

Projektarbeit HS 2011

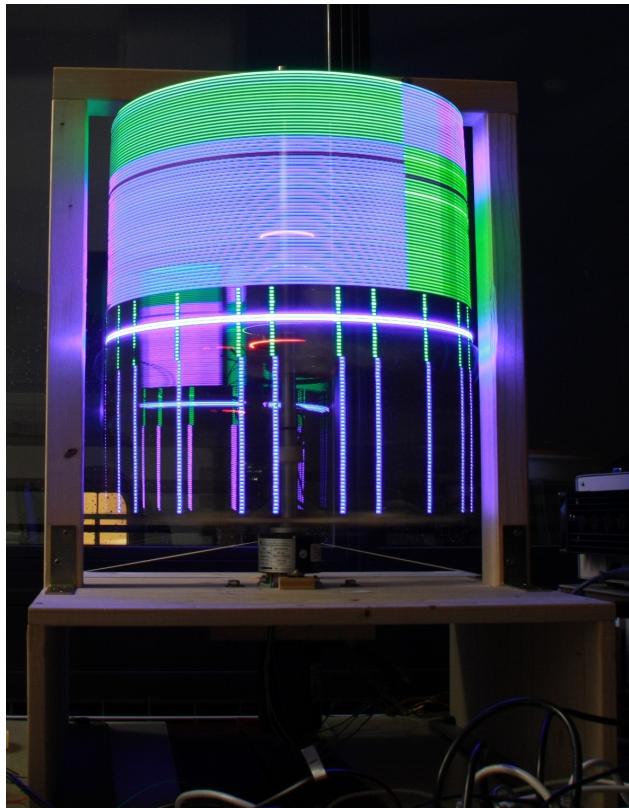
Andreas Butti, David Halter

30.12.2011



Rotierendes LED Display

Betreuer Hanspeter Hochreutener
Studiengang Systeminformatik
Klasse SI09a



Inhaltsverzeichnis

Abstract.....	5
Zusammenfassung.....	6
Vorwort.....	7
Einleitung.....	9
Darstellung.....	10
Vorgabe.....	II
Übersicht.....	I3
Werkzeuge und Hilfsmittel.....	I4
Programmiersprachen.....	I4
Entwicklungsumgebung.....	I4
Für Java und Python und das Syncboard (Andreas Butti).....	I4
Für C und Python (David Halter).....	I4
Testing.....	I5
Layout Software.....	I6
Löten.....	I7
Vorgehen.....	I8
Machbarkeitsanalyse.....	I8
LED Elektronik.....	I9
LED Auswahl.....	I9
FET Auswahl.....	20
Schema.....	I2I
Platinen herstellen.....	22
Platinen bestücken.....	23
Mechanik.....	25
Aluminium-Gerüst.....	25

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Motor.....	26
Schleifringe.....	26
Synchronisation.....	27
1. Messung der Zeit.....	27
2. Synchronisation der LED Ausgabe.....	28
3. Datenrate.....	28
LED-Controller.....	29
Controller Auswahl.....	29
USB.....	31
Bulk USB Mode.....	31
Isochronous USB Mode.....	31
Hauptcontroller.....	32
Auswahl.....	32
Datenübertragung.....	33
PC Applikation.....	34
Spaltensynchronisation.....	35
Resultate.....	38
Diskussion & Ausblick.....	39
Glossar.....	40
Anhang.....	41

Abstract

The aim of this project was to create a rotating LED device which would be able to display a picture with two wings. The task included three main parts, an electronical, a mechanical and a software part. For the mechanical part, we had to build a carrier for the LED boards, choose an engine and slip rings. The electronical part consisted of a PCB layout, mounting the electrical components and creating a synchronisation board.

Software was widely used for file processing and streaming via USB and WLAN. It also included programming ARM and AVR microcontrollers for the synchronisation boards and the LED boards. The last piece of software was Linux programming which was used to send the files to the boards and control everything.

This project will provide a substantial part of the end product. The goal of the end product is to be presented at “Nacht der Technik”, which is organized by the ZHAW. However, due to limited time it was not possible to implement video streaming to the display, though it was possible to display various images on the Board.

Zusammenfassung

Das Ziel dieser Projektarbeit war es Bilder auf einem rotierenden LED Display anzuzeigen. Die Arbeit lässt sich in drei Hauptteile aufteilen, einen mechanischen, einen elektronischen und einen Software Teil. Im mechanischen Teil ging es darum ein Gerüst für unsere LED Boards zu bauen, sowie einen Motor und Schleifringe auszuwählen. Der elektronische Teil bestand aus Platinen Layouting, dem bestücken eben dieser Board mit den elektrischen Komponenten, sowie dem kreieren eines Synchronisationsboards. Software wurde in verschiedenen Bereichen gebraucht, vor allem aber bei der Übertragung über WLAN und USB. Des weiteren wurde Software verwendet um ARM und AVR Mikrocontroller für die LED Boards, sowie das Synchronisationsboard zu programmieren. Der letzte grosse Softwareteil war das Programmieren eines Linux-boards, dass für die Dateiübertragung zuständig ist und uns zur Steuerung des Displays nützlich war.

Diese Projektarbeit wird einen substantiellen Anteil an das Endprodukt liefern. Das Ziel des Endprodukts ist es an der "Nacht der Technik" ausgestellt zu werden, welche von der ZHAW organisiert wird. Da die Arbeit aber zeitlich eingeschränkt ist, war es uns aber leider nicht möglich Videos anzuzeigen. Dennoch ist es möglich nach Belieben einzelne Bilder auf unserem Display anzuzeigen.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Vorwort

Wer im Mikrocontroller Bereich aktiv ist, stösst immer mal wieder auf kleine, rotierende LED Displays. Meistens nur mit 8 LEDs und einem AVR oder PIC. Andreas Butti wollte schon immer so etwas bauen, hatte jedoch weder die Zeit dazu, noch das Wissen für den mechanischen Aufbau. David Halter schlug dann spontan bei einem Mittagessen vor, dass wir ein solches Display als PA machen könnten. So begann alles...

Wir haben uns dann einen Dozenten gesucht und sind mit Herrn Hanspeter Hochreutener fünfzig geworden, der die Arbeit ausgeschrieben und auch geleitet hat.

Viel Hilfe haben wir auch von den wissenschaftlichen Mitarbeitern Daniel Früh und Michael Jäger erhalten. Daniel Früh hat den kompletten mechanischen Aufbau inklusive Beschaffung aller Teile wie Motor, Motorsteuerung und Schleifringe durchgeführt. Michael Jäger hat uns geholfen Probleme wie einen falsch angeschlossenen Boot Pin zu suchen, und stand uns bei technischen Fragen zu STM32 zur Seite. Ohne Sie wäre diese Projektarbeit nicht realisierbar gewesen.

Deshalb möchten wir hier am Anfang dieser Arbeit auch herzlich allen beteiligten Personen danken. Ohne Hilfe der wissenschaftlichen Mitarbeiter und Herrn Hochreutener wäre dieses Projekt sicher nicht möglich gewesen.

[PA] **Rotierendes LED Display**

Andreas Butti, David Halter

Erklärung betreffend des selbständigen Verfassens einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbstständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe).

Der / die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum

Unterschriften

.....
Andreas Butti

.....
David Halter

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Einleitung

Rotierende LED-Displays sind in der Anwendung wohl eher ein Spielzeug, als konkurrenzfähige Anzeigen. Die Displays haben durch die Drehung einen massiv höheren Stromverbrauch und damit auch eine höhere Abnutzung der Komponenten (Schleifringe, Motor, etc). Die Alternative ist es den ganzen Bildschirm mit LEDs auszustatten. Damit hat man einen helleren Bildschirm, und kann normale Bildschirmansteuerungen verwenden.

Das einzige Beispiel für kommerzielle LED-Displays, dass wir kennen, steht in einem Metro Bahnhof von Seoul, Korea.

Hier ist zu sehen, dass das Display aus sehr vielen Flügeln besteht (ca. 12), während unser LED-Display nur aus zwei solchen besteht. Mit mehr Flügeln ist eine kleinere Drehzahl möglich, und es kann trotzdem eine gute Qualität des Bildes gehalten werden. Für dieses Display in Seoul wurden auch SMD RGB LEDs verwendet. Dies ist sichtbar, wenn man die unten stehenden Bilder stark vergrößert.



Abbildung 1: Lange Belichtungszeit

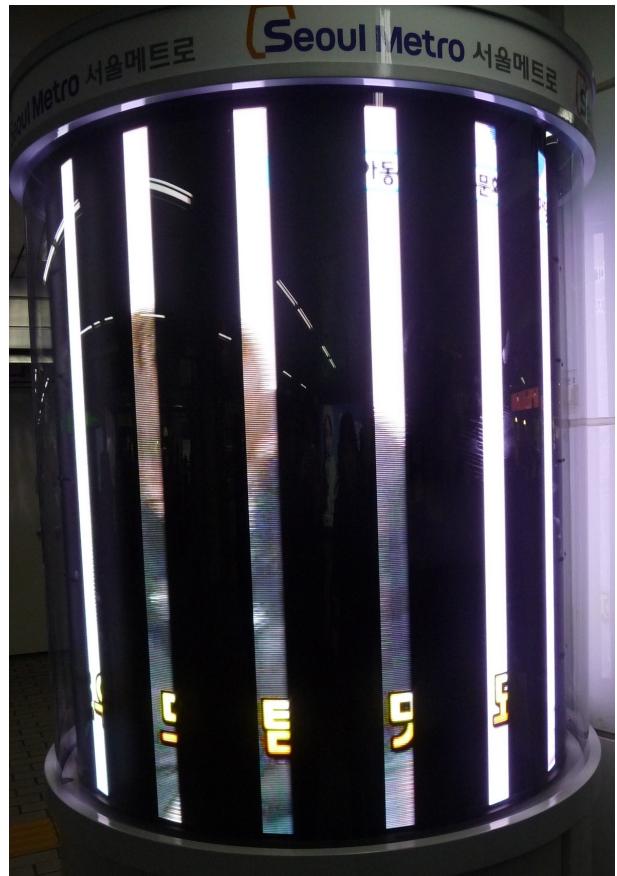


Abbildung 2: Kurze Belichtungszeit

Die Bilder wurden uns freundlicherweise von Adrian Bieri zur Verfügung gestellt, die er bei einer Reise nach Südkorea geschossen hat.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Trotzdem haben LED Displays ihren Reiz. Mit wenigen LEDs kann ein Display mit mehreren tausenden Pixeln realisiert werden. Weiterhin ist es interessant zu sehen, wie so ein Display startet und mit der Beschleunigung ein immer regelmässigeres Bild anzeigt.

Darstellung

Ein handelsüblicher TFT Bildschirm hat quadratische Pixel, die horizontal auf 3 Subpixel aufgeteilt sind, welche rot, grün und blau eingefärbt sind. Werden diese Pixel zeitlich unterschiedlich ein- und ausgeschaltet, können fast beliebige Farben dargestellt werden. Die Bilder werden normalerweise in Zeilen ausgegeben, also zuerst in X-Richtung und dann in Y-Richtung.



Abbildung 3: TFT Nahaufnahme

Rechts ist noch eine Nahaufnahme eines LCD zu sehen, bei dem zu erkennen ist, dass die einzelnen Farbpixel in X-Richtung zu einander verschoben sind. Links im Bild, wird die Farbe weiss abgebildet, indem alle Farben angeschaltet werden.

Unser Bild muss etwas anders aufgebaut sein, obwohl wir uns bei den Farben an das gleiche Konzept halten, müssen wir die Spalten und Zeilen umgekehrt darstellen, also zuerst die Spalten und dann die Zeilen, da wir nur eine Spalte abbilden (Unsere LED Boards). Diese Spalte wird dann mechanisch verschoben. Wenn die Drehung schnell ausgeführt wird, nimmt der Mensch ein Bild wahr.

Das Konzept eines rotierenden LED Displays ist also prinzipiell das gleiche, wie das eines TFT oder Röhren Bildschirms.

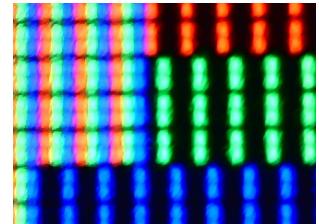


Abbildung 4: LCD-Nahaufnahme

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Vorgabe

Bildschirm mit rotierenden LED-Balken

BetreuerInnen:	Hanspeter Hochreutener, hhrt
Fachgebiete:	Digitale Signalverarbeitung (DSV)
Studiengang:	ET / ME / SI
Zuordnung:	Zentrum für Signalverarbeitung und Nachrichtentechnik (ZSN)
Gruppengrösse:	2

Kurzbeschreibung:

Vertikale LED-Balken rotieren um eine horizontale Achse. Wenn die LEDs im richtigen Moment ein- / ausgeschaltet werden, entsteht ein Bild. Statt vieler Worte zwei Links:

Funktionsweise gut sichtbar: <http://akikorhonon.org/projects.php?action=view&id=216>

Anspruchsvolle Version: http://hackedgadgets.com/wp-content/rotating_display.jpg

Ziele

- Demonstrations-Objekt für die ZSN-Vitrine (45x45x30cm) oder die Nacht der Technik" mit rund 100 LED vertikal.
- Webcam oder Videostream über Notebook anschliessbar für bewegte Bilder
- Standbild bei Standalone-Betrieb
- Farb-Wiedergabe mit 2 RGB-LED-Balken oder mit 4 LED-Balken (rot+grün+blau+gelb = weiss)

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Teil-Projekte

- Mechanischer Aufbau möglichst mit fertigen Komponenten (Motor, Ansteuerung, Positionserfassung), rotierende LED-Arme ev. direkt als Print
- HW für Motor-Ansteuerung
- HW für LED-Ansteuerung mit ein FET (angesteuert vom DO des Microcontrollers) und ein Widerstand (Strombegrenzung) pro LED
- Microcontroller evaluieren bez. Anzahl DO und Geschwindigkeit. Die µC müssen vermutlich direkt auf den LED-Prints platziert werden, da viele DO zu den LEDs führen (Print herstellen). Die hohe Anzahl benötigter DOs muss auf mehrere uC aufgeteilt werden: z.B. 1 pro Farbe + 1 für die Kommunikation, Decodierung und Datenaufsplittung.
- SW für die Farbe und Helligkeit Bei einem Bild mit 1000 Pixeln pro Umdrehung, 50 Umdrehungen/Sekunde und 4 Helligkeitsstufen für 4 LED-Farben (= total 256 Farben) ergibt sich eine Taktrate von $1000*50/s*4 = 200\text{kHz}$ Bei 128 LED in der Höhe müssen also pro Farbe 16 Ports a 8 Bit mit 200kHz aktualisiert werden.
- SW für Videostream-Decodierung auf separatem µC oder im Notebook.
- HW und SW für Video-Übertragung auf rotierendes System möglichst mit Standard-Komponenten, z.B. Bluetooth.
- Energie-Übertragung auf rotierendes System z.B. mittels Schleifringen.

Voraussetzungen

Voraussetzung: Micro-Controller-Programmierung

Sinnvoll: Bildverarbeitung und Digitale Signalverarbeitung

Vorteilhaft: Erfahrung in Print-Layout und HW-Aufbau

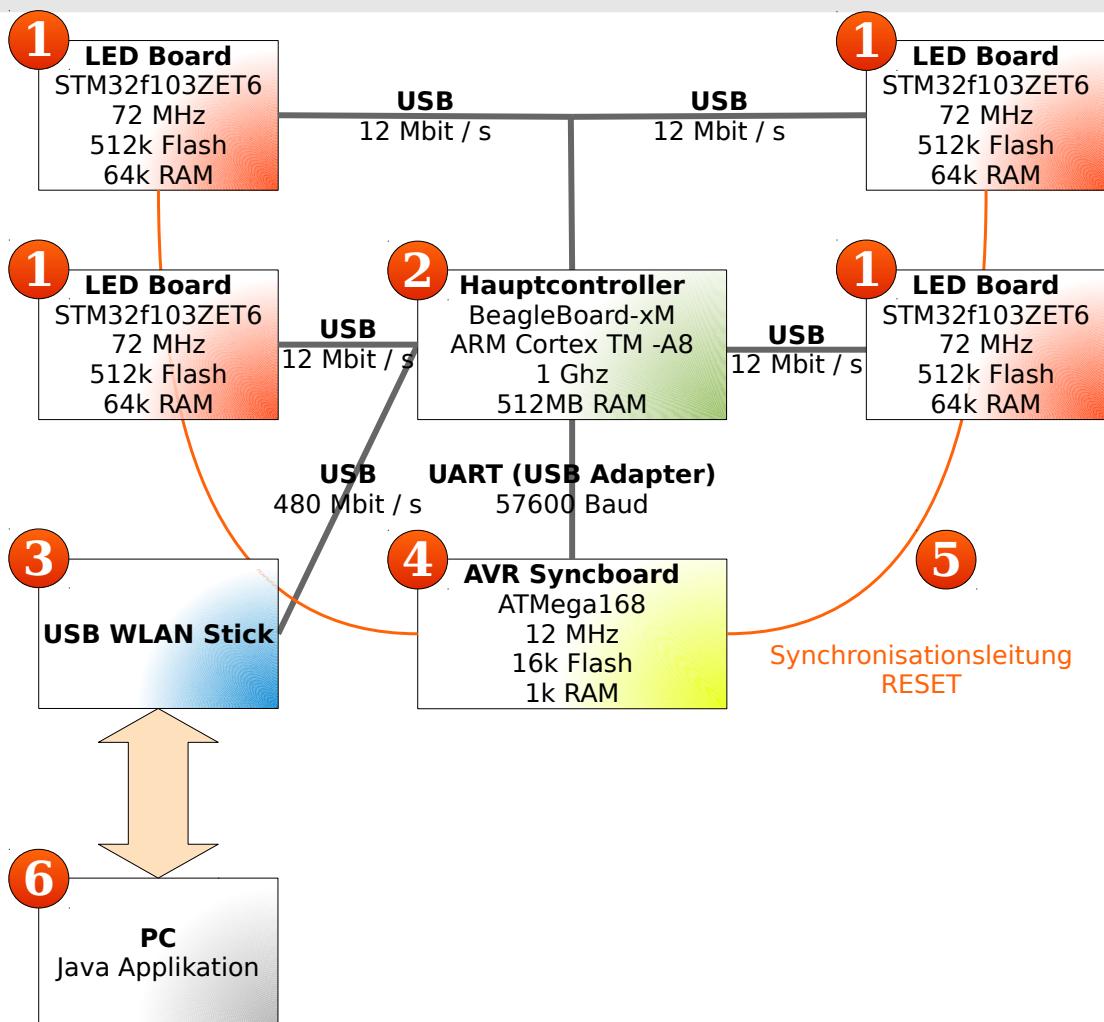
Die Arbeit ist reserviert für

Andreas Butti (buttiand)

David Halter (haltedav)

Dienstag 19. Juli 2011 15:58

Übersicht



- 1 **LED Boards**, Seite 19 [LED Elektronik], Seite 22 [Platinen herstellen]
- 2 **Beagle Board**, Seite 32 [Hauptcontroller]
- 3 **WLAN Verbindung / Protokoll**, Seite 33 [Datenübertragung]
- 4 **AVR Synchronisationsboard**, Seite 35 [Spaltensynchronisation]
- 5 **Synchronisation, Steuerleitungen**, Seite 35 [Synchronisationsboard Protokol]
- 6 **PC Applikation**, Seite 34 [PC Applikation]

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Werkzeuge und Hilfsmittel

Programmiersprachen

Wenn man ein Projekt realisiert, entscheidet man sich normalerweise anfangs für eine Programmiersprache. Dies war bei uns wegen mehreren Gründen nicht so einfach möglich. Einerseits haben wir viele verschiedenen Komponenten im Einsatz und andererseits haben wir nicht beide die gleichen Vorlieben bezüglich Programmiersprachen und sind deshalb auch in unterschiedlichen Programmiersprachen stärker.

Für die Mikrocontroller war die Programmiersprache eigentlich klar, es kamen nur C und Assembler in Frage, wobei Assembler für uns keine Option war, da USB Transfers damit enorm kompliziert werden.

Für die Client Applikation haben wir uns für Java entschieden, da Java Systemunabhängig läuft, und eine GUI mit sich bringt. Allerdings ist es nicht ganz so angenehm eine Java VM auf einem ARM Board zum laufen zu kriegen. Natürlich kann man auf Android zurückgreifen und die DalvikVM nehmen, die für ARM Prozessoren optimiert ist. Jedoch ist das Einrichten nicht unbedingt einfach. Zudem bietet Java keine API für USB. Daher haben wir uns für Python auf dem Beagle Board entschieden. Für Python gibt es zahlreiche gute Bibliotheken für USB und andere Schnittstellen. Python hat im Vergleich mit C den Vorteil, dass man sich nicht um systemnahe Dinge kümmern muss. Weiterhin ist es mit Python sehr einfach möglich performancekritische Datenoperationen in C auszulagern.

Somit standen unsere Wahlen fest:

ARM Mikroprozessor	Beagle Board	Client
C mit inline ASM	Python mit C	Java

Entwicklungsumgebung

Für Java und Python und das Syncboard (Andreas Butti)

Für Java kam Eclipse zum Einsatz, ebenfalls für die Entwicklung der AVR Firmware und Teile der STM32 Firmware. Für Python kam Gedit zum Einsatz.

Für C und Python (David Halter)

Für C und Python wurde ein stark konfigurierter VIM in Kombination mit vielen Unix Tools verwendet. Unix ist gerade für eine solche Arbeit sehr stark. So war es für uns in kurzer Zeit möglich zwei zusätzliche Leitungen vom AVR Board zu den LED-Boards zu ziehen, um Reset und Boot0 setzen zu können. Dies führte dazu, dass wir mit einem Befehl von unseren eigenen PCs aus die Boards neu flashen können. Dazu wird ein Befehl über SSH / WLAN auf das Beagle

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Board gesendet. Dieses löst ein Befehl über die serielle Schnittstelle auf das AVR Board aus, welches wiederum die Output Ports Boot0 und Reset schaltet. Dass klingt jetzt etwas kompliziert, hat uns die Arbeit aber extrem erleichtert.

Gerade hier hätte ein klassisches Embedded Studio (Microvision, Keil) wahrscheinlich versagt, da solche Unix Spielereien einfach nicht möglich sind. Dennoch: Der Nachteil der Shell ist, dass man sich damit wirklich gut auskennen muss. VIM, make, ssh, und so weiter sind mächtige Tools, die man erst einmal lernen muss.

Ein Grossteil der Software wurde direkt auf dem Beagle Board entwickelt und konnten somit direkt da getestet werden. Dies konnte nur gemacht werden, weil VIM ein Editor ist, der sich ohne Maus über die Konsole bedienen lässt.

Testing

In der Theorie hätten für viele unserer Programmteile Testfälle wie Unitests erfasst werden können. Damit hätten wir paralleler entwickeln können und wäre nicht so stark auf Fortschritte in anderen Bereichen angewiesen gewesen. In unserem Fall aber, wo auf mehreren unabhängigen Prozessoren gearbeitet wird und das Verhalten zudem von äusseren Einflüssen abhängig ist, ist das erstellen von Unitests nicht trivial, beziehungsweise in unserem Fall nicht praktikabel.

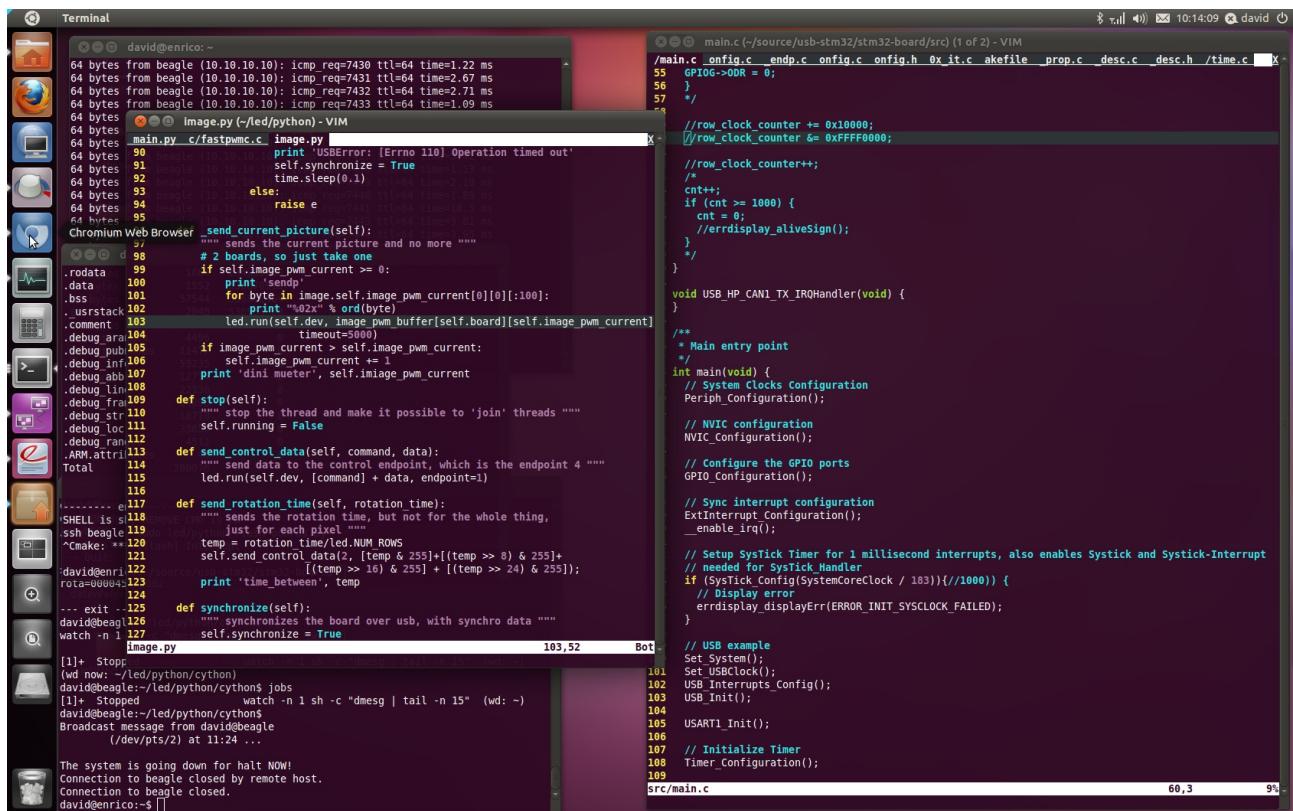


Abbildung 5: Entwicklung in der Shell: mit VIM, Make, SSH

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Layout Software

Als Layoutsoftware kam KiCAD zum Einsatz, da es frei ist (GPL) und auch unter Linux läuft. Eagle würde auch unter Linux laufen, ist jedoch nur bis 80×100 mm kostenlos, für grössere Layouts ist es kostenpflichtig. Zudem ist KiCAD einfacher zu bedienen.

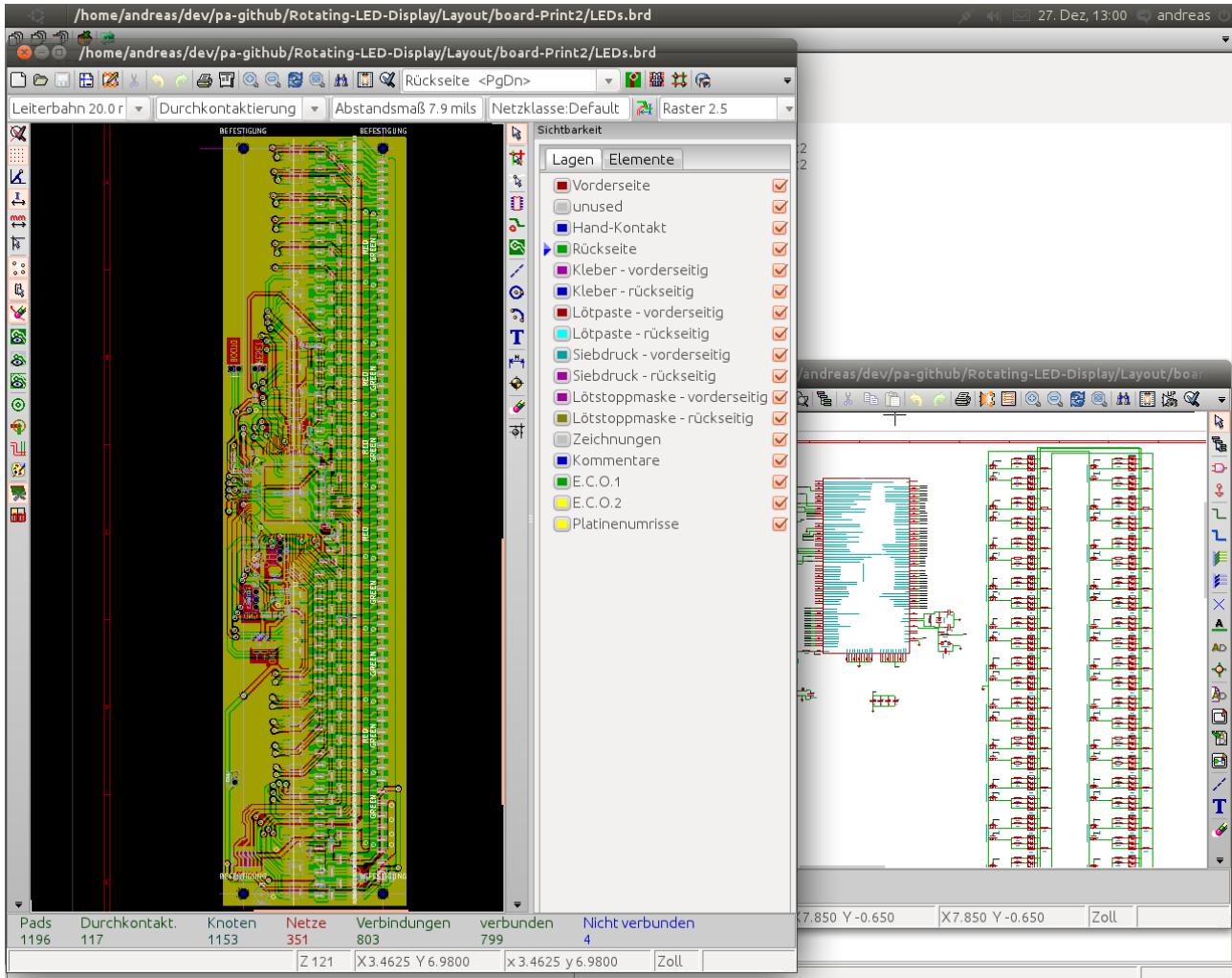


Abbildung 6: KiCAD unter Ubuntu

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Löten

In Bild 8 zu sehen: Der Lötarbeitsplatz der ZHAW im Zimmer TB 611. Hier haben wir alle unsere Boards gelötet. Einzig der Prozessor, wurde von Daniel Früh bzw. Michael Jäger gelötet, da dieser eher schwierig zum löten ist. Einen der Prozessoren haben wir versucht selbst aufzulöten, was nicht gut geklappt hat. Michael Jäger musste danach Lötbrücken entfernen die wir nicht mehr weg gekriegt hatten.

Das Hauptproblem waren die kleinen Anschlüsse, beziehungsweise der grosse Lötkolben. Auf dem Bild ist zu sehen wie klein die Pins verglichen mit dem Lötkolben sind. Man muss extrem präzise agieren um keine Lötbrücken zu erstellen. Wir hatten genug dünnes Lötzinn, dass es tatsächlich möglich ist mit diesem Lötkolben diese feinen Pins zu löten, wie Michael Jäger auch bewiesen hat.

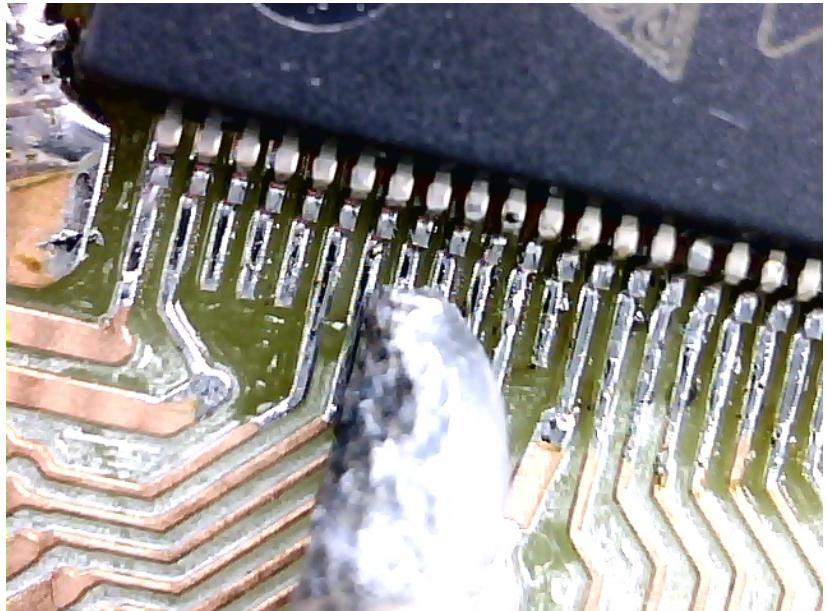


Abbildung 7: Lötkolben (kalt) an ARM Pins



Abbildung 8: Lötarbeitsplatz

Vorgehen

Der erste Schritt in unserem Projekt war die Definition, wie viele LEDs wir ansteuern wollten. Dies wurde mittels einer Machbarkeitsanalyse bestimmt. Des weiteren haben wir dann die elektronischen Bauteile evaluiert und ausgewählt. Der Controller und die LEDs wurden bestimmt, sowie auch die FETs [Seite 19 LED Elektronik, Seite 20 FET Auswahl]. Daraus haben wir dann einen Schaltplan entwickelt, womit auch das Layout gemacht werden konnte [Seite 21 Schema, Seite 22 Platinen herstellen]. Parallel dazu wurden erste Tests mit dem isochronen USB-Modus gemacht [Seite 31 Isochronous USB Mode]. Erst mit dem kompletten Layout war es möglich mit der Arbeit des mechanischen Teils zu beginnen [Seite 25 Mechanik]. Hier lag auch unser grosses Problem: Dieser Teil der Arbeit nahm uns einen grossen Anteil der zur Verfügung stehenden Zeit ab. Erst etwa zur Halbzeit konnten wir mit der Mechanik beginnen. Parallel dazu folgte nun das Löten der elektronischen Bauteile [Seite 17 Löten]. Weiterhin entschieden wir uns 2 Flügel zu machen, mit je zwei Boards.

Danach folgte die Programmierung unserer Komponenten [Seite 14 Programmiersprachen]. Zuerst wurden die ARM und AVR Boards programmiert [Seite 29 LED-Controller, Seite 35 Spalten-synchronisation]. Danach wurde das Beagle Board programmiert [Seite 32 Programmierung / Installation]. Parallel dazu wurde auch die Client Software für eine potenzielle Webcam oder Video Streaming entwickelt [Seite 34 PC Applikation]. Diese konnte aber gegen Schluss des Projekts nicht verwendet werden, da die Zeit nicht mehr reichte.

Machbarkeitsanalyse

Da wir unsere LEDs mit PWM ansteuern, konnten wir berechnen, wie viele LEDs wir maximal auf ein Board setzen könnten. Wir sind aber wegen Layout Grenzen von fixen Zahlen ausgegangen und haben geprüft, ob unsere Anzahl LEDs überhaupt möglich sind. Mit 25 Bildern pro Sekunde (= 25 Umdrehungen pro Sekunde) und einer Breite von 1000 Pixeln, ergibt sich ein Spaltentakt von 25 kHz. Das heisst, dass wir pro Spalte 40 μ s zur Verfügung haben. Diese Zeit dividieren wir durch unsere drei Farben, womit wir pro Farbe noch 13.3 μ s zur Verfügung haben. Da die Farben über einen Bus gesteuert werden, müssen wir vor jeder Farbe eine Busumschaltung machen, damit die Farbe gewählt werden kann.

Für unsere Boards haben wir 64 RGB-LEDs vorgesehen. Mit unserem STM32-Controller lassen sich 16 LEDs gleichzeitig schalten (16-bit Bus). Das Umschalten benötigt gemäss Oszilloskop 28 ns. Da wir mit 4-bit Farbtiefe rechnen, kommen wir auf folgende Rechnung:

$$(4 \text{ Busumschaltungen} \times \text{Anzahl PWM Stufen} + 2 \text{ Bus}) \times 28 \text{ ns} = 13.3 \mu\text{s}$$

Daraus ergeben sich maximale 118.25 PWM Stufen. Da aber ein kleinerer Teil davon für Synchronisationsberechnungen und die Datenübertragung benötigt werden, kann man diese 118.25 Stufen sicher nicht vollständig ausnutzen. Für 4 Bit hat man 15 PWM Stufen, die benötigt werden. Da wir Farben aber logarithmisch wahrnehmen, benötigen wir aber nicht 2^{bits-1} PWM Stufen sondern etwa $3^{bits}-1$ Stufen. Damit käme man dann auf 80 PWM Stufen, was aber immer noch machbar ist.

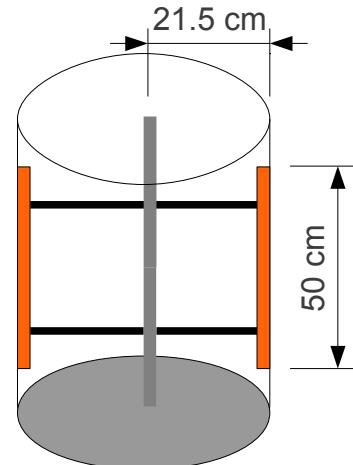
[PA] Rotierendes LED Display

Andreas Butti, David Halter

Die zweite grosse Frage war, ob die USB Geschwindigkeit ausreichend ist, für unsere Spezifikation. Dies haben wir aber dummerweise beim ersten Mal nicht korrekt gerechnet. So haben wir vergessen die Umdrehungszahl mit einzubeziehen. Ein Bild mit 1000×128 Pixeln benötigt 192 kByte Speicher. Da wir die Umdrehungszahl nicht eingerechnet haben, war diese Zahl natürlich weit unter den maximal möglichen 1500 kByte / s unserer Full Speed USB Connection. Erst mit dem vollständig funktionsfähigen Konstrukt haben wir herausgefunden, dass wir die notwendige Geschwindigkeit nicht erreichen. Deshalb haben wir dann die Farbtiefe um einen Faktor zwei reduziert, sowie die Pixelreihen um circa einen Faktor 3 auf 344.

Dieser Faktor entstand daraus, dass unsere Pixel wenn schon kleiner, wenigstens quadratisch sein sollten. Im nebenstehenden Bild sieht man wie die Größenverhältnisse sind. Teilt man den Umfang 135 durch die Höhe 50, erhalten wir einen Faktor von 2.7. Dies kann man dann mit den 2×64 -LED-Boards multiplizieren und man erhält eine Auflösung in der Breite von 345.6 Pixeln. Dies haben wir dann auf 344 Pixel gerundet.

Mit diesem Faktor und dem Faktor 2 der Farbtiefe, war es uns dann möglich unsere LED-Balken rotieren zu lassen und uns nicht mehr mit USB-Geschwindigkeitsengpässen zu beschäftigen.



Umfang 135 cm

Abbildung 9: Skizze Aufbau

LED Elektronik

LEDs und die dazugehörige Ansteuerung sind die Hauptpunkte unserer Arbeit. Wichtige Punkte für die LED-Evaluation sind natürlich die Wellenlängen und die Geschwindigkeit der Schaltelektronik.

LED Auswahl

Um auszuwählen welche LEDs sich für uns eignen könnten, haben wir einige vom Typ her verschiedene LEDs bestellt. Darunter waren sowohl RGB-LEDs, wie auch einzelne Farben (Rot, Grün, Blau, Gelb). Ziemlich schnell haben wir bemerkt, dass sich einige LEDs schon von ihrer Größe her nicht eignen würden, da einige mit 5 mm viel zu gross waren und andere mit einer Größe von einem Millimeter und

Pins auf der Rückseite kaum lötbar waren. Wir haben uns dann auch gegen eine Linse entschieden, da diese ab gewissen Winkeln massiv schlechter wahrnehmbar ist.

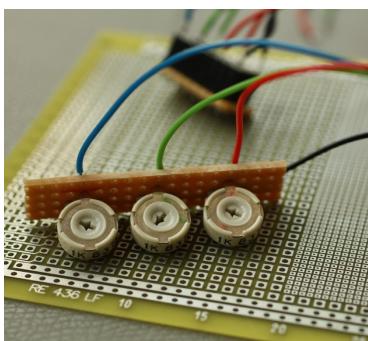


Abbildung 11: Potentiometer für RGB Test

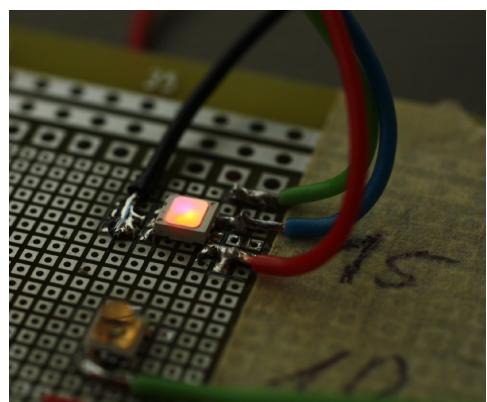


Abbildung 10: RGB LED

Schlussendlich haben wir uns dann für RGB-LEDs des Typs SFT 825N-S, in der Größe $2.8 \times 3.2 \times 1.4$ mm (B \times T \times H) entschieden, weil RGB LEDs für uns im Umgang leichter sind. Damit haben wir uns ein paar Bauteile gespart, worüber wir auch froh sind, gerade bezüglich Layout.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Bei den RGB LEDs ist aber das Problem, dass diese bei voller Aussteuerung nicht weiss, sondern leicht bläulich leuchtet, daher wäre eine Möglichkeit zusätzlich zu rot, grün und blau eine gelbe LED anzubringen. Da unsere LEDs aber sowieso alle Vorwiderstände brauchen, machte dies für uns keinen Sinn. Daher haben wir getestet ob auch nur mit dem RGB ein klares weiss mischbar ist. Mit $3 \times 1 \text{ k}\Omega$ Potentiometer konnten wir mit dem ausgewählten RGB LED ein klares weiss mischen und die Vorwiderstände für die einzelnen Farben entsprechend auswählen.

Die RGB LEDs werden multiplexed nacheinander angesteuert, um Leitungen und Ports zu sparen. Das heisst konkret, dass wir einen Zeilen- und einen Spaltentreiber ansteuern. Die Spalten sind jeweils die 3 verschiedenen Farben und die Zeilen die 64 LEDs, die sich auf jedem Board befinden. Als Treiber wurden FETs verwendet, um den Strom der LEDs schalten zu können.

Jede RGB-LED wird über einen FET angesteuert, alle 3 Kathoden der RGB LEDs sind zusammengehängt, um die Farben einzeln anzusteuern werden die Farben (alle Anoden) über einen Zeilentreiber gesteuert. Diese Ansteuerung besteht aus einem NOT Gatter als Treiber für die 3 FETs, die sich unten auf dem Board befinden. Dieses Gatter liefert uns 40 mA bei 5 V.

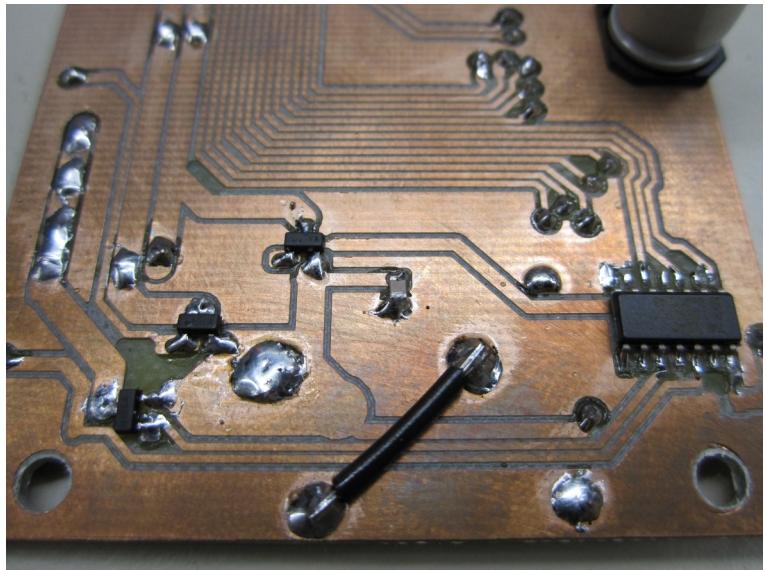


Abbildung 12: Zeilentreiber

Anhang CD: RGB-LED.pdf

FET Auswahl

Für die Schaltung der LEDs verwenden wir FETs. Der Vorteil gegenüber Transistoren ist für uns hauptsächlich, dass wir keine Vorwiderstände benötigen.

Bei der Auswahl der FETs hat uns Herr Hochreutener beraten. Für den Zeilentreiber haben wir die Geschwindigkeit berechnet, mit der wir die FETs schalten können.

C	Kapazität
U	Spannung
I	Strom
t	Zeit
Q	Elektrische Ladung

$$C = \frac{\Delta Q}{\Delta U} = \frac{I \cdot \Delta t}{\Delta U}$$

$$Q = I \cdot \Delta t$$

$$1) \Delta t = \frac{C \cdot \Delta U}{I} = \frac{10 \mu\text{F} \cdot 5\text{V}}{50\text{mA}} = 1\mu\text{s}$$

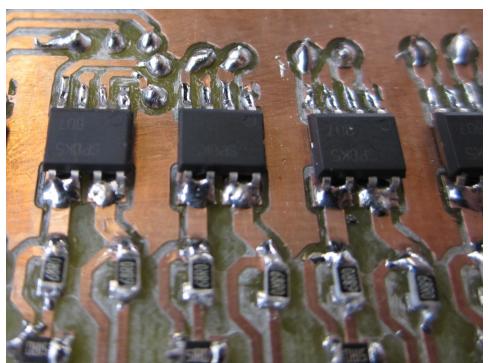


Abbildung 13: FETs für LED Ansteuerung

$$2) \Delta t = \frac{Q}{I} = \frac{180\text{pC}}{40\text{mA}} = 4.5\text{ns}$$

Der erste berechnete FET mit einer max. Schaltgeschwindigkeit von $1 \mu\text{s}$ war für uns zu langsam. Der zweite berechnete FET mit einer Schaltgeschwindigkeit von 4.5 ns haben wir dann gewählt.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Jedoch haben wir beim Testen festgestellt, das unsere FETs nicht 100%ig korrekt schalten. Beim ausschalten glimmen die LEDs noch kurz nach. Auch wenn dies nur sehr kurz ist, kann dieses nachglimmen der LEDs wahrgenommen werden. Dies ist aber nur der Fall, wenn die Farben sehr dunkel ausgegeben werden (das heisst mit PWM: sehr kurz).

Da wir aber sowieso nichts mehr an der Hardware hätten ändern können, und es auch im Betrieb nicht oder kaum sichtbar ist, haben wir das ganze so belassen. Wir haben auch nicht genau herausgefunden was das Problem ist.

Anhang CD: FET-Zeilentreiber.pdf, FET-Treiber-RGB.pdf, Dual-FET-LED.pdf

Schema

Um die Programmierung einfach zu gestalten wollten wir die LEDs zuerst in der Pinreihenfolge anhängen. Dies war jedoch nicht möglich, da die Pinreihenfolge nicht sinnvoll angeordnet war, sondern wie in Abb. 14 zu sehen ist: Wild verteilt.

Daher mussten wir beim Erstellen des Layouts das Schema immer wieder anpassen, damit es übereinstimmte, was ein zusätzlicher Aufwand war.

Figure 5. STM32F103xC and STM32F103xE performance line LQFP144 pinout

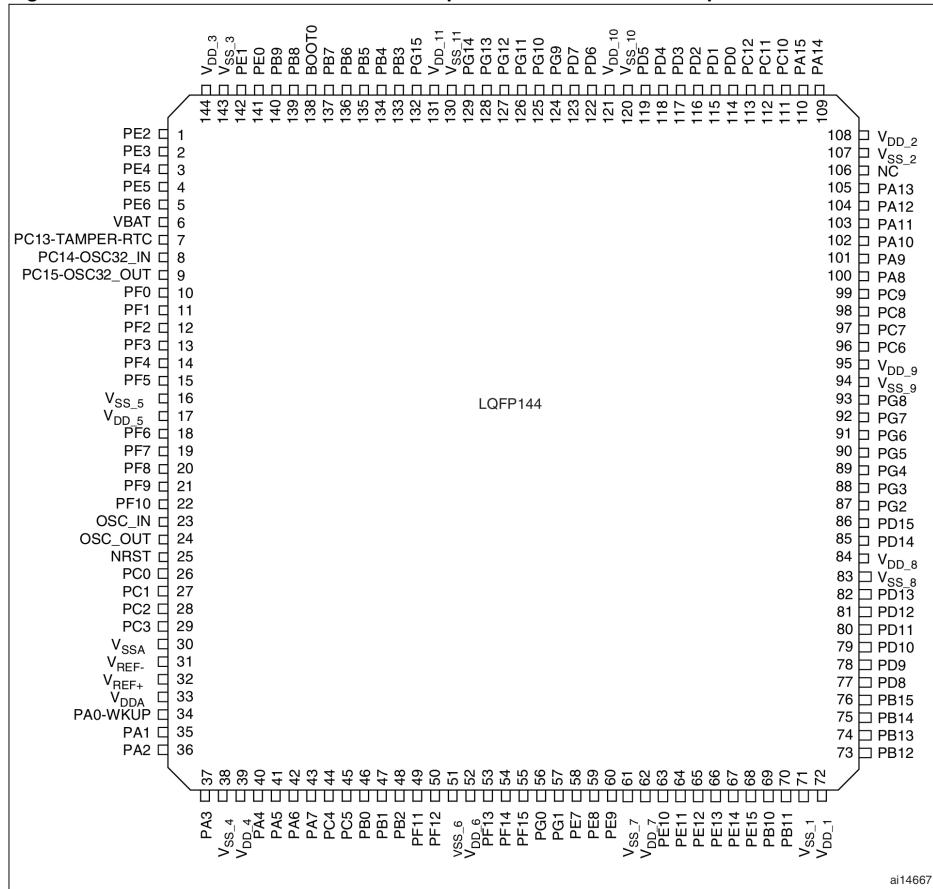


Abbildung 14: Pinout ARM Controller

Anhang: Das komplette Schema ist im Anhang zu finden auf Seite II, III, IV

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Platinen herstellen

Die Platinen konnten wir intern an der ZHAW fräsen lassen. Laut Angaben der zuständigen Personen können Bahnen bis 0.1 mm Dicke gefräst werden. Der einzuhaltende Mindestabstand ist ebenfalls 0.1 mm, wobei sowohl der Mindestabstand als auch die Mindestleiterbahnbreite auf unseren Platinen 0.2 mm betragen. Nur die Durchkontaktierungen müssen leider von Hand gemacht werden. Bei ca. 130 Stück × 4 Boards war dies doch ein beträchtlicher Aufwand.

Nach einer Woche Layouten haben wir beim Überprüfen festgestellt, dass der Footprint der LEDs zu gross ist. Was wir übersehen hatten war, dass es verschiedene PLCC-6 LED-Gehäusegrößen gibt und wir die falsche Grösse erwischt haben. Glücklicherweise konnte das Layout innerhalb eines Tages (dass heisst ein voller Tag Arbeit!) korrigiert werden, es waren keine Komplettänderungen notwendig, nur die LEDs mussten neu platziert und verkabelt werden.

Um das erste Layout zum Laufen zu kriegen waren leider noch ein paar Handkontakteierungen nötig. Das Datenblatt des STM32F103-ZET6 ist nicht besonders übersichtlich, daher haben wir BOOT0 und BOOT1 falsch verkabelt. Wir haben mit Hilfe von Michael Jäger stundenlange gesucht. Zuerst hat er einige nicht korrekt gelöste Stellen gefunden, bis wir dann endlich herausgefunden hatten, dass BOOT1 verbunden werden muss. Das finden des Fehlers war aber mehr Glück als Verstand, wir wollten eigentlich die JTAG Pins herausführen. Dann hat Michael Jäger aber auch noch BOOT0 fest verkabelt, was dann tatsächlich das Problem gelöst hat.

Nach dieser Erfahrung haben wir dann auch die JTAG Pins zum Anlöten herausgeführt, falls weitere Probleme auftreten sollten. Hätten wir die Möglichkeit gehabt JTAG zu verwenden wäre uns der Fehler aufgefallen. Aufgrund Platzmangels haben wir auf diesen Anschluss verzichtet. Um nicht nochmals den selben Fehler zu machen haben wir die JTAG Pins zumindest so herausgeführt, dass wir an Kabel anlöten hätten können. Dafür haben wir beim zweiten Layout den Uhrenquarz weggelassen, den wir nicht gebraucht hatten. Damit hatten wir etwas Platz gewonnen.

Die zweite Version des Layouts hat dann auf Anhieb geklappt. Alle Fehler konnten behoben werden. Wobei auch die erste Platine für das Projekt verwendet werden konnte, nachdem die Fehler von Hand korrigiert wurden.

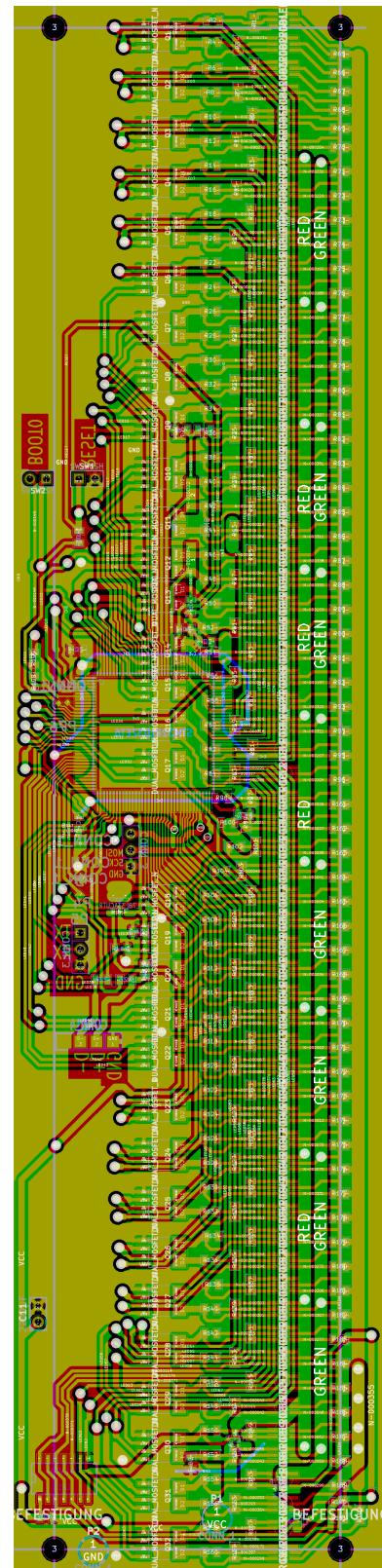


Abbildung 15: Platine

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Auf der rechten Seite ist das Layout abgebildet. Das Board ist 25 cm hoch. Rot ist die Vorderseite, Grün ist die Rückseite und Blau sind Handverbindungen, die leider nicht zu vermeiden waren. Zu sehen ist, dass alle LEDs auf der Vorderseite angebracht wurden, der Controller hingegen auf der Rückseite angelötet ist.

Um Durchkontaktierungen zu vermeiden wollten wir zuerst den Controller und die LEDs auf der gleiche Seite platzieren. Dadurch wäre das ohnehin schon breite Board aber noch breiter geworden. Einer der Gründe warum das erstellen des Layouts mehr als zwei Wochen gedauert hat: Um festzustellen das dies nicht geht haben wir natürlich versucht ein brauchbares Layout zu erstellen, was jedes mal eine Menge Zeit in Anspruch nahm.

Platinen bestücken

Da wir als Informatiker noch nie so kleine Pinabstände wie beim LQFP144 gelötet haben, haben wir das erste Board vom Assistenten Daniel Früh löten lassen. Er hat den Controller in 15 min aufgelötet.

Den zweiten Controller haben wir dann selbst versucht aufzulöten, was in einigen Lötbrücken geendet hat, die nur sehr schwer zu entfernen waren. Schlussendlich haben wir die Hilfe vom wissenschaftlichen Mitarbeiter Michael Jäger in Anspruch genommen, der mit etwas Geduld unsere Lötbrücken wieder entfernt hat. Aufgrund dieser Erfahrung durfte er dann auch die beiden anderen Controller auflöten, was nicht länger gedauert hat als das entfernen unserer Lötbrücken.

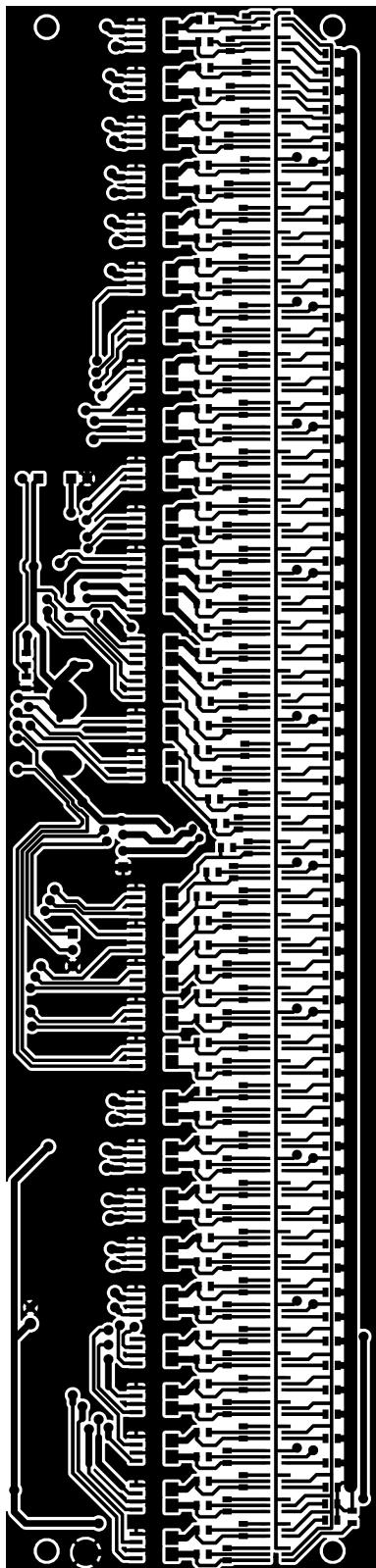


Abbildung 17: Vorderseite

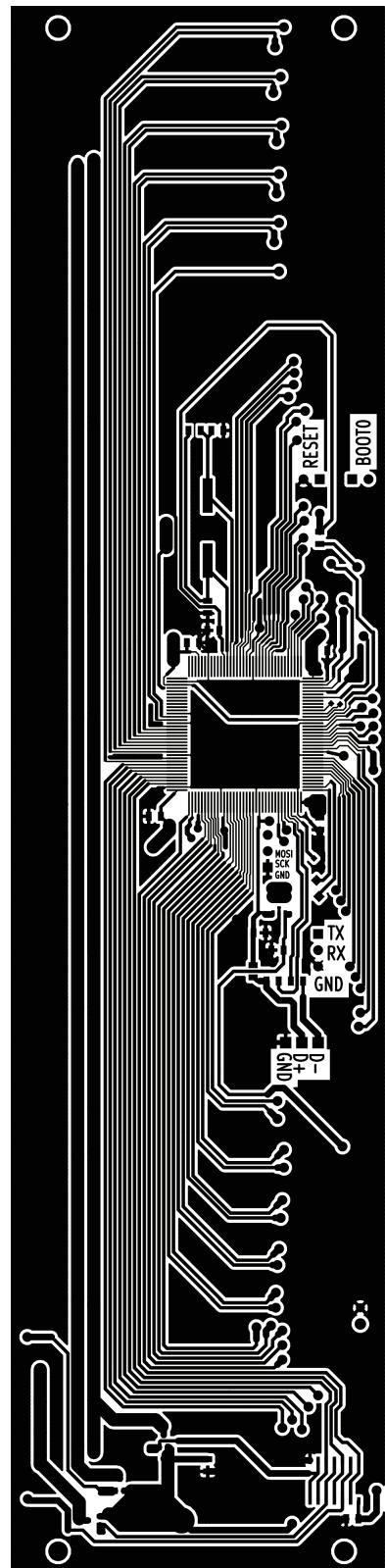


Abbildung 16: Rückseite

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Das Auflöten der FETs war kein Problem, da diese die Bauteile mit den grössten Pins und Abständen auf dem ganzen Board waren. Bei den SMD Widerständen war viel Geduld gefragt, aber trotzdem hatten wir am Schluss Kurzschlüsse. Das unangenehme daran: Wir konnten nicht einfach herausfinden wo. Wir mussten die Leiterbahn durchschneiden, so in der Art eines Binary Search, um herauszufinden wo der Kurzschluss ist. Logischerweise hatten wir keine Lust nochmals alle Widerstände zu entfernen. Wir lernten aus diesem Vorfall, wobei wir beim Löten der letzten beiden Boards Kabel angelötet haben, die wir mit dem Multimeter verbunden haben. So konnten wir direkt beim Löten erkennen, wann wir Kurzschlüsse produzieren. Damit mussten wir nur die zuletzt aufgelöten Widerstände kontrollieren.

Die LEDs wurden dann mit noch mehr Geduld komplett unter dem Mikroskop gelötet, weil sich Lötbrücken ziemlich fatal auf das Board auswirken. Von Kurzschluss bis Prozessorreset war da so ziemlich alles darunter. Unter dem Mikroskop konnten wir sehr genau arbeiten und haben dann vor allem auch gegen Ende kaum mehr Fehler gelötet.

Aufgrund eines falsch aufgelöten NOT Gatters, das wir als Treiber für die Zeilentreiber FETs verwenden, durften wir dann auch den Heissluft-Entlötzapparat testen, der viel besser funktionierte als das entlöten mit dem Lötkolben. Das Bauteil konnte wieder korrekt aufgelötet werden, womit bei den ganzen Lötarbeiten zwar einige Fehler passiert sind, aber kein Bauteil nachhaltig beschädigt wurde.

Anhang CD: KiCAD Projekt und Gerber Files des endgültigen Layouts

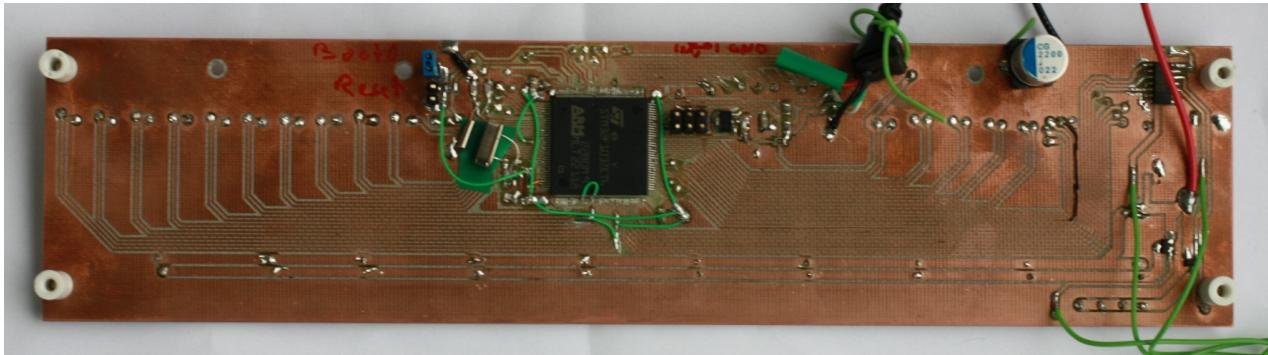


Abbildung 18: Board Rückseite

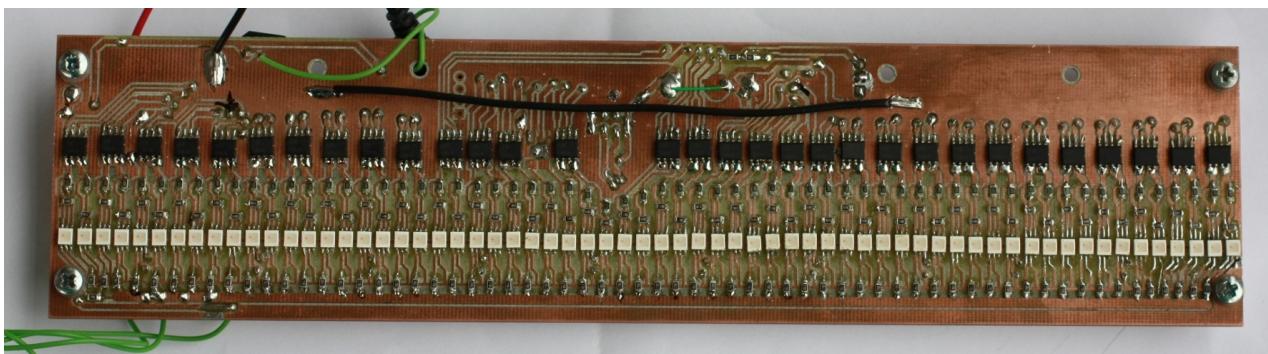


Abbildung 19: Board Vorderseite

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Mechanik

Der Antrieb und die ganze Mechanik wurde zusammen mit dem wissenschaftlichen Mitarbeiter Daniel Früh entwickelt. Er war komplett verantwortlich für den Aufbau eines Grundgerüstes, auf dem wir unsere LEDs rotieren lassen konnten. Das Gerüst besteht aus einem Holz- und einem Aluminiumteil. Auf das Holzgerüst werden wir nicht weiter eingehen, da dies lediglich zur Stabilität des ganzen beiträgt und nicht wirklich relevant ist für die Entwicklung.

Für uns war es sehr wichtig, dass wir uns nicht auch noch um Motoren, Schleifringe und andere mechanische Dinge kümmern mussten, da wir davon nicht gerade viel Ahnung haben. Dennoch wurde etwa wöchentlich (und gegen Schluss öfters) mit Daniel Früh über den Fortschritt diskutiert und allfällige Probleme angeschaut.

Für die Mechanik wurden keine Rechnungen gemacht. Wir haben versucht genug grosse Toleranzen zu haben. Gerade auch, weil wir nicht von Anfang an wussten, wie gross unsere Flügel werden sollten. Die Vorgabe waren 100 LEDs in der Höhe. Jedoch wussten wir erst nach der Evaluation wie gross die LEDs wirklich waren. Und aufgrund des 16 Bit breiten Bus haben wir festgelegt das wir 128 LEDs verwenden. Die Breite war für das Gerüst weniger relevant, und sollte einfach möglichst schmal sein, damit der Luftwiderstand minimiert werden kann. Schlussendlich wird die Breite vom Controller und den vielen Bahnen zum Controller bestimmt.

Aluminium-Gerüst

Die LED-Platinen sind auf einem Gerüst angemacht, welches grösstenteils aus Aluminium besteht. Für die Konstruktion war auch hier Daniel Früh verantwortlich. Die erste Version, die sehr wenig Luftwiderstand hatte, konnte den Fliehkräften nicht

standhalten und streifte beim testen den Holzrahmen. Deshalb wurden die Aluminiumflügel verstärkt. Es musste verhindert werden, dass unsere Prints das Holzgerüst streifen. Die Konsequenz davon war allerdings ein schwereres Gerüst und auch ein höherer Luftwiderstand, was in einer tieferen maximalen Drehzahl von ca. 800 Umdrehungen pro Minute resultierte. Herr Früh



Abbildung 20: Mechanischer Aufbau, halbfertig

[PA] Rotierendes LED Display

Andreas Butti, David Halter

hat dann diesen Aufbau wieder entfernt, und stattdessen die Alubleche mit Gewindestangen verbunden, wodurch wir nun wieder eine Drehzahl von circa 1300 bis 1400 Umdrehungen pro Minute erreicht haben (ohne Platinen).

Motor

Als Motor wurde ein 150 Watt Motor ausgewählt, der das Gerüst mit bis zu 1500 Umdrehungen pro Sekunde drehen lassen sollte. Wie sich aber herausstellte beim testen, wurde diese Drehzahl nicht mehr erreicht, da einerseits durch unsere Platinen und die Befestigungen ein starker Luftwiderstand vorherrschte und andererseits die Kugellager einen relativ hohen Reibungsverlust mit sich brachten.

Wie schnell das Gerüst schlussendlich drehen sollte, wissen wir noch nicht, da wir das System noch nicht auswuchten konnten. Deshalb war es auch noch nicht möglich mit mehr als 400 Umdrehungen pro Minute zu testen, da es bereits da stark beginnt zu schwanken.



Abbildung 22: Lasermessgerät

Die Umdrehungszahl haben wir anfangs mit einem Lasermessgerät erfasst. Später konnten wir die Umdrehungen direkt mit unserem Synchronisationsboard berechnen und von der Konsole ablesen.

Siehe Spaltensynchronisation, Seite 35

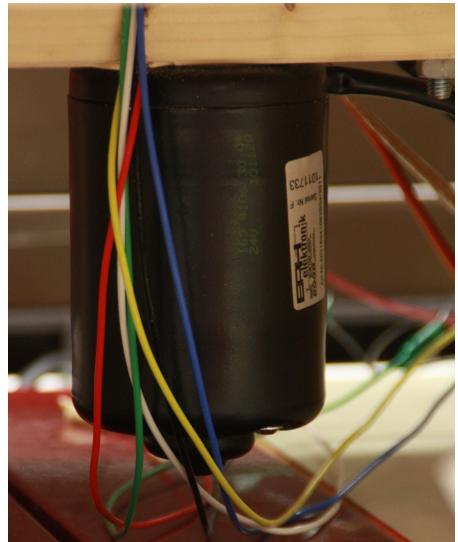


Abbildung 21: 150W Motor

Schleifringe

Um Strom auf das drehende Gerüst zu bringen wurde ein Satz Schleifringe von Daniel Früh evaluiert, mit der Spezifikation von 2000 Umdrehungen pro Minute und 10 Ampère.

Hersteller haben wir nur einen geeigneten gefunden, mit einer Lieferfrist von 6-8 Wochen. Für uns war das natürlich keine Option, da das Semester lediglich 14 Wochen andauert und wir uns erst nach ein paar Wochen Gedanken über die Schleifringe gemacht hatten. Deshalb haben wir dann Schleifringe mit nur 1000 Umdrehungen pro Sekunde (garantiert) gewählt.

Es wurde auch darüber nachgedacht die Stromübertragung über Induktion zu machen. Diese Idee wurde aber verworfen, bevor wir uns über die Technologie selbst Gedanken machten. Eine mehr oder weniger stabile Stromversorgung ist sehr angenehm und wichtig. Ansonsten hätte man auch noch mit Batterien arbeiten müssen. Darum haben wir uns dann sehr schnell für Schleifringe entschieden, da wir weder Zeit noch die Möglichkeit hatten selbst etwas in dieser Richtung herzustellen.

Als der Aufbau dann soweit mechanisch stand, haben wir die Schleifringe getestet. Wir haben über eine Heizung die Spannung gemessen. Es wurden zwei Kabel zusammen gelötet und von aussen gemessen. Insgesamt gibt es 12 Unterbrechungen pro Umdrehung. Entweder hat unser Satz Schleifringe 12 Segmente, oder 6, die aber jeweils versetzt sind.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

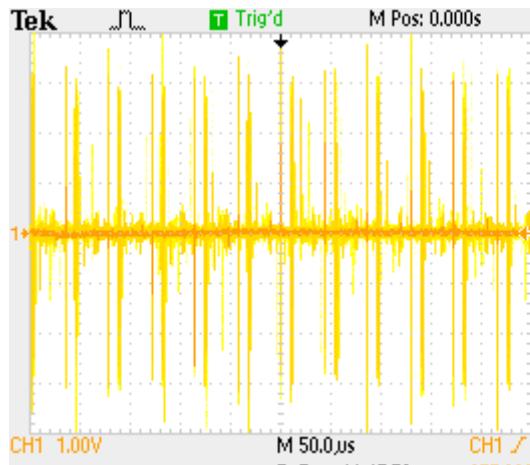


Abbildung 24: Schleifringe Störungen

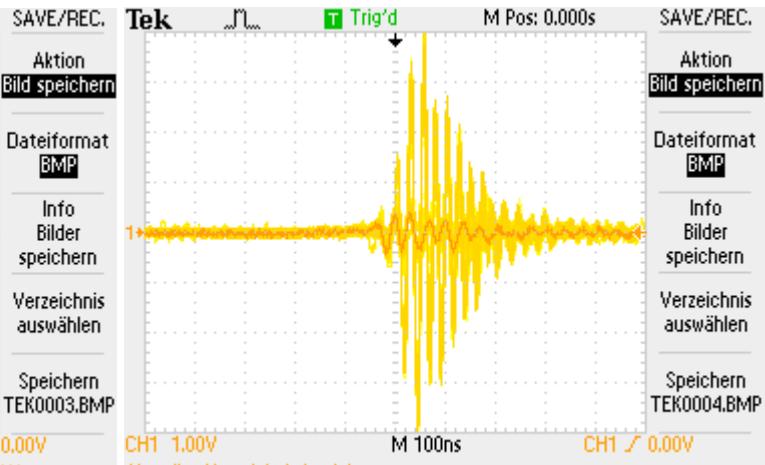


Abbildung 23: Schleifringe Einzelne Störung

Diese Unterbrechung ist für uns aber kein Problem, da wir vor jedem Board einen Elko mit 2200 μ F verbaut haben, der genug Energie liefern kann für diese kurze Zeit. Zudem sind in Prozessornähe auch noch einige 100nF Kondensatoren.

Synchronisation

Der komplexeste Teil unserer Arbeit war die Synchronisation der LED Boards mit der Drehung. Leider können wir den Motor nur von Hand regeln, daher müssen wir die LED Ausgabe auf die Geschwindigkeit der Rotation anpassen.

Technisch gesehen läuft diese Synchronisation in 3 Teilen ab.

1. Messung der Zeit

Auf dem AVR Board wird die Zeit gemessen. Dies funktioniert folgendermassen:

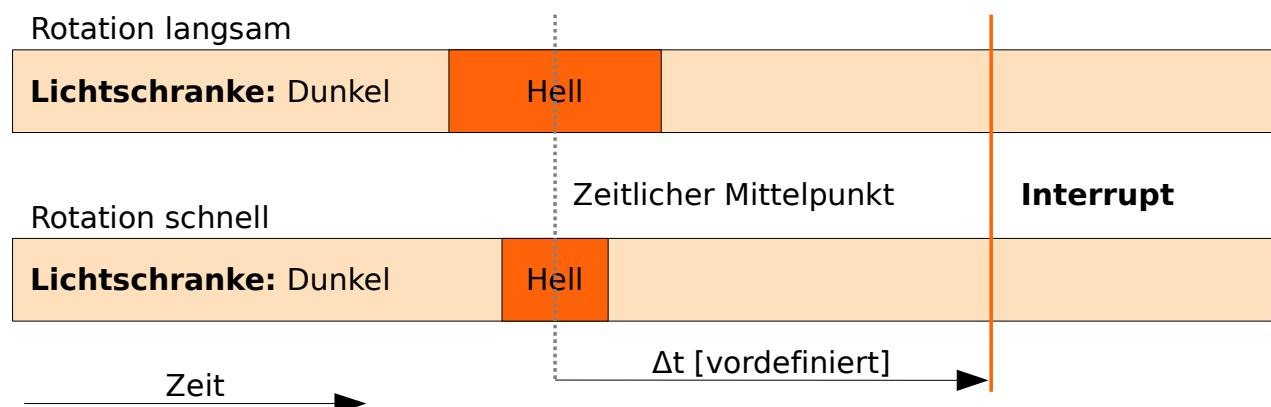


Abbildung 25: Skizze Lichtschrank

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Es wird bei jeder Rotation gemessen wann Licht auf den Sensor trifft und wann es wieder dunkel wird. Aus diesen beiden Zeiten wird der Mittelpunkt berechnet, und Δt dazu addiert. Diese Zeit kann konfiguriert werden und muss so gross sein das auch bei langsam Drehungen genug Zeit bleibt um einen Interrupt zu schalten.

Der Interrupt wird durch einen Timer ausgelöst (compare match, bzw. nach genügend Überläufen, je nach dem wie gross der Timer eingestellt ist).

2. Synchronisation der LED Ausgabe

Beim Eintreffen eines Interrupts muss an die richtige Stelle im Bild gesprungen werden. Dies kann bedeuten, dass einzelne Zeilen verworfen werden oder vice versa einfach doppelt ausgegeben werden müssen. Für uns ist der Interrupt der Nullpunkt des Bildes.

3. Datenrate

Das Beagle Board erhält die gemessene Rundenzeit über die serielle Schnittstelle und übermittelt sie weiter über USB an die LED Boards. Diese Zeit ist auf $\frac{1}{12'000'000} s$ aufgelöst, wenn auch nicht ganz so präzise. Dadurch kann berechnet werden wie viele Spalten pro Sekunde ausgegeben werden müssen, damit die Spalten immer zum richtigen Zeitpunkt angezeigt werden. Das Beagle Board sendet immer so viele Bilddaten wie nur möglich. Die LED Boards bremsern automatisch aus, wenn ihre Buffer voll sind.

Wenn das Gerüst allerdings zu schnell rotiert, kann es vorkommen, dass zu wenig Daten vorhanden sind, was das Bild dann aus der Synchronisation bringt. Die Synchronisation wird dann neu gestartet.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

LED-Controller

Controller Auswahl

Schon zu Beginn entschieden wir uns für ARM-Controller, da dies ein Standard ist im Mikrocontroller-Bereich und wir diesen Prozessor gerne näher kennen lernen wollten. Die einzige wirklich brauchbare Alternative wären FPGAs gewesen. Wir kennen uns aber im Mikrocontroller Bereich besser aus. Die Auswahl des Mikrocontrollers erfolgte vor allem nach dem Kriterium der maximal benötigten Anzahl Output-Ports und der Geschwindigkeit (MHZ bzw. MIPS). Es war uns wichtig, dass wir mit einem Controller möglichst viele LED's ansteuern konnten. Wie sich später aber heraus stellte, war die Anzahl der Output-Ports nicht so entscheidend, wie die CPU-Geschwindigkeit, beziehungsweise der Output Clock. Das Problem mit den Output Ports ist, dass man sowieso immer nur maximal 16 bit Verarbeitung pro Systemclock gleichzeitig hat und somit doppelt so viele Output Ports nur mit doppeltem CPU-Speed betrieben werden können.

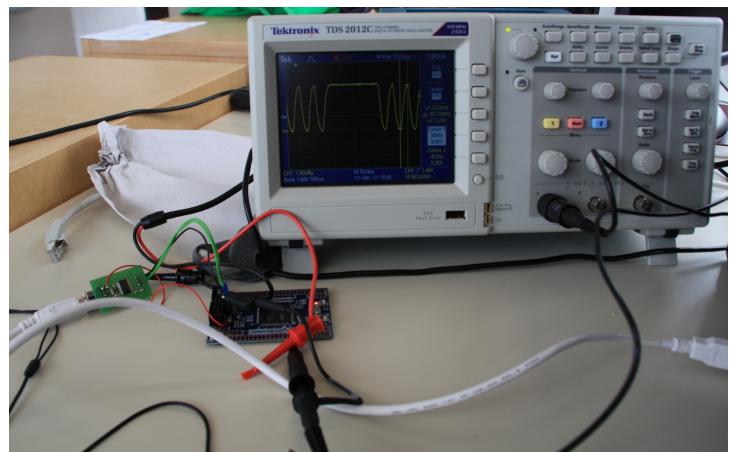


Abbildung 26: ARM Evalboard mit Osziloskop

Der Controller selbst musste aber auch einen Hardware USB Stack haben, da wir sonst unsere Kommunikation nicht aufrecht halten hätten können (viele Daten über USB → viele Interrupts). Deshalb entschieden wir uns dann für einen 72 MHz ARM der Marke ST, einen STM32F103ZET6.

Programmiert haben wir den Controller nicht über JTAG, sondern über UART. Wir haben auf dem Evalboard den Controller zuerst mit dem JTAG programmiert, jedoch war das für uns ziemlich umständlich, da wir unter Linux entwickeln und der JTAG den wir verwendet hatten, nicht gut unterstützt war.

Anhang CD: ARM-Controller.pdf

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Unser ST-Microcontroller zeichnet sich vor allem durch seine schlechte Dokumentation aus. Das mag jetzt ein wenig zynisch formuliert sein, aber die Dokumentation ist wirklich sehr unklar und aufgeteilt an diversen Orten.

“

Im Datasheet sind die speziellen Eigenschaften einer bestimmten Modellreihe beschrieben und die exakten Daten und Pinouts aufgeführt. Die Peripheriemodule werden nur aufgeführt, nicht detailliert beschrieben. In der Referenz ist der gesamte Controller mit Peripheriemodulen im Detail beschrieben, gültig für alle STM32 Controller. Details zum Prozessorkern selbst und den nicht STM32-spezifischen mit dem Cortex-M3 Core assoziierten Modulen wie dem Interrupt-Controller und dem Systick-Timer findet man jedoch nicht dort, sondern im Cortex-M3 Manual. Wer nicht die ST Firmware-Library verwendet, der benötigt zusätzlich die Flash Programming Reference für die Betriebsart des Flash-ROMs, d.h. die frequenzabhängige Konfiguration der Waitstates. Hinzu kommen optionale Dokumente von ARM, die den Cortex-M3 Kern beschreiben. Hier gibt es den Opcode wenn man ihn in Assembler programmieren möchte. Zusätzlich sollten auch die Errata Sheets beachtet werden. Empfohlen sei auch die Appnote "STM32F10xxx hardware development: getting started".

”

Zitat: <http://www.mikrocontroller.net/articles/STM32>, Stand 22.12.2011

Leider hatten wir erst gegen Ende der PA alle Unterlagen zusammengefunden. Daher war es von Anfang an eine grosse Herausforderung am Controller überhaupt einmal ein LED blinken zu lassen. ST stellt zwar einige Beispiele bereit, die aber nur auf ihren Testboards, mit proprietärer Software laufen. Die Testboards die wir gekauft haben, wurden für weniger als 20 CHF aus China importiert und sind nur mit dem nötigsten bestückt. Sie funktionieren deshalb auch nicht mit den Beispielen von ST. Wir haben bewusst ein minimales Board gewählt. Zuerst hatten wir von der ZHAW ein Evalboard, wo aber sehr viele Pins bereits belegt waren. Dies war für uns aber eher hinderlich, da wir ganze Ports schalten müssen und die Geschwindigkeit dieser Ports testen, etc.

Da wir beide sehr nahe an der Open-Source Szene sind, mit Linux arbeiten und vereinzelt an Open Source Projekten mitgearbeitet haben, war für uns von Anfang an klar, dass wir unsere Software mit einem GNU-ARM Compiler kompilieren wollten. Allerdings war uns dabei nicht von Anfang an klar, dass damit so einige Probleme auftreten würden. Neben einem stark angepassten Makefile, mussten auch gewisse Komponenten der Software (Vektortabelle) überhaupt erst einmal geschrieben werden für den GNU Compiler.

Glücklich wie wir sind, haben wir dann im Internet sowohl ein Makefile, wie auch eine Vektortabelle gefunden, was wir dann mit den ST Beispielen mischen konnten, um eine erste funktionierende Version zu schreiben, die ein LED blinken liess.

Link: http://code.google.com/p/flyless/source/browse/branches/michael/firmware/startup_stm32f10x_md_mthomas.c?r=7

Da wir aber erst gegen Ende der PA die komplette Dokumentation zusammengefunden hatten, war es für uns teilweise sehr schwer Probleme zu orten. Die USB Dokumentation im Datenblatt ist beispielsweise etwa 5 Sätze lang. Damit kommt man nicht sehr weit. Analog dazu war natürlich das restliche Datenblatt gestaltet.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

USB

Der erste grössere Meilenstein war dann die USB Kommunikation mit dem Board. Früh hatten wir erste Beispiele am laufen, die gut funktionierten. Allerdings funktionierte die Kommunikation nur mit einem anderen Mikrocontroller, auf einer proprietären Entwicklungsumgebung. Bis wir dann einen ersten funktionierenden Transfer auf unserem Testboard laufen lassen konnten, vergingen etwa drei Wochen. Daneben haben wir aber natürlich noch an anderem gearbeitet.

Bulk USB Mode

Der Bulk Mode (64 Bytes pro Paket) hat sehr schnell funktioniert. Für Audio und Video Streams ist dieser aber nicht optimal, da in Fehlerfällen eine "Retransmission", das heisst Neuübertragung durchgeführt wird. Bei Streams will man wenn möglich einen gleichmässigen Datenstrom. Dennoch, der Bulk Mode hat sehr schnell funktioniert, gerade auch weil er ein Standard ist um grössere Datenmengen zu übermitteln.

Für die Übertragung von Ubuntu zum LED Board wurde mit "pyusb" gearbeitet, einem Python Paket für die USB Übertragung. Diese Library ist eine Schnittstelle zur Systembibliothek "libusb".

Isochronous USB Mode

Da wir wie oben im "Bulk USB Mode" beschrieben aber mit Streams arbeiten, wollten wir den isochronen Modus nutzen. Eine Implementation von "pyusb" mit isochronem Modus gab es aber nicht. Deshalb machten wir uns daran die isochrone Schnittstelle in pyusb nachzuführen. Nach wochenlanger Arbeit ergab sich dann eine Python Bibliothek, die vom Programmierer von pyusb auch sehr schnell übernommen wurde. Dieser Programmierer hat uns dann auch schon nach wenigen Tagen noch wenige Fehler in unserem Code aufgezeigt (Memory Leaks und Design Kleinigkeiten). Dies hat uns auch wieder sehr schön gezeigt, wie angenehm Open Source Software ist. Wir profitieren nicht nur selbst davon, sondern kriegen bei einer Beteiligung sogar noch gratis Bugfixes. Gerade bei Bibliotheken wäre es also auch für die eine oder andere Firma interessant sich an Open Source zu beteiligen. Weil man eben auch auf Dinge aufmerksam gemacht wird, die einem selbst nicht auffallen.

Trotz all diesen schönen Dingen mussten wir nach wochenlanger Arbeit feststellen, dass sich auf dem Controller gewisse Bits sich falsch verhielten und haben somit den isochronen USB Modus wieder aufgegeben. Konkret heisst das, dass Bits in zwei verschiedenen Adressbereichen im Memory immer auf 0 geschaltet waren, womit jede Datenübertragung scheiterte. Schlussendlich mussten wir also wieder auf den Bulk Modus zurück wechseln. Das ist zwar keine schöne Lösung, aber ohne saubere Dokumentation seitens ST, für uns eine Unmöglichkeit herauszufinden, warum der isochrone Modus bei uns nicht sauber funktioniert.

Links: <https://github.com/davidhalter/pyusb>, <https://github.com/walac/pyusb>

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Hauptcontroller

Die Aufgaben des Hauptcontrollers sind vielseitig. Er ist aber der Master über alle anderen Boards und kann sowohl Resets auslösen, wie auch die Daten an die LED Boards senden

Anforderungen:

Hauptcontroller	AVR Board	LED Boards
<ul style="list-style-type: none">● WLAN Empfang der Daten● Aufbereitung der Daten PWM-Daten berechnen● SSH für die Programmierung● Steuerung der anderen Boards - „Master“● Linux ist zwingend● Mehrere USB Anschlüsse (evtl. Mit USB Hub)	<ul style="list-style-type: none">● Synchronisation● Reset und Boot0 der LED Boards schalten● Interrupt Leitung an die LED Boards● Serielle Schnittstelle an das Beagle Board, wird damit gesteuert	<ul style="list-style-type: none">● Wird per USB ange-steuert● schaltet LEDs und muss deshalb schnell sein.● Viele Output Ports

Auswahl

Ein Punkt der Aufgabenstellung war, dass wir die Übertragung per WLAN unterstützen, um live Filme zum Beispiel von einer Webcam zu übertragen. Dafür brauchen wir einen genug leistungsstarken Prozessor. Es gibt natürlich viele Embedded Boards, aber nur wenige davon kennen wir auch. Eines der wenigen bekannten Boards war das Beagle Board: Ein Open Source ARM Board, das in verschiedenen Varianten verfügbar ist und für das (Open Source) Software bereits verfügbar ist. Wir haben uns für die schnellste und vor allem mit 4 USB Anschlüssen ausgestattete Version entschieden. Darauf ist ein ARM mit 1 GHz und 512 MB RAM. Als persistenter Speicher ist eine SD Karte verbaut.

Programmierung / Installation

Da wir unter Ubuntu entwickeln, haben wir auf dem Beagle Board auch Ubuntu installiert. Dafür war ein fertiges Image bereits verfügbar, welches ohne Konfiguration und Installation läuft. Dieses Image enthält alle Software-Packages des Standard-Ubuntu. Da wir unsere Ansteuerung in Python geschrieben haben, können wir diese ohne Anpassungen direkt auf dem Beagle Board ausführen.

Es hat uns aber erstaunt, dass das Beagle Board auf Anhieb mit allen Komponenten sauber lief. ARM ist im Server / Desktop Bereich (noch) kein Standard, so laufen sowohl Windows, wie auch Macintosh Systeme (noch) nicht auf ARMs. Sowohl die Treiber für WLAN, wie auch alle benötigten Packages waren direkt verfügbar, wir mussten also nichts selbst kompilieren. Das ist schon sehr erstaunlich und angenehm. Gerade auch, weil dies uns viel Zeit abgenommen hat. Wir haben also mehr oder weniger (abgesehen von der Geschwindigkeit) ein System, das identisch ist mit unseren Computern. Dies ist wohl ein weiterer Punkt, wo man die Macht von offenen Systemen sieht.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Datenübertragung (WLAN / Funk / Induktion)

Um Daten zum Beagle Board zu übertragen standen uns mehrere Möglichkeiten zur Verfügung:

- Induktion
 - Erfordert spezielle Hardware auf Sender- und Empfängerseite
 - Hardware, z.B. aus Lesekopf einer Video Rekorders
- Funk
 - Funkmodul, SPI oder ähnliche
 - Erfordert spezielle Sender- und Empfängerhardware
- Bluetooth
 - Keine Spezielle Hardware erforderlich, schon vorhanden auf Beagle Board
- WLAN
 - Nicht vorhanden auf Beagle Board, aber per USB Stick einfach realisierbar

Bluetooth ist zu langsam für uns und relativ kompliziert um Daten zu übertragen. Am einfachsten ist für uns WLAN, da wir einfach einen TCP Socket öffnen können, um Daten zu streamen. Daher haben wir uns für WLAN entschieden.

WLAN USB-Sticks gibts wie Sand am Meer. Wir mussten aber beachten das es einen Open Source Treiber für Linux gibt, da wir keinen precompilierten x86 Treiber auf unserem ARM Board ausführen können.



Abbildung 27: Gewählter WLAN-USB Stick

Leider ist nur bei wenigen USB-Sticks Linux Support angegeben, auch wenn die meisten wohl von Linux unterstützt würden. Daher mussten wir mit Google herausfinden, zu welchen USB-Sticks tatsächlich ein Open Source Treiber verfügbar ist. Wir haben uns daher schlussendlich für einen **D-Link DWL-G122** entschieden. Dieser Stick verfügt über eine RaLink RT73 Chipset, für welches ein Open Source Treiber verfügbar ist.

Wir mussten jedoch nicht selbst kompilieren, da Ubuntu 11.10, welches auf unserem Beagle Board läuft, den Treiber bereits enthielt. Mit Ubuntu war es dann auch möglich ein Ad-Hoc Netzwerk einzurichten. Dies hatte allerdings so seine Tücken, da der Network-Manager von Ubuntu nicht so funktionierte, wie wir ihm befahlen.

Datenübertragung

Die Datenübertragung wurde leider nicht mehr komplett realisiert, da die Zeit nicht gereicht hat. Da wir aber anfangs noch viel Wartezeiten hatten, weil nur eine Person hundertprozentig am Projekt arbeiten konnte, arbeiteten wir bereits an der Implementierung dieser Software. Wir haben als erstes versucht die Bilder mit einem kleinen Webserver und dann per PUT über HTTP zu übertragen, haben jedoch bald festgestellt, dass der HTTP Overhead uns unnötig ausbremsst.

Daher haben wir uns entschieden die Bilder nur per TCP ohne Protokoll zu streamen. Damit erreichen wir eine viel höhere Geschwindigkeit, zumindest dann wenn wir nicht auf eine Antwort

[PA] Rotierendes LED Display

Andreas Butti, David Halter

warten. Sobald wir bei TCP auf die Antwort unseres Servers warten, bricht die Framerate extrem ein. Auf dem getesteten Netbook kamen wir gerade noch auf 2 übertragene Bilder pro Sekunde, während ohne Antwort locker 30 Bilder pro Sekunde erreicht werden konnten.

Somit steht fest, dass wir für die Übertragung der Bilder einen einfachen TCP Stream verwenden werden, und einen zweiten TCP Stream oder einen HTTP Webserver für die Steuerdaten. Damit können wir dann zum Beispiel anzeigen, wie schnell das Gerüst aktuell dreht.

Für den Test und die Entwicklung haben wir SSH verwendet. Darüber haben wir unsere Testbilder einfach als Binärdatei übertragen und dann die Bilder mithilfe der Python Skripts angezeigt.

PC Applikation

Unsere PC Applikation ist leider nicht fertig geworden. Wir haben nur Teile davon geschrieben, wie eine Übertragung von Bildern auf das Beagle Board über TCP, Siehe Seite 33 [Datenübertragung].

Wir haben zudem schon eine Implementation für das Konvertieren von Bildern in unser eigenes Format. Somit konnten wir zum Beispiel auch das ZHAW Logo, dass auf der Titelseite abgebildet ist erstellen.

Das Format ist ganz einfach aufgebaut. Der Bytestrom wird nicht mehr als Bytestrom, sondern als Bitstrom angesehen. Wir lesen / schreiben immer 2 Bit als pro Pixelfarbe. Drei „2 bit Werte“ ergeben also einen Pixel (rot, grün, blau). Diese Pixelwerte werden in der richtigen Reihenfolge in das File geschrieben. Zuerst in der Y-Achse, dann in der X-Achse, da dies auch so angezeigt wird. So werden alle 128×344 Pixel ($y \times x$) in der Datei abgelegt. Die Dateien haben keinen Header.

Die Ansteuerung einer Webcam mit Java ist kein Problem, wir hätten unter Linux V4LVJ (Video for Linux for Java) verwendet, welches wir bereits kannten und ziemlich einfach zu verwenden ist. Damit würde zwar die Webcam Ansteuerung unter anderen Plattformen als Linux nicht funktionieren. Dies sollte aber kein Problem sein, für eine Präsentation muss einfach ein Notebook mit Linuxinstallation genommen werden, was an einer technischen Fachhochschule nicht alzu schwer zu finden sein sollte.

Links: <http://code.google.com/p/v4l4j/>

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Spaltensynchronisation

Da der Motor nicht genau geregelt werden kann und wir zum testen der passendsten Geschwindigkeit sowieso gerne variieren würden haben wir festgelegt, dass wir für die Synchronisation ein zusätzlichen Mikrocontroller nehmen. Wir haben uns für einen ATMega168 in SMD Form entschieden.

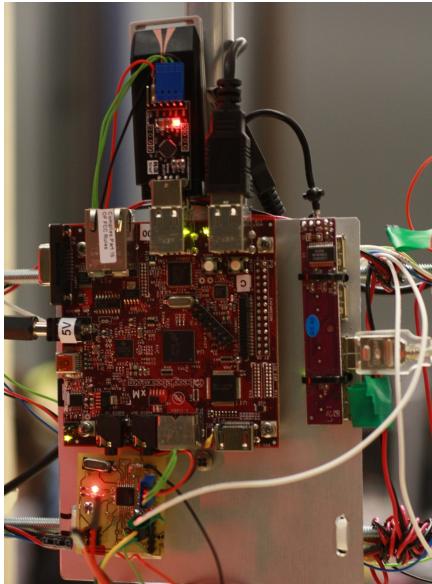


Abbildung 29: Beagle- und Synchronisationsboard

Für diesen Controller haben wir dann unser erstes Board fräsen lassen, und das Ergebnis war sehr gut. Der Controller liess sich auch gut auflöten.

Mit einem IR Transistor, der am ADC angeschlossen ist wird die Helligkeit gemessen.

Die beiden Widerstände die Rechts in Abbildung 6 zu sehen sind bestimmen die Empfindlichkeit. In unserem Fall einen $10\text{k}\Omega$, und da bei diesem die Empfindlichkeit noch viel zu hoch war noch einen 680Ω Huckepack. Später haben wir die Widerstände doch durch ein Potentiometer ersetzt, da eine Abstimmung sonst fast nicht möglich ist.

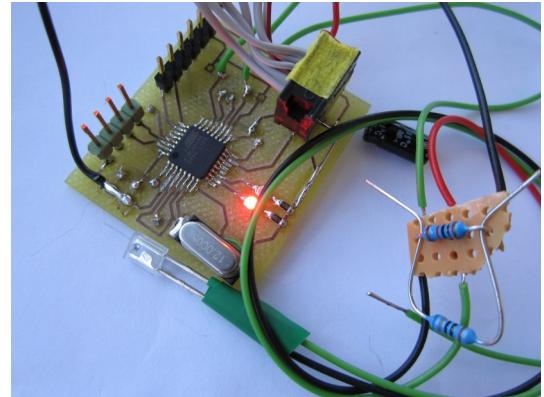


Abbildung 28: Synchronisationsboard

Anhang CD: KiCAD Projekt und Gerber Files, Datenblatt ATMega168.pdf

Synchronisationsboard Protokol

Das Synchronisationsboard wollten wir zuerst über SPI mit den einzelnen LED-Boards verbinden. Nach einigen Überlegungen haben wir gemerkt, dass dafür vom AVR-Controller mehrere Leitungen an jedes Board herausgeführt werden müssten, worauf wir aber verzichten wollten (dank BOOT0 und RESET Verkabelung haben wir nun genau diesen Kabelsalat trotzdem). Daher haben wir uns entschieden vom AVR lediglich eine Leitung als Interrupt Leitung an die ARMs zu hängen. Die Kommunikation erfolgt direkt über UART mit dem Beagle Board. Das Beagle Board verfügt über eine RS232 Schnittstelle, die mit einem MAX232 mit dem AVR verbunden werden sollte. Leider hat aber die serielle Schnittstelle des Beagle Board nicht funktioniert. Wir konnten diese softwaremässig nicht ansprechen und haben uns daher kurzfristig für ein USB UART Modul entschieden. Die Übertragungsgeschwindigkeit beträgt 57600 Baud.

Im Normalbetrieb wird einfach bei jeder Umdrehung die Zeit gesendet, die die Umdrehung gedauert hat. Hier ist die Ausgabe unseres Python-Testskripts zu sehen, die erste Spalte ist Umdrehungen pro Sekunde, die zweite Umdrehungen pro Minute.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Die folgende Ausgabe ist bereits aufbereitet durch ein Python Script,

5.59635192473	335.781115484
5.27396489649	316.437893789
5.43036797531	325.822078519
5.43037534754	325.822520853
5.43038026238	325.822815743
5.42942940769	325.765764461
5.43951755106	326.371053063
5.43446510598	326.067906359
5.43037534754	325.822520853
5.44045714348	326.427428609
5.60537331086	336.322398651
5.27531425487	316.518855292
5.43524293045	326.114575827

Die zwei Fett markierten Messzeilen zeigen Messfehler. Die Firmware hat bezüglich Messgenauigkeit noch einiges an Potential. Momentan wird jedes mal der ADC getriggert. Besser wäre es über einen IO Pin zu gehen, der sofort reagiert, und nicht wie der ADC unterschiedlich lange braucht. Wie für so einiges hat aber auch hier die Zeit dafür nicht gereicht.

Wird ein Zeichen 'a' an den Controller gesendet antwortet dieser mit dem aktuellen ADC Wert.

```
aadc ->7  
aadc ->8  
aadc ->15  
aadc ->34  
aadc ->29
```

Wird ein Zeichen 'c' gesendet so geht der Controller in den Konfigurationsmodus.

```
c>list  
$ list  
$ adc: adcThreshold = 200  
$ adcd: displayAdcDebug = 0  
$ int: interruptDelay = 100  
$ led: ledMask = 3  
$ dip: debugInterruptPrescaler = 255  
$ boot0 = 0  
$ reset = 0  
>
```

[PA] Rotierendes LED Display

Andreas Butti, David Halter

In diesem Modus werden keine Interrupts mehr ausgelöst, auch wenn die Lichtschranke ausgelöst wird. Es können dann alle wichtigen Parameter ausgelesen oder gesetzt werden. Dies funktioniert einfach über ein 'list'. Dieses Kommando stellt alle Parameter zeilenweise dar. Über ein 'set variable=wert' können diese dann verändert werden.

Es kann über eine Variable auch ein Testsignal konfiguriert werden, welches dann mit ca. 1 - 180 Hz ein Interrupt auslöst.

Die folgende Ausgabe ist direkt vom AVR-Board

und nicht aufbereitet. Zeitbasis ist $\frac{1}{12'000'000}$ s

Hier werden jeweils HEX-Zahlen ausgegeben. Die Zahl bedeutet die Zeit zwischen zwei Interrupts.

```
rota=00030cf5da02
rota=000000c9aa80
rota=000000a637be
rota=000000b4e8bf
rota=000000e021be
rota=0000012aa7fe
rota=000001de1185
rota=000001677e3c
rota=000000e89e81
rota=000000d9fdc1
```

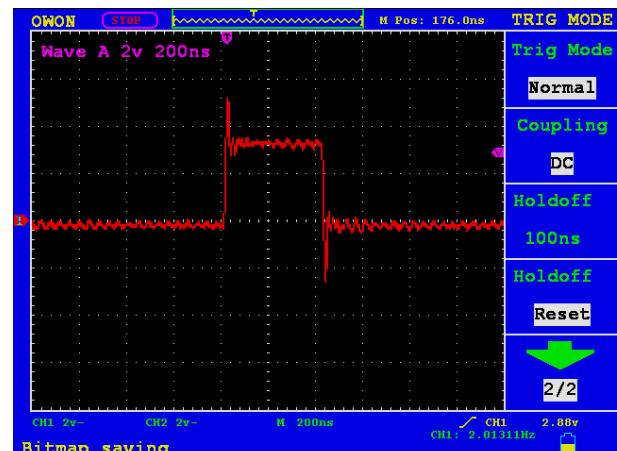


Abbildung 30: Testsignal mit 2 Hz

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Resultate

Der mechanische Aufbau funktioniert, ist jedoch nicht ausgewuchtet. Die LED Boards sind angebracht, jedoch ist nur ein Board in Betrieb. Aus zeitlichen Gründen konnten wir die restlichen Boards nicht mehr in Betrieb nehmen.

Es sind Tools vorhanden die Bilder (JPG, PNG, GIF...) in unser gewünschtes Format konvertieren können. Diese Bilder können dann angezeigt werden, mit einer Auflösung von 344 x 128 und einer Farbtiefe von 2bit. Leider konnten wir das Livestreaming noch nicht in Betrieb nehmen, auch wenn grundsätzlich vorhanden.

Die konvertierten Bilder können manuell angezeigt werden, ohne Animationen. Allerdings wird jetzt schon das gleiche Bild mit jeder Umdrehung übertragen.

Unten der Aufbau in Betrieb, mit einem Testbild. Rechts der obere Teil des ZHAW Logos.



Abbildung 31: ZHAW Logo

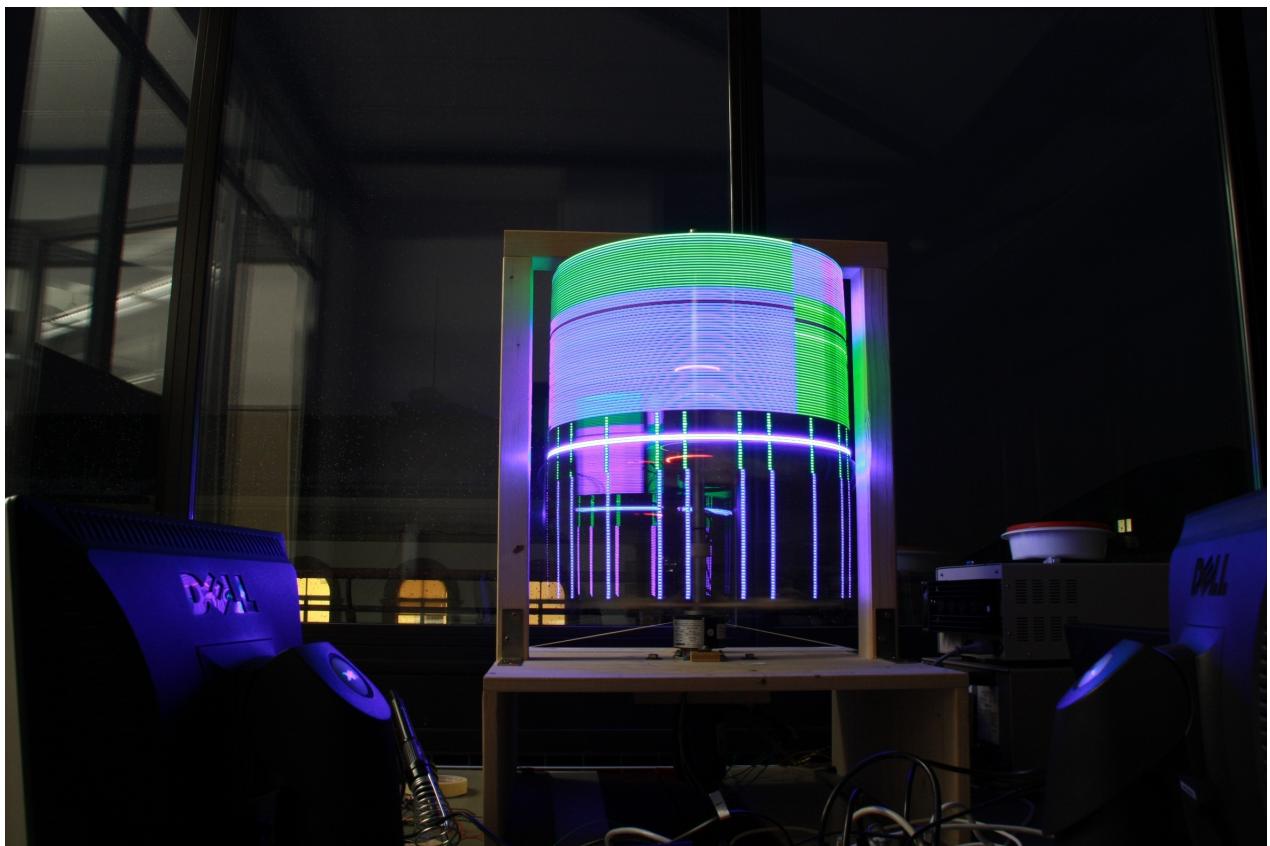


Abbildung 32: Testbild

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Diskussion & Ausblick

Wir haben es zwar geschafft ein Standbild anzuzeigen, jedoch hat die Zeit nicht gereicht um alle Ziele der PA zu erreichen. Wir haben noch nicht alle 4 LED Boards in Betrieb, nur eines. Leider ist es momentan noch nicht möglich Videos oder Bilder über WLAN auf das Board zu streamen.

Die beteiligten Personen und wir haben das Projekt unterschätzt. Es sind viele Probleme und Verzögerungen aufgetreten, mit denen wir nicht gerechnet hätten. So haben uns alleine Lieferfristen teilweise einige Tage gekostet. Gerade bei einer Arbeit über 14 Wochen können Lieferfristen von 3 Wochen einem schon fast das Genick brechen. Das Erstellen des Layouts hat schon so lange gedauert wie wir für das Bauen der Boards gerechnet hatten. Nur das wir bis zu diesem Zeitpunkt noch kein einziges Board gelötet haben. Das Löten war sehr zeitaufwändig, da wir immer wieder an Kurzschlüssen stehen blieben. Weiterhin haben wir das Projekt in seiner Grunddefinition eher erweitert, als verkleinert. So wurden anstatt circa 100 (Projektdefinition), 128 LEDs verbaut. Anstatt einem Flügel wurden zwei gebaut. Vielleicht wären wir mit einer etwas abgespeckteren Spezifikation zeitlich durchgekommen.

Wir vermuten allerdings, dass dieses Projekt für eine PA einfach zu gross ist und vermutlich eher in Kombination PA / BA ausgeführt hätte werden sollen. Dann hätte in einem ersten Teil als PA der elektronische Teil (Layout, Bestücken, erste Programme) der LED-Boards gemacht werden können. Im zweiten Teil dann wäre es dann vor allem um den mechanischen Aufbau und die Software gegangen. Zeitlich wäre es vermutlich immer noch eine grosse Auslastung gewesen.

In Absprache mit Herr Hochreutener haben wir uns darauf geeinigt, dass wir nach Beendigung der PA das Projekt noch fertigstellen, damit dieses an der „Nacht der Technik“ ausgestellt werden kann.

Wir glauben auch, dass dieses Projekt an der „Nacht der Technik“, einige Leute für Informatik und Elektrotechnik faszinieren könnte. Sogar wir selbst haben nach wochenlanger Arbeit irgendwie auch staunen müssen, als wir die ersten Bilder auf unserem Display anzeigen konnten.

[PA] Rotierendes LED Display

Andreas Butti, David Halter

Glossar

Beagle Board

Ein Board mit einem 1GHz ARM und 512 Mbyte RAM, auf einer 4GB SD Karte haben wir Ubuntu installiert. Wir haben das BeagleBoard-xM verwendet.

Link: <http://beagleboard.org/hardware-xM>

Syncboard

Unser Synchronisationsboard, das mit einem AVR-Controller versehen wurde und für die Generierung der Interrupts und weiteren Steuerelementen verantwortlich ist.

STM32

Unser 32-Bit ARM-Cortex-M3 Microcontroller der Firma ST.

ARM

Bezieht sich in diesem Dokument auf einen STM32 Controller.

AVR

Bezieht sich in diesem Dokument auf einen AVR-ATMega168 Microcontroller der Firma Atmel.

PA

Projektarbeit. Eine Arbeit, die im Rahmen der Ausbildung an der ZHAW im zweit-letzten Semester stattfindet und mit 6 Credits bewertet wird.

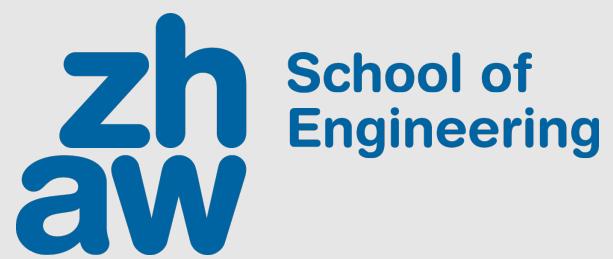
BA

Bachelorarbeit. Die Bachelorarbeit wird an der ZHAW mit 12 Credits bewertet.

[PA] Rotierendes LED Display

Anhang

Andreas Butti, David Halter



Anhang

[PA] Rotierendes LED Display

Anhang

Andreas Butti, David Halter

Zeitplan

Name	Arbeit	2011, Qrt 4		
		Okt	Nov	Dez
Vorbereitung	10 T			
Entwicklungsboard bauen	10 T			
Mechanischer Aufbau	15 T			
LED Board Bauen	20 T			
Entwickeln	35 T			

Leider konnten wir den Zeitplan nicht ganz einhalten. Genau genommen blieben für die Entwicklung nur noch 5 Tage, da sich der Rest aufgrund von vielen verschiedenen Problemen so stark verzögerte, dass eigentlich keine Zeit mehr für die Entwicklung blieb.

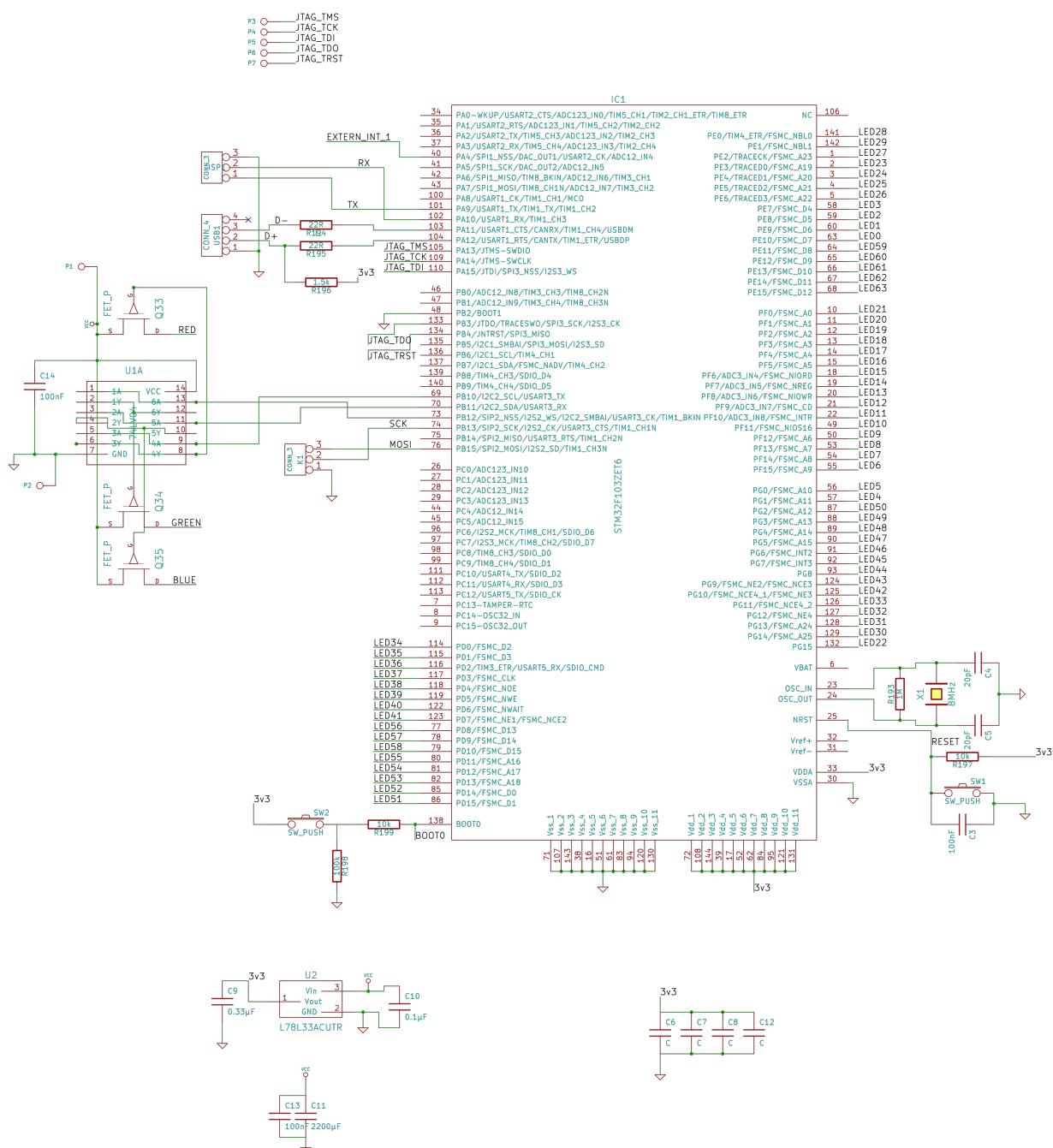
[PA] Rotierendes LED Display

Anhang

Andreas Butti, David Halter

Schema LED Board

Teil 1, Controller

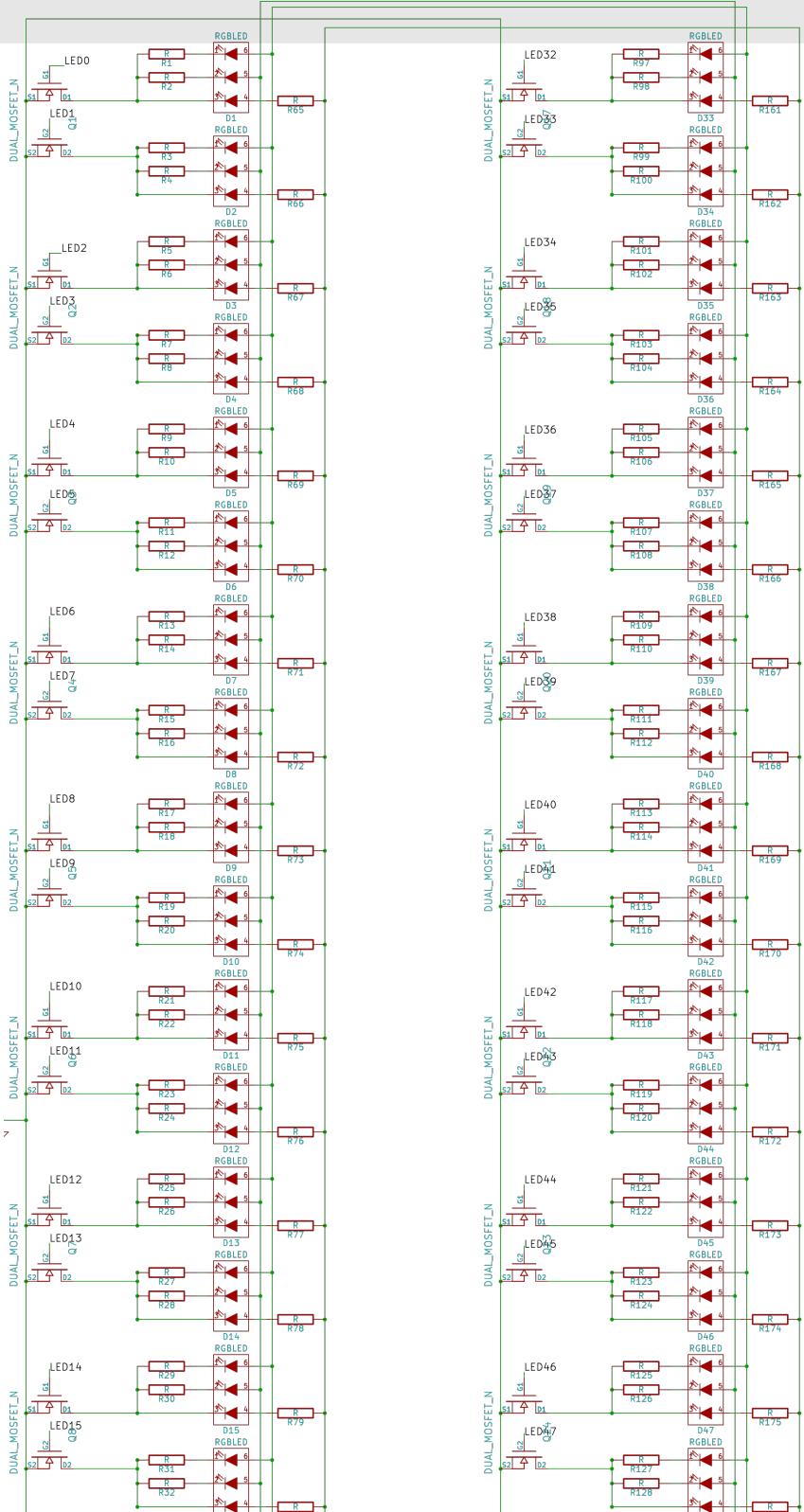


[PA] Rotierendes LED Display

Anhang

Andreas Butti, David Halter

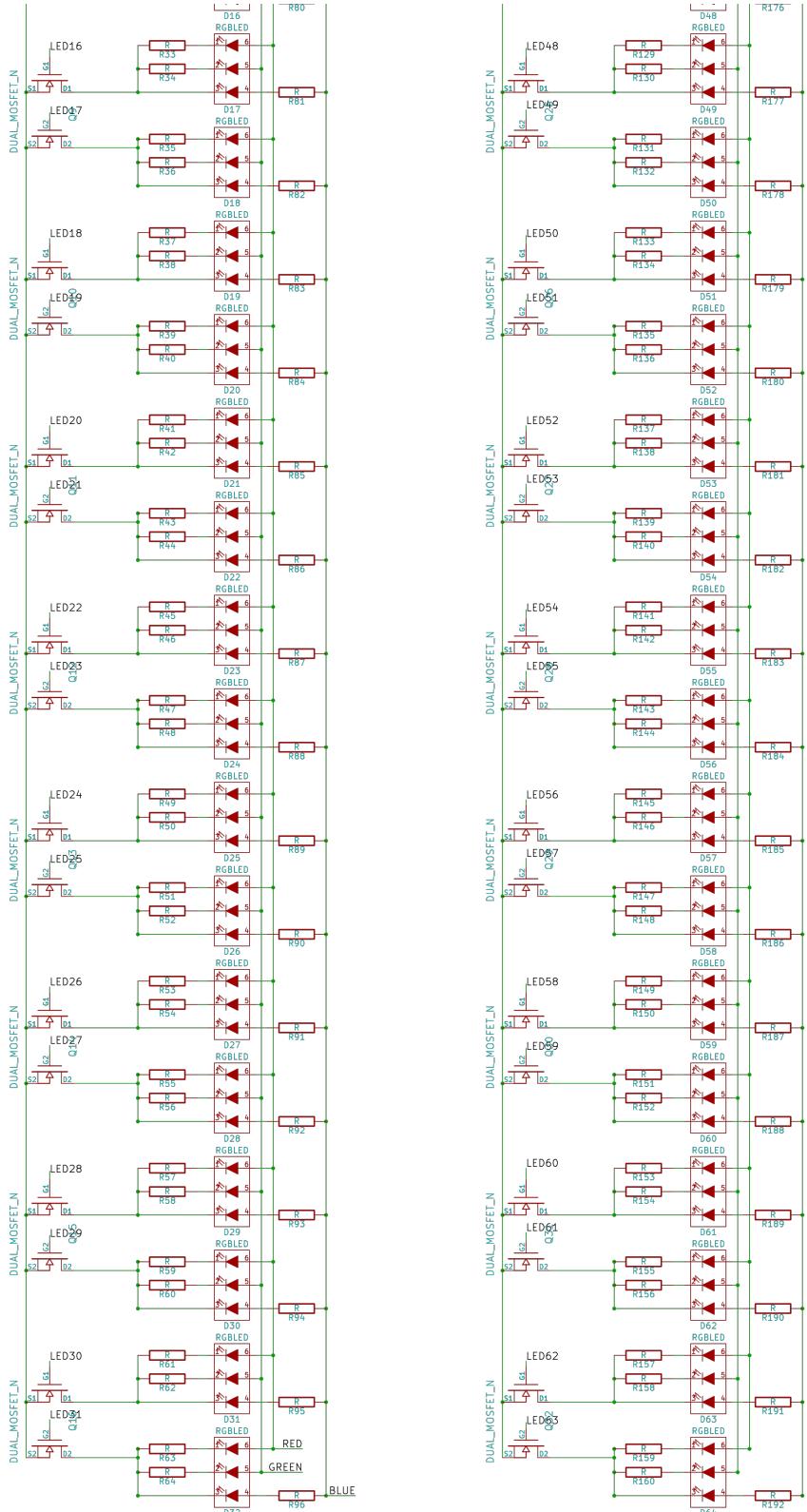
Schema Teil 2, LEDs



[PA] Rotierendes LED Display

Anhang

Andreas Butti, David Halter



[PA] Rotierendes LED Display

Anhang

Andreas Butti, David Halter

CD Ordnerstruktur

Hier noch eine Auflistung was sich wo auf der CD befindet, nach Ordnern gegliedert.

Code

Ordner	Beschreibung
stm32-board	Software für die ARM Boards
python	Python Ansteuerung für das Beagle Board, enthält auch die C Codestücke die aus Python verwendet werden
client	Die Java Software für den Client. Keine Komplettlösung, mehrere Projekte mit Codefragmenten, u.A. den Code zur Bildkonvertierung
syncboard	Die Software für das AVR Board

Datasheet

Datei / Ordner	Beschreibung
ARM-Controller.pdf	STM32 Datasheet
stm32	Die restlichen Unterlagen zum STM32, die wir im Verlauf der PA zusammengesucht haben
FET-Zeilentreiber	Datenblatt des FETs für den Zeilentreiber (P-Type)
Dual-FET-LED.pdf	FET für die LEDs (N-Type)
RGB-LED.pdf	Datenblatt der verwendeten LEDs
Atmega168.pdf	Datenblatt verwendeter AVR

Layout

Ordner	Beschreibung
Led-board	KiCAD Files für das LED-Board, Fehlerkorrigierte Version, incl. SVG Files vom Schema und Gerber Files vom Layout
Syncboard	KiCAD file für das AVR Board. Von Hand vorgenommene Änderungen wurden nicht nachgeführt!