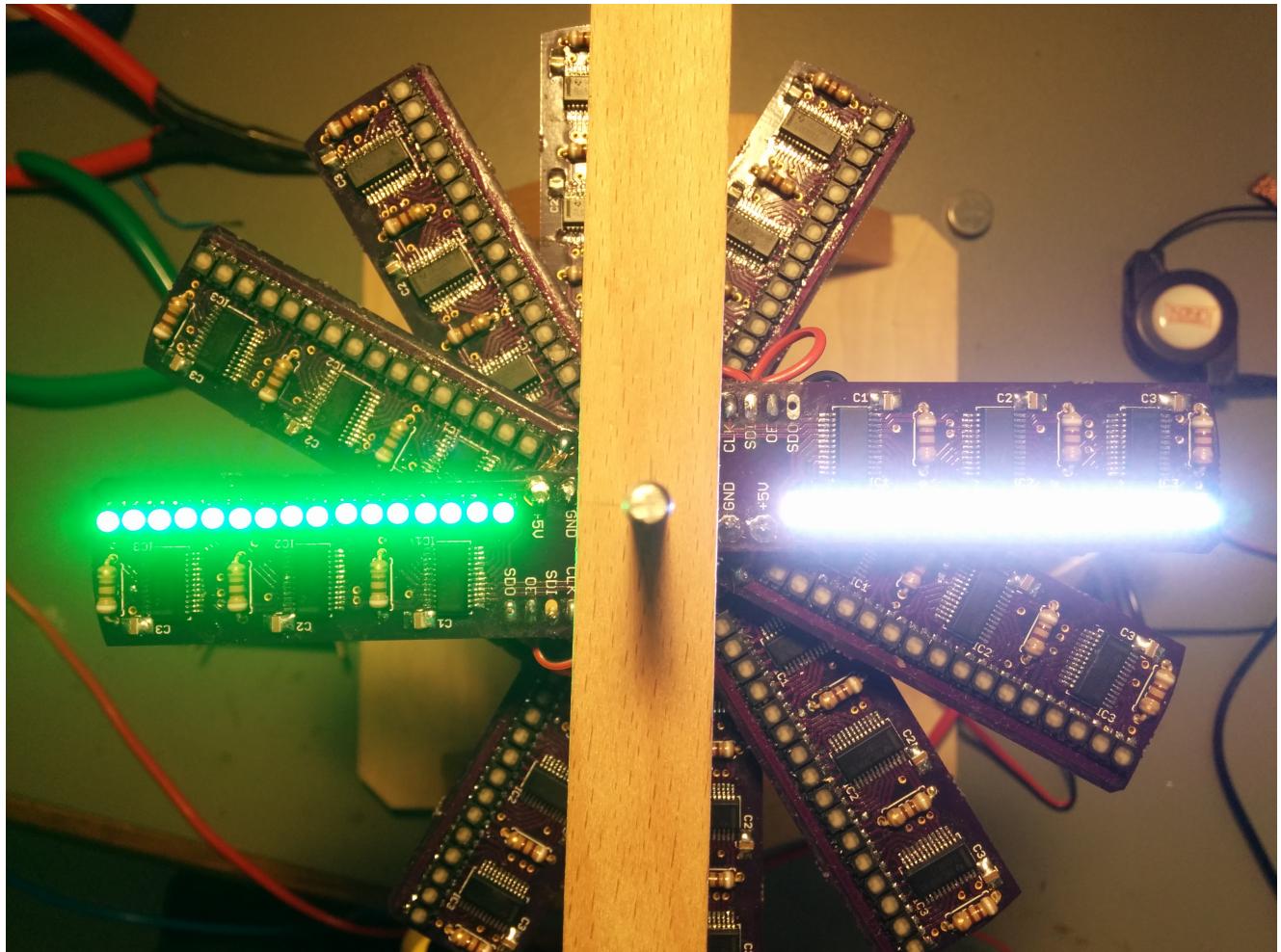


Entwicklung und Bau eines dreidimensionalen Displays



Balduin Dettling, 6d

Maturitätsarbeit 2014

Kantonsschule Wiedikon, Zürich

Betreut von Patrick Spengler

Vorwort

Motivation

Schon länger war mir klar, dass mir eine praktische Arbeit besser liegen würde als eine rein schriftliche. Ich wollte irgendein technologisches Produkt herstellen, da ich schon seit Jahren für Technologie jeglicher Art begeistert bin. Bald habe ich mich auf das Gebiet der Elektronik fokussiert, um dort relativ lange nach Ideen zu suchen. Letzten Endes habe ich mich dann für den Bau eines dreidimensionalen Displays entschieden, das sich das Prinzip der Nachbildungswirkung zunutze macht.

Auf diese Idee kam ich, als ich eines Abends im Internet unterwegs war und auf ein Projekt stieß, das später die Grundlage für das meinige sein sollte. Es handelt sich dabei um eine rotierende Reihe von LEDs, die unabhängig voneinander an gewissen Stellen ein- und an anderen Stellen ausgeschaltet werden können. Diese Vorrichtung zeichnet mit der richtigen Software ein rundes Bild in die Luft. Doch zu der genauen Funktionsweise folgen später mehr Details.

Daran fasziniert mich vor allem das Zusammenspiel zwischen mechanischen und elektronischen Komponenten mit Software. Ebenfalls sehr erstaunlich fand ich, dass es erst zwei Leute gibt, die ein solches 3D-Display bereits gebaut haben. Obwohl die einfachere, zweidimensionale Version unter Bastlern weit verbreitet ist, habe ich in meinen Kreisen praktisch niemanden angetroffen, dem diese Technologie bekannt war.

Dank

Zuerst möchte ich mich bei Patrick Spengler bedanken, der nicht nur die ganze Arbeit betreut hat und etlichen Fehlfunktionen auf den Grund gegangen ist, sondern der mich auch in erster Linie für die Elektronik begeistert hat.

Mein Dank gilt auch Ursula Schamberger, die mir als Lehrerin der AG Werken bei der Konstruktion des Gestells eine grosse Hilfe war. Ebenfalls danke ich Hanspeter Rieder, der mir freundlicherweise Zugang zur Werkstatt gewährt hat.

Authentizitätserklärung

Ich erkläre hiermit, dass die nachfolgende Arbeit komplett von mir selbst verfasst wurde und dass alle externen Informationsquellen korrekt angegeben wurden.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Aktueller Stand der Technik	1
1.2 Zielsetzung	1
2 Theorie und Planung	2
2.1 Funktionsweise	2
2.1.1 Nachbildungswirkung	2
2.1.2 Die Nachbildungswirkung ausnutzen	2
2.2 Spezifikationen und Komponenten	3
2.2.1 Spezifikationen	3
2.2.2 Steuerung	3
2.2.3 Datenübertragung	3
2.2.4 LED-Treiber	4
2.2.5 LEDs	5
2.2.6 Kondensatoren	5
2.2.7 Motor und Antrieb	6
2.2.8 Stromübertragung	6
2.3 Schaltplan	7
2.3.1 Datenschnittstelle	7
2.3.2 LED-Treiber	7
2.3.3 LEDs	8
2.4 PCB-Design	9
2.4.1 Allgemeines Layout	9
2.4.2 Platzsparende Massnahmen	9
2.4.3 Wärmeabfuhr	10
3 Fertigung	11
3.1 Leiterplatten	11
3.1.1 Fabrikation	11
3.1.2 Bestückung	11
3.1.3 Test und Troubleshooting	11
3.2 Gestell	12
3.2.1 Konstruktion	12
3.2.2 Schleifkontakte	12
3.2.3 Controller	13
3.2.4 Leiterplatten	13
4 Programmierung	14
4.1 Verwendete Software	14
4.2 Timing	14
4.2.1 Theorie	14
4.2.2 Implementation	14
4.3 Speicherung und Anzeige der Daten	15
4.3.1 Theorie	15

4.3.2 Implementation	17
4.4 Weiteres	17
5 Inbetriebnahme	19
5.1 Stromversorgung	19
5.2 Weiteres	19
6 Beurteilung	20
6.1 Mögliche Verbesserungen	20
6.1.1 Controller und Datenübertragung	20
6.1.2 Antrieb	20
6.1.3 Schleifkontakte	20
6.1.4 Energieeffizienz	21
6.2 Ausblick	22
6.2.1 Animierte Bilder	22
6.2.2 Interaktive Programme	22
6.2.3 Einsatzgebiete	22
7 Quellenverzeichnis	23
7.1 Bildnachweise	24
8 Anhang	25
8.1 Datenblatt des TLC5927 LED-Treibers	25
8.1.1 Seite 2: Block Diagram	25
8.1.2 Seite 3: Pin Descriptions	26
8.1.3 Seite 15: Adjusting Output Current	27
8.1.4 Seite 27: Mechanical Data	28
8.1.5 Seite 28: Land Pattern Data	29
8.2 Datenblatt der CLVBA-FKA-CAEDH8BBB7A363 RGB-Leuchtdiode	30
8.2.1 Seite 7: Mechanical Dimensions	30
8.3 Datenblatt des US5881 Hall-Sensors	31
8.3.1 Seite 1: Functional Diagram	31
8.4 Quellcode des Programmes	31
8.4.1 _3DPOV.ino	31

1 Einleitung

1.1 Aktueller Stand der Technik

Im Internet gibt es bereits zahllose Beispiele von zweidimensionalen Displays, die sich die Nachbildungswirkung zunutze machen [1, 2, 3]. Es hat sich der Name „POV-Display“ eingebürgert, wobei „POV“ für „Persistence of Vision“ steht. Ein anderer weit verbreiteter Name ist „Propeller Clock“, da die runde Form dieser Monitore sich bestens für die Darstellung einer analogen Uhr eignet.

Meines Wissens haben jedoch erst zwei Personen eine funktionstüchtige dreidimensionale Version gebaut und diese im Internet publiziert [4, 5]. Keine der beiden wird industriell gefertigt, stattdessen stellen sie – wie die meisten zweidimensionalen Versionen – Freizeitprojekte von Bastlern oder Ingenieuren dar.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, einen funktionierenden Prototypen eines dreidimensionalen Displays herzustellen und den Prozess zu dokumentieren. Dazu gehören die Erläuterung der Funktionsweise des Gerätes und die daraus resultierenden technischen Anforderungen. Anschliessend sind die Spezifikationen grob festzulegen und entsprechend die Bauteile auszuwählen. Diese werden plangemäss zusammengebaut und mit einem passenden Programm zum Leben erweckt.

2 Theorie und Planung

2.1 Funktionsweise

2.1.1 Nachbildungswirkung

Das menschliche Auge ist langsam. Da wir uns ein ganzes Leben lang an diese Langsamkeit gewöhnt haben, stört sie uns nicht und ist meistens auch nicht bemerkbar. Doch es gibt einige Situationen, in denen die Nachbildungswirkung klar sichtbar wird. Wenn das Auge in einer düsteren Umgebung einen hellen Punkt wahrnimmt, der im Verhältnis zum Augapfel eine hohe Winkelgeschwindigkeit hat, nehmen wir diesen nicht einfach als Punkt wahr, sondern sehen zusätzlich eine Linie, die er hinter sich her zieht. Dies ist zum Beispiel sichtbar, wenn ein Auto in der Dämmerung schnell vorbeifährt oder wenn man den Tanz eines Glühwürmchens zu Gesicht bekommt.

Dieses Phänomen hat den Grund, dass ein eingehender Lichtreiz auf der Netzhaut nicht sofort wieder verschwinden kann, sondern erst nach einem kurzen Augenblick nachlässt. Laut dem englischen Wikipedia-Eintrag „Persistence of Vision“ beträgt dieser Zeitraum etwa 1/25 einer Sekunde [6]. Dieser Effekt, den man auch „Nachbildungswirkung“ oder eben „Persistence of Vision“ nennt, ist essentiell für den erfolgreichen Betrieb eines POV-Displays [7].

2.1.2 Die Nachbildungswirkung ausnutzen

Normalerweise ist die Nachbildungswirkung höchstens ein interessanter, aber manchmal auch störender Effekt. Doch indem man die sich bewegende Lichtquelle entsprechend kontrolliert, lässt sich diese Schwäche des menschlichen Auges nützlich machen.

Wenn nämlich diese Lichtquelle schnell genug rotiert, nimmt das Auge nicht mehr einen Punkt wahr, der eine Kreisform beschreibt. Stattdessen sieht man einen leuchtenden Kreis, der sich nicht mehr bewegt. Diese Lichtquelle kann bei jeder Umdrehung an den gleichen Stellen eingeschaltet werden, sodass verschiedene Segmente des Kreises unabhängig voneinander illuminiert werden.

Mit einem einzelnen Lichtpunkt oder mit einem 0-dimensionalen Display lässt sich also durch eine schnelle Kreisbewegung ein eindimensionales Bild darstellen. Mit mehreren solchen Lichtquellen nebeneinander kann man ein eindimensionales Display rotieren lassen. Insgesamt resultiert dies in einem zweidimensionalen Bild. Diese Version entspricht dem bereits vielfach realisierten POV-Display.

Doch es geht noch einen Schritt weiter: Mehrere dieser Pixelreihen können übereinander angeordnet werden, sodass ein zweidimensionales Display entsteht. Wenn dieses wiederum schnell genug durch die dritte Dimension rotiert, kann es ein dreidimensionales Bild darstellen.

2.2 Spezifikationen und Komponenten

2.2.1 Spezifikationen

- 16 dreifarbig Pixel parallel zum Radius auf jedem Arm
- 10 vertikale Pixel
- 100 Pixel entlang dem Radius (kann auch später in der Software geändert werden)
- 30 Umdrehungen/Sekunde oder mehr, da dies die typische Framerate für Animationen ist und da eine Periode von 1/30 einer Sekunde ein wenig kürzer ist als die nötige Periode von 1/25 einer Sekunde

2.2.2 Steuerung

Um all die LEDs zu steuern, habe ich ein Teensy 3.1 Mikrocontroller-Entwicklungsboard gekauft [9]. Es ist mit der Arduino IDE kompatibel und somit relativ einfach zu programmieren. Auf einer kleinen Fläche – ideal für die schnelle Rotation – werden 256 kB an Flash-Speicher, ein 32-Bit Prozessor mit 96 MHz Taktrate und 34 I/O-Pins untergebracht

Damit das Bild immer die korrekte Ausrichtung hat und die Geschwindigkeit nicht genau einem festen Wert entsprechen muss, empfängt der Controller bei jeder Umdrehung ein Signal von einem Hall-Sensor, der an einem stationären Magneten vorbeikommt. An diesem Punkt wird gemessen, wie lange die letzte Umdrehung dauerte. Davon ausgehend wird ausgerechnet, wie schnell die Erneuerungen während der nächsten Umdrehung erfolgen müssen, damit ein stehendes Bild entsteht.

2.2.3 Datenübertragung

Bei voller 8-Bit-Farbtiefe sowie 16 dreifarbigem LEDs pro Arm, 10 Armen und 100 Erneuerungen pro Umdrehung würde der nötige Datenstrom das folgende Ausmass annehmen:

$$\begin{aligned} & 8 \text{ Bit/Farbe} * 3 \text{ Farben/Pixel} * 16 \text{ Pixel/Arm} * 10 \text{ Arme/Erneuerung} * 100 \\ & \text{Erneuerungen/Umdrehung} * 30 \text{ Umdrehungen/s} = \mathbf{11'520'000 \text{ Bit/s}} \end{aligned}$$

11.52 Megabit pro Sekunde ist ein beträchtlicher Wert. Der Teensy 3.1 hätte bei 96 MHz nur 8.333 Taktzyklen zur Verfügung, um ein Datenbit an die LED-Treiber zu schicken. Das wäre zwar noch machbar, doch es muss auch noch genügend Zeit für die Timing-Berechnungen übrig bleiben. Ein Bild mit 8-Bit Farbtiefe hätte eine Grösse von 48 kB, womit bereits ein erheblicher Teil der 256 verfügbaren Kilobyte besetzt wäre.

Ein weiterer Grund, der gegen die Implementation von 8-Bit-Farbtiefe spricht, ist die kleine Auswahl an passenden LED-Treibern. Die meisten Modelle weisen eine relativ langsame PWM-Frequenz von einigen hundert Hz oder wenigen kHz auf, was bei einer schnellen Drehung in einem unschönen Bild resultieren würde. Auch die vorgefertigten LED-Streifen

haben den selben Nachteil. Die wenigen Treiber, die eine genügend hohe PWM-Frequenz aufweisen, werden mit I²C und nicht mit SPI angesteuert.

Da die I²C-Linien über einen Pullup-Widerstand auf 5V gehalten werden, dauert es relativ lange, bis deren Logikpegel gewechselt hat. Die I²C-Interfaces des Teensy sind deshalb selbst mit einer optimierten Library (i2c_t3.h) auf 2.4 MHz limitiert [10]. Im Gegensatz dazu kann die SPI-Library bei jedem zweiten Taktzyklus ein Bit schicken, womit die maximale Bus-Geschwindigkeit ganze 48 MHz beträgt.

Ein weiterer Nachteil von I²C ist die hohe Menge an Verwaltungsdaten. Vor jeder Übertragung muss die Adresse des angesteuerten Chips übertragen werden, und nach jedem gesendeten Byte verlangt der Master vom Slave ein ACK-Bit, damit er weiß, dass die Daten angekommen sind [11]. Diese Steuersignale rauben den Nutzdaten wertvolle Zeit und stellen einen Overhead dar, der bei der Verwendung von SPI wegfallen würde.

Um diesen Problemen auszuweichen, reduzierte ich die gewünschte Farbtiefe von einem Byte auf ein Bit. So können pro Pixel noch $2^3 = 8$ verschiedene Farben dargestellt werden. Dafür sind acht mal weniger Daten nötig, also 1'440'000 Bit pro Sekunde. Es bleiben für ein Bit ganze 66.66... Taktzyklen. Somit bleibt sogar noch Luft nach oben frei; es könnte die Anzahl an Erneuerungen pro Umdrehung oder die Drehzahl erhöht werden.

2.2.4 LED-Treiber

Nach längerer Suche bin ich auf den TLC5927 von Texas Instruments gestossen [12, 13]. Dies ist ein LED-Treiberchip, der im Prinzip einem Schieberegister mit einigen Zusatzfunktionen entspricht. Wie jedes Schieberegister besitzt er einen seriellen Dateneingang (SDI, Serial Data Input), einen seriellen Datenausgang (SDO, Serial Data Output) und einen Clock-Pin (CLK). Dazu kommt, wie bei vielen anderen Schieberegistern auch, ein Latch-Pin (LE). Dank diesem ist es möglich, die Daten erst anzuzeigen, wenn die Datenübertragung vorüber ist.

Darüber hinaus verfügt der Chip über einen „Output Enable“-Pin (OE). Wenn dieser auf ein logisches 1 gesetzt wird, funktioniert alles normal, wenn man ihn jedoch mit 0V verbindet, werden alle Outputs ausgeschaltet. Die betreffende Seite in Datenblatt, die die Funktionen der einzelnen Pins erläutert und von der die Abbildung 1 stammt, befindet sich im Anhang.

GND	1	24	VDD
SDI	2	23	R-EXT
CLK	3	22	SDO
LE(ED1)	4	21	OE(ED2)
OUT0	5	20	OUT15
OUT1	6	19	OUT14
OUT2	7	18	OUT13
OUT3	8	17	OUT12
OUT4	9	16	OUT11
OUT5	10	15	OUT10
OUT6	11	14	OUT9
OUT7	12	13	OUT8

Abbildung 1: Pinbelegung des TLC5927 LED-Treibers

Das meiner Meinung nach wichtigste Feature des Chips ist die Konstantstromregelung. Bei einem normalen Schieberegister bräuchte jede LED ihren Vorwiderstand. Doch beim TLC5927 wird der Strom intern limitiert. Eingestellt wird dieser Strom über einen einzigen externen Widerstand, auf den über den R_{EXT}-Pin 1.25 V angewendet werden. Der LED-Treiber misst den Strom, der durch R_{EXT} fließt, multipliziert ihn mit 15 und nimmt ihn als Referenzwert für die Ausgangsstrom.

Im Gegensatz zu den meisten simplen Schieberegistern sind die Outputs des TLC5927 keine Stromquellen, sondern Stromsenken. Dies wird bei der Auswahl der LEDs von Bedeutung sein: Damit die drei Farben einzeln angesteuert werden kann, müssen die jeweiligen Kathoden getrennt sein.

2.2.5 LEDs

Als Lichtquelle kommen nur LEDs in Frage, da Glühlampen zahlreiche Nachteile aufweisen. Sie brauchen nämlich eine vergleichsweise lange Zeit, um ein- und auszuschalten, was deren Verwendung bereits verunmöglicht. Ein weiterer Nachteil ist die hohe Wärmeproduktion und somit Ineffizienz. Aufgrund dieser Wärme können sie auch nicht beliebig klein gebaut werden, da eine kleine Verpackung sehr schnell überhitzt würde. Letztendlich haben LEDs auch noch den Vorteil, dass es sie in Verpackungen mit mehreren LEDs unterschiedlicher Farbe gibt.

Die Bauteile habe ich alle auf digikey.com bestellt. Diese Webseite beinhaltet ein exzellentes Tool, das den Kunden ermöglicht, seine Auswahl mit den verschiedensten Kriterien einzuschränken [8]. Meine Anforderungen für die LEDs waren die folgenden:

- Farben: Rot, Grün, Blau für farbige Anzeige
- Strom: 20mA oder mehr. Da die LED nur einen kurzen Moment am selben Punkt ist, wird sie essentiell durch PWM verdunkelt.
- Montagetyp: Oberflächenmontage, damit die LEDs klein und leicht genug sind und auf der selben Seite der Leiterplatte wie die LED-Driver montiert werden können
- Gemeinsame Anode und getrennte Kathoden, da die Outputs der LED-Treiber Stromsenken darstellen
- Grösse: Höchstens 5 mm * 5 mm, aber immer noch gross genug, um die LED gut von Hand löten zu können
- Diffundierte Linse, damit die Farben sich gut mischen, obwohl der Betrachter direkt in die LED schaut und nicht auf eine beleuchtete Fläche

Diese Auswahl schränkte die ganze Kategorie „LED-Anzeige – Diskret“, die insgesamt 18'940 Produkte umfasst, auf 130 Produkte ein. Von diesen wählte ich das Produkt, von dem 160 Stück am günstigsten waren. So kam ich auf eine LED mit dem wohlklingenden Namen CLVBA-FKA-CAEDH8BBB7A363. Die Verpackung von 3.2mm * 2.8mm besitzt 4 Pins, nämlich eine gemeinsame Anode und drei Kathoden, davon je eine für Rot, Grün und Blau.

2.2.6 Kondensatoren

Ein digitaler Schaltkreis wie der LED-Treiber kann seinen Stromverbrauch innerhalb kürzester Zeit ändern. Da aber die Verbindungen zur Stromversorgung relativ lange sind, besitzen diese eine gewisse Induktivität. Wenn also der integrierte Schaltkreis plötzlich mehr Strom braucht, verstreicht ein Augenblick, bis dieser Strom auch bereitgestellt werden kann. In solch einer Situation kann es sein, dass der Schaltkreis sich fehlerhaft verhält und die Daten nicht korrekt übertragen oder verarbeitet werden.

Um dem vorzubeugen, muss ein kleiner Kondensator ($\sim 1 \mu\text{F}$) zwischen den V_{DD} - und GND-Pins der LED-Treiber eingebaut werden. Dieser lädt sich beim Einschalten der

Stromversorgung auf und speichert eine kleine Energiemenge in einem elektrischen Feld. Wenn nun der Stromverbrauch des Chips plötzlich ansteigt, kann der nötige Strom kurzfristig aus dem Kondensator bezogen werden, bis das Netzteil reagiert. Auch wenn sich Störsignale in die Versorgungsspannung einschleichen, werden diese vom Kondensator grösstenteils absorbiert, sodass sie in den Chips keinen Schaden anrichten können.

2.2.7 Motor und Antrieb

Um das ganze Gebilde rotieren zu lassen, braucht es natürlich einen Motor. Ein passendes Exemplar habe ich aus einem alten ferngesteuerten Motorboot gerettet. Es handelt sich um einen relativ grossen Bürstenmotor. Dass dieser auf hohe Leistung ausgelegt ist, ist am eingebauten Lüfter zu erkennen. Im Motorboot wurde er zusätzlich sogar mit Frischwasser gekühlt.

In meiner Arbeit wird dieser Motor wohl nur einen Bruchteil seiner Maximalleistung umsetzen. Doch das ist kein Problem; ganz im Gegenteil: Dank dem hohen Drehmoment, das der Motor liefert, ist es möglich, ihn ohne Untersetzung zu verwenden. So kann er mit einer tieferen Drehzahl und mit weniger Lärm und Verschleiss operieren. Es kam später jedoch heraus, dass er zu viel Strom für ein normales Computernetzteil brauchte, sodass ich auf das Labornetzteil der Schule ausweichen musste.

2.2.8 Stromübertragung

Um den Strom vom stationären Netzteil zur rotierenden Last zu bringen, müssen Schleifkontakte verwendet werden. Diese werde ich relativ simpel gestalten, und zwar bestehend aus zwei runden Kupferplatten auf dem rotierenden Teil und zwei langen Kupferplatten, die mit ihrer eigenen Federkraft auf die Kupferplatten gedrückt werden.

Ein einzelne LED braucht zwar keine bedeutende Strommenge, doch bei 160 dreifarbigem LEDs beträgt der maximale Strom:

$$20 \text{ mA/LED} * 480 \text{ LEDs} = 9.6 \text{ A}$$

Ausgenommen ist natürlich der Strom für den Controller und die Logik in den LED-Drivern, doch dieser ist vernachlässigbar. Ebenfalls ausgenommen ist der Strom für den Motor: Da der Motor stationär ist, muss er nicht über Schleifkontakte versorgt werden.

Abgesehen davon ist dies der Strom, den die LEDs brauchen, wenn sie alle eine weisse Farbe annehmen. Der Stromverbrauch wird in der Praxis nicht annähernd so hoch sein, aber dennoch ist es eine gute Idee, alle Komponenten nach diesem theoretischen Maximalwert auszurichten.

Da solch ein Mechanismus den Strom keineswegs zuverlässig leitet, habe ich einen relativ grossen ($2200 \mu\text{F}$) elektrolytischen Kondensator vorgesehen, um die Spannungsspitzen der Bürsten zu glätten. Auch wenn die Bürsten an einer bestimmten Stelle nicht korrekt leiten, kann der Kondensator einspringen und für einen kurzen Moment das Netzteil ersetzen.

2.3 Schaltplan

Um den Schaltkreis sowie das Layout der Leiterplatte zu erstellen, benutzte ich das CAD-Programm Eagle [14]. Dessen kostenlose Version ist zwar auf zweischichtige Leiterplatten bis zur Grösse 10 cm * 8 cm limitiert, doch das reicht für meine Zwecke aus.

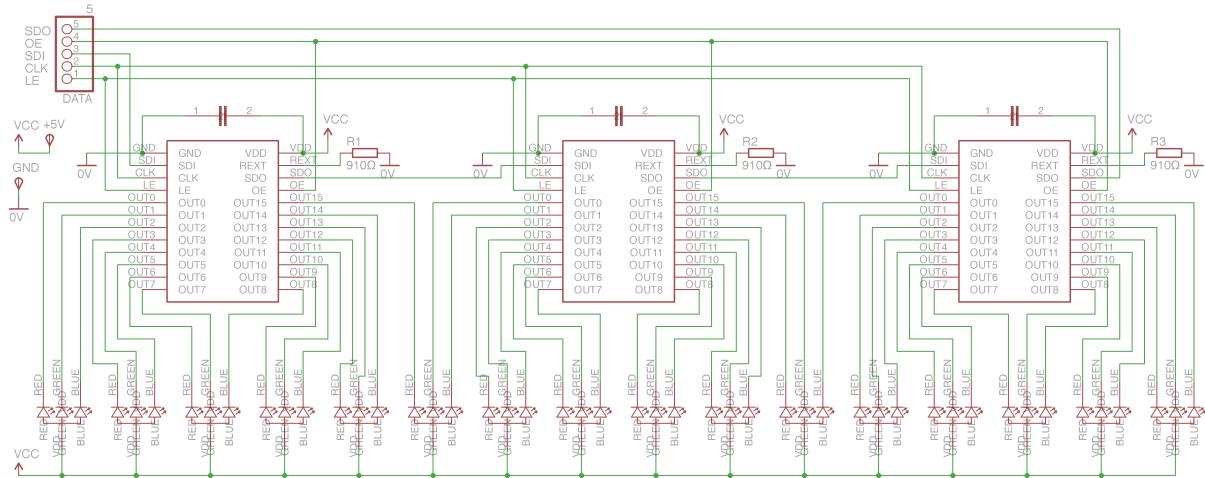


Abbildung 2: Der fertiggestellte Schaltplan. Grün = Verbindungen, Rot = Symbole.

In der Abbildung 2 ist die finale Version des Schaltplans ersichtlich. Der Einfachheit und der Übersicht zuliebe entsprechen die Pinbelegungen, Ausrichtungen und Positionen der Bauteile in etwa dem späteren PCB-Layout.

Die Fussabdrücke und Symbole der Komponenten musste ich mit Ausnahme der Kondensatoren selbst konfigurieren, da ich keine ähnlichen Dateien online fand. Dazu entnahm ich die Pinout-Diagramme der LED-Driver und LEDs sowie deren Dimensionen dem Datenblatt, dessen relevante Passagen sich im Anhang befinden.

2.3.1 Datenschnittstelle

Zur Datenübertragung habe ich eine Reihe von fünf Pins verwendet, die später auf der Leiterplatte die Datenschnittstelle darstellen. Sie enthält die fünf Verbindungen zu SDO, OE, SDI, CLK und LE. Die Reihenfolge dieser Signale legte ich erst später fest, damit das PCB-Layout möglichst simpel wird.

2.3.2 LED-Treiber

Die LED-Treiber sind in einer sogenannten „Daisy-Chain“-Konfiguration angeordnet, bei welcher der Datenausgang (SDO) des Chips n mit dem Dateneingang (SDI) des Chips n+1 verbunden ist. Den Datenausgang des letzten Chips habe ich später an den Dateneingang des ersten Chips des nächsten Boards angeschlossen. Auf diese Weise werden alle 30 LED-Driver zusammen in Serie geschaltet.

Da jeder 16 Outputs hat, braucht er bei einem Bit Farbtiefe 16 Bit für eine volle Aktualisierung der Ausgangswerte. 30 Chips multipliziert mit 16 Bit/Chip sind 480 Bit oder 60 Byte. Diese Datenmenge wird bei jeder Aktualisierung der Outputs gesendet.

Alle anderen Signale, nämlich OE, CLK und LE, sind nicht nur mit dem ersten, sondern mit allen Chips gleichzeitig verbunden. Dies sind alles Signale, die anstatt der Datenübertragung der Steuerung der Chips dienen. Als solche müssen sie auf alle LED-Driver gleichzeitig einwirken.

Mehrere Schieberegister (oder TLC5927 LED-Treiber) in einer Daisychain-Konfiguration haben den selben Schaltkreis wie ein einzelnes, grosses Schieberegister. Wenn alle Chips zusammen als einzelner Schaltkreis angesehen werden, wird die Datenübertragung sehr viel leichter. Die Software verhält sich bei 30 kleinen Schieberegistern identisch wie bei einem gigantischen; das Programm muss nicht einmal wissen, wo ein Chip zu Ende ist und wo der nächste beginnt, um die Daten korrekt zu übertragen.

Der externe Widerstand R_{EXT} , der sich an jedem Chip befindet, legt den Ausgangsstrom fest. Folgende Gleichung, die auf der Seite 15 des Datenblattes zu finden ist, macht den Ausgangsstrom und R_{EXT} voneinander abhängig:

$$I_{OUT, Target} = (1.25 \text{ V}/R_{EXT}) * 15$$

Wenn man den Ausgangsstrom $I_{OUT, Target} = 20 \text{ mA}$ setzt und nach R_{EXT} auflöst, offenbart sich die folgende Gleichung:

$$R_{EXT} = (15 * 1.25 \text{ V})/20 \text{ mA} = 937.5 \Omega$$

Der nächstkleinere Standardwert der E24-Reihe liegt bei 910Ω . Bei diesem Wert ist der Ausgangsstrom 20.6 mA , also noch nahe genug bei den gewünschten 20 mA . Von diesem Wert kaufte ich 40 Stück in durchlochmontierter Version, da ich zu diesem Zeitpunkt noch im Sinn hatte, das PCB selbst zu ätzen.

Natürlich sind auch noch die Pins für die Stromversorgung mit den entsprechenden Symbolen versehen. Links im Schaltplan sind diese Signale mit Pads verbunden, an die man später auf dem Board die Versorgungskabel anlöten kann.

2.3.3 LEDs

Wie schon erwähnt haben die LEDs eine gemeinsame Anode. Die Kathoden für jede Farbe sind in der Reihenfolge Rot – Grün – Blau mit den Ausgängen der Chips verbunden, und die Anode wird mit $+5V$ versorgt. Vorwiderstände zur Begrenzung des Stroms braucht es nicht, da der LED-Treiber deren Aufgabe mit seiner Konstantstromregelung übernimmt.

2.4 PCB-Design

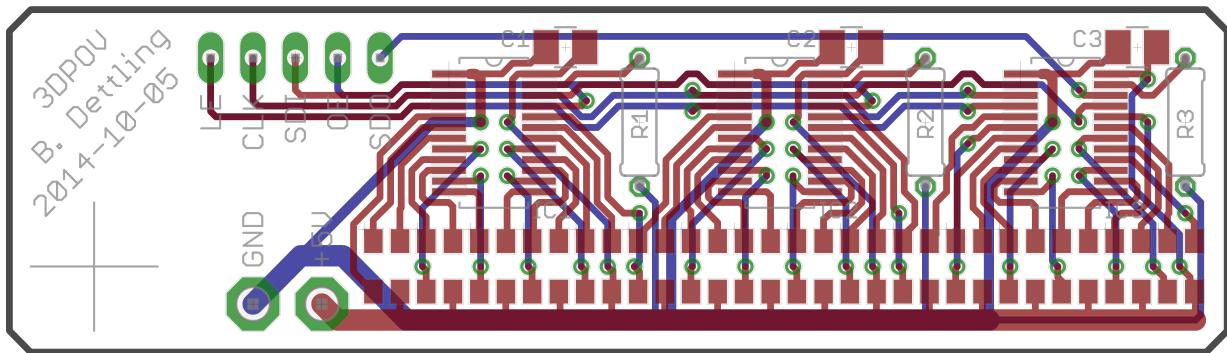


Abbildung 3: Das Layout der Leiterplatte in Eagle. Rot = obere Kupferschicht, Blau = untere Kupferschicht, Grün = Durchkontaktierung, Grau = Beschriftung.

2.4.1 Allgemeines Layout

Mein oberstes Ziel beim Design der Leiterplatte war es, den Schaltkreis in einem sinnvollen, aufgeräumten Layout unterzubringen. Durch die Pinbelegung der LED-Driver war aber die Anordnung der Komponenten schon so gut wie vorbestimmt. Es befinden sich nämlich alle Ausgänge auf den unteren acht Pins des Chips. Darüber folgen die Pins zur Datenübertragung und Kontrolle, und zuoberst finden sich noch die beiden Versorgungspins. Damit war klar, dass die LEDs auf die untere Seite kommen. Die Kondensatoren platzierte ich möglichst nahe bei den zugehörigen Pins, +5V und GND. Für die Datenspuren blieb bei den zugehörigen Pins noch eine passende Lücke.

Die Versorgungsbahnen platzierte ich unter den LEDs, und zwar aus zwei Gründen. Der erste ist, dass jede der 16 LEDs auf ihrer unteren Seite eine gemeinsame Anode hat. Wenn also ich die Bahn oberhalb der LEDs hätte platzieren wollen, wäre ich trotzdem gezwungen gewesen, diese 16 Mal auf die andere Seite der LED-Reihe zu führen. Die ICs brauchen hingegen nur je eine Verbindung zu 5 Volt und eine zu GND, also sechs Bahnen insgesamt.

Der zweite Grund ist die elektromagnetische Interferenz. Wenn alle LEDs eingeschalten sind, fliessen auf einem Board 0.96A. Dieser Strom und sein Magnetfeld ändern sich bis zu 3000 Mal pro Sekunde, sodass in den Chips störende oder gefährliche Spannungen induziert werden können. Um dies zu verhindern, ist es ratsam, den Versorgungsstrom möglichst weit weg von der sensiblen Logik fliessen zu lassen. Zusätzlich platzierte ich die beiden Leitungen direkt übereinander, sodass sich die Magnetfelder der beiden entgegengesetzt fliessenden Ströme weitgehend auflösen.

2.4.2 Platzsparende Massnahmen

Damit die Produktion möglichst günstig und die rotierende Masse minimal wird, strebte ich die kleinstmöglichen Dimensionen an. Letzten Endes brachte ich diese bis auf 73 mm * 21 mm herunter.

Um dies zu erreichen, aber auch um grosse Lücken zwischen den Pixeln zu vermeiden, platzierte ich die LEDs so nahe aneinander, wie es nur ging. Im Datenblatt ist eine Breite von

2.8 ± 0.2 mm angegeben, also beschloss ich, jeder LED mindestens 3mm oder 118.1 mil Platz zu lassen. Auf dem Gitter von 12.5 mil war die nächst grössere Distanz 125 mil, also 3.175 mm.

Die Pins zu den Kathoden der roten LED befinden sich jeweils auf der „unteren“ Seite der Verpackung. Wenn die Abstände zwischen diesen grösser wären, hätte ich die Kupferspur zu den roten Kathoden einfach zwischen den LEDs hindurchführen können. Doch dazu sind sie zu nahe beieinander; ich musste auf die zweite Kupferschicht ausweichen und die Verbindung unter der LED verlegen. Die Durchkontaktierungen hatten direkt vor den LEDs keinen Platz, also brachte ich sie auf der freien Fläche unter den LED-Treibern unter.

Die Taktfrequenz der Treiber ist mit maximal 30 MHz nicht allzu hoch, und ich werde wahrscheinlich nur einen Bruchteil dieses Maximalwertes nutzen müssen. Also nahm ich ein wenig Induktivität zwischen den Kondensatoren und den ICs in Kauf, damit ich die Kondensatoren seitlich von den Chips platzieren und ein wenig Platz sparen konnte.

2.4.3 Wärmeabfuhr

Ein positiver Nebeneffekt der sechs oder sieben Durchkontaktierungen unter den Chips ist deren thermische Leitfähigkeit. So kann ein wenig zusätzliche Abwärme vom Chip in die Leiterplatte verteilt werden. Die maximale Leistung des Chips – wenn alle Ausgänge aktiv sind – beträgt:

$$P_{MAX} = (5V - V_F) * 20 \text{ mA/Ausgang} * 16 \text{ Ausgänge} = 757.3 \text{ mW}$$

V_F ist je nach Farbe verschieden, also habe ich in die Gleichung einfach den Durchschnittswert aller drei Farben, nämlich 2.633 V eingesetzt. Die wahren Werte sind 1.9 V für Rot und 3 V für Grün und Blau.

757 mW ist zwar ein nicht zu unterschätzender Wert, und Tests haben gezeigt, dass die Boards bei voller Leistung tatsächlich sehr warm werden. Doch in der Praxis wird diese Situation kaum erreicht; die LEDs werden nur während einem Bruchteil der Umdrehung eingeschaltet. Zudem ist bei der schnellen Bewegung die Luftzirkulation exzellent, sodass die produzierte Abärme ständig abgegeben werden kann.

3 Fertigung

3.1 Leiterplatten

3.1.1 Fabrikation

Es gibt eine Vielzahl von Anbietern, die Leiterplatten nach Mass produzieren. Als einer der günstigsten Anbieter, die kleine Serien in kurzer Zeit liefern und eine gute Fabrikationsqualität aufweisen, stellte sich OSHPark heraus [15]. Es können nur Vielfache von drei Boards bestellt werden, also bestellte ich 12 anstatt 10. Am 5. Oktober 2014 gab ich die Bestellung auf und bezahlte dafür 47 US-Dollar, und am 15. Oktober wurden die Boards fertiggestellt und verschickt. Bei mir sind sie am 25. Oktober angekommen.

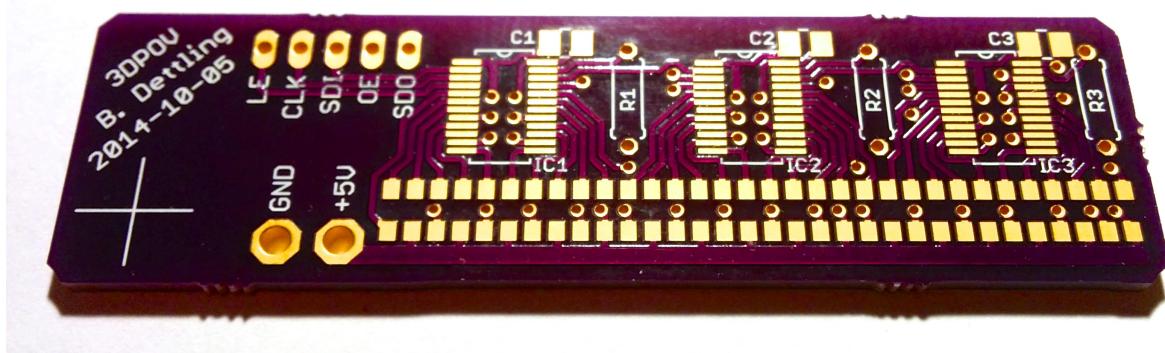


Abbildung 4: Eine fertig fabrizierte Leiterplatte

3.1.2 Bestückung

Sobald ich die Leiterplatten bekam, ging ich daran, die Komponenten darauf zu löten. Ich begann mit den LEDs, da ich sonst in der engen Lücke zwischen dem Treiber und den LEDs hätte löten müssen. Für das erste Board voll LEDs brauchte ich volle zwei Stunden, doch mit zunehmender Übung reduzierte sich die Zeit, in der ich eine Reihe von 16 LEDs verlötete auf 30 oder 25 Minuten.

Die Chips waren wesentlich einfacher zu löten als die LEDs. Bei diesen konnte ich einfach alle Pins mit Lötzinn überschwemmen, um dann die Brücken mit Entlötlitze zu entfernen. So werden auf einen Schlag zwölf Kontakte gelötet. Auch die Kondensatoren und Widerstände stellten kein Problem dar; dank deren tiefer Anzahl dauerte es nicht lange, bis diese angebracht waren.

3.1.3 Test und Troubleshooting

Um die fertigen Boards zu testen, erstellte ich ein kleines Programm, welches die LEDs abwechselnd in den Farben Rot, Blau, Grün und Weiss aufleuchten lässt. Leider funktionierte es nicht beim ersten Versuch. Also prüfte ich mit dem Oszilloskop, ob auch wirklich Daten aus dem Teensy geschickt wurden. Dies war sehr wohl der Fall; die SPI-Library erfüllte ihren Dienst tadellos.

Schlussendlich stellten sich die schlechten Kontakte als Grund des Problems heraus. Da sie nicht gelötet waren, hatten sie eine sehr kleine Kontaktfläche und eine zu hohe parasitische Kapazität. So hatten die Signale, die mit einer Taktrate von 4 MHz geschickt wurden, keine Zeit, um bis zu den Chips zu propagieren.

Durch Anlöten der Kabel sowohl am Teensy als auch an den LED-Boards konnte das Problem schliesslich behoben werden. Die Boards akzeptieren nun Daten, die über das SPI-Interface gesendet werden, und zeigen diese korrekt an.

Aus mysteriösen Gründen leuchteten jedoch bei einem bestimmten Board alle LEDs des letzten Treibers auf, sobald ich das Netzteil anschloss – auch wenn keine Daten in den Chip geschickt wurden. Es lag nahe, dass dessen Dateneingang fälschlicherweise mit V_{DD} verbunden war, doch nach genauer Inspektion stellte sich heraus, dass dem nicht so war. (Selbst wenn es doch so gewesen wäre, hätten die LEDs erst nach 16 steigenden Flanken auf der CLK-Linie sowie einem Latch-Puls aufleuchten müssen; sie taten es aber auch ohne jegliche Verbindung zum Teensy.) In der Annahme, dass der Chip einen Produktionsfehler aufwies, ersetzte ich ihn mit einem meiner drei Ersatzexemplare. Danach funktionierte auch dieses Board wie gewünscht.

Es gab drei LEDs, die nicht plangemäss aufleuchteten. Bei der einen war es eine kleine Brücke zwischen der gemeinsamen Anode und der roten Kathode, die ein Spannungsgefälle über die LED verunmöglichte. Die anderen beiden LEDs waren schlecht angelötet und somit nicht mit dem Chip verbunden.

3.2 Gestell

3.2.1 Konstruktion

Aus Buchenholzstäben mit quadratischem Querschnitt von 1.5 cm Seitenlänge baute ich das Gerüst, um den Motor und die dazugehörige Welle zu montieren. Dazu befestigte ich zwei senkrechte 20 cm lange Stäbe auf eine Grundplatte und versah sie mit je drei Streben zur Stabilisation. Ein waagrechtes Brett, das auf einer Höhe von 7.5 cm an den senkrechten Stäben angebracht ist, trägt den Motor.

Zuoberst an den senkrechten Stäben montierte ich ein weiteres Brett, um die Welle lotrecht zu halten. Als Lager für die Welle dient ein simples Loch von 4.5 mm Durchmesser.

3.2.2 Schleifkontakte

Um die Stromversorgung für die rotierenden Bestandteile zu gewährleisten, brachte ich zuunterst auf der Welle eine runde Holzscheibe an. Diese ist oben und unten mit je einem ebenfalls runden Stück Kupferblech versehen. Von diesen führt je ein Kabel nach oben zur Elektronik.

Auf der stationären Seite bestehen die Schleifkontakte aus zwei langen Stücken Kupferblech, die durch ihre federähnlichen Eigenschaften auf die runde Scheibe gedrückt werden. Sie sind an der Seite des einen senkrechten Buchenholzstabes angebracht, damit sie möglichst lang sind.

Trotzdem war ein relativ hoher Druck nötig, um einen zuverlässigen Kontakt bilden zu können. Der Anpressdruck erwies sich auch als ein einfacher Weg, um die Drehzahl zu limitieren, wenn auch kein effizienter: der Motor wird trotz internem Lüfter spürbar warm. Leider verschleissen die stationären Kontakte bei einem so hohen Druck schnell, doch ein Stück Kupferblech ist schnell ersetzt.

3.2.3 Controller

Der Controller befindet sich auf einer weiteren runden Holzscheibe, die eine Fingerbreite oberhalb der Schleifkontakte angebracht ist. Auf dieser befindet sich auch der Hall-Sensor, mithilfe dessen das Programm das Timing an die aktuelle Geschwindigkeit anpasst (mehr dazu im nächsten Kapitel, Programmierung). Der dazugehörige Magnet befindet sich auf einer Gewindestange, die auf dem selben Brett wie der Motor befestigt ist. Auch der 2200 μ F-Kondensator findet auf dieser Scheibe Platz. Letzten Endes wurde dieser Bereich eher unansehnlich und unübersichtlich, doch während sich die Apparatur dreht, sieht man das ja kaum.

3.2.4 Leiterplatten

Die Boards hatte ich zuvor bereits zusammengelötet, es fehlte jedoch noch das Loch, um sie auf der Welle zu befestigen. Der Durchmesser der Welle beträgt exakt 4 mm, also führte ich bei einem der nicht verwendeten PCBs einige Testbohrungen durch. Ein 4 mm grosses Loch erwies sich als zu klein, doch bei 4.5 mm hatte die Leiterplatte sehr viel Spiel und hielt überhaupt nicht fest. Nachdem ich jedoch das Gewinde von der Welle wegsägte und deren Ende mit einer kleinen Fase versah, passte die Welle sehr knapp in ein Loch mit 4 mm Durchmesser. Dies erwies sich als Vorteil, da das Drehmoment zuverlässig von der Welle auf die Boards übertragen werden kann und die Boards nur minimal verrutschen.

4 Programmierung

4.1 Verwendete Software

Das nachfolgend erläuterte Programm wurde mit der Arduino IDE 1.0.6 [16] und dem Teensyduino-Plugin 1.20 [17] kompiliert und auf den Controller geschrieben. Den grössten Teil des Programmes habe ich jedoch in Sublime Text 2 [18] mit dem Stino-Plugin [19] geschrieben. Diese Kombination hat gegenüber der Arduino IDE einige Vorteile wie besseres Code-Highlighting, eine Autocomplete-Funktion, mehrere Schreibmarken sowie eine umfassendere Such- und Ersetzfunktion. Diese Programme installierte und benutzte ich auf Mac OS 10.9 und später 10.10.

4.2 Timing

4.2.1 Theorie

Damit ein stehendes Bild auch dann dargestellt werden kann, wenn die Drehzahl kleine Schwankungen aufweist, muss die Frequenz der Erneuerungen der Umdrehungsperiode angepasst werden. Pro Umdrehung sind es 100 Erneuerungen, also muss der Abstand zwischen jeder Erneuerung 1/100 der Umdrehungsperiode betragen.

Dieser Wert kann nun bei jeder Umdrehung errechnet und für die nächste Umdrehung verwendet werden. Wenn sich die Drehzahl ändert, ist das ein kleines Problem: Beim Beschleunigen sieht man die letzten Pixel einer Umdrehung nicht mehr, und beim Bremsen ist das Programm bereits vorzeitig mit der Anzeige des Bildes fertig. Doch bei konstanter Geschwindigkeit entspricht der letzte Messwert der aktuellen Periode, sodass ein stehendes Bild zu sehen ist.

4.2.2 Implementation

Nach jeder vollendeten Umdrehung müssen folgende Aufgaben erledigt werden:

- `outputTimer stoppen`
- `currentPixel auf 0 setzen`
- Dauer der letzten Umdrehung messen
- `outputTimer wieder starten, dass er die Methode sendData() 100 Mal während der nächsten Umdrehung aufruft`

`outputTimer` ist ein Objekt der `IntervalTimer`-Klasse [20], das bereits zuvor erstellt wurde. Durch das Aufrufen von `outputTimer.begin(funktion, zeitintervall)` beginnt es, die angegebene Funktion jedesmal aufzurufen, wenn das Zeitintervall (in μs) vorbei ist. Diese Funktion, `sendData()`, wird im nächsten Kapitel ausführlich erläutert. Mit dem Befehl `outputTimer.end()` wird der Timer wieder angehalten.

Die Variable `currentPixel` wird von `sendData()` bei jedem Aufruf um eins inkrementiert. Sie entspricht jeweils dem Winkel des obersten PCBs in Hundertsteln eines vollen Kreises,

also in Einheiten von 3.6°. Sobald der Hall-Sensor sich über dem Magneten befindet, wird `currentPixel` wieder auf 0 zurückgesetzt.

Die Implementation dieser Befehlsfolge ist relativ simpel. Sie befindet sich im Anhang, wo auch der Rest des Programmes aufgeführt ist. Es handelt sich dabei um die Funktion `timerUpdate()`.

Nun muss diese Funktion nach jeder Umdrehung aufgerufen werden. Hardwaremäßig habe ich hierzu einen digitalen Hall-Sensor vorgesehen, der immer wieder an einem stationären Magneten vorbeipassiert. Es gibt zwei Optionen, dessen Signal aufzuspüren, nämlich Polling und Interrupts. Polling besteht darin, immer wieder den Status des Signals abzufragen. Somit besitzt es eine limitierte zeitliche Auflösung und verhindert, dass während dem Polling nützliche Berechnungen erledigt werden können.

Ein Interrupt ist eine sehr viel elegantere Lösung. Sobald sich der Logikpegel auf einem bestimmten Pin ändert, wird der laufende Code unterbrochen und stattdessen eine zuvor festgelegte Funktion aufgerufen. In `setup()` kann `attachInterrupt(pin, funktion, modus)` aufgerufen werden, um den Interrupt zu erstellen [21]. Beim Teensy 3.0 und 3.1 ist jeder Pin mit Interrupt-Fähigkeit ausgerüstet [22]. Der `funktion`-Parameter legt fest, welche Funktion beim Eintritt des Interrupts aufgerufen werden soll. In meinem Fall ist dies `timerUpdate()`. Der letzte Parameter, `modus`, kann fünf Werte annehmen: LOW, HIGH, CHANGE, RISING und FALLING. Da der Hall-Sensor bei der Präsenz eines Magnetfeldes logisch 0 ausgibt, setzte ich den Modus auf FALLING.

Diesem Verhalten des Hall-Sensors liegt dessen invertierter Output zugrunde; der Schaltkreis aus dem Datenblatt ist im Anhang zu finden. Die Spannungsdifferenz, die über der Hall-Platte vorhanden ist, wird verstärkt und durch einen Schmitt-Trigger geschickt. Wenn dieser eine logische 1 ausgibt, wird der Output-Pin des Sensors mit GND verbunden. Bei einer logischen 0 wird der Pin aber nicht etwa auf V_{DD} gehalten, sondern einfach potentialfrei gelassen. Um in diesem Falle doch eine 1 zu lesen, muss extern ein Pullup-Widerstand verwendet werden. Zum Glück ist in jedem I/O-Pin des Teensy ein solcher vorhanden; um ihn zu aktivieren, musste im Code anstatt `pinMode(hallpin, INPUT)` einfach `pinMode(hallpin, INPUT_PULLUP)` verwendet werden [23].

4.3 Speicherung und Anzeige der Daten

4.3.1 Theorie

Um dem Controller mitzuteilen, wann er die LEDs ein- und ausschalten soll, erstellte ich im Arduino-Sketch ein dreidimensionales Array von $100 * 10 * 6$ Byte. Die erste Indexnummer steht für den Winkel eines bestimmten 2D-Bildes in Hundertsteln eines Kreises, die zweite für die Höhe einer Pixelreihe und die letzte für die Position eines Bytes auf einer Pixelreihe.

Mithilfe des einzigen gegebenen Wertes, `currentPixel`, soll die Methode `sendData()` aus dem `byteArray` die passenden Daten finden und diese mit der Methode `SPI.transfer(byte)`, die in der SPI-Bibliothek [24] enthalten ist, an die LED-Boards schicken.

Da die Boards nicht übereinander angeordnet sind, entsprechen die Daten von einem Refresh nicht den Pixeln, die übereinander dargestellt werden. Um das Schreiben eines

Bildes nicht unnötig zu erschweren, habe ich ein Programm geschrieben, das die wendeltreppenartige Anordnung der Boards berücksichtigt und aus verschiedenen Stellen des Arrays die passenden Daten sammelt. Dank diesem Programm ist es nun möglich, die Daten im Array genau so einzugeben, wie sie auch schlussendlich dargestellt werden sollen.

Dazu müssen zuerst die drei Indexnummern des byteArray sowohl von offsetPixel als auch von der Nummer der Platine abhängig gemacht werden. In der nachfolgenden Tabelle habe ich genau das getan. Die Zeilennummer entspricht jeweils der Nummer des Boards, wobei das oberste Board der obersten Zeile entspricht. Auch im byteArray entspricht die oberste Zeile jeweils dem obersten Board.

0. byteArray[(currentPixel+00+00)%100][0][j]
1. byteArray[(currentPixel+00+50)%100][1][j]
2. byteArray[(currentPixel+08+00)%100][2][j]
3. byteArray[(currentPixel+08+50)%100][3][j]
4. byteArray[(currentPixel+16+00)%100][4][j]
5. byteArray[(currentPixel+16+50)%100][5][j]
6. byteArray[(currentPixel+24+00)%100][6][j]
7. byteArray[(currentPixel+24+50)%100][7][j]
8. byteArray[(currentPixel+32+00)%100][8][j]
9. byteArray[(currentPixel+32+50)%100][9][j]

j ist die Laufvariable einer for-Schleife, die rückwärts von 5 bis 0 zählt. Sie ist dafür zuständig, die Daten des Arrays für eine der LED-Reihen, die ja immer nebeneinander stehen, von rechts nach links hinauszuschicken.

Nach zwei Boards wird stets ein weiteres mal 8 addiert. Jedes PCB-Paar ist gegenüber dem jeweils darüber liegenden um 8 Pixel beziehungsweise 28.8° versetzt, also braucht es auch die Daten, die im byteArray 8 Pixel weiter vorne liegen.

Bei jedem ungerade nummerierten Board wird zusätzlich 50 zum Wert von offsetPixel addiert. Das hat den Grund, dass dieses sich gegenüber dem darüber liegenden gerade nummerierten Board befindet. 50 Pixel entsprechen genau dem halben Kreis, der zwischen diesen beiden Boards liegt.

Es könnte nun nach all diesen Additionen vorkommen, dass eine Position aufgerufen wird, die über 99 liegt und somit ungültig ist. Da sich die Position 100+n an der gleichen Stelle befindet wie die Position n, können durch einen Modulo100-Operator alle Werte unter 100 gezwungen werden.

Ein weiterer Effekt des Modulo-Operators tritt ans Licht, wenn die Geschwindigkeit des Apparates abnimmt. In diesem Fall ist der letzte Messwert der Periode ein wenig zu kurz, und offsetPixel erreicht und überschreitet bereits vor dem Ende der Umdrehung den Wert 99. So beginnt das Programm einfach wieder mit der Darstellung der ersten Pixel, anstatt inexistente Positionen zuzugreifen.

4.3.2 Implementation

Um diese Tabelle zu implementieren, eignet sich natürlich eine for-Schleife. Damit das zu addierende Vielfache von 8 von deren Laufvariable *i* abhängig gemacht werden kann, muss die Schleife sowohl ein gerade nummeriertes als auch das zugehörige ungerade nummerierte Board in einem Durchlauf mit Daten versorgen.

Um dies zu tun, gibt es in der Schleife wiederum zwei innere for-Schleifen. Die erste sendet die Daten für das Board $2*i$. Bevor diese ausgeführt wird, wird `currentPixel+8*i` in einer anderen Variable, `offsetPixel`, gespeichert. Da `currentBoard` nach der Erneuerung nicht mehr gebraucht wird, wird es direkt auf $2*i$ gesetzt. Schliesslich werden die Daten in der ersten inneren Schleife entsprechend diesen Werten gesendet. Diese Schleife zählt mit der Laufvariable *j* von 5 bis 0, sodass die Daten von rechts nach links geschickt werden. Damit dies auch innerhalb eines einzelnen Bytes der Fall ist, wird das `SPISettings`-Objekt `einstellungen` mit dem Parameter `LSBFIRST` erstellt.

Die zweite innere Schleife versorgt die Platine $2i + 1$ mit ihren Daten, also alle ungerade nummerierten. Vor deren Durchlauf wird `offsetPixel` auf $(offsetPixel+50)\%100$ gesetzt, sodass die Daten für die Platine auf der anderen Seite geschickt werden. `boardAktuell` wird um 1 erhöht und die selbe innere for-Schleife wird noch einmal mit den neuen Werten ausgeführt.

Nachdem die Daten für einen Refresh gesendet sind, werden diese mit einem kurzen Puls des Latch-Pins auf die Ausgänge angewandt. Danach wird `offsetPixel` um 1 inkrementiert, sodass beim nächsten Aufruf von `sendData()` wieder die richtigen Daten ihren Weg in die LEDs finden.

Der erläuterte Code ist ebenfalls im Anhang aufgeführt, und zwar in der Methode `sendData()`, die vom `outputTimer` 100-Mal pro Umdrehung ausgeführt wird.

4.4 Weiteres

Das Programm erfüllte so zwar seine Funktion, doch es gab noch das eine oder andere verbesserungswürdige Detail.

Da ich die Position des Hall-Sensors willkürlich gewählt hatte und nicht Rücksicht auf die Position der ersten Erneuerung

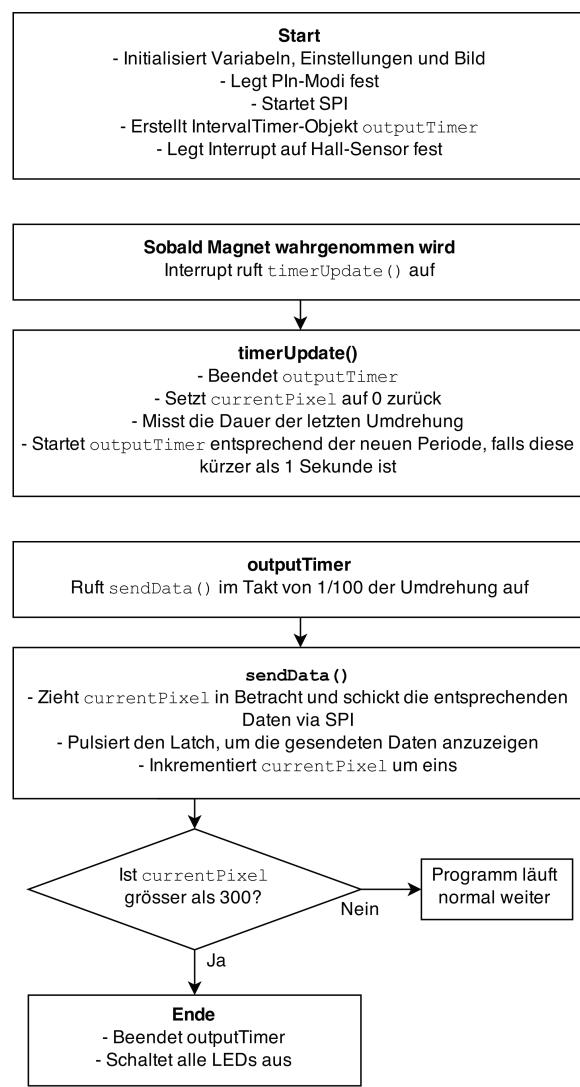


Abbildung 5: Flussdiagramm zur Veranschaulichung des Programmablaufs

genommen hatte, befand sich das `currentPixel` mit dem Index 0 bei etwa 245°. Indem ich zu bei der Zuweisung von `offsetPixel` noch 68 (=245/3.6) addierte, platzierter ich das nullte Pixel bei 0°.

Damit das Programm beim Aufhören der Drehbewegung nicht ewig weiter läuft, habe ich am Ende von `sendData()` eine kleine Verzweigung eingeschoben. Wenn `currentPixel` grösser als 300 ist (also wenn die aktuelle Umdrehung bereits dreimal so lange dauert wie die letzte), dann wird die Funktion `shutDownLEDs()` aufgerufen, die 60 Mal das Byte 0x00 an die LED-Treiber schickt und die Latch-Linie pulsiert.

In `timerUpdate()` fügte ich eine ähnliche Verzweigung hinzu, die den Timer nur dann startet, wenn die Rotationsperiode kleiner als eine Sekunde ist. So kann es höchstens drei Sekunden dauern, bis `currentPixel` 300 erreicht hat. Falls die letzte Umdrehung länger als eine Sekunde dauert, wird genau wie oben `shutDownLEDs()` aufgerufen.

5 Inbetriebnahme

5.1 Stromversorgung

Anfangs hatte ich vor, die Boards mit 5V zu versorgen. Doch der Teensy sendet seine Logiksignale mit 0V oder 3.3V. Die Chips lesen erst ab $0.7 * V_{DD}$ eine logische 1; und bei 5V ist dieser Wert 3.5V. So versorgte ich die Boards mit 3.3V, wobei ich erfreut feststellte, dass auch die Wärmeproduktion erheblich gesunken ist.

Der Teensy 3.1 braucht gemäss Hersteller mindestens 3.7V [25], vermutlich weil er die Eingangsspannung mit einem Regulator auf 3.3V hält. Es ergaben sich jedoch keine Probleme, als ich auch den Controller mit 3.3V betrieb. Obwohl diese Spannung durch den Regulator noch weiter reduziert wird, läuft selbst bei den vollen 96 MHz alles wie gewollt.

Leider gab das ATX-Computernetzteil, das ich gebraucht gekauft hatte, den Geist auf. Wie sich später herausstellte, konsumierte der Motor viel mehr Strom, als ich vermutet hatte. Zum Glück konnte ich stattdessen die beiden Netzteile der Schule verwenden. Das eine ist in der Lage, bis zu 25 A zu liefern; dieses schloss ich an den Motor an. Bei etwa einem Volt verbraucht dieser 5-7 A, sobald die Endgeschwindigkeit erreicht ist. Wenn man die Spannung auf 2 V erhöht, werden es etwa 10 A.

Diese Spannung ist nicht nur zu tief für die Elektronik, sondern sie ist auch zu instabil. So verwendete ich für die Elektronik das zweite Netzteil. Die LEDs erreichen bei etwa 3.5V ihre volle Helligkeit. Auch der Controller funktioniert zwar bei dieser Spannung, doch um die Herstellerangaben zu befolgen, verwendete ich 4 V.

5.2 Weiteres

Die Schleifkontakte sind in ihrer Position nicht fixiert und rutschen nach längerem Betrieb zur Seite. Also montierte ich ein kleines Holzbrett, das die Schleifkontakte in ihrer Position hält.

Da die Boards nicht exakt im korrekten Winkel zueinander montiert waren, wurde das Testbild nicht optimal angezeigt: Pixel, die eigentlich übereinander sein sollten, waren leicht versetzt. Ich musste deshalb die meisten Boards nochmal adjustieren. Beim Betrieb verrutschen die Boards langsam, sodass die Winkel zwischen ihnen immer wieder nachkorrigiert werden müssen.

Die Stoppmuttern, welche den Motor unter seinem Brett fixieren, lockerten sich durch die Vibrationen. Damit das nicht wieder vorkommt, benutzte ich eine zweite Mutter und kontrte damit die bereits vorhandene.

6 Beurteilung

6.1 Mögliche Verbesserungen

6.1.1 Controller und Datenübertragung

Der Teensy 3.1 ist aktuell relativ lose auf seiner Holzscheibe befestigt. Ein Stück Draht geht durch zwei unbenutzte Kontakte und durch zwei Löcher in der Holzscheibe. Auch die Kabel für das SPI-Interface sind alles andere als säuberlich verlegt; viele sind noch zu lang und beschreiben Umwege.

Diese Situation wäre zu verbessern, indem die obere Holzscheibe durch ein PCB ersetzt wird. Dann könnten alle Verbindungen elegant auf das PCB geroutet werden, und mit zwei Flachbandkabeln wäre es möglich, die Daten ohne Kabelsalat zu den LED-Boards hinaufzusenden. Auch der Hall-Sensor könnte besser angelötet werden.

Auch zwischen den LED-Boards wären Flachbandkabel eine geeigneter Lösung als die aktuell verbauten einzelnen Kabel. Es sind jedoch nur CLK, LE und OE immer mit dem darüber liegenden Pin verbunden, während SDI und SDO mit einem anderen Board verbunden werden müssen. Diese Verbindungen müssten auf jeden Fall mit einzelnen Kabeln realisiert werden.

6.1.2 Antrieb

Den direkten Antrieb ohne jegliche Untersetzung habe ich gewählt, weil er so simpel ist. Es gibt nur einen beweglichen Bestandteil und wenige Reibungspunkte. Doch das konnte ich nur erreichen, weil der Motor mehr als stark genug war. Leider konsumiert dieser auch eine beträchtliche Strommenge. Beim Einschalten ist diese so hoch, dass sich mein PC-Netzteil sofort wieder ausschaltete.

Wenn mein Display als kommerzielles Produkt gefertigt werden würde, wäre die Verwendung eines kleineren Motors und einer Untersetzung ratsam. Durch den Einbau von Kugellagern und Schleifkontakte mit weniger Reibung wäre ein schwächerer Motor kein schwerwiegender Nachteil mehr.

6.1.3 Schleifkontakte



Abbildung 6: Drehender Teil des unteren Schleifkontakte

Da die obere Kupferplatte nicht perfekt eben ist, oszilliert deren Höhe während der Drehung ständig. Die Amplitude der Schwankung beträgt etwa einen Millimeter. Wenn man als Rotationsfrequenz 30 Hz annimmt und für die x-Achse Sekunden verwendet, kann die Abweichung durch folgende Funktion approximiert werden:

$$\text{Vertikale Abweichung [mm]} \approx \sin(30 \cdot 2\pi \cdot x) / 2$$

Die zweite Ableitung dieser Funktion entspricht der Beschleunigung:

$$\text{Vertikale Beschleunigung [mm/s}^2] \approx -3600 \cdot \pi^2 \cdot \sin(60 \cdot \pi \cdot x)$$

Da ausserhalb des Sinus nur Konstanten vorhanden sind, muss nur der Sinus gleich 1 gesetzt werden, um die ganze Funktion zu maximieren. Die maximale vertikale Beschleunigung der Kupferscheibe ist somit etwa 35.5 m/s^2 . Um unter diesen Umständen immer Kontakt halten zu können, muss der stationäre Kontakt relativ stark gespannt sein. Somit ist auch die Reibungskraft, die zwischen den Kontakten wirkt, relativ hoch. Da diese Kraft nicht nahe an der Welle, sondern bei einem Radius von etwa 2 cm wirkt, ist wiederum das Drehmoment, das der Motor aufwenden muss bedeutend. Somit konsumiert dieser aufgrund der Schleifkontakte mehr Strom als nötig.

Bereits nach kurzem Betrieb von etwa einer halben Stunde sind die Schleifkontakte stark verschlissen (Abbildung 6). Kupfer ist zwar wegen seiner Leitfähigkeit für Schleifkontakte geeignet, doch leider ist es auch sehr weich.

Bei den stationären Kontakten ist der Verschleiss stärker, da sie während der ganzen Umdrehung in Kontakt mit der drehenden Kupferscheibe sind. Doch das ist kein grosses Problem; diese Kontakte lassen sich relativ simpel austauschen. Um die drehenden Kontakte zu ersetzen, wäre jedoch ein beträchtlicher Aufwand nötig. Es müsste sämtliche Elektronik von der Welle entfernt werden und neben den Kupferscheiben müsste auch die Holzscheibe ersetzt werden, auf der sie angeleimt sind.

6.1.4 Energieeffizienz

Um den hohen Energieverbrauch zu senken, wäre vor allem eines nötig, nämlich bessere Schleifkontakte. Bereits am Anfang hatte ich eine alternative Möglichkeit im Sinn, den Strom zu übertragen, und zwar über einen 3.5-mm-Klinkenstecker, wie er in jedem Kopfhörer vorhanden ist [26]. Ich entschloss mich dagegen, den Strom so zu übertragen, und zwar aus zwei Gründen. Erstens hatte ich Bedenken, dass der Stecker die 9.88 A aushalten kann. In der Praxis ist der RMS-Strom jedoch bedeutend kleiner. Selbst wenn der Widerstand des Steckers immer noch zu hoch wäre, wäre als Alternative der grössere Klinkenstecker mit 6.35 mm Durchmesser bereitgestanden.

Der zweite Grund, der gegen die Verwendung eines solchen Steckers sprach, waren die Schwierigkeiten, ihn zu montieren. Er müsste nämlich am Ende der Welle Platz finden und somit die Funktion des Lagers übernehmen. Um die auf ihn wirkenden Kräfte zu minimieren, müsste er exakt in der Mitte der Welle angebracht werden. Diese wären selbst dann beträchtlich gewesen, sodass die Verbindung sehr stark hätte sein sollen.

Angenommen, diese Probleme seien gelöst, könnte mit einem Klinkenstecker als Schleifkontakt der Energieverbrauch des Motors erheblich gesenkt werden. Eventuell wäre es sogar möglich, einen kleineren Motor zu verwenden.

6.2 Ausblick

6.2.1 Animierte Bilder

Eine Erweiterung, die ich bereits am Anfang der Planung im Hinterkopf hatte, sind animierte Bilder. Da der Teensy 3.1 mit 256 kB an Flash-Speicher ausgestattet ist und ein Bild 6 kB gross ist, können höchstens 42 verschiedene Bilder gespeichert werden. Das entspricht einer guten Sekunde an Animationsdauer. Um längere Sequenzen zeigen zu können, könnte man anstatt einem Bild, das alle 100 Pixel überspannt, ein kleineres mehrmals anzeigen. Wenn zum Beispiel vier Mal die gleichen 25 Pixel angezeigt werden, könnten bereits 170 individuelle Frames gespeichert werden, was bei 30 fps einer Animationsdauer von 5.7 Sekunden entspricht.

Um das Limit der Flash-Kapazität zu umgehen, wäre eine externer Speicher notwendig, zum Beispiel in Form einer SD-Karte. Dies wäre sowohl hardware- als auch softwaremässig eine aufwändige Lösung, doch ein einziges Gigabyte würde für 46.3 Stunden an Playback reichen. Somit wäre diesbezüglich praktisch jedes Limit entfernt.

6.2.2 Interaktive Programme

Interessant wäre es auch, anstelle von fixen Bildern ganze Programme laufen zu lassen. Für ein brauchbares Programm muss der Benutzer natürlich Eingaben machen können. Auch das würde die Komplexität des Projektes freilich stark erhöhen. Um nämlich zur Laufzeit mit der Aussenwelt zu kommunizieren, ist ein drahtloses Interface notwendig. Ein entsprechender Empfänger braucht – wie ein Slot für eine SD-Karte – zusätzlichen Platz auf der drehenden Apparatur und die entsprechende Software, um die erhaltenen Signale zu interpretieren und verarbeiten.

6.2.3 Einsatzgebiete

Aufgrund der tiefen Auflösung, dem hohen Energieverbrauch, der Lautstärke und dem schnellen Verschleiss ist mein Display wohl kaum als Ersatz für den gewöhnlichen Computerbildschirm oder Fernseher geeignet.

Ein Gebiet, in dem sich der Einsatz eines derartigen Displays jedoch lohnen könnte, wäre die Werbung. Da noch kaum jemand eine solche Technologie zu Gesicht bekommen hat, wirkt sie sehr anziehend. Indem ein Geschäft ein Display wie das meinige im Schaufenster stehen hat, kann es potentielle Kunden anziehen. Ganz nebenbei, aber nicht übersehbar kann ein zu bewerbendes Produkt oder Firmenlogo auf dem Display dargestellt werden.

Ein weiteres geeignetes Gebiet wären Videospiele. Natürlich können keine modernen Spiele mit detaillierter Graphik angezeigt werden. Vielmehr könnten klassische Spiele, die tiefe Ansprüche an Prozessorleistung und Bildschirmauflösung stellen, in drei Dimensionen wiederbelebt werden. Tetris, Snake oder Pong in 3D wären sicherlich einen Versuch wert.

7 Quellenverzeichnis

- [1] POV-Display, 2014-10-06, <http://www.mikrocontroller.net/articles/POV-Display>
- [2] ATtiny85 POV Display, 2014-10-06, <http://www.instructables.com/id/ATtiny8545-POV-Display/>
- [3] Build an LED Propeller Clock, 2014-10-06,
<http://www.jameco.com/Jameco/workshop/MyStory/diy-led-propeller-clock.html>
- [4] Maximilian Mali: VoLumen – Volumetric 3D Display Device, 2014-10-06,
<https://hackaday.io/project/1737-volumetric-3d-display-device>
- [5] Jason Hotchkiss: 3D POV Display... Work in Progress, 2014-10-06,
<https://www.youtube.com/watch?v=rlptJzWwCXg>
- [6] Wikipedia: Persistence of Vision, 2014-10-06,
http://en.wikipedia.org/wiki/Persistence_of_vision
- [7] Lexikon der Film begiffe: Nach bild, 2014-10-06, <http://filmlexikon.uni-kiel.de/index.php?action=lexikon&tag=det&id=6114>
- [8] LED Anzeige – Diskret, 2014-10-06, <http://www.digikey.ch/product-search/de/optoelectronics/led-indication-discrete/>
- [9] Teensy USB Development Board, 2014-10-06, <https://www.pjrc.com/teensy/>
- [10] New I²C Library for Teensy 3, 2014-12-12, <http://forum.pjrc.com/threads/21680-New-I2C-library-for-Teensy3>
- [11] I²C Tutorial, 2014-12-12, <http://www.best-microcontroller-projects.com/i2c-tutorial.html>
- [12] TLC5927, 2014-10-06, <http://www.ti.com/product/tlc5927/>
- [13] TLC5927IDBQR, 2014-10-06, <http://www.digikey.ch/product-detail/de/TLC5927IDBQR/296-24762-1-ND/>
- [14] EAGLE PCB – Cadsoft USA, 2014-12-12, <http://www.cadsoftusa.com/eagle-pcb-design-software/product-overview/>
- [15] OSH Park – An electric Ecosystem, 2014-10-17, <https://oshpark.com/>
- [16] Download the Arduino Software, 2014-11-28, <http://arduino.cc/en/main/software>
- [17] Download Teensyduino, Version 1.20, 2014-11-28,
https://www.pjrc.com/teensy/td_download.html
- [18] Sublime Text 2, 2014-11-28, <http://www.sublimetext.com/2>
- [19] Robot-Will/Stino auf GitHub, 2014-11-28, <https://github.com/Robot-Will/Stino>
- [20] IntervalTimer, 2014-11-28, https://www.pjrc.com/teensy/td_timing_IntervalTimer.html

- [21] attachInterrupt() – Arduino Reference, 2014-11-28,
<http://arduino.cc/en/Reference/attachInterrupt>
- [22] Kommentar von Paul Stoffregen auf der Teensy 3.0 Kickstarter-Seite, 2014-11-28,
<https://www.kickstarter.com/projects/paulstoffregen/teensy-30-32-bit-arm-cortex-m4-usable-in-arduino-a/comments?cursor=1654857>
- [23] Using Digital I/O Pins, 2014-11-28, https://www.pjrc.com/teensy/td_digital.html
- [24] SPI Arduino Library, connecting SPI devices to teensy, 2014-11-14,
https://www.pjrc.com/teensy/td_libs_SPI.html
- [25] Teensy 3.1 Pin Assignments, 2014-11-28, <https://www.pjrc.com/teensy/pinout.html>
- [26] Klinkenstecker – Wikipedia, 2014-12-12, <http://de.wikipedia.org/wiki/Klinkenstecker>

7.1 Bildnachweise

Titelbild: Eigene Aufnahme

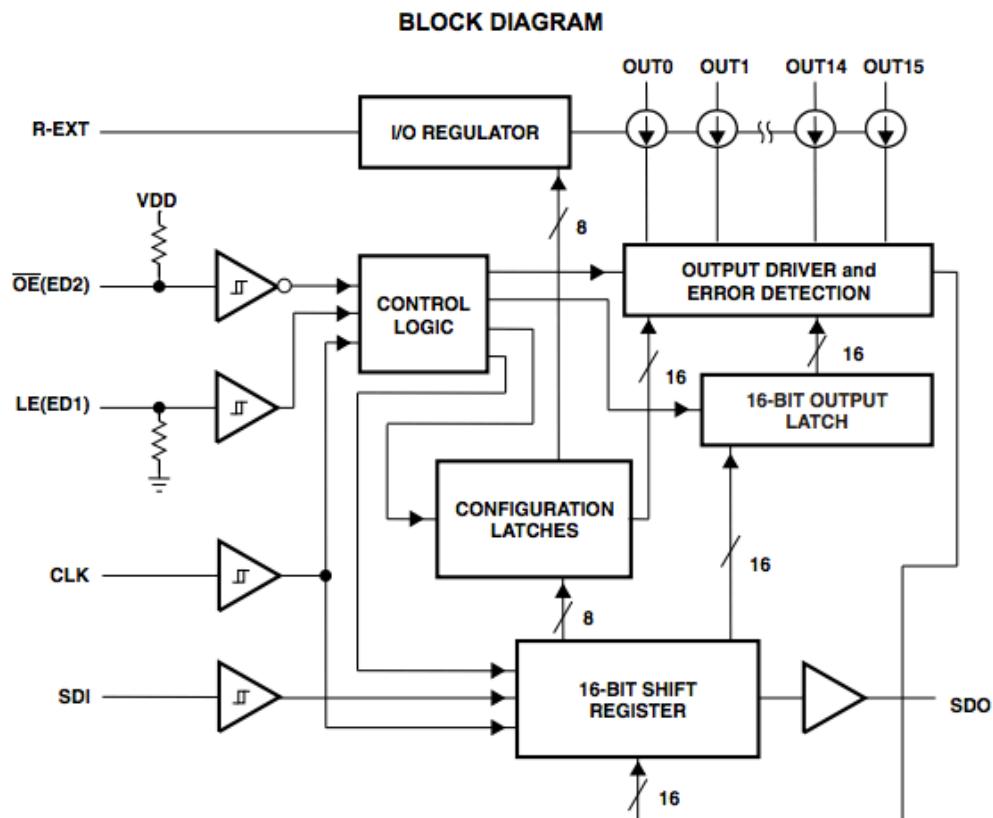
1. Texas Instruments, Datenblatt TLC5926/5927
2. Eigene Grafik, erstellt mit Eagle
3. Eigene Grafik, erstellt mit Eagle
4. Eigene Aufnahme
5. Eigene Grafik, erstellt mit <https://www.draw.io/>
6. Eigene Aufnahme

8 Anhang

8.1 Datenblatt des TLC5927 LED-Treibers

(Ganzes Datenblatt: <http://www.ti.com.cn/lit/ds/symlink/tlc5926.pdf>)

8.1.1 Seite 2: Block Diagram



8.1.2 Seite 3: Pin Descriptions

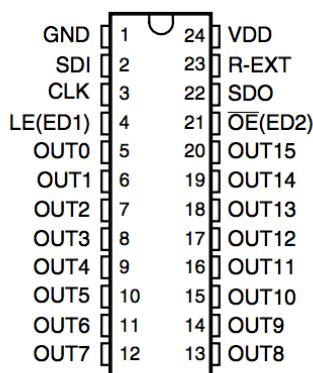


TLC5926, TLC5927

www.ti.com

SLVS677–JULY 2008

**DBQ, DW, OR PWP PACKAGE
(TOP VIEW)**



Pin Descriptions

PIN NAME	DESCRIPTION
CLK	Clock input pin for data shift on rising edge
GND	Ground pin for control logic and current sink
LE(ED1)	Data strobe input pin Serial data is transferred to the respective latch when LE(ED1) is high. The data is latched when LE(ED1) goes low. Also, a control signal input for an Error Detection mode and Current Adjust mode (See Timing Diagram). LE(ED1) has an internal pulldown.
OE(ED2)	Output enable pin. When OE (ED2)(active) is low, the output drivers are enabled; when OE(ED2) is high, all output drivers are turned OFF (blanked). Also, a control signal input for an Error Detection mode and Current Adjust mode (See Timing Diagram). OE(ED2) has an internal pull-up.
OUT0–OUT15	Constant-current output pins
R-EXT	Input pin used to connect an external resistor for setting up all output currents
SDI	Serial-data input to the Shift register
SDO	Serial-data output to the following SDI of next driver IC or to the microcontroller
VDD	Supply voltage pin

8.1.3 Seite 15: Adjusting Output Current

Adjusting Output Current

TLC5926/TLC5927 scales up the reference current, I_{ref} , set by the external resistor R_{ext} to sink a current, I_{out} , at each output port. Users can follow the below formulas to calculate the target output current $I_{OUT,target}$ in the saturation region:

$$V_{R-EXT} = 1.26 \text{ V} \times VG$$

$I_{ref} = V_{R-EXT}/R_{ext}$, if another end of the external resistor R_{ext} is connected to ground.

$$I_{OUT,target} = I_{ref} \times 15 \times 3^{CM - 1}$$

Where R_{ext} is the resistance of the external resistor connected to the R-EXT terminal, and V_{R-EXT} is the voltage of R-EXT, which is controlled by the programmable voltage gain (VG), which is defined by the Configuration Code. The Current Multiplier (CM) determines that the ratio $I_{OUT,target}/I_{ref}$ is 15 or 5. After power on, the default value of VG is $127/128 = 0.992$, and the default value of CM is 1, so that the ratio $I_{OUT,target}/I_{ref} = 15$. Based on the default VG and CM.

$$V_{R-EXT} = 1.26 \text{ V} \times 127/128 = 1.25 \text{ V}$$

$$I_{OUT,target} = (1.25 \text{ V}/R_{ext}) \times 15$$

Therefore, the default current is approximately 52 mA at 360Ω and 26 mA at 720Ω . The default relationship after power on between $I_{OUT,target}$ and R_{ext} is shown in Figure 11.

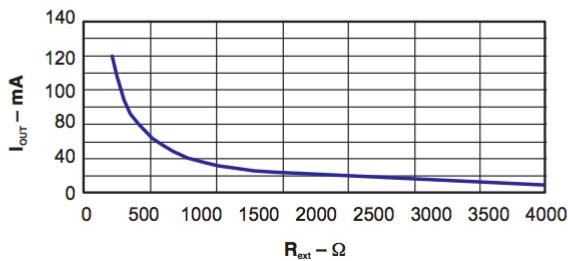
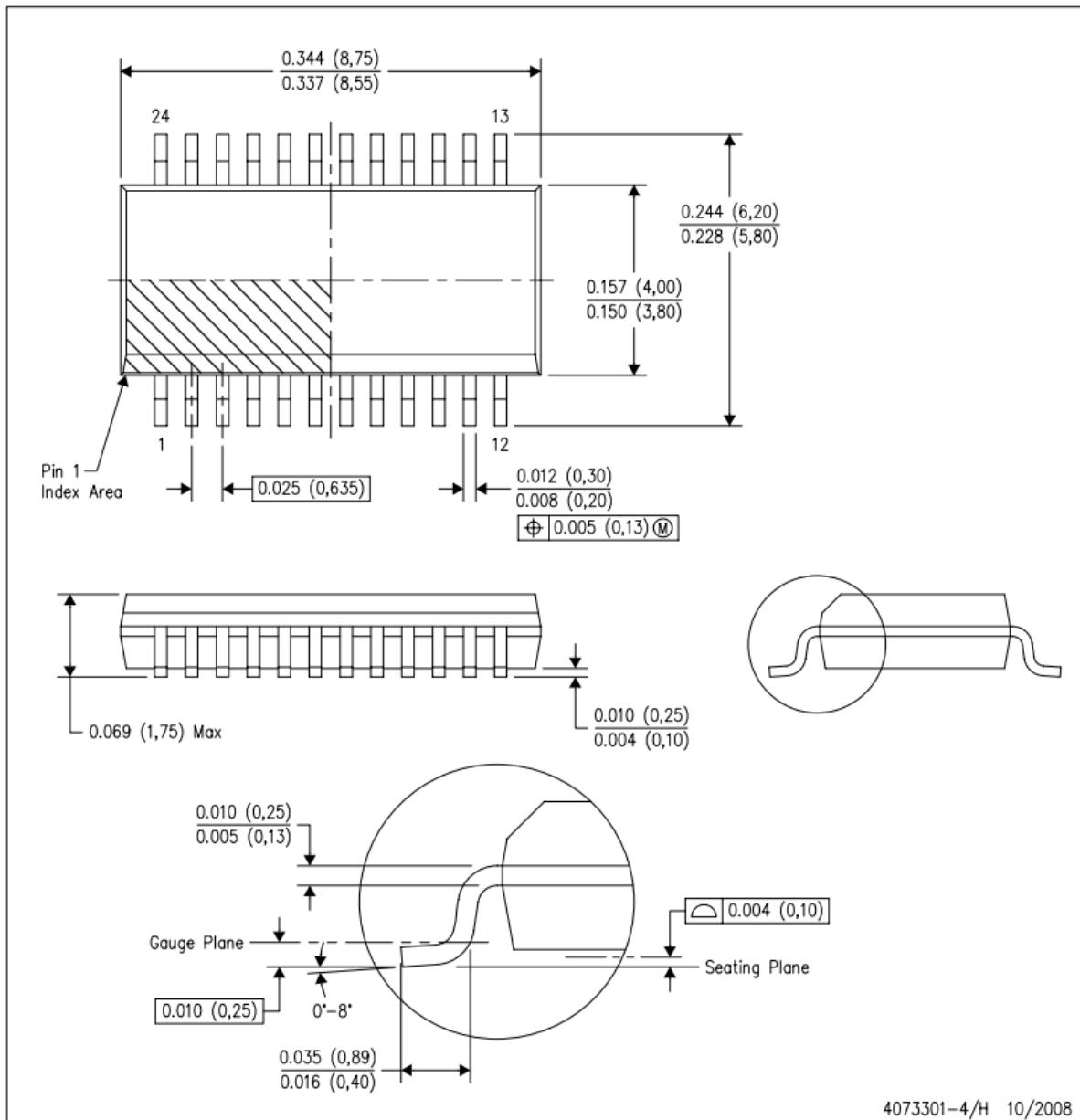


Figure 11. Default Relationship Curve Between $I_{OUT,target}$ and R_{ext}

8.1.4 Seite 27: Mechanical Data

DBQ (R-PDSO-G24)

PLASTIC SMALL-OUTLINE PACKAGE

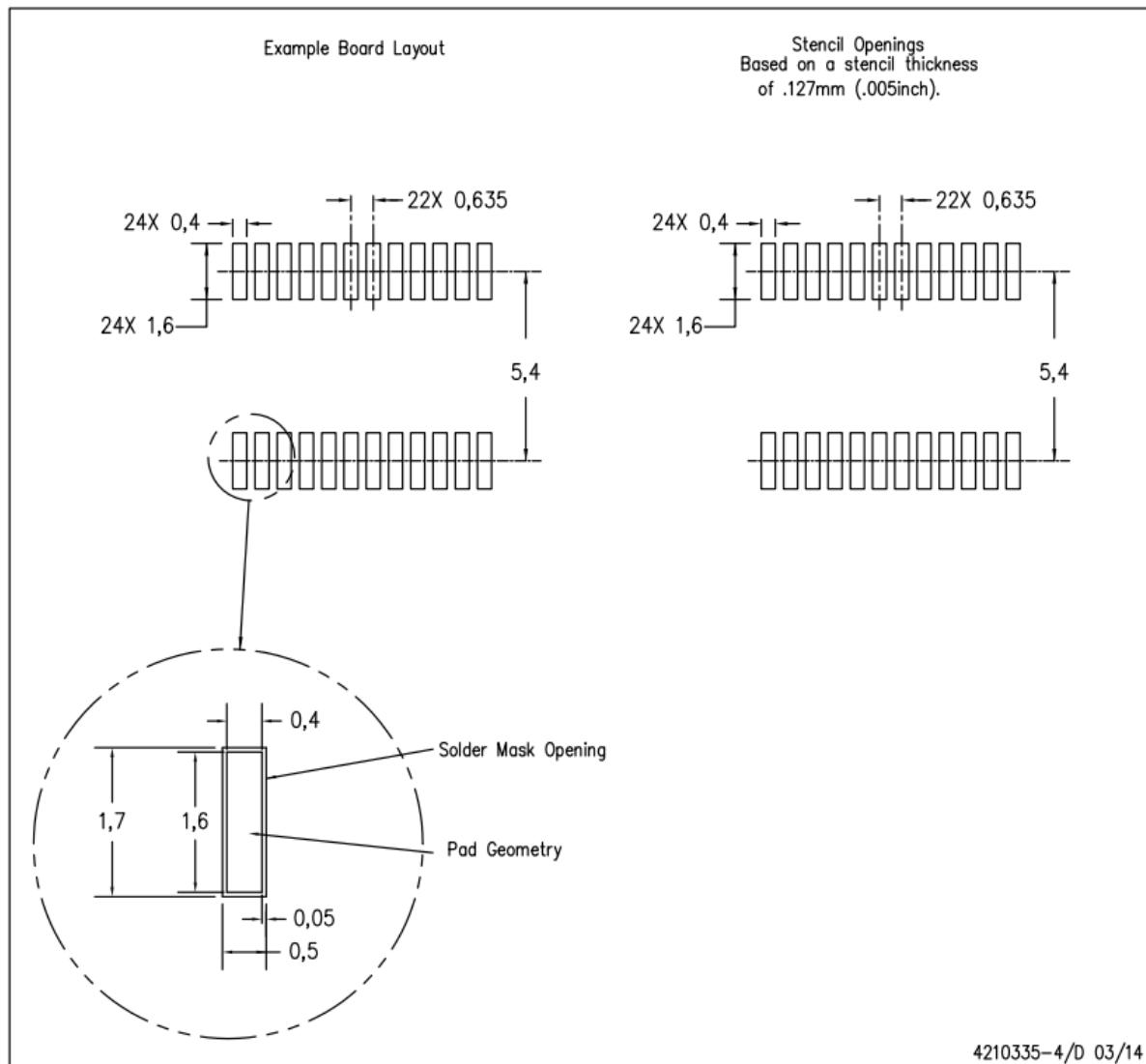


- NOTES:
- All linear dimensions are in inches (millimeters).
 - This drawing is subject to change without notice.
 - Body dimensions do not include mold flash or protrusion not to exceed 0.006 (0,15) per side.
 - Falls within JEDEC MO-137 variation AE.

8.1.5 Seite 28: Land Pattern Data

DBQ (R-PDSO-G24)

PLASTIC SMALL OUTLINE PACKAGE

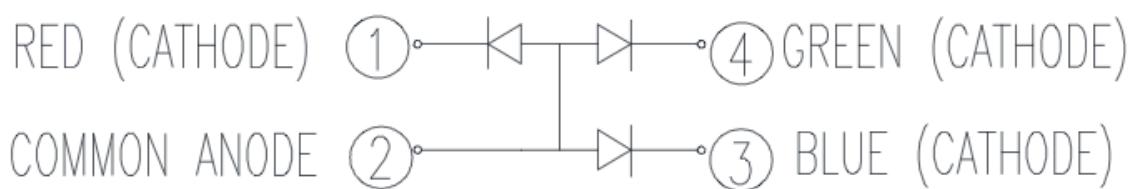
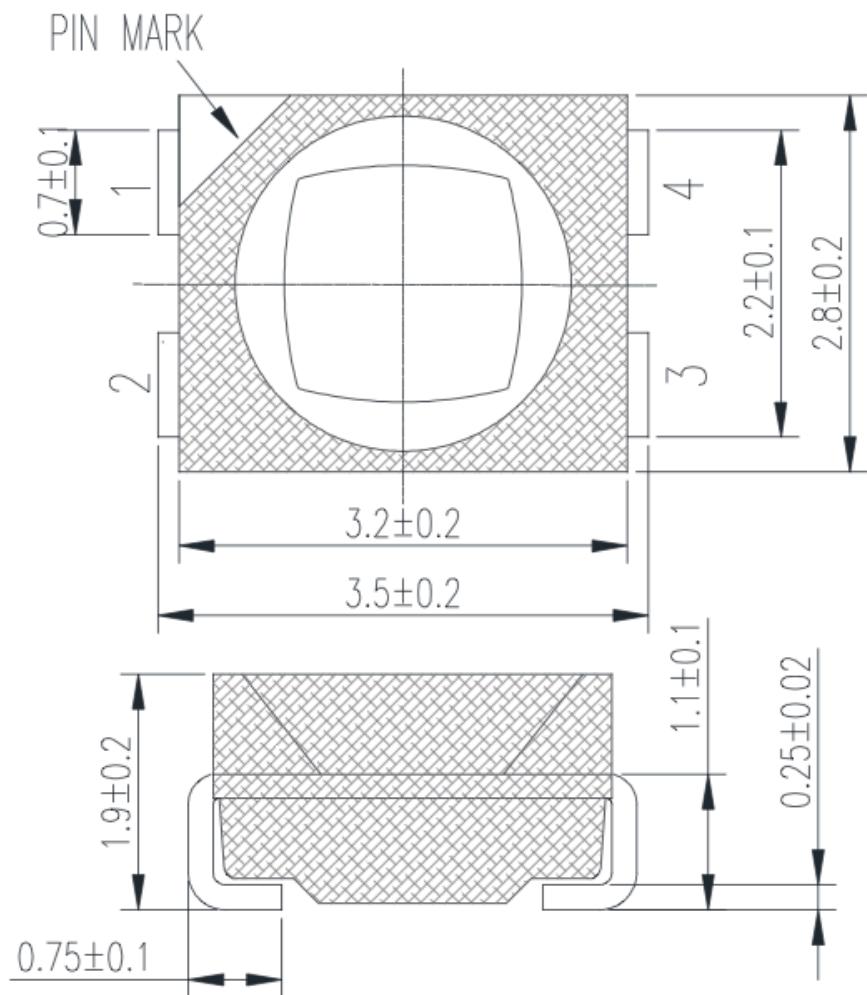


- NOTES:**
- All linear dimensions are in millimeters.
 - This drawing is subject to change without notice.
 - Publication IPC-7351 is recommended for alternate designs.
 - Laser cutting apertures with trapezoidal walls and also rounding corners will offer better paste release. Customers should contact their board assembly site for stencil design recommendations. Example stencil design based on a 50% volumetric metal load solder paste. Refer to IPC-7525 for other stencil recommendations.

8.2 Datenblatt der CLVBA-FKA-CAEDH8BBB7A363 RGB-Leuchtdiode

(Ganzes Datenblatt: <http://www.cree.com/~media/Files/Cree/LED Components and Modules/HB/Data Sheets/CLVBAFKA.pdf>)

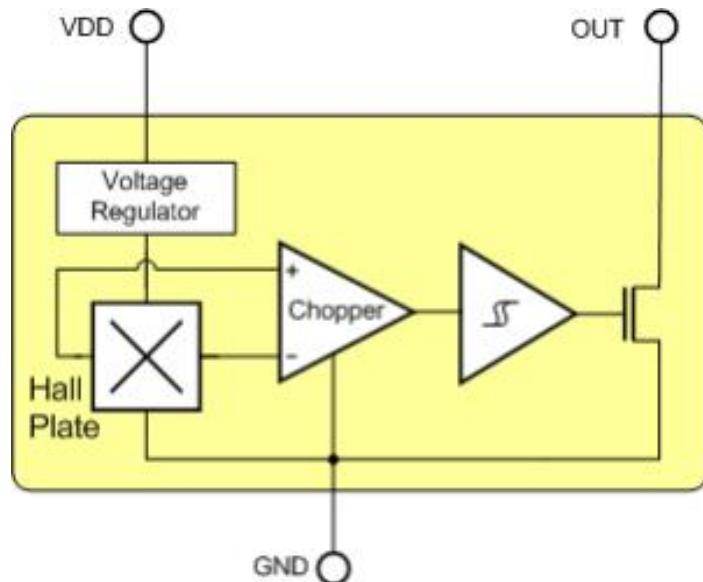
8.2.1 Seite 7: Mechanical Dimensions



8.3 Datenblatt des US5881 Hall-Sensors

(Ganzes Datenblatt: http://www.melexis.com/prodfiles/0004824_US5881_rev008.pdf)

8.3.1 Seite 1: Functional Diagram



8.4 Quellcode des Programmes

8.4.1 _3DPOV.ino

```
#include <SPI.h>

byte byteArray[100][10][6] = {
    //byte array that defines the displayed image
} //End of byteArray
//Changing values. Volatile because they are handled within an Interrupt
volatile int period = 0;
volatile int currentPixel = 0;
volatile int microsOld = 0;
volatile int offsetPixel = 0;
volatile int currentBoard = 0;

//Fixed values
const int LE=5;
const int OE=4;
const int hallpin = 20;
```

```
//SPISettings object is used to begin SPI transaction with
SPISettings mySettings(16000000, LSBFIRST, SPI_MODE0);

IntervalTimer outputTimer;

void setup() {
    //Initialise pins
    pinMode(hallpin, INPUT_PULLUP);
    pinMode(OE, OUTPUT);
    pinMode(LE, OUTPUT);

    SPI.begin();

    //As soon as the magnet is detected, the Interrupt calls timerUpdate()
    attachInterrupt(hallpin, timerUpdate, FALLING);
} //End of setup()

void loop() {}

void timerUpdate() {
    //Reset everything
    outputTimer.end();
    currentPixel = 0;

    //Find out rotational period
    period = micros() - microsOld;
    microsOld = micros();

    //Start timer, if not too slow. Otherwise shut down LEDs
    if(period<1000000) {
        outputTimer.begin(sendData, period/100);
    } else {
        shutDownLEDs();
    } //End if
} //End of timerUpdate()

void sendData() {
    SPI.beginTransaction(mySettings);

    for(int i=0; i<5; i++) {
        //Refresh even layer 2*i. +68 so that pixel #0 is displayed at 0°.
```

```
currentBoard = 2*i;
offsetPixel = (currentPixel + 8*i + 68)%100;
for(int j=5; j>=0; j--) {
    SPI.transfer(byteArray[offsetPixel][currentBoard][j]);
}

//Refresh uneven layer 2*i+1
currentBoard++;
offsetPixel = (offsetPixel+50)%100;
for(int j=5; j>=0; j--) {
    SPI.transfer(byteArray[offsetPixel][currentBoard][j]);
}
} //End of outer for loop

//Pulse the latch
digitalWrite(LE, HIGH);
delayMicroseconds(1);
digitalWrite(LE, LOW);

SPI.endTransaction();
currentPixel++;

//End the program if there's no rotation anymore
if (currentPixel>300) {
    outputTimer.end();
    shutDownLEDs();
}
} //End of sendData()

void shutDownLEDs() {
    //Fill all shift registers with 0's
    SPI.beginTransaction(mySettings);
    for(int i=0; i<60; i++) {
        SPI.transfer(0x00);
    }

    //Latch the 0's to the output to turn everything off
    digitalWrite(LE, HIGH);
    delayMicroseconds(1);
    digitalWrite(LE, LOW);
} //End of shutDownLEDs()
```