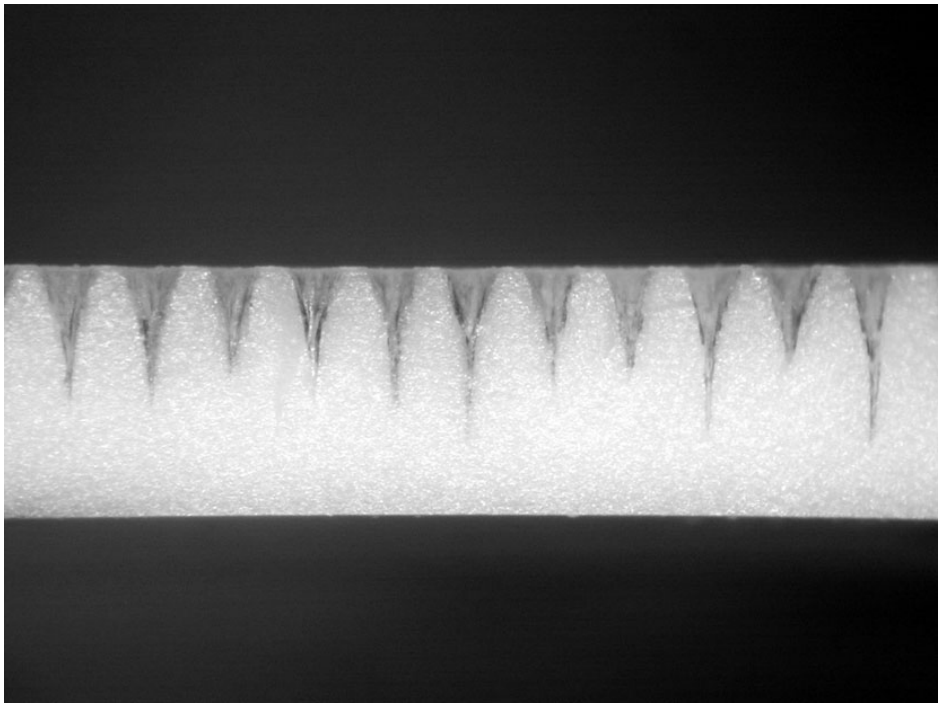


Abbildung C.11: Bruchfläche der in Feinstbohrtechnik perforierten Keramik

## C.2 Hardware

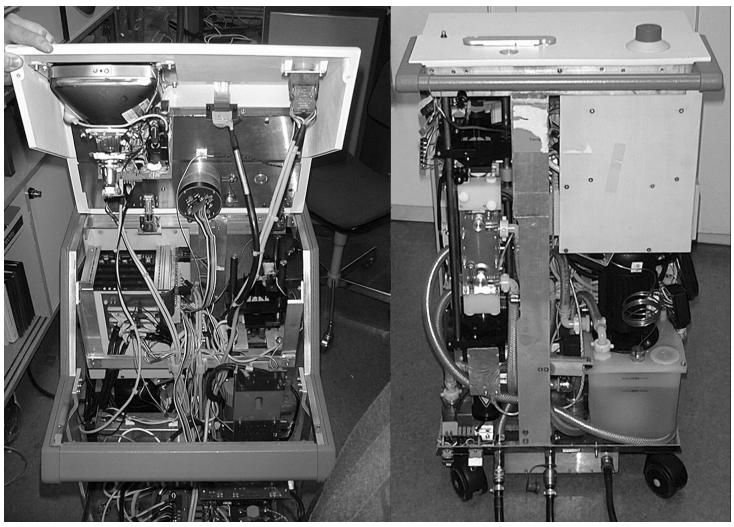


Abbildung C.12: Originalzustand der Laserquelle

Abbildung C.13: Zustand nach dem Umbau

Abbildung C.14: Zustand nach dem Umbau, mit geöffnetem Gehäuse

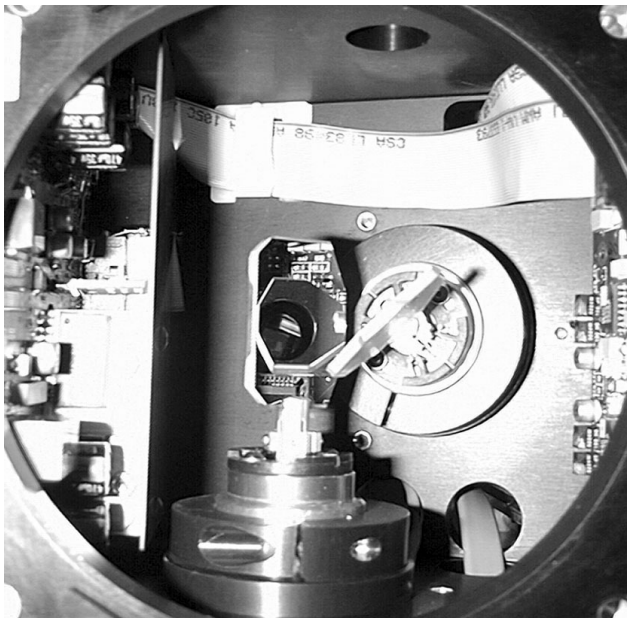


Abbildung C.15: Galvo-Kopf, Sicht durch das Strahlaustrittsfenster

# Anhang D

# GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## D.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and

2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## D.2 GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is

covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to

decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## D.3 NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE

QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

    END OF TERMS AND CONDITIONS

## D.4 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
```

```
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.