Lauflicht

Assignment zum Modul:

Mikrocomputer-Systeme (MCS40)

25. Mai 2019

Vladimir Zhelezarov

.....

Studiengang: Digital Engineering und angewandte Informatik - Bachelor of Engineering

AKAD University

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Ziel dieser Arbeit ist der Aufbau und die Programmierung einer LED-Lampensteuerung. Die Kontrolle erfolgt über einem ATMEGA328 Mikrocontroller. Dank dem Arduino-Projekt[1] sind für dieser Entwicklungsumgebung, Werkzeuge zum Hochladen der Software und Kommunikation leicht zugänglich. Diese werden wir benutzen, um ein schnellen Prototypenbau zu sichern.

Es ist allerdings keine Arduino-Hardware notwendig, um mit dem ATMEGA328 ein Projekt zu bauen. Die benötigten Komponenten vom Controller sind wenig und leicht beschaffbar. Damit bestücken wir eine eigene Platine und so die Probleme mit einem Steckbrett, wie Wackelkontakte oder Unordnung, vermeiden.

Zuerst klären wir was für LEDs wir steuern wollen, weil das direkt die LED-Treiber und die Spannungsversorgung bestimmt. Bestimmte Besonderheiten beim Ansteuern von LEDs sind auch zu beachten. Für die vorgestellte Lichtsteuerung bietet sich eine Konstantstromquelle an, als ein gutes Preis-/Leistungsverhältnis. Ihre Nachteile versuchen wir durch geeignete Auswahl von Komponenten und Stromversorgung zu minimieren.

Die Schaltsignale für die Treiber kommen aus dem Mikrocontroller. Mit mehreren LEDs werden die verfügbaren Ausgänge schnell knapp. Deshalb benutzen wir Schieberegister, um ihre Anzahl zu erhöhen.

Der Hauptansatz der Software für die Lichtsteuerung ist die Betrachtung der LEDs als ein binäres Wort. Die verschiedensten Effekte sind dann über Bitmanipulationen realisierbar. Wir versuchen möglichst viel Flexibilität anzubieten und sehen keine vorprogrammierten Muster vor. Diese können vom Benutzer nach seinen Wünschen eingestellt werden. Um so benutzerfreundlich wie möglich zu sein, muss die Steuerung auch schnell auf die Benutzereingaben reagieren und diese auch speichern können.

Dem Benutzer wird eine serielle Schnittstelle mit einem definiertem Protokoll für den Datenaustausch angeboten. Für diese Interaktion kann ein Seriell zum Bluetooth Modul benutzt werden, um mit einer App auf dem Smartphone zu kommunizieren. Als kleines Beispiel wird eine Android-App vorgestellt.

In der Software wird ein Gleichgewicht von Komplexität und Funktionalität gesucht.

[1] https://www.arduino.cc/

2 Grundlagen

2.1 Testschaltung

Als Testschaltung wird die Steuerung eine Signallampe mit 9x2 High-Power LEDs kontrollieren, also 18 LEDs paarweise in Reihe. Weil kein Datenblatt dafür verfügbar ist, wurden die benötigten Arbeitsparameter durch Tests und Messungen wie folgt ermittelt:

- Durchlassspannung: 4,5V
- Betriebsstrom: 250mA

2.2 Der ATMEGA328 Mikrocontroller

Im Herzen dieses Projektes steht der ATMEGA328 von Atmel - ein Mikrocontroller mit 8-Bit RISC Harvard Architektur[1].

Als Zwischenstelle um Software über die serielle Schnittstelle installieren zu können[2], wird auf den hier benutzten Controller Arduino-Bootloader aufgespielt[3].

Intern am Chip vom ATMEGA-Controller ist ein Speicher von 1024 Bytes als EEPROM angeboten, der für die dauerhafte Speicherung von Daten in kontrollierten Mengen geeignet ist[4].

2.3 Schieberegister

Um die Ausgänge vom Controller zu erweitern, benutzen wir Schieberegister. Im Prinzip ist ein Schieberegister ein Paar in Reihe geschalteten Flip-Flops[5]. So übernimmt er ein seriellen Wert an seinem Eingang und bildet diesen an seine Ausgänge parallel ab. Einen solchen Verlauf kann man wie folgt beschreiben[6]:

1. Befehl: Bereitstellen für Datenübertragung (storageClock-Pin negativ);
2. Wert einstellen (auf data-Pin);
3. Befehl: Wert ablesen (clock-Pin positiv, clock-Pin negativ);
4. evtl. 2. und 3. wiederholen;
5. Befehl: Wert übernehmen (storageClock-Pin positiv).

[1] Microchip Technology Inc. (2018), S.1
[2] Microchip Technology Inc. (2018), S.272
[3] https://www.arduino.cc/en/Hacking/Bootloader, (Zugriff am 19.05.2019)
[4] Microchip Technology Inc. (2018), S.16
[5] Horowitz, P./ Hill, W. (2017), S.744
[6] Vgl. Texas Instruments Incorporated (2015), S.13

Wenn das binäre Wort länger ist als das vom Schieberegister unterstützte, wird der Übertrag am Pin $Q_{H'}$ übergeben. So lassen sich Schieberegister nacheinander schalten um längere Wörter abzuarbeiten.

2.4 Ansatz der Steuerung: „Licht-Wort“

Eine Möglichkeit die einzelnen Ausgänge der Steuerung zu beachten, ist als eine binäre Zahl, wofür diese Arbeit den Begriff „Licht-Wort“ benutzt. Nach diesem Ansatz ist das einfachste Beispiel vom Lauflicht nur ein Schieben vom Licht-Wort nach links oder rechts, wobei der Übertrag zu der anderen Seite übernommen wird (circular shift / Ringschieberegister).

Grundsätzlich können die verschiedensten Muster für die Lichteffekte in zwei Kategorien unterteilt werden:

- Blinken-muster - entstehen durch wiederholtem Negieren vom binären Licht-Wort;
- Bewegung-muster - diese sind durch Schieben von Wörter und deren Kombinationen beschreibbar.

Durch geschickte Auswahl vom Licht-Wort und Muster entstehen zahlreiche Muster.

2.5 Stromversorgung von LEDs

Eine LED kann nicht direkt an der Versorgungsspannung angeschlossen werden. Es müssen ihre Arbeitsspannung, aber auch ihr Arbeitsstrom beachtet und kontrolliert werden. Die nicht-lineare Charakteristik von LEDs setzt eine zusätzliche Kontrolle des Stromes voraus. Oft benutzte Ansätze um den Strom in Grenzen zu halten sind[1]:

- Vorwiderstand - einfach und billig, aber sehr problematisch, weil schon bei einer kleinen Änderung der Spannung der Strom schon zu hoch werden kann. Der Unterschied zwischen der Leistung der LED und dem Netzteil wird als Wärme im Widerstand verbrannt.
- Konstantstromquelle - elektronische Schaltung, die automatisch den Strom relativ unabhängig von der Last und Versorgungsspannung konstant hält. Dabei entsteht Wärme nach dem selben Prinzip, wie bei dem Vorwiderstand. Allerdings ist die Stromregelung wesentlich stabiler.
- Schaltregler - wesentlich kompliziertere Schaltung, die so gut wie keine Leistung als Wärme verschwendet und eine noch stabilere Regelung anbietet.

[1] Vgl. Maxim Integrated Products, Inc. (2005), Internetquelle

2.6 Serielle Schnittstellen

Der ATMEGA verfügt über eine in der Hardware eingebauten seriellen Schnittstelle - die Pins 2 und 3, jeweils Rx und Tx. Wenn eine Schnittstelle nicht reicht, kann sie auch per Software auf beliebige Pins aufgebaut werden, auf Kosten der Geschwindigkeit[1]. Weil in diesem Projekt die Geschwindigkeit der seriellen Schnittstelle nicht besonders wichtig ist, ist dieser Nachteil akzeptabel.

3 Hardware

3.1 Mikrocontroller Schaltung

Der ATMEGA328 benötigt für seine Funktion manche externe Komponenten wie hier angezeigt:

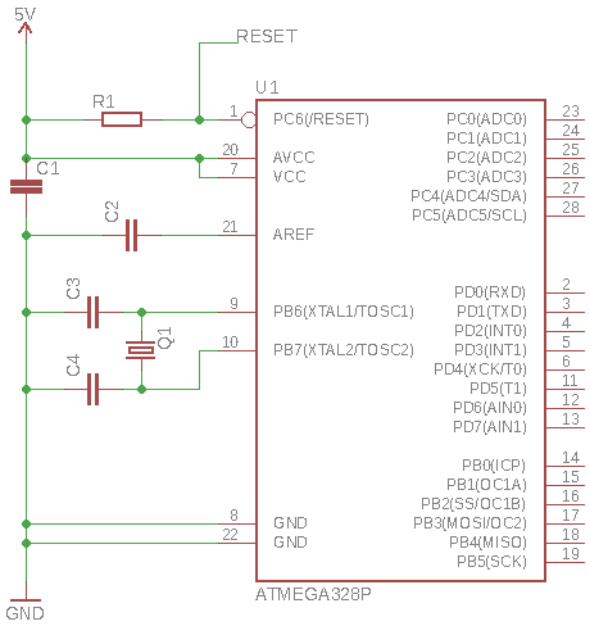

Abbildung 1: ATMEGA328 Schaltung

Der Pull-Up R1 bietet die Möglichkeit an, den Controller neuzustarten, wenn der Reset-Pin zum Grund gezogen wird[2]. C_1 und C_2 sind Entkopplungskondensatoren. Der Schwingungskreis aus C_3, C_4 und Q_1 bestimmt die Taktfrequenz, womit der Controller arbeitet.

Wir benutzen die typischen Werte C3 = C4 = 22pF, Q1 = 16MHz um mit 16MHz zu betreiben[3]. Diese Frequenz kann mit einer Versorgungsspannung von 5 Volt funktionieren[4].

[1] https://www.arduino.cc/en/Reference/SoftwareSerial, (Zugriff am 19.05.2019)
[2] Microchip Technology Inc. (2018), S.13
[3] Microchip Technology Inc. (2018), S.39
[4] Microchip Technology Inc. (2018), S.312

Pull-up: R1 = 10k, Entkopplungskondensatoren: C1 = C2 = 100n.

3.2 LED-Treiber Schaltung

Als Kompromiss zwischen Zuverlässigkeit und Komplexität benutzen wir die folgende Schaltung [1]:

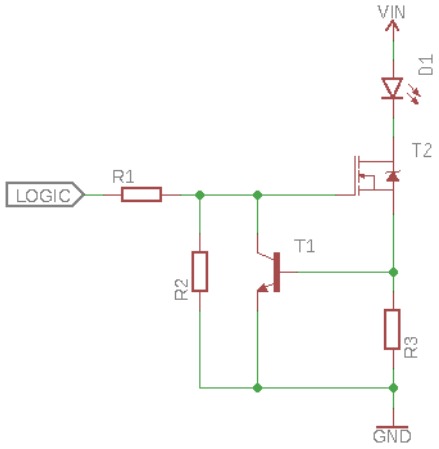

Abbildung 2: LED Treiber

Wenn am Logik-Eingang ein Signal anliegt, wird T_2 geschaltet. Dann fließt durch R_3 der selbe Strom wie durch die LED. Nach dem ohmschen Gesetzt verursacht dieser Strom ein Spannungsabfall:

$U = I_f / R_3$

Durch die Auswahl von R_3 kann kontrolliert werden, dass der T_1 öffnet und dadurch das Gate vom T_2 zum Grund zieht - als Folge leitet T_2 nicht mehr. So wird geregelt, dass der maximale Strom, der durch die LED fließt, nie I_f übersteigt. Widerstand für die Base wird nicht benötigt, weil sie höchstens mit 0,6Volt getrieben wird.

R_1 schützt von Gate-Kapazität Effekte vom MOSFET und R_2 ist Pull-Down der dafür sorgt, dass das Gate bei Abwesenheit des Logik-Signals nicht einen unbestimmten Zustand hat[2]. Weil der MOSFET mit Spannung gesteuert wird, können R_1 und R_2 groß sein - z.B. 10k und 100k. So fließt weniger Strom im Kollektor vom T1 und er kann kleiner (SMD) ausgewählt werden, wie die beiden Widerstände.

R_3 berechnet sich mit der Spannung, wobei T1 anfängt zu öffnen (0,6V), und dem Arbeitsstrom der LED (250mA)[3]. Also:

$R_3 = 0,6V / 0,25A = 2,4Ohm$

[1] In Anlehnung an Boughton, D. Jr. (2010), Design Idea 43, Figure 2
[2] Vgl. Horowitz, P./ Hill, W. (2017), S. 192ff
[3] Vgl. Horowitz, P./ Hill, W. (2017), S. 623

Wegen der Empfindlichkeit dieses Wertes ist für R_3 ein Widerstand mit weniger Toleranz zu wählen. An Leistung muss der Widerstand:

$P_{R3} = 0,6V * 0,25A = 0,15Watt$ mindestens verfügen.

Als T2 brauchen wir Logik-Level-Gate MOSFET, der V_{in} und die Leistung aushalten kann. Die Leistung ist wie folgt:

$P = (V_{in} - 0,6V).I_f = 0,75Watt$

Eine mögliche Auswahl wäre der IRL3705N. Weil die Verlustleistung (=Wärme) meistens in dem MOSFET abgeleitet wird, ist zu prüfen ob die Betriebstemperatur in den erlaubten Grenzen bleibt. Bei einer Umgebungstemperatur von 25° und $R_{\Theta JA} = 62°C/W$[1] folgt:

$T = 25° + (P * R_{\Theta JA}) = 71°C$

Das ist deutlich unter der $T_J = 175°C$, also in Ordnung, obwohl es sich auch schon warm anfühlt .

Es ist zu beachten, dass die berechnete Leistung und Temperatur im Dauerbetrieb des Verbrauchers sind. Die Leistung und dadurch die Temperatur können außerdem auf Kosten von etwas Lichtstärke geregelt werden, wenn R_3 größer gewählt wird.

3.3 Stromversorgung

Die benötigte Stromversorgung muss auf jeden Fall den Bedarf der LEDs und der Schaltung decken, darf aber nicht zu viel Spannung anbieten, weil wie gezeigt diese extra Leistung als Wärme verschwendet wird und andere Probleme verursachen kann - Bedarf an Kühlung, andere Arbeitsparameter der Schaltung wegen Erwärmung etc.

Strom:

9 LEDs x 0,25A = 2,25A

Die Bedürfnisse vom Controller und der Rest der Schaltung sind im mA-Bereich[2].

Spannung:

LED: 4,5V

R_3: 0,6V

Das verfügbare Netzteil mit den nächstliegenden Parameter ist 7,5V 5A.

4 Software

4.1 Einstellung-Makros

Damit die Software maximale Flexibilität anbieten kann, sind die wichtigsten Parameter über Makros einstellbar. Dazu gehören unter anderem:

[1] Infineon Technologies AG (2018), S.1

[2] Microchip Technology Inc. (2018), S. 597ff

1. LEDCOUNT - Anzahl der Ausgänge. Das ist im Grunde die Länge des Licht-Wortes. Der Wert wird auch bei allen weiteren Berechnungen und Bitmanipulationen benutzt.

- MINDELAY, MAXDELAY - die minimal/maximal erlaubte Intervalllänge;
- EE_ Makros - diese bestimmen die Positionen im EEPROM, wo Daten gespeichert werden;
- COMM_ und CMD_ Makros - Parameter des Protokolls für die Benutzerinteraktion.

4.2 Status-Variablen

Die *stateStruct* Variable *state* speichert alle Parameter, die den aktuellen oder den ursprünglichen Status beschreiben. Geändert werden diese durch Benutzereingaben oder im Programmverlauf. Damit der vom Benutzer zuletzt ausgewählte Status nicht nach dem Neustart verloren geht, wird diese Struktur laufend im EEPROM gespeichert.

- *sPattern* : das ist das Licht-Wort;
- *sShiftType* : Schiebeart: links, rechts oder nur blinken;
- *sShift* : Schiebeweite;
- *sDelay* : der Zeitintervall zwischen der Iterationen.

4.3 Speicherung vom Muster

Nach dem Neustart werden die Status-Variablen vom EEPROM geladen. Jede Änderung wird auch dort gespeichert. Weil im EEPROM die Schreibzyklen begrenzt sind[1], ist es besser die Werte der Variablen mit dem Gespeichertem erst zu vergleichen und dann nur die geänderte Variablen zu überschreiben.

Der EEPROM ist als Bytes verteilt. Weil die *int* Zahlen zwei Bytes benötigen, verteilt sie die Funktion *saveState* auf zwei Bytes durch Maskieren. Der selbe Ansatz benutzt auch *initState* zum Ablesen.

4.4 *workOnPattern* - Bitmusterbearbeitung

Diese Funktion führt die tatsächlichen Änderungen an den Licht-Wörtern durch. Die möglichen Szenarien sind:

- Blinken : die Funktion gibt die negierte Zahl zurück. Dadurch entsteht Blinken;

[1] Microchip Technology Inc. (2018), S. 29

- Schieben mit Rotieren nach links:

  1. ~(ALLON >> shift) : bereitet eine Maske für die *shift* am weitesten links stehenden Bits vor;
  2. (n & ~(ALLON >> shift)) : wendet die Maske an;
  3. >> (LEDCOUNT - shift) : Die im letzten Schritt ausgewählten Bits werden auf der anderen Seite (rechts) gespiegelt. Das sind praktisch die übertragenen Bits;
  4. ((n << shift) & ALLON) : Die ganze Zahl wird nach links geschoben und der Übertrag wird durch die Maske verworfen;
  5. Mit dem Bitweise-Oder (|) werden dann die in den Schritten 3. und 4. erzeugten Zahlen kombiniert.

- Schieben mit Rotieren nach rechts. Verläuft nach dem selben Prinzip wie links:

  1. ((n & (ALLON >> (LEDCOUNT - shift))) << (LEDCOUNT - shift)) : der Überlauf nach rechts wird berechnet und auf links gespiegelt;
  2. (n >> shift) : die Zahl wird nach rechts geschoben;
  3. Mit Oder (|) werden 1. und 2. kombiniert.

4.5 „Wach“ Warten

Die Ereignisse vom Lauflicht passieren nicht mit der Geschwindigkeit des Prozessors. Dafür wird immer wieder Wartezeit benötigt, bevor die nächste Änderung in Kraft tritt.

Es kann aber sehr leicht vorkommen, dass der Benutzer einen neuen Wunsch hat, während das vorhandene Muster ausgeführt wird. Wenn der Code erst nach dem Ablauf des Musters nach Benutzereingaben prüft und vor allem mit längerer Intervallzeit kombiniert, würde sich die Software als schwer reagierend anfühlen.

Das vermeidet die hier angebotene Software mit der wiederholten Überprüfung von zwei Schlussbedingungen in einer endlosen Schleife:

1. Liegt Benutzereingabe an?
2. Ist die Zeit für das aktuelle Intervall schon abgelaufen?

Antworten mit „Ja“ auf einer der beiden Fragen führt zum Abbruch der Schleife und weiterer Bearbeitung der Benutzereingabe oder des Programmfortschrittes.

4.6 Schnittstelle zum Benutzer

Die Software setzt voraus, dass alle Daten 1-Byte sind, bis auf Licht-Wort- und Intervall-, die 2-Byte sind. Synchronisiert mit der Benutzersoftware wird durch eine feste Länge des Protokolls und festgelegte Anfangs- und Endzeichen. Somit ist eine gewisse Sicherheit gegen Fehlbefehle oder Störungen auf dem Kommunikationsmedium gewährleistet. Allerdings sichert aber dieser Ansatz nicht von geänderten Daten im Körper des Protokolls.

Es können folgende Wünsche vom Benutzer erfüllt werden:

- ein neues Muster einstellen;
- die Geschwindigkeit vom vorhandenen Muster ändern.

Es lassen sich natürlich auch andere definieren, wie z.B. Richtung ändern. Um den Code nicht komplizierter als nötig zu machen, werden solche Funktionalitäten der Benutzersoftware (z.B. Smartphone-App) überlassen.

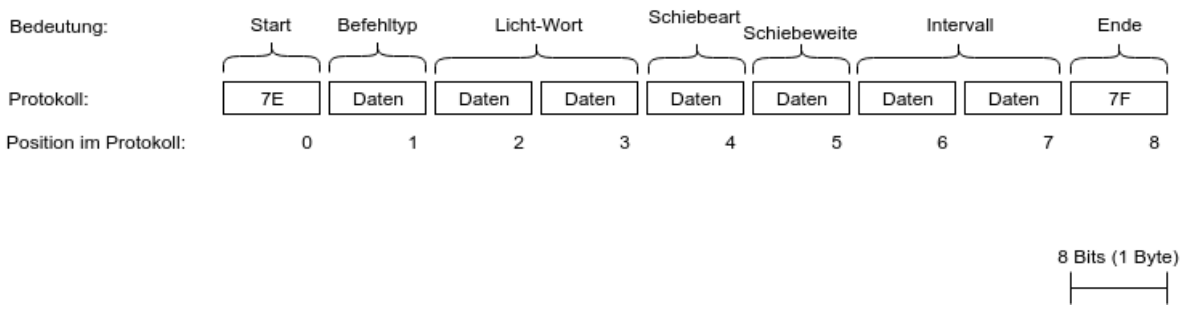

Abbildung 3: Kommunikationsprotokoll: Aufbau

Unser Paradenbeispiel mit dem Lauflicht sieht dann so aus:

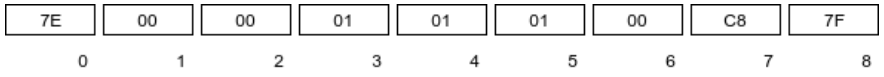

Abbildung 4: Kommunikationsprotokoll: Einfaches Lauflicht

Die Interpretation davon ist:

- Pos. 0: Start Symbol;
- Pos. 1: Befehlstyp - „set“, also neues Muster eingeben;
- Pos. 2 und 3: Das Licht-Wort;
- Pos. 4: Schieben nach links;

- Pos. 5: Schiebeschritt 1;
- Pos. 6 und 7: Wartezeit von 200ms.

4.7 Smartphone-App

Der Sockel für die serielle Schnittstelle ist mit dem leicht zugänglichen Bluetooth-Module HC-06 (angeboten von dem chinesischen Hersteller Guangzhou HC Information Technology Co., Ltd.[1])kompatibel. Somit kann der Benutzer gemütlich die Lichteffekte von seinem Handy aus steuern.

Die ausführliche Beschreibung einer Android oder iOS App liegt außerhalb dem Umfang dieser Arbeit. Als kleines Beispiel, ohne Anspruch auf Vollständigkeit, ist nur eine Android-App angeboten mit einem Paar vorprogrammierten Lichteffekten angeboten. Die App basiert auf dem „Bluetooth-Chat“ Mustercode, der von Google über eine Apache-Lizenz angeboten wird[2].

Der Hauptansatz der App ist das Versehen aller Buttons mit *Tags*[3]. Jeder *Tag* besteht aus einem Befehl für die Lichtsteuerung. Die in der App verfügbaren Effekte sind in der Klasse *Macros.java* eingestellt. Beim Aufbau des Chat-Fragments werden alle Buttons mit einem *onClickListener*[4] versehen, der beim Aufruf den *Tag* des Buttons als Befehl über die Bluetooth Verbindung schickt.

[1] http://www.wavesen.com/, (Zugriff am 19.05.2019)
[2] https://github.com/googlesamples/android-BluetoothChat, (Zugriff am 19.05.2019)
[3] https://developer.android.com/reference/android/view/View.html#tags, (Zugriff am 19.05.2019)
[4] https://developer.android.com/reference/android/view/View.OnClickListener?hl=en, (Zugriff am 19.05.2019)

5 Zusammenfassung

Obwohl die Steuerung von Power-LEDs anspruchsvoll ist, haben wir eine Schaltung entwickelt, die mit Anforderungen zurecht kommt, ohne dabei viel komplizierter oder teurer zu werden. Die Schaltung bietet eine Konstantstromquelle an, die relativ stabil und zuverlässig ist. Das Arbeitsprinzip basiert auf der fixierten Basenspannung eines bipolaren Transistors, bei der er öffnet. Kombiniert mit einem präzise eingestellten Widerstand, führt das zu konstantem Strom durch die LED, auch bei Variationen in der Versorgungsspannung.

Als geeignete Wahl des Mikrocontrollers hat sich ATMEGA328 angeboten, nicht zuletzt dank der Arduino Entwicklungsumgebung und der Werkzeuge für den schnellen Prototypenbau.

Der vorgestellte Ansatz für die LED-Steuerung ist die Betrachtung der Ausgänge als ein binäres Wort und seine entsprechenden Manipulationen.

Dem Benutzer ist eine Schnittstelle angeboten, mit zahlreichen Möglichkeiten für Licht- und Bewegungsmuster. Das Programm verläuft ohne Blockierungen um schnelle Reaktionszeiten auf den Benutzereingaben anzubieten. Aktuelle Muster speichert die Steuerung im internen Speicher vom Controller.

Aufgrund dem begrenzten Umfang dieser Arbeit wurde nicht näher an der Benutzersoftware angegangen, wo z.B. eine Smartphone-App sich als sehr geeignet zeigt. Die Kommunikation mit dem Benutzer ist sehr eingeschränkt und könnte z.B. mit Bestätigungen erweitert werden, wenn eine Nachricht angekommen ist. Die korrekte Lieferung wird auch grob kontrolliert - nach Länge und den Start-/End- Symbolen. Geänderte Daten im Hauptteil der Nachricht werden gar nicht erkannt. Eine mögliche Lösung wäre das Stopp-Symbol durch eine Prüfsumme zu ersetzen. Abstriche haben wir auch bei der Qualität der Platine gemacht - wenn diese professionell hergestellt wird, kann sie auch kleiner und auch optisch besser aussehend incl. Beschriftungen sein. Der Programmierungssockel ist etwas chaotisch und kann besser mit einem serienmäßigen ISP-Sockel ersetzt werden, inklusive dem Umzug von serieller Programmierung mithilfe des Arduino-Bootloaders zum ISP Anschluss. Das kann auch die Möglichkeit bieten, ein Bootloader auf den Controller direkt auf der Platine aufzuspielen, oder gar komplett auf Bootloader zu verzichten und den Controller direkt zu programmieren.

Der Controller benutzt *digitalWrite* um die Schieberegister zu steuern. Die ausgewählten Pins verfügen aber auch über eine Hardware-Unterstützung von SPI[1]. Damit kann die Kommunikation wesentlich schneller passieren und evtl. die Möglichkeit von PWM-Kontrolle[2] der Leuchtstärke anbieten.

[1] https://www.arduino.cc/en/Reference/SPI, (Zugriff am 19.05.2019)

[2] https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/, (Zugriff am 19.05.2019)

Literaturverzeichnis

Microchip Technology Inc. (2018)
megaAVR Data Sheet,
http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf (Zugriff am 19.05.2019)

Horowitz, P./ Hill, W. (2017)
The art of Electronics Third Edition, New York

Texas Instruments Incorporated (2015)
SNx4HC595 8-Bit Shift Registers With 3-State Output Registers,
https://www.ti.com/lit/ds/symlink/sn54hc595.pdf (Zugriff am 19.05.2019)

Boughton, D. Jr. (2010)
Circuit achieves constant current over wide range of terminal voltages, in: The EDN Network: Design Ideas, Design Idea 43
https://m.eet.com/media/1154796/25696-circuit_achieves_constant_current_over_wide_range_of
(Zugriff am 19.05.2019)

Maxim Integrated Products, Inc. (2005)
High-Efficiency Current Drive for High-Brightness LEDs,
https://www.maximintegrated.com/en/app-notes/index.mvp/id/3668 (Zugriff am 19.05.2019)

Infineon Technologies AG (2018)
IRL3705NPbF,
https://www.infineon.com/dgdl/irl3705npbf.pdf?fileId=5546d462533600a40153565f31232534
(Zugriff am 19.05.2019)

Arduino (o.J.)
Language Reference,
https://www.arduino.cc/reference/en/ (Zugriff am 19.05.2019)

Google Inc. (o.J.)
API reference,
https://developer.android.com/reference (Zugriff am 19.05.2019)