Bachelorarbeit

Oliver Köster

Entwicklung eines Mikrocontroller-gesteuerten
RGB LED Cubes mit nativer USB Schnittstelle

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Oliver Köster

Entwicklung eines Mikrocontroller-gesteuerten RGB LED Cubes mit nativer USB Schnittstelle

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter : Prof. Dr. rer. nat. Thomas Canzler

Abgegeben am 15. Dezember 2010

Oliver Köster

Thema der Bachelorarbeit

Entwicklung eines Mikrocontroller-gesteuerten RGB LED Cubes mit nativer USB Schnittstelle.

Stichworte

3D Display, Cube, Borg, RGB, LED, Konstantstrom, Schieberegister, MOSFET, USB, CDC, Atmel, AVR, ATmega, JTAG, Mikrocontroller, SPI, Platine, Leiterplatte, CAD

Kurzzusammenfassung

In dieser Arbeit wird die Entwicklung eines Cubes mit 512 RGB LEDs beschrieben, welcher über die USB Schnittstelle gesteuert werden kann. Die Entwicklung umfasst sowohl das Design und die Herstellung der Hardware, als auch die Programmierung der verwendeten Mikrocontroller.

Oliver Köster

Title of the paper

Development of a microcontroller-controlled RGB LED cube with native USB port.

Keywords

3D Display, Cube, Borg, RGB, LED, constant current, shift register, MOSFET, USB, CDC, Atmel, AVR, ATmega, JTAG, microcontroller, SPI, board, PCB, CAD

Abstract

In this paper the development of a cube with 512 RGB LEDs is described, which can be controlled via the USB interface. The development includes the design and manufacture of the hardware and the programming of the microcontrollers used.

Danksagung

An dieser Stelle möchte ich allen danken, die mich bei dieser Arbeit unterstützt haben und mir jederzeit mit Rat und Tat und viel Geduld zur Seite standen.

Mein größter Dank gilt **Prof. Dr. rer.nat. Gunter Klemke** für seine Bereitschaft, diese Arbeit zu betreuen und zu unterstützen und die während des Studiums sehr interessanten Vorlesungen. Ebenfalls gehört mein Dank **Prof. Dr. rer. nat. Thomas Canzler** dafür, dass er sich bereiterklärt hat, das Zweitgutachten zu erstellen. Zudem danke ich **Prof. Dr. rer. nat. Kai von Luck**, der mich auf das Thema dieser Arbeit gebracht und meinen Ehrgeiz geweckt hat, dieses Projekt in die Tat umzusetzen.

Herzlich bedanken möchte ich mich bei **Dipl.-Inf. (FH) Martin Ongsiek** für die Hilfsbereitschaft und die freundliche Unterstützung bei Fragen bezüglich der PWM Ansteuerung und Dimmung der LEDs. Ohne seine Erfahrung hätte sich die Hardwareprogrammierung vielleicht in die falsche Richtung entwickelt.

Weiter danke meinem Kommilitonen **B. Sc. Simon Hillebrecht** für die Entwicklung einer alternativen 3D-Steuersoftware in Python und OpenGL um die Visualisierung des Cubes am PC zu ermöglichen. Meinem Bruder **Florian Köster** danke ich für sein intensives Korrekturlesen und die Verbesserungsvorschläge. Ohne ihn wäre diese Arbeit nicht auf ihrem jetzigen sprachlichen Niveau.

Ebenfalls bedanken möchte ich mich bei **Johann Abrams** und **Peter Radzuweit** aus dem Labor für Allgemeine Informatik an der HAW Hamburg für die schnelle und unkomplizierte Bestellung von Bauteilen, ebenso bei **Jane Upham** von NXP Semiconductors und der Firma **Texas Instruments Inc.** für die kostenlose Bereitstellung und schnelle Lieferung von Halbleiterbauteilen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einführung

In der heutigen Zeit sind Leuchtdioden nicht mehr wegzudenken. In einem Großteil der sich auf dem Markt befindlichen elektronischen Geräten werden diese light emmitting diodes, kurz LEDs, oft als Kontrolllampen oder Status-Indikatoren eingesetzt. Seit der kostengünstigen Herstellung von blauen und weißen LEDs sind sie nun auch eine echte Alternative zu Leuchtstofflampen geworden und können sowohl als Hintergrundbeleuchtung in LC-Displays als auch als Ersatz für handelsübliche Glühlampen bzw. Energiesparlampen eingesetzt werden. Eines der vielleicht interessantesten Einsatzgebiete von LEDs sind aber sicherlich die Verwendung als Pixel in einem LED-Display, welche oft als Laufschrift in Schaufenstern oder Anzeigetafeln in U-Bahn-Stationen benutzt werden. Dies sind jedoch oftmals nur einfarbige, relativ grob aufgelöste Anzeigen, welche keinesfalls einen Computermonitor ersetzen könnten. Aus diesem Grund wurde weiter in dieser Richtung geforscht und 2006 von Samsung das das erste funktionsfähige Display mit organischen Leuchtdioden (OLED) in einem sehr geringen Pixelabstand und 18 Bit Farbtiefe vorgestellt. Mittlerweile sind selbst große Fernsehgeräte mit OLED-Technologie verfügbar, deren Technik immer weiter verbessert wird.

1.1 Motivation

Es scheint als gäbe es ausreichend verschiedene Technologien, um zweidimensionale Bilder darzustellen, doch wie sieht es mit „echten“ 3D-Anzeigen aus? - Also Anzeigen/Monitore, die nicht auf eine optische Täuschung des menschlichen Auges angewiesen sind, um einen dreidimensionalen Effekt zu erzielen, sondern tatsächlich ein dreidimensionales Bild darstellen, welches von jeder Perspektive aus betrachtet werden kann? Bis auf einige wenige (meist einfarbige) Prototypen existiert noch keine sinnvolle Alternative und es wird Zeit, sich außerhalb des Hobbybereiches auf ein solches Projekt zu konzentrieren.

1.2 Zielsetzung

Das Ziel dieser Arbeit soll die Entwicklung eines Controllers mit passendem Anzeigemedium sein, der es ermöglicht, mit möglichst wenig Verdrahtungsaufwand eine große Anzahl von LEDs anzusteuern, welche symmetrisch in einer kubischen Form angeordnet sind, um so ein dreidimensionales Bild zu erzeugen. Außerdem soll dieser Controller möglichst einfach ansteuerbar sein, ohne dabei auf ein proprietäres Datenprotokoll setzen zu müssen.

2 Analyse

In diesem Kapitel wird analysiert, welche prinzipiellen Möglichkeiten es gibt, einen LED Cube physikalisch aufzubauen und zu beschalten, welche Hardware von Nöten ist um diesen anzusteuern und ob es bereits fertige Lösungen für diesen Anwendungsbereich gibt.

2.1 Anforderungen (1)

Der erste Gedanke, mit dem man sich beschäftigen muss, ist der prinzipielle Aufbau eines 3D-Displays. Hier gibt es verschiedene Ansätze, bei denen die Kugelform und die Würfelform sicher zu den interessantesten gehören. Da die Kugelform leider zu großen Problemen in der Verdrahtung und Beschaltung führen würde, beschäftigt sich diese Arbeit ausschließlich mit der Würfelform - dem 3D Kubus, oder „Cube“.

Wie der Name vermuten lässt, besteht ein LED Cube aus einer festen Anzahl von LEDs, welche in einer kubischen Form angeordnet sind. Sinnvolle Auflösungen sind 4x4x4=64, 8x8x8=512 oder 16x16x16=4096 LEDs, doch auch asymmetrische und somit rechteckige Formen sind möglich. Bei einer so großen Anzahl von einzelnen Verbrauchern sollte die Frage der Verdrahtung dieser „Pixel“ im Vordergrund stehen. „Die direkte statische Ansteuerung jeder einzelnen LED in einer 8x8x8-Matrix ist etwas unrealistisch, da man mindestens 513 Leitungen verdrahten und ansteuern müßte“ (Berger 2009, 3.2.3).

Es ist also wichtig, möglichst viele Leitungen zu sparen, da es kein kommerziell zu erwerbendes, paralleles Interface mit einer solchen Busbreite gibt. Ideal wäre hierbei ein einfaches Interface mit wenigen Leitungen zu einem externen Controller oder PC, über den die Daten als Vektoradressen in einem seriellen Datenstrom generiert werden.

2.2 Bekannte Cube-Projekte

2.2.1 3D Display Cube V1 von James Clar

Die erste Version des 3D Display Cubes von James Clar besteht aus 10 klaren Plexiglas-Platten, in denen jeweils 100 LEDs eingelassen wurden (siehe Abb. 2.1). Jede LED wurde, wie in Abbildung 2.2 ersichtlich, einzeln verdrahtet und von insgesamt 40 Mikrocontrollern angesteuert. Der Vorteil von diesem Aufbau ist die Tatsache, dass hier nicht die Trägheit des menschlichen Auges benutzt wird, um den Effekt der gleichzeitigen Ansteuerung zu simulieren, sondern dass jede LED tatsächlich einzeln angesteuert werden kann. Dies erfordert jedoch einen sehr hohen Schaltungsaufwand und einen enormen Platzbedarf (siehe 2.3). Außerdem ist dieses Design fehleranfällig, da die Anzahl der Zuleitungen proportional zur Anzahl der LEDs ansteigt.

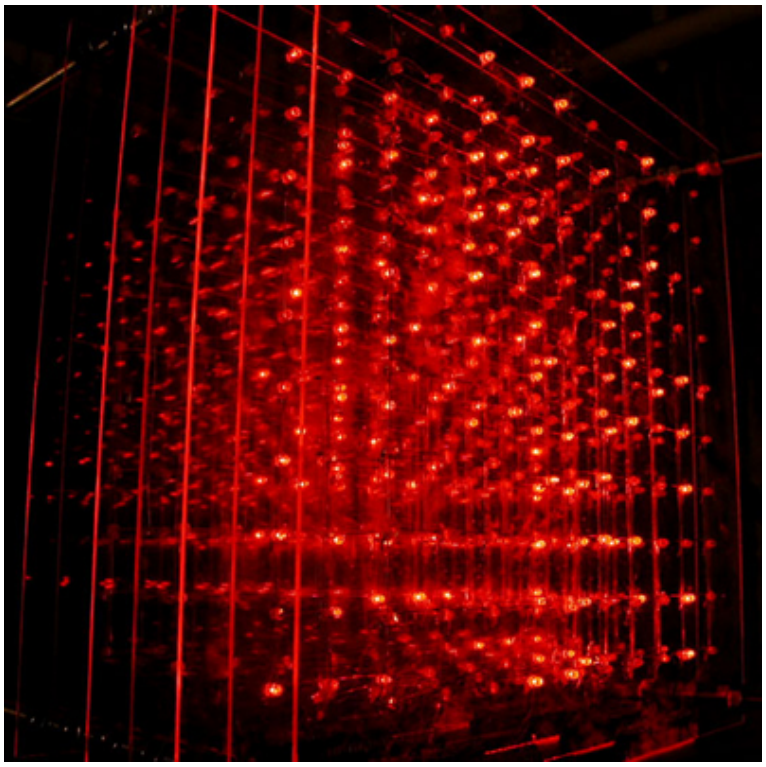

Abbildung 2.1: Die Plexiglasplatten im Detail

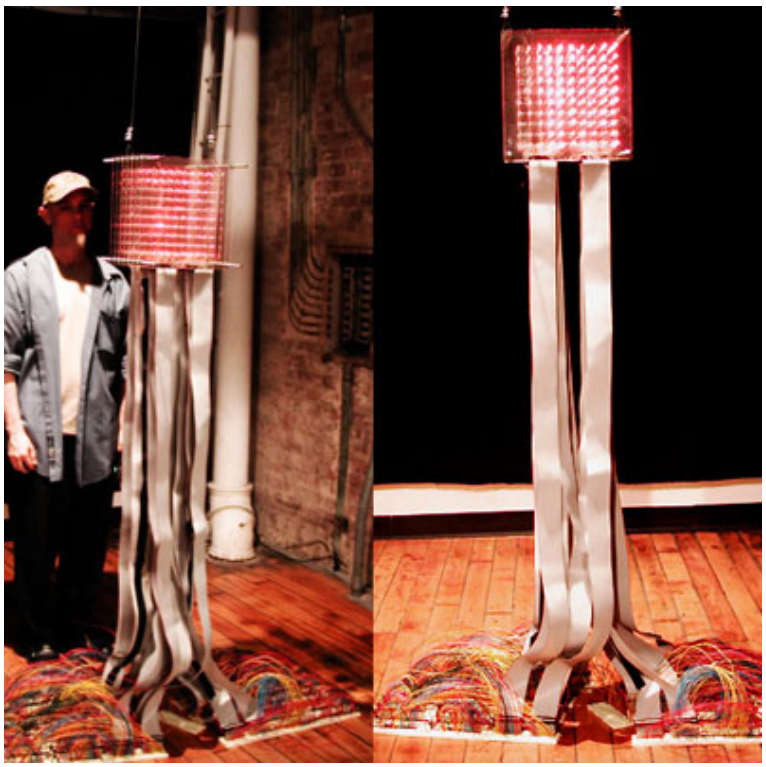

Abbildung 2.2: Zuleitungen vom Controller zum Cube

Abbildung 2.3: Detailansicht des Controllers, aufgebaut auf Breadboards

Die Vorteile des LED Cubes von James Clar sind in erster Linie die Größe und die hohe Anzahl von LEDs. Außerdem ist der Bildaufbau nahezu flimmerfrei da mit 40 Mikrocontrollern sehr hohe PWM Frequenzen zur Dimmung möglich sind. Ein weiterer interessanter Aspekt dieses Projektes ist die Skalierbarkeit. Jede Ebene kann getrennt von den anderen betrieben werden und somit kann der Cube theoretisch auch problemlos um Ebenen erweitert werden.

Bei dem 3D Display Cube handelt es sich um einen Prototypen, welcher weder komplett, noch in Teilen käuflich zu erwerben ist. Auch der Quellcode der Mikrocontroller ist nicht für die Öffentlichkeit bestimmt. Dieser wäre allerdings auch nur zu Informationszwecken zu gebrauchen, da es auch keine Schaltpläne gibt, die einen Nachbau ermöglichen würden.

2.2.2 Borg3D von LABOR e.V.

Von dem LABOR e.V. in Bochum wurde ein 8x8x8 LED Cube in Abbildung 2.4 entwickelt, welcher den Namen „Borg3D“ trägt. Er bietet nur eine mögliche Farbe, wird jedoch im von einem ATmega32 im Zeitmultiplexverfahren zu 8x64 LEDs angesteuert und beherrscht die Dimmung in 4 Stufen pro Ebene. Die Dimmung wird hier durch einen Teiler bei der sequenziellen Ansteuerung der Ebenen realisiert, so dass für die volle Helligkeit jede Ebene bei jedem Durchgang geschaltet wird. Für eine geringere Helligkeit geschieht dies nur noch bei jedem zweiten/dritten/vierten Durchgang. Die einzelnen Ebenen werden bei diesem Cube über ein einzelnes Schieberegister mit nachgeschalteten Leistungstransistoren angesteuert. Da zur Ansteuerung der einzelnen LEDs in einer Ebene nun noch weitere 64 Leitungen benötigt werden, können diese nicht direkt vom Mikrocontroller getrieben werden, sondern werden jeweils in Gruppen von je 8 LEDs und einem 8-fach Flipflop Baustein zusammengefasst. Diese Flipflops werden dann der Reihe nach, mit einem Schieberegister, mit den jeweils gewünschten Bitmustern für LED 1-8, 9-16,(...),56-64 angesprochen und durchgeschaltet. (Ongsiek u. a. 2008, vgl.).

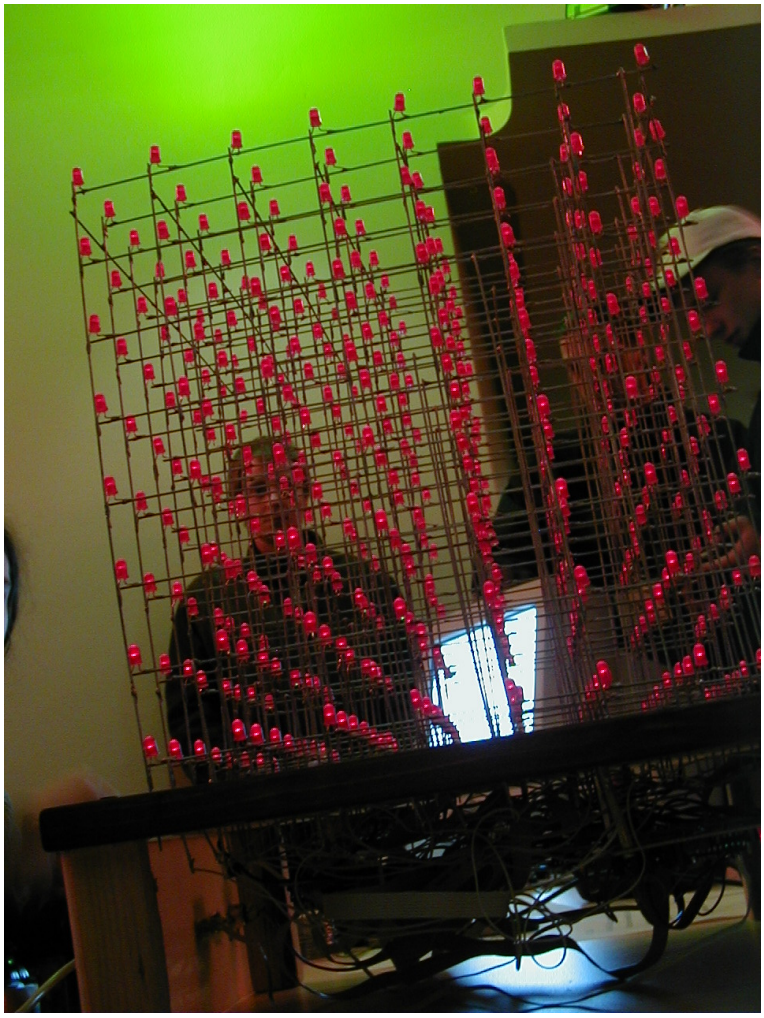

Abbildung 2.4: Borg 3D

Der Borg3D ist mit einer Größe von 512 LEDs relativ groß und wird trotzdem mit nur einem Mikrocontroller gesteuert. Dieser wird über 16 Ausgänge an die Schaltung des Cubes angeschlossen. Sowohl Schaltpläne als auch Quellcode sind öffentlich zugänglich und es existiert sogar eine Bauanleitung zum Löten des Drahtgittermodells. Leider ist dieses Modell nur schwer skalierbar, da der Aufbau der Matrix auf genau 512 LEDs ausgelegt ist. Eine Verkleinerung ist umständlich, eine Vergrößerung unmöglich. Trotzdem bietet dieser sehr gute Ansätze, insbesondere die elektrische Verschaltung betreffend - und das obwohl auch dieser nur fest vorgegebene Animationen darstellen kann, und keine externe Schnittstelle besetzt.

2.2.3 5^3 LED CUBE Controller von Picprojects

Der 5^3 LED Cube Controller ist kein vollständiger Cube, sondern ein gut dokumentierter, und frei erhältlicher 5x5x5 Singe-Color-LED-Cube Controller für den Microchip PIC16F688 Flash Mikrocontroller (Abbildung 2.5 und 2.6 links im Bild). Aufgrund der noch eher geringen Anzahl von LEDs, werden bei diesem Projekt nur die Ebenen gemultiplext, so dass eine Gesamtanzahl der Zuleitungen von genau 30 entsteht (25 für jede LED + 5 Ebenen). Das Besondere hierbei ist jedoch, dass hier zwei STP16CP05 16-Bit Konstantstrom LED-Treiber verwendet werden, die die Anpassung der Ausgangsströme zu den LEDs auf einem Level halten, und so nicht auf eine feste Aktualisierungsrate des Cubes angewiesen sind. Zu sehen sind diese auch in Abbildung 2.5 rechts. In der Abbildung 2.7 wurden diese unterhalb der Platine in SMD Bauweise aufgelötet.

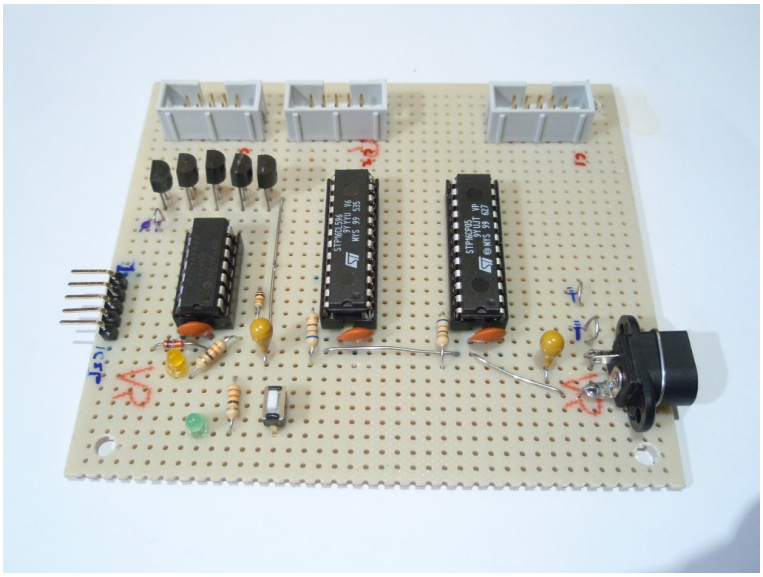

Abbildung 2.5: Prototyp des 5^3 Controllers

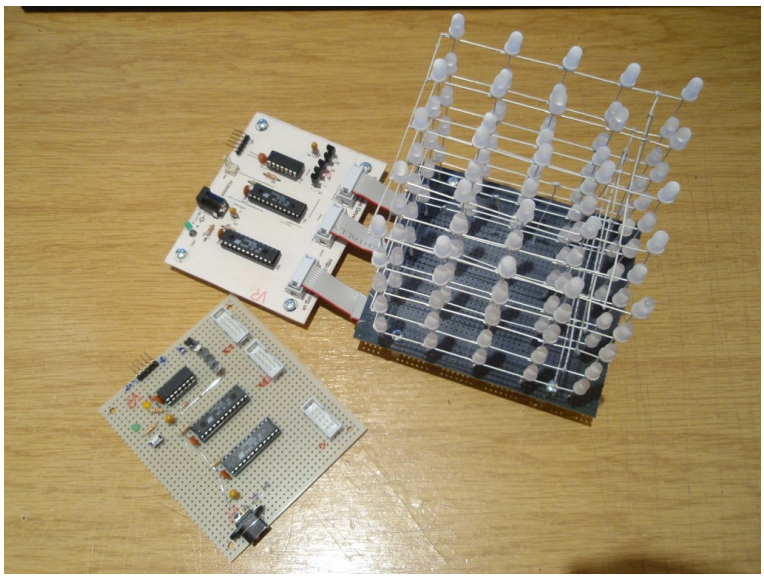

Abbildung 2.6: Prototyp des 5^3 Controllers mit Cube

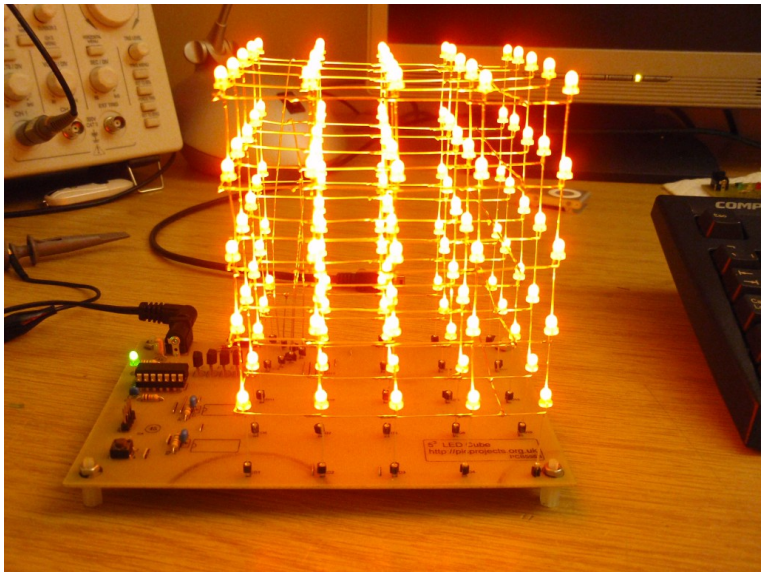

Abbildung 2.7: Testaufbau des 5^3 Cubes mit geätzter Platine und STP16CP05 LED-Treibern in SMD Bauform auf der Unterseite

Der 5^3 LED CUBE Controller ist eine platzsparende Alternative, um den theoretischen Aufbau eines LED Cubes zu visualisieren und einzelne Elemente zu testen. Er ist günstig in der Herstellung, quelloffen und dank der vorhandenen Bauanleitung mit relativ geringen Grundkenntnissen aufzubauen. Durch die Verwendung von Konstantstrom LED-Treibern fällt auch das Vorberechnen und das Einbauen der Vorwiderstände weg, weshalb der Cube auch für jede Farbe geeignet ist, da diese unterschiedliche Durchlassspannungen aufweisen.

2.2.4 Hypnocube v1.1 von Chris Lomont und Gene Foulk

Von Chris Lomont wurde Anfang 2005 ein 4x4x4 BiColor (rot/grün) Cube entwickelt, welcher von einem 32 Mhz Microchip PIC18F4620 gesteuert und komplett in C programmiert wurde (siehe Abb. 2.8). Anfänglich wurde jede LED einzeln beschaltet, später die Ebenen als Matrix zusammengefasst (siehe Abb. 2.9). Durch diesen Schritt war es möglich, die BiColor LEDs zu modulieren, so dass es zwischen rot und grün insgesamt 254 Zwischenschritte gab. Möglich war dies durch die Abstufung, ähnlich wie bei dem Borg3D Projekt, in jeweils 16 Schritten pro Farbe.

Das gesamte Konzept des Hypnocubes wurde Ende 2005 überarbeitet, so dass der Cube nun mit RGB LEDs vom Typ RL5-RGB-D bestückt wurde. Als Hauptprozessor dient weiterhin der PIC18F4620, welcher vier 74HC574N Oktal Typ-D FlipFlops treibt. Da jede LED nun intern aus drei einzelnen Farben besteht, welche jeweils mit 16 Helligkeitsstufen angesprochen werden kann, sind nun pro LED 4096 verschiedene Farben möglich.

Durch den Schritt von einem Testaufbau zur maschinell hergestellen Platine in Abbildung 2.10 war es so möglich, die LED Cubes in größeren Mengen herzustellen. In Verbindung mit einem Programmierinterface nach außen und einem Plexiglasgehäuse (siehe Abb. 2.11) sind die Cubes auch heute noch käuflich zu erwerben.

Leider ist der Cube mit 64 LEDs weder in irgendeiner Form erweiterbar, noch lassen sich eigene Programme darauf installieren. Weder liegt der Quellcode vor, noch könnte man eine veränderte Firmware installieren, da die Möglichkeit zur externen Programmierung deaktiviert wurde. Es lassen sich lediglich Animationen über eine proprietäre Software von außen aufspielen. Außerdem ist der Anschaffungspreis im Vergleich zu den verwendeten Bauteilen sehr hoch.

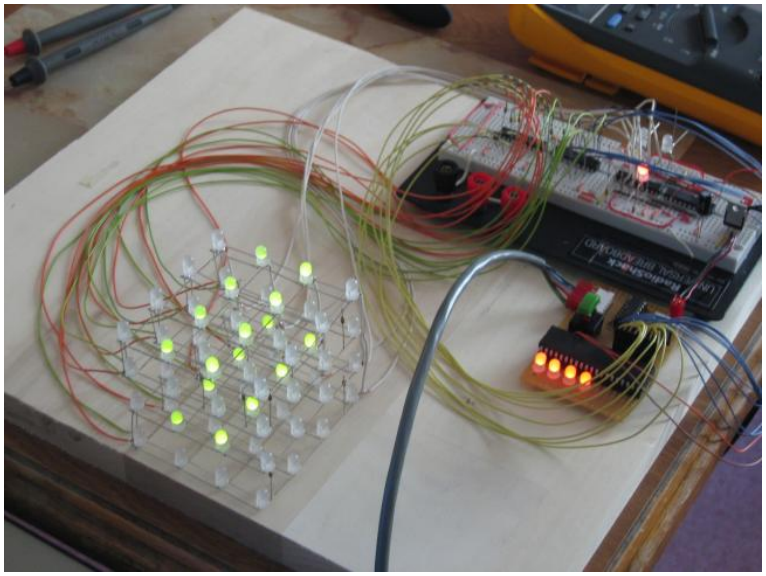

Abbildung 2.8: Prototyp des Hypnocubes im Aufbau auf einem Breadboard

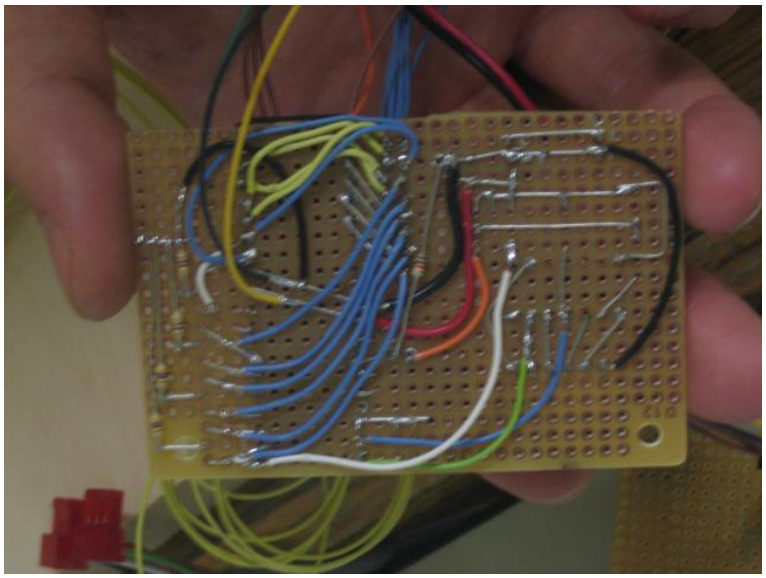

Abbildung 2.9: Prototyp des Hypnocube-Controllers auf einer Lochrasterplatine

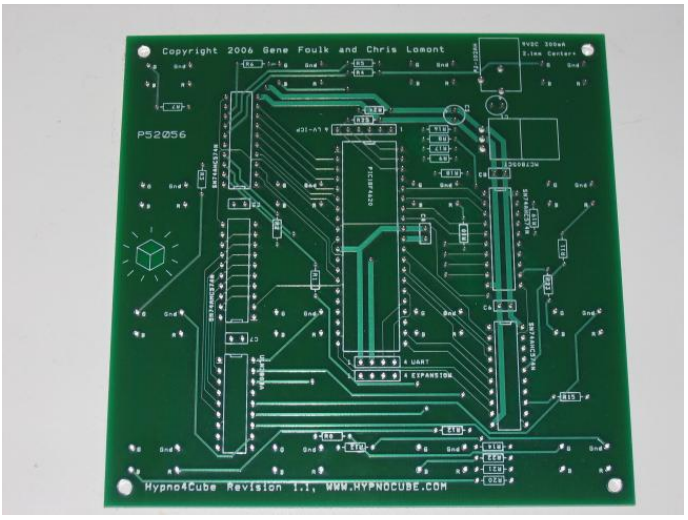

Abbildung 2.10: Maschinell geätzte Hauptplatine in Version 2.0 des RGB Hypnocubes

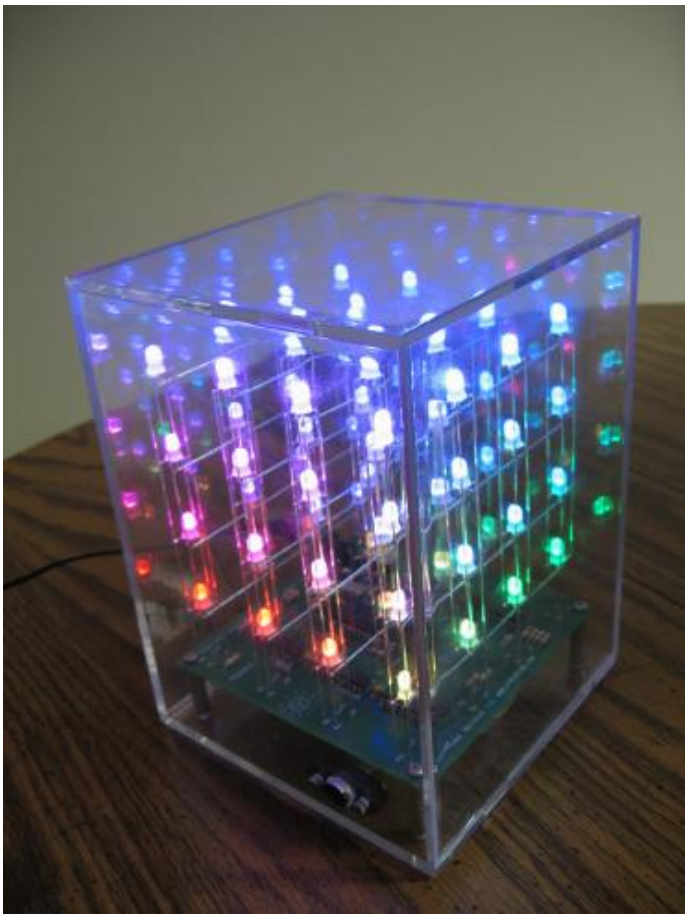

Abbildung 2.11: Endgültige Version des Hypnocubes im Plexiglasgehäuse

2.2.5 Cube3D von ultratechnik/techtalk

Der von Johannes Claudius Hagen entwickelte Cube3D (Abb. 2.12) ist ein vom Borg3D und Hypnocube inspirierter 4x4x4 RGB Cube auf Basis eines Atmel AVR ATmega32-16AI. Zur Ansteuerung der einzelnen LEDs in einer Ebene werden hier sechs 8-Bit Schieberegister vom Typ 74HC595D verwendet, welche wie bei anderen Cubes auch seriell geschaltet sind und so genau 48 Ausgänge bieten. Der Unterschied zu den vorangegangenen Cubes ist jedoch, dass hier nicht die LEDs komplett als Einheit angesprochen werden, sondern jede der Farben rot, grün und blau separat. In Abbildung 2.13 ist dies im Schaltplan ersichtlich.

Durch die relativ hohe Frequenz von 8 kHz und einer 6-Bit PWM Dimmung ist es so möglich, jeden Farbkanal in 64 Stufen zu kodieren, und so eine Gesamtanzahl von 262.144 Farben pro LED zu erzielen. Dieses Verfahren ist jedoch nur bei kleinen Cubes sinnvoll, da nicht nur die Anzahl der Leitungen, sondern vor allem die Frequenz kubisch ansteigen muss. Ein ATmega32 mit 16 Mhz, welcher auch die Ebenentreiber (pro Ebene ein MOSFET vom Typ IRF540N) schalten muss, befindet sich schon bei einem 4x4x4 Cube im zeitkritischen Bereich.

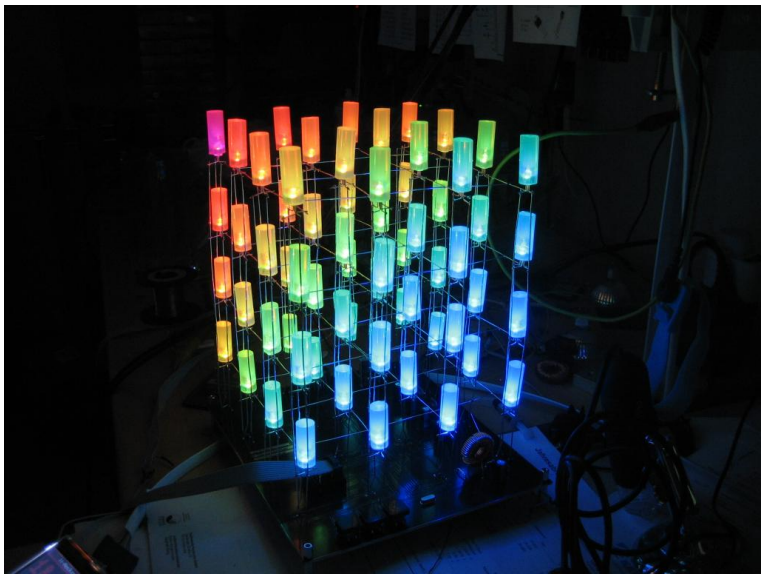

Abbildung 2.12: Vollständig aufgebauter Cube3D

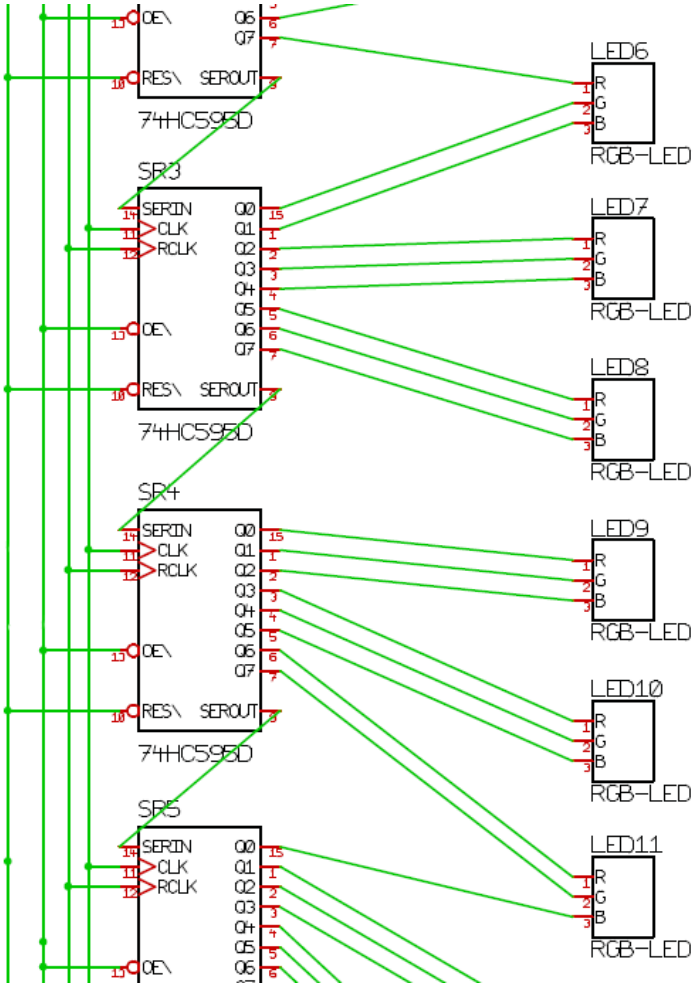

Abbildung 2.13: Ansteuerung der Farbkanäle über Schieberegister

Zu den Vorteilen dieses Cubes zählen das große Farbspektrum und die Möglichkeit der sehr flüssigen Animationen. Außerdem ist sowohl eine Bauanleitung, als auch der gesamte Quellcode vorhanden, was einen relativ einfachen Nachbau ermöglicht. Dieser erfordert durch den hohen Verdrahtungsaufwand jedoch viel Zeit und ist, wegen der vergleichsweise teuren Bauteile mit bereits 6 Schieberegistern für nur 64 LEDs, nicht für größere Projekte zu empfehlen.

2.2.6 eightCubed von Lumisense

Der von Ted Markson und Edmundas Balciunas entwickelte eightCubed (siehe Abb. 2.14) ist ein 8x8x8 RGB LED Cube auf Basis eines Microchip PIC30F4011, welcher in Abbildung 2.15 ersichtlich ist. Ein weiterer Microchip PIC18F4550 dient als Kommunikationseinheit und stellt eine USB Schnittstelle bereit, über welche die zu visualisierenden Daten beinahe in Echtzeit von einem PC zum Cube übertragen werden.

Beide Mikrocontroller bilden zusammen den Hauptcontroller für die gesamte Ansteuerung. Eine weitere Einheit bilden die 24 Schieberegister vom Typ M74HC595B1R (siehe Abb. 2.17). Sie dienen zur vertikalen Ansteuerung des Cubes und verfügen über TriState Ausgangspuffer, die den seriellen Datenstrom wieder parallelisieren. Die einzelnen Ebenen werden durch acht 2N6488 Leistungstransistoren (Abb. 2.16) geschaltet, so dass der gesamte Cube 60 mal in der Sekunde aktualisiert werden kann.

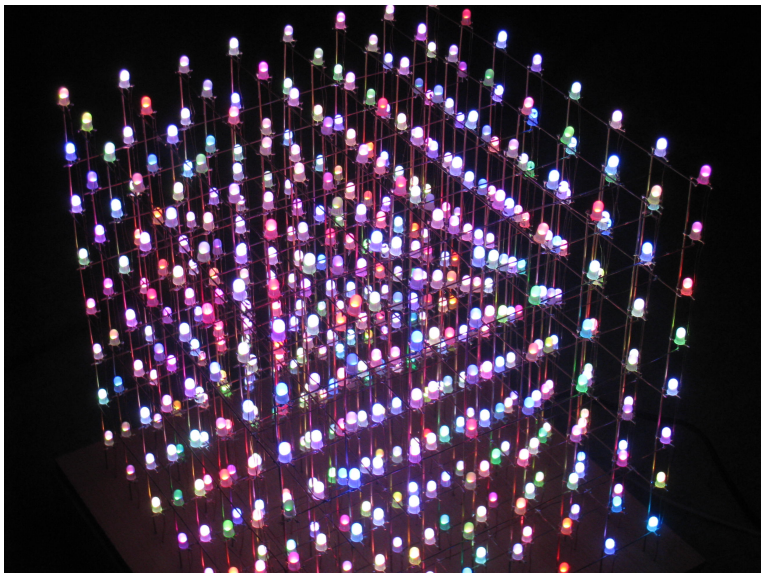

Abbildung 2.14: eightCubed

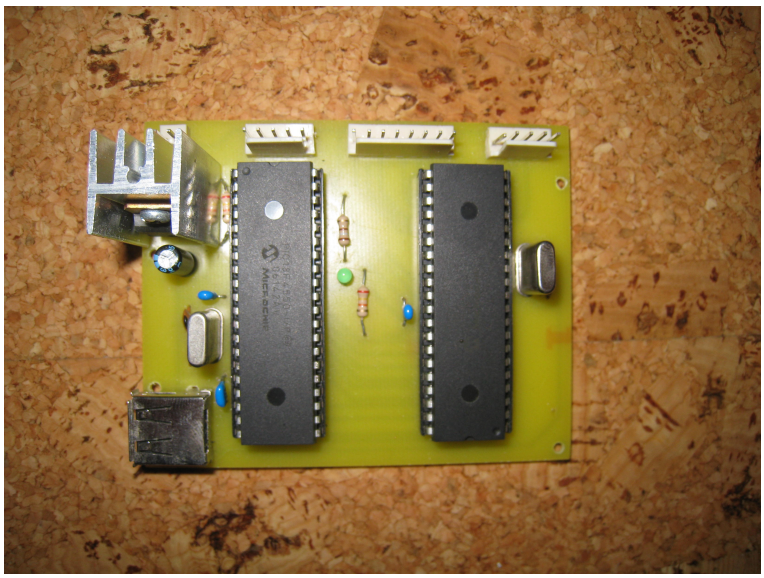

Abbildung 2.15: Hauptcontroller des eightCubed mit USB Anschluss

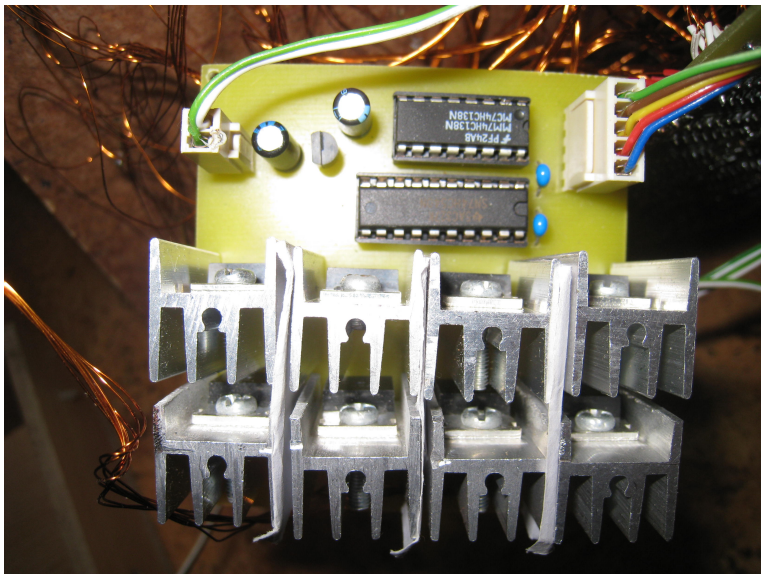

Abbildung 2.16: Ebenentreiber des eightCubed

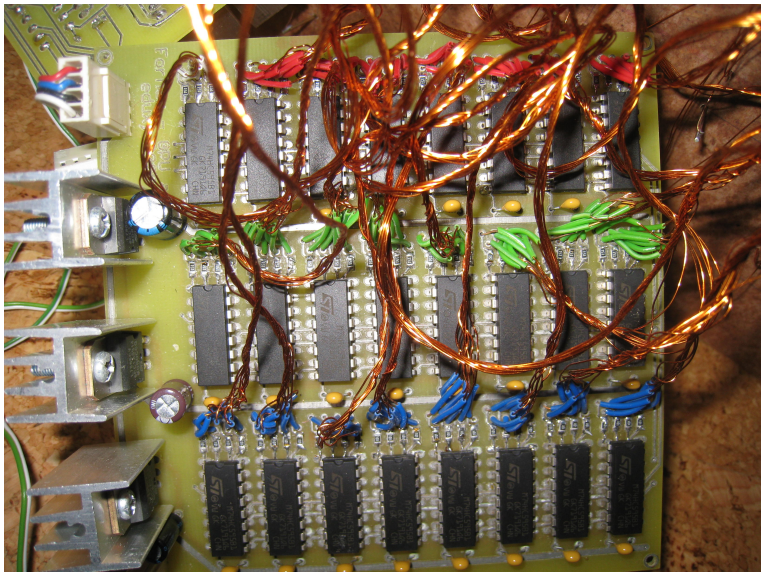

Abbildung 2.17: LED Treiber des eightCubed

Der eightCubed ist schon nah an den Anforderungen, die ein echtes 3D-Display und somit diese Arbeit stellt. Er ist mit 512 LEDs relativ groß und verfügt über ein Farbspektrum von 4096 Farben. Schaltpläne und Quellcode sind frei verfügbar. Des weiteren ist dieser Cube der einzige, welcher über eine Schnittstelle nach außen verfügt und somit in Echtzeit mit Bilddaten versorgt werden kann. Nachteilig ist, dass die LEDs durch die Verwendung von einfachen Schieberegistern noch Vorwiderstände für jede Farbe benötigen und zudem der Drahtgitteraufbau sehr fehleranfällig ist. Außerdem wird für die USB Realisierung ein veränderter *mchpusb.sys* Treiber verwendet, welcher nur unter Windows lauffähig ist.

2.2.7 A08/A16 3D LED Cube von Seekway Technology Ltd.

Seekway Technology mit Sitz in China ist die erste Firma weltweit, die RGB LED Cubes mit einer Größe von bis zu 16x16x16 in Serie fertigt und kommerziell anbietet. Anders als die oben genannten Projekten, werden hier keine Drahtmodelle aufgebaut, sondern alle LEDs auf Platinenstreifen aufgelötet, welche als identische Module (Ebenen) auf ein Mainboard aufgesteckt werden. Alles zusammen ergibt wieder die typische Würfelform. Auf jedem dieser Platinenstreifen befinden sich die zugehörigen LED Treiber vom Typ Macroblock MBI5026. Angesteuert wird jeder dieser Ebenen seriell, von einem auf dem Mainboard befindlichen Mikrocontroller vom Typ Atmel ATmega48. Anders als bei vielen anderen Cubes, sind die visuellen Effekte hier jedoch nicht hart in den Mikrocontroller programmiert, sondern werden über einen seriellen Datenstrom von einem sich ebenfalls auf dem Mainboard befindlichen Prozessor versorgt, der im Falle des A08 (8x8x8) Daten aus einer Secure Digital Karte ausliest. Auf dieser SD-Karte befinden sich die Animationen in Form einer Binärdatei.

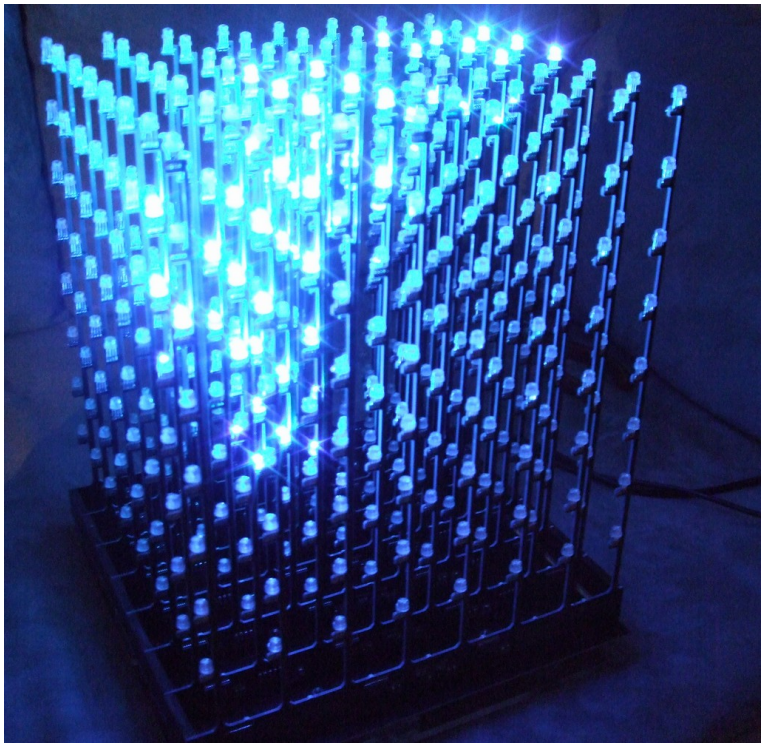

Abbildung 2.18: A08 von Seekway ohne Plexiglas-Gehäuse

Abbildung 2.19: Hauptplatine des A08

Der A08 von Seekway ist durch verfügbare Größe und die Farbanzahl von 4096 Farben beinahe allen Cubes technisch überlegen. Durch den modularen Aufbau, welcher komplett auf flexible Kabel verzichtet ist er theoretisch beliebig skalier- und reprogrammierbar. Die mitgelieferte Software, welche die die Binärdatei für die Animationen erzeugt ist zwar nicht quelloffen und zudem nur in chinesischer Sprache erhältlich, funktioniert jedoch.

Ein Nachbau ist allerdings unmöglich, da auch auf Anfrage keine Schaltpläne für die Öffentlichkeit verfügbar sind, und zudem die erzeugte Binärdatei auf der SD-Karte sowie der serielle Datenstrom zwischen den Prozesoren, mit einem proprietären Datenformati kodiert sind. Außerdem verfügt dieser Cube weder über eine externe Programmierschnittstelle, noch über eine Schnittstelle wie USB, mit der man Bilddaten streamen könnte.

2.2.8 Cubatron Jr. von 3waylabs

Die Cubatron-Serie ist eine Reihe von RGB Cube Projekten der Firma 3waylabs (ehemals Network Wizards) von Mark Lottor in Kalifornien. Der Cubatron Jr. (Abb. 2.20) ist dabei das größte Farb-RGB-Display der Welt und besteht aus einer 9x9x9 Matrix von LEDs, welche in in Tischtennisbälle (40 mm Durchmesser) eingelassen wurden und so die Pixel des Displays darstellen. Der Abstand zwischen diesen Leuchtkörpern beträgt etwa 255 Millimeter.

Jeder der 729 Leuchtkörper (Abb. 2.21) beinhaltet einen eigenen Mikrocontroller vom Typ PIC12F629 und einen MOSFET zur Ansteuerung der LED. Jeweils 27 dieser Leuchtkörper wurden zu einem von 27 Verbunden zusammengefasst und zusammen über ein proprietäres Datenprotokoll angesteuert.

Der Datenstrom für den Cube wird von einem PC bereitgestellt und über ein Netzwerkkabel an einen Print-Server geschickt, welcher den TCP/IP Datenstrom wieder parallelisiert und an den sogenannten "Voxel Driver", der Hauptsteuerplatine (Abb. 2.22) des Cubes, schickt. Dieser Hauptcontroller besteht hauptsächlich aus einem PIC18F452, welcher die eingehenden Daten auflöst und an die 27 Zusammenschlüsse des Cubes schickt. Die Updatefrequenz des gesamten Cubes beträgt, je nach Farbtiefe, 30 bis 50 Bilder pro Sekunde. (Lottor 2008, vgl.)

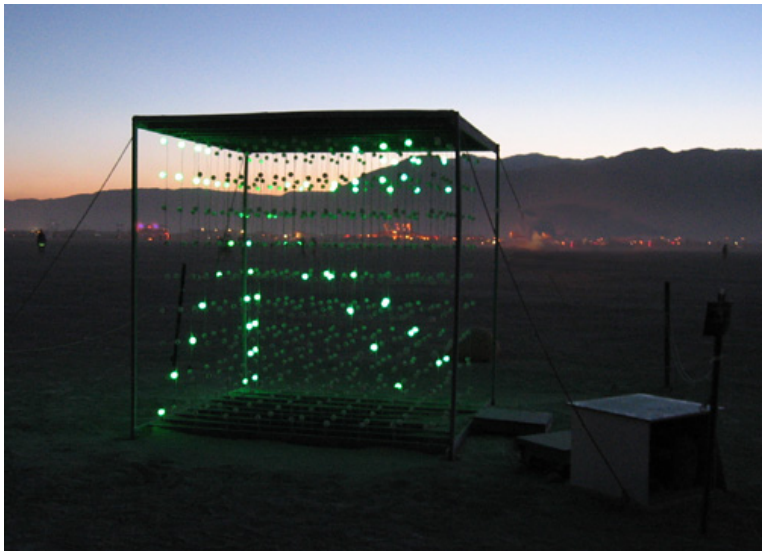

Abbildung 2.20: Cubatron Jr. mit Testprogramm

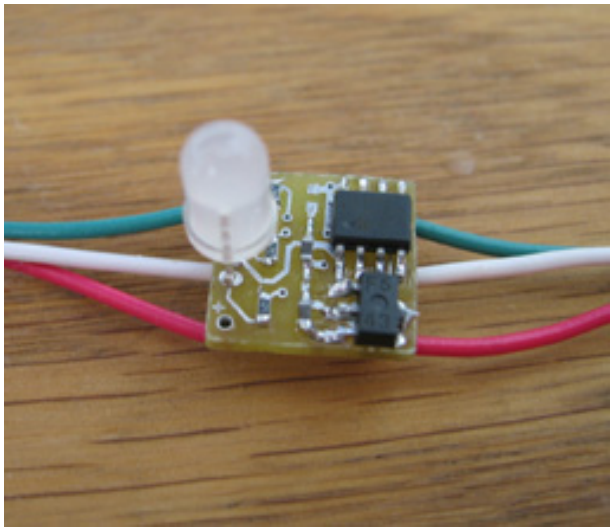

Abbildung 2.21: Innenleben eines einzelnen Leuchtkörpers mit LED, Mikrocontroller und MOSFET

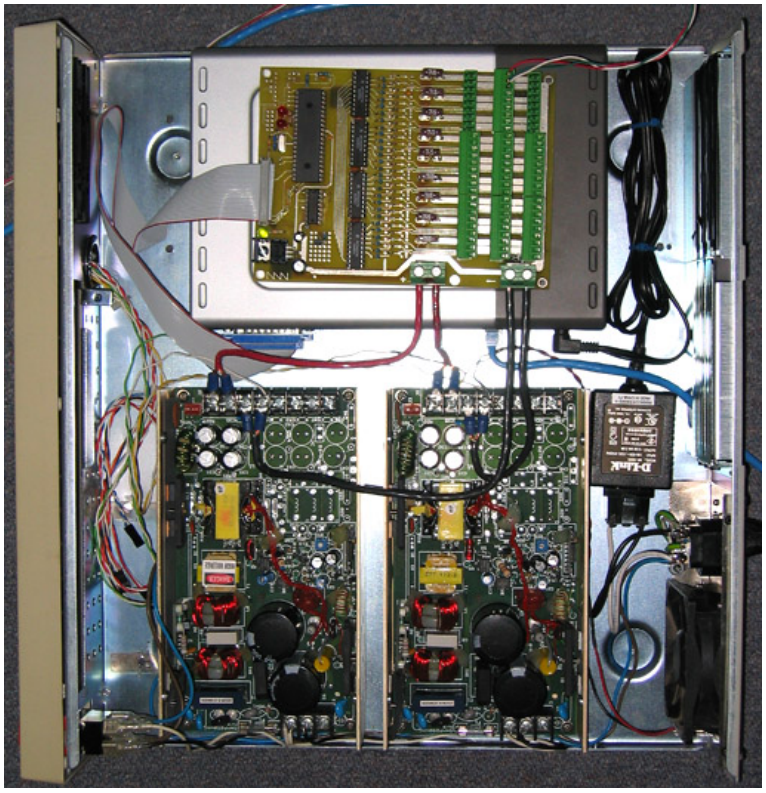

Abbildung 2.22: Innenansicht der Treiber-Box

Der Cubatron Jr. ist nur eines der Beispiele aus der Cubatron Serie von Mark Lottor. Er bietet interessante Lösungsmöglichkeiten für die Vernetzung von LEDs über größere Distan-

zen und wäre als Anzeigemedium aus sehr großen Entfernungen tatsächlich einsatzfähig. Der Cube verfügt über insgesamt 729 LEDs und durch die integrierten Controller in jedem Leuchtkörper ist dieser auch erweiterbar. Das hohe Farbspektrum von 2.097.152 Farben (21 Bit) oder 4096 Farben (12 Bit) und der Ethernet-Anschluss steigern nochmals die Attraktivität dieses Projektes. Lediglich der enorme Herstellungsaufwand und der hohe Preis disqualifizieren das Projekt für den Einbezug in diese Bachelorarbeit. Außerdem ist es aufgrund des Aufbaus der Leuchtkörper kaum möglich, diese kleiner zu fertigen, was somit immer in einer Cube-Größe von ca. 2x2x2 Metern resultiert.

2.3 Fazit der Analyse

	Display Cube V1	Borg3D	5^3 Control	Hypnocube	Cube3D	eightCubed	A08	cubatron jr.
Anzahl Farben	1	1	1	4096	262144	4096	4096	2097152
Anzahl LEDs	1000	512	125	64	64	512	512	729
Hardware Erweiterbar	Ja	-	-	-	-	-	Ja	ja
Schaltplan vorhanden	-	Ja	Ja	-	Ja	Ja	-	Ja
PCB Layout vorhanden	-	-	-	-	-	Ja	-	-
Quellcode vorhanden	-	Ja	Ja	-	Ja	Ja	-	-
Externe Schnittstelle	-	-	-	-	-	USB	-	Ethernet
Plattform-unabhängig	-	-	-	-	-	-	-	Ja

Tabelle 2.1: Übersicht bekannter Cube-Projekte

Wie in Tabelle 2.1 zu erkennen ist, kann keines der in diesem Kapitel genannten Designs die Anforderungen vollauf erfüllen. Zwar existieren einige wenige Projekte, wie beispielsweise der *eightCubed* oder *cubatron*, die bereits über eine Schnittstelle verfügen, doch diese sind entweder sehr aufwändig nachzubauen, oder können nicht erweitert werden. Seekways *A08* dagegen bietet zwar eine ausgereifte Technik und könnte durch den modularen Aufbau erweitert werden, doch ohne Dokumentationen und ohne Schnittstelle nach außen, ist dieser Cube nur als Anhaltspunkt für das spätere Design verwendbar. Eine Eigenentwicklung auf Basis der gesammelten Erkenntnisse scheint also am sinnvollsten, damit alle Vorteile vereint und die Nachteile minimiert werden.

3 Design

Dieses Kapitel befasst sich mit der Entscheidung, welche Komponenten, welche Hardware und welche Software von Nöten ist, um den Anforderungen gerecht zu werden.

3.1 Anforderungen (2)

Bei der Wahl des optimalen Designs zur Ansteuerung und zum Aufbau eines Cubes und zur gleichzeitigen Erfüllung der Anforderungen aus Kapitel 2 sind viele Aspekte von Bedeutung. Um den Verdrahtungsaufwand zu minimieren stellt die Seekway-Version einen guten Ansatz dar, da sie völlig ohne flexible Kabel auskommt. Es müssen 8 Ebenen gefertigt werden, die jeweils über 64 LEDs verfügen. Diese sollen identisch aufgebaut und somit austauschbar sein. Dies ermöglicht die Skalierung für spätere, größere Projekte. Die Größe des Cubes gibt daher die Grundplatte oder Hauptplatine vor, welche den eigentlichen Controller enthält. Dieser muss über eine Schnittstelle verfügen, um von außerhalb neue Animationsdaten erhalten zu können. Als externe Schnittstelle bietet sich ein serieller Bus an, da dieser die geringste Anzahl an Leitungen benötigt und dadurch kostengünstig ist.

Als serieller Bus kommt sowohl der Universal Serial Bus (USB) als auch Ethernet in Frage, da diese Schnittstellen beide einheitlich in heutigen Personal Computern vorhanden sind. Auch die die freie Wahl des Betriebssystems (Plattformunabhängigkeit), ist bei beiden Schnittstellen gegeben, da insbesondere Ethernet keinen Treiber erfordert. Allerdings ist zu beachten, dass Ethernet immer einen gewissen Konfigurationsaufwand mit sich bringt. Es müsste ein Protokoll wie TCP/IP implementiert werden, welches erfordert, dass für das Endprodukt eine IP Adresse konfiguriert wird, um Konflikte mit anderen Netzwerkkomponenten zu vermeiden. Die andere Möglichkeit zur Konfiguration wäre ein DHCP Client, welcher automatisch die erforderlichen Daten bezieht. Dieser setzt einen DHCP Server voraus, welcher auf den wenigsten eigenständigen Computern arbeitet. Aus diesen Gründen ist USB die bessere Lösung, da hier keine Vorkonfiguration nötig ist, um auf das Endgerät zuzugreifen. Die Plattformunabhängigkeit ist zwar nicht hundertprozentig gegeben, da es möglich ist einen proprietären Treiber zu verwenden, doch wenn man sich an Standard-USB-Klassen hält und diese implementiert, wird beinahe jedes Betriebssystem unterstützt.

Die USB Funktionalität kann über verschiedene Wege in das Hardwaredesign integriert werden. Zum einen über einen externen USB-zu-UART Chip (z.B. FTDI FT232), zum anderen über einen Mikrocontroller, welcher schon über eine integrierte USB Funktionalität verfügt.

Die erste Möglichkeit ist aufwendiger in Hardware zu realisieren, da noch zusätzliche Bauteile erforderlich sind, doch aufgrund der Tatsache, dass keine weitere Programmierung nötig ist, bleibt dies die einfachste Methode eine USB Schnittstelle in Hardware zu implementieren. Nachteilig dabei ist jedoch die Geschwindigkeit. Wird RS232 Kompatibilität vorausgesetzt, liegt die maximale Übertragungsrate bei 1 Megabit pro Sekunde. Dies kann bei der geforderten Datenmenge einen Flaschenhals darstellen, weshalb es sinnvoller ist, auf einen Mikrocontroller mit integrierter USB Unterstützung zu setzen. Hier sind theoretische Datenraten von bis zu 12 MBit/s möglich.

3.2 Wahl der Komponenten

3.2.1 Hauptprozessor

Um die genannten Anforderungen zu erfüllen, ist ein Hauptprozessor mit integrierter USB Schnittstelle nötig. Hier gibt es verschiedene Alternativen beispielsweise von Fujitsu, Cypress, Atmel, Microchip und Philips/NXP. Während hier die verfügbaren Controller von Fujitsu und Cypress meist auf spezielle Einsatzgebiete ausgelegt sind und nur wenige Zusatzfunktionen bieten, können die Mikrocontroller von Atmel, Microchip und NXP für nahezu jedes Kleinprojekt eingesetzt werden. Die Entscheidung, welcher Controller nun am sinnvollsten ist, liegt im Auge des Betrachters. Grundsätzlich muss man sich aber zwischen hauseigenen Kernen, beispielsweise dem *AVR* von Atmel oder *PICmicro* von Microchip, und *ARM*-basierten Controllern entscheiden. Der Unterschied ist hierbei, dass AVR und PICmicro Kerne direkt vom Chiphersteller entwickelt wurden, während z.B. der Kern eines NXP LPC2x oder Atmel AT91SAMx auf einem Kern der Firma ARM basieren. Alle drei genannten Prozessorkerne basieren zwar auf einer 8, 16 oder 32 Bit RISC Architektur, unterscheiden sich aber in ihrer Programmierung und besonders auch (Herstellerabhängig) in ihrer Dokumentation. Letztere ist gerade bei Atmel AVR sehr umfangreich.

Aus den genannten Gründen kommt als Hauptprozessor ein Atmel AT90USB1287[1] zum Einsatz. Dieser soll sowohl die serielle Ansteuerung der LED Treiber und der Leistungstransistoren, als auch die Verarbeitung des Datenstroms über die USB Schnittstelle übernehmen.

[1] Der hier verwendete AT90USB1287 verfügt über zwei Betriebsmodi der USB Schnittstelle. Im USB-Device-Modus arbeitet der AVR als frei programmierbares USB Endgerät, welches mit einem USB Host verbunden werden kann. Der AVR kann hierbei entweder in eine Standard-Geräteklasse eingeteilt oder über einen proprietären Treiber angesprochen werden. Im zweiten Modus, dem „USB-on-the-go“ (OTG) arbeitet der AVR selbst als USB Host und kann verschiedene USB-Endgeräte direkt mit einer Punkt-zu-Punkt Verbindung ansteuern. Für diese Arbeit bleibt aber nur der Device-Modus relevant, da der Cube von einem PC aus mit Bilddaten versorgt werden soll. Durch eine spätere Anpassung des Programms auf dem AVR wäre aber die Implementierung des OTG Modus nachträglich möglich um beispielsweise vorgefertigte Bilddaten und Animationen von einem USB Massenspeichergerät auszulesen und ganz auf einen zusätzlichen Host-PC zu verzichten.

Der AT90USB1287 (Blockschaltbild in Abbildung 3.1) verfügt über eine 8 Bit RISC CPU, 128 Kilobyte Flash Speicher, ein 4 Kilobyte EEPROM und 8 Kilobyte internes statisches RAM, welches jedoch extern mit maximal 64 Kilobyte erweitert werden kann. Ein großer Vorteil des Mikrocontrollers der AT90USB-Serie ist die Möglichkeit, diesen auch über USB zu programmieren und so Firmwareupdates ohne JTAG/ISP Interface durchzuführen.

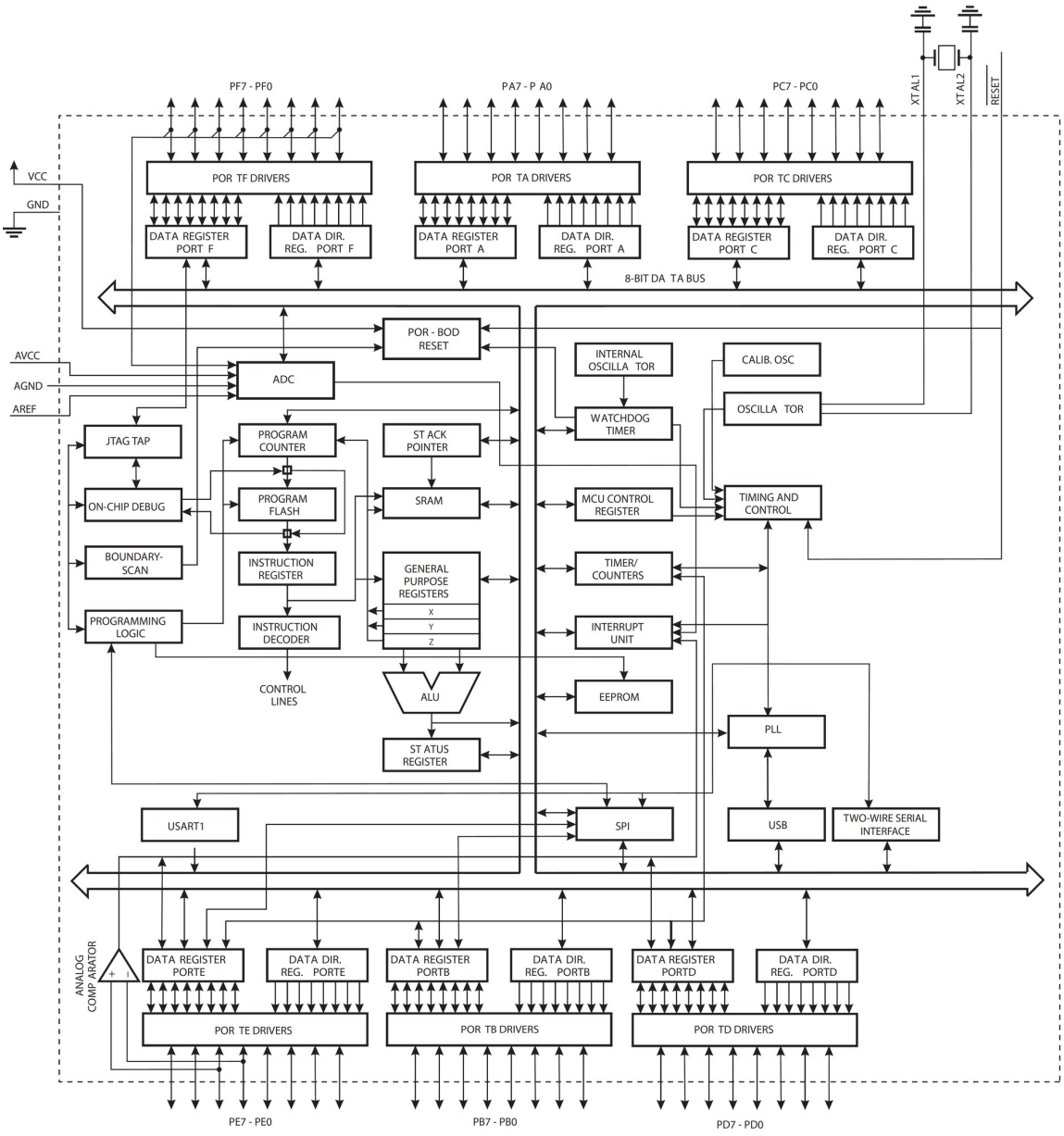

Abbildung 3.1: Blockschaltbild des AT90USB1287

Unter der Annahme, dass jede LED mindestens 4096 Farben darstellen soll, ist es nötig, jede der drei Grundfarben mit 4 Bit Pulsweitenmodulation anzusteuern, um eine Abstufung in je-

weils 16 Schritten zu ermöglichen. Folglich werden für jede LED zur Laufzeit 12 Bit Speicher benötigt. In der Theorie reicht der Speicher des AT90USB1287 also aus, da ein Cube mit 512 USBs für einen kompletten Durchlauf 768 Byte Speicher benötigt. Es muss jedoch bedacht werden, dass dies nicht die einzige Aufgabe des Controllers bleibt und besonders für die Implementierung der USB Schnittstelle noch mehr Arbeitsspeicher benötigt wird. Des Weiteren können bei den verwendeten Computerarchitekturen keine Speicherbereiche kleiner als 1 Byte (8 Bit) angelegt werden, was zur Folge hat, dass zum Speichern des Wertes von einer LED kein passender Datentyp verfügbar ist, da ein solcher immer aus einem Vielfachen von 8 Bit bestehen muss. Es bleibt die Frage offen, ob es sinnvoll ist, alle LEDs gleichzeitig in einen 768 Byte großen Datenbereich zu speichern - oder aber jeweils 16 Bit für jede LED vorzusehen, um eine einfachere Adressierung eben dieser zu ermöglichen. Die Gesamtgröße des Arbeitsspeichers muss folglich 1 Kilobyte betragen.
Es bleibt zu beachten, dass in einem Kilobyte auch nur ein einziges statisches Bild gespeichert werden kann. Wird aber angenommen, dass für eine Ruckel- und Flimmerfreie Darstellung, der Cube bis zu 60 mal in der Sekunde aktualisiert werden muss, so erfordert dies einen kontinuierlichen Datenstrom von 46,08 KB/s respektive 60 KB/s für die Rohdaten der LEDs. Hinzu kommen Steuerzeichen und/oder Bitmuster für die Adressierung und Synchronisierung der Bilder. Wieviel Bandbreite also tatsächlich von dem USB abverlangt wird hängt von der Ausgestaltung des Datenprotokolls ab.

Eine wichtigere Frage als die Speicher- und Bandbreitennutzung, muss die Frage nach der Geschwindigkeit des Mikrocontrollers sein:
Um eine einzelne LED innerhalb einer Ebene zu setzen, müssen über insgesamt 64 Takte sowohl ein einzelnes High-Bit, als auch 63 Low-Bits in die Schieberegister der LED Treiber geschrieben werden. Dieser Vorgang wiederholt sich, wenn eine andere LED aktiviert werden soll - in diesem Fall 64 Mal für eine Ebene.

$$64 * 64 * 60 = 245.760$$

Wie zu sehen ist, sollte der interne Timer, welcher die Interrupt Service Routine (ISR) anstößt mit etwa 246 Khz arbeiten. Der Teiler wird hierbei auf 65 eingestellt, da der Referenztakt des Mikrocontrollers 16 Mhz beträgt.

$$16.000.000/65 = 246.154$$

Wie man sieht, kommt ein 16 Mhz Mikrocontroller bereits bei einer Ebene in den zeitkritischen Bereich. Zwar können theoretisch in der ISR nun 65 elementare Befehle ausgeführt werden, doch das Sichern und Zurückkopieren der Register beim Einsprung und Verlassen der ISR erfordert vergleichsweise viel Rechenzeit. Diese Zeit darf nicht zu lang sein, da der AT90USB1287 noch auf den USB-General-Interrupt-Vector und den USB-Endpoint/Pipe-Interrupt-Vector reagieren muss. Werden diese Interrupts nicht zeitnah behandelt oder ganz

übergangen, so ist die USB Funktionalität nicht mehr sichergestellt. Es kann also nötig sein, die ISR komplett in Assembler zu programmieren um das Sichern der Register zu umgehen. Folglich können diese Register dann nicht mehr verwendet werden. Eine andere Möglichkeit, Rechenzeit zu sparen, ist das „Schieben“ des High-Bits. Hierbei wird eine Eins am Anfang von einem Ebenen-Zyklus generiert, die dann von 63 Nullen gefolgt durch die Schieberegister *wandert*. Diese Methode setzt aber voraus, dass das das Schieben niemals sichtbar wird und im Schiebeprozess die Ausgänge abgeschaltet werden. Welche der Möglichkeiten also die bessere ist und ob es noch andere Alternativen gibt, werden die weiteren Kapitel zeigen.

3.2.2 Pegelwandler/Signalverstärker

Die frei programmierbaren Ein- und Ausgänge des AT90USB1287 sind leider nicht stark genug belastbar, um eine Schaltung dieser Größenordnung direkt anzusteuern. Der Signalpegel an den LED Treibern wäre nicht mehr eindeutig. Außerdem riskiert man eine Überlastung der Ausgangstreiber des Atmel AVR.
Aus diesem Grund wird ein Tri-State Oktal D-Typ Transparent Latch als Signalverstärker verwendet, da dieser Baustein bereits acht verfügbare Ein- und Ausgänge besitzt. Intern besteht dieses Latch aus einer Kaskade von Operationsverstärkern, die alle gleichzeitig über ein Latch-Enable Signal aktiviert werden können. Zusätzlich kann der Ausgang über den OE-Pin aktiviert- und deaktiviert werden.

Bei der Wahl eines Transparent Latches fiel die Entscheidung auf den 74HC573 von Philips. Dieser kann an jedem Ausgang bis zu 35 Milliampere schalten und verwendet die eigene Versorgungsspannung zur Signalerzeugung. Abbildung 3.2 zeigt den internen Aufbau und die Tabelle 3.1 die Pinbelegung dieses ICs. Die Schaltzeit beträgt bei der verwendeten Versorgungsspannung von 5 Volt im Normalfall weniger als 6 Nanosekunden, was eine maximale theoretische Taktfrequenz von 166 Mhz bedeutet. Dieser Wert wird jedoch in diesem Szenario niemals erreicht, weshalb dieser Baustein keinesfalls zum Engpass der Schaltung wird.

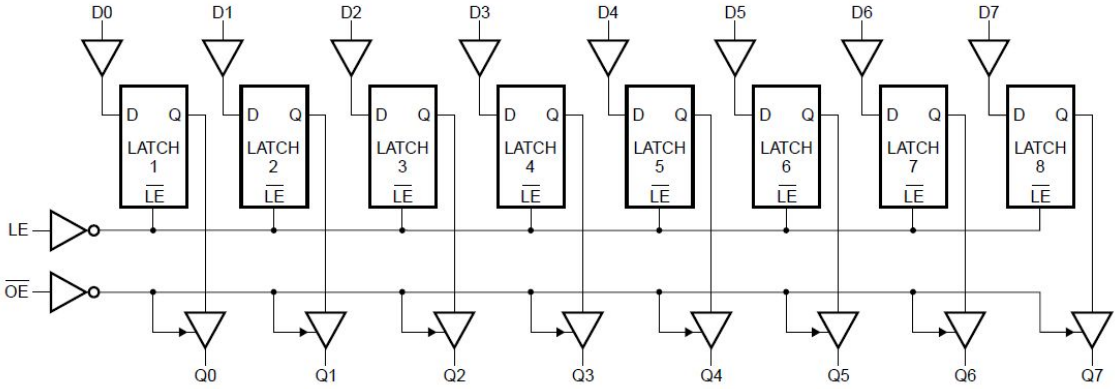

Abbildung 3.2: Der logische Aufbau des 74HC573

Pin	Bezeichnung	Funktion
1	OE	TriState Output Enable
2-9	D0-D7	Daten-Eingang
10	GND	Masse (Negativ)
11	LE	Latch-Enable (High-Aktiv)
12-19	Q7-Q0	TriState Latch Output/Daten-Ausgang
20	VCC	Versorgungsspannung (max. 7 Volt)

Tabelle 3.1: Pinbelegung des 74HC573

3.2.3 LED Treiber

Als LED Treiber können verschiedene baugleiche Komponenten verwendet werden, wie beispielsweise der STP16CPS05 von STMicroelectronics, MBI5026CP von MBI oder der TLC5925 von Texas Instruments. Es handelt sich hierbei um getaktete 16-Kanal Konstantstrom LED Treiber, welche jeweils hauptsächlich aus einem 16-Bit Schieberegister mit nachgeschaltetem 16-Bit Latch bestehen. Die Ausgänge sind speziell bei diesen Treibern noch mit einer Schaltung versehen, die den Ausgangsstrom in Abhängigkeit von einem externen Widerstand R-EXT regeln und so im gewünschten Bereich zwischen 3 und 45 Milliampere konstant halten. Aufgrund der besseren Verfügbarkeit, kommt bei diesem Projekt der TLC5925 zum Einsatz, dessen logischer Aufbau in der Abbildung 3.3 zu sehen ist. Um die Funktionsweise und das Verhalten der Ausgänge zu verdeutlichen, zeigt Abbildung 3.4 das zugehörige Timingdiagramm. Hieraus ist ersichtlich, wie dieser Baustein arbeitet und in welchem Zusammenhang die parallelen Ausgänge OUT0 bis OUT15 zum seriellen Eingang SDI

stehen. SDO ist hier der Serial-Data-Output Pin, welcher das Signal von LED nach Durchlaufen des Schieberegisters weiterreicht.

Bei der Verdrahtung zwischen den LED Treibern und den LEDs gibt es zwei Möglichkeiten. Zum einen können jeweils drei Ausgänge direkt an die einzelnen R, G und B Pins der LED angeschlossen werden, was bedeutet, dass man für eine Ebene aus 64 LEDs folglich 12 Treiber mit insgesamt 192 Ausgängen benötigt. Der Vorteil ist, dass so das größtmögliche Tastverhältnis von 1:1 erreicht werden kann. Allerdings erfordert diese Konfiguration auch, dass alle 192 Leitungen direkt zu den LED Treibern führen, was bei einer reinen Leiterplattenlösung zu einen sehr hohen Routingaufwand führt und auch sehr dünne Leiterbahnen voraussetzt. Diese Lösung setzt LEDs mit einer gemeinsamen Anode voraus.
Die zweite Möglichkeit ist das Multiplexing der Farben. Alle *gleichfarbigen* Pins der LEDs werden zusammengefasst und so für jede LEDs die Möglichkeit gegeben, eine der 3 Grundfarben auszuwählen. Unter der Voraussetzung, dass LEDs mit einer gemeinsamen Kathode verwendet werden, können diese dann über die LED Treiber ein- und ausgeschaltet werden. Folglich werden für jede Ebene nur noch 4 Treiber benötigt - allerdings liegen bei jeder Ebene die zusammengefassten Anoden der Farben in Form von 3 einzelnen Pins frei. Diese benötigen daher noch einen zusätzlichen Treiber.

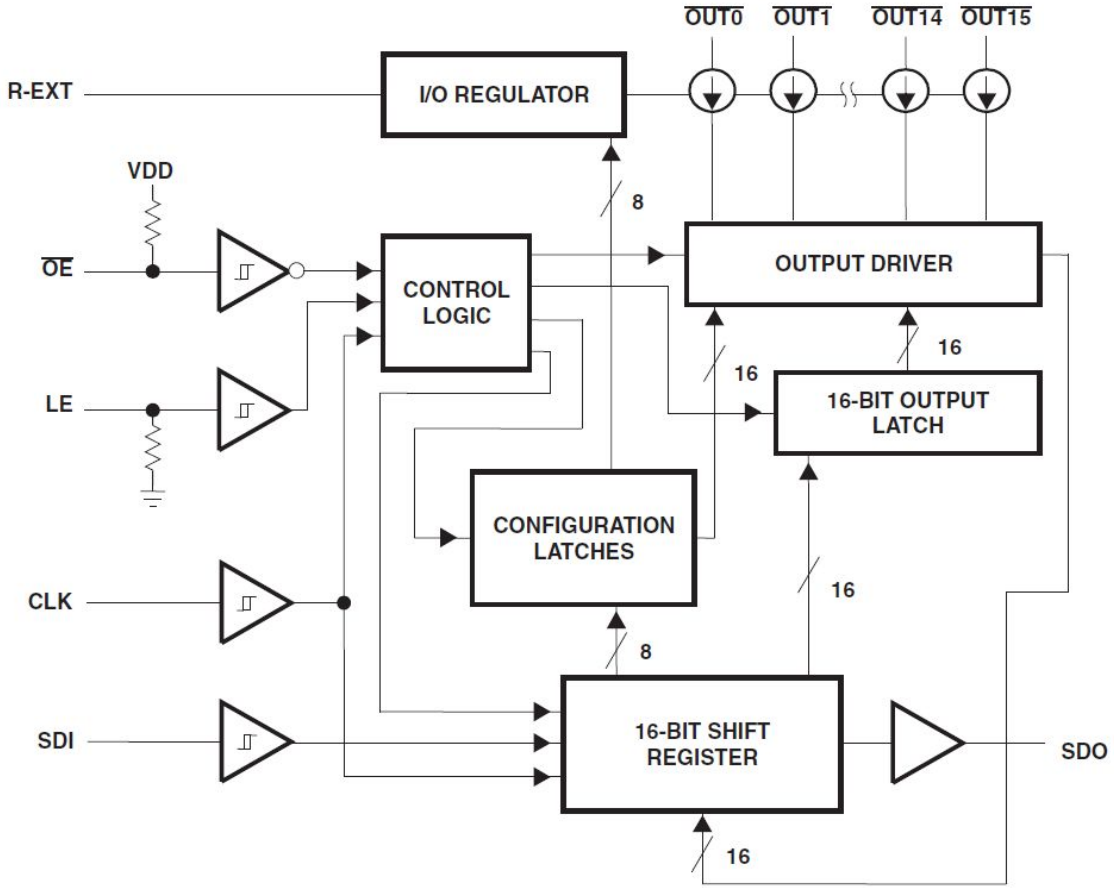

Abbildung 3.3: Der logische Aufbau des TLC5925

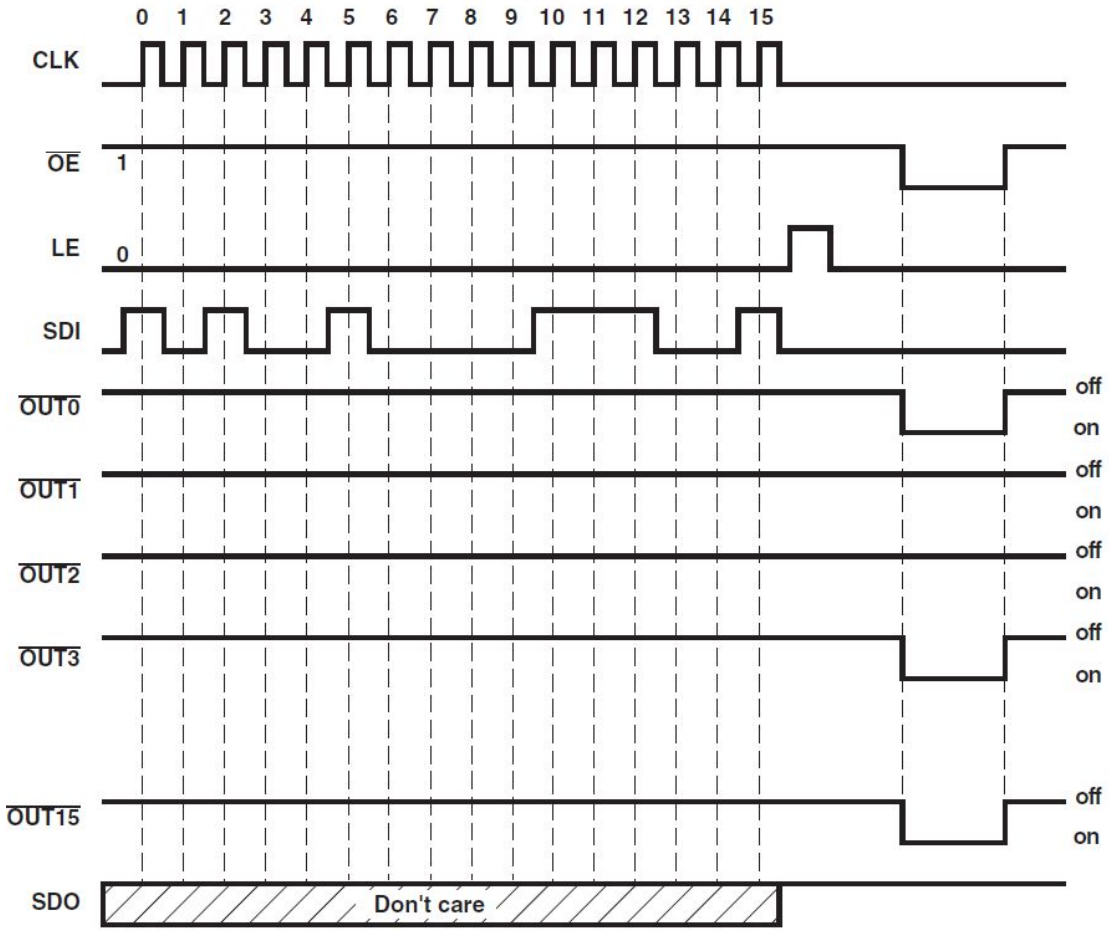

Abbildung 3.4: Timingdiagramm TLC5925

3.2.4 Farb-Treiber (Leistungstransistor)

Da die einzelnen LEDs nun über die oben genannten TLC5925 getrieben werden, und somit die Kathode (negativ) gegen Masse gezogen wird, fehlt noch der positive Pol zur Anode. Dieser soll über einen Metall-Oxid-Feldeffekttransistor realisiert werden, welcher als getakteter Schalter dient.

Obwohl allgemein jeder N-Kanal MOSFET leistungsfähiger ist, brächte er in diesem Fall Probleme mit sich. Bei einem MOSFET der geforderten Größe mit n-dotiertem Halbleitermatierial, muss zum Schalten ein positives Potenzial von mindestens 3 Volt zwischen Gate- und Source-Spannung bestehen. Im Fall des LED Cubes, beträgt die Source-Spannung aber maximal 5 Volt, was voraussetzen würde, dass die Gate-Spannung mindestens 8 Volt beträgt. Diese Spannung kann jedoch nicht von dem verwendeten AT90USB1287 bereitgestellt werden.

Als Lösung werden deshalb zwei dual P-Kanal MOSFETs eingesetzt, welche ein negatives Potenzial von mindestens 3 Volt zwischen Gate- und Source-Spannung erfordern um zu Schalten. Um eine Ebene zu versorgen genügt es also, das Gate vom MOSFETs auf Masse zu ziehen, und so eine Spannung von -5V zu erzeugen. Der Strom wird dann von Source zu Drain durchgeschaltet.

Bei der Wahl des geeigneten P-Kanal MOSFETs fiel die Entscheidung auf den PHP225 von Philips/NXP. Hierbei handelt es sich um einen Dual P-Kanal MOSFET im SOT96-1 (SO8) Gehäuse, welcher durch den doppelten internen Aufbau zwei Farben gleichzeitig schalten kann (siehe Abb. 3.5). Für die dritte Farbe ist somit nur noch ein zusätzlicher MOSFET nötig.

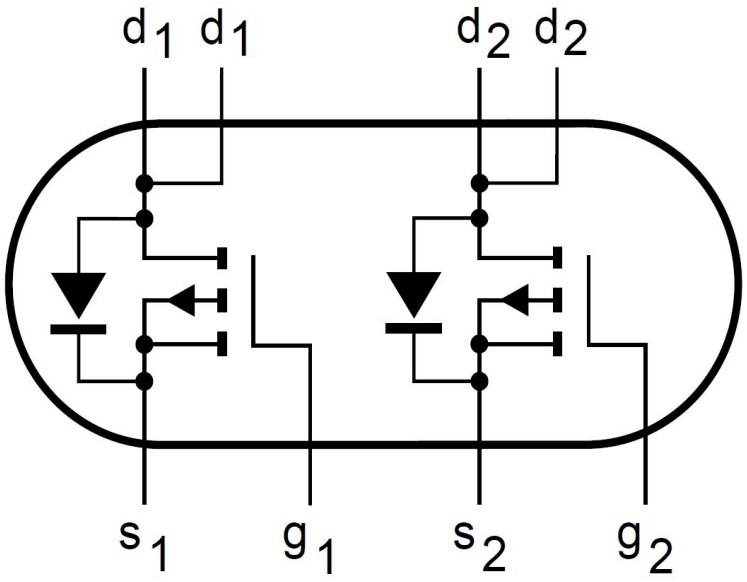

Abbildung 3.5: Interner Aufbau eines PHP225

3.2.5 Leuchtdiode

Bei der Auswahl für den Cube geeigneter LEDs kommen mehrere Faktoren zum Tragen. Insbesondere sollen die LEDs einen großen Abstrahlwinkel sowie eine hohe Helligkeit aufweisen, sowie über ein 4-Pin Gehäuse mit gemeinsamer Anode verfügen. Weiterhin sollen die Betriebsströme der einzelnen Farben möglichst ähnlich sein, um eine einfachere PWM Ansteuerung zu gewährleisten.

Aus diesen Gründen werden für den Cube die mattierten und abgeflachten LEDs von Seekway verwendet. Sie verfügen zwar ursprünglich nur über einen Öffnungswinkel von 25°, wurden jedoch dahingehend bearbeitet, dass die Linse um etwa 50 Prozent verkürzt wurde, und so das Licht von dem innenliegenden LED Plättchen direkt gesehen werden kann. Die Leuchtfläche ähnelt somit der einer SMD LED.

- Betriebsspannung ROT: 1,9V
- Betriebsspannung GRÜN: 2,3V
- Betriebsspannung BLAU: 3,4V
- Betriebsstrom: 20mA je Farbe
- Wellenlänge: 640nm (Rot), 515nm (Grün), 465nm (Blau)

Wie aus den technischen Daten zu erkennen ist, werden unterschiedliche Betriebsspannungen für die einzelnen Farben bei gleichem Storm benötigt. Um dieses Problem zu umgehen, werden über die Farb-Treiber (MOSFETs) unterschiedliche Frequenzen moduliert, damit jede Farbe die gleiche Helligkeit erreichen kann. Es ist auch zu beachten, dass im Design der Hauptplatine auf eine ausreichende Stromversorgung geachtet wird, da bei einer Vollauslastung der Stromverbrauch erheblich ansteigt.

Folgendes Rechenbeispiel veranschaulicht den theoretischen Gesamtenergiebedarf eines LED Cubes mit 512 LEDs, welche alle mit voller Helligkeit in der Farbe weiß leuchten:

$$0,02 * 1,9 * 512 + 0,02 * 2,3 * 512 + 0,02 * 3,4 * 512 \; = \; 77,824\, Watt$$

Das Beispiel ist nicht hundertprozentig auf einen PWM gesteuerten LED Cube übertragbar, da niemals alle LEDs mit allen drei Grundfarben gleichzeitig leuchten. Es ist jedoch ein guter Anhaltspunkt, da der Betriebsstrom einer Grundfarbe proportional zum Tastverhältnis angehoben werden sollte, um die maximale Helligkeit zu erzeugen.
Es gilt: Bei einer Betriebsspannung von 5 Volt müssen die Leiterbahnen für Ströme von rund 16 Ampere ausgelegt sein, da bei dieser Rechnung die Schaltverluste der LED Treiber und Leistungstransistoren nicht mit eingerechnet wurde. Ein Teil der geforderten Energie wird immer in Wärme umgesetzt.

Um sicherzugehen, dass die Leiterbahnen diesen Stromstärken standhalten, muss ermittelt werden, welches die minimale Breite für eine (im Normalfall) 35μm dicke Leitung auf einer Leiterplatte erforderlich ist.

Schicht-stärke *Thickness copper layer*	Leiterbahn-breite *Track-width*	Max. Strom in Abhängigkeit zur Temperaturerhöhung. *Max. current depending on temperarature increase.*				
		10 °C	20 ° C	30 °C	45 °C	60 ° C
35 µm	0,25 mm	0,5 A	0,8 A	1,0 A	1,3 A	1,6 A
	0,50 mm	1,0 A	1,6 A	2,0 A	2,5 A	3,0 A
	1,00 mm	2,2 A	3,0 A	3,6 A	4,2 A	4,8 A
	1,50 mm	3,0 A	3,8 A	4,6 A	5,3 A	6,5 A
	2,00 mm	3,8 A	5,0 A	6,5 A	7,5 A	8,5 A
	3,00 mm	4,5 A	6,5 A	8,0 A	9,5 A	11,0 A
	4,00 mm	6,0 A	8,5 A	10,0 A	12,0 A	13,5 A
	5,00 mm	7,0 A	10,0 A	12,0 A	14,5 A	16,0 A
	6,00 mm	7,5 A	11,0 A	14,0 A	16,0 A	18,0 A
	8,00 mm	9,0 A	14,0 A	17,0 A	20,0 A	22,5 A
	10,00 mm	10,0 A	16,0 A	20,0 A	23,0 A	26,0 A
70 µm	0,25 mm	1,0 A	1,6 A	2,0 A	2,5 A	3,0 A
	0,50 mm	2,0 A	2,8 A	3,5 A	4,0 A	4,5 A
	1,00 mm	3,5 A	4,7 A	5,8 A	6,8 A	8,0 A
	1,50 mm	4,5 A	6,2 A	7,5 A	9,0 A	10,5 A
	2,00 mm	6,0 A	8,5 A	10,0 A	12,0 A	13,5 A
	3,00 mm	7,5 A	11,0 A	14,0 A	16,0 A	18,0 A
	4,00 mm	9,0 A	13,5 A	17,0 A	19,0 A	22,0 A
	5,00 mm	10,0 A	15,0 A	19,0 A	23,0 A	25,0 A
	6,00 mm	11,0 A	18,0 A	22,0 A	26,0 A	28,0 A

Tabelle 3.2: Strombelastbarkeit von Kupfer (CU)-Leiterbahnen auf Basismaterial

Wie man anhand Tabelle 3.2 erkennen kann, wird eine Leiterbahn mit einer Breite von mindestens 10mm benötigt. Und selbst dann findet bei Volllast eine nicht unerhebliche Erwärmung der Hauptplatine statt. Es erscheint sinnvoll, entweder eine 70μm Kupferauflage zu wählen, oder beim Design der Platine auf möglichst große Kupferflächen zur Wärmeableitung zu achten.

4 Realisierung

Es gibt bisher keine vergleichbare Lösung, die diesem Projekt nahekommt. Daher ist es unumgänglich, sowohl einen Prototypen zu Testzwecken und Fehlerbeseitigung herzustellen, als dann auch die eigentliche Hauptplatine, welche für den Cube eingesetzt wird. Der Aufbau soll, wie bei dem Cube von Seekway, eine reine Leiterplattenlösung sein, welche komplett auf ein Drahtgittermodell verzichtet. So ist es möglich, den Cube modular aufzubauen und widerstandsfähiger gegen äußere Einflüsse zu machen. Außerdem ermöglicht dies eine sauberere Verarbeitung und den einfacheren Austausch, im Falle von defekten Komponenten. Die Reihenfolge der Hardwareentwicklung richtet sich nach der Komplexität der Einzelkomponenten.

4.1 LED Modul (Steckkarte)

Die acht LED Module beinhalten jeweils 64 einzelne RGB LEDs, vier Konstantstrom LED Treiber und zwei Dual-Channel MOSFETs. Da die Qualität der späteren Lichteffekte stark von dem vorgeschalteten Controller und dessen Programmierung abhängt, ist der Aufbau eines LED Moduls vergleichsweise simpel. Durch die Vorgaben, die aus dem Datenblatt der LED Treiber zu entnehmen sind, bleibt nicht viel Freiraum bei der Entwicklung der Schaltung.

4.1.1 Schaltplan eines LED Moduls

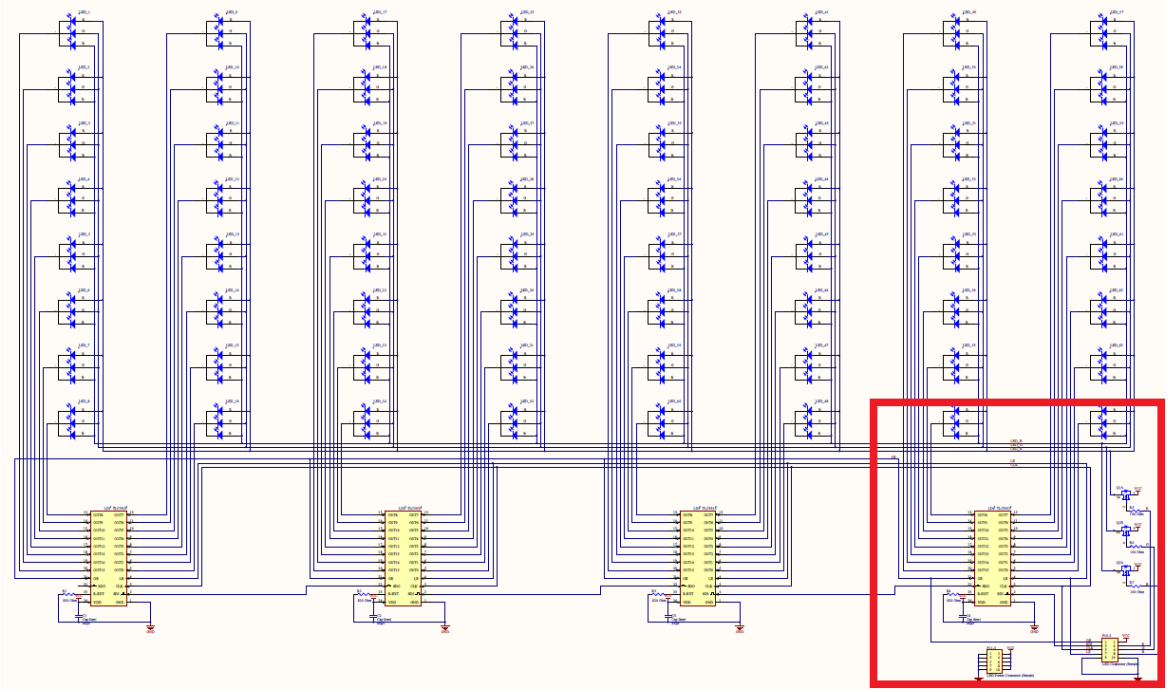

Abbildung 4.1: Schaltplan eines LED Moduls

4.1.2 Erläuterungen zum Schaltplan des LED Moduls

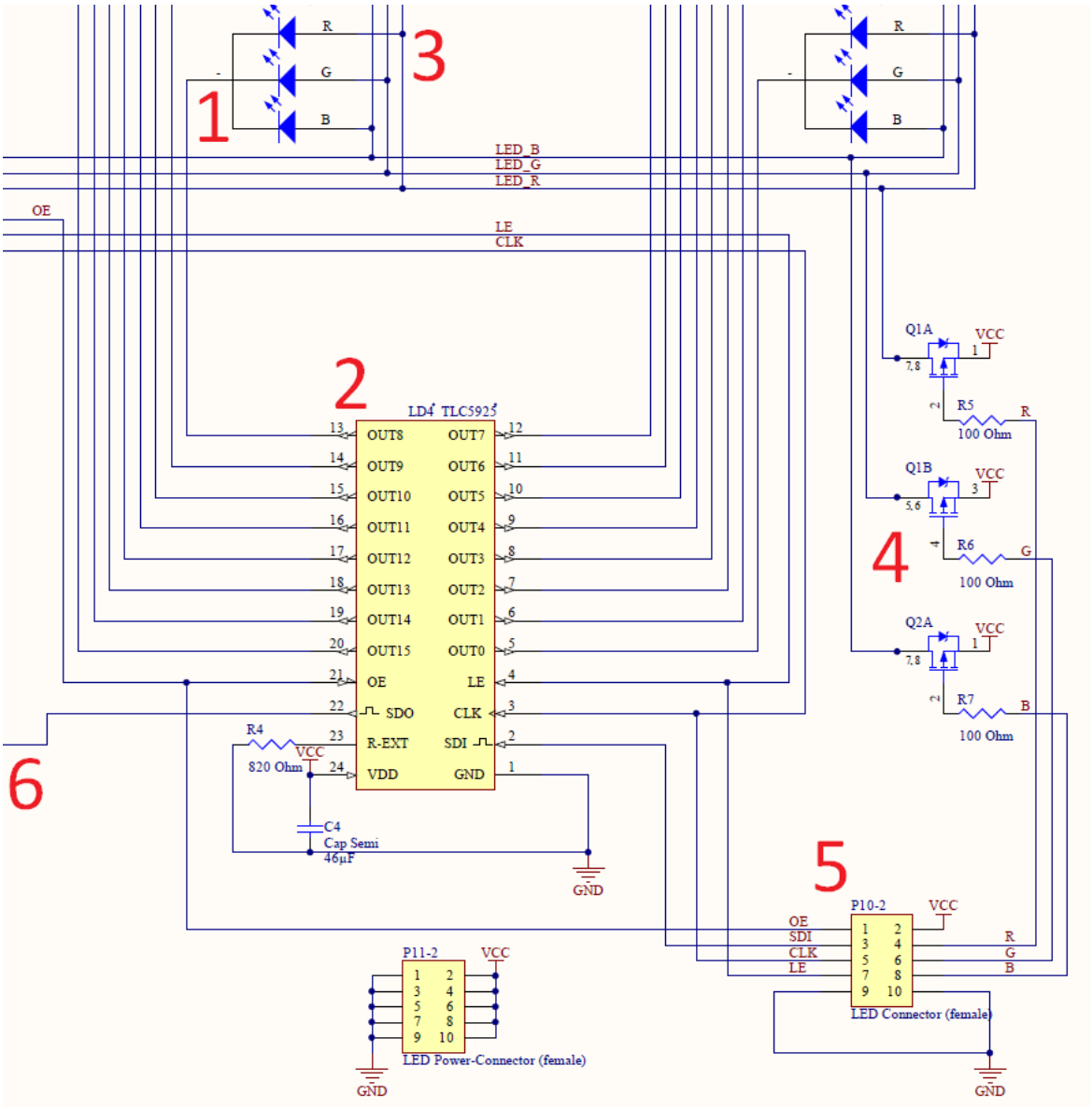

Abbildung 4.2: Vergrößerung aus Abb. 4.1: Detailansicht eines LED Moduls

Die Abbildung 4.2 zeigt die Funktionsweise von einem der acht LED Module. Die Kathode (1) der LED wird über einen der Ausgänge des LED Treibers (2) beim Schaltvorgang auf Masse gezogen, während die Farbe über die entsprechende Anode (3) von dem zugehörigen MOSFET (4) mit der Versorgungsspannung getrieben wird. Sowohl die Steuerimpulse für die MOSFETs, als auch die seriellen Daten für die LED Treiber, bezieht das LED Modul über einen einzelnen 10-Pin Steckverbinder (5). Serielle Daten, die bereits einmal das Schieberegister des LED Treibers passiert haben, werden über den SDO Ausgang (6) an

den nächsten LED Treiber weitergereicht. Hierbei müssen die Daten nicht vom ersten LED Treiber verwendet werden, sondern können diesen auch durchfließen, so dass der typische Aufbau einer sogenannten „Daisy Chain“ entsteht.
Es ist wichtig zu beachten, dass bei diesem Aufbau, aufgrund des Multiplexingverfahrens, zur Laufzeit jede aktive LED die selbe Farbe hat, da es nur drei Farb-Leitungen für ein gesamtes LED Modul gibt.

4.2 Testplatine für 64 LEDs

4.2.1 Vorüberlegungen

Die Teststeuerung für nur eine einzelne LED-„Karte“ müsste lediglich den Hauptprozessor AT90USB1287, den 8-Kanal Signalverstärker 73HC573 und einen USB Anschluss beinhalten, doch zur Programmierung und zu Testzwecken muss die Platine noch weitere Bauteile und Schnittstellen bereitstellen:

- Ausführung von 7 weiteren IO Ports zum Test der Latch-Enable Leitungen für die Signalverstärker.
- Reset-Taster zum Rücksetzen des Programms.
- HWB-Taster um den Prozessor in den Programmierzustand über USB zu setzen.
- Per Jumper trennbare Stromversorgung inkl. Buchse für 5V Netzteil.
- JTAG Interface zum Programmieren und Debuggen.
- ISP Interface zum schnellen Programmieren und für Software-Updates.
- Serielle Schnittstelle inkl. RS232 Transceiver zum Debuggen, wenn die Programmierung über USB erfolgt.
- Feinsicherung gegen Überlast am USB Anschluss.

4.2.2 Schaltplan der Testplatine

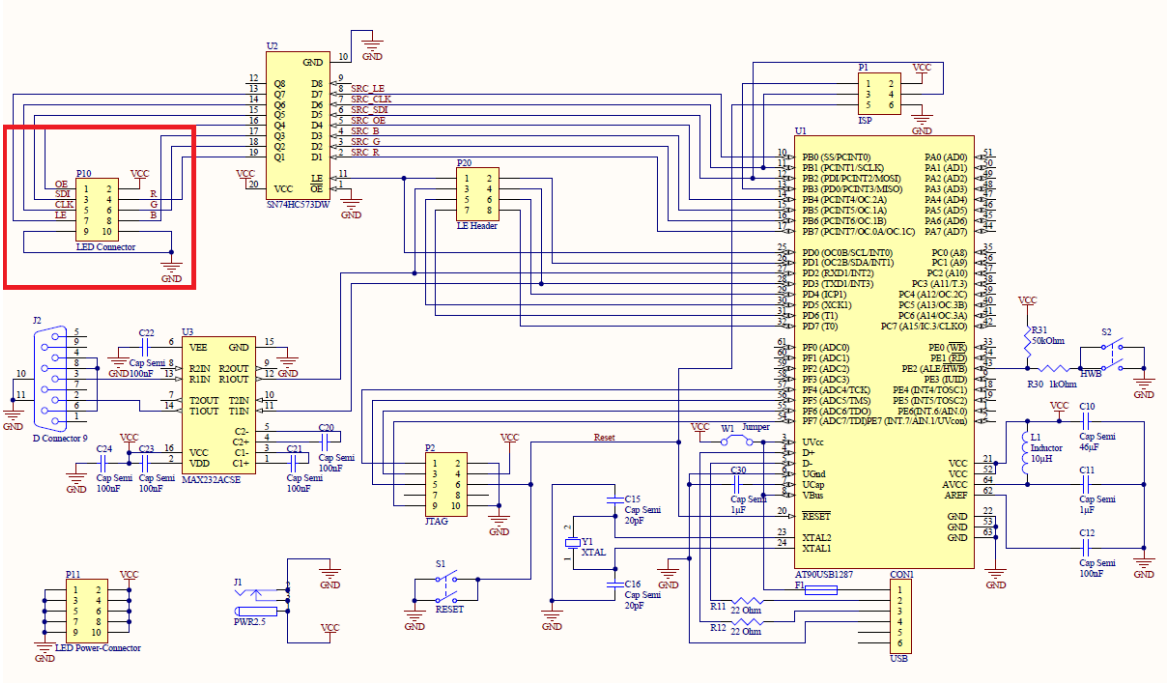

Abbildung 4.3: Schaltplan der Testplatine für 64 LEDs

Wie zu erkennen ist, wurden im Schaltplan 4.3 die LED Treiber und die MOSFETs zur Farbsteuerung weggelassen. Der Grund hierfür ist, dass diese Bauteile nicht direkt zu dem Controller gehören, sondern auf den jeweiligen LED-Modulen zum Einsatz kommen. Als Schnittstelle dient hierzu der LED Connector (P10).

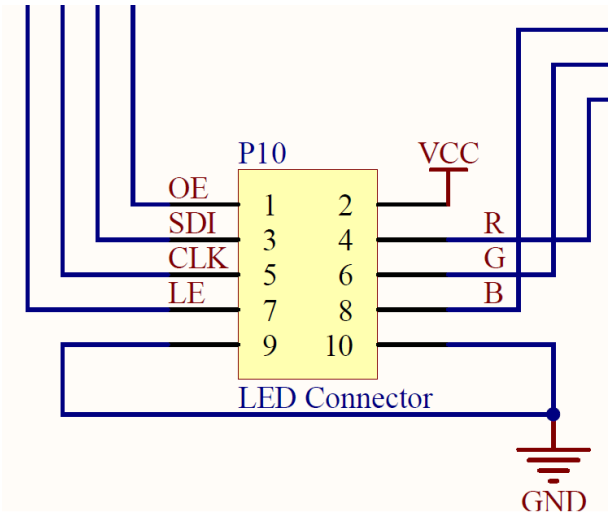

Abbildung 4.4: Vergrößerung aus Abb. 4.3: LED Connector P10 mit Pinbelegung

In der Abbildung 4.4 ist die Pinbelegung des Steckverbinders zu sehen, welche sowohl die seriellen Daten über die jeweiligen Schaltzustände der LEDs über SDI (Serial Data In) übermittelt, als auch analog dazu die zeitlich aufeinander abgestimmten Steuerimpulse für die Farbtreiber (R, G und B). Das Latch-Enable (LE) dient hierbei zur Aktivierung des Zwischenspeichers um den Zustand der gesetzten LED zu speichern. Befinden sich 0 Volt an dem LE Pin, so können neue Daten seriell über SDI eingelesen werden. Wird LE auf 1 (in diesem Fall 5 Volt) gebracht, so werden die Latches aktiviert und die neuen Daten parallel an den Ausgängen übernommen. Die Ausgänge der LED Treiber sind nur aktiv, wenn diese auch über Output-Enable (OE) aktiviert wurden.

Der nächste Schritt nach Erstellung des Schaltplanes ist das Layout und das Routing. Während im Layout die Bauteile an die richtigen Stellen platziert werden, stellt das Routing physikalische Verbindungen zwischen den Pins der Komponenten anhand des Schaltplanes her. Beide Vorgänge sollten aufeinander abgestimmt sein, so dass Bauteile nicht zu dicht nebeneinander liegen, was das nachfolgende Routing verhindern könnte. Außerdem ist es sinnvoll, die Bauteile schon vorher so auszurichten, dass die Länge der späteren Leiterbahnen möglichst kurz gehalten wird. Dies kommt sowohl den Signallaufzeiten, als auch dem Leitungswiderstand zugute und sollte bei jeder Platinenherstellung beachtet werden.

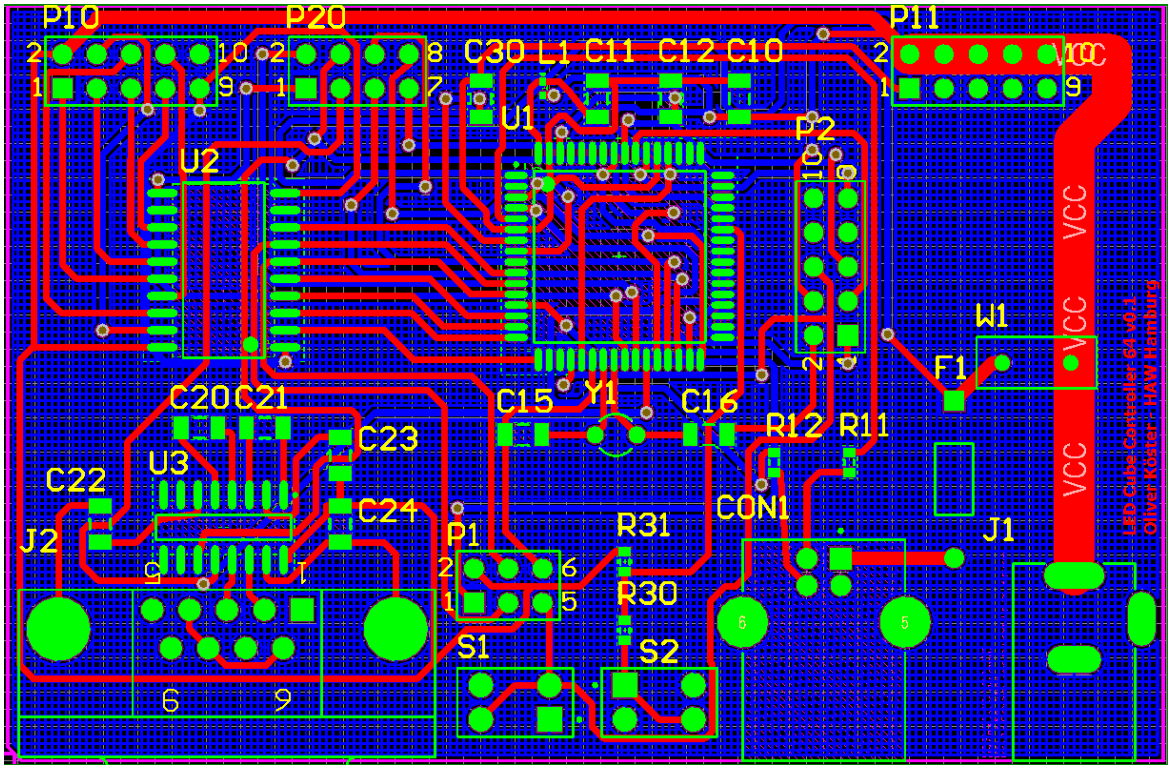

Abbildung 4.5: Layout der Testplatine

Bei dem LED Controller in Abb. 4.5 ist zu sehen, dass die Ausrichtung des AT90USB Mikrocontrollers (U1) in der Mitte so gewählt wurde, dass die Ausgangspins dicht an dem 74HC573 (U2) liegen. Auch der MAX232 Pegelwandler (U3) befindet sich so dicht wie möglich an der Buchse J2. Eine weitere Auffälligkeit stellen die VCC und GND Netze dar. Für VCC wurde ein vergleichsweise hoher Querschnitt bzw. Leiterbahnbreite von 3 Millimetern gewählt um den Leitungswiderstand zu minimieren. Gleiches gilt für das GND (Ground/Masse) Netz, welches sich über die gesamte Rückseite der Platine erstreckt. Dies hilft sowohl den Leitungswiderstand zu minimieren, als auch die Bauteile gegen äußere elektromagnetische Einflüsse abzuschirmen.

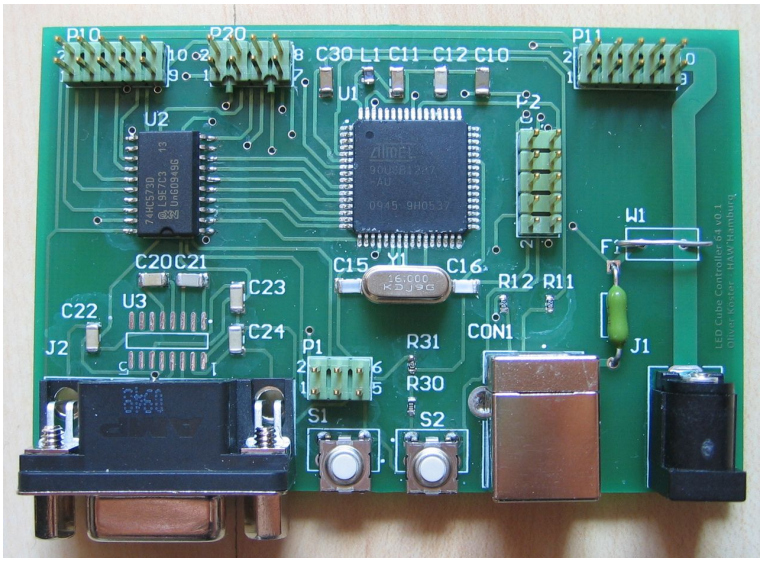

Abbildung 4.6: Die bestückte Testplatine - Hier noch ohne MAX232 Wandler

4.3 Implementierung der USB Schnittstelle

4.3.1 Grundlagen auf Hardwareebene

In der Tabelle 4.1 findet sich die Erklärung zur Beschaltung des USB Ports an einem Mikrocontroller der AT90USB Serie. Die wichtigsten Leitungen sind **UVcc**, **D+** und **D-** um die allgemeine USB Funktionalität herzustellen. Der Pegel der beiden Signalleitungen beträgt im Regelfall 3,3 Volt. 0 bis 0,8 V repräsentieren bei USB einen Low-Pegel, 2,0 bis 3,6 V einen High-Pegel. Eine Wandlung von der Versorgungsspannung ist also unumgänglich. Hierzu könnten theoretisch Spannungsregler vom Typ TPS71533 eingesetzt werden, oder - falls vorhanden - der interne Regler des AVR. Da letzteres der Fall ist, wird auch dieser sogenannte Pad-Regulator verwendet.

Pin	Bezeichnung	Funktion
2	UVCON	Dient zum Ein- oder Ausschalten der Versorgungsspannung für externe USB Geräte (nur im OTG Modus!)
3	UVcc	Versorgungsspannung des Pad-Regulators. Dieser interne Spannungsregler erzeugt die für die USB-Kommunikation nötigen 3,3 Volt Pegel.
4	D-	Negative USB Signalleitung zur Datenübertragung.
5	D+	Positive USB Signalleitung zur Datenübertragung.
6	UGND	Ground/Masse (negativ)
7	UCap	Pin zur Spannungsstabilisierung des Data-Pad-Regulators. An diesen Pin muss ein 1μF Kondensator gegen Masse angeschlossen werden, um die 3,3 Volt Pegel der D+ und D- Leitungen zu filtern und zu stabilisieren. Andernfalls wäre keine stabile Datenübertragung möglich.
8	VBus	Überwachung der USB-Versorgungsspannung. Wird im Normalfall direkt mit VCC (nicht UVcc!) verbunden.
9	UID	Pin zur Umschaltung zwischen dem USB-Host und OTG Modus. Wird der Pin gegen Masse gezogen, aktiviert der AVR den OTG Modus und arbeitet selbst als Host. In diesem Fall wird UID jedoch nicht beschaltet, da der LED Cube für diese Arbeit als Device arbeitet und zudem die OTG Funktionalität auch per Software initialisiert werden kann.

Tabelle 4.1: Pinbelegung des USB Ports am AT90USB

4.3.2 Grundlagen auf Softwareebene

Der Controller soll einen Datenstrom für die LED Module bereitstellen. Dies geschieht durch schnelle, zyklische Wiederholungen des selben Bitmusters um eine Bildwiederholfrequenz zu erreichen, die für das menschliche Auge wie ein kontinuierliches Leuchten der LEDs aussieht. Ein einzelnes Bild, welches aus 64 Farbwerten für jedes LED Modul besteht, kann im Hauptspeicher des AVR gepuffert werden. Die Veränderung der einzelnen Bilder soll aber nahezu in Echtzeit von einem externen Host bereitgestellt werden, damit der Cube den geforderten Einsatz als 3D-Display erfüllt. Diese Verbindung von einem Host zum Cube wird über den Universal Serial Bus (USB) hergestellt, welcher ohne zusätzliche Hardware direkt über den AT90USB1287 realisiert wird. Einschränkend ist jedoch zu beachten, dass der AT90USB1287 den High-Speed Modus nicht unterstützt, weshalb nur eine Übertragungsrate von theoretisch maximal 12 MBits/s erzielt werden kann. Aufgrund der vergleichsweise langsamen Arbeitsgeschwindigkeit des AVR sind aber auch 12 MBit/s kaum zu erreichen, da diese Datenmengen mit einem Referenztakt von 16 Mhz nicht verarbeitet werden können. 1,5 bis 3 MBit/s sind realistischer.

Das Projekt des LED Cubes wird hierbei als Abstract Control Model (ACM) Subklasse innerhalb der Communications Device Class (CDC) implementiert, um eine plattformunabhängige und unkomplizierte Treiberentwicklung zu gewährleisten. Für dieses Protokoll sind zwei Transfer Interfaces nötig. Eines, welches zwei Endpoints (Bulk-Data-IN und Bulk-Data-OUT) beinhaltet, und eines, welches aus einem Interrupt-IN Endpoint besteht, welcher zum Konfigurieren der Stittstelle nötig ist. Es wäre auch möglich, auf einen Endpoint für Konfigurationsfragen zu verzichten und sich auf die beiden Bulk-Transfer-Endpoints zu beschränken, doch dies würde zum Einen zu Problemen bei der Konfiguration führen, beispielsweise bei der Änderung der Baudrate der virtuellen Schnittstelle, und zum anderen nicht den CDC/ACM Spezifikationen entsprechen. Die Einhaltung der USB Spezifikationen ist nicht unbedingt notwendig, doch ein proprietäres USB-Endgerät würde auch immer einen proprietären Treiber voraussetzen, welche für jedes Betriebssystem angepasst werden muss. CDC/ACM wird von Linux und MacOS direkt unterstützt und kann unter Windows mit nur sehr geringem Aufwand verwendet werden. Die automatische Namenszuweisung des virtuellen COM-Ports funktioniert unter allen Betriebssystemen überwiegend problemlos und selbst Baudraten, welche außerhalb der Spezifikationen liegen, werden unterstützt. Es ist also möglich, beinahe die volle USB Geschwindigkeit über ein virtuelles serielles Interface auszunutzen ohne ein neues proprietäres Protokoll entwickeln zu müssen.

Als Basis für die USB Schnittstelle dient das OpenSource Framework „LUFA"[1] von Dean Camera, welches über eine große Anzahl von Bibliotheken für die Ansteuerung der internen USB Schnittstelle eines AVR Mikrocontrollers verfügt.
Durch die Einbindung der passenden Bibliotheks-Header kann so auch das CDC Proto-

[1] Lightweight USB Framework for AVRs

koll implementiert und auf die Endpoints direkt zugegriffen werden. Die Konfiguration dieser Endpoints wird dann direkt im Hauptprogramm vorgenommen. Bei der AT90USB Serie von ATMEL lassen sich bis zu sieben Endpoints einrichten, die sich 832 Bytes internes DPRAM teilen.
Es ist hierbei zu beachten, dass die Endpoints mit der Nummer 0 (EP0) auch die Klassenspezifischen Control-Requests beinhalten, welche in den CDC Spezifikationen definiert sind. Die EP0 müssen bei jedem Standard-USB-Gerät vorhanden sein, damit bei einer Verbindung der Host sofort eine Message-Pipe zur Kommunikation erstellen kann. Dadurch ist ist die Konfiguration der sog. Default Control Pipe beendet und der Host kann das Endgerät identifizieren und die Konfiguration des von diesem vornehmen.

4.3.3 Beispiel zur USB Implementierung

Ist LUFA als Grundbaustein für die USB Funktionalität implementiert und die Endpunkte konfiguriert, so muss lediglich die CDC Klasse eingebunden und konfiguriert (siehe Listing 4.1 und 4.2). werden. Das USB Gerät kann sich so wie eine serielle Schnittstelle verhalten. Der Datenstrom wird im Code als Datei (Typ FILE) behandelt um den Zugriff auf die empfangenen Daten zu erleichtern.

Listing 4.1: Eingebundene Header-Dateien

```
...
#include <LUFA/Version.h>
#include <LUFA/Drivers/USB/USB.h>
#include <LUFA/Drivers/USB/Class/CDC.h>
...
```

Listing 4.2: CDC Initialisierung

```
...
USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface =
{
  .Config =
  {
    .ControlInterfaceNumber         = 0,

    .DataINEndpointNumber           = CDC_TX_EPNUM,
    .DataINEndpointSize             = CDC_TXRX_EPSIZE,
    .DataINEndpointDoubleBank       = false,

    .DataOUTEndpointNumber          = CDC_RX_EPNUM,
    .DataOUTEndpointSize            = CDC_TXRX_EPSIZE,
    .DataOUTEndpointDoubleBank      = false,

    .NotificationEndpointNumber     = CDC_NOTIFICATION_EPNUM,
    .NotificationEndpointSize       = CDC_NOTIFICATION_EPSIZE,
    .NotificationEndpointDoubleBank = false,
  }
}
```

```
21 ...
22 static FILE USBSerialStream;
23 CDC_Device_CreateBlockingStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
24 ...
```

4.3.4 Test der USB-Verbindung

Nach der Implementierung der USB Schnittstelle als Communications Device Class, wird im Normalfall der Mikrocontroller als eigenständiges Gerät direkt vom Betriebssystem erkannt, und der interne USB Treiber geladen. Dies ist jedoch - gerade unter Windows - nur möglich, wenn sich das Endgerät strikt an die USB Spezifikationen hält und zudem über eine USB Vendor ID (VID) und eine Product ID (PID) verfügt, welche vom USB Implementers Forum (www.usb.org) vergeben wurde.

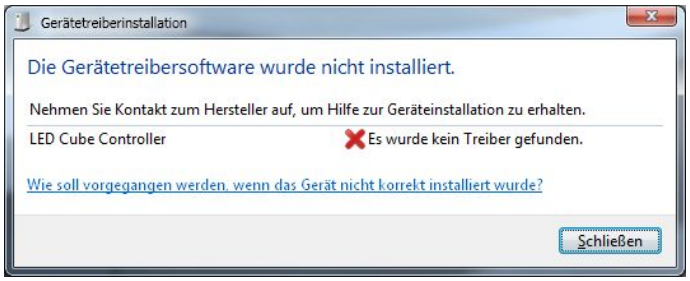

Abbildung 4.7: Windows erkennt das Gerät noch nicht als CDC Device

Ein einfaches Verbinden des Endgerätes ist also auf Windows-Betriebssystemen in Abbilung 4.7 nicht ohne Weiteres möglich, da kein passender Treiber gefunden wurde. Tatsächlich jedoch, existiert schon ein Treiber namens usbser.sys welcher alle Voraussetzungen erfüllt. In Winows 2000/XP ist dieser im Verzeichnis *C:\Windows\system32\drivers* vorhanden und bei Windows Vista/7 im Treiberarchiv unter *C:\Windows\System32\DriverStore\FileRepository*. Einzig die Zuordnung zwischen dem Gerät und dem internen Treiber ist nicht vorhanden.
Abhilfe schafft das Erstellen einer *.inf Datei, welche der Geräteklasse den Treiber zuweist und im jeweiligen Windows-Betriebssystem installiert. In dieser Datei wird dem USB Gerät eine VID und eine PID zugewiesen, welche von ATMEL für private Zwecke freigegeben wurden. Das bedeutet zwar, dass es zu Konflikten kommen kann, wenn mehrere ATMEL-basierte USB Geräte mit einem Host verbunden sind, doch da das LED Cube Projekt nicht für kommerzielle Zwecke bestimmt ist, kann man diesen Aspekt vernachlässigen.

Listing 4.3: Eine Beispielkonfiguration für Windows 2000/XP/Vista (32 Bit)

```
;*************************************************************
;  Klassenzuweisung
;
;  Class und ClassGuid sind von Microsoft fuer die COM und
;  LPT Ports vorgegeben und duerfen nicht veraendert werden.
;*************************************************************
[Version]
Signature="$Windows NT$"
Class=Ports
ClassGuid={4D36E978-E325-11CE-BFC1-08002BE10318}
Provider=%MFGNAME%
LayoutFile=layout.inf
CatalogFile=%MFGFILENAME%.cat
DriverVer=05/20/2010,0.0.0.2

[Manufacturer]
%MFGNAME%=DeviceList, NTamd64

;*************************************************************
;  Zielverzeichnis fuer den Treiber
;
;  11 = \system32
;  12 = \Drivers
;*************************************************************
[DestinationDirs]
DefaultDestDir=12

[DriverInstall.nt]
include=mdmcpq.inf
CopyFiles=DriverCopyFiles.nt
AddReg=DriverInstall.nt.AddReg


;*************************************************************
;  Liste der Dateien, welche in das Zielverzeichnis kopiert
;  werden muessen, wenn diese nicht vorhanden sind.
;*************************************************************
[DriverCopyFiles.nt]
usbser.sys,,,0x20

;*************************************************************
;  Registrierung des Treibers. Ausserdem die automatische
;  Auswahl der virtuellen Portnummer.
;*************************************************************
[DriverInstall.nt.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,%DRIVERFILENAME%.sys
HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider"

[DriverInstall.nt.Services]
AddService=usbser, 0x00000002, DriverService.nt

[DriverService.nt]
DisplayName=%SERVICE%
ServiceType=1
StartType=3
ErrorControl=1
ServiceBinary=%12%\%DRIVERFILENAME%.sys

```

```
60 ;************************************************************
61 ;  Vendor und Product ID Definitionen
62 ;
63 ;  Es wird die von ATMEL freigegebene VID "03EB" und
64 ;  die PID "2044" verwendet.
65 ;************************************************************
66 [SourceDisksFiles]
67 [SourceDisksNames]
68 [DeviceList]
69 %DESCRIPTION%=DriverInstall, USB\VID_03EB&PID_2044
70
71 ;************************************************************
72 ;  String Definitionen
73 ;
74 ;  Dies ersetzt die o.A. Variablen und ist mit dem Befehl
75 ;  "define" in der Sprache C vergleichbar.
76 ;************************************************************
77 [Strings]
78 MFGFILENAME="CDC_vista"
79 DRIVERFILENAME ="usbser"
80 MFGNAME="Oliver Koester"
81 INSTDISK="LED Cube CDC Driver Disk"
82 DESCRIPTION="RGB LED Cube Serial Port"
83 SERVICE="USB RS-232 Emulation Driver"
```

Abbildung 4.8: Der LED Cube ist als Serielles Interface verfügbar

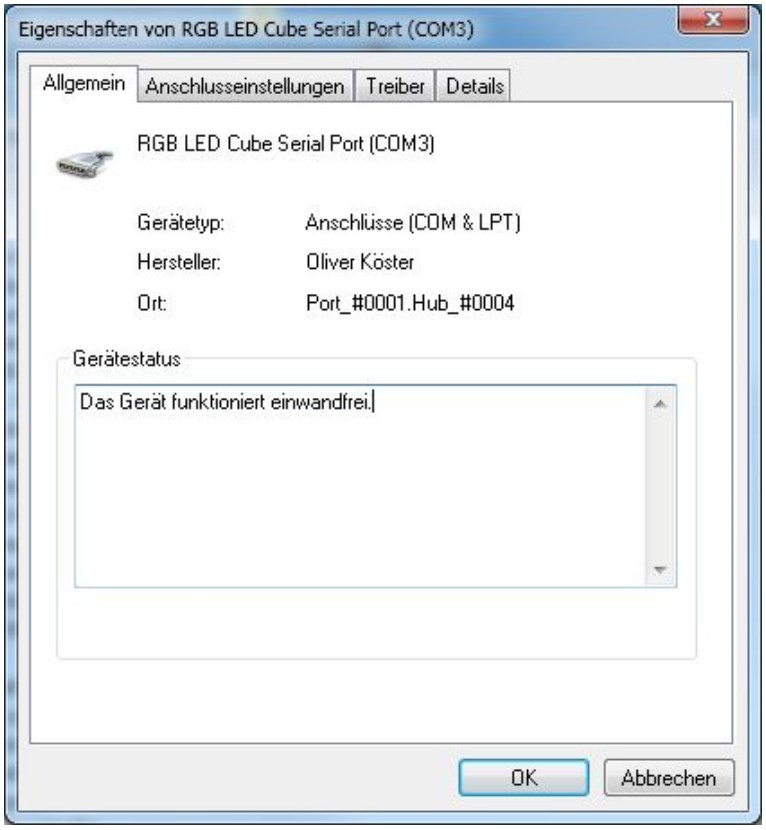

Abbildung 4.9: Geräteeigenschaften nach Treiberinstallation

Nach dem Erstellen der genannten *.inf-Datei anhand des Beispiels in dem Listing 4.3, wird automatisch der usbser.sys Treiber installiertund der Cube ist als virtuelles serielles Interface im Gerätemanager (Abb. 4.8) verfügbar. Von nun an ist der LED Cube voll plug-and-play fähig. Die Namensvergabe für den virtuellen COM-Port erfolgt automatisch, indem jeweils der nächste freie Port verwendet wird. In Abbildung 4.9 ist zu sehen, dass er die Bezeichnung COM3 bekommen hat. Dies hat den Grund, dass COM1 der echte Serielle Anschluß auf dem Mainboard des Host-PCs ist, und COM2 durch einen weiteren virtuellen Port belegt ist. Dieses USB Gerät ist jedoch zum Zeitpunkt des Tests nicht angeschlossen, weshalb es nicht im Gerätemanager auftaucht. Die Namensvergabe bleibt jedoch weiterhin erhalten, solange der Treiber für dieses Geräte nicht gelöscht wird.

4.3.5 Test der seriellen Datenübertragung über USB

Um die Datenübertragung über USB zu testen, wird der Controller über einen JTAG-ICE MKII verbunden. Dadurch kann der Inhalt von jeder beliebigen Variable zur Laufzeit in der Entwicklungsumgebung AVRStudio4 überwacht werden. Über ein Testprgramm werden zufällige Bitmuster in 4*8-Bit Blöcken gesendet und diese Daten in ein 32-Bit Array eingelesen.

Bei immer verschiedenen Bitmustern, welche beispielsweise aus 4 Hexadezimalzahlen besteht, gab es bei keinem Test Probleme, doch bei bei sich wiederholenden Mustern ergab sich das Bild aus Abbildung 4.10:

Gesendet wurde 3 mal hintereinander die Zeichenfolge **AAAA**

Watch

Name	Value	Type	Location
input_stream	[...]	uint8_t[4]	0x0125 [SRAM]
[0]	0x41 'A'	unsigned char	0x0125 [SRAM]
[1]	0x41 'A'	unsigned char	0x0126 [SRAM]
[2]	0xC1 'Á'	unsigned char	0x0127 [SRAM]
[3]	0x41 'A'	unsigned char	0x0128 [SRAM]

Abbildung 4.10: Überwachung der Variablenwerte in AVRStudio

In unregelmäßigen Abständen entstanden Übertragungsfehler in Form von „gekippten“ Bits. Diese Fehler häuften sich, bei der Verwendung von USB Kabeln mit einer Länge von über 1,5 Metern. Ein Fehler im Programm war durch Softwaretests ausgeschlossen. Eine Messung des Spannungsverlaufes am USB Anschluss ergab ein elektrisches Problem. Bei jeder Übertragung war ein kurzes Ausbrechen des Spannungsverlaufes auf der VBUS Leitung zu erkennen. Grund hierfür ist ein Übersprechen der Datenleitungen D+ und D- auf VBUS/UVCC und umgekehrt. Abhilfe schaffte ein 1 μF Kondensator, welcher direkt an der USB Buchse parallel zu VBUS und GND angeschlossen wurde. Dieser eliminierte alle Übertragungsfehler auch bei langen USB Kabeln.

4.4 PWM Ansteuerung

4.4.1 PWM Test mit einer einzelnen LED

Da sowohl die Erkennung als auch der Datentransfer über USB fehlerfrei funktioniert, ist es nun mit geringem Aufwand möglich, den Controller zur Laufzeit mit neuen Daten zu versorgen. Über einen einfachen Testaufbau auf einer Lochrasterplatine kann so die PWM Signalerzeugung mit einer einzelnen RGB LED getestet werden. Der Schaltplan in Abb. 4.11 zeigt, dass hier mehrere Widerstände zum Einsatz kommen. Zum einen die bekannten 100 Ohm Widerstände, welche als Vorwiderstand und Überlastschutz der MOSFETs dienen, und zum anderen drei Vorwiderstände für die RGB LED. Diese sind unbedingt nötig, um den Strom zu begrenzen, welcher beim Schaltvorgang durch die MOSFETs fließt. Dieser könnte sonst bei einer zu breiten Pulsdauer zu einer undefinierten Größe ansteigen und die LED zerstören. Im Falle der LED Module des Cubes sind diese Widerstände jedoch nicht nötig, da die Strombegrenzung über die LED Treiber geregelt wird.

Die Vorwiderstände errechnen sich wie folgt:

$$Vorwiderstand\ R_V = \frac{Betriebsspannung\ U_B\ -\ Flussspannung\ U_{LED}}{Strom\ I_{LED}}$$

Setzt man nun die bekannten Größen in die Gleichung ein, erhält man die Vorwiderstandswerte für jede einzelne Farbe:

$$Rot: \frac{5V - 1,9V}{0,02A} = 155\Omega$$

$$Gruen: \frac{5V - 2,3V}{0,02A} = 135\Omega$$

$$Blau: \frac{5V - 3,4V}{0,02A} = 80\Omega$$

Aufgrund der schlechten Verfügbarkeit von den o.g. Widerstandswerten, wurden jedoch in der Schaltung die nächsthöheren Widerstandswerte verwendet.

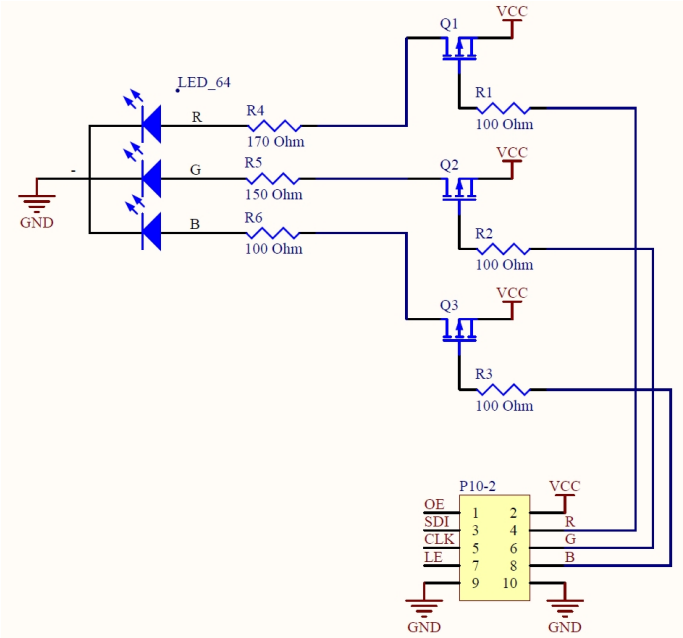

Abbildung 4.11: Schaltplan der Lochrasterplatine

Die Abbildung 4.13 zeigt die bestückte Lochrasterplatine mit Pinleiste, Vorwiderständen und P-Kanal MOSFETs. Bei der RGB LED handelt es sich um eine 6-Pin-LED vom Typ Kingbright LF5WAEMBGMBC.

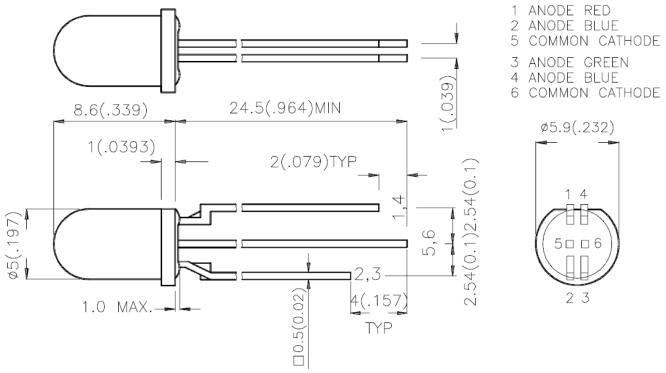

Abbildung 4.12: Abmessungen und Anschlussbelegung der Kingbright LF5WAEMBGMBC

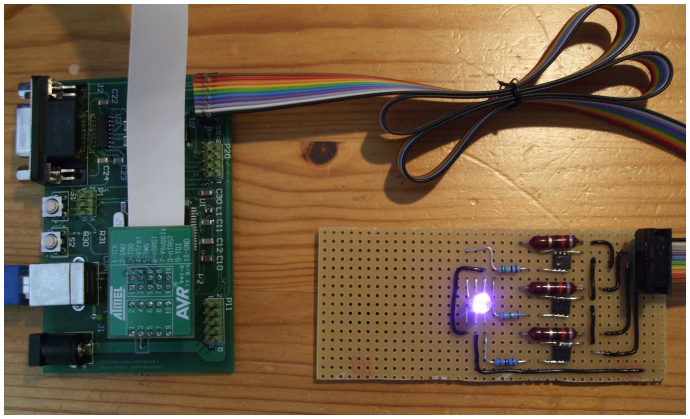

Abbildung 4.13: Test der PWM Ansteuerung für eine RGB LED über P-Kanal MOSFETS

Sowohl das Ansteuern als auch das Mischen der Farben innerhalb der LED funktioniert problemlos. Jede Farbe der LED kann über verschiedene Pulsweiten in der Helligkeit verändert werden und so nahezu jede beliebige Farbe dargestellt werden. Es gibt jedoch eine Auffälligkeit bezüglich der Abstufungen im Helligkeitsbereich:
Die Kennlinie des menschlichen Auges ist nicht linear, sondern nahezu logarithmisch. Das bedeutet, dass Unterschiede in der Helligkeit einer Lichtquelle im unteren Bereich viel stärker wahrgenommen werden, als im oberen- sehr hellen Bereich. Dies hat zur Folge, dass schon bei der kleinsten PWM Abstufung innerhalb der 4 Bit pro Farbe ein deutliches Leuchten zu erkennen ist, welches exponentiell zunimmt. Der Unterschied zwischen den oberen Stufen ist jedoch kaum wahrnehmbar. Bei der späteren Farbabmischung und der Entwicklung eines Kontrollprogramms für den fertigen Cube muss dies berücksichtigt werden.

4.4.2 Aufbau des Datenstroms vom Controller zum LED Modul - Versuch 1

Der Datenstrom jedes einzelnen LED Moduls besteht aus zwei Phasen, welche jeweils in 64 Takte eingeteilt sind. Abbildung 4.14 zeigt wie in der ersten Phase über SDI in Abhängigkeit von CLK eine einzelne LED gesetzt wird. Da alle Daten erst den letzten LED-Treiber durchfließen und über SDO weitergereicht werden, entspricht ein gesetztes Bit an dritter Stelle auch der dritten LED auf einem LED-Modul.

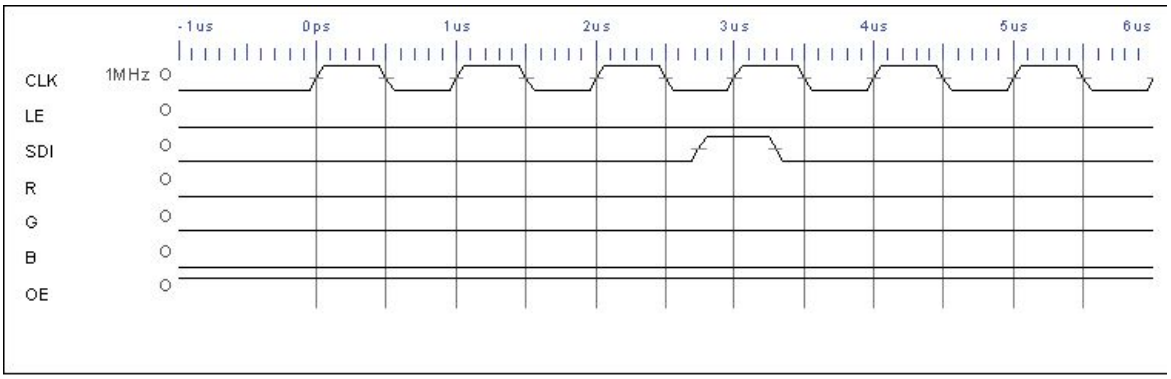

Abbildung 4.14: Bit für gewünschte LED setzen

Nach Ablauf von 64 Takten (0-63) wird das LE für einen Takt aktiviert, um die bis dahin gesendeten Daten in die Output-Latches der LED-Treiber zu übernehmen. Die Daten für die gewünschte LED liegen zu diesem Zeitpunkt bereit, werden aber noch nicht verwendet oder angezeigt.

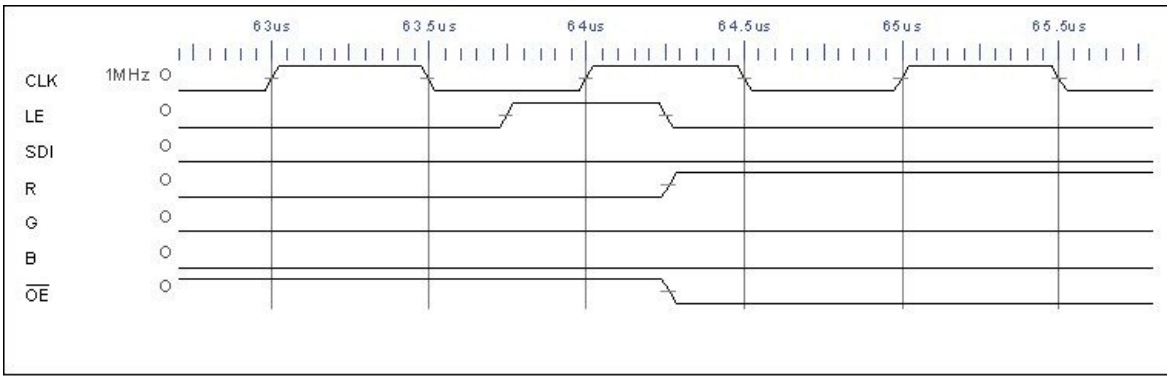

Abbildung 4.15: Farbe rot für LED zuweisen

Erst nachdem alle Daten geschrieben und zwischengespeichert wurden, werden die Ausgänge über OE aktiviert und gleichzeitig die Farbkodierung über die drei MOSFETs vorgenommen (siehe Abb. 4.15) . Dieser Vorgang findet in Phase 2, also wieder 64 Takten statt und beinhaltet lediglich eine Veränderung der Pulsweite für jede Grundfarbe. Die PWM Signale

liegen jedoch immer an, und werden über einen eigenständigen Timer im Mikrocontroller erzeugt. So wird gewährleistet, dass die jeweilige LED für die größtmögliche Zeit mit einem Farbsignal versorgt wird.

4.4.3 Codebeispiel für eine Ebenensteuerung

Nachfolgendes Beispiel (Listing 4.4) wurde nach den in Abschnitt 4.4.2 besprochenen Vorüberlegungen programmiert. Es veranschaulicht die Ansteuerung einer Ebene mit 64 LEDs. Um die größtmögliche Übersicht zu schaffen, wurden keine Optimierungen vorgenommen und alle Ports direkt an Ort und Stelle gesetzt. Eine derartige Umsetzung ist daher nicht praxistauglich. Die maximale Ausführungsgeschwindigkeit der ISR wäre viel zu langsam.

Listing 4.4: Codebeispiel der Ebenensteuerung

```
25 ISR(TIMER0_COMPA_vect, ISR_BLOCK)
26 {
27     PORTB |= 0x02; //CLK am Anfang hochsetzen...
28
29     if(++LEDSubCount == 64)
30     {
31       /*
32        * Werte fuer die Farbe der aktuellen LED
33        * an den PWM Generator fuer die MOSFETS uebergeben
34        */
35       SoftPWM_Channel_R_Duty = colbuffer[R][LEDCount];
36       SoftPWM_Channel_G_Duty = colbuffer[G][LEDCount];
37       SoftPWM_Channel_B_Duty = colbuffer[B][LEDCount];
38
39       LEDCount++;
40       LEDSubCount = 0;
41
42       /*
43        * PORTB Bit 0 = Latch Enable fuer alle
44        * LED Treiber auf einem Modul.
45        */
46       PORTB |= 0x01; //LE=1
47     }else{
48       PORTB &= 0xFE; //LE=0;
49     }
50
51     /*
52      * Bei steigender Flanke die gewaehlte LED (von 64)
53      * innerhalb eines Modules setzen.
54      * LEDSubCount ist hierbei die 'innere' Schleife
55      *
56      * PORTB Bit 1 = CLK
57      * PORTB Bit 2 = SDI
58      */
59     if(LEDSubCount == LEDCount)
60     {
61       PORTB |= 0x04; //SDI=1
62     }else{
63       PORTB &= 0xFB; //SDI=0
64     }
65
```

```
66      if(LEDCount == 64)
67      {
68        LEDCount = 0;
69        PORTB |= 0x10; //OE=0
70      }else{
71        PORTB &= 0xEF; //OE=1
72      }
73
74      /*
75       * PWM Generator fuer die Farbansteuerung
76       *
77       * PORTB Bit5 = blau
78       * PORTB Bit6 = gruen
79       * PORTB Bit7 = rot
80       *
81       */
82      if (++SoftPWM_Count == 0b00010000)
83          SoftPWM_Count = 0;
84
85      //ROT
86      if (SoftPWM_Count >= SoftPWM_Channel_R_Duty)
87      {
88        PORTB |= 0x80;
89      }else{
90        PORTB &= 0x7F;
91      }
92
93      //GRUEN
94      if (SoftPWM_Count >= SoftPWM_Channel_G_Duty)
95      {
96        PORTB |= 0x40;
97      }else{
98        PORTB &= 0xBF;
99      }
100
101     //BLAU
102     if (SoftPWM_Count >= SoftPWM_Channel_B_Duty)
103     {
104       PORTB |= 0x20;
105     }else{
106       PORTB &= 0xDF;
107     }
108   PORTB &= 0xFD; // CLK am ende runtersetzen
109   }
110 }
```

4.4.4 Testergebnis der Ebenenansteuerung

Die Versorgung einer Ebene mit Daten erfordert ein hohes Maß an Präzision bezüglich des Zusammenspiels der Bitmuster für die LED Treiber und den Schaltzeiten der MOSFETs. Bei diesem Test war es darum nötig, die Aktualisierungsfrequenz des angeschlossenen Moduls (Ebene) nicht zu hoch zu setzen, da ein Betrieb im zeitkritischen Bereich eine Divergenz zwischen den Datensignalen (CLK, SDI, OE) und den R, G und B Leitungen zur Folge hat. Werden alle Signale innerhalb einer Interrupt-Service-Routine gesetzt, so beträgt die

maximale Aktualisierungsrate rund 200 KHz. Da jedoch für eine Einzelne LED 64x64 Takte benötigt werden, ist trotzdem noch ein starkes Flimmern zu sehen, welches bei einer geringeren Helligkeit/Pulsweite einer Farbe noch stärker Auftritt und eher ein Blinken als ein Leuchten darstellt.

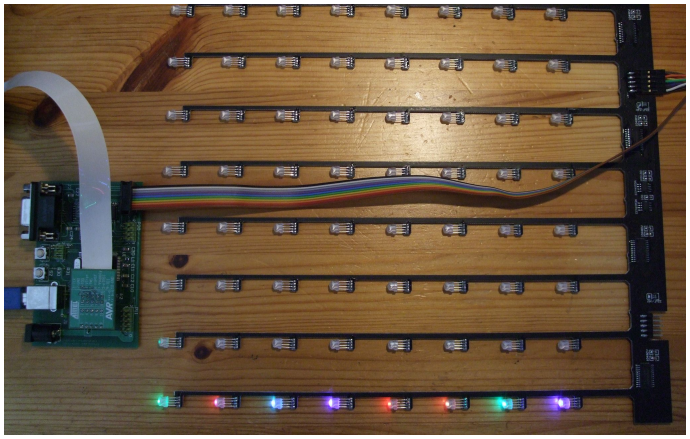

Abbildung 4.16: Der Controller-Prototyp im Zusammenspiel mit einem LED Modul

In der Abbildung 4.16 ist zu erkennen, dass die Farben der einzelnen LEDs eine Art „Schatten“ auf die nachfolgende LED wirft. Dieses Phänomen ist besonders gut an den beiden roten LEDs 3 und 4 (v.r.n.l) zu sehen. LED 3 weist hier einen leichten Grünanteil von LED 2 auf, obwohl beide mit dem selben Farbwert angesteuert wurden. Dieser Effekt tritt ab einer ISR-Frequenz von 260 Khz sichtbar auf. Bei 320 KHz ist er so dominant, dass eine konkrete Farbabmischung unmöglich ist.

4.4.5 Verbessertes Codebeispiel für die Ebenensteuerung

Zur besseren Übersicht wurden die Ausgänge der Ports durch Makros/Defines ersetzt.

Am Anfang wird eine 1 über SDI in die Schieberegisterkette geschoben und gleichzeitig der aktuelle Farbwert angelegt. Mit jedem Zyklus der ISR wird diese 1 weitergeschoben und jeweils immer die aktuelle Farbe geändert. Nach 63 Takten ist das Ende der Schieberegister erreicht und die 1 „fällt heraus“. Zeitgleich wird durch die if-Abfrage über den *LEDCount* eine neue 1 generiert. - All das hat zur Folge, dass dieser Code um den Faktor 64 schneller ist. Eine Ebene lässt sich so völlig problemlos und flimmerfrei über den einzelnen AT90USB1287 ansteuern. Auch eine zweite Ebene ist von der Geschwindigkeit her möglich, ohne dass ein Flimmern sichtbar wird.

Listing 4.5: Optimiertes Codebeispiel der Ebenensteuerung

```
1 ISR(TIMER0_COMPA_vect, ISR_BLOCK)
2 {
3   /*
4    * Output, Clock und Latch-Enable auf "high" setzen um neue
```

```
5     * Daten in die Schieberegister zu uebernehmen und die Ausgaenge
6     * abzuschalten.
7     */
8    PORTB |= ((1<<OE)|(1<<CLK)|(1<<LE));
9
10
11   if (++SoftPWM_Count == 0b00001111)
12     SoftPWM_Count = 0;
13
14
15   if (SoftPWM_Count < framebuffer[R][LEDCount])
16   {
17     PORTB &= ~(1<<RED);
18   }else{
19     PORTB |=  (1<<RED);
20   }
21
22
23   if (SoftPWM_Count < framebuffer[G][LEDCount])
24   {
25     PORTB &= ~(1<<GREEN);
26   }else{
27     PORTB |=  (1<<GREEN);
28   }
29
30
31   if (SoftPWM_Count < framebuffer[B][LEDCount])
32   {
33     PORTB &= ~(1<<BLUE);
34   }else{
35     PORTB |=  (1<<BLUE);
36   }
37
38   if(++LEDCount == 64)
39   {
40     LEDCount = 0;
41     PORTB |=  (1<<SDI); //SDI=1
42   }else{
43     PORTB &= ~(1<<SDI); //SDI=0
44   }
45
46   /*
47    * Output-Enable aktivieren, Latch-Enable und Clock wieder runtersetzen
48    * um die LED zu deaktivieren und eine fallende Clockflanke
49    * zu erzeugen.
50    */
51   PORTB &= ~((1<<OE)|(1<<CLK)|(1<<LE));
52 }
```

Leider bestehen weiterhin die Konvergenzprobleme im Codebeispiel 4.5, und durch die nun sehr hohe Geschwindigkeit tritt der Verwisch-Effekt noch stärker auf. Eine Minderung hat das Ein- und Abschalten der Ausgänge (OE) am Anfang und Ende der ISR bewirkt, jedoch noch nicht eleminiert. Auch die Helligkeit der LEDs ist immer noch eher gering, da das Tastverhältnis noch immer 1/64 beträgt. Die theoretische Möglichkeit dies durch eine Erhöhung des Stroms um den Faktor 64 zu beheben entfällt, da der maximale Ausgangsstrom der LED Treiber nur 45mA beträgt. Sollte eine oder mehrere Ebenen zusätzlich angesteuert werden,

verringert sich jeweils das Tastverhältnis weiter auf 1/128...1/192 usw. Ab diesem Punkt ist die Helligkeit keinesfalls mehr ausreichend.

4.5 Fazit der Entwicklung der Testplatine

Die Tests haben gezeigt, dass sowohl eine stabile Stromversorgung als auch die Eliminierung von Störsignalen eine entscheidende Rolle bei der Entwicklung von Platinenlayouts spielen. Auch wenn es bei einer vergleichsweise kleinen Schaltung unnötig erscheint, so ist die Glättung und Entstörung mittels Kondensatoren unbedingt erforderlich um einen zuverlässigen Betrieb zu gewährleisten. Ferner stellt bei dem ersten Testaufbau mit einem einzelnen Mikrocontroller, die CPU Auslastung bei der USB Datenübertragung einen entscheidenden zeitkritischen Faktor dar, wenn auch die PWM Signale durch denselben Controller realisiert werden sollen. 320 KHz entspricht der maximal möglichen ISR-Frequenz. Es ist hierbei jedoch nicht mehr möglich, Daten mit voller USB Geschwindigkeit an den Controller zu senden, da dieser das hohe Interrupt-Aufkommen nicht mehr bewältigen kann. Die Folge ist ein Verbindungsverlust zum Host-PC. Im zweiten Test mit optimierter ISR ist das Geschwindigkeitsproblem für eine Ebene zwar nahezu gelöst, jedoch nicht das der Helligkeit. Würde man bei mehreren Ebenen jedoch mehrere Einsen in die Schieberegister schieben und zudem jeweils zusätzlich eigene Farb-Leitungen zu den MOSFETs legen, könnte man mit nur einem Mikrocontroller zwei, maximal drei Ebenen zeitgleich im Tastverhältnis 1/64 ansteuern. Ein Cube benötigt allerdings 8 Ebenen. Für einen RGB LED Cube der geforderten Größe ist es also durchaus sinnvoll, die USB Funktionalität und die Signalgeneration zu trennen und dedizierte Controller hierfür zu verwenden um wenigstens das Geschwindigkeitsproblem zu lösen.
Das Problem welches jedoch noch nicht gelöst wurde, ist das Verschmieren der Farben. Obwohl die MOSFETs genau zeitgleich mit den Datenbits für die LEDs geschaltet werden, besteht eine Zeitverzögerung zwischen dem Anlegen der (negativen) Gate-Spannung, und dem Aktivieren- bzw. Deaktivieren des Ausgangs (Drain).

4.6 Entwicklung der Hauptplatine für 512 LEDs

4.6.1 Vorüberlegungen und Schaltplan

Wie im letzten Abschnitt festgestellt, benötigt der Prozessor auf der Hauptplatine des Cubes mehr Rechenleistung um sowohl 512 LEDs anzusteuern als auch den USB Datenstrom schnell genug verarbeiten zu können. Hierzu stehen zwei alternative Lösungen zur Verfügung:

Möglichkeit A: Der AT90USB1287 wird durch einen wesentlich schnelleren Mikrocontroller ersetzt. Bedenkt man, dass bereits für USB und 64 LEDs die aktuellen 16 Mhz kaum ausreichen, erscheint eine Verzehnfachung der Rechenleistung für 8 mal 64 LEDs sinnvoll. Die AT91SAM-Serie von ATMEL mit Taktfrequenzen von bis zu 500 Mhz wäre für dieses Vorhaben ideal, obwohl für den Betrieb ein zusätzlicher Festspannungsregler eingesetzt werden müsste, da die maximale Versorgungsspannung dieser Mikrocontroller 3,6 Volt beträgt. Ein praktisches Problem besteht jedoch darin, dass diese Controller aufgrund der hohen Anzahl von IO-Pins nur in Ball-Grid-Array (BGA) Gehäusen verfügbar sind. Ohne Reflow-Anlage lässt sich so ein Controller nicht auflöten.

Möglichkeit B: Jede Ebene, welche aus 64 LEDs besteht, wird über einen separaten 8-Bit Controller der ATmega16-Serie angesteuert. Die Versorgung mit Rohdaten und die USB Kommunikation wird über den AT90USB abgewickelt. Dieser übernimmt auch das Routing der neuen Bilddaten. So ist die Last auf insgesamt 9 Mikrocontroller verteilt und durch ein hohes Interrupt-Aufkommen durch den USB kommt es nicht zu Verzögerungen bei der Signalgenerierung. Der Nachteil bei dieser Lösung ist der relativ hohe Preis, welcher sich aus den Anschaffungskosten der Controller und die der zusätzlichen Bauelemente wie Kondensatoren und Schwingquarze errechnet. Dafür lassen sich Chips mit einem Thin-Quad-Flat-Pack (TQFP) relativ einfach mit einem handelsüblichen Lötkolben auflöten. Dadurch ist diese Lösung, welche im Übrigen an den Aufbau des RGB Cubes von Seekway (Siehe Abb. 2.19) erinnert, die einzig realisierbare. Alle weiteren Teile dieser Arbeit beziehen sich auf Möglichkeit B und werden auch so umgesetzt.

Der erste Schritt zur Realisierung ist, wie schon bei dem Prototypen, die Erstellung eines Schaltplanes (Abb. 4.17) mit nachfolgendem Layout (Abb. 4.21) der Platine. Die Abbildung 4.18 zeigt hierbei eine Detailansicht der Verbindung zwischen dem AT90USB und den einzelnen ATmega164.

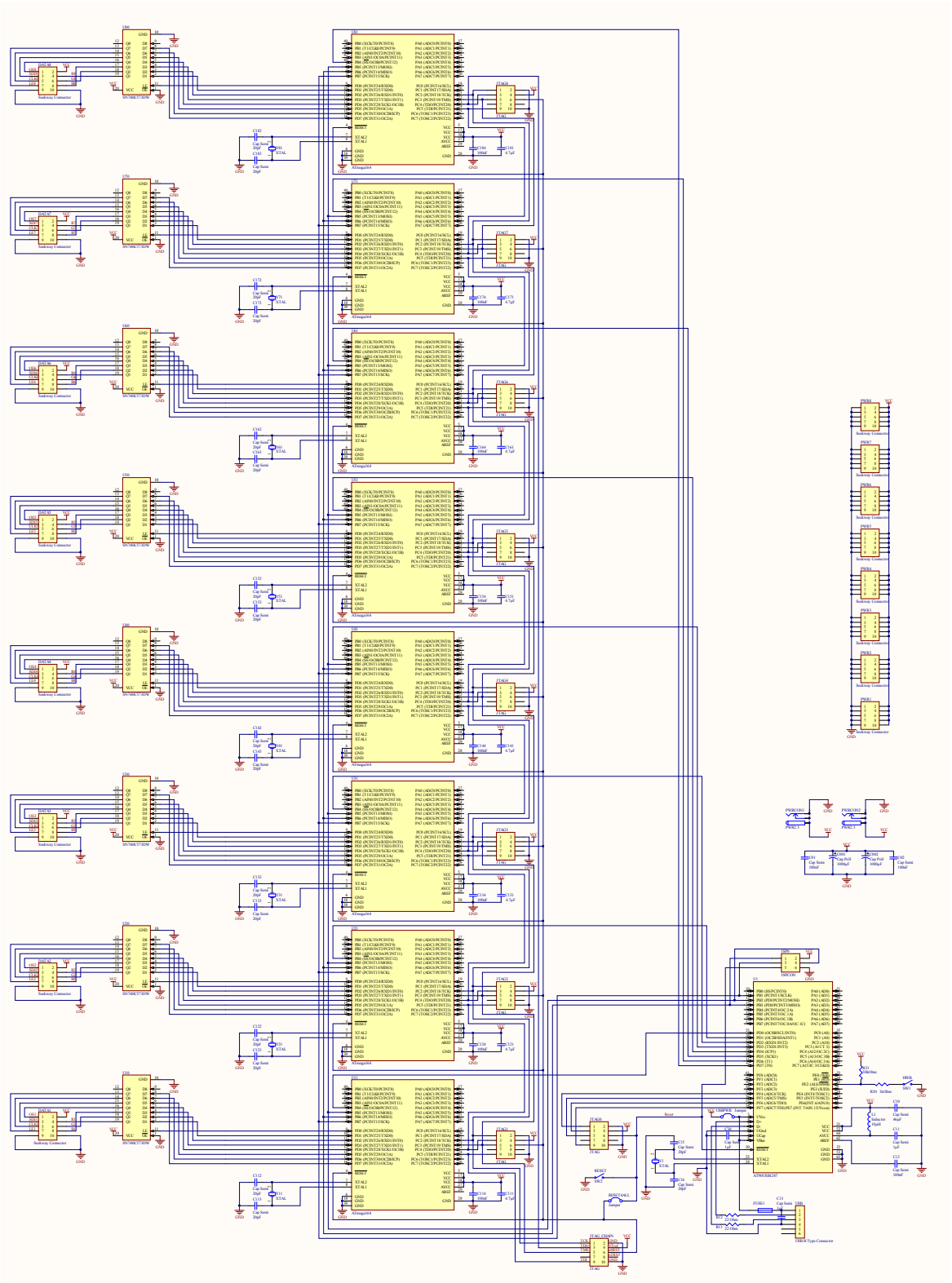

Abbildung 4.17: Schaltplan Hauptplatine für den LED Cube

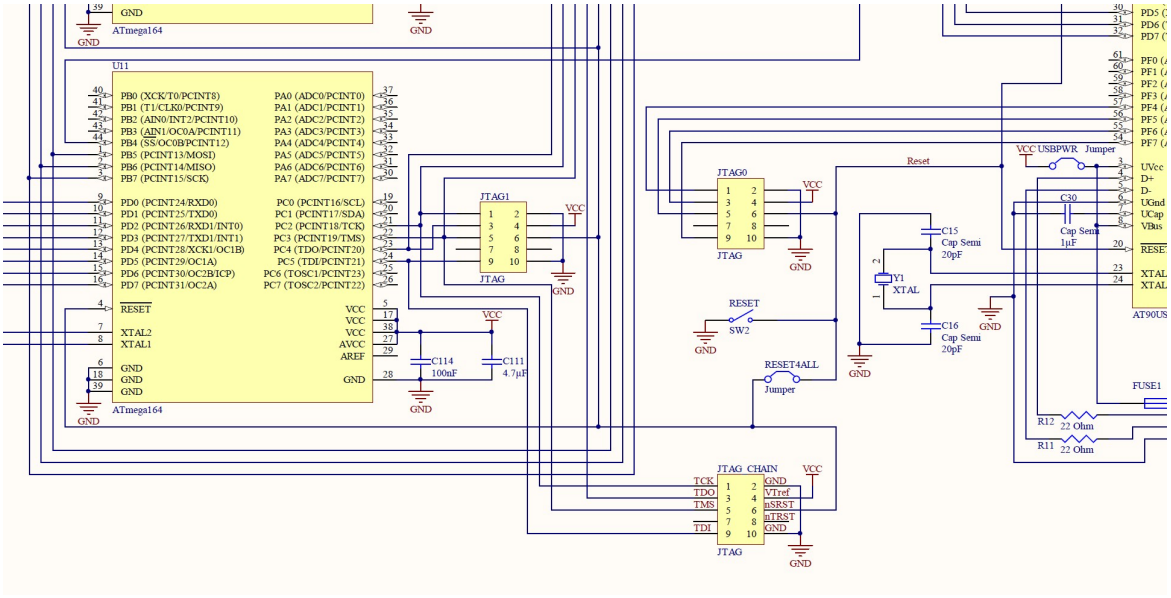

Abbildung 4.18: Schaltplan Hauptplatine für den LED Cube (Detailansicht)

4.6.2 Programmierschnittstelle - JTAG Daisy Chain

Der Aufbau der des Controllers auf der Hauptplatine wurde stark parallelisiert (Schaltplan Abb. 4.17). Jeder Ausgang zu einer Ebene wird über einen eigenen ATmega164 gesteuert, welcher über einen Referenztakt von 20 Mhz verfügt. Die Programmierung jedes einzelnen Mikrocontrollers geschieht wahlweise über das JTAG Interface direkt am Controller, oder über die JTAG-Daisy-Chain in Abbildung 4.19. Hierbei wird per Software gewählt, welcher Controller programmiert werden soll. Das Signal wird dann durch alle Mikrocontroller „durchgereicht“, jedoch nur der Controller, welcher über die gewählte ID verfügt, setzt die Programmierung um.

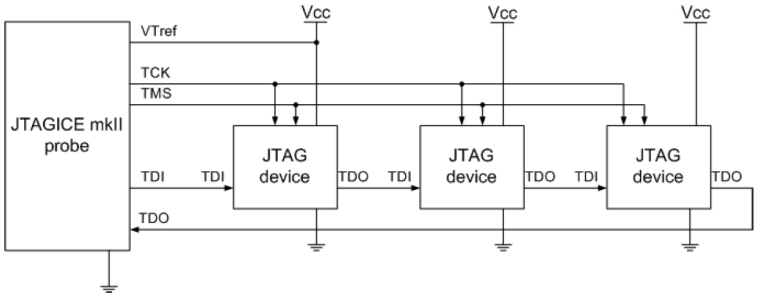

Abbildung 4.19: Typischer Aufbau einer JTAG-Daisy-Chain

4.6.3 Datenübertragung zwischen den Controllern - SPI

Die Kommunikation zwischen dem AT90USB1287 und den ATmega164 Controllern erfolgt ausschließlich unidirektional über das Serial Peripheral Interface (SPI). Jeder ATmega164 befindet sich im Slave-Modus und ist so immer bereit, neue Daten vom Master - dem AT90USB - zu erhalten. Diese Daten bestehen immer aus einer Liste von 64 RGB-Farbwerten. Diese werden nach Erhalt zyklisch auf einer Ebene dargestellt, bis neue Daten eintreffen. Durch diese Art von Puffer ist eine Synchronisierung der Ebenen und des Master-Controllers nicht mehr nötig. Es wird lediglich vom Master der entsprechende Slave mit Hilfe der Slave-Select bzw- Chip-Select Leitung ausgewählt und dann die Daten seriell übertragen.

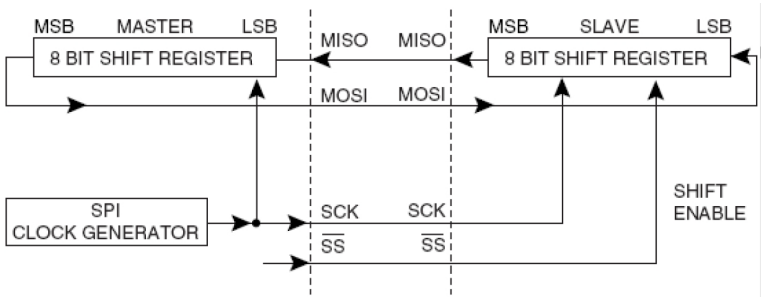

Abbildung 4.20: Prinzipielle Funktionsweise von SPI

Die Abbildung 4.20 zeigt die Funktionsweise von SPI. Sowohl Master, als auch Slave besitzen ein Schieberegister von dem aus die Daten bitweise über den SPI Bus „geschoben“ werden. Die Verbindungspunkte am AVR sind MOSI (Master-Out Slave-In) und MISO (Master-In Slave-Out) über welche immer genau ein Byte[2] übertragen werden. An SCK liegt das zugehörige Taktsignal an. SS ist die Low-aktive Slave-Select Leitung.
Da das Serial Peripheral Interface kein eigenes Datenprotokoll besitzt und Rohdaten Byteweise und ohne Fehlerkorrektur transportiert, ist es mit bis zu CLK/2 - also die Hälfte vom Referenztakt der CPU - sehr schnell und trotzdem sehr einfach zu konfigurieren. Nach erfolgreicher Initialisierung über das SPI Control-Register (SPCR) können beim Master direkt Daten in das SPI Data-Register (SPDR) geschrieben werden. Das SPI Transmission Flag (SPIF) gibt hierbei an, ob die Übertragung beendet ist. Selbiges gilt für den Slave, welcher aus dem SPDR ausliest und bis das SPIF gesetzt wurde.
Da der Cube über 512 LEDs verfügt und jede LED intern in einem 16-Bit Farbwert gespeichert ist, müssen für jede Ebene 1024 Bit Daten über das SPI gesendet werden. Der Gesamte Cube benötigt also für ein Bild 8192 Bit Rohdaten. Hinzu kommen bei jeder Ebene noch die Vorberechnung und jeweils zwei Bitänderungen an dem Port für die Slave-Select Leitungen. Letztere sind aber zu vernachlässigen. Unter der Annahme, dass der Cube sich

[2] Es werden immer 8 Bit übertragen - unabhängig von der Prozessorarchitektur

selbst mit mindestens 60 Bildern pro Sekunde aktualisieren kann, müssen also in jeder Sekunde 491.520 Bit die SPI Leitungen passieren. Die Frequenzeinstellung von f/32 für 500 KBit/s erscheint daher prädestiniert. Leider fehlt bei dieser Annahme der Einbezug der Vorberechnung bzw. das Auslesen der jeweils neuen LED Werte zwischen jedem gesendetem Byte. Damit das SPI bei der Aktualisierung keinen Flaschenhals darstellt, muss die Frequenz so hoch wie möglich gewählt werden.

Listing 4.6: Codebeispiel SPI

```
1  //Master-Initialisierung mit clk/16
2  SPCR = ((1<<SPE)|(1<<MSTR)|(1<<SPR0));
3
4  void SPISend(char cData)
5  {
6    SPDR = cData;
7    while(!(SPSR & (1<<SPIF)));
8  }
9
10 //Slave-Initialisierung
11 SPCR |= (1<<SPE);
12
13 char SPIReceive(void)
14 {
15   while(!(SPSR & (1<<SPIF)));
16   return SPDR;
17 }
```

Die beiden Funktionen im Beispiel 4.6 sind nach Initialisierung mit SPE (SPI Enable) im SPCR sofort einsatzbereit. SPISend kann bei Bedarf ausgeführt werden. SPIReceive muss jedoch in einer Endlosschleife kontinuierlich aufgerufen werden. Dies stellt gerade bei größeren Programmen ein Problem dar. Eine bessere Lösung ist es, wenn das Empfangen von Daten über den SPI Interrupt wie in Listing 4.7 geschieht. Dieser Interrupt wird ausgelöst, wenn ein neues Byte im SPDR steht und erfordert somit keine Endlosschleife.

Listing 4.7: Codebeispiel SPI Interrupt

```
1  //Slave-Initialisierung mit aktiviertem Interrupt
2  SPCR |= (1<<SPE)|(1<<SPIE);
3
4  ISR(SPI_STC_vect)
5  {
6    uint8_t NeueDaten = SPDR;
7    ...
8  }
```

4.6.4 Fertigstellung der Hauptplatine

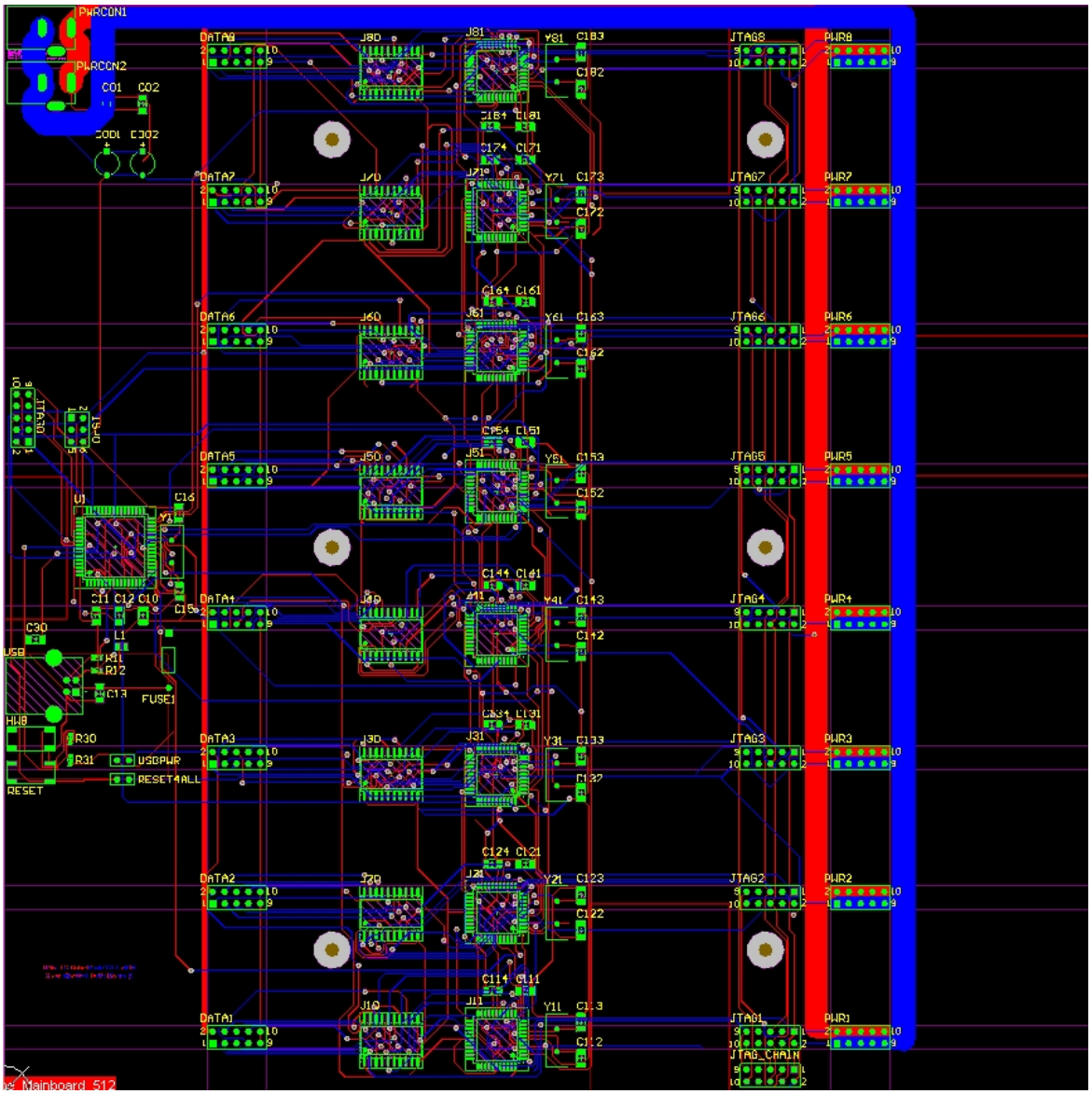

Abbildung 4.21: CAD Layout der Hauptplatine - Ansicht Unterseite

Die Abbildung 4.21 zeigt das Layout der Hauptplatine von der Unterseite. Oben links befinden sich zwei Buchsen parallel zur Stromversorgung um Übergangswiderstände an den Kontakten zu minimieren. Die Platzierung der Komponenten wurde so gewählt, dass die LED Module (Ebenen) in die dafür vorgesehenen Steckplätze gesteckt werden können und eine weitere Verkabelung entfällt. Durch die Platzierung der Steckkontakte ist auch ein einheitlicher Abstand zwischen den Ebenen sichergestellt. In der Grafik fehlt die Massefläche (GND),

die die gesamte Platine überdeckt und so sowohl als Hochfrequenzschirmung als auch als Minus-Pol für alle Komponenten dient.

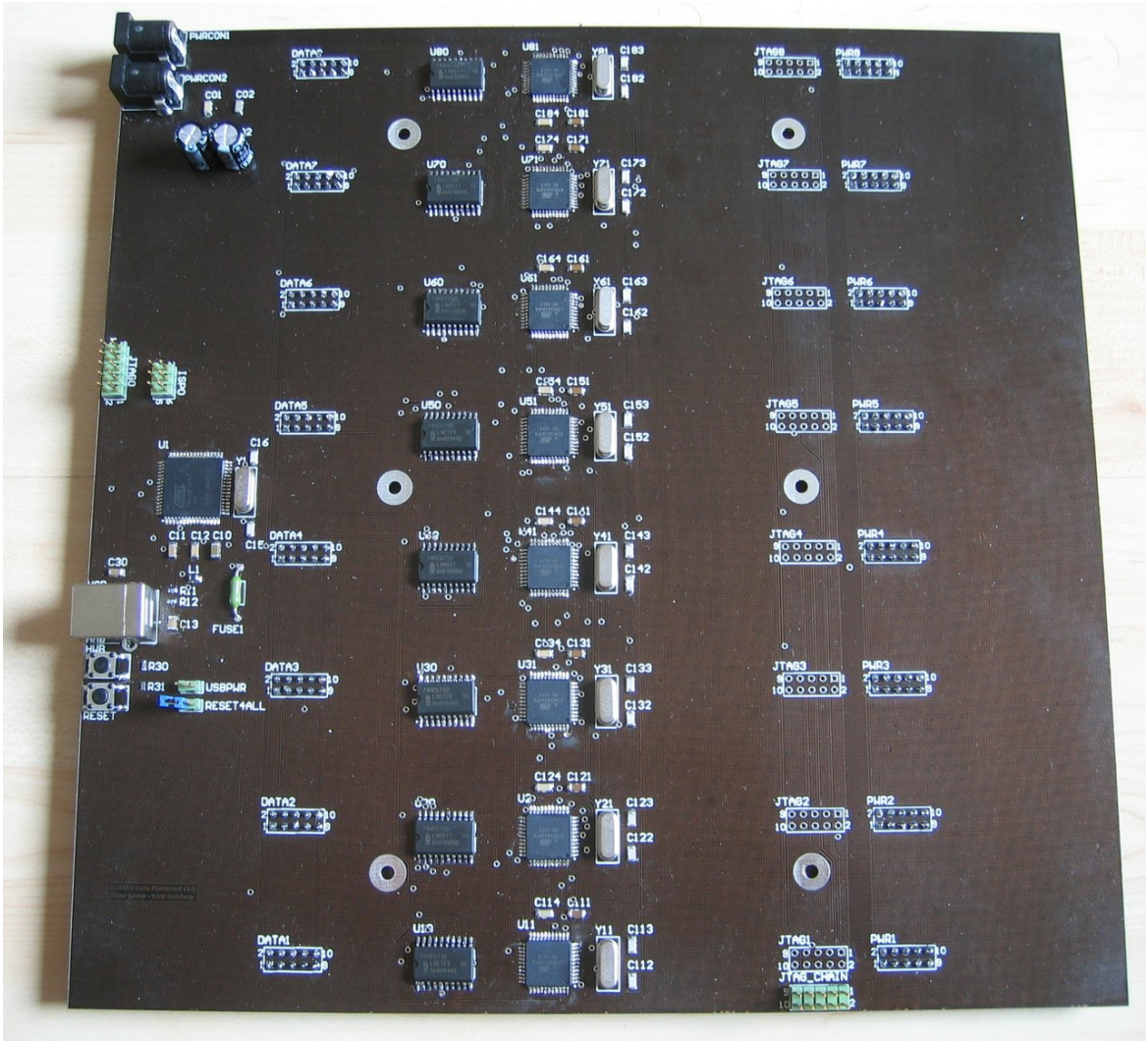

Abbildung 4.22: Fertig bestückte Hauptplatine - Ansicht Unterseite

Auf der linken Seite der Platine (Abb. 4.22) befindet sich der AT90USB1287 mit einem 16 Mhz Quarz und dem darüberlegenden eigenen JTAG und ISP Interface. Darunter direkt die USB-B Buchse und die Taster für Reset und HWB. Diese sind in der Praxis nicht nötig, doch für spätere Firmwareupdates ohne JTAG Adapter von Vorteil. Der Jumper USBPWR schaltet die Stromversorgung über den USB-Port ein und dient ausschließlich zum Programmieren des AT90USB1287 ohne externe Stromversorgung. Der darunterliegende RESET4ALL Jumper dient zur Trennung der Reset-Leitung zwischen dem USB Controller und den ATmega164 PWM-Controllern und sollte im Normalbetrieb gesteckt sein um einen gleichzeitigen

Reset auszulösen. Dieser ist bei der Programmierung und beim Debuggen jedoch eher unerwünscht, da sonst beim Programmieren über die JTAG Daisy-Chain jedes Mal auch die USB Verbindung unterbrochen wird.

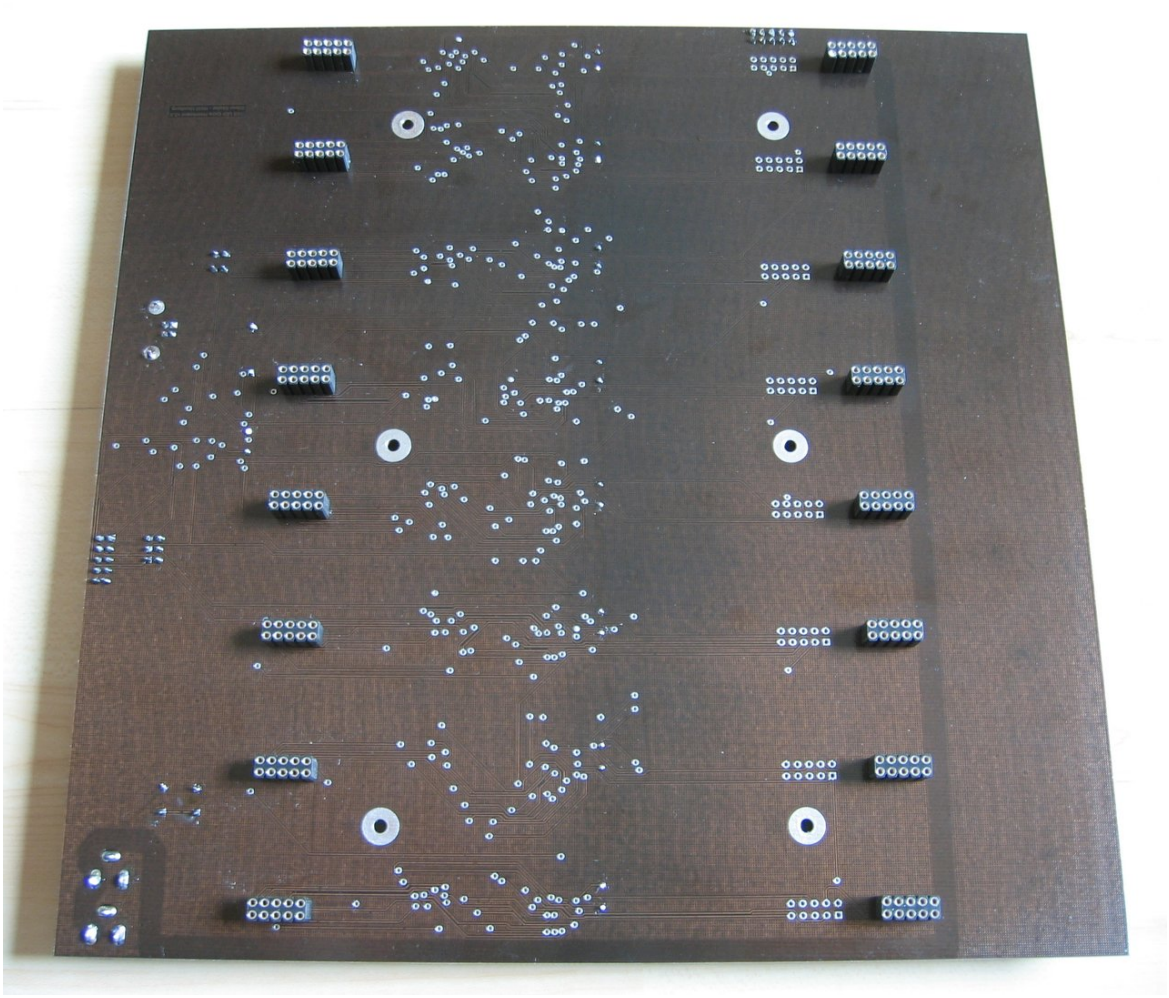

Abbildung 4.23: Oberseite der Hauptplatine

Die Oberseite der Platine ist nicht mit Bauteilen bestückt (siehe Abb. 4.23), da diese später als Bodenplatte für die aufgesteckten LED Module dient und somit für den Endanwender sichtbar bleibt. Gut sichtbar sind hier die Datenports auf der linken Seite und die Ports für die Stromversorgung auf der Rechten.

4.6.5 Test des fertigen Cubes

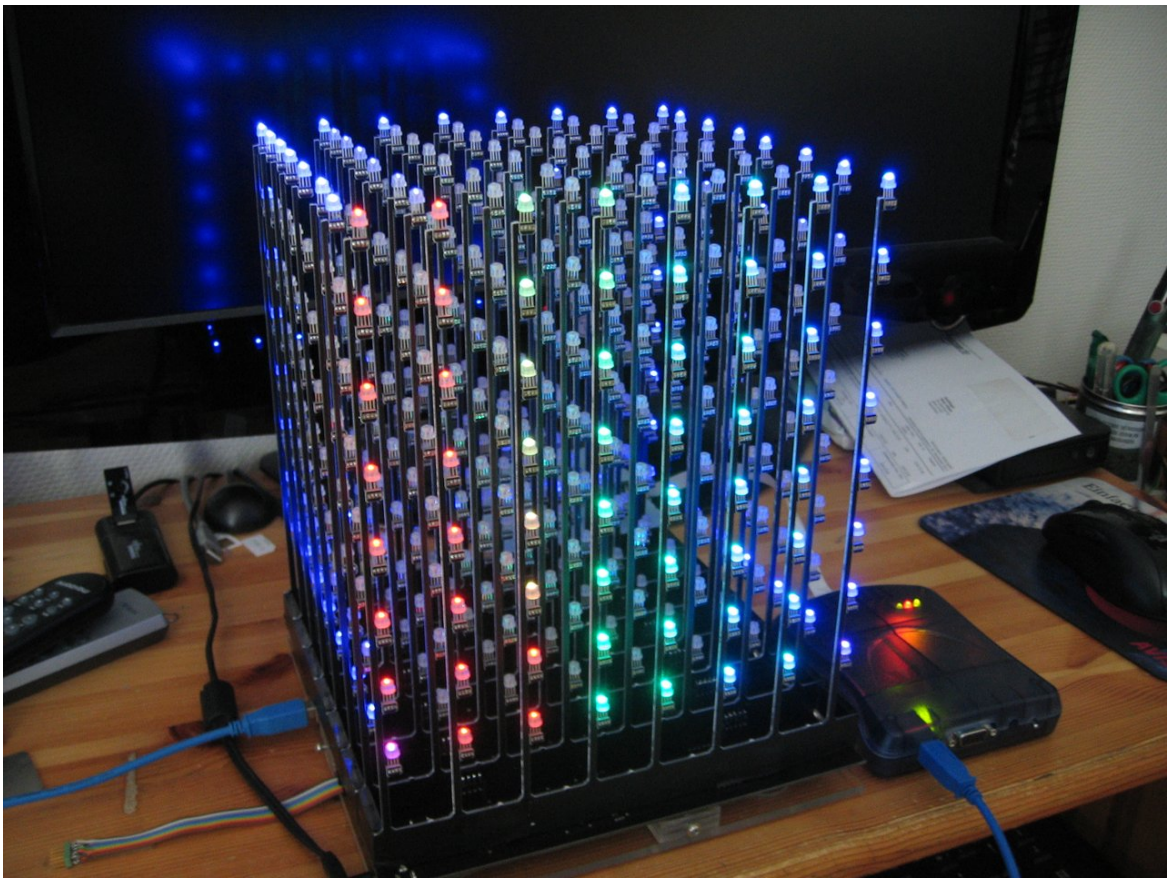

Abbildung 4.24: Mainboard mit aufgesteckten LED-Modulen

Der Test zur Ansteuerung aller Ebenen stellte sich als Teilerfolg heraus. Der Bildaufbau bei einer Übertragungsgeschwindigkeit von 250 kBit/s über SPI ist deutlich zu sehen und selbst 500 kBit/s erzeugt noch immer eine schwache aber doch sichtbare Verzögerung. Erst bei Takt von CLK/16, also 1 Mhz und einer Geschwindigkeit von 1 Megabit/s erzeugt einen Bildaufbau, der mit dem menschlichen Auge nicht mehr als solcher wahrzunehmen ist. Allerdings ist durch den erhöhten Takt von nun 20 statt 16 Mhz die „Schattenbildung“ wieder dominanter. Die erste LED links unten (welche eigentlich den RGB-Wert [15,0,0] haben soll) weist einen deutlichen Blauanteil von der letzten blauen LED auf. Dieses Phänomen kann bei jeder LED im Cube beobachtet werden (siehe Abb. 4.26) und ist auf die Schaltzeiten der MOSFETs zurückzuführen. Diese liegen bei einer Betriebsspannung von unter 10 Volt bei bis zu 80 ns für das Einschalten, und bei 140 ns beim Ausschalten. Zudem weisen die MOSFETs bei solch niedrigen Spannungen zwischen Source und Drain eine realitv hohe Kapazität auf (Abb. 4.25), was eine weitere Verzögerung beim Ausschalten zur Folge hat. Dies per Software auszugleichen ist nahezu unmöglich, da die Schaltzeit nicht konstant ist,

sondern sich mit der Spannung zwischen Source und Drain am MOSFET ändert. Auch ein Anheben der Versorgungsspannung ist keine Alternative, da kein Bauteil im Cube für eine Spannung über 5,5 Volt ausgelegt ist.

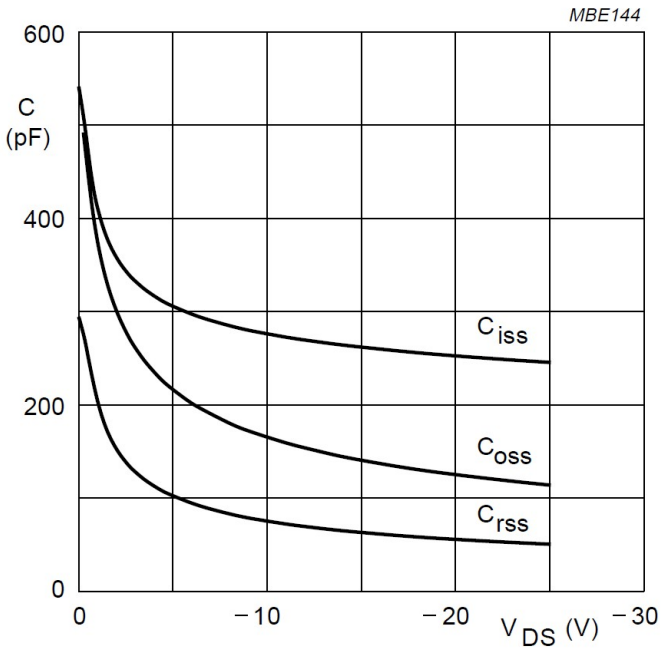

Abbildung 4.25: Kapazität[a] als Funktionsdiagramm der Drain->Source Spannung

[a] C_{iss} = input capacitance, C_{oss} = output capacitance, C_{rss} = reverse transfer capacitance

Verantwortlich für die schlechte Farbwiedergabe und das Dominieren von rot und grün gegenüber blau ist der gemeinsame Vorwiderstand, welcher durch den LED Treiber erzeugt wird. Dieser wird, abhängig von der Betriebsspannung beziehungsweise des Innenwiderstandes der LED, zur Laufzeit geregelt. Die Betriebsspannung für eine rote LED ist bedeutend geringer als die einer Blauen. Dadurch kann nie ein gleiches Mischverhältnis zwischen den drei Grundfarben erzeugt werden.

Das Problem der fehlenden Helligkeit ist auch zu einem Teil durch den gemeinsamen Vorwiderstand zu erklären, doch viel mehr fällt das Tastverhältnis von 1/64 ins Gewicht. Den Betriebsstrom der LEDs in diesem Verhältnis zu erhöhen ist keine Alternative, da weder die Leiterbahnen, noch die LED Treiber für viel höhere Ströme ausgelegt sind. Lediglich die MOSFETs könnten einen solchen Strom schalten - doch auch hier würde vermutlich die Schattenbildung einen Wert erreichen, der dazu führt, dass LEDs nicht mehr einzeln angesteuert werden können.

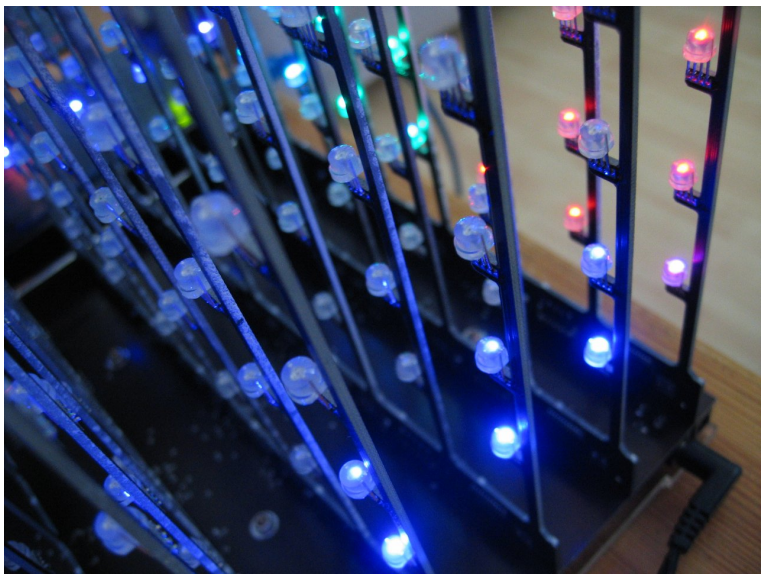

Abbildung 4.26: Schattenbildung durch Timing-Fehler

4.7 Neue Lösung der LED-Ansteuerung

Nach dem letzten Hardwaretest ist klar, dass die PWM Ansteuerung der LEDs grundlegend überarbeitet werden muss. Sowohl das träge Umschalten der Farben, als auch das Tastverhältnis der LEDs muss verbessert werden, um eine deutliche, kräftige und eindeutige Farbdarstellung zu gewährleisten.

4.7.1 Ansatz

Um den schnellen Wechsel der Farbe und gleichzeitig das Teilen des Vorwiderstandes zu umgehen, wird in jedem Durchgang nur noch eine einzelne Farbe angesteuert. Dieser Vorgang muss dadurch jedoch für jede Grundfarbe wiederholt werden. Das Tastverhältnis sinkt hierbei auf ein drittel von 1/64, also 1/192. Dies ist aber nur dann ein Problem, wenn bei jedem Durchgang nur eine LED gesetzt wird. Wenn nur eine Farbe zur Zeit anliegt, können auch mehrere LEDs aktiviert werden, die auf den aktiven Grundfarben-Anteil angewiesen sind. Dieser Gedanke wurde bereits im Abschnitt 4.1 angesprochen.
Ist nun also das Ziel, eine Ebene komplett rot zu färben, so werden High-Bits in die Schieberegister geschoben und danach der MOSFET für die Farbe Rot durchgeschaltet. Wiederholt man dies zyklisch für jede Farbe, erhält man die Farbe weiß mit einem Tastverhältnis von 1/3. Um verschiedene Farben zu mischen, reicht es nicht, bei jedem Durchgang die Farbe zu wechseln, da so nur 9 verschiedene Farben möglich wären. Die Abstufung in 16 Schritten pro Farbe erfolgt, ähnlich wie bei dem ersten Ansatz, durch Pulsweitenmodulation. Diese wird realisiert, indem der Durchgang für jede Farbe 15 mal wiederholt wird - also insgesamt

16 mal 64 Bit für eine Farbe (z.B. Rot) in die Schieberegister geschoben werden. Eine LED, die ein helles Rot darstellen soll, wird also bei jedem der 16 Rot-Durchgänge eine 1 erhalten. Dafür wird sie bei den Grün- und Blau-Durchgängen nur mit einer 0 gespeist und somit für diese Farbwerte abgeschaltet. Gleiches gilt natürlich auch für alle anderen Farben und deren Mischverhältnisse.

4.7.2 Beispielcode zur Ebenensteuerung - Versuch 3

Listing 4.8 zeigt den C-Quellcode einer ISR für den oben genannten Ansatz. In jedem Durchgang wird ein neues Bit über SDI in die Schieberegister geschoben. Alle 64 Durchgänge erhöht sich der PWM Zähler, welcher die Helligkeits-Abstufungen der jeweiligen Farbe angibt. Es sind also für jede Grundfarbe 1024 - für ein vollständiges Bild 3072 - Durchgänge nötig. Dieser Wert muss außerdem rund 60 Mal in der Sekunde erreicht werden um ein Flimmern zu vermeiden. Die ISR Frequenz sollte daher bei 184,32kHz liegen.

Listing 4.8: Neuer Ansatz zur Ebenensteuerung

```
1  ISR(TIMER0_COMPA_vect, ISR_BLOCK)
2  {
3    switch(color)
4    {
5      case RED:
6        if(pwm_value < framebuffer[RED][led])
7        {
8          PORTD |=  (1<<SDI)|(1<<CLK);
9        }else{
10         PORTD &= ~(1<<SDI);
11         PORTD |= (1<<CLK);
12       }
13       if(++led == 64)
14       {
15         led = 0;
16         PORTD |=  (1<<nG)|(1<<nB)|(1<<LE);
17         PORTD &= ~(1<<nR);
18         if(++pwm_value == 16)
19         {
20           pwm_value = 0;
21           color = GREEN;
22         }
23       }
24       break;
25
26     case GREEN:
27       if(pwm_value < framebuffer[GREEN][led])
28       {
29         PORTD |=  (1<<SDI)|(1<<CLK);
30       }else{
31         PORTD &= ~(1<<SDI);
32         PORTD |= (1<<CLK);
33       }
34
35       if(++led == 64)
36       {
37         led = 0;
```

```
38          PORTD |=  (1<<nR)|(1<<nB)|(1<<LE);
39          PORTD &= ~(1<<nG);
40          if(++pwm_value == 16)
41          {
42            pwm_value = 0;
43            color = BLUE;
44          }
45        }
46        break;
47
48      case BLUE:
49        if(pwm_value < framebuffer[BLUE][led])
50        {
51          PORTD |=  (1<<SDI)|(1<<CLK);
52        }else{
53          PORTD &= ~(1<<SDI);
54          PORTD |= (1<<CLK);
55        }
56        if(++led == 64)
57        {
58          led = 0;
59          PORTD |=  (1<<nR)|(1<<nG)|(1<<LE);
60          PORTD &= ~(1<<nB);
61          if(++pwm_value == 16)
62          {
63            pwm_value = 0;
64            color = RED;
65          }
66        }
67        break;
68      }
69      PORTD &= ~((1<<CLK)|(1<<LE));
70    }
71  }
```

Fazit: Im Vergleich zum Versuch 2 (Abschnitt 4.4.5) ist dieser Ansatz um den Faktor 48 langsamer, doch angesichts der Tatsache, dass sich das Tastverhältnis um den Faktor 21,3 (vorher 1/64, jetzt 1/3) gesteigert hat und außerdem sowohl die fehlerhafte Farbabmischung als auch die Schattenbildung vollständig eliminiert wurde, kann dies als die beste Lösung angesehen werden. Einzig das sehr hohe Interrupt-Aufkommen durch den Timer und die vergleichsweise lange Ausführungszeit der ISR kann als Problem gesehen werden, wenn die SPI Frequenz weiter erhöht wird. Tritt während der Ausführung der PWM-ISR ein SPI-Interrupt auf, wird dieser nicht erkannt und die Daten nicht gelesen. Die Folge sind fehlerhafte Daten im Framebuffer und somit eine fehlerhafte Darstellung im Gesamtergebnis.

Lösung des Interrupt-Problems: Um während der Ausführung der ISR keine SPI Interrupts zu „übersehen“, gibt es verschiedene Möglichkeiten. Zum einen kann die PWM-ISR Frequenz herabgesetzt werden, was aber zur Folge hat, dass der Bildaufbau flimmert. Zum anderen könnte man die SPI Frequenz am AT90USB herabsetzen um weniger SPI-Interrupts am ATmega164 auszulösen. Diese müsste aber wieder unter 1 Mhz liegen und ist folglich zu

langsam für eine flüssige Darstellung. Eine wirkliche Alternative ist es jedoch, bei dem PWM Generator komplett auf einen Timer und eine ISR zu verzichten, sondern die Ausführung in die Hauptschleife (Mainloop) zu verlegen. Diese arbeitet immer mit voller Geschwindigkeit (hier 20 Mhz) und kann durch Interrupts unterbrochen werden. Es kann jedoch nur noch ein Interrupt durch eingehende SPI Daten ausgelöst werden, welcher den Framebuffer aktualisiert und somit kurzzeitig die Darstellung unterbricht. Dieser Effekt ist jedoch in keinem Test bisher sichtbar geworden. Der einzig denkbare Nachteil dieser Lösung ist, dass keine Nebenberechnungen mehr ausgeführt werden können, ohne den Bildaufbau zu stören. Da es jedoch keine Funktionen gibt, die dies verlangen, kann man diesen Aspekt vernachlässigen.

4.8 Datenprotokoll zur Steuerung des Cubes

Das Datenprotokoll (Abb. 4.27), welches verwendet wird um den Cube zu steuern, basiert auf einem 32-Bit-Wert. Dieser Wert repräsentiert eine einzelne LED und beinhaltet sowohl deren Position im Cube, als auch die zugehörige Farbe. Ferner sind in diesen 32 Bit noch Steuerbefehle zum Aktualisieren und zur Auswahl von Animationen inbegriffen.

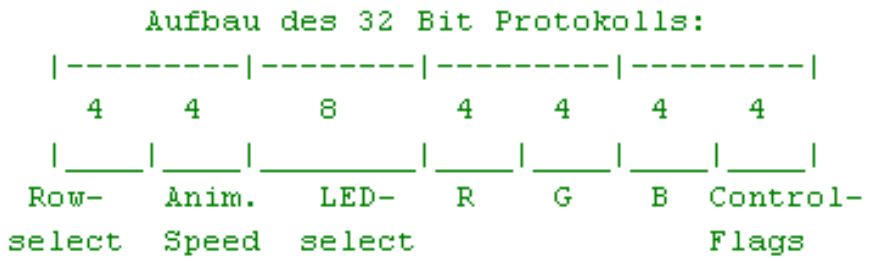

Abbildung 4.27: Datenprotokoll zur Steuerung des Cubes

Row-Select gibt an, in welcher Ebene sich die gewünschte LED befindet. Der Wert ist von vorne nach hinten durchnummeriert. Der Wertebereich liegt zwischen **0x0** und **0x7**.

Anim. Speed ist die Geschwindigkeit der gewählten Animation. Mögliche Werte sind **0x0** (keine Animation) bis **0xF** (schnelle Animation).

LED-Select ist die Auswahl der gewünschten LED innerhalb der Ebene, welche mit Row-Select gewählt wurde. Die Nummerierung erfolgt nach dem Schema der Abbildung 4.28 und umfasst den Wertebereich von **0x00** bis **0x3F**

Abbildung 4.28: Cube-Ebene durchnummeriert

R beinhaltet den 4-Bit Wert des Rot-Anteils der mit Row-Select und LED-Select gewählten LED. Der Wert darf im Bereich von **0x0** (aus) und **0xF** (maximale Helligkeit) liegen.

G beinhaltet den 4-Bit Wert des Grün-Anteils der mit Row-Select und LED-Select gewählten LED. Der Wert darf im Bereich von **0x0** (aus) und **0xF** (maximale Helligkeit) liegen.

B beinhaltet den 4-Bit Wert des Blau-Anteils der mit Row-Select und LED-Select gewählten LED. Der Wert darf im Bereich von **0x0** (aus) und **0xF** (maximale Helligkeit) liegen.

Control-Flags dienen zur Steuerung des Updates, zum Löschen des Cubes und zur Auswahl von Animationen.

Mögliche Werte sind:

0x0..0x7 - Update der Ebene 0 bis 7 (andere Ebenen sind dadurch nicht beeinflusst).

0xA,0xB - Auswahl einer der vorprogrammierten Animation[3].

0xC *clear* - Schaltet alle LEDs ab und löscht den Framebuffer.

0xF *full Update* - Gleichzeitiges Update aller Ebenen mit den Werten aus dem Framebuffer.

[3] Zum Zeitpunkt der Erstellung dieser Arbeit noch nicht vorhanden

Werden andere Werte als die oben angegebenen gesetzt (0x8, 0x9, 0xD oder 0xE), so wird keine Veränderung vorgenommen, sondern nur der LED-Wert im Framebuffer abgelegt. Dies ist hilfreich, um beispielsweise erst ein ein gesamtes Bild in den Cube zu laden und dieses vollständig mit dem Wert 0xF zu aktualisieren. Verzögerungen bei der LED Übertragung sind so nicht sichtbar.

Beispiele Um in Ebene 7 der dritten LED die Farbe Blau mit mittlerer mittlerer Helligkeit (Wert 8) zuzuweisen, genügt es, den Wert **0x70030087** zu senden. Die Aktualisierung erfolgt sofort mit dem Beenden der Übertragung.

Soll in Ebene 7 die LED-Reihe auf der rechten Seite vollständig grün mit maximaler Helligkeit gefärbt werden, so erfordert dies das Senden von mehreren Befehlen:

0x70300f0e
0x70370f0e
0x70380f0e
0x70390f0e
0x703a0f0e
0x703b0f0e
0x703c0f0e
0x703d0f0e
0x703e0f0e
0x703f0f0f

Wie man Sieht, erfolgt von LED **30** bis **3e** keine Aktualisierung. Die Daten werden lediglich in den ersten Zwischenspeicher übernommen und noch nicht an die Ebenen weitergeleitet. Erst mit dem letzten Befehl, welcher am Ende ein **f** enthält, wird der gesamte Cube mit den Werten aus dem ersten Zwischenspeicher aktualisiert. Auf diesem Wege lassen sich beliebig viele Änderungen - beispielsweise ein gesamtes *Bild* im Cube - einspeisen, ohne dass die Veränderung nach außen sichtbar ist.

5 Fazit

Die Aufgabenstellung war, einen Cube mit 512 RGB LEDs zu entwickeln. Dieser sollte über Mikrocontroller gesteuert werden und über eine USB Schnittstelle verfügen. Die Anforderungen wurden im Rahmen dieser Bachelorarbeit vollständig umgesetzt und in Form eines funktionierenden Prototyps hergestellt. Dieser Prototyp arbeitet unter Laborbedingungen fehlerfrei und kann sowohl unter Windows als auch unter Linux als 3D Display eingesetzt werden. Wird der Cube auch als solches eingesetzt, und immer vollständig mit 512 neuen Farbwerten beschrieben, so liegt die maximale Aktualisierungsrate bei 122 Hz, da diese von der Geschwindigkeit des SPI Busses abhängt. Maßgeblich für die tatsächliche Aktualisierungsrate ist jedoch das Steuerprogramm des Cubes. Tests haben gezeigt, dass unter Windows im Vollbildverfahren der Cube mit durchschnittlich 20 Bildern in der Sekunde beschrieben werden kann. Dies ist jedoch nicht zwingend praxisnah, da auch nur einzelne Ebenen beschrieben und aktualisiert werden können, was mehr Freiraum zur Optimierung lässt.

Bekannte Probleme und ihre Lösungsvorschläge

Stromversorgung: Der Cube muss mit einer konstanten Gleichspannung von 5 Volt betrieben werden und erfordert somit Ströme von bis zu 16 Ampere. Dadurch müssen die Zuleitungen möglichst kurz gehalten werden und die Steckverbindungen einen geringen Übergangswiderstand aufweisen. Außerdem ist der Cube weder gegen Verpolung, noch gegen Überspannung gesichert. Aus diesen Gründen wäre es von Vorteil gewesen, eine interne Stromversorgung in Form eines getakteten Schaltnetzteils oder verteilter Festplanungsregler im Design mit einzuplanen und die Versorgungsspannung so auf beispielsweise 12 oder gar 24 Volt anzuheben. Schon bei 12 Volt würde der Strom auf 6 bis 7 Ampere absinken, was eine deutliche Entlastung der Stromversorgungseinheit bedeuten würde.

USB Geschwindigkeit: Die aktuelle Datenübertragung arbeitet auf CDC/ACM Ebene und erfordert für jede LED vier einzelne Datenpakete von jeweils einem Byte Größe und alle 4 Byte insgesamt 5 Funktionsaufrufe. Der Vorteil ist natürlich, dass man so auch einzelne LEDs setzen kann, ohne komplette Bilder zu übertragen, doch für jedes übertragene Byte wird auch ein ganzer USB-Frame verwendet, welcher die minimale Größe von 8 Byte hat und auch jedes mal mit einem 8 Byte großen ACK bestätigt werden muss. Dies kostet Bandbreite. Für Animationen wäre es sinnvoller, komplett auf CDC zu verzichten, sondern die Daten in größeren Blöcken als Bulk Transfer zu senden. Theoretisch könnte man so nicht

nur die vollen 8 Byte nutzen, sondern auch die Endpoint-Size auf bis zu 64 Byte[1] vergrößern. Dies würde jedoch voraussetzen, dass für jedes Betriebssystem ein individueller Treiber geschrieben wird, der wiederum die Schnittstelle zwischen dem Betriebssystem und dem Cube herstellt.

Farbdarstellung: Durch den Zusammenhang zwischen der Helligkeitswahrnehmung des menschlichen Auges und der linearen Abstufung der Helligkeiten der drei Grundfarben, erscheinen die Unterschiede in den ersten 7 Stufen sehr deutlich und werden verschwindend gering in den Stufen 8 bis 15. Der Grund hierfür ist die nahezu logarithmische Kennlinie des menschlichen Auges, welche schon durch das Weber-Fechner-Gesetz beschrieben wurde (Books 2010, vgl.). Abhilfe schafft daher nur die Erweiterung des PWM Algorithmus' mit einer Tabelle aus logarithmischen Werten, welche die neuen Helligkeitswerte repräsentieren. Hierbei ist zu beachten, dass hierzu zusätzliche PWM Stufen benötigt werden um diese Werte auch zu erreichen. Da dies für jede Grundfarbe geschehen muss, erfordert jede weitere Stufe drei zusätzliche Aktualisierungszyklen einer Ebene. Nicht zu vergessen ist auch die zusätzlich benötigte Rechenzeit, die die großen Variablenwerte und die Zugriffe auf Arrays verlangen. Schon die Erweiterung von 16 auf 48 Stufen pro Farbe erzeugt ein leicht wahrnehmbares Flimmern im unteren Helligkeitsbereich. Aus diesem Grund ist diese Funktion bisher nur zu Testzwecken im Quellcode implementiert und nicht im endgültigen Produkt vorgesehen.

[1] Im HighSpeed Modus sind auch 512 Byte möglich

6 Aussichten

Der RGB LED Cube ist voll funktionsfähig und in diesem Aufbau sofort einsatzbereit. Das potenzielle Einsatzgebiet ist breit gefächert und durch die USB Schnittstelle und das leicht verständliche Datenprotokoll lässt sich der Cube an nahezu jedem Endgerät betreiben, welches USB-CDC/ACM „versteht“. Dadurch, dass das Datenprotokoll einheitlich gehalten ist, besteht auch keine Beschränkung bezüglich der Programmiersprache der Steuersoftware. Der Zugriff über die Windows- oder Linux-API auf einen virtuellen COM-Port gestaltet sich beispielsweise unter C genau so einfach wie unter Java oder Python.

Erweiterung/Update der Software: Es ist jederzeit möglich, einen Teil des Cube-Controllers (AT90USB1287) mit einem Firmwareupdate zu versehen, da dieser mit einem USB Bootloader ausgestattet wurde. Aktivieren kann man diesen durch einen Druck auf den Reset-Taster bei gleichzeitigem Halten des HWB-Tasters. Eine neue Firmware kann dann z.B. über das ATMEL-eigene Programm „FLIP3“ in Form einer .HEX-Datei aufgespielt werden. Eine Programmierung über JTAG oder ISP ist nicht nötig. JTAG wird nur benötigt, wenn die Firmware der einzelnen Ebenen geändert werden soll, da diese über keinen eigenen Bootloader verfügen.

Erweiterung/Update der Hardware: Prinzipiell ist es möglich, die Größe des Cubes durch Vervielfachung der Bauteile zu erweitern. Mit 12-Bit-PWM wäre es sogar möglich 16x16, also 256 LEDs pro Ebene mit nur einem Atmega164 anzusteuern und so einen Cube mit 4096 LEDs aufzubauen - selbstverständlich mit USB Schnittstelle. Dies würde jedoch voraussetzten, dass alle Punkte im letzten Kapitel berücksichtigt werden und das Hardwaredesign - insbesondere die Stromversorgung - angepasst wird.

Abkürzungsverzeichnis

3D	Dreidimensional
PC	Personal Computer
LED	Light Emmitting Diode
OLED	Organic Light Emmitting Diode
PWM	Pulse Width Modulation
LC	Liquid-Crystal
SMD	Surface Mount Device
RGB	Red Green Blue
MOSFET	metal oxide semiconductor field-effect transistor
USB	Universal Serial Bus
SD	Secure Digital
TCP	Transmission Control Protocol
IP	Internet Protocol
DHCP	Dynamic Host Configuration Protocol
UART	Universal Asynchronous Receiver Transmitter
OTG	On-The-Go
RISC	Reduced Instruction Set Computer
CPU	Central Processing Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
JTAG	Joint Test Action Group

ISP	In Circuit Programming
ISR	Interrupt Service Routine
IC	Integrated Circuit
IO	Input/Output
HWB	Hardware Bootloader
SDI	Serial Data In
SDO	Serial Data Out
LE	Latch Enable
OE	Output Enable
CLK	Clock
VCC	Voltage of the Common Collector
GND	Ground
ACM	Abstract Control Model
CDC	Communications Device Class
ACK	Acknowledgement Packet
LUFA	Lightweight USB Framework for AVRs
DPRAM	Dual Port Random Access Memory
VID	Vendor Identification Number
PID	Product Identification Number
ICE	In Circuit Emulator
BGA	Ball Grid Array
TQFP	Thin Quad Flat Package
SPI	Serial Peripheral Interface
MOSI	Master-Out Slave-In
MISO	Master-In Slave-Out

SS Slave Select

SCK Source Clock

SPCR SPI Control-Register

SPDR SPI Data-Register

SPIF SPI Transmission Flag

Literaturverzeichnis

Atmel 2006

ATMEL: *AVR109: Self Programming*. 2006. – http://www.atmel.com/dyn/resources/prod_documents/doc1644.pdf - Aktualisiert: 2006-04-01

Atmel 2008a

ATMEL: *AVR042: AVR Hardware Design Considerations*. 2008. – http://www.atmel.com/dyn/resources/prod_documents/doc2521.pdf - Aktualisiert: 2009-12-01

Atmel 2008b

ATMEL: *AVR151: Setup And Use of The SPI*. 2008. – http://www.atmel.com/dyn/resources/prod_documents/doc2585.pdf - Abruf: 2008-07-01

Axelson 2007

AXELSON, Janet L.: *USB 2.0 Handbuch für Entwickler*. Lakeview Research, 2007. – ISBN 3-826-61690-1

Bauckholt 2004

BAUCKHOLT, Heinz-Josef: *Grundlagen und Bauelemente der Elektrotechnik*. Hanser Fachbuchverlag, 2004. – ISBN 3-446-22708-3

Berger 2009

BERGER, Uwe: *Brandenburger Linux User Group - 3D-LED-Display*. 2009. – http://www.bralug.de/wiki/3D-LED-Display - Abruf: 2010-01-18

BetaLayout 2009

BETALAYOUT, GmbH: *Strombelastbarkeit von Kupfer(CU)-Leiterbahnen auf Basismaterial*. 2009. – http://www.pcb-pool.com/download/spezifikation/deu_cmso001_strombelastbarkeit.pdf - Abruf: 2010-04-14

Books 2010

BOOKS, LLC: *Psychophysics: Mel Scale, Psychoacoustics, Stroop Effect, Detection Theory, Stevens' Power Law, Weber-Fechner Law, Absolute Threshold*. Life Journey, 2010. – ISBN 1-156-90775-6

Camera 2010

CAMERA, Dean: *Four Walled Cubicle - Lightweight USB Framework for AVRs*. 2010. – http://www.fourwalledcubicle.com/LUFA.php - Abruf: 2010-06-03

Clar u. a. 2002

CLAR, James ; HOLOUBEK, Todd ; JEFFERS, Cindy ; LEE, Danielle: *James Clar and Associates - Lighting Design [3D Display Cube v1]*. 2002. – http://www.jamesclar.com/product/2002/3dcubev1/index.html - Abruf: 2010-01-19

ECS 2010

ECS, INC. I.: *Oscillation Circuit Design Considerations*. 2010. – http://www.ecsxtal.com/store/pdf/oscir_des.pdf - Abruf: 2010-06-08

Hulzebosch 2008

HULZEBOSCH, Jürgen: *USB in der Elektronik: Die USB-Schnittstelle für praktische Anwendunden am PC einsetzen*. Franzis, 2008. – ISBN 3-772-34089-X

Lampert 2000

LAMPERT, Timm: *Zur Wissenschaftstheorie der Farbenlehre*. Books on Demand Gmbh, 2000. – ISBN 3-898-11893-2

Lohninger 2008

LOHNINGER, Hans: *eBook: Angewandte Mikroelektronik*. 2008. – http://www.vias.org/mikroelektronik/bin/mikroelektronik.zip - Abruf: 2010-07-03

Lomont und Foulk 2008

LOMONT, Chris ; FOULK, Gene: *The 3D LED Cube*. 2008. – http://www.lomont.org/Projects/LEDCube/LEDCube.php - Aktualisiert: 2008-06-17

Lottor 2008

LOTTOR, Mark: *Cubatron Jr.* 2008. – http://www.3waylabs.com/projects/cubatron/ - Abruf: 2010-02-22

Markson und Balciunas 2010

MARKSON, Ted ; BALCIUNAS, Edmundas: *Lumisense*. 2010. – http://www.lumisense.com/?page=eightcubed - Aktualisiert: 2010-01-19

Ongsiek u. a. 2008

ONGSIEK, Martin ; FUHRMANN, Peter ; LÜMMER, Jannah ; BRONSTEIN, Jörg: *Borg3d Bauanleitung*. 2008. – http://www.das-labor.org/wiki/Borg3d_Bauanleitung - Aktualisiert: 2009-01-13

Peacock 2007
PEACOCK, Craig: *USB in a Nutshell*. 2007. – http://www.beyondlogic.org/usbnutshell/usb1.htm - Abruf: 2010-06-03

Salewski 2010
SALEWSKI, Dr. S.: *Homepage von Stefan Salewski - Experimentierplatine für den AT90USB (1287)*. 2010. – http://www.ssalewski.de/AT90USB_board.html.de - Aktualisiert: 2010-01-19

Schmitt 2010
SCHMITT, Günter: *Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie: Programmierung in Assembler und C - Schaltungen und Anwendungen*. Oldenbourg Verlag München Wien, 2010. – ISBN 3-486-58988-1

Seekway 2009
SEEKWAY, Jiangmen Technology L.: *3D LED Cube,LED Cube,3D LED Display,Light Cube*. 2009. – http://www.seekway.com.cn/e/ledsys10.htm - Abruf: 2010-01-18

Tietze und Schenk 1989
TIETZE, Ulrich ; SCHENK, Christoph: *Halbleiter-Schaltungstechnik*. Springer, 1989. – ISBN 3-540-19475-4

usb.org 2000
USB.ORG: *Universal Serial Bus Specification Rev. 2.0 (27.04.2000)*. 2000. – http://www.usb.org/developers/docs/ - Abruf: 2010-06-08

Wiegelmann 2009
WIEGELMANN, Jörg: *Softwareentwicklung in C für Mikroprozessoren und Mikrocontroller: C-Programmierung für Embedded-Systeme*. Hüthig Verlag Heidelberg, 2009. – ISBN 3-778-54052-1

Williams 2003
WILLIAMS, Al: *Build Your Own Printed Circuit Board*. Mcgraw-Hill Professional, 2003. – ISBN 0-071-42783-X

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 15. Dezember 2010	
Ort, Datum	Unterschrift