Aufbau einer LED-Statusanzeige für ein Mikrofonarray

Bachelorarbeit

Brandenburgische Technische Universität Cottbus
Fakultät 3 für Maschinenbau, Elektrotechnik und Wirtschaftsingenieurwesen

Institut für Elektrotechnik und Informationstechnik
Lehrstuhl Kommunikationstechnik

eingereicht von: Robert Härtel
geboren am: 10.02.1990, Frankfurt (Oder)

Markus Marks
30.10.1989, Berlin

Betreuung: Dipl.-Ing. Christian Richter

eingereicht am: 06.08.2012

Übersicht

Im Rahmen dieser Arbeit wird eine LED Statusanzeige entworfen. Diese ist ein Teil des neu entwickelten „Speech LAB“ an der Brandenburgischen Technischen Universität in Cottbus. Das Labor ist mit 64 Mikrofonen und mehreren Monitoren ausgestattet, um an Spracherkennung zu forschen und diese weiterzuentwickeln. Zu diesen 64 Mikrofonen, die sich auf zwei getrennten Feldern verteilen, soll jedes Mikrofon jeweils mit einer RGB-LED ausgestattet werden, um gewisse Betriebszustände, auf die später noch genauer eingegangen wird, zu visualisieren. Außerdem soll jede LED von einem PC aus gesteuert werden und einen Farbbereich von 256 verschiedenen Farben abdecken.
Anhand dieser Aufgabe stellen sich nun folgende Teilaufgaben:

- **Theoretisches Wissen**

  In diesem Teil der Arbeit wird der theoretische Hintergrund etwas näher beleuchtet, um so mehr Wissen zu diesem Thema zu liefern.

- **Konzeption der Schaltung**

  Dieser Abschnitt beschäftigt sich mit der Frage, welcher Mikrocontroller zur Anwendung kommen soll und mit welchen Bauteilen die Platine bestückt werden muss. Außerdem wird erklärt, wie der Schaltplan und das dazugehörige Layout der Treiberplatine entwickelt werden. Zudem gibt es eine kurze Erklärung, wie das Layout des Schaltplans aus einem Programm zu einer fertigen und einsatzbereiten Platine entwickelt wird. Im Anschluss beginnt die Montage und der Test der entworfenen Platine.

- **Programmierung des Mikrocontrollers**

  In diesem Abschnitt wird erklärt, welche Programmiersprachen zum Einsatz kommen und weshalb diese von Vorteil für die Problemstellung sind. Außerdem wird eine Schaltoberfläche für Testzwecke entworfen und diese mit der Platine verbunden und getestet

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Selbstständigkeitserklärung

Eidesstattliche Erklärung

Die Verfasser erklären an Eides statt, dass sie die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als die angegebenen Hilfsmittel angefertigt haben. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Cottbus, den 06.08.2012

Theoretischer Teil

1 Aufbau und Funktionsweise einer Diode/Leuchtdiode

1.1 Theoretische Grundlage und Prinzip einer Diode

Die Diode zählt zu den Halbleiterbauelementen und besitzt eine Anode (p-dotiert) sowie eine Kathode (n-dotiert). Diese findet oft ihren Einsatz in Schaltungen, in denen Gleichrichter, elektronische Schalter oder aber auch Spannungsstabilisierungen benötigt werden.

Eine Diode funktioniert nach dem Prinzip eines pn-Überganges. Dabei werden an die Anode und der dazugehörigen Kathode eine Spannung angelegt. Ist die Diode in Durchlassrichtung geschaltet, so wird diese ab einer bestimmten Spannung leitend. Bei einer handelsüblichen Siliziumdiode liegt diese Spannung bei ca. 0,7 V.

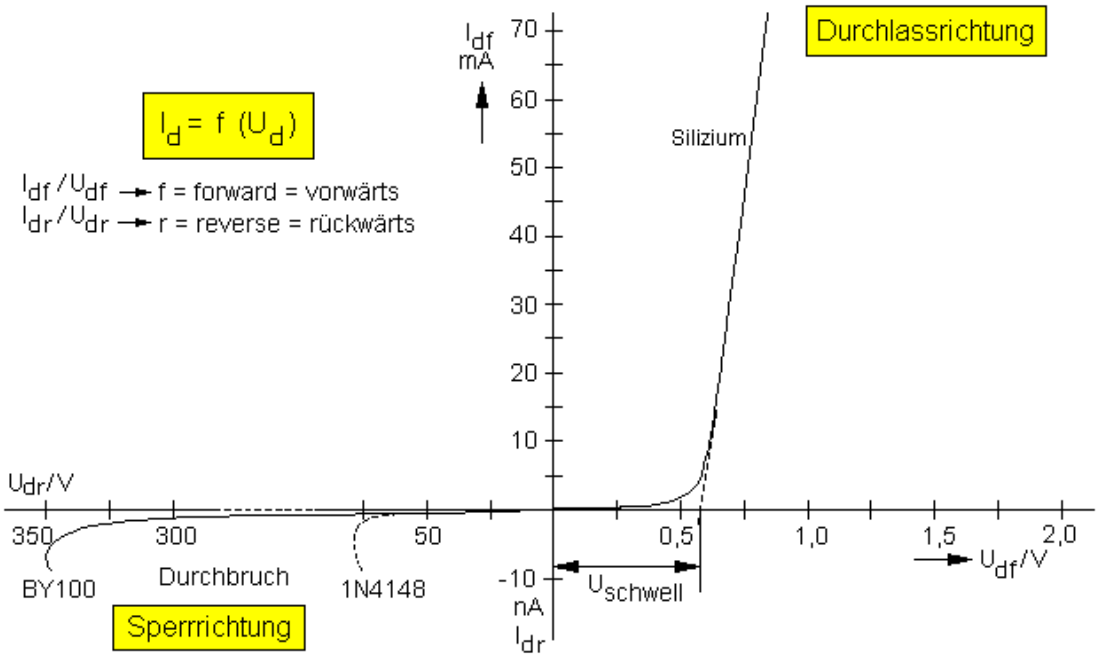


Abbildung 1.1.1: Kennlinie einer Siliziumdiode [11]

Liegt die angelegt Spannung unter diesen 0,7 V oder ist in Sperrrichtung eingebaut, so sperrt die Diode und es entsteht somit ein sehr hoher Widerstand. Ein zweites Problem kann außerdem entstehen, wie man aus der Grafik entnehmen kann, dass, wenn die angelegte Spannung zu hoch ist, ein exponentiell großer Strom über diese Diode fließt. Dies führt zu einer zu hohen Leistung, die an der Diode abfällt und es folgt die thermische Zerstörung.

Der Grund weshalb eine Diode aber nur so funktionieren kann, liegt an der Tatsache, dass zwischen dem p-dotierten Gebiet und dem n-dotierten Gebiet ein pn-Übergang stattfindet.

Dies lässt sich wie folgt erklären: Vorausgesetzt die Diode ist in Durchlassrichtung geschaltet (p-Schicht an Plus und n-Schicht an Minus) und es liegt eine äußere Spannung $U_{pn} > 0$ V an, so gelangen Elektronen bzw. Löcher durch die Raumladungszone (neutrale Zone, an der sich Elektronen und Löcher gegenüberstehen und sich gegenseitig neutralisieren → Sperrschicht entsteht), wo sich diese mit den dortigen Majoritätsträgern rekombinieren. Diese Majoritätsträger werden dann aus den neutralen Gebieten nachgeliefert, was einem Stromfluss I_D entspricht. „Es lässt sich also sagen, desto größer die Spannung U_{pn} ist, desto mehr Ladungsträger diffundieren über den Übergang und umso größer ist der dazugehörige Strom I_D." [1, S. 45ff]

1.2 Aufbau und Funktionsweise einer Leuchtdiode

Leuchtdioden oder auch LEDs („Light Emitting Diode" in deutsch „Licht emittierende Diode") sind für sich normale Halbleiterdioden. Diese haben allerdings die Besonderheit, wie der Name schon sagt, in Durchlassrichtung Licht zu erzeugen. Auch diese Art von Diode besitzt eine Anode sowie eine Kathode, die man leicht an den unterschiedlich langen „Beinen" erkennt (ausgenommen Sonderbauarten wie zum Beispiel SMD-LEDs). Die lange Seite spiegelt dabei die Anode wieder und die kurze Seite der LED-Beine somit die Kathode. [2]

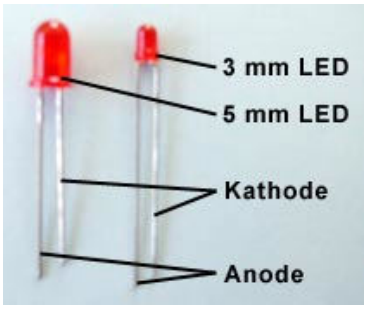


Abbildung 1.2.1: Abbildung einer Leuchtdiode [2]

Die Funktionsweise der LED ist vom Aufbau also relativ identisch. Es gibt genauso ein n- und ein p-Gebiet. Der Unterschied ist nun, dass das n-leitende Gebiet die Basis- bzw. Trägerschicht darstellt. Auf diese Weise wird nun das p-leitende Halbleitermaterial aufgebracht. Es entsteht auch hier der gleiche Effekt wie bei einer normalen Diode. Es bildet sich eine Raumladungszone. Wird nun eine Spannung an Anode und Kathode angelegt, so rekombinieren diese genauso mit den Löchern und es entsteht ein Ladungsträgertransport, welcher sich im Strom I_D wiederspiegelt. Bei dieser Rekombination entsteht Energie in Form von Lichtblitzen. Da bei LEDs das p-Gebiet sehr dünn ist, kann das Licht durch diese Schicht entweichen und ist auch schon bei einem Strom von wenigen Milliampere wahrnehmbar. Außerdem lässt sich die Lichtstärke mit dem Strom I_D proportional ins Verhältnis setzen. [2]

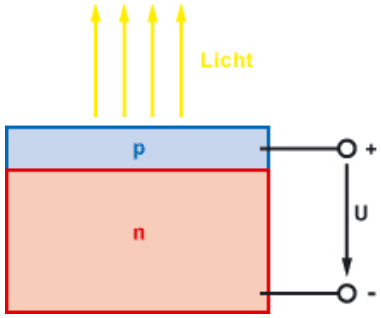


Abbildung 1.2.2: Schematischer Aufbau einer Leuchtdiode [2]

Ein weiteres wichtiges Merkmal für eine Leuchtdiode ist die Farbe, in der sie leuchten soll. Dies lässt sich mit verschiedenen Halbleitermaterialien verändern. Um etwa die Farbe Grün (Wellenlänge: 500 nm bis 570 nm) zu erreichen, benötigt man das Halbleitermaterial Galliumphosphid oder aber Silliziumkarbid, um die Farbe Blau (Wellenlänge: 450nm bis 500nm) zu realisieren. [2]

1.3 Arten und Einsatzgebiete verschiedener Leuchtdioden

Das Einsatzgebiet von Leuchtdioden ist sehr weitreichend. Aus kaum einer alltäglichen Situation sind sie heute noch wegzudenken. Ob es die Nadelbeleuchtung des Tachos im Auto ist oder die Statusanzeige der Kaffeemaschine. Für all diese Situationen gibt es speziell angefertigte Leuchtdioden in sämtlichen Formen und Farben. Auch Leuchtdioden mit Farbmischung sind heutzutage in der Herstellung kein Problem mehr. In diesem Beispiel werden die Grundfarben Rot, Grün und Blau in einem Gehäuse untergebracht. Je nach gewünschter Farbe werden dann die einzelnen LEDs unterschiedlich stark belastet. Es entstehen also verschiedenste Farbmischungen. Der Vorteil dieser Leuchtdioden ist dann ganz klar die Platzreduzierung, die eine solche Diode beansprucht. Außerdem lassen sich somit wesentlich mehr Informationen ablesen im Gegensatz zu drei herkömmlichen LEDs. Aus diesem Grund werden diese sogenannten RGB-LEDs für das Speechlab verwendet.

2 Das Kondensatormikrofon

2.1 Allgemeine Informationen zum Kondensatormikrofon

Das Kondensatormikrofon ist im Stande, akustische Schwingungen mit Hilfe einer Versorgungsspannung und einer Verstärkerschaltung in analoge Signale zu wandeln. Dabei haben Kondensatormikrofone zwei entscheidende Vorteile. Zum einen können sie durch ihr Verfahren bis hin zu sehr hohen Frequenzbereichen im kHz Bereich die Schwingungen sehr detailtreu und impulsgenau wandeln. Zum anderen können sie wie im Falle von Kleinmembranmikrofonen sehr kleine Abmaße und ein geringes Gewicht aufweisen. Somit sind Kondensatormikrofone in vielen Einsatzbereichen einsetzbar.
Auf der anderen Seite hat das Kondensatormikrofon aber auch einen großen Nachteil. Es benötigt für seine Funktionsweise eine Versorgungsspannung von 48 V. Diese kann vereinzelt unter Umständen zu Störungen führen. Diese Spannung ist aber notwendig, da sonst diese Art von Mikrofon nicht funktionieren kann. [13, 14]

2.2 Funktionsweise eines Kondensatormikrofons

Ein Kondensatormikrofon besteht grob gesagt aus fünf Teilen. Dazu gehören eine Membran, die aus einer Metallfolie oder einer metallisierten Kunststofffolie besteht, einer dazugehörigen Gegenelektrode, einer Spannungsversorgung, einem Mikrofonverstärker und einem sehr hochohmigen Widerstand.
Das Prinzip ist nun folgendes: Tritt der Schall auf die Mikrofonkapsel, so gelangt dieser auf die gespannte Membran. Diese wird nun in Schwingung versetzt. In Verbindung mit der ebenfalls aus Metall bestehenden Gegenelektrode und dem dazwischenliegenden Isolator hat die Membran eine Wirkung eines Plattenkondensators. Damit nun eine akustoelektrische Wandlung erfolgen kann, muss zuvor der Kondensator mit Hilfe einer Phantomspannung über einen großen Widerstand aufgeladen werden. Die Phantomspannung stellt dabei eine 48 Volt große Gleichspannung dar, die meist aus einem Mischpult in das Mikrofon eingespeist wird, an dem das Mikrofon angeschlossen ist. „Der hochohmige Widerstand hat hingegen die Aufgabe, während der Schallanregung den Ladungsausgleich über die Spannungsquelle zu verhindern.“ [13]
Dabei lässt sich nun sagen, dass, wenn sich der Plattenabstand eines Plattenkondensators verändert, sich auch die Spannung des Kondensators proportional dazu ändert.

$$U = \frac{Q}{C} \tag{2.2.1}$$

$$C = \varepsilon_0 \frac{A}{d} \tag{2.2.2}$$

$$U = \frac{Q}{\varepsilon_0 \frac{A}{d}} \tag{2.2.3}$$

Unter der Annahme, dass die Ladung Q und die Fläche A der Membran konstant ist und der Abstand d zwischen der Membran und der Gegenelektrode steigt, folgt ein proportionaler Anstieg der Spannung:

$$U \uparrow = \frac{Q}{\varepsilon_0 \frac{A}{d \uparrow}} \tag{2.2.4}$$

Da diese Spannungsänderung aber nur sehr gering ist, muss eine Verstärkerschaltung nachgeschaltet werden. Dazu bietet sich eine Sourceschaltung mit einem NMOS Transistor an, da diese einen hohen Eingangswiderstand besitzt und somit die Quelle nicht belastet. Deshalb kann das Signal mit einer Impedanz von ca. 200 Ohm an das Mischpult weitergegeben werden. [13, 14]

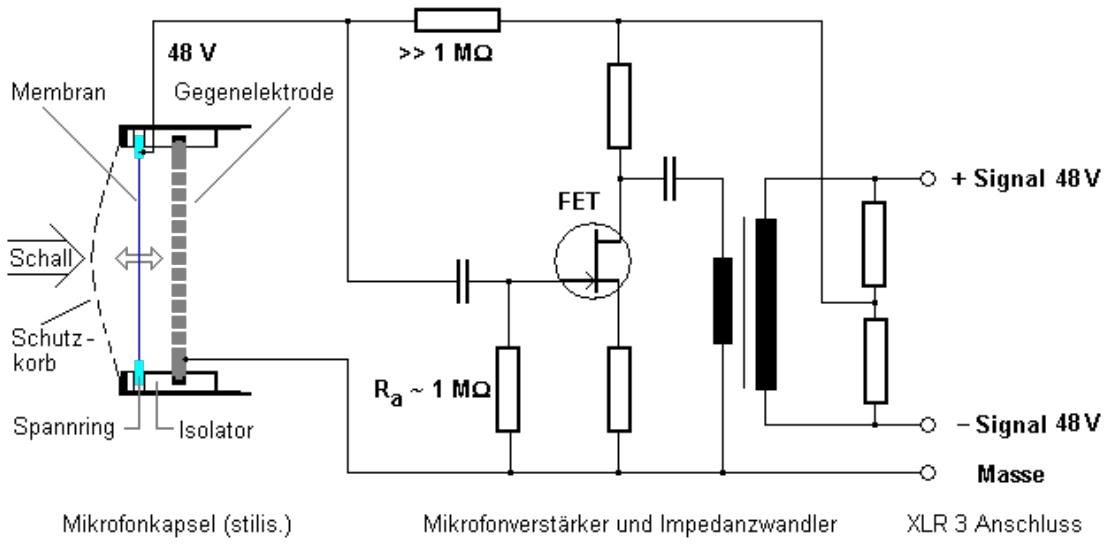


Abbildung 2.2.1: Schalplan eines Kondensatormikrofons [13]

3 Delay- und Sum-Beam-Forming

3.1 Schallausbreitung

Ein Schall breitet sich immer gleichförmig als eine Welle nach allen Seiten in einem homogenen Medium aus. Dabei wird das Medium zum Schwingen angeregt, zum Beispiel durch die Membran eines Lautsprechers, Stimmbänder im Kehlkopf oder einer Saite eines Musikinstruments (z.B. Gitarre). Die Ausbreitungsgeschwindigkeit des Schalls ist abhängig vom Medium. Somit breitet sich der Schall in Luft mit einer Geschwindigkeit von 343 Meter pro Sekunde aus. Die Intensität des Schalls oder auch Schalldruckpegel nimmt mit zunehmendem Abstand der Schallquelle ab. Um den Schalldruckpegel zu messen, ist das Speech Lab mit 64 Mikrofonen ausgestattet. Damit aber eine optimale Messung erfolgen kann, muss gewährleistet sein, dass der Schall an den Wänden nicht reflektiert wird. Sonst würde es zu einer Überlagerung von Schallwellen kommen. Aus diesem Grund sind die Wände mit Schaumstoffmatten verkleidet und der Fußboden ist mit einem speziellen Teppich ausgelegt. Außerdem müssen auch die Fenster mit schallabsorbierenden Vorhängen abgedeckt werden, da auch dort sonst Reflektionen entstehen können.
Der Absorbtionsgrad der schallisolierenden Stoffe ist abhängig von den Frequenzen des Schalls. Ein Schall mit hohen Frequenzen wird mit fast 100% absorbiert. Im Gegensatz zu niedrigen Frequenzen können die Schaumstoffmatten nur bis zu etwa 50% des Schalls aufnehmen. [15, 16]

Material	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz
Beton	1%	1%	1%	1,5%	2%	3%
Holzboden	15%	11%	10%	7%	6%	7%
Holzverkleidung mit Schaumstoff Unterkonstruktion	30%	25%	20%	17%	15%	10%
Nadelfilz	5%	8%	20%	30%	35%	40%
Vorhänge	5%	10%	25%	30%	40%	50%
Akustik Noppenplatte 4 cm	10%	20%	55%	80%	85%	85%
Absorber aus 10 cm PUR Schaumstoffmatten	35%	45%	90%	95%	98%	98%
Verbundplattenabsorber 20 cm Gesamtschichtdicke	55%	70%	98%	98%	98%	98%

Tabelle 3.1.1: Beispiele für Luftschall – Absorptionsgrad [16]

3.2 Messung der Schallwellen

Das Speech Lab wird mit zwei Mikrofonarrays mit jeweils 32 Mikrofonen ausgestattet. Mit diesen beiden Arrays ist es möglich, Schallquellen zu orten und deren Position zu bestimmen. Das Orten der Schallquelle ist abhängig von den Array-Eigenschaften, wie der Array-Größe, Anzahl und Anordnung der Mikrofone. Die aufgenommen Schallwellen werden durch Beam-Forming-Algorithmen ausgewertet. Dabei ist darauf zu achten, dass die Schallquelle nicht immer direkt vor den Mikrofonen platziert wird. Somit ist die Weglänge zwischen Schallquelle und den einzelnen Mikrofonen nicht immer gleich und es ergeben sich Phasenunterschiede bei den aufgenommen Zeitsignalen. Die Abbildung 3.2.1 zeigt, dass die Signale mit unterschiedlicher Verzögerung am Summierer ankommen. Wird dieses Phänomen nicht berücksichtigt, kann es vorkommen, dass sich die Signale falsch überlagern oder sich auslöschen. Somit kommt es zu einer falschen Auswertung, wodurch die Ortung des Schallobjekts nicht mehr möglich ist. [3]

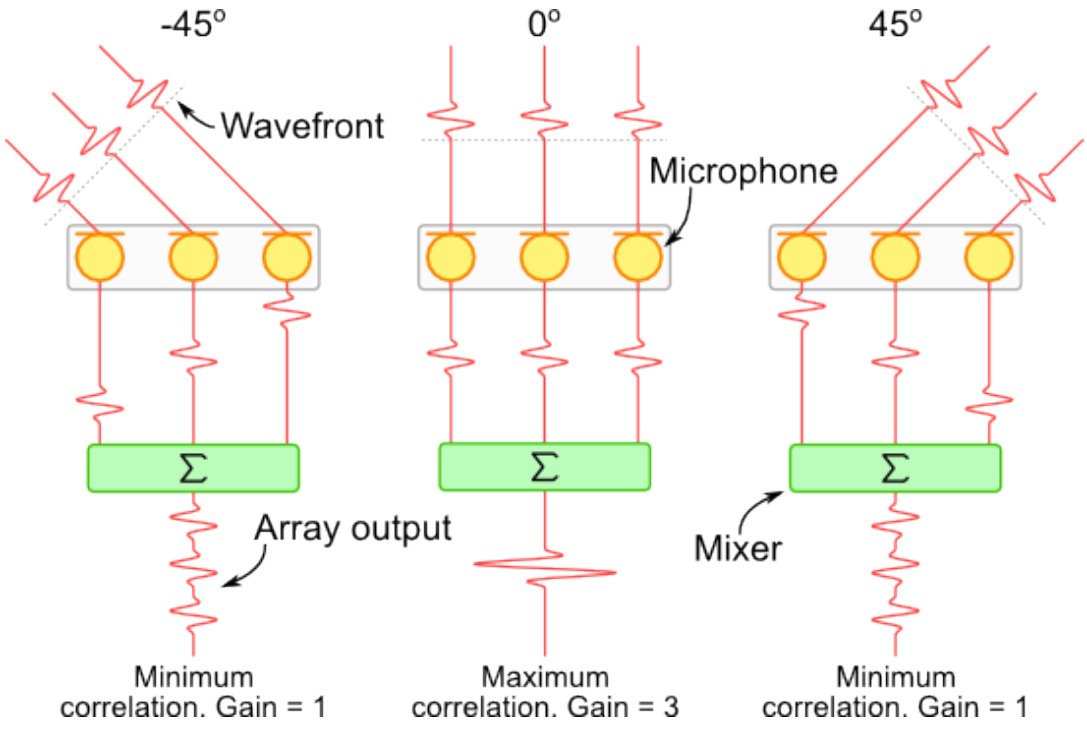


Abbildung 3.2.1: Verzögerung der Signallaufzeiten [3]

In der Abbildung 3.2.1 werden die Amplitudenunterschiede nicht berücksichtigt. Das Beispiel in der Abbildung 3.2.2 zeigt jeweils, wie die Signale unterschiedlich empfangen werden. Mit zunehmender Entfernung der Schallquelle, nimmt die Phasenverschiebung des Signals zu und die Amplitude wird kleiner. Diese Unterschiede müssen bei der Korrektur berücksichtigt werden. Wenn alle Signale richtig analysiert werden, entsteht das gleiche Signal an jedem Mikrofonausgang, die dann miteinander addiert werden können. Dadurch wird die empfangende Schallwelle verstärkt.
Andererseits ist es möglich, dass die aufgenommen Schallwellensignale sich gegenphasig überlagern. Die Interferenz der Signale kann zum Auslöschen führen. [3]

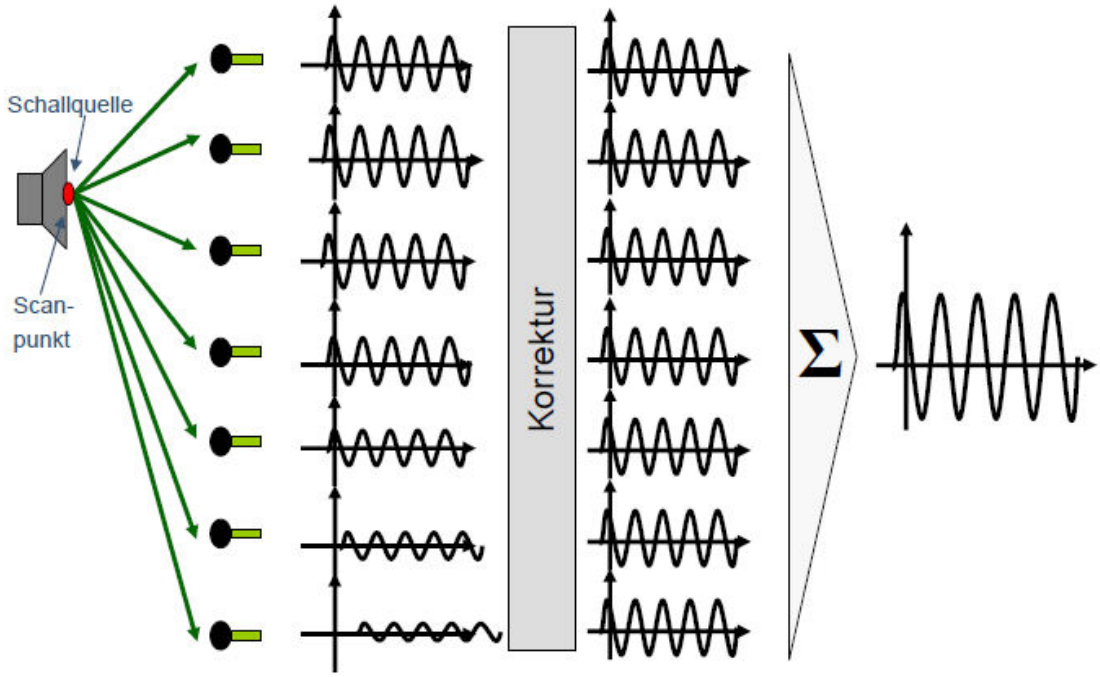


Abbildung 3.2.2: Verzögerung der Signallaufzeiten unter Berücksichtigung der Amplituden [5]

Dieses Verfahren nennt man Time-Delay-und-Sum-Beam-Forming. Um ein Beam-Forming-Algorithmus aufzustellen, werden vorher verschiedene Schallquellen mit einem definierten Abstand zum Array ermittelt. Dadurch können die Phasen- und Amplitudenunterschiede ausgewertet und analysiert werden. Dies wird in Abbildung 3.2.3 in einem Blockschaltbild dargestellt:

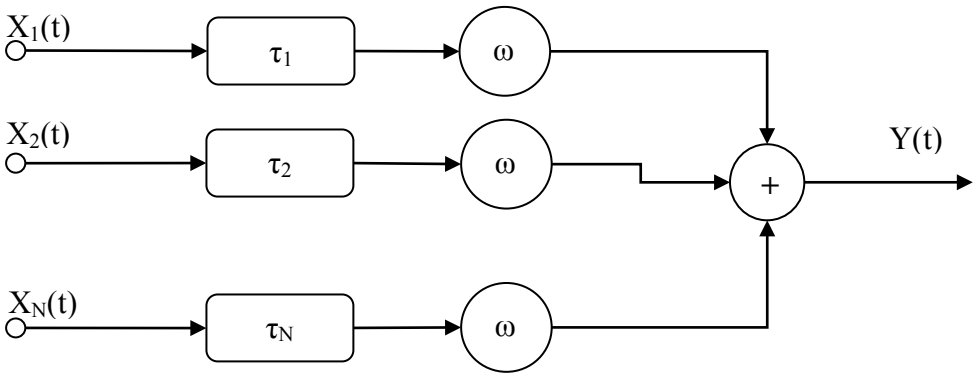


Abbildung 3.2.3: Blockschaltbild eines Time-Delay-und-Sum-Beam-Formers [4]

Das aufgenommene Zeitsignal des i-ten Mikrofones (i = 1, 2 bis N) wird durch die Funktion X_i beschrieben. ω_i ist die dazugehörige Amplitudengewichtungsfunktion. Die zeitlichen Verschiebungen zwischen den Signalen werden mit τ_i dargestellt. Die einzelnen Signale werden miteinander summiert. [4]

$$Y(t) = \sum_{i=1}^{N} \omega_i \cdot X_i(t - \tau_i) \tag{3.1}$$

Somit lässt sich eine Schallquellenlokalisation und Bewertung realisieren, das auch im Speech Lab zur Anwendung kommt.

3.3 Ausrichtung der Mikrofone

Das Ausrichten der Mikrofone zu einer Schallquelle kann nur realisiert werden, wenn mindestens zwei Mikrofone vorhanden sind. Im Speech Lab gibt es zwei Mikrofonarrays mit jeweils 32 Mikrofonen. Dadurch, dass das eine Mikrofonarray am Wandmonitor angebracht ist und die zweite an der Decke, ist es möglich, die Schallquelle in allen drei Dimensionen zu orten. Die Verzögerungszeit ist vom Winkel der eintreffenden Schallwellen abhängig und kann am folgenden Beispiel berechnet werden.

Beispielrechnung:
Es sind zwei Mikrofone gegeben, die einen Abstand von 0,25 m haben. Die maximale Frequenz des Eingangssignals beträgt 24 kHz. Im Speech Lab werden die aufgenommen Signale der Mikrofone in digitale Signale umgewandelt. Das zeitkontinuierliche Signal sollte nach dem Nyquist-Theorem abgetastet werden. Das bedeutet, die Abtastfrequenz muss doppelt so groß sein wie die maximale Eingangsfrequenz. In diesem Beispiel beträgt die Abtastfrequenz 48 kHz.
Trifft die Schallwelle in einem 45° Winkel auf die Mikrofone, so lässt sich die Delay-Zeit wie folgt berechnen. [3]

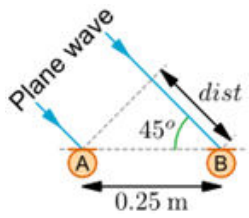


Abbildung 3.3.1: Eintrittswinkel der Schallwelle auf die Mikrofone [3]

Wenn die Schallwelle am Mikrofon „A“ auftrifft, muss die Distanz berechnet werden, die die Schallwelle zurücklegen muss, um am Mikrofon „B“ anzukommen. Diese Berechnung erfolgt mit folgender Formel:

$$dist = 0{,}25\ m\ \cdot \cos(45°) = 0{,}1768\ m \tag{3.2}$$

Damit die Signale von den beiden Mikrofonen gleichzeitig am Summierer anliegen, lässt sich die Verzögerungszeit des Signals vom Mikrofon „A“ folgendermaßen berechnen:

$$Delay = \frac{dist}{V} = \frac{0{,}1768\ m}{343{,}3\ \frac{m}{s}} = 0{,}515\ ms \tag{3.3}$$

Um diese Verzögerungszeit zu realisieren, muss eine Anzahl von Taktperioden bei einer Frequenz von 48 kHz abgewartet werden.

$$Anzahl\ Taktperioden = \frac{48000\ Hz\ \cdot 0{,}25\ m\ \cdot \cos\,(45°)}{343{,}3\ \frac{m}{s}} = 24{,}717 \tag{3.4}$$

In der Abbildung 3.3.2 ist erkennbar, dass das Signal vom Mikrofon A zuerst am Summierer ankommt und es somit verzögert wird.

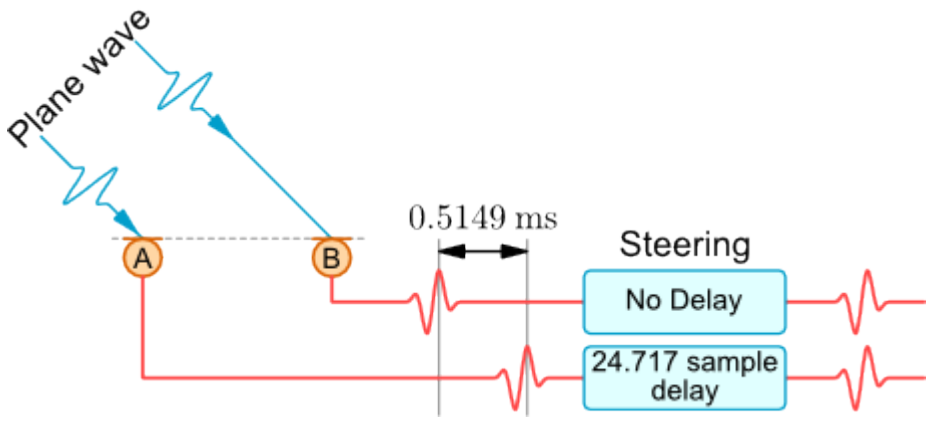


Abbildung 3.3.2: Visualisierung der Verzögerungszeiten durch Taktperioden [3]

Durch die einzelnen Verzögerungszeiten der Mikrofone lässt sich die Ausrichtig der Mikrofone in einem Polardiagramm darstellen.
In der Abbildung 3.3.3 ist der Vergleich von einem Mikrofon und einem Mikrofonarray erkennbar. Somit ist es möglich, den Focus der Mikrofone in eine Richtung zu lenken und die Störsignale herauszufiltern.

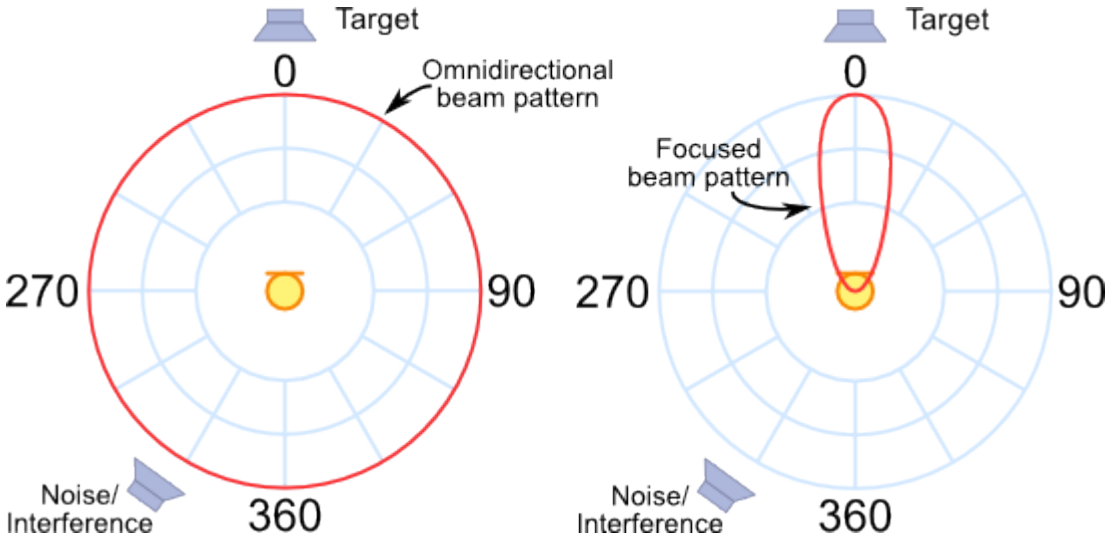


Abbildung 3.3.3: Ausrichtung der Mikrofone in einem Polardiagramm [3]

Die Ausrichtung der Hauptkeule des Mikrofonarrays lässt sich durch die Variation der Verzögerungszeiten der Phasen verändern. Mit zwei Mikrofonen ist es also nicht genau möglich, die Schallquelle im Raum zu lokalisieren. Demnach ist mit zwei Mikrofonen nur ersichtlich, ob sich die zu ortende Schallquelle links oder rechts der beiden Mikrofone befindet. Wird ein zweites Mikrofonpaar im Raum angebracht, ist es nun

möglich, durch deren Ausrichtung ihrer Hauptkeulen und deren resultierenden Schnittpunkt die Schallquelle zu orten. Aus diesem Grund werden im Speech Lab zwei Arrays mit je 32 Mikrofonen verwendet, um so die Schallquelle möglichst genau zu orten. [3]

4 Konzept zur Visualisierung diverser Parameter über das LED Array

4.1 Wozu wird ein LED Array in einem Mikrofonfeld verwendet?

Ein LED Array dient zum Visualisieren verschiedener Parameter in einem Mikrofonfeld. Aus diesem Grund wird an jedem Mikrofon eine RGB-LED verbaut. Diese sollte möglichst nah an dem jeweiligen Mikrofon liegen. Nur so kann erkannt werden, welches Mikrofon in diesem Array angesprochen wird. Durch ein solches Feld ist es möglich, eine Schallquelle punktgenau zu lokalisieren und mit Hilfe des LED Arrays farblich zu markieren und zu visualisieren.

4.2 Konzepte der Visualisierung

4.2.1 Visualisierung der Amplitudenwerte des Eingangssignals des Mikrofons

Da in diesem Mikrofonfeld Hochleistungs-RGB-Leuchtdioden verbaut werden, lassen sich dementsprechend 256 Farben pro LED anzeigen. Diese können im Modus „Amplitude“, 10 mal pro Sekunde den aktuellen Wert des Lautstärkepegels anhand eines Farbwertes wiedergeben. Es wäre außerdem möglich, falls das Mikrofon nicht angesprochen wird, keine Helligkeit auszustrahlen. Sollte dann ein definierter Grenzwert überschritten werden, so soll die LED hellgrün zu leuchten beginnen. Nimmt das Mikrofon eine höhere Lautstärke wahr, so soll der Farbverlauf langsam von Grün über Gelb zu einem satten Rot wechseln. Allerdings muss demzufolge ein Maximalwert festgelegt werden, bei dem die LED ihr Maximum an Leuchtkraft erreichen soll. Es ist also wichtig, dass im Modus „Amplitude“ ein Grenzwert für das anfängliche Leuchten der LED eingestellt wird sowie ein Maximalwert, bei dem die LED ihr hellstes Rot ausstrahlt und somit ein Maximum an erreichtem Lautstärkepegel symbolisiert. Da diese Werte meist von der Messaufgabe abhängen, sollten diese Werte gegebenenfalls direkt in der Software unter diesem gewählten Modus einzugeben sein. So ist es also möglich, das Array möglichst gut auf die zu messende Aufgabe einzurichten.

4.2.2 Visualisierung des Betriebszustandes des Mikrofons

Um die Betriebszustände zu visualisieren, ist es in erster Linie wichtig zu wissen, welche Betriebszustände für ein Mikrofon relevant sind. Zum einen muss erkenntlich sein, ob das Mikrofon eingeschaltet oder ausgeschaltet ist. Außerdem wäre es von Nutzen, ein oder mehrere Mikrofone auf Standby zu schalten, falls man nicht das gesamte Array auswerten möchte. Für diese 3 Betriebszustände bieten sich die 3 Grundfarben der LED an, um im Modus „Betriebszustand“ eine bestmögliche Visualisierung ohne Farbmischung zu gewährleisten.

Betriebszustand	**Farbe**
Aus	Rot
An	Grün
Standby	Blau

Tabelle 4.2.2.1: Betriebszustände der LED’s

4.2.3 Visualisierung der Delaytime (Verzögerungszeit)

Wird eine Schallquelle im Raum erkannt, wird an jedem Mikrofon der Schallpegel mit einer bestimmten Verzögerungszeit und Amplitude wahrgenommen. Diese Verzögerungszeiten kann man nun mit Hilfe der RGB-LEDs darstellen. Somit kann der Benutzer später erkennen, welcher Punkt im Raum fokussiert wird. Die Mikrofone, die Signale mit hohen Amplituden aufnehmen, kann man den Farbewert grün zuordnen. Unterschreitet die Amplitude des Signalpegels einen bestimmten Wert, den man vorher in der Programmierung festlegen muss, ändert sich die Farbe der RGB-LED langsam zu gelb. Die Mikrofone, die noch weiter von der Schallquelle entfernt sind, nehmen ein Signal mit einem noch kleineren Amplitudenwert auf, der dann mit einem rötlichen Farbton mit der RGB-LED dargestellt werden kann. Durch diese farblichen Abstufungen, lässt sich erkennen, wie sich der Schall ausbreitet und gleichzeitig die Amplitude über die Entfernung von der Schallquelle zum Mikrofon abnimmt.
Eine weitere Möglichkeit zur Visualisierung der Amplitudenwerte der aufgenommenen Schallwellen kann mit Hilfe eines Blinkzustandes der RGB-LEDs erfolgen. Eine hohe Blinkfrequenz lässt auf einen großen Amplitudenwert schließen bzw. man kehrt diese Funktion um.

4.2.4 Visualisierung der Verstärkung eines Mikrofons

Soll die Verstärkung eines Mikrofons visualisiert werden, so ist es wichtig, genau zu wissen, aus welchem Grund dies geschehen soll. Eine sinnvolle Anwendung im Bereich Spracherkennung ist es, alle Mikrofone in einem Array so zu verstärken, dass sie den gleichen Ausgangspegel haben. Diese Verstärkung muss in einen Faktor umgewandelt werden, mit dem der Eingangspegel multipliziert werden muss, sodass wie schon beschrieben, alle Mikrofone eines Arrays den gleichen Ausgangspegel erreichen. Dieser Faktor lässt sich nun leicht durch einen Farbwert an dem LED Array ausgeben. Auch hier wäre es wieder sinnvoll, von der Verstärkung gleich eins (Grün) zu einem Maximalwert einen Farbverlauf zu realisieren, der über die Farben Gelb bis hin zu Rot erreicht wird.

Praktischer Teil

5 Konzeption der Schaltung

5.1 Entwurf der Ansteuerung

5.1.1 Wahl des Bussystems

Für die Ansteuerung der LEDs muss ein Mikrocontroller zum Einsatz kommen, der per USB an einen Computer angeschlossen wird. Da aber die Anzahl der zur Verfügung stehenden Pins an dem Mikrocontroller auf ca. 100 begrenzt ist und insgesamt 194 Pins benötigt werden, sind folgende Aspekte zu berücksichtigen:

Durch eine Vielzahl von möglichen Helligkeitsstufen einer LED können verschiedene Farben dargestellt werden. Es ist nicht möglich, mit Hilfe der Spannung die Helligkeit der LED zu verändern. Durch die Strom-Spannungs-Kennlinie wird beschrieben, wie ein Verbraucher auf eine angelegte Spannung reagiert. Bei einem ohmschen Verbraucher nimmt der Strom linear mit der Spannung zu. Leuchtdioden besitzen eine exponentielle Kennlinie. Kleine Schwankungen in der Spannung verursachen große Stromänderungen. Aus diesem Grund ist es schwierig, die Helligkeit über die Spannung zu regeln. Somit ist es besser, die LED über eine bestimmte Zeit mit einer Anzahl von Impulsen zu versorgen.

Dieses Verfahren nennt man Pulsweitenmodulation (PWM). Hierbei wird das generierte Signal immer mit der gleichen Periodendauer erzeugt und nur lediglich das Verhältnis der Ein- und Ausschaltzeit pro Impuls wird verändert.

Ein PWM-Signal wird durch einen Zähler generiert. Die Impulsdauer wird bei einer Acht-Bit Auflösung in 256 Schritte unterteilt. Am Anfang wird das Signal auf eins gesetzt. Erreicht der Zähler einen vorher eingestellten Wert, der dem Helligkeitswert entspricht, wird der Impuls auf null gesetzt. Der Impuls bleibt null, bis der Zähler den vorgegebenen Endwert erreicht hat. Dann wird das Signal wieder auf eins gesetzt und der Zähler beginnt von vorn. Somit beginnt die Prozedur erneut. In der Abbildung 5.1.1.1 sind verschiedene PWM-Signale dargestellt. [6, 7]

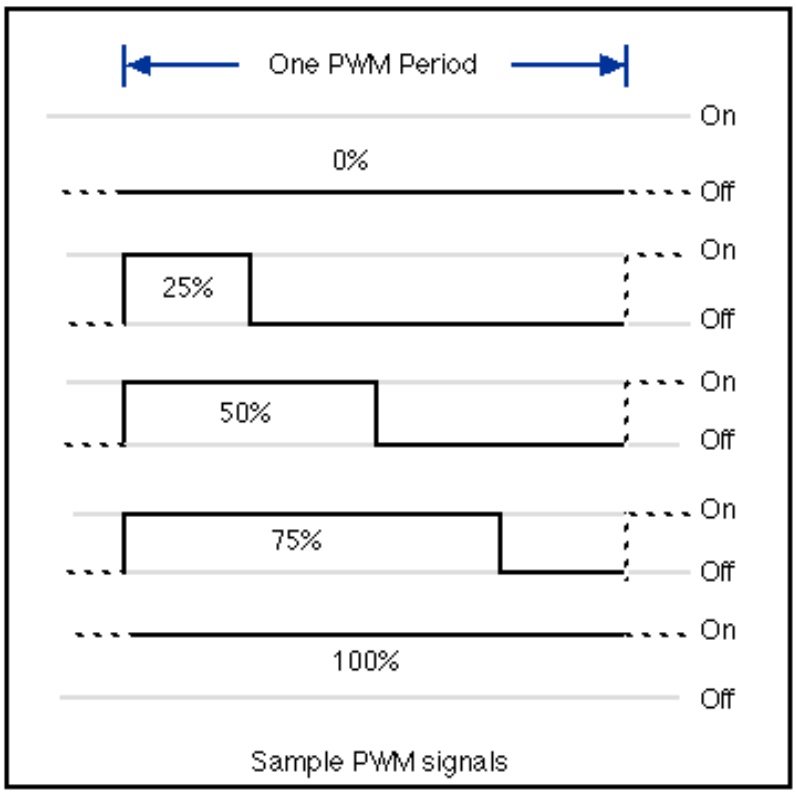


Abbildung 5.1.1.1: PWM Signale [9]

Eine weitere Einschränkung des Mikrocontrollers ist die beschränkte Anzahl von integrierten Zählern. Nur mit einem hohen Programmieraufwand wäre es überhaupt erst möglich, an jeden Pin des Mikrocontrollers eine LED anschließen zu können, damit jede LED mit einem unterschiedlichen Helligkeitswert versorgt werden kann.
Da diese Dinge berücksichtigt werden müssen, muss eine Möglichkeit gefunden werden, um auf der einen Seite den Programmieraufwand in Grenzen zu halten und auf der anderen Seite den schaltungstechnischen Aufwand zu minimieren.
Deswegen gibt es spezielle Treiber, an denen man mehrere LEDs anschließen kann und jede mit einem unterschiedlichen PWM-Signal versorgen kann. Der Treiber ist über einen Bus mit Mikrocontroller verbunden und lässt sich steuern.
Durch Recherchen ist der I^2C Bus am besten geeignet, da dieser Bus als Standard in fast allen Mikrocontrollern vorhanden ist und er sich einfach programmieren lässt. Dieser Bus wurde Anfang der 90er Jahre von Philips entwickelt und ist heute in vielen Systemen in der Industrie zu finden. Der Grund dafür ist, dass alle Komponenten an einem Bus hängen, wodurch nur zwei Leiterbahnen zu jedem entsprechenden IC-Chip verlaufen und die Einsparung an Platinenfläche ein großer Vorteil ist. Dadurch entsteht eine bessere Übersicht auf der Platine und es lassen sich Kosten sparen.

Der Bus besteht aus zwei Leitungen: die Taktleitung (SCL) und der Datenleitung (SDA). Beide Leitungen sind mit einem Pullupwiderstand mit der Betriebsspannung verbunden. Da jeder Slave (Peripheriebaustein) mit diesen Leitungen verbunden ist, wie es in Abbildung 5.1.1.2 zu sehen ist, ergibt sich daraus eine Wired-And-Schaltung.
Es handelt sich hierbei um einen bidirektionalen Zweidraht-Bus mit einer Master/Slave Architektur und integriertem Übertragungsprotokoll. [6]

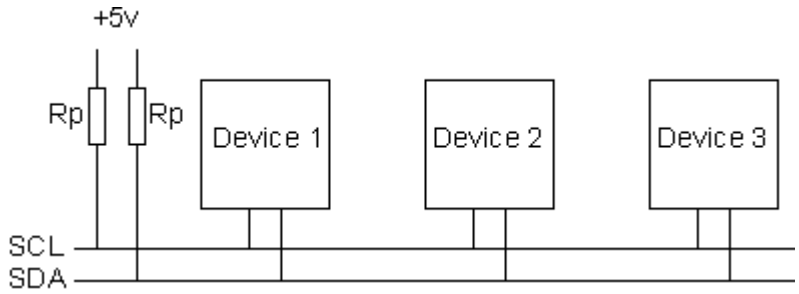


Abbildung 5.1.1.2: Aufbau des I^2C Bus [6]

Die Datenübertragung erfolgt bitseriell und synchron, da jedes Datenbit auf der Datenleitung mit dem Takt der Taktleitung synchron übertragen wird. Der Takt wird vom Master erzeugt. Dadurch muss nicht jeder Slave mit einem Takt generiert werden. Damit die Daten beim entsprechenden Slave ankommen, wird jedem Slave eine bestimmte Adresse zugeordnet, wodurch dieser einwandfrei am I^2C-Bus identifiziert werden kann.
Die beiden wichtigsten Signale am I^2C-Bus ist einmal die Startbedingung und die Stoppbedingung, wenn eine Übertragung gestartet oder abgeschlossen wird. Diese beiden Signale werden vom Master generiert. Dabei wird die Datenleitung auf das Lowpotenzial oder Highpotenzial gezogen, während die Taktleitung high bleibt.

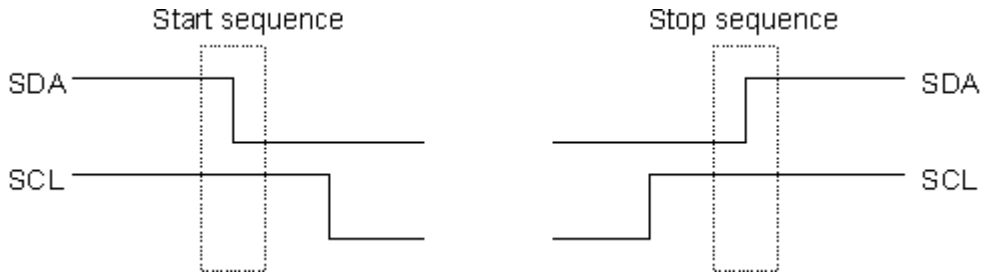


Abbildung 5.1.1.3: Start- und Stoppbedingung [6]

Wenn die Startbedingung erfolgt ist, wird vom Master die 7-bit Adresse vom Slave gesendet. Es werden dabei genau 8 Bit gesendet, da das achte Bit dem Slave mitteilt, ob eine Lese- oder Schreiboperation ansteht.
Hat der angesprochene Slave seine Adresse erkannt, wird ein Bestätigungsbit an den Master zurückgesendet.

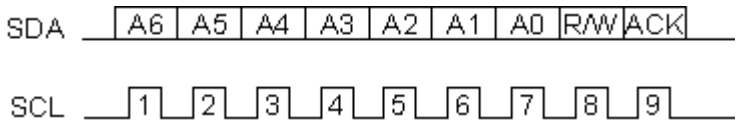


Abbildung 5.1.1.4: Übertragungsprotokoll der Adresse des Slaves [6]

Nach dem Empfang des Bestätigungsbits, kann der Master das nächste Byte senden. Der Slave quittiert dieses wieder mit einem Bestätigungsbit.

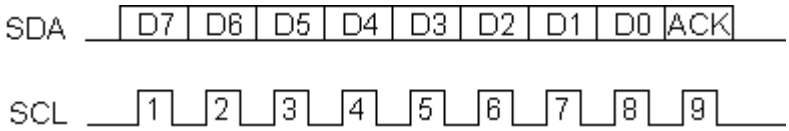


Abbildung 5.1.1.5: Übertragungsprotokoll eines Bytes [6]

Um die Übertragung abzuschließen, wird nun die Stoppbedingung durch den Master gesendet.
Eine wesentliche Einschränkung bei Einsatz dieses Bustyps ist die relativ geringe Reichweite. Ab einem Meter wird die zulässige Kapazität von 400 pF der Leitung überschritten. Die Folge ist, dass die Pullupwiderstände zu lange brauchen, um einen high-Zustand an den Busleitungen zu erzeugen. Als Folge daraus resultieren Timeout-Probleme, indessen auf der Datenleitung unbestimmte Zustände entstehen.
Um größere Reichweiten überbrücken zu können, ist der Einsatz von speziellen Treibern notwendig. Doch auch dann muss weiterhin mit Einschränkungen bei größeren Reichweiten in der Übertragungsgeschwindigkeit gerechnet werden. Da die LEDs in Echtzeit bestimmte Zustände darstellen sollen, ist der Einsatz von diesen speziellen Treibern keine Lösung. Als Lösungsansatz für dieses Problem werden zwei Mikrocontroller verwendet. Somit wird der jeweilige Mikrocontroller in der Nähe des Mikrofonarrays verbaut. Jeder Mikrocontroller steuert anschließend jeweils 32 RGB-

LEDs. Um die Schnittstelle zwischen Computer und Mikrocontroller zu bedienen, ist ein spezielles USB-Kabel, das mit einem Repeater ausgestattet ist, notwendig. Nur so ist es überhaupt erst möglich, größere Entfernungen (10m) störungsfrei zu überbrücken. [6, 12]

5.1.2 Wahl der Bauelemente

Bei der Wahl des LED-Treibers ist es wichtig, dass dieser über eine I^2C-Schnittstelle verfügt und möglichst viele LEDs ansteuern kann. Der TLC59116 von Texas Instruments entspricht diesen Voraussetzungen. Dieser LED-Treiber hat 16 Ausgänge und wird in SMD-Bauweise gefertigt. Der größte Vorteil ist, dass jeder Pin mit einem unterschiedlichen PWM-Signal mit einer 8-Bit Breite programmiert werden kann. In diesem Treiber ist ein interner 25 Mhz Oszillator eingebaut. Deshalb ist keine externe Taktversorgung notwendig. Bei Verwendung dieses Treibers ist darauf zu achten, dass die LEDs mit ihren Kathoden an dem Treiber angeschlossen werden. Aus diesem Grund wird eine RGB-LED mit einer gemeinsamen Anode benötigt. Der Mikrocontroller LPC1768 von NXP ist ein Cortex-M3 mit einer 32 Bit RISC Architektur. Er wird mit 96 Mhz getaktet und besitzt 512 kByte Flash Speicher und 32 kByte SRAM. Des Weiteren besitzt er auch zwei I^2C Schnittstellen. Diese Eigenschaften sind ausreichend, um dieses Projekt zu realisieren. Der Controller wird in der Programmiersprache C/C++ programmiert. In der Abbildung 5.1.2.1 sind die einzelnen Anschlüsse des verwendeten MBED-Boards, auf dem der Mikrocontroller aufgelötet ist, zu erkennen. [8, 10]

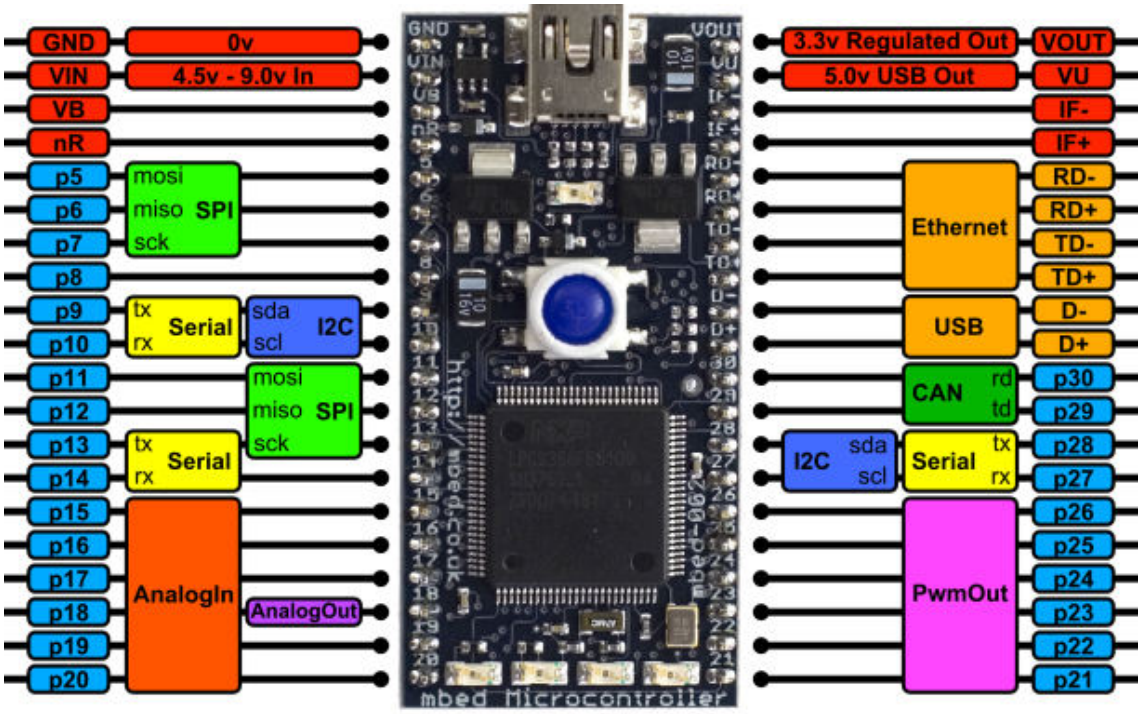


Abbildung 5.1.2.1: Pinbelegung des MBEB-Boards (mbed NXP LPC1768) [8]

5.2 Testphase der Platine und erste Schritte der Ansteuerung und Aufbau der Schaltung auf dem Steckboard

Der Entwurf des Schaltplans beginnt auf dem Steckbrett. Dabei wird die Schaltung provisorisch aufgebaut. Das Beispiel aus dem Datenblatt des LED-Treibers dient dabei als Vorlage.

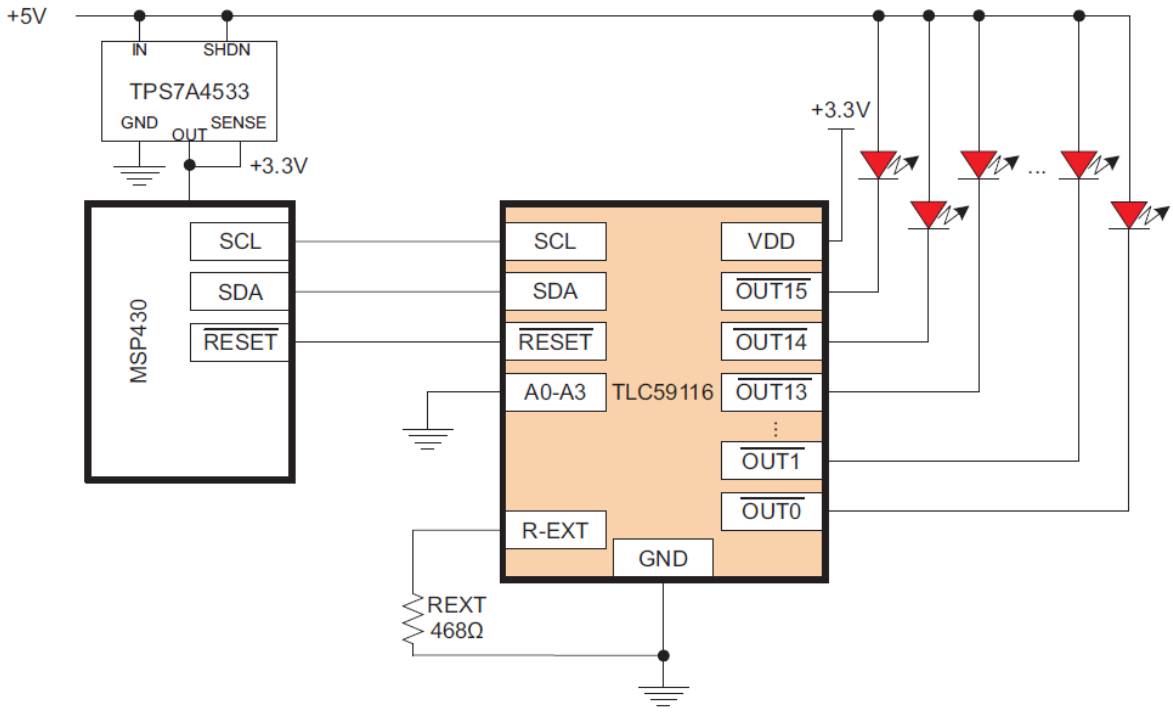


Abbildung 5.2.1.1: Schaltplan des LED-Treibers (TLC59116) [10]

Da der LED-Treiber nicht in DIP-Bauweise gefertigt ist, sondern als Thin-Shrink-Small-Outline-Package (TSSOP), ist er auf einem Breakoutboard aufgelötet, bei dem die Pins auf das normale Steckboardraster herausgeführt sind. Die Schaltung kann somit als Prototyp aufgebaut und getestet werden. Um sie zu testen, wird im ersten Schritt der Mikrocontroller programmiert. Dabei wird ein Testprogramm entworfen, das den LED-Treiber per I^2C-Schnittstelle anspricht. An die Ausgänge werden einige LEDs angeschlossen, um zu erkennen, ob der LED-Treiber richtig programmiert wurde. Nach mehreren Tests wurde dieser als tauglich empfunden. [10]

5.3 Ausarbeitung des Schaltplans

5.3.1 Schaltplanentwurf

Der Entwurf der Schaltung beginnt mit der Überlegung, an welchen Punkten die RGB-LEDs angebracht werden sollen. Entsprechend den Vorgaben werden 32 RGB-LEDs am großen Wandmonitor und die restlichen 32 RGB-LEDs an der Decke befestigt. Aufgrund der großen Entfernung zwischen den beiden Mikrofonarrays müssen mindestens zwei Platinen hergestellt werden.

Da aber der LED-Treiberbaustein nur 16 Ausgänge besitzt, werden insgesamt vier Boards mit jeweils 16 LEDs benötigt. Jedes Board wird mit drei Treiberbausteinen bestückt. Jeder Treiberbaustein wiederum steuert dann eine Grundfarbe der RGB-LEDs.

Der Schaltplanentwurf wird in der Regel mit einem Schaltplan-Editor vorgenommen. Bei diesem Projekt wird die Designsoftware „Eagle“ verwendet. Am Anfang wird überprüft, dass alle Bauteile, die in einem Schaltplan verwenden werden sollen, in der Bauteilbibliothek vorhanden sind. Aufgrund der Besonderheiten dieses LED-Treibers, ist es erforderlich, diesen in die Bibliothek manuell hinzuzufügen. In dieser Bibliothek sind für jedes Bauteil wichtige Information enthalten, wie zum Beispiel die originalen Abmaße der Bauelemente, sodass diese später auf die Platine gelötet werden können.

Weil die LED-Treiber alle an einen I^2C Bus angeschlossen werden, muss jedem Treiber eine unterschiedliche 7 Bit-Adresse zugeordnet werden.

Die Adressierung erfolgt auf Hardwareebene durch die Verbindung der vorgesehenen Pins entsprechend mit der Masse oder der Versorgungsspannung. Bei diesem Treiber sind die letzten drei Bits fest kodiert und nur die ersten vier Bits sind einstellbar, wie in der Abbildung 5.3.1.1 ersichtlich ist. [10]

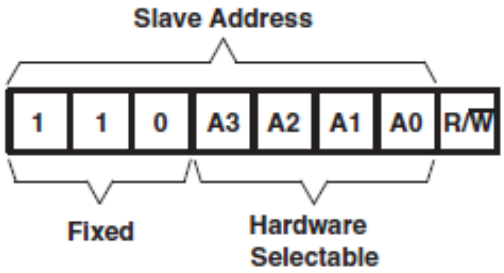


Abbildung 5.3.1.1: Hardwareadressierung des Slaves [10]

Nach Fertigstellung des ersten Boards, ist es wichtig die Adressen der LED-Treiber nicht erneut zu verwenden. Es ist darauf zu achten, dass jeder LED-Treiber eine

eindeutige Adresse besitzt. Eine Doppelbelegung würde den Bus in einen undefinierten Zustand versetzen.
Um diese Voraussetzungen zu schaffen, werden „Lötjumper“ verwendet. Somit ist es jederzeit mit relativ geringem Aufwand möglich, bei Bedarf die Adressen der LED-Treiber zu ändern.
Wenn diese Vorüberlegungen abgeschlossen sind, wird als nächstes der Schaltplan mit dem Programm „Eagle“ entworfen.
Bei der Konfiguration des LED-Treibers und Design des Schaltplanes werden alle wesentlichen Informationen aus dem vorhandenen Datenblatt berücksichtigt.
Zur besseren Lesbarkeit des Schaltplans sollten alle Bauteile übersichtlich angeordnet werden. Nun ist eine schnellere und einfache Fehlersuche möglich.

5.3.2 Layoutdesign

Aus diesem entworfenen Schaltplan bzw. Stromlaufplan kann jetzt mit dem Programm zur Schaltplanentwicklung „Eagle“ das Layoutdesign entwickelt werden. Hierfür besitzt das Programm eine Funktion, mit der alle verwendeten Bauteile grafisch und in Originalgröße in einem neuen Fenster dargestellt werden. Diese müssen dann so auf dem Board platziert werden, wie sie später auf das Board gelötet werden sollen. Die Bauteile müssen allerdings zuvor so dimensioniert werden, dass diese mit den originalen Bauteilen in sämtlichen Abmaßen übereinstimmen. Zudem werden alle Teile mit einer Luftlinie verbunden. Nun gibt es die Möglichkeit, auf einer vorher erstellten virtuellen Platine, die Bauteile anzuordnen und zu verschieben. Die Schwierigkeit liegt jetzt darin, die zuvor vom Programm gezeichneten Luftlinie mit virtuellen Leiterbahnen nachzumodulieren, sodass keine Überschneidungen auf einem Layer stattfinden. Dazu wurde für diese Schaltung eine Duallayer Euro Platine genutzt. Diese hat die Abmaße von 16 cm x 10 cm und kann von beiden Seiten belichtet und geätzt werden. So ergibt sich die Möglichkeit auf beiden Seiten Bauteile zu verbauen oder von dem einen auf den anderen Layer mittels Brücken zu wechseln. Es lassen sich so ohne weitere Umstände zwei relativ weit entfernte Bauteile auf der Platine mittels Leiterbahnen verbinden.
Allerdings hat diese Methode auch Nachteile. Auf der einen Seite ist der Aufwand erheblich höher, da im Endeffekt viele Brücken gelötet werden müssen und dies den Aufwand der Herstellung erhöht. Zudem steigert dies außerdem die Wahrscheinlichkeit auf „kalte Lötstellen“ und diverse andere Fehler.

Ein weiterer Nachteil stellt die Übersichtlichkeit dar. Durch die vielen Leitungen auf dem Board, die auf beiden Seiten verteilt sind, lässt sich nun die Leitung schwerer nachvollziehen.

5.4 Physikalische Herstellung der Platine der Treiberelemente

5.4.1 Ablauf des Ätzvorganges

Als Ausgangsmaterial zum Ätzen wird nun das fertige Schaltungslayout benötigt. Dieses wird jeweils von jeder der zwei Seiten angefertigt. An allen Stellen, an denen später schwarze Farbe zu sehen ist, wird die Kupferschicht auf der Platine durch den darüber liegenden Fotolack nicht abgelöst und es entstehen somit leitende Bahnen. Zudem werden alle größeren Flächen mit einer sogenannten Massefläche ausgefüllt, um nicht unnötig viel Material von der Platine zu lösen.

Die nun fertigen Layouts, die in der Software entstanden sind, werden mithilfe eines Grafikprogramms und unter Berücksichtigung der korrekten Pixelzahl auf eine durchsichtige Folie gedruckt. Diese Folie wird anschließend zurecht geschnitten und mit einem speziellen Spray eingesprüht. Dieses Spray hat die Aufgabe, den Toner bzw. die Farbe auf der Folie zu verdichten. Dies hat zur Folge, dass bei der anschließenden Belichtung weniger Licht durch die Folie tritt und somit die Konturen der dünnen Leiterbahnen besser geätzt werden können.

Die damit entstandenen Folien werden auf die Europlatinen gelegt und fixiert. Hierbei ist darauf zu achten, dass die Folien exakt übereinander liegen und möglichst keinen Versatz untereinander aufweisen. Beim Bohren der Löcher auf der Platine würden sonst bei nicht korrekter Überlagerung Probleme entstehen und im schlimmsten Fall die Signale nicht mehr korrekt übermittelt werden.

Der nächste Schritt stellt das Belichten dar. Dabei wird die mit den Folien beklebte Platine auf ein Belichtungsgerät gelegt. Hierbei wird durch den Fotolack, der sich auf der Platine befindet, das Layout übertragen. Die Belichtungszeit beträgt in dieser Größenordnung 10 min. Nach dieser Zeit kann die Platine gedreht werden und von der anderen Seite belichtet werden. Es entsteht so ein Abbild des Layouts auf der Platine.

Um das Layout allerdings sichtbar zu machen, ist der nächste Schritt vonnöten. In diesem wird die belichtete Platine für kurze Zeit in ein Entwicklerbad gegeben. Durch

eine chemische Reaktion löst sich an den belichteten Stellen der Fotolack von der Platine und ist nun vorbereitet für das Ätzbad.

Das Ätzbad stellt den eigentlichen Ätzvorgang dar. Die Platine wird vorher noch einmal von Entwickelrückständen gereinigt. Im Anschluss wird sie in eine Vorrichtung eingeklemmt und in das ca. 50° C warme Ätzbad gehängt. Bei einer Temperatur von 45° C bis 55° C und einer Zufuhr von Luftbläschen im Ätzbad arbeitet dieses am effizientesten. Nach einer Zeit von ca. 30 min ist dieser Vorgang abgeschlossen und die Platine kann aus dem Bad entfernt werden. Nun sollte die geätzte Platine im Anschluss gründlich gereinigt werden.

Auf der fertig geätzten Platine sind ab jetzt die einzelnen Leiterbahnen und die jeweiligen Pads deutlich erkennbar. An den Stellen, an denen zuvor die Platine belichtet wurde, hat sich vollständig das Kupfer abgelöst und bildet so keinen leitenden Kontakt mehr zwischen den einzelnen Leiterbahnen.

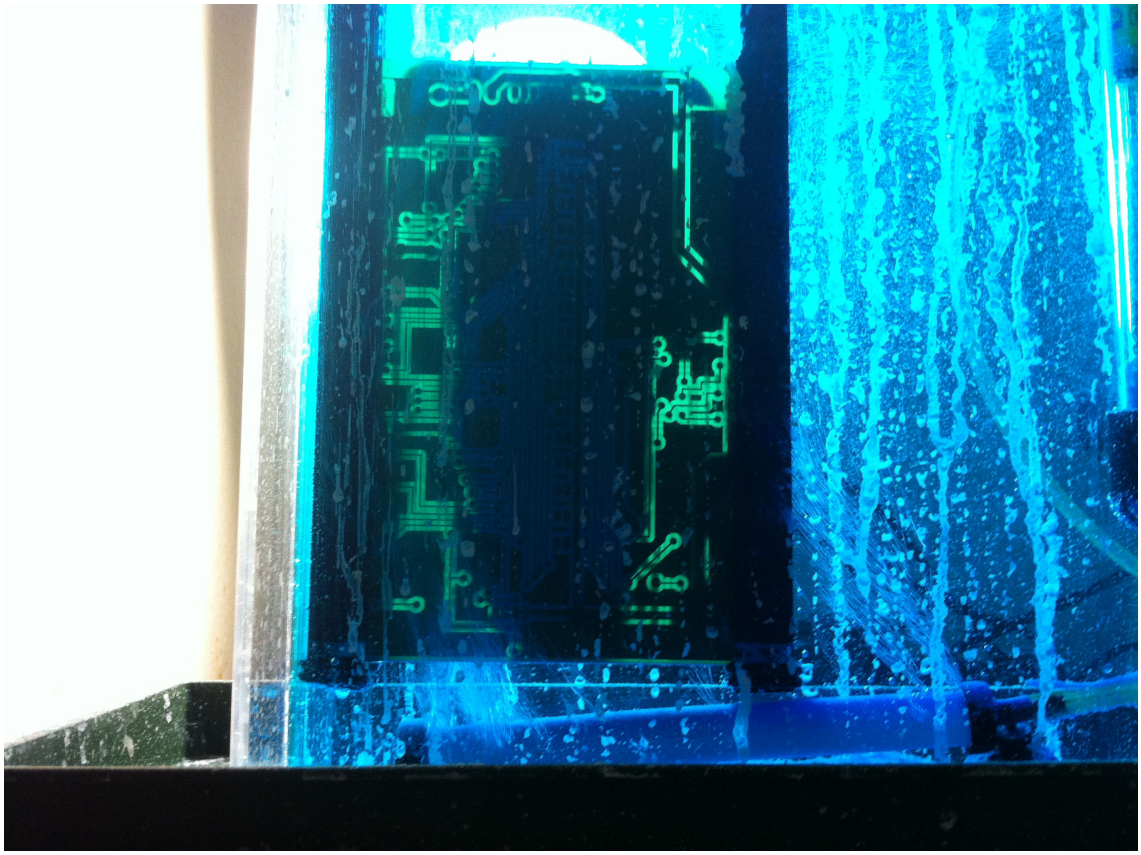


Abbildung 5.4.1.1: Platine der Treiberbausteine im Ätzbad

5.4.2 Fertigstellen der endgültigen Platine

Nachdem die Platine geätzt wurde, muss diese nun von dem Fotolack auf der Kupferschicht befreit werden. Der Lack wurde bei der Belichtung nicht mit UV Licht bestrahlt und hat sich so weder im Entwicklerbad noch im Ätzbad von der Kupferschicht abgelöst. Allerdings lässt sich diese verbleibende Schicht leicht mit Brennspiritus und einem Tuch von der Platine lösen. Ist das geschehen, so ist die Kupferschicht auf der Oberfläche leitend und kann nun weiter bearbeitet werden.

Der folgende Schritt im Ablauf der Fertigstellung stellt das Bohren der Löcher dar. Diese werden zum einen benötigt, um die Brücken zwischen den beiden Layern zu setzen und zum anderen, um die Steckbrücken für die Anschlüsse mit den Leiterbahnen zu verbinden. Es werden auf der Platine zwei verschiedene Durchmesser zum Bohren der Löcher verwendet. Es handelt sich dabei um einen Bohrer mit dem Durchmesser von d = 0,8 mm. Dieser hat die Aufgabe, die Brückenlöcher zu bohren. Da hier nur ein dünner Draht verwendet wird, ist ein Loch mit einem kleinen Durchmesser ausreichend. Zum anderen wird ein Bohrer mit dem Durchmesser von d=1 mm für die Steckbrückenlöcher verwendet. Da in diesem Fall die Pins der Steckbrücken genau 1mm betragen, ergibt sich eine gewisse Spannung und die Steckbrücken lassen sich gleichmäßig einlöten.

Für die Treiber und die Widerstände auf der Platine sind keine Löcher vorgesehen, da diese in SMD Bauweise zum Einsatz kommen und so lediglich Pads verwendet werden, auf denen diese festgelötet werden. Der große Vorteil besteht darin, dass diese Bauweise zum einen weniger Platz auf einem Layer in Anspruch nimmt und zum anderen nicht beide Layer der Platine beansprucht.

Sind alle Löcher gebohrt, werden sämtliche Brücken, Steckverbinder und Bauteile auf der Platine festgelötet. Dies geschieht mit einem Lötkolben und Lötzinn. Der Lötkolben wird dabei auf rund 250° C erwärmt und an die zu lötende Stelle gehalten. Dabei muss das Bauteil richtig platziert werden, was in aller Regel vorher geschieht. Wird nun der heiße Lötkolben an das Bauelement in Verbindung mit Lötzinn herangeführt, schmilzt das Lötzinn und legt sich durch das im Zinn enthaltene Flussmittel und seiner Oberflächenspannung genau auf das Pad und um das Bohrloch. Es entsteht eine leitende Verbindung zwischen Bauteil und dem Kupferpad. Außerdem erhärtet das Lötzinn nach raschem Abkühlen wieder. Es geht somit eine feste Verbindung zwischen Platine und Bauelement ein.

6 Programmierung des Mikrocontrollers

6.1 Entwicklung des Testprogrammes und experimentelle Versuche

6.1.1 Entwicklung der Schaltoberfläche auf dem PC

Um die Funktionalitäten der RGB-LEDs testen zu können, wird ein Testprogramm in Java entwickelt. Außerdem bietet dies den Vorteil, dass sämtliche Bibliotheken schon für das spätere Hauptprogramm implementiert sind und diese nur noch übertragen werden müssen.

Die Schaltoberfläche des entwickelten Testprogramms stellt in diesem Fall eine kleine grafische Oberfläche dar, mit dem es möglich ist, verschiedene Farben auf dem Mikrofonfeld zu realisieren. Außerdem lässt sich dabei jede LED einzeln in ihrer Farbe und ihrer Helligkeit beeinflussen. Im Prinzip ergibt sich die Möglichkeit eine der drei Grundfarben Rot, Blau und Grün aufleuchten zu lassen oder jeweils eine Farbmischung aus zwei verschiedenen Grundfarben. Des Weiteren besitzt das Programm eine Taste um alle LEDs auf dem Array auszuschalten. Man erhält die Farbe Weiß.

6.1.2 Verknüpfung von Steuerplatine und Computer

Die Verbindung zwischen Mikrocontroller und Computer erfolgt über die USB 2.0 Schnittstelle. Da der USB-Serial Wandler in dem MBED-Board, das in der Abbildung 5.1.2.1 zu sehen ist, keinen Standardtreiber von Windows unterstützt, muss ein spezieller Treiber installiert werden, damit das MBED-Board als COM-Port unter Windows im Gerätemanager angezeigt wird. Dieser Treiber lässt sich auf der MBED-Internetseite (www.mbed.org) herunterladen.

In der Standardkonfiguration von Java sind in den Bibliotheken keine Klassen enthalten, die es ermöglichen die COM-Ports unter Windows anzusprechen. Um diese Funktionalität nutzen zu können, müssen externe Bibliotheken hinzugefügt werden und spezielle Systemdateien in den Java Ordner kopiert werden.

Auf der Internetseite „http://rxtx.qbang.org“ sind diese Dateien in der 32 und 64 Bit Version zum Download verfügbar. Um mit der GUI von dem verwendeten Testprogramm, auf den COM-Port zuzugreifen, gibt es extra eine Klasse „Verbindung.java“. Von dieser Klasse wird ein Objekt in der GUI erzeugt. Bei der Initialisierung wird der COM-Port und die Baudrate übergeben. Damit die richtige

COM-Portnummer übergeben wird, ist diese über den Gerätemanager in Windows zu ermitteln.
Die Datenkommunikation erfolgt ausschließlich über dieses Objekt. Für die Datenkommunikation ist die folgende Reihenfolge zu beachten.

- Aufruf der Methode „oeffneSerialPort“
    - Verbindung zwischen Programm und COM-Port wird hergestellt
- Aufruf der Methode „sendeSerialPort“
    - Datenübertragung möglich
- Aufruf der Methode „schliesseSerialPort“
    - Verbindung wird zwischen Programm und COM-Port wird geschlossen

Wird diese Reihenfolge nicht beachtet, kommt es beim Schließen des Programms zu Fehlern, da das Programm nicht ordnungsgemäß geschlossen werden kann. In der Abbildung 6.1.2.1 ist das UML-Diagramm des Testprogramms dargestellt.

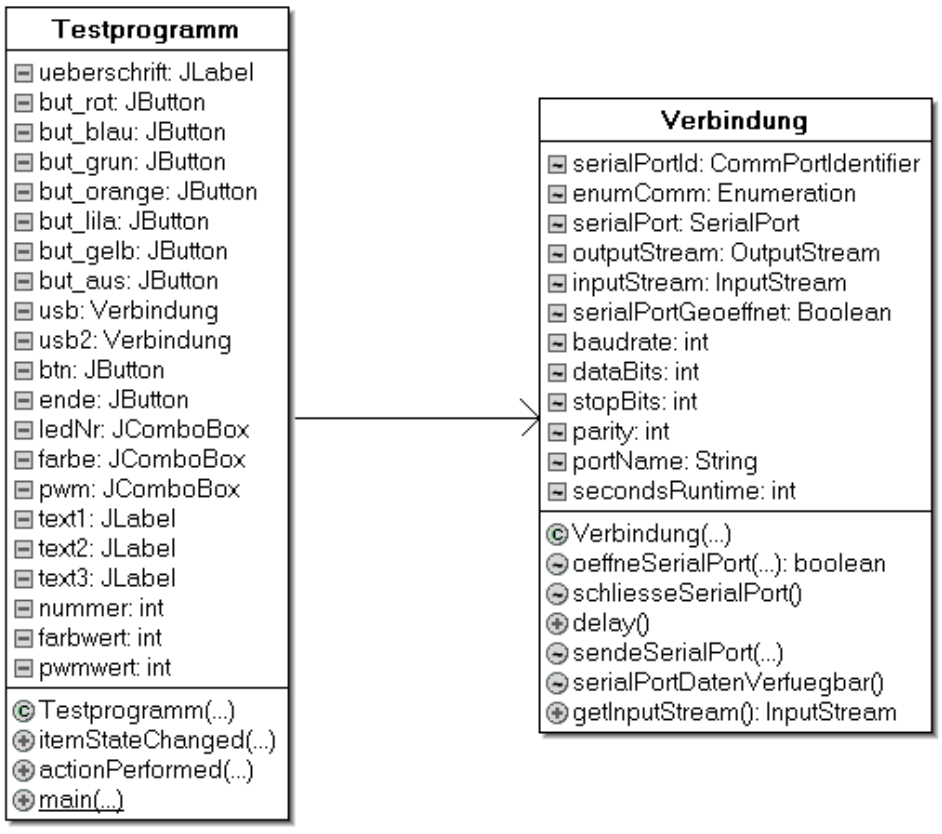


Abbildung 6.1.2.1: UML-Diagramm des Testprogrammes

6.1.3 Programmierung der LED-Treiber in Verbindung mit dem Mikrocontroller

Der für die Realisierung des Projekts verwendete Mikrocontroller ist bereits fertig auf einem Board aufgelötet, das von der Firma MBED gefertigt wird.
Dieses Board besitzt einen Mini-USB-Port, über dem die Programmierung des Mikrocontrollers als auch die Datenübertragung möglich ist.
Weiterhin besitzt dieses Board eine Taste, mit der das Programm neugestartet werden kann.
Wird das MBED-Board an den Computer angeschlossen, wird dieses als Massenspeichergerät erkannt. Das Programm zur Steuerung der LED's kann jetzt in den Speicher geladen werden. Durch Drücken der Taste auf dem Board wird das Programm gestartet.
Nach dem Start des Programmes wird ein Objekt vom I^2C Bus und vom UART-Port initialisiert. Da in der Bibliothek bereits die Klassen der beiden Objekte vorhanden sind, wird automatisch die Taktfreigabe und die Programmierung der Steuerwörter für die Initialisierung des entsprechenden Moduls abgearbeitet. [8]

Die UART-Schnittstelle wird mit folgenden Parametern initialisiert:

Einstellung	Wert
Baudrate	9600 Bits pro Sekunde
Datenbits	8 Bits
Parität	Keine
Stoppbits	1 Bit

Tabelle 6.1.3.1: UART-Einstellung des Mikrocontrollers

Bevor die LEDs angesteuert werden können, muss im Programm der jeweilige LED-Treiber initialisiert werden. Das geschieht über den I^2C Bus, indem die entsprechenden Steuerwörter an die LED-Treiber gesendet und in die vorgesehenen Register gespeichert werden. Der LED-Treiber besitzt zwei Register für die Initialisierung. Das „MODE1 Register" mit der Registeradresse 00h ist in der Tabelle 6.1.3.2 zu sehen. In diesem Register können verschiedene Einstellungen vorgenommen werden. In dem vom Mikrocontroller gesendeten Steuerwort wird nur das vierte Bit verändert. Dadurch wird

der interne Oszillator des LED-Treibers eingeschaltet, der für die PWM-Signale wichtig ist.

Bit	Symbol	Access	Value	Description
7	AI2	Read	0(Default Value)	Register auto-increment disable
			1	Register auto-increment enable
6	AI1	Read	0(Default Value)	Auto-increment bit 1 = 0
			1	Auto-increment bit 1 = 1
5	AI0	Read	0(Default Value)	Auto-increment bit 0 = 0
			1	Auto-increment bit 0 = 1
4	SLEEP	Read/ Write	0	Normal Mode
			1(Default Value)	Low-power mode. Oscillator off.
3	SUB1	Read/ Write	0(Default Value)	Device does not respond to I^2C bus subaddress 1.
			1	Device responds to I^2C bus subaddress 1.
2	SUB2	Read/ Write	0(Default Value)	Device does not respond to I^2C bus subaddress 2.
			1	Device responds to I^2C bus subaddress 2.
1	SUB3	Read/ Write	0(Default Value)	Device does not respond to I^2C bus subaddress 3.
			1	Device responds to I^2C bus subaddress 3.
0	ALLCALL	Read/ Write	0	Device does not respond to LED All Call I^2C bus address.
			1(Default Value)	Device responds to LED All Call I^2C bus address.

Tabelle 6.1.3.2: Beschreibung des MODE1-Registers [10]

Das zweite Register ist das „MODE2 Register“, das in Tabelle 6.1.3.2 ersichtlich ist und die Registeradresse 01h besitzt. In diesem Register wird das Blinken oder die Helligkeit der LEDs festgelegt. Die „Defaultwerte“ sind so eingestellt, dass die Helligkeit der LEDs geregelt wird. Somit werden keine Parameterwerte in diesem Register geändert.

Bit	Symbol	ACCESS	Value	Description
7	EFCLR	Read/Write	0(Default value)	Enable error status flag
			1	Clear error status flag
6:4		Read	000(Default value)	Reserved
5	DMBLNK	Read/Write	0(Default value)	Group control = dimming
			1	Group control = blinking
3	OCH	Read/Write	0(Default value)	Outputs change on Stop command
			1	Outputs change on ACK
2:0		Read	000(Default value)	Reserved

Tabelle 6.1.3.3: Beschreibung des MODE2-Registers [10]

Um die Register mit den jeweiligen Steuerwörtern zu konfigurieren, muss eine bestimmte Sendereihenfolge eingehalten werden. Es muss zuerst die Hardwareadresse des entsprechenden LED-Treibers gesendet werden, um die Verbindung zwischen Slave und Master herzustellen. Der Slave ist nun bereit, ein weiteres Byte zu empfangen. Danach sendet der Mikrocontroller die entsprechende Registeradresse des LED-Ports. Wird das siebente Bit der Registeradresse auf eins gesetzt, ist es möglich, mehrere Steuerwörter hintereinander zu senden, ohne jedes Mal die Registeradresse zu ändern. Dabei muss beachtet werden, dass die Registeradresse immer nur um eines erhöht wird.

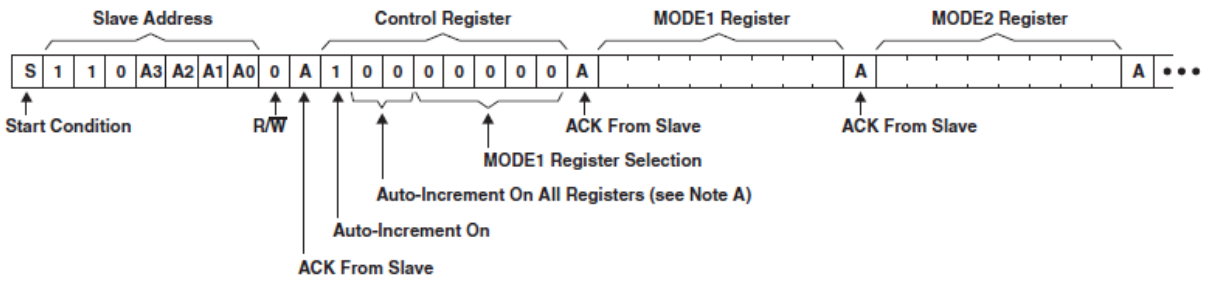


Abbildung 6.1.3.1: Programmierung des LED-Treibers mit dem I^2C Bus [10]

Wenn nun die Moderegister entsprechend programmiert sind, müssen danach noch die einzelnen Ports des LED-Treibers freigeschaltet werden. Dazu ist es erforderlich, dass die vier Register „LEDOUT“ geändert werden. Mit jedem dieser Register lassen sich genau vier Ports freischalten. Für jeden Port sind genau zwei Bits vorgesehen. Damit das PWM-Signal am jeweiligen Ausgangsport anliegt, muss der Binärcode 10binär für jeden Port programmiert werden.

ADDRESS	REGISTER	BIT	SYMBOL	ACCESS	VALUE (Default Value)	DESCRIPTION
14h	LEDOUT0	7:6	LDR3[1:0]	Read/Write	00	LED3 output state control
		5:4	LDR2[1:0]	Read/Write	00	LED2 output state control
		3:2	LDR1[1:0]	Read/Write	00	LED1 output state control
		1:0	LDR0[1:0]	Read/Write	00	LED0 output state control
15h	LEDOUT1	7:6	LDR7[1:0]	Read/Write	00	LED7 output state control
		5:4	LDR6[1:0]	Read/Write	00	LED6 output state control
		3:2	LDR5[1:0]	Read/Write	00	LED5 output state control
		1:0	LDR4[1:0]	Read/Write	00	LED4 output state control
16h	LEDOUT2	7:6	LDR11[1:0]	Read/Write	00	LED11 output state control
		5:4	LDR10[1:0]	Read/Write	00	LED10 output state control
		3:2	LDR9[1:0]	Read/Write	00	LED9 output state control
		1:0	LDR8[1:0]	Read/Write	00	LED8 output state control
17h	LEDOUT3	7:6	LDR15[1:0]	Read/Write	00	LED15 output state control
		5:4	LDR14[1:0]	Read/Write	00	LED14 output state control
		3:2	LDR13[1:0]	Read/Write	00	LED13 output state control
		1:0	LDR12[1:0]	Read/Write	00	LED12 output state control

Tabelle 6.1.3.4: Beschreibung der Ausgangsports des LED-Treibers [10]

Der Mikrocontroller muss sich jederzeit in einem Status befinden, in dem er Daten empfangen kann. Aus diesem Grund wird die UART Schnittstelle mit dem Interruptsystem des Mikrocontrollers verbunden. Der Nested Vectored Interrupt Controller (NVIC) im Cortex M3 unterstützt bis zu 112 Interrupts und es ist möglich jedem eine Priorität von 0 bis 31 zuzuordnen. Null besitzt in diesem Fall die höchste Priorität. Somit ist es möglich, dass ein bestimmter Interrupt immer zuerst abgearbeitet werden soll. In dem Programm wird als nächstes der NVIC initialisiert, indem wieder die Taktfreigabe erfolgt und bestimmte Steuerwörter programmiert werden.

Empfängt der Mikrocontroller ein Byte, erfolgt eine sofortige Programmunterbrechnung und es beginnt die Abarbeitung des entsprechenden Interrupt-Unterprogramms. Da es drei verschiedene Farben gibt und 64 Mikrofone, muss der PC dem Mikrocontroller zuerst mitteilen, welche Farbe und welches Mikrofon angesteuert werden soll. Dazu müssen insgesamt zwei Bytes gesendet werden. Das erste Byte soll die Mikrofonnummer und die Farbe beinhalten. Dabei ist es wichtig, dass die Zahl in den Binärcode umgewandelt wird. Mithilfe des ersten Bytes können die LED-Treiber

zurücksetzt und neu initialisieren werden. Die Kodierung des ersten Bytes ist in der Tabelle 6.1.3.5 dargestellt:

Binärcode	Bedeutung
00 xxxxxxb	Farbe Rot und Mikrophon zwischen 1 bis 32
01 xxxxxxb	Farbe Grün und Mikrophon zwischen 1 bis 32
10 xxxxxxb	Farbe Blau und Mikrophon zwischen 1 bis 32
11000000b	Initialisierung

Tabelle 6.1.3.5: Kodierung des Übertragungsbytes eines Mikrofons

Mit dem nächsten Byte soll der Helligkeitswert übergeben werden. So ist es möglich, mit einem Byte 256 verschiedene Zahlenwerte zu senden. Hat der Mikrocontroller die beiden Bytes empfangen, werden nun die Daten ausgewertet und an den entsprechenden LED-Treiber gesendet. Danach ist der Mikrocontroller sofort wieder bereit, Daten zu empfangen.
Wird das Initialisierungs-Byte gesendet, kann kein weiteres Byte gesendet werden. Des Weiteren ist es vorgesehen, die Ambientebeleuchtung über den Mikrocontroller einzuschalten. Das kann erfolgen, indem die Nummer 33 an den jeweiligen Mikrocontoller gesendet wird. [10]

6.1.4 Versuchsdurchführung und Problemlösung

Da nun die Testoberfläche fertig ist, kann man die Platine mit dem Mikrocontroller und diese anschließend mit dem PC verbinden. Dabei stellt sich heraus, dass bei einer einzelne LED diese ihre volle Helligkeit erreicht und das System stabil läuft. Werden nun alle 32 LEDs auf einem Panel angeschaltet, so zeigte sich in den ersten Versuchen, dass die Stromversorgung nicht ausreicht und das System abstürzt. Das Ergebnis ist, dass die Treiber über die I^2C Schnittstelle nicht mehr anzusteuern sind und nun die gesamte Schaltung neu gestartet werden muss. Durch die Möglichkeit den PWM Wert für die Helligkeit herabzusetzen, ziehen die Leuchtdioden automatisch weniger Strom.

Somit ist es möglich, auch 2 Farben miteinander zu kombinieren und mehr als die 3 Grundfarben darzustellen.
Später wurde das Problem dann mit einer zusätzlichen Spannungsversorgung behoben, sodass es auch möglich ist, alle 3 Grundfarben in Kombination zu schalten und bis hin zur Farbe Weiß alle Farben darzustellen.

7 Zusammenfassung

Im Laufe der Arbeit traten immer wieder Probleme auf, die vorher nicht einkalkuliert waren. Somit war es für uns zwingend, Lösungen zu finden.
Unter anderen traten Probleme mit dem Lieferanten der LEDs auf, weshalb sich im Endeffekt eine nicht wünschenswerte Farbmischung einstellte. Zum anderen mussten mehrere Treiberplatinen hergestellt werden, da sich im Laufe der Arbeit zeigte, dass die Leiterbahnen für die Stromaufnahme nicht geeignet waren. Deshalb lief das System anfangs recht instabil. Dieses Problem wurde gelöst, indem die stromführenden Leiterbahnen verstärkt wurden und ein externes, leistungsstarkes Netzteil an die Treiberplatinen angeschlossen wurde.
Insgesamt lässt sich sagen, dass die Arbeit im Speech Lab uns als Studenten gezeigt hat, dass es ein großer Unterschied ist, Dinge theoretisch zu betrachten und diese dann praktisch umzusetzen, als diese nur auf dem Blatt zu berechnen.

Anlagenverzeichnis

Anlage 1: Quellcode Mikrocontroller (C++)

```
/*
 * Programmierer: Markus Marks, Robert Haertel
 * Version 1.0: 1.8.2012
 * Ansteurung der LED-Treiber ueber I2C
 */

#include "mbed.h"

    I2C         i2c(p28, p27);
    Serial      pc(USBTX, USBRX);
    uint8_t     mikrofonNr = 0;
    uint8_t     counter = 0;
    uint8_t     farbe = 0;
    uint8_t     pwmWert = 0;
    PwmOut      b(p21);
    PwmOut      g(p22);
    PwmOut      r(p23);


    //LED-Treiber I2C-Adressen Board1

    char        a1 = 0xC0;  //rot
    char        a2 = 0xD8;  //gruen
    char        a3 = 0xD0;  //blau

    //LED-Treiber I2C-Adressen Board2

    char        b1 = 0xC8;  //rot
    char        b2 = 0xC2;  //gruen
    char        b3 = 0xC6;  //blau



/*  Daten senden ueber I2C
 *
 */
void sendData(char address,char c1,char c2) {
    int i = 1;
    int j = 0;
    char data[2];
    data[0] = c1;
    data[1] = c2;
    i2c.start();
    wait_ms(1);
    while (i != 0 && j < 10) {
        i = i2c.write(address, data,sizeof(data));
        j++;
    }
    i2c.stop();
```

```
}


void setAmbiente_blau(float c){
  float f = c/255.0;
  b.write(f);

}

void setAmbiente_rot(float c){
  float f = c/255.0;
  r.write(f);

}

void setAmbiente_gruen(float c){
  float f = c/255.0;
  g.write(f);

}


/*  Daten senden ueber I2C
 *
 */
void setLED(){
    if(farbe == 0){//Farbe rot
        if(mikrofonNr <= 0x0F)
            sendData(a1,mikrofonNr + 0x02,pwmWert);
        else if(mikrofonNr < 0x20)
            sendData(b1,mikrofonNr - 0x10 + 0x02,pwmWert);
        else
            setAmbiente_rot(pwmWert);
    }
    else if(farbe == 1){//Farbe gruen
        if(mikrofonNr <= 0x0F)
            sendData(a2,mikrofonNr + 0x02,pwmWert);
        else if(mikrofonNr < 0x20)
            sendData(b2,mikrofonNr - 0x10 + 0x02,pwmWert);
        else
            setAmbiente_gruen(pwmWert);
    }
    else if(farbe == 2){//Farbe blau
        if(mikrofonNr <= 0x0F)
            sendData(a3,mikrofonNr + 0x02,pwmWert);
        else if(mikrofonNr < 0x20)
            sendData(b3,mikrofonNr - 0x10 + 0x02,pwmWert);
        else
            setAmbiente_blau(pwmWert);
    }
}
```

```
/*  Initialisierung MODE1 Register der LED-Treiber ueber I2C
 *  Ports der LED-Treiber freischalten
 */

void init(){


    //Mode1-Register schreiben
    sendData(a1,0x00,0x00);
    sendData(a2,0x00,0x00);
    sendData(a3,0x00,0x00);

    sendData(b1,0x00,0x00);
    sendData(b2,0x00,0x00);
    sendData(b3,0x00,0x00);
    //Ports freischalten
    sendData(a1,0x14,0xAA);
    sendData(a1,0x15,0xAA);
    sendData(a1,0x16,0xAA);
    sendData(a1,0x17,0xAA);

    sendData(a2,0x14,0xAA);
    sendData(a2,0x15,0xAA);
    sendData(a2,0x16,0xAA);
    sendData(a2,0x17,0xAA);

    sendData(a3,0x14,0xAA);
    sendData(a3,0x15,0xAA);
    sendData(a3,0x16,0xAA);
    sendData(a3,0x17,0xAA);

    sendData(b1,0x14,0xAA);
    sendData(b1,0x15,0xAA);
    sendData(b1,0x16,0xAA);
    sendData(b1,0x17,0xAA);

    sendData(b2,0x14,0xAA);
    sendData(b2,0x15,0xAA);
    sendData(b2,0x16,0xAA);
    sendData(b2,0x17,0xAA);

    sendData(b3,0x14,0xAA);
    sendData(b3,0x15,0xAA);
    sendData(b3,0x16,0xAA);
    sendData(b3,0x17,0xAA);
}


/*  Diese Funktion ist die Interrupt-Funktion von UART0
 *
 */
extern "C" void UART0_IRQHandler(void) __irq {
    NVIC_ClearPendingIRQ(UART0_IRQn);
    char wert = pc.getc();
```

```
    //if(wert == 0xC0){
    if(wert == 0xC0 && counter == 0){
        init();
    }
    else if((wert & 0x3f) >= 0x00 && (wert & 0x3f) < 0x21 &&
counter == 0 ){ //Wert wird ueberprueft, ob er zwischen 0 und 32
liegt
        farbe = ((wert & 0xC0) >> 6)&0x03;
        mikrofonNr = (wert & 0x3f);
        counter++;
    }
    else if(counter == 1){
        pwmWert = wert;
        counter++;
    }

    if(counter == 2){
        //printf("%i ",mikrofonNr);
        setLED();
        counter = 0;
    }
}

int main() {
    pc.baud(9600);
    //i2c.frequency(500000);
    b.period_ms(1);
    r.period_ms(1);
    g.period_ms(1);
    b.write(1);
    r.write(1);
    g.write(1);
    //NVIC initialisieren
    NVIC_EnableIRQ(UART0_IRQn);
    LPC_UART0->IER = 0x01;
    while(1);
}}
```

Anlage 2: Quellcode Testprogramm (Java)

```
/*
*@author Robert H%rtel, Markus Marks
* LED-Class f¸r die Ansteurung der LED-Treiber f¸r SpeechLab
Version 1.0
*/

import java.awt.*;

public class LED{
  private Verbindung usb;
  private Verbindung usb2;
  private Color werte_board0 [] = new Color [32];
  private Color werte_board1 [] = new Color [32];
  private Color ambiente [] = new Color [2];

  public LED(String com1, String com2){
    //COM Schnittstellen mit Programm verbinden und LED-Treiber
initialisieren
    usb = new Verbindung(9600,com1); //Mikrofonarray 1  -->COM-
Port muss richtig eingestellt sein !!
    usb.oeffneSerialPort();
    usb.sendeSerialPort(192);

    usb2 = new Verbindung(9600,com2); //Mikrofonarray 2 -->COM-
Port muss richtig eingestellt sein !!
    usb2.oeffneSerialPort();
    usb2.sendeSerialPort(192);

    for(int i = 0; i < this.werte_board0.length; i++){
      this.werte_board0[i] = new Color(0,0,0);
    }
    ambiente [0] = new Color(0,0,0);
    ambiente [1] = new Color(0,0,0);
  }
  /* @param Board-Nummer (0,1); LED-Nummer(0,31); Color-
Objektmit RGB-Werten
  *  Wenn die Werte falsch sind, wird eine Exception ausgel^st
  *
  */

  public void setLedColor(int board, int led, Color color)
throws Exception{

    if(board == 0 && led >= 0 && led < 32){
      this.werte_board0 [led] = color;

      usb.sendeSerialPort(led);//Farbe rot und LED-Nummer senden
      usb.sendeSerialPort(color.getRed());

      usb.sendeSerialPort(led | 0x40);  //Farbe gruen und LED-
Nummer senden
      usb.sendeSerialPort(color.getGreen());
```

```
      usb.sendeSerialPort(led | 0x80);  //Farbe blau und LED-
Nummer senden
      usb.sendeSerialPort(color.getBlue());


    }
    else if(board == 1 && led >= 0 && led < 32){
      this.werte_board1 [led] = color;

      usb2.sendeSerialPort(led);
      usb2.sendeSerialPort(color.getRed());

      usb2.sendeSerialPort(led | 0x40);
      usb2.sendeSerialPort(color.getGreen());

      usb2.sendeSerialPort(led | 0x80);
      usb2.sendeSerialPort(color.getBlue());

    }
    else{
      throw new Exception();
    }
  }
  /* @param Board-Nummer (0,1); LED-Nummer(0,31);
  *  @return Color Objekt mit RGB-Werten
  *  Bei falscher Parameter¸bergabe wird eine Exception
ausgel^st
  */
  public Color getLedColor(int board, int led) throws Exception{
    if (board == 0 && led >= 0 && led < 32) {
      return this.werte_board0[led];
    } // end of if
    else if (board == 1 && led >= 0 && led < 32) {
      return this.werte_board1[led];
    } //
    throw new Exception();
  }
  /* @param Color Objekt mit den RGB Farben
  *
  *
  */
  public void setAllLedColors(Color color){
    //Farbe rot senden mit PWM-Wert von 100
    for(int i = 0; i < 32; i++){
      usb.sendeSerialPort(i);//Farbe rot und LED-Nummer senden
      usb.sendeSerialPort(color.getRed());

      usb.sendeSerialPort(i | 0x40);
      usb.sendeSerialPort(color.getGreen());

      usb.sendeSerialPort(i | 0x80);
      usb.sendeSerialPort(color.getBlue());

      usb2.sendeSerialPort(i);//Farbe rot und LED-Nummer senden
      usb2.sendeSerialPort(color.getRed());
```

```
      usb2.sendeSerialPort(i | 0x40);
      usb2.sendeSerialPort(color.getGreen());

      usb2.sendeSerialPort(i | 0x80);
      usb2.sendeSerialPort(color.getBlue());
    }
  }
  /*
  *  @param Color Objekt und Nummer der Ambienten
Beleuchtung[0,1]
  *  Bei falscher Parameter¸bergabe wird eine Exception
ausgel^st
  *
  */
  public void setColorAmbiente(Color color, int number) throws
Exception{
    this.ambiente[0] = color;
    this.ambiente[1] = color;
    if (number == 0) {

      usb.sendeSerialPort(32);
      usb.sendeSerialPort(color.getRed());

      usb.sendeSerialPort(32 | 0x40);
      usb.sendeSerialPort(color.getGreen());

      usb.sendeSerialPort(32 | 0x80);
      usb.sendeSerialPort(color.getBlue());
    }else if(number == 1){
      usb2.sendeSerialPort(32);
      usb2.sendeSerialPort(color.getRed());

      usb2.sendeSerialPort(32 | 0x40);
      usb2.sendeSerialPort(color.getGreen());

      usb2.sendeSerialPort(32 | 0x80);
      usb2.sendeSerialPort(color.getBlue());
    }
    else{
      throw new Exception();
    }
  }
  /*
  *  @param  Nummer der Ambienten Beleuchtung[0,1]
  *  Bei falscher Parameter¸bergabe wird eine Exception
ausgel^st
  *
  */
  Color getColorAmbiente(int number) throws Exception{
    if (number  == 0) {
      return this.ambiente[0];
    }
    else if(number == 1) {
      return this.ambiente[1];
```

```
      }
      throw new Exception();
    }
    /*
    *  COM-Ports werden geschlossen
    *
    */
    void schliessePort(){
      usb.schliesseSerialPort();
      usb2.schliesseSerialPort();
    }
}
```

Anlage 3: Quellcode Ansteuerung des Comports (Java)

```
import gnu.io.*;
import java.io.*;
import java.util.Enumeration;
import java.util.TooManyListenersException;

public class Verbindung  {

  /**
  * @param args

  public static void main(String[] args)
  {
  Runnable runnable = new EinfachSenden();
  new Thread(runnable).start();
  System.out.println("main finished");
  }

  /**
  *
  */

  CommPortIdentifier serialPortId;
  Enumeration enumComm;
  SerialPort serialPort;
  OutputStream outputStream;
  InputStream inputStream;
  Boolean serialPortGeoeffnet = false;
  int baudrate;
  int dataBits;
  int stopBits;
  int parity;
  String portName;
  int secondsRuntime;
  /**
  * @param Com-Port,Baudrate
  */
  public Verbindung(int baud,String port){
    baudrate = baud;
    dataBits = SerialPort.DATABITS_8;
    stopBits = SerialPort.STOPBITS_1;
    parity = SerialPort.PARITY_NONE;
    portName = port;
    secondsRuntime = 20;
  }
  /**
  * Port oeffnen
  * @return true-erflogreich        false-nicht erfolgreich
  */

  public boolean oeffneSerialPort(){
    Boolean foundPort = false;
    if (serialPortGeoeffnet != false) {
      System.out.println("Serialport bereits geˆffnet");
```

```
      return false;
    }
    System.out.println("÷ffne Serialport");
    enumComm = CommPortIdentifier.getPortIdentifiers();
    while(enumComm.hasMoreElements()) {
      serialPortId = (CommPortIdentifier)
enumComm.nextElement();
      if (portName.contentEquals(serialPortId.getName())) {
        foundPort = true;
        break;
      }
    }
    if (foundPort != true) {
      System.out.println("Serialport nicht gefunden: " +
portName);
      return false;
    }
    try {
      serialPort = (SerialPort) serialPortId.open("÷ffnen und
Senden", 500);
    } catch (PortInUseException e) {
      System.out.println("Port belegt");
    }
    try {
      outputStream = serialPort.getOutputStream();
    } catch (IOException e) {
      System.out.println("Keinen Zugriff auf OutputStream");
    }

    try {
      inputStream = serialPort.getInputStream();
    } catch (IOException e) {
      System.out.println("Keinen Zugriff auf InputStream");
    }


    try {
      serialPort.setSerialPortParams(baudrate, dataBits,
stopBits, parity);
    } catch(UnsupportedCommOperationException e) {
      System.out.println("Konnte Schnittstellen-Paramter nicht
setzen");
    }

    serialPortGeoeffnet = true;
    return true;
  }
  /**
  * Port schlieflen
  *
  */
  public void schliesseSerialPort()
  {
    if ( serialPortGeoeffnet == true) {
      System.out.println("Schliefle Serialport");
```

```
      serialPort.close();
      serialPortGeoeffnet = false;
    } else {
      System.out.println("Serialport bereits geschlossen");
    }
  }

  public void delay(){
    try{
      Thread.sleep(10);
    }
    catch(Exception e){

    }
  }
  /**
  * Wert senden
  *
  */
  public void sendeSerialPort(int nachricht)
  {
    if (serialPortGeoeffnet != true)
    return;
    try {
      outputStream.write(nachricht);
    } catch (IOException e) {
      System.out.println("Fehler beim Senden");
    }
  }
  /**
  * Daten verf¸gbar
  *
  */
  public void serialPortDatenVerfuegbar() {
    try {
      byte[] data = new byte[150];
      int num;
      while(inputStream.available() > 0) {
        num = inputStream.read(data, 0, data.length);

        System.out.println(new String(data, 0, num));
      }
    } catch (IOException e) {
      System.out.println("Fehler beim Lesen empfangener Daten");
    }
  }
  /**
  * Inputstream einlesen
  *
  */
  public InputStream getInputStream(){
    return this.inputStream;
  }
}
```

Anlage 4: Quellcode Verbindung zwischen Testprogramm und Comport

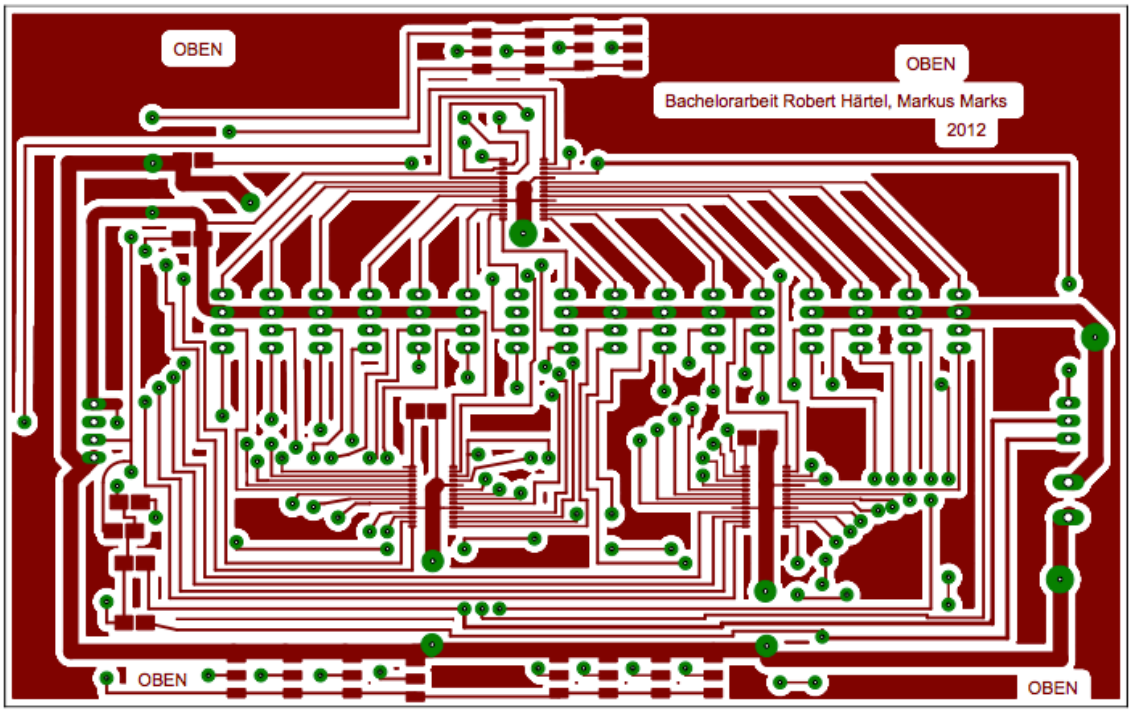
```
/*
*@author Robert H%rtel, Markus Marks
* LED-Class f¸r die Ansteurung der LED-Treiber f¸r SpeechLab
Version 1.0
*/
import java.awt.*;

public class LED{
  private Verbindung usb;
  private Verbindung usb2;
  private Color werte_board0 [] = new Color [32];
  private Color werte_board1 [] = new Color [32];
  private Color ambiente [] = new Color [2];

  public LED(String com1, String com2){
    //COM Schnittstellen mit Programm verbinden und LED-Treiber
initialisieren
    usb = new Verbindung(9600,com1); //Mikrofonarray 1  -->COM-
Port muss richtig eingestellt sein !!
    usb.oeffneSerialPort();
    usb.sendeSerialPort(192);

    usb2 = new Verbindung(9600,com2); //Mikrofonarray 2 -->COM-
Port muss richtig eingestellt sein !!
    usb2.oeffneSerialPort();
    usb2.sendeSerialPort(192);

    for(int i = 0; i < this.werte_board0.length; i++){
      this.werte_board0[i] = new Color(0,0,0);
    }
    ambiente [0] = new Color(0,0,0);
    ambiente [1] = new Color(0,0,0);
  }
  /* @param Board-Nummer (0,1); LED-Nummer(0,31); Color-
Objektmit RGB-Werten
  *  Wenn die Werte falsch sind, wird eine Exception ausgel^st
  *
  */
  public void setLedColor(int board, int led, Color color)
throws Exception{

    if(board == 0 && led >= 0 && led < 32){
      this.werte_board0 [led] = color;

      usb.sendeSerialPort(led);//Farbe rot und LED-Nummer senden
      usb.sendeSerialPort(color.getRed());

      usb.sendeSerialPort(led | 0x40);  //Farbe gruen und LED-
Nummer senden
      usb.sendeSerialPort(color.getGreen());

      usb.sendeSerialPort(led | 0x80);  //Farbe blau und LED-
```

```
Nummer senden
      usb.sendeSerialPort(color.getBlue());


    }
    else if(board == 1 && led >= 0 && led < 32){
      this.werte_board1 [led] = color;

      usb2.sendeSerialPort(led);
      usb2.sendeSerialPort(color.getRed());

      usb2.sendeSerialPort(led | 0x40);
      usb2.sendeSerialPort(color.getGreen());

      usb2.sendeSerialPort(led | 0x80);
      usb2.sendeSerialPort(color.getBlue());

    }
    else{
      throw new Exception();
    }
  }
  /* @param Board-Nummer (0,1); LED-Nummer(0,31);
  *  @return Color Objekt mit RGB-Werten
  *  Bei falscher Parameter¸bergabe wird eine Exception
ausgel^st
  */
  public Color getLedColor(int board, int led) throws Exception{
    if (board == 0 && led >= 0 && led < 32) {
      return this.werte_board0[led];
    } // end of if
    else if (board == 1 && led >= 0 && led < 32) {
      return this.werte_board1[led];
    } //
    throw new Exception();
  }
  /* @param Color Objekt mit den RGB Farben
  *
  *
  */
  public void setAllLedColors(Color color){
    //Farbe rot senden mit PWM-Wert von 100
    for(int i = 0; i < 32; i++){
      usb.sendeSerialPort(i);//Farbe rot und LED-Nummer senden
      usb.sendeSerialPort(color.getRed());

      usb.sendeSerialPort(i | 0x40);
      usb.sendeSerialPort(color.getGreen());

      usb.sendeSerialPort(i | 0x80);
      usb.sendeSerialPort(color.getBlue());

      usb2.sendeSerialPort(i);//Farbe rot und LED-Nummer senden
      usb2.sendeSerialPort(color.getRed());
```

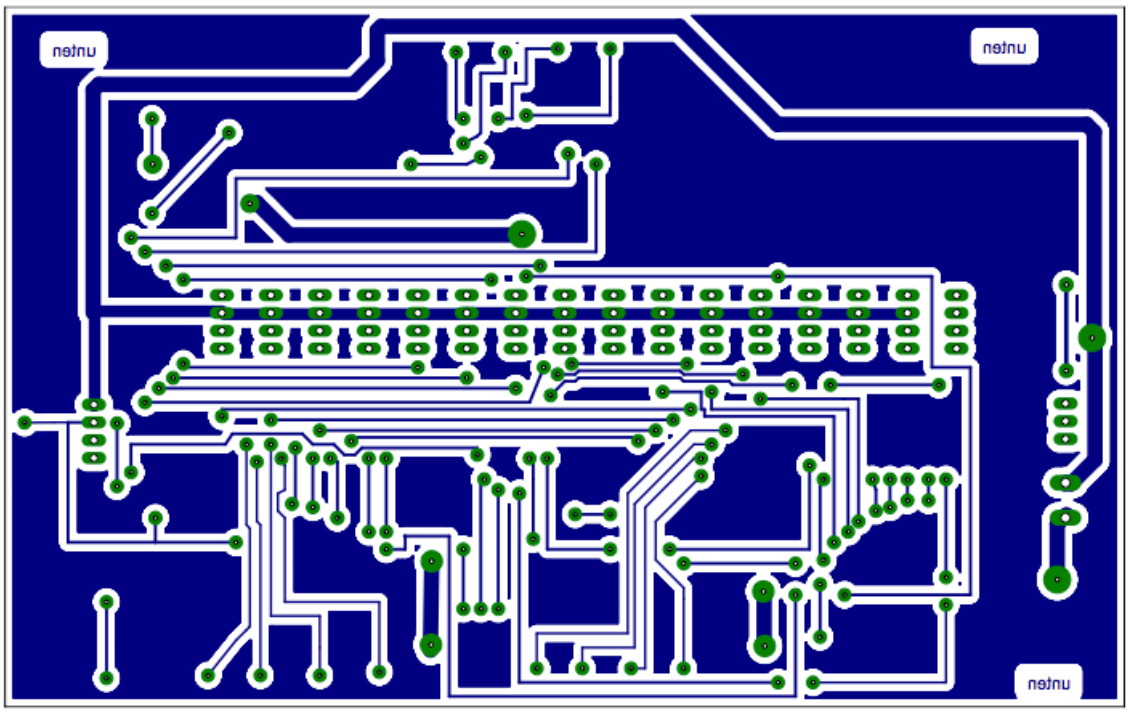
```
      usb2.sendeSerialPort(i | 0x40);
      usb2.sendeSerialPort(color.getGreen());

      usb2.sendeSerialPort(i | 0x80);
      usb2.sendeSerialPort(color.getBlue());
    }
  }
  /*
  *  @param Color Objekt und Nummer der Ambienten
Beleuchtung[0,1]
  *  Bei falscher Parameter¸bergabe wird eine Exception
ausgel^st
  *
  */
  public void setColorAmbiente(Color color, int number) throws
Exception{
    this.ambiente[0] = color;
    this.ambiente[1] = color;
    if (number == 0) {

      usb.sendeSerialPort(32);
      usb.sendeSerialPort(color.getRed());

      usb.sendeSerialPort(32 | 0x40);
      usb.sendeSerialPort(color.getGreen());

      usb.sendeSerialPort(32 | 0x80);
      usb.sendeSerialPort(color.getBlue());
    }else if(number == 1){
      usb2.sendeSerialPort(32);
      usb2.sendeSerialPort(color.getRed());

      usb2.sendeSerialPort(32 | 0x40);
      usb2.sendeSerialPort(color.getGreen());

      usb2.sendeSerialPort(32 | 0x80);
      usb2.sendeSerialPort(color.getBlue());
    }
    else{
      throw new Exception();
    }
  }
  /*
  *  @param  Nummer der Ambienten Beleuchtung[0,1]
  *  Bei falscher Parameter¸bergabe wird eine Exception
ausgel^st
  *
  */
  Color getColorAmbiente(int number) throws Exception{
    if (number  == 0) {
      return this.ambiente[0];
    }
    else if(number == 1) {
      return this.ambiente[1];
```

```
    }
    throw new Exception();
  }
  /*
  *  COM-Ports werden geschlossen
  *
  */
  void schliessePort(){
    usb.schliesseSerialPort();
    usb2.schliesseSerialPort();
  }
}
```

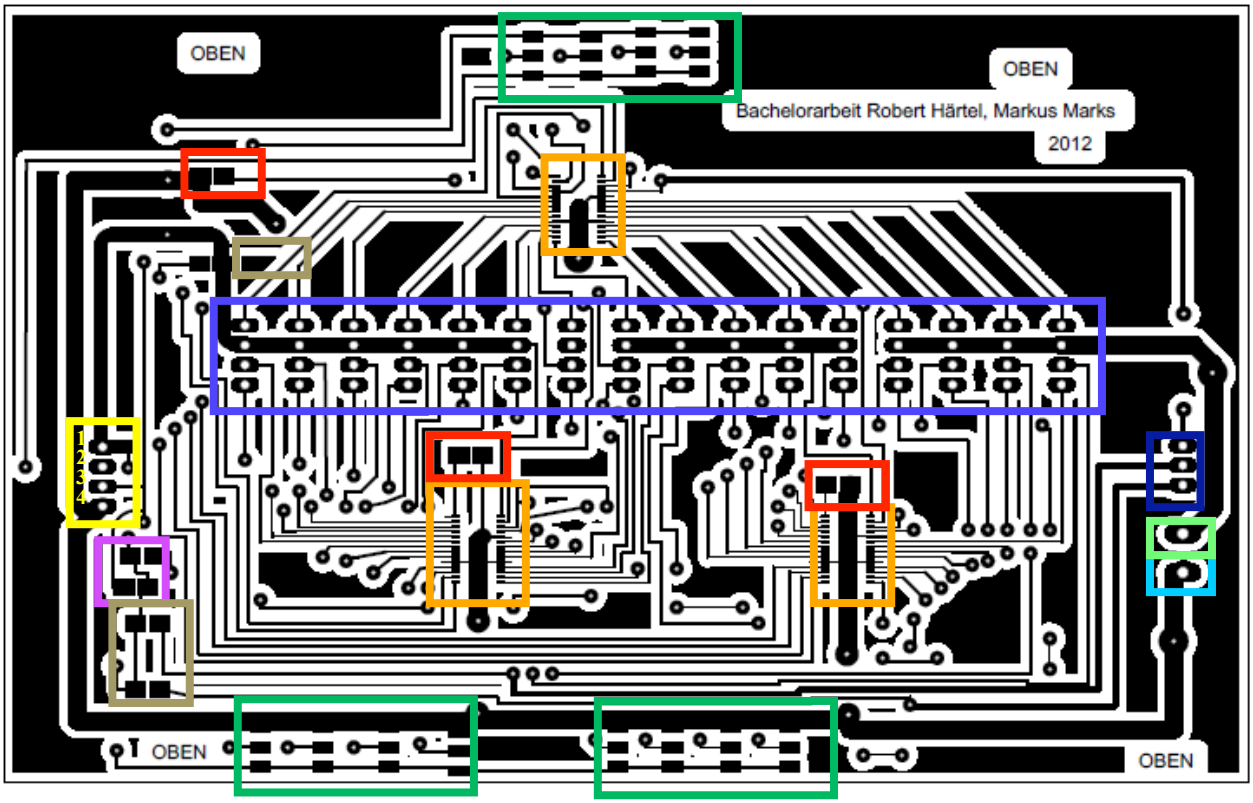
Anlage 5: Layout der Oberseite der Treiberplatine



Anlage 6: Layout der Unterseite der Treiberplatine

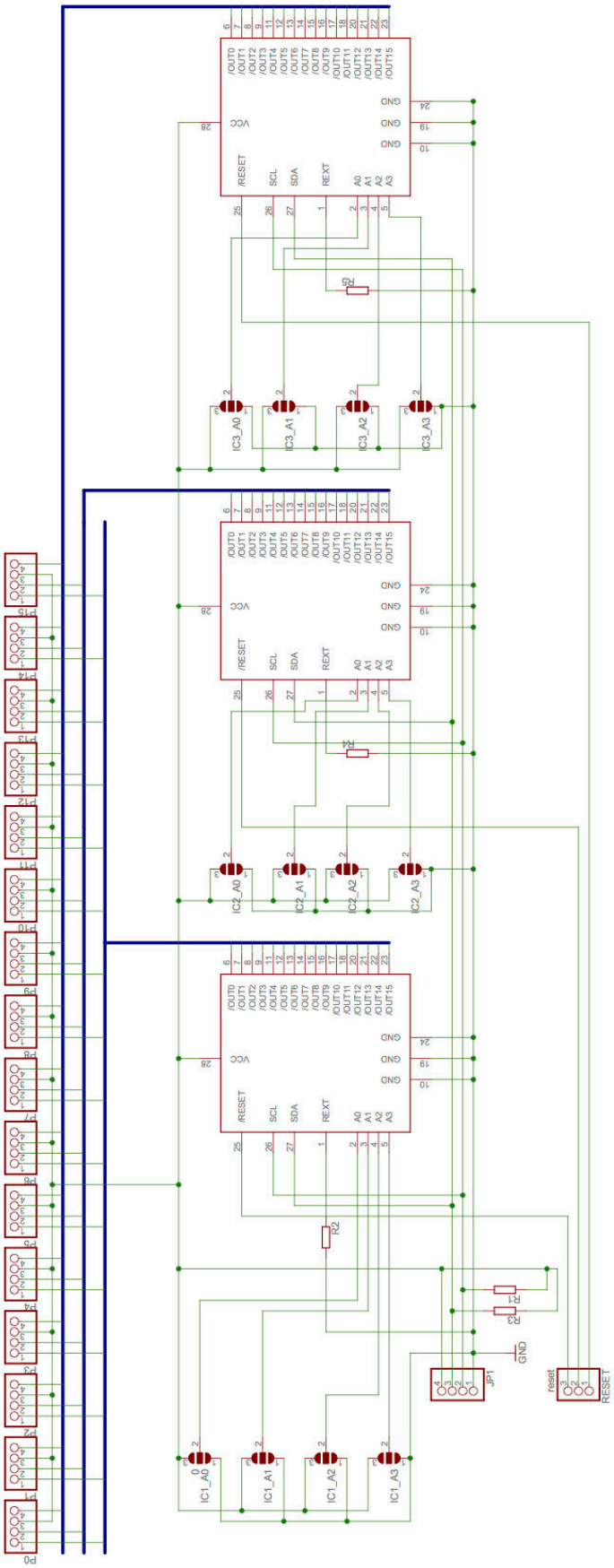


Anlage 7: Bestückungsliste der Treiberplatine

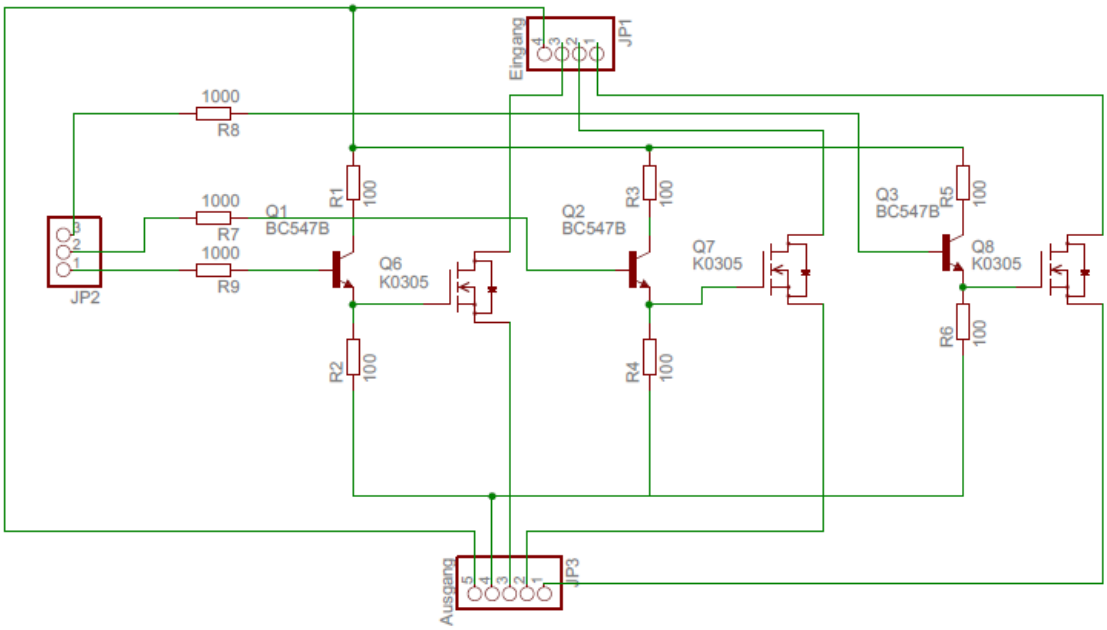


- Connector: 1:VCC
  2:SDA
  3:SCL
  4:GND
- RGB-LED-Ports: 0 bis 15
- LED-Treiber
- VCC (Anschluss für Netzteil)
- GND (Anschluss für Netzteil)
- herausgeführte Reset-Pins (für Testzwecke)
- Pullupwiderstände 1,8 kΩ
- “Lötjumper“ für die Adresssierung
- Widerstand zur Strom-Einstellung 480 Ω
- Reset-Widerstände 0 Ω

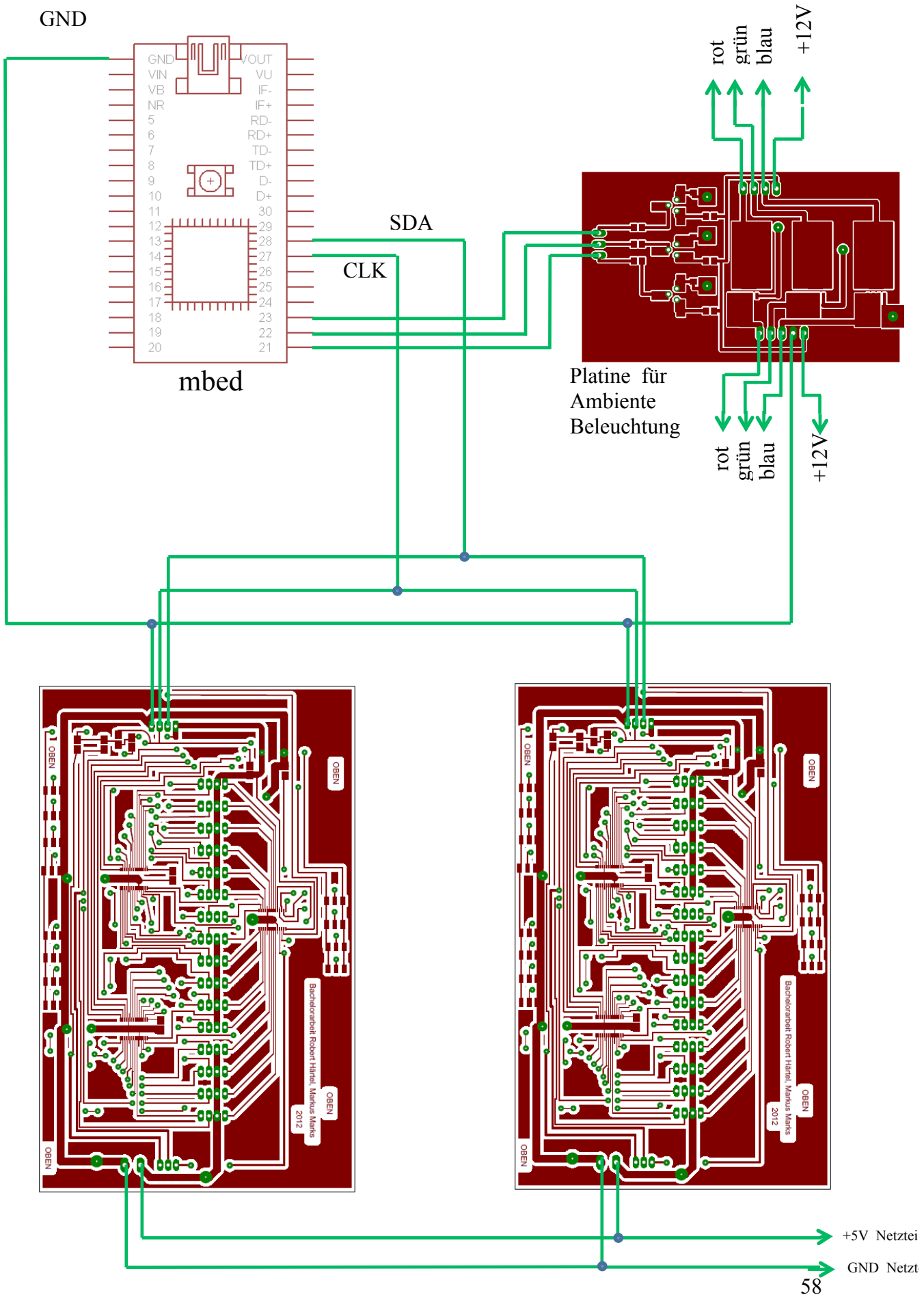
Anlage 8: Layout des Stromlaufplans der Treiberplatine



Anlage 9: Schaltplan Ambiente – Beleuchtung



Anlage 10: Steckplan der Schaltung



Abkürzungsverzeichnis

LED	Leuchtdiode
RGB-LED	Farbleuchtdiode, die durch Mischung der Farben Rot, Blau und Grün jede Farbe im Farbspektrum erzeugen kann
SMD	Surface Mounted Device
NVIC	Nested Vectored Interrupt Controller
TSSOP	Thin Shrink Small Outline Package
USB	Universal Serial Bus
IC	Integrated Circuite
RISC	Reduced Instruction Set Computer
DIP	Dual in-line Package

Quellenverzeichnis

Literaturverzeichnis

[1] Göbel, H.: *Einführung in die Halbleiterschaltungstechnik.* 3. Bearbeitete und erweiterte Auflage. Springer-Verlag Berlin Heidelberg, 2008.

Internetverzeichnis

[2] Elektronik Kompendium.: *LED - Leuchtdioden*
http://www.elektronik-kompendium.de/sites/bau/0201111.htm
(abgerufen am 06.07.2012)

[3] Dr. Andrew Greensted.: *The Lab Book Pages*
http://www.labbookpages.co.uk/audio/beamforming.html
(abgerufen am 29.06.2012)

[4] Prof. Dr. Ing.- D. Kraus.: *Time-Delay-and-Sum-Beamforming an den gemessenen Sensordaten eines Experimental-Sediment-Sonars*.
http://homepages.hs-bremen.de/~krausd/iwss/SA_Goldenbaum.pdf
(abgerufen am 29.06.2012)

[5] HEAD acoustics.: *Arraytechnologie und Beamforming.*
http://www.head-acoustics.de/downloads/de/head-visor/Array_Beamforming_07_09d.pdf
(abgerufen am 29.06.2012)

[6] Robot Electronics.: *Using the I2C Bus*
http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html
(abgerufen am 29.06.2012)

[7] Wikipedia - Die freie Enzyklopädie.: *I^2C*
http://de.wikipedia.org/wiki/I%C2%B2C
(abgerufen am 30.06.2012)

[8] Amit Naran.: *Handbook mbed NXP LPC1768*
http://mbed.org/handbook/mbed-NXP-LPC1768
(abgerufen 30.06.2012)

[9] Intuitive Roboter in der Schule.: *Pulsweitenmodulation*
http://robowiki.rids.de/Pulsweitenmodulation
(abgerufen am 03.07.2012)

[10] Texas Instruments.: *TLC59116 Datasheet*
www.ti.com/lit/ds/symlink/tlc59116.pdf
(abgerufen am 01.04.2012)

[11] Didatronic.de.: *Diodenkennlinie*
http://www.didactronic.de/Halbleiter+Dioden/diodenkennl.htm
(abgerufen am 25.07.2012)

[12] mikrocontroller.net.: *I2C als Hausbus*
http://www.mikrocontroller.net/articles/I2C_als_Hausbus
(abgerufen am 08.04.2012)

[13] Detlef Mietke.: *Funktionsweise von Kondensatormikrofonen*
http://www.elektroniktutor.de/techno/c_mikro.html
(abgerufen am 12.07.2012)

[14] Markus Hausmann.: *Mikrofonie 3, Kondensatormikrofon*
http://www.markushausmann.de/teckiblog/?tag=kondensatormikro
(abgerufen am 12.07.2012)

[15] Wikipedia – Die freie Enzyklopädie.: *Schallausbreitung*
http://de.wikipedia.org/wiki/Schallausbreitung
(abgerufen am 15.07.2012)

[16] schaumstoff.com.: *Schalldämmung, Schallisolierung mittels Schaumstoff*
http://schaumstoff.com/index.htm?schallisolierung%2FSchallisolierung.htm
(abgerufen am 15.07.2012)