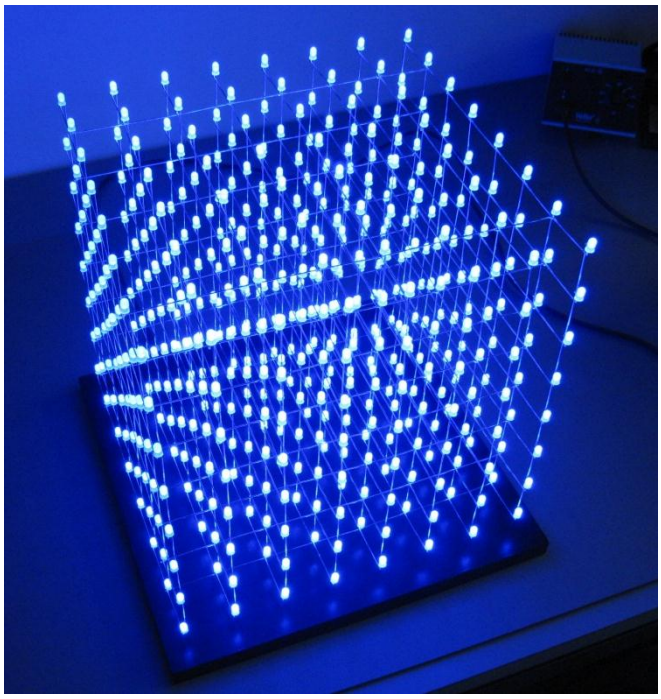
LED-Cube

Projektbeschreibung



Version 1.0

Inhalt

1. Vorwort

LED-cubes findet man mittlerweile viele im Netz. Auch einige Anleitungen zum Bau und deren Programmierung gibt es. In meiner Freizeit habe ich ebenfalls solch ein Projekt gestartet. Die folgende Beschreibung soll für Interessierte einige Anregungen liefern. Sie stellt aber keinesfalls ein ausführliches Tutorial dar. Insgesamt ist der Text sehr knapp gehalten.

Grob ist die Projektbeschreibung in die folgenden drei Kategorien eingeteilt:

1. Hardware
2. Software
3. Stromversorgung

2. Hardware

2.1. Würfelgröße

Die Größe der LED-cubes liegt im Allgemeinen zwischen 3x3x3 und 10x10x10. Die Zahlen bezeichnen dabei die LEDs pro Reihe, Spalte und Ebene. Ich habe mich für 8x8x8 entschieden. Dies ist die maximale Größe, die zur Ansteuerung mit 8bit-Mikrocontrollern noch gut geeignet ist. Je mehr LEDs, desto besser sieht der Cube am Ende natürlich aus, aber auch der Aufwand von Bau und Programmierung steigt an. Mit 8x8x8 Kubus ist man bei einer LED Anzahl von 512.
Der Abstand zwischen zwei direkt benachbarten LEDs beträgt bei mir 4 cm. Der Würfel besitzt demnach eine Kantenlänge von 28 cm.

2.2. Auswahl der LEDs

Die Farbauswahl ist Geschmackssache. Dennoch einige Kommentare zu verschiedenen Farben:

- Bei Grün (ca. 555nm) ist die Farbempfindlichkeit beim menschlichen Auge am höchsten.
- Bei LEDs in den Farben Blau und Weiß liegt die Durchflussspannung bei ca. 3,2 V, während sie z.B. bei den Farben Rot oder Gelb bei ca. 2,1 V liegt.
- Blaue und weiße LEDs sind meist etwas teurer
- Pinke und türkise LEDs sind mittlerweile zu akzeptablen Preisen verfügbar

Die meisten LED-cubes, die ich gesehen habe, sind blau und auch ich habe mich für diese Farbe entschieden.
Es gibt auch die Möglichkeit RGB-LEDs zu benutzen, sodass der Würfel nahezu jede beliebe Farbe annehmen kann. Der Hardware und Software Aufwand steigt aber enorm an.

Ein LED-Würfel wird mit bedrahteten LEDs aufgebaut. Diese sind standardmäßig in Gehäusen mit 3 mm und 5 mm Durchmesser erhältlich, wobei hier für den Würfel letztere verwendet werden.

Standard LEDs oder Low-current LEDs sind für den Einsatz in einem LED-cube aufgrund der relativ geringen Lichtintensität nicht geeignet (Bild 1, Nr. 3). Deren sind ultrahelle LEDs vorzuziehen. Mit der Lichtintensität sollte aber nicht übertrieben werden, da sonst die Nachbarleds, vor allem die LED auf der nächst höheren Ebene zu stark angestrahlt werden. Aus diesem Grund ist auch die Abstrahlcharakteristik der LEDs äußerst wichtig. Klare LEDs besitzen einen kleinen Abstrahlwinkel von ca. 20°, sie leuchten also hauptsächlich nach oben (Bild 1, Nr. 4). Bei LEDs mit diffusem Gehäuse liegt der Winkel

bei ca. 60° und gleichzeitig kann hier das Leuchten auch gut von der Seite wahrgenommen werden (Bild 1, Nr. 1 & Nr. 5). Es sind auch LEDs mit 120° Abstrahlwinkel (Bild 1, Nr. 2) verfügbar, die aber wieder ein klares Gehäuse besitzen. Zusammenfassend konnte ich die besten Ergebnisse bei diffusen LEDs erzielen. Bevor die LEDs für den Würfel in großen Stückzahlen gekauft werden, sollte man zunächst einige verschiedene LEDs bestellen und vorab testen.
Bild 1 zeigt den Vergleich einiger LEDs bei einem Strom von $I = 28\ \mathrm{mA}$. Achtung: Auf dem Bild sehen die Farben anders aus als in der Realität.

Bild 1: Vergleich von LEDs, $I = 28\ \mathrm{mA}$, von links: rot diffus; rot φ=120°; rot Standard; weiß klar φ=20°; blau diffus

2.3. Prinzip der Ansteuerung und IO-Ports am µC

Ein LED-cube besitzt eine große Anzahl an LEDs, die nicht direkt von einem Mikrocontroller angesprochen werden können. Aus diesem Grund bedient man sich zwei Prinzipien, um die Anzahl der benötigten IO-Ports auf ein sinnvolles Maß zu reduzieren. Diese sollen hier aber nur kurz angerissen werden.

Zeitmultiplex:
Die einzelnen Ebenen des Würfels werden zeitlich hintereinander angesteuert. Dies erfolgt so schnell, dass der Eindruck entsteht, alle Ebenen würden gleichzeitig leuchten. Das Prinzip soll anhand Bild 2 noch einmal beispielhaft mit fünf Ebenen verdeutlicht werden. Ein ausgefülltes Kästchen bedeutet dabei, dass die entsprechende Ebene aktiviert ist. Für ein flimmerfreies Bild sind Wiederholfrequenzen über 50 Hz, besser noch über 100 Hz nötig.

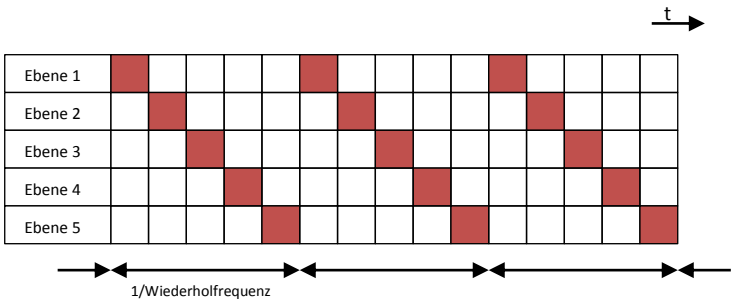


Bild 2: Prinzip des Zeitmultiplex

Seriell-zu-Parallel (SIPO):

Ein Schieberegister wird seriell mit Daten geladen (SI: serial in), die parallel/gleichzeitig an die LEDs ausgegeben werden können (PO: parallel out).

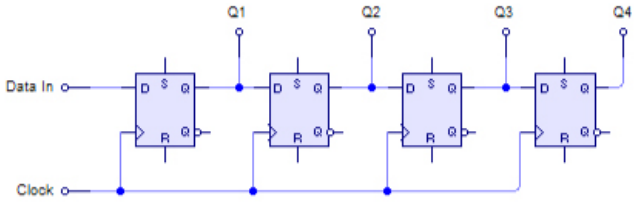


Bild 3: Schieberegister[1]

IO-Ports

Für meinen 8x8x8 LED-cube benötige ich mit acht 8-bit Schieberegistern für die Spalten und einer direkten Ansteuerung der acht Ebenen (ohne 1-aus-8 Decoder)

Data In	8
Clock	1
Latch Enable	1
Auswahl der Ebene	8
	18

IO-Ports am Mikrocontroller.

2.4. Steuerelektronik

Die Steuerelektronik besteht hauptsächlich aus einer Treiberstufe für die LEDs und einem Mikrocontroller zur Berechnung der Animationen. Beim Mikrocontroller habe ich mich für den ATmega32 entschieden, der mit 32 kByte Flash ausreichend Speicherplatz für Animationen bietet und mit einer Taktfrequenz von 16 MHz betrieben werden kann.

Beim Aufbau der Treiberstufe gibt es vielfältige Möglichkeiten. Da die Leuchtstärke von LEDs um einen Betriebspunkt annähernd linear zum eingeprägten Strom ist, ist auch eine Treiberstufe mit Stromausgang sinnvoll. Dies hat den Vorteil, dass der Strom durch die LED auf einen vorgegeben Wert geregelt wird und der Wert der Betriebsspannung, sowie Widerstände im Leistungspfad keine Auswirkung auf die Leuchtstärke haben. Der MAX6968 von Maxim erfüllt die von mir gestellten Anforderungen. Dabei handelt es sich um ein 8-bit Schieberegister mit seriellem Dateneingang und acht parallelen Konstantstromausgängen. Der LED Strom wird über einen externen Widerstand am IC eingestellt und beträgt maximal 55 mA pro Ausgang. Da die LED-Treiber Strom nach GND „sinken“,

[1] Quelle: www.en.wikipedia.org

müssen die Ebenen über Schalter auf die positive Versorgungsspannung gelegt werden. Da zur Ansteuerung von n-Kanal MOSFETs eine zweite Versorgungsspannung oder Levelshifter nötig sind, bieten sich hierfür p-Kanal MOSFETs an. Die geringere Löcherbeweglichkeit ist in dieser Leistungsklasse und Einsatzart unerheblich. Durch die Versorgungsspannung des Cubes von 5 V, sind hier Logic-Level FETs mit einer niedrigen Einsatzsspannung zu verwenden. Um den Mikrocontroller etwas zu entlasten und das Ansteuersignal zu invertieren, werden die MOSFETs über zusätzliche Treiberbausteine angesprochen. Wegen der relativ niedrigen Schaltfrequenz von ca. 100 Hz könnten diese aber auch direkt vom Mikrocontroller angesteuert werden.
Bild 4 zeigt den Schaltplan der Steuerelektronik, Bild 5 das zugehörige Layout. Das Board, hergestellt von einem Leiterplattenservice, besitzt zwei Lagen und ist beidseitig mit Lötstopplack versehen.

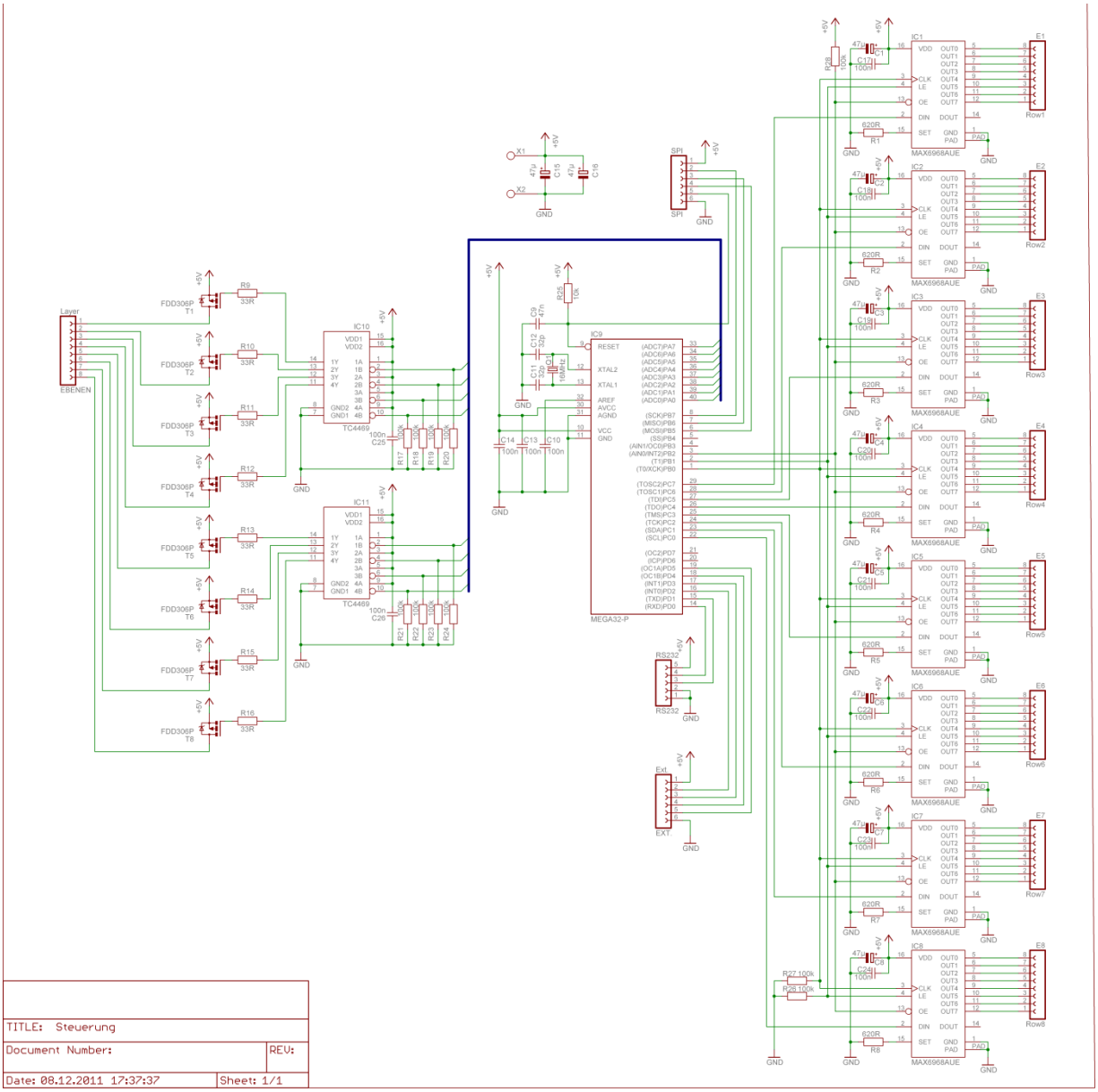


Bild 4: Schaltplan der Steuerelektronik

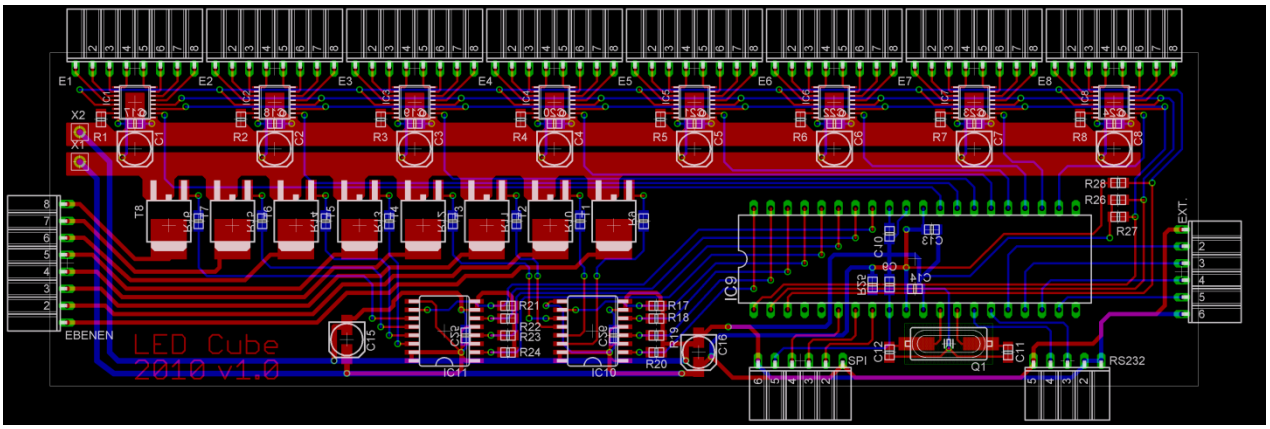


Bild 5: Layout der Steuerelektronik

2.5. LED-Matrix

Anhand einiger Bilder soll im Folgenden der Aufbau der LED-Matrix gezeigt werden. Dabei werden zunächst die acht Ebenen einzeln aufgebaut und diese anschließend zu dem Würfel zusammengesetzt.

Für eine gleichmäßige Struktur der einzelnen Ebenen ist es sinnvoll, vorab eine Schablone anzufertigen. Dazu kann beispielsweise eine handelsübliche Sperrholzplatte benutzt werden, in der im gewünschten Abstand, Löcher für die LEDs gebohrt werden.

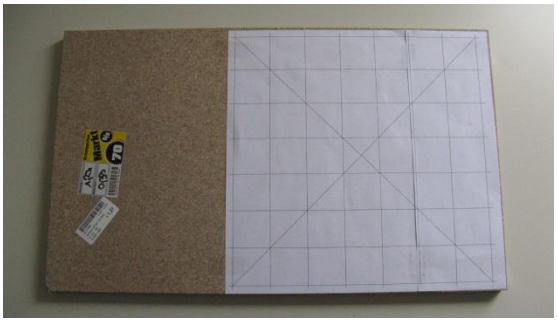


Bild 6: Schablone

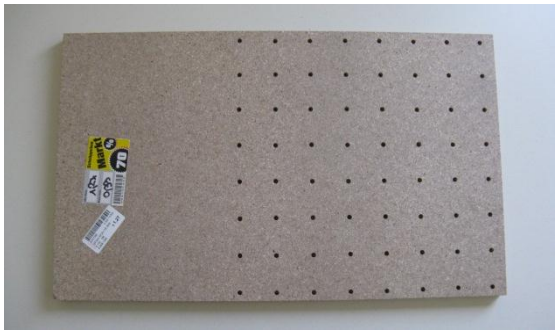


Bild 7: Fertige Schablone

Die einzelnen LEDs werden mit Silberdraht verschaltet. Dieser hat zum einen die Aufgabe des Stromtransports, als auch die komplette Konstruktion zu tragen. Für eine gewisse Steifigkeit wurde ein Drahtdurchmesser von 1 mm gewählt. Es ist darauf hinzuweisen, dass Silberdraht (vor allem an den Lötstellen) mit der Zeit etwas anlaufen kann. Da der Silberdraht aufgerollt in Rollen geliefert wird, sind vorab gerade Stäbe daraus herzustellen. Ich habe dazu den Draht auf die benötigte Länge gekürzt und anschließend mit einer Seite in einen Schraubstock gespannt. Durch kräftiges ziehen auf der anderen Seite kann der Draht dann gerichtet werden.

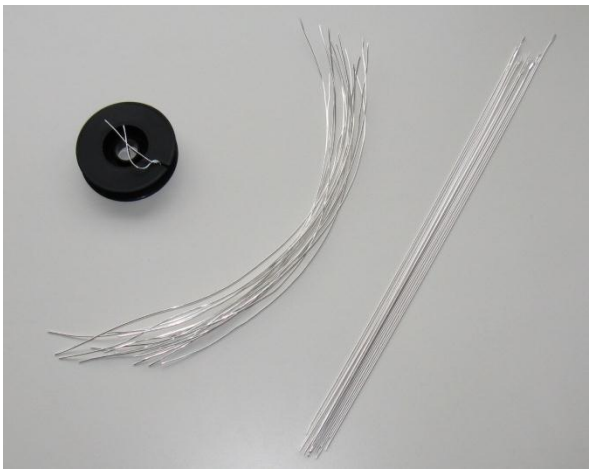


Bild 8: Silberdraht zur Verschaltung der LEDs (links: Rolle, mitte: gekürzt, rechts: gerichtet)

Bild 9: Materialen zum Aufbau der Ebenen

Die LEDs in die Schablone einsetzen und dabei auf die richtige Polarität achten. Die Anoden der LEDs jeder Reihe mit dem vorgefertigtem Draht verlöten (Bild 10). Die Kathoden um 90° in eine Richtung umbiegen. Im Anschluss zwei Querverbindungen einfügen, sodass nun alle Anoden der LEDs der Ebene miteinander verbunden sind (Bild 11).

Ob nun alle Anoden oder alle Kathoden der LEDs einer Ebene miteinander verbunden werden, hängt von den verwendeten Treiberbausteinen für die LEDs ab. Da die Ausgänge meiner verwendeten Treiber eine Stromsenke darstellen, müssen diese an die Kathode angeschlossen werden. Entsprechend sind dann die Anoden der LEDs einer Ebene miteinander zu verbinden, die später dann zur Selektion über einen Schalter an die positive Versorgungsspannung gelegt werden.

Bild 10: Verbindung der Anoden einer Reihe

Bild 11: Einfügen zweier Querverbindungen

Bevor die verlötete Ebene aus der Schablone entfernt wird, ist es ratsam eine Funktionsprüfung durchzuführen, um defekte oder falsch eingelötete LEDs zu entdecken. Um Frühausfälle einzelner LEDs vorzubeugen kann zusätzlich ein erweiterter Stresstest durchgeführt werden, bei dem die LEDs über einen längeren Zeitraum (zyklisch) betrieben werden.
Bild 12 Zeigt die fertige Ebene.

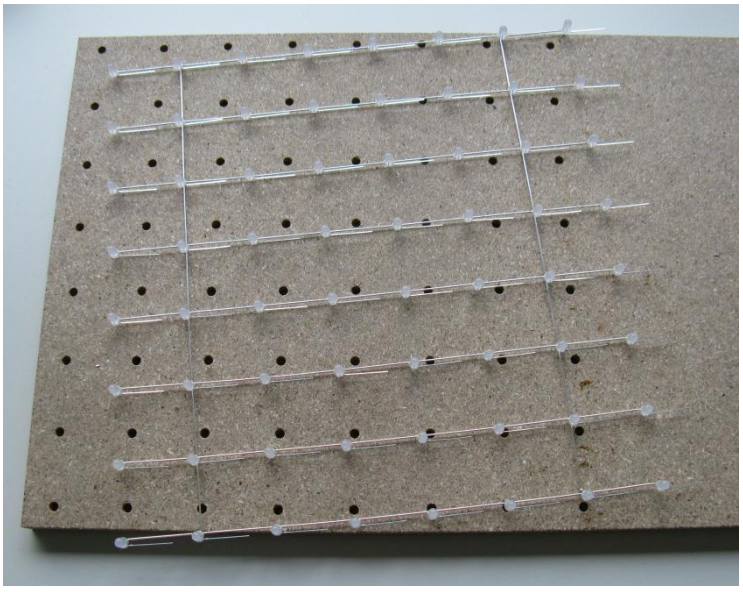


Bild 12: LED-Ebene

Sind alle acht Ebenen angefertigt, müssen diese zu einer würfelförmigen Matrix verschaltet werden. Dabei werden nun alle Kathoden übereinander liegender LEDs (Säule) miteinander verbunden. Dies geschieht ebenenweise. Gemäß Bild 13 werden an den Kathoden der LEDs der ersten (=obersten) Ebene wieder die vorgefertigten Stäbe aus Silberdraht angelötet.

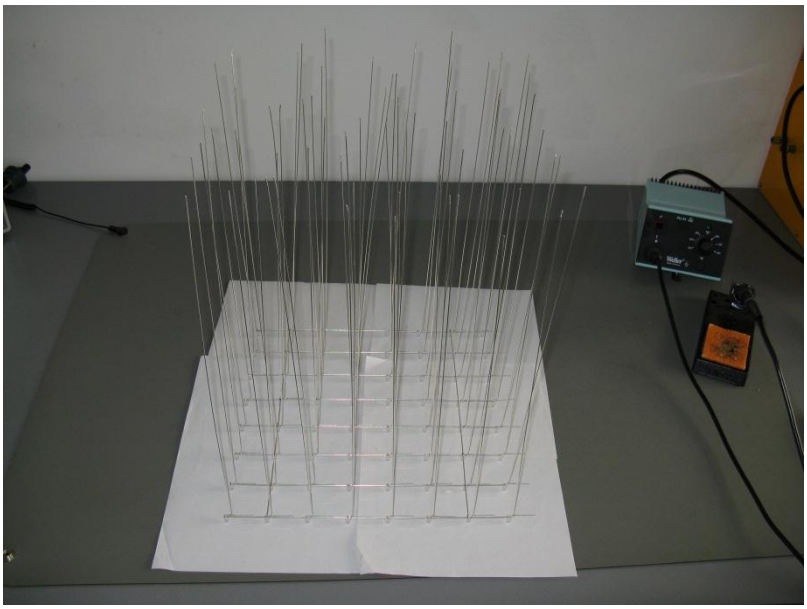


Bild 13: Aufbau der Matrix, erste Ebene

Die weiteren Ebenen sind dann von oben einzufädeln und die Kathoden der LEDs mit den Stäben zu verlöten. Vorgefertigte Hölzchen als Abstandhalter helfen, den richtigen Abstand der Ebenen einzuhalten. Nach jeder weiteren Ebene ist sicherheitshalber noch einmal die Funktionstüchtigkeit der LEDs zu testen, da ein späterer Austausch sich schwieriger gestaltet.

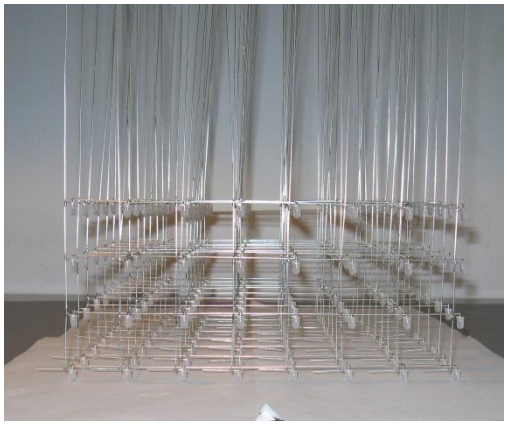


Bild 14: LED-Matrix 4 Ebenen

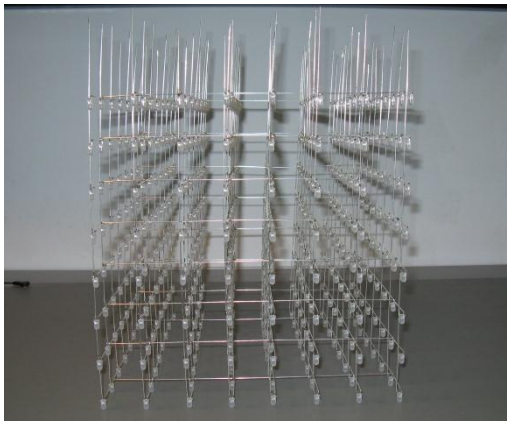


Bild 15: LED-Matrix 8 Ebenen

Die Stromzuführungen zu den gemeinsamen Anoden jeder Ebene sind noch nach unten zu führen. Die Wahl fiel hier auf Kupferlackdraht. Schrumpfschlauch fixiert die Drähte an einem der hinteren Silberdrähte.

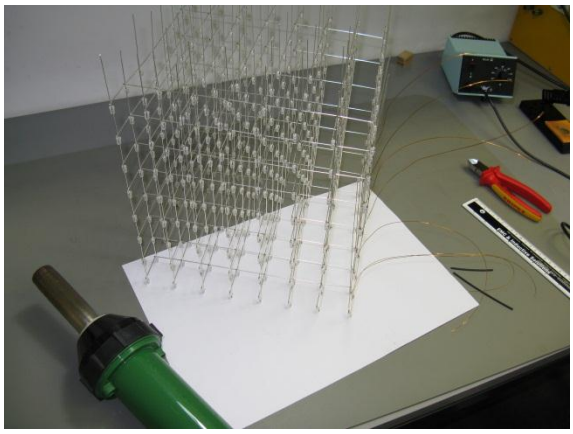


Bild 16: Anschluss der Ebenen

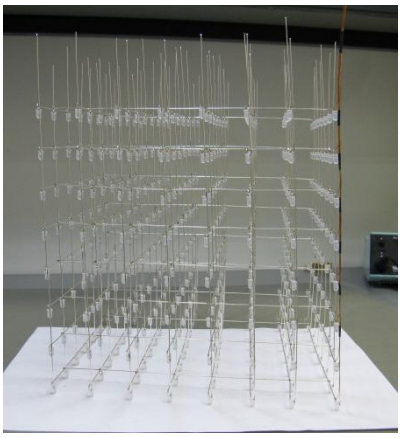


Bild 17: Fertige LED-Matrix

2.6. Sockel

Der fertige Cube braucht natürlich auch einen Sockel, auf dem die LED-Matrix befestigt wird und in dem die Steuerelektronik sowie die Verdrahtung untergebracht ist. Meine Materialwahl fiel auf Holz. Eine ansprechende Platte aus dem Baumarkt mit einer Dicke von ca. 18 mm wurde dazu auf ein Quadrat der Seitenlängen 32 cm passend zurechtgesägt. Zur Befestigung der LED-Matrix sind von oben Löcher gebohrt, die etwas größer als der Durchmesser der Silberdrähte der Säulen sind. Ein Bohrer mit 1,1 mm war in meinem Fall ausreichend. Von unten sind diese Löcher auf ca. 5 mm vergrößert, aber nicht komplett durchgebohrt. Zu beachten ist, dass die Säulen der Kathoden sich nicht direkt unterhalb der LEDs, sondern denen gegenüber leicht versetzt befinden. Zur mittigen Ausrichtung des Cubes auf dem Sockel ist dies zu berücksichtigen und die zugehörigen Positionen der Löcher sind dementsprechend zu berechnen. Ein Bild des Sockels von oben zeigt Bild 18. Zur Aufnahme der Verdrahtung und der Elektronik wurden mit Hilfe einer Oberfräse passende Kanäle und Aussparungen gefräst, zusätzlich seitlich dazu noch Aussparungen für Netzschalter, Netzstecker und einer Buchse für den Programmieradapter (Bild 19).

Als letzter Schritt wurde mein Sockel noch mit schwarzem Strukturlack lackiert.

Bild 18: Sockel von oben

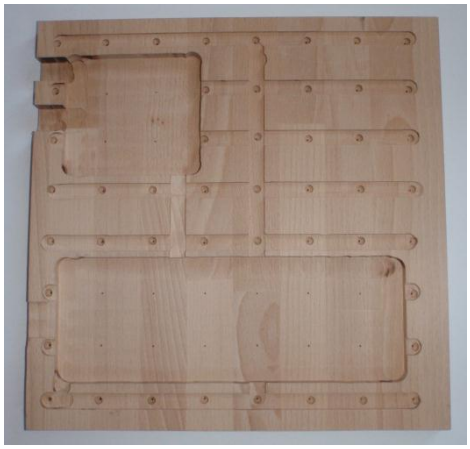


Bild 19: Sockel von unten

Bild 20: Fertiger, lackierter Sockel von oben

2.7. Aufbau der Steuerelektronik

Als erster Schritt müssen die Treiberbausteine MAX6968 aufgelötet werden. Die Bild 21 zeigt die gefertigte Leiterplatte und die ICs.

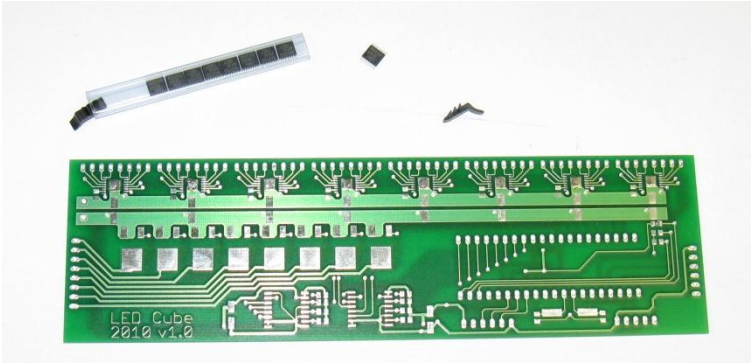


Bild 21: Leiterplatte und Treiberbausteine

Der MA6968 besitzt im TSSOP Gehäuse ein „exposed die pad“ (Groundplane) auf der Unterseite zur besseren Wärmeabführung, welches mit GND zu verbinden ist. Die ICs sind deshalb im Reflow Verfahren aufzulöten. Dazu wurde ein Reflow-Ofen benutzt. Einer der verlöteten Treiberbausteine bei der Kontrolle unter dem Mikroskop ist in Bild 22 und Bild 23 zu sehen.

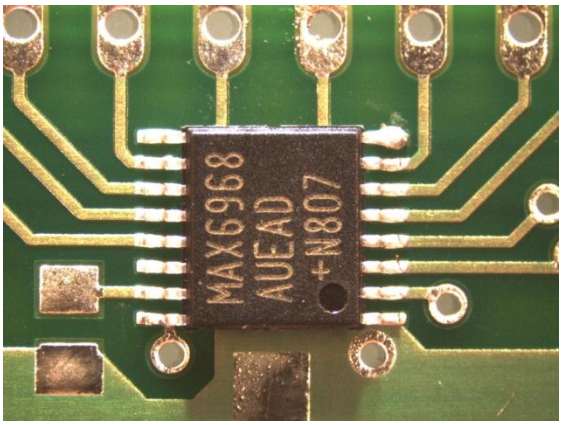


Bild 22: Treiber von oben

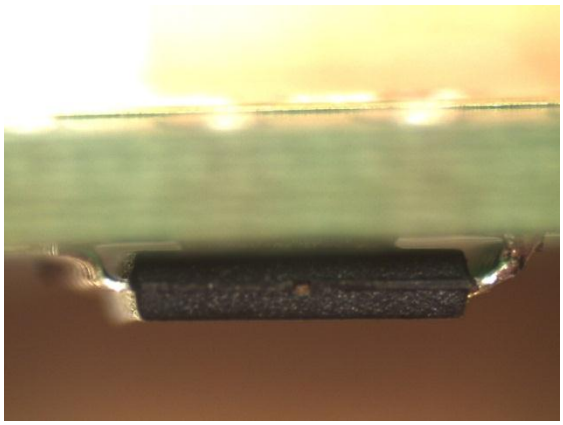


Bild 23: Treiber von der Seite

Die restlichen Bauteile wurden mit der Hand bestückt und verlötet (Bild 24). Das Ergebnis zeigt Bild 25 und Bild 26. Den ATmega32 gibt es auch im TQFP Gehäuse als SMD Version. Dadurch kann die Bauhöhe der Steuerung gegebenenfalls noch etwas verringert werden.

Bild 24: Material zum Aufbau der Steuerelektronik

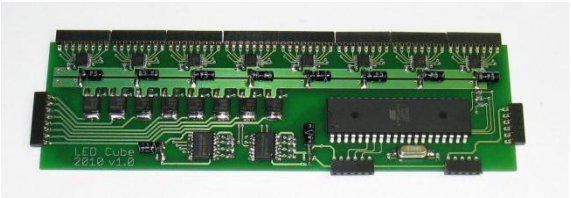


Bild 25: Steuerelektronik Oberseite

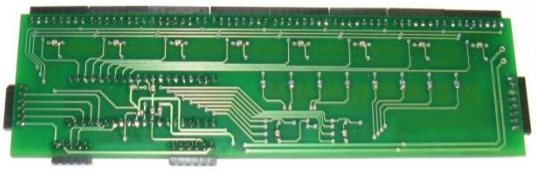


Bild 26: Steuerelektronik Unterseite

2.8. Montage

Einer der letzten Schritte bei der Hardware ist das Zusammenfügen von LED-Matrix, Sockel und Steuerelektronik. Dazu werden zunächst die senkrechten Stäbe der LED-Matrix in die Löcher des Sockels gefädelt. Dabei aufpassen, den Lack nicht zu zerkratzen. Sind alle Drähte in Position und die unterste Ebene im gewünschten Abstand zur Bodenplatte, dann das Gebilde umdrehen. Die Stäbe können nun mit Heißkleber in den 5 mm Löchern fixiert werden. An jeder Säule sind Leitungen zu Ansteuerung anzubringen. Meine Wahl fiel hier wieder auf Kupferlackdraht, der dann gebündelt in den gefrästen Kanälen zu verlegen ist. An den Enden der Kabel sind Stiftleisten angelötet, die später in die Buchsenleisten der Steuerelektronik eingesteckt werden. Bild 27 zeigt den Zwischenstand.

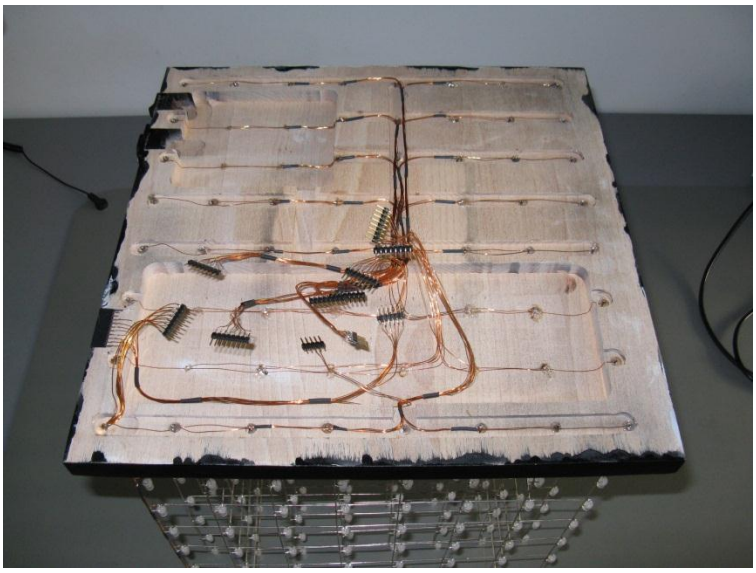


Bild 27: Verkabelung der LED-Matrix

Als letzter Schritt folgt der Einbau der Steuerelektronik. Auch diese ist mit jeweils einem Tropfen Heißkleber an den Ecken mit der Grundplatte verbunden. Zur Isolierung befindet sich zwischen der Leiterplatte und der Bodenplatte eine Isolierfolie.

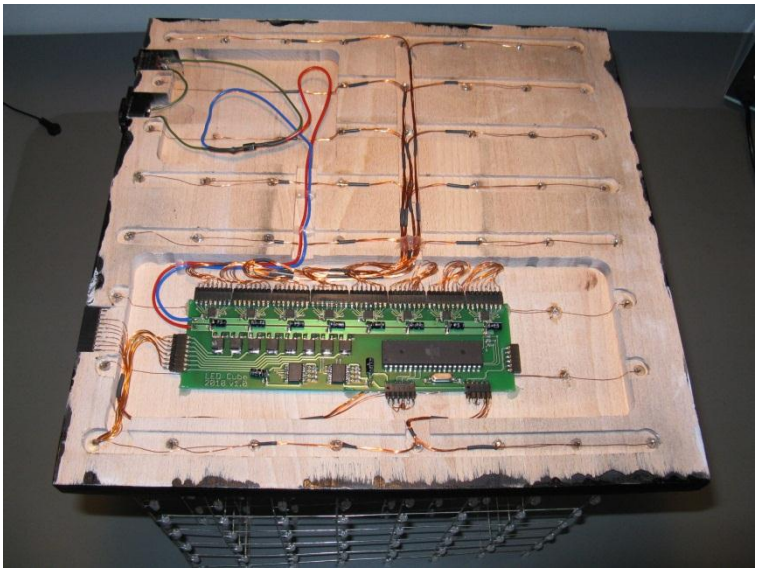


Bild 28: Verkabelung der LED-Matrix und Steuerelektronik

Dem Aufbau einer eigenen Stromversorgung ist ein extra Kapitel gewidmet worden. Eine Versorgung mit einem externen Netzgerät ist aber auch möglich.

2.9. Funktionstest

Zum Abschluss des Hardwarekapitels mit Bild 29 und Bild 30 zwei Oszilloskopbilder mit dem LED-Strom in einer Säule gemessen. Der Strom pro LED beträgt 32 mA, die Bildwiederholfrequenz 100 Hz.

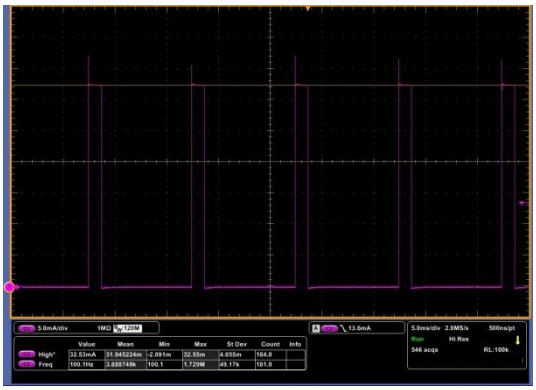


Bild 29: Strom einer Säule, eine LED an

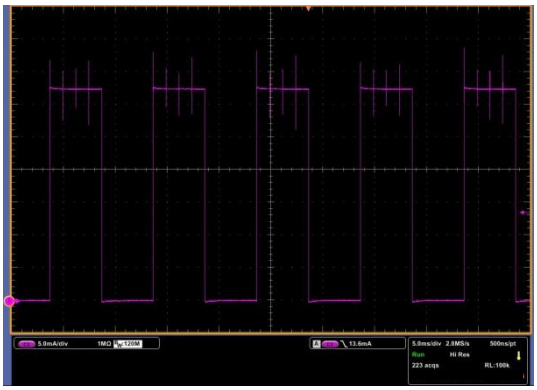


Bild 30: Strom einer Säule, vier LEDs an

3. Software

Die Software für den LED-Cube ist in *C* programmiert. Als Entwicklungsumgebung benutze ich AVR-Studio 5.

Es sei gleich darauf hingewiesen, dass dies mein erstes Projekt in *C* ist. Der von mir erstellte Code könnte also bestimmt noch effizienter oder ressourcenfreundlicher aufgebaut werden. Aber dennoch, er funktioniert, passt in den Flash und die Anforderung an die Rechenzeit pro Frame wird erfüllt.

Im Folgenden soll kurz die prinzipielle Funktionsweise meines Programms vorgestellt werden. Für genauere Details ist der (kommentierte) Source-Code heranzuziehen.

3.1. Abbild im µC

Der LED-Cube bzw. einen Frame davon kann man sich unterteilt in Zeilen, Spalten und Ebenen vorstellen. Dafür verwende ich die Variable `y` für die Nummerierung der Zeilen und `z` für die Ebenen, welche die Werte 0,1…7 annehmen können. Die Variable `x` repräsentiert eine Position innerhalb der Zeile bzw. eine Spalte. Wie man das Koordinatensystem in den Würfel legt hängt zum einen von der Softwareimplementierung, zum anderen von der Verschaltung der Steuerelektronik mit der LED-Matrix ab. Bild 31 zeigt eine Definition der Koordinatenachsen meines Würfels.

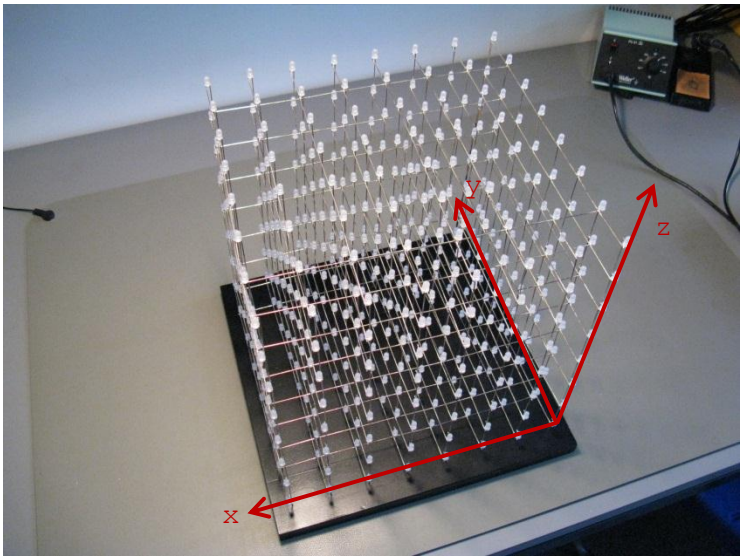


Bild 31: Koordinatenachsen

Bei meiner Routine befindet sich je ein Abbild des aktuellen Frames und des nächsten Frames im RAM des Mikrocontrollers. Das Abbild ist eine Art zwei-dimensionales 8x8 Array. Jeder Eintrag (=8bit Zahl) entspricht einer Zeile. Um auf die Daten im Speicher zugreifen zu können sind zwei Zeiger zu den Einträgen definiert:

```
volatile uint8_t Mem1 [8][8];                // Speicher reservieren für die Frames
volatile uint8_t Mem2 [8][8];                // Speicher reservieren für die Frames
volatile uint8_t *currentFrame = &Mem1[0][0]; // Zeiger für den aktuellen Frame
volatile uint8_t *nextFrame    = &Mem2[0][0]; // Zeiger für den nächsten Frame
```

Mit folgendem Befehl kann dann beispielsweise auf die Zeile des aktuellen Frames an der Position definiert durch die Variablen `y` und `z` zugegriffen werden:

```
xData = *(currentFrame+z*8+y);            // lesender Zugriff auf das Abbild
*(currentFrame+z*8+y) = xData;            // schreibender Zugriff auf das Abbild
```

3.2. Timing

- Bildwiederholfrequenz:
  Mit dem Multiplexen der einzelnen Ebenen, wird zu einem Zeitpunkt nur eine einzelne Ebene des Frames angezeigt. Die Bildwiederholfrequenz gibt an, wie oft der aktuelle Frame pro Sekunde komplett aufgebaut wird. (Vergleiche Bild 2.) Durch eine hinreichend hohe Bildwiederholfrequenz entsteht der Eindruck eines kompletten Bildes auf dem Cube. Meine Bildwiederholfrequenz liegt bei 100 Hz.

- Bildrate:
  Die Bildrate gibt an, wie viele (unterschiedliche) Frames pro Sekunde angezeigt werden. Durch eine hinreichend hohe Bildrate entsteht der Eindruck eines flüssigen Bewegungsablaufs. Meine Bildrate variiert leicht von Animation zu Animation.

Aufgabe des Mikrocontrollers ist es nun, die Daten des aktuellen Frames anzuzeigen und in der Zwischenzeit, den neuen Frame zu berechnen. Mit den zwei getrennten Speicherabbildern können unabhängig von der Berechnungsdauer keine fehlerhaften Frames kurzzeitig angezeigt werden. Zum anderen ist die Framerate konstant.

Für das zeitliche Timing zur Bildwiederholfrequenz und Bildrate sind zwei Timer zuständig. Timer 2 mit der Bildwiederholfrequenz ist für den ebenenweise Aufbau des Bildes in der entsprechenden Interrupt Routine verantwortlich. Er greift dabei auf das Abbild des aktuellen Frames zu. Da diese Interruptroutine mit der 8-fachen Bildwiederholfrequenz aufgerufen wird, sollte diese auf eine möglichst kurze Durchlaufzeit optimiert werden.

Timer 1 mit der Bildwiederholrate tauscht die Verweise auf die beiden Speicherabbilder. Der neue, bereits berechnet Frame wird dann als nächstes angezeigt und es wird mit der Berechnung des darauf folgenden Frames begonnen.

3.3. Animationen

In der main-Schleife werden die nächsten Frames der Animationen berechnet. Die Animationen sind dabei in extra Funktionen (in *Animations.c*) definiert. Diese greifen gegebenenfalls auf programmierte Transformationen (in *Transformations.c*) zurück. Transformationen sind einfache Abfolgen von Anweisungen, die häufig gebraucht werden. So z.B. das Verschieben oder Rotieren des Würfelinhalts, aber auch das Kopieren des vorhergehenden Frames in den aktuellen Frame. Die Transformationen sind so programmiert, dass sie als Quellabbild immer den aktuellen Frame (`currentFrame`) nehmen und als Zielabbild den nächsten Frame (`nextFrame`). Da bei einem Wechsel des Frames nur die Verweise auf die Speicherinhalte vertauscht werden, muss jede Animation entweder mit einer

Transformation beginnen oder den aktuellen Frame komplett neu aufbauen. Ist der nächste Frame einer Animation berechnet, wird die Funktion `waitForNextFrame()` aufgerufen. In einer Schleife wartet der Mikrocontroller jetzt, bis der nächste Frame angezeigt werden soll. Anschließend wird mit der Berechnung des darauffolgenden Frames der Animation begonnen. Die Zyklen einer Animation werden gegebenenfalls mehrfach wiederholt.

Framerate einstellen		
Wiederholung der Grundsequenz		
	Berechnung neuer Frames	
		Transformation
		Sequenz zur Berechnung des nächsten Frames
		`waitForNextFame()`

Bild 32: Struktogramm einer Animationsfunktion

In *Characters.c* sind Zahlen, Großbuchstaben und einige Zeichen zur Darstellung von Text auf dem Cube implementiert. Zwei Animationen können dazu Strings ausgeben. Zum einen ein Textband, welches um den Cube läuft, zum anderen Text, der buchstabenweise von hinten nach vorne durch den Cube wandert.

Insgesamt kann man bei der Programmierung der Animationen seiner Kreativität freien Lauf lassen. Anregungen dazu findet man auch im Internet, z.B. bei youtube.

3.4. Dimming

Die Möglichkeit zum Dimmen des Cubes über PWM wurde nachträglich noch hinzugefügt. Dazu wird der Timer0 des Mikrocontrollers benutzt, der hardwaremäßig den OC0-Pin entsprechend toggelt. Das PWM Signal ist dem „Output Enable“ Eingang des LED Treiber zugeführt.

4. Stromversorgung

Steckernetzteil ran und fertig. An dieser Stelle hätte das Project „LED cube“ abgeschlossen sein können. Dennoch wollte ich eine eigene, im Sockel integrierte Stromversorgung aufbauen. Sie hat die Aufgabe aus dem europäischen AC-Versorgungsnetz eine konstante DC-Spannung von 5 V zur Versorgung des Cubes zu generieren.
Aufgrund des Umfangs dieses eigentlich eigenständigen Projekts wird der Stromversorgung ein eigenes Dokument gewidmet.

Ein paar Eckdaten der Stromversorgung:

- Ausgangsspannung 5 V DC
- Maximale Ausgangsleistung 12 W
- Flyback im DCM mit Valley-switching/Valley-skipping
- Synchrongleichrichtung
- Bauhöhe 8 mm (10 mm mit Filter)
- Sichere Netztrennung nach DIN EN 60065, DIN EN 61558
- Leitungsgeführte Störungen nach DIN EN 55011
- Abgestrahlte Störungen nach DIN EN 55011
- Surgefestigkeit nach DIN EN 61000-4-5 (4 kV)

Bild 33 zeigt den aufgebauten Flyback. In Bild 34 ist er im eingebauten Zustand zu sehen.

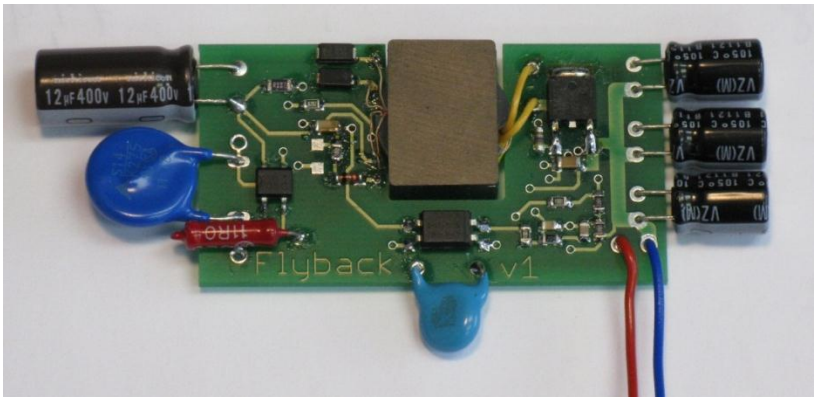


Bild 33: Flyback (Oberseite)

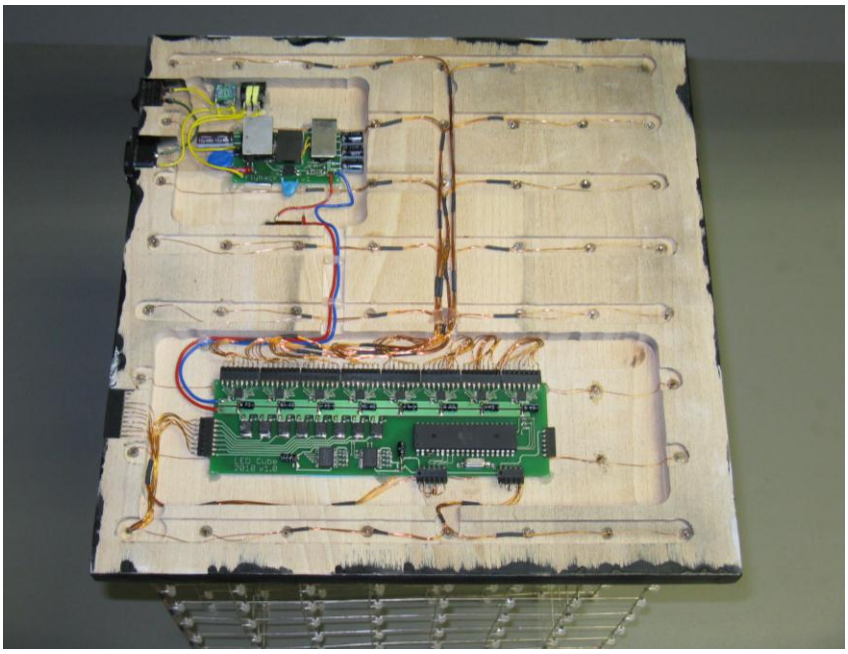


Bild 34: LED cube Unterseite

5. Weiterführende Literatur

http://www.mikrocontroller.net/articles/LED_cube
http://leyanda.de/light/ledcube.php
http://www.mikrocontroller.net/articles/LED-Fading
http://www.instructables.com/id/Led-Cube-8x8x8/