Hochschule Bochum
Bochum University
of Applied Sciences

Projektarbeit

RGB-Beleuchtung eines Kombiinstrumentes

von

Tobias Brächter

014215532

Studiengang: Master of Engineering Mechatronik und Informationstechnologie
Modul: Treiberentwicklung, Echtzeit- und Betriebssysteme
Dozent: Peter Gerwinski

Inhalt

1 Einleitung

Das Hauptziel dieses Projektes war es, die Beleuchtung eines Kombiinstrumentes farblich veränderlich zu gestalten, wobei die Veränderung der Farben in Abhängigkeit der Motordrehzahl stattfinden sollte. Zu diesem Zweck sollte ein einfacher Mikrocontroller verwendet werden, der die Motordrehzahl ermittelt und in Abhängigkeit davon RGB-LEDs ansteuert. Zusätzlich soll über das System die Freigabe eines Starterknopfes realisiert werden, um eine versehentliche Betätigung bei laufendem Motor zu verhindern. Neben dem entstandenen System sollte das Projekt dazu dienen, verschiedene Disziplinen der Programmierung zu erlernen, die der Autor bis dahin nicht beherrschte.

In den folgenden Abschnitten wird zunächst das Konzept vorgestellt, gefolgt von einigen Grundlagen, die zum Verständnis des Projektes benötigt werden. Anschließend werden die einzelnen Teilaspekte des Projektes im Detail erläutert. Zum Schluss steht eine Zusammenfassung mit Ausblick.

2 Anforderungen und Konzept

Die Kernkomponente des Systems ist die Hauptplatine mit Mikrocontroller, welche die Steuerung aller Funktionen übernimmt und im Folgenden **Controller** genannt wird. Der Controller soll folgende Aufgaben übernehmen:

- Messung der Motordrehzahl
- Ansteuerung von RGB-LEDs entsprechend eines Datensatzes
- Erteilen der Starterfreigabe
- Ansteuerung des Starters
- Einstellung des Datensatzes mittels serieller Kommunikation

Die folgende Abbildung soll eine Übersicht über die relevanten Komponenten geben:

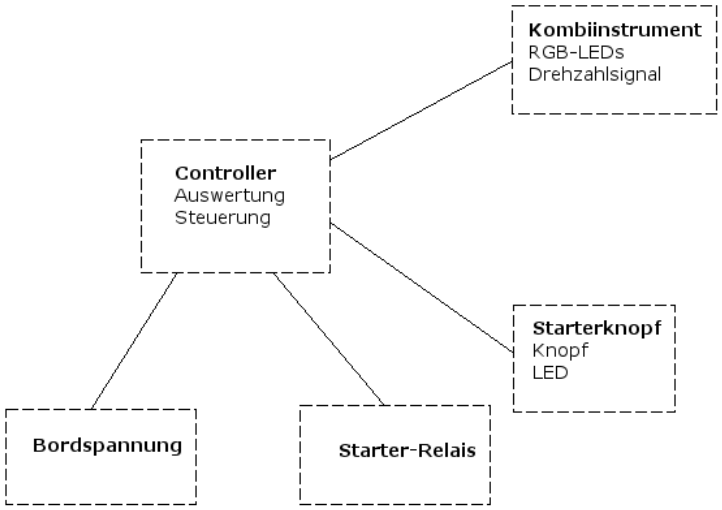

Abbildung 1 Übersicht Konzept

Wie man der Abbildung entnehmen kann, muss der Controller mit vier anderen Komponenten verbunden werden. Das Kombiinstrument beinhaltet die RGB-LEDs, welche vom Controller angesteuert werden müssen. Da das Kombiinstrument über einen Drehzahlmesser und somit auch das dafür benötigte Signal verfügt, kann in der Verbindung zwischen Kombiinstrument und Controller neben der Ansteuerungssignale für die LEDs auch das Drehzahlsignal übermittelt werden. Der Starterknopf dient dazu, den Starter zu betätigen. Eine integrierte LED signalisiert dabei die Freigabe des Knopfes. Da zum Ansteuern des Starters ein recht hoher Strom benötigt wird (das Kabel verfügt über einen Querschnitt von $4mm^2$, vgl. [HEI], S. 2f), wird zur Leistungswandlung ein Relais verwendet. Das Starter-Relais wiederum wird vom Controller angesteuert. Die Bordspannung wird vom zentralen Sicherungskasten entnommen, wobei auf dem Controller eine Anpassung der Spannung vorgenommen werden muss, um die Anforderungen des Mikrocontrollers zu erfüllen.

Nicht im Schaubild zu sehen ist die serielle Schnittstelle, mit dessen Hilfe der Datensatz im Controller angepasst werden kann. Der Grund dafür ist, dass es sich dabei um eine temporäre Verbindung handelt, während das Schaubild den dauerhaften Betrieb darstellen soll.

Neben der vorgestellten Hardware sind im Rahmen des Projektes drei Software-Lösungen entstanden, und zwar die Software des Controllers selbst, ein Interface für den Computer zur Manipulation und Übertragung des Datensatzes, sowie ein Frequenzgenerator zum Testen des Systems.

An das Konzept werden weiterhin einige Anforderungen gestellt. So sind z.B. alle Verbindungen steckbar ausgeführt, damit die Komponenten einzeln ausgebaut werden können, wenn dies nötig werden sollte. Ferner sind die verwendeten Kabelwege im Fahrzeug zum Teil sehr eng, sodass es schwierig bis unmöglich werden kann, alle Komponenten mit festen Kabelverbindungen an ihren Montageort zu bewegen. Außerdem soll es für das System nicht möglich sein, Einfluss auf die sonstigen Systeme des Fahrzeugs zu nehmen, sodass zum Beispiel der Starter nicht unerwartet durch einen Fehler im Controller betätigt werden darf. Weiterhin soll es wie bereits erwähnt möglich sein, dass der für die Funktion zu Grund liegende Datensatz austauschbar ist, dauerhaft gespeichert wird und beim Start des Systems automatisch geladen wird. Ein qualitativer Aspekt, der erfüllt werden soll, ist, dass die Veränderung der Farbgebung „hochwertig“ ist. Darunter wird unter anderem verstanden, dass keine plötzlichen Änderungen auftauchen oder wahrnehmbares Flackern festzustellen ist.

Damit sind Konzept und Anforderungen beschrieben und es folgen einige Grundlagen.

3 Grundlagen

3.1 Mikrocontroller

Der Mikrocontroller ist im Kontext zusammen mit dem Mikroprozessor und dem Mikrocomputer zu betrachten. Bei dem Mikroprozessor handelt es um einen Chip, auf dem in der Regel ein Steuerwerk und ein Rechenwerk untergebracht sind, ebenso wie Schnittstellen zur Ansteuerung. Seine Bestimmung ist es, ein Programm abzuarbeiten, welches aus Befehlen besteht, die im Befehlssatz des Mikroprozessors definiert wurden. Es können auch weitere Bestandteile hinzukommen, wie z.B. ein lokaler Cache-Speicher für schnellen Speicherzugriff. Ein Mikrocomputer dagegen ist ein System, das einen oder mehrere Mikroprozessoren enthält und darüber hinaus Speicher und verschiedene Ein-/Ausgabeschnittstellen, sowie ein Verbindungssystem, um all diese Komponenten zu verbinden. Der Mikrocontroller implementiert einen Mikrocomputer auf einem einzigen Chip. Mikrocontroller werden in der Regel für kleine Steueraufgaben verwendet, bei denen möglichst wenige verschiedene Bauteile eingesetzt werden sollen, was dadurch erreicht wird, dass viele der benötigten Funktionen bereits im Mikrocontroller implementiert sind (vgl. [BRI02] S. 1).

Der in diesem Projekt verwendete Mikrocontroller ist ein Atmel ATmega8, welcher mit einer Taktfrequenz von 8 MHz betrieben wird und unter anderem über einige digitale Ein- und Ausgänge verfügt, ebenso wie einen Analog-Digital-Wandler, Hardware-Timer und im Programm nutzbaren EEPROM (vgl. [ATM13], S. 1).

3.2 UART

Der Universal Asynchronous serial Receiver and Transmitter ist ein auf Mikrocontrollern häufig anzutreffender Baustein, mit dessen Hilfe man eine serielle Kommunikation zu anderen Mikrocontrollern oder UART-fähigen Geräten herstellen kann. Hardwareseitig existieren ein RX- und ein TX-Anschluss, welche über Kreuz mit dem anderen Kommunikationspartner verbunden werden. Am RX-Anschluss empfängt der jeweilige Kommunikationspartner Daten, auf dem TX-Anschluss sendet er. UART arbeitet symbolorientiert und transportiert in einer Nachricht immer nur ein Symbol, welches je nach Konfiguration zwischen 5 und 9 Bits lang ist. Im Leerlauf liegt auf der Leitung immer ein High-Pegel an, jede Nachricht beginnt mit einem Wechsel auf Low (Start-Bit). Daraufhin folgen die 5 bis 9 Datenbits. Zum Abschluss folgt je nach Konfiguration ein Paritätsbit zur Datenkontrolle sowie 1 bis 2 Stop-Bits. Bei UART müssen bei beiden Kommunikationspartnern sowohl das Format der Nachricht, als auch die Baudrate eingestellt werden. Dadurch, dass beiden Kommunikationspartnern die Baudrate und das Nachrichtenformat bekannt sind, reicht eine Synchronisation durch das Start-Bit aus, um die Daten zu übertragen.

St	Start bit, always low
(n)	Data bits (0 to 8)
P	Parity bit. Can be odd or even
Sp	Stop bit, always high
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high

Abbildung 2 Schaubild: UART-Nachricht (Quelle: [ATM13], S. 133)

3.3 SPI

Das Serial Peripheral Interface ist ebenso wie UART ein Baustein zur seriellen Kommunikation. Der erste Unterschied ist, dass SPI 4 statt 2 Anschlüsse verwendet: MISO (Master in, Slave out), MOSI (Master out, Slave in), SCK (Clock) und SS (Slave Select). MISO und MOSI sind die Äquivalente zu RX und TX. Die Synchronisation erfolgt bei SPI über eine separate Taktleitung (SCK). Im Gegensatz zu UART ist bei SPI vorgesehen, dass mehrere Slaves mit einem Master verbunden werden. Jeder Slave wird dabei separat über den Slave Select vom Master angesteuert, sodass immer nur der Slave Daten sendet und empfängt, der vom Master angesteuert wird. Dies soll nur eine kurze, unvollständige Übersicht darstellen, für nähere Informationen ist z.B. [ATM13], S. 121-128 zu lesen.

SPI wird häufig und auch in diesem Projekt verwendet, um das Programm auf den Mikrocontroller zu übertragen.

3.4 Interrupt

Geht man von einem in C geschriebenen Programm aus, so wird beim Starten des Mikrocontrollers die Funktion *int main(void)* (im Folgenden **Hauptprogramm** genannt) aufgerufen. Nachdem diese Funktion erfolgreich ausgeführt wurde, gilt das Programm als beendet und das weitere Verhalten des Mikrocontrollers ist undefiniert. Aus diesem Grund wird im Allgemeinen so vorgegangen, dass im Hauptprogramm in einer Endlosschleife abwechselnd verschiedene Funktionen aufgerufen werden, je nach Zustand des Systems. Auf diese Weise wird das Hauptprogramm niemals vollständig ausgeführt und der Zustand des Mikrocontrollers bleibt definiert. Ein Interrupt ist nun eine Unterbrechung des Hauptprogramms: Trifft ein bestimmtes Hardware-Ereignis ein, für welches ein Interrupt existiert und eingerichtet ist, so wird der Zustand des Hauptprogramms eingefroren und stattdessen die entsprechende Interrupt Service Routine (ISR) ausgeführt. Nachdem die ISR ausgeführt wurde, wird das Hauptprogramm wieder fortgeführt. Auf diese Weise lässt sich z.B. realisieren, dass eine bestimmte Aktion in festen Zeitabständen zyklisch durchgeführt wird, indem einer der Hardware-Timer so konfiguriert wird, dass er entsprechend einen Interrupt generiert. Dieses Verhalten wird in diesem Projekt bspw. für einen Zeitgeber genutzt (siehe Abschnitt 5.4.1).

3.5 PWM

Die Pulsweitenmodulation ist ein Verfahren, mit welchem man aus einer konstanten Spannung V_{on} einen beliebigen Mittelwert zwischen 0V und V_{on} generieren kann. Folgendes Schaubild stellt ein solches PWM-Signal dar:

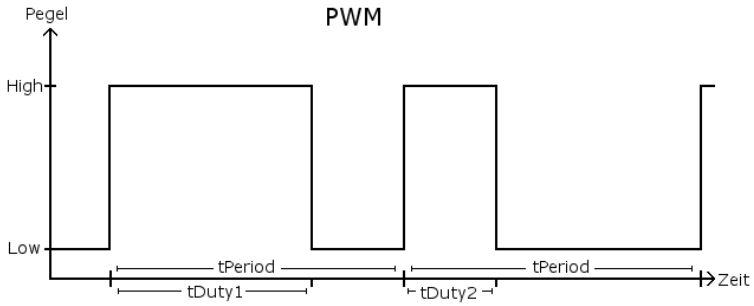

Abbildung 3 Schaubild: PWM-Signal

Während die Periodendauer T konstant bleibt, wird das Tastverhältnis t_d (dutycycle) variiert, welches das Verhältnis der An-Zeit zur Periodendauer angibt (0…1). Der Mittelwert einer zeitlich veränderlichen Spannung berechnet sich nach [HAG01], S. 204 folgendermaßen:

$$\bar{U} = \frac{1}{T}\int_0^T |u| dt$$

Da nur Spannungen ungleich 0 einen Beitrag zum Integral leisten und V_{on} stets positiv ist, lässt sich das Integral analytisch lösen:

$$\bar{U} = \frac{1}{T} t_d T V_{on} = t_d V_{on}$$

Der Mittelwert entspricht also einfach der Spannung V_{on} multipliziert mit dem Tastverhältnis t_d. Wenn man nun eine Frequenz wählt, die höher liegt, als das menschliche Auge wahrnimmt, lassen sich damit LEDs dimmen, was im Rahmen dieses Projektes genutzt wird, um die Intensität der verschiedenen Farben zu steuern.

3.6 Bordspannung PKW

Um die verschiedenen Verbraucher, z.B. Licht, Radio, Motorsteuergerät aber auch den Starter betreiben zu können, benötigt man eine Spannungsquelle. Da das Fahrzeug mobil sein soll, ist die derzeitig einzig sinnvolle Lösung eine entsprechend groß dimensionierte Batterie, welche auch Starterbatterie genannt wird. Würde man nur eine solche Batterie verbauen, wäre das Fahrzeug nur solange mobil, bis diese Batterie entladen wäre, eine sog. Tiefenentladung vermindert zudem erheblich die Lebenserwartung der Batterie. Aus diesem Grund wird zusätzlich zur Batterie ein Generator verbaut, welcher mit Hilfe eines Riemens vom Motor angetrieben wird und bei laufendem Motor die Versorgung der Verbraucher sicherstellt, während er gleichzeitig die Batterie wieder auflädt. Somit hat die Batterie nur noch zwei wesentliche Aufgaben: Die Versorgung des Starters bei stehendem Motor und eine Glättung der vom Generator erzeugten Stromimpulse. Während die Bordspannung bei stehendem Motor 12V beträgt, steigt sie bei laufendem Motor auf 14V an, da die Batterie sich nur mit einer Spannungsdifferenz laden lässt (vgl. z.B. [KOR06], S. 182). Interessant ist, dass es einige Klemmenbezeichnungen gibt, die sich Herstellerübergreifend durchgesetzt haben, sodass z.B. Klemme 15 allgemein als Zündungsplus bekannt ist (führt bei eingeschalteter Zündung Spannung).

3.7 Kombiinstrument

Das Kombiinstrument, umgangssprachlich häufig als Tacho bezeichnet, befindet sich hinter dem Lenkrad im Blickfeld des Fahrers und stellt die relevantesten Informationen für ihn bereit. Darunter fallen z.B., je nach Fahrzeug, Fahrzeuggeschwindigkeit, Drehzahl des Motors, Füllungsstand des Tanks oder Hinweisleuchten für Störungen.

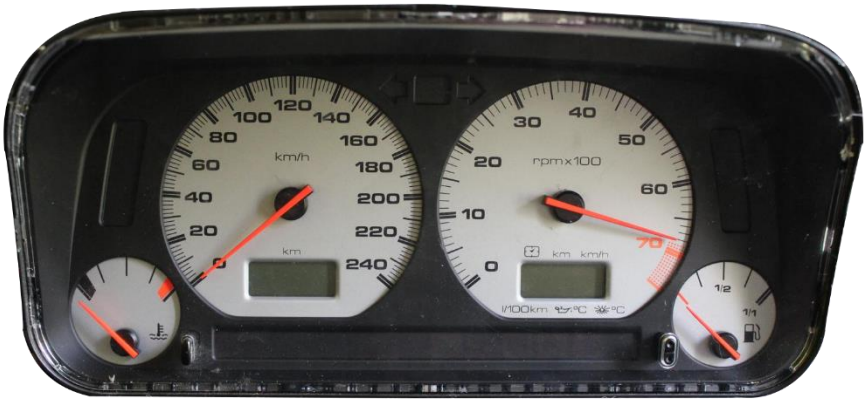

Abbildung 4 Foto: Kombiinstrument aus Golf 3

3.8 Unterbrecherzündung

Das Prinzip des Ottomotors, wie im Zielfahrzeug einer verbaut ist, macht es nötig, dass das im jeweiligen Zylinder befindliche Gasgemisch im richtigen Augenblick durch eine entsprechende Zündquelle entzündet wird. Zu diesem Zweck wird in den Zylinderkopf eine sogenannte Zündkerze in jeden Zylinder eingeschraubt, zwischen deren Elektroden ein Zündfunke entsteht, der das Gasgemisch entzündet. Der Zündfunke entsteht dadurch, dass zwischen den Elektroden eine so hohe Spannung angelegt wird, dass sie die Luft (das Gasgemisch) durchbricht. Diese Spannung wird in der sogenannten Zündspule generiert und durch einen Zündverteiler an den jeweiligen Zylinder geleitet. Um die hohe Spannung generieren zu können, wird in der Zündspule durch eine Primärwicklung mit wenigen Windungen der Eisenkern stark magnetisiert. Unterbricht man den Stromfluss in der Primärwicklung, so wird das Magnetfeld abgebaut und es kommt zur Selbstinduktion, wobei in der Sekundärwicklung eine Spannung generiert wird, die proportional zur Windungszahl ist (vgl. [HAG01], S. 183). Aus diesem Grund ist die Windungszahl der Sekundärseite sehr hoch gewählt und so lässt sich aus der geringen Spannung des Bordnetzes (12-14V) die benötigte Spannung in Höhe von mehreren Kilovolt erzeugen. Da zur Erzeugung des Zündfunkens der Stromfluss in der Primärspule unterbrochen werden muss, spricht man auch von einer Unterbrecherzündung (vgl. [KOR06], S. 189).

Es entsteht an der Primärseite somit eine Rechteckspannung, deren Frequenz abhängig ist von der Motordrehzahl. Da beim Viertaktmotor jeder Zylinder nur bei jeder zweiten Umdrehung einen Zündfunken benötigt, werden pro Kurbelwellenumdrehung bei einem Vierzylinder-Motor 2 Zündfunken generiert. Geht man von einer Umdrehung pro Minute aus, so entspricht dies 1/60 Umdrehung pro Sekunde, bzw. die Umdrehung dauert 60 Sekunden. Da pro Umdrehung 2 Zündfunken generiert werden, erfolgt am Anschluss der Primärwicklung alle 30 Sekunden eine negative Flanke. Steigt die Drehzahl, so verkürzt sich diese Zeit entsprechend.

3.9 Spannungsteiler

Ein Spannungsteiler ist eine sehr einfache Schaltung, mit der man aus einer anliegenden Spannung eine niedrigere generieren kann. Sie wird folgendermaßen aufgebaut:

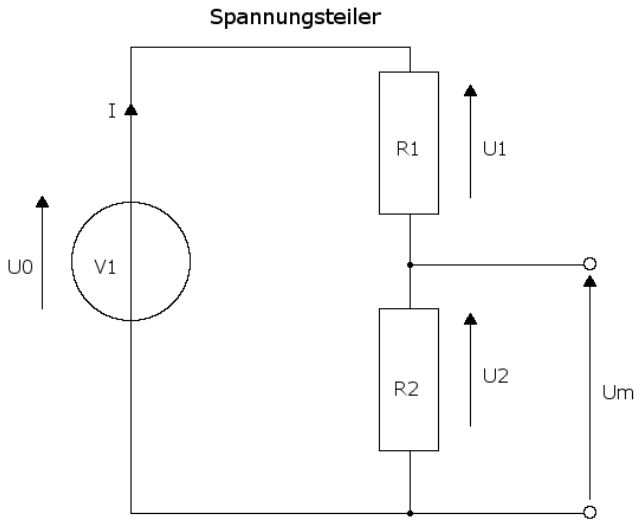

Abbildung 5 Schauild: Spannungsteiler

Aus dem ersten Kirchhoff'schen Gesetz folgt, dass (wenn der Abgriff zwischen den beiden Widerständen ungenutzt bleibt) der in der Schaltung fließende Strom I überall in der Schaltung gleich ist (vgl. [HAG01], S. 26). Ferner gilt für den fließenden Strom folgender Zusammenhang:

$$I = \frac{U_0}{R} = \frac{U_0}{R_1 + R_2}$$

Für die am jeweiligen Widerstand R abfallende Spannung gilt wiederum:

$$U = R * I$$

Die Spannung U_m, welche die gewünschte Ausgangsspannung ist, entspricht U_2 und lässt sich somit folgendermaßen bestimmen:

$$U_m = U_0 * \frac{R_2}{R_1 + R_2}$$

Dieser Zusammenhang gilt allerdings nur, solange am Ausgang kein weiterer Strom fließt. Fließt dort ein weiterer Strom, so erhöht sich der Strom durch R_1 und dort fällt eine höhere Spannung ab, sodass die eingestellte Spannung mit steigender Belastung des Ausgangs abnimmt. Aus diesem Grund eignet sich diese Schaltung nur, um im Verhältnis geringe Lasten zu betreiben, wie zum Beispiel zu Messzwecken.

3.10 RGB-LED

Bei einer RGB-LED handelt es sich um einen LED-Chip, in welchem 3 LEDs in den Farben Rot, Grün und Blau untergebracht sind, welche sich einzeln ansteuern lassen. An dieser Stelle wird vorausgesetzt, dass dem Leser bekannt ist, was eine LED ist. Durch Einstellen verschiedener Intensität für die drei Farben lässt sich dadurch, dass die Lichtquellen dicht beieinander liegen und der Mensch das emittierte Licht somit nicht voneinander unterscheiden kann, für ihn eine (theoretisch) beliebige Lichtfarbe darstellen. Weit verbreitet sind heute sogenannte RGB-LED-Streifen, bei denen auf einem relativ flexiblen Streifen mehrere solcher LEDs untergebracht werden. Mit solchen RGB-LED-Streifen kann man zusammen mit einem entsprechenden Controller bestimmte Objekte oder Räume in einer Farbe aufleuchten lassen, die dem persönlichen Geschmack entspricht. Das folgende Foto zeigt einen solchen RGB-LED-Streifen:

Abbildung 6 Foto: RGB-LED-Streifen

Der RGB-LED-Streifen besteht aus vielen kurzen Segmenten, deren Verbindungsstellen auftrennbar und lötfähig sind (die im Bild sichtbaren, offenen Kupferflächen), sodass sich ein solcher Streifen mit Einschränkung durch die Segmentlänge auf eine Wunschlänge kürzen oder verlängern lässt.

4 Hardware

4.1 RGB-LED-Streifen

Für die Realisierung der farbigen Beleuchtung des Kombiinstrumentes kommen RGB-LED-Streifen zum Einsatz. Dabei handelt es sich um flexible Streifen, welche in verschiedenen Längen erhältlich sind und üblicherweise aufgerollt vertrieben werden. Diese Streifen bestehen aus mehreren Segmenten, auf denen jeweils vier RGB-LEDs zusammen mit passenden Vorwiderständen angebracht und in Reihe verschaltet sind. Die Streifen lassen sich überall zwischen diesen Segmenten auftrennen und so auf die gewünschte Länge (vielfache der Segmentlänge) bringen. Die vier vorhandenen Anschlüsse werden durchgeschliffen. Es handelt sich dabei um eine von allen Farben gemeinsam genutzte Spannungsversorgung sowie die drei Kathoden der einzelnen Farben. Die Streifen sind auf eine Spannung von 12 Volt ausgelegt und lassen sich somit ideal direkt im Fahrzeug anschließen. Es ist jedoch zu beachten, dass die Spannung bei laufendem Motor etwa 14 Volt beträgt (siehe Abschnitt 3.6), sodass die Betriebsspannung um 2 Volt höher liegt, als vom Hersteller angegeben. Potentiell können die LEDs sich dadurch stärker erwärmen und die Lebensdauer verkürzt sich theoretisch. Da die Differenz jedoch relativ gering ist (ca. 17%), und sich durch die Reihenschaltung noch auf je vier LEDs aufteilt, wird dieser Aspekt nicht weiter berücksichtigt und in Kauf genommen, dass die LEDs unter Umständen früher ausfallen, als erwartet. Das folgende Foto zeigt einen solchen LED-Streifen:

Abbildung 7 Foto: RGB-LED-Streifen

4.2 Steckverbinder

Wie bereits in Abschnitt 2 erwähnt, ist eine Anforderung an das System, dass alle Verbindungen steckbar sind. Während beim Kombiinstrument und dem Starterknopf auf beiden Seiten die Verbindung angelötet werden muss und somit eine freie Steckerwahl möglich ist, ist für die Bordspannung und das Relais der Steckertyp vorgegeben und es werden Flachstecker verwendet.

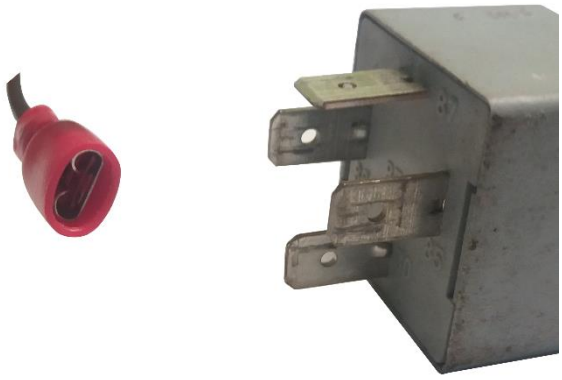

Abbildung 8 Foto: Flachstecker neben Relais

Die Verbindung zum Kombiinstrument benötigt vier Adern, drei zur Ansteuerung der Farben sowie eine zur Rückführung des Drehzahlsignals. Die Spannungsversorgung der LEDs erfolgt durch die Spannungsversorgung des Kombiinstrumentes selbst. Bei vier benötigten Adern bietet sich der Einsatz von USB-Steckern an, welche relativ robust sind (Steckertyp A) und für welche kurze Verlängerungen erhältlich sind. Durch den Einsatz einer Verlängerung und kurzer Anschlussstücke an den jeweiligen Komponenten wird ein Ausbau dieser zusätzlich erleichtert, da die Verlängerung im Fahrzeug verbleiben kann.

Abbildung 9 Foto: USB-Stecker und -Buchse

Der Starterknopf benötigt drei Adern: eine Spannungsversorgung, eine geschaltete Masse-Verbindung für die LED und eine Rückführung des Signals des Knopfes. Hierfür werden ein Miniklinkenkabel (3.5mm Durchmesser) sowie entsprechende Buchsen an Controller und Starterknopf verwendet.

Abbildung 10 Foto: Miniklinke

Somit konnte die Anforderung, alle Verbindungen steckbar zu gestalten, um einen leichten Ein-/Ausbau der einzelnen Komponenten zu ermöglichen, erfüllt werden.

4.3 Starterknopf

Der Starterknopf dient dazu, den Starter zu betätigen. Zum Einsatz kommt ein Drucktaster, welcher durch eine eingebaute LED über die Möglichkeit verfügt, zu leuchten. Über das Leuchten der LED soll die Freigabe zu Betätigung des Starters signalisiert werden. Da der Starterknopf lediglich einen Transistor ansteuert, ist das Schaltvermögen des Tasters vernachlässigbar. Der Einbau des Starterknopfes erfolgte in einer Leerblende in der Mittelkonsole des Fahrzeuges.

Abbildung 11 Foto: Starterknopf (Freigabe nicht erteilt)

Abbildung 12 Foto: Starterknopf (Freigabe erteilt)

4.4 Umbau des Kombiinstrumentes

Im originalen Zustand des Kombiinstrumentes werden die beiden LCD-Anzeigen jeweils von hinten mit einer Glühlampe angeleuchtet. Zur Beleuchtung der Ziffernblätter befinden sich Glühlampen im oberen Gehäuserand vor den Ziffernblättern. Die farbige Gestaltung der Beleuchtung erfolgt über Farbfilterfolien. Die vorhanden Farbfilterfolien sowie Glühlampen wurden entfernt und an deren Stelle die RGB-LED-Streifen angebracht. Weiterhin wurde das USB-Kabel zur Verbindung mit dem Controller eingelötet. Um Zugang zur Spannungsversorgung (Kl. 15, Zündungsplus) und zum Drehzahlsignal (Kl. 1) zu erhalten,

wurden die Adern direkt auf die entsprechenden Pins des Hauptsteckers gelötet. Die notwendigen Informationen über die Steckerbelegung stammen aus [HEI], S. 4, 5 und 31.

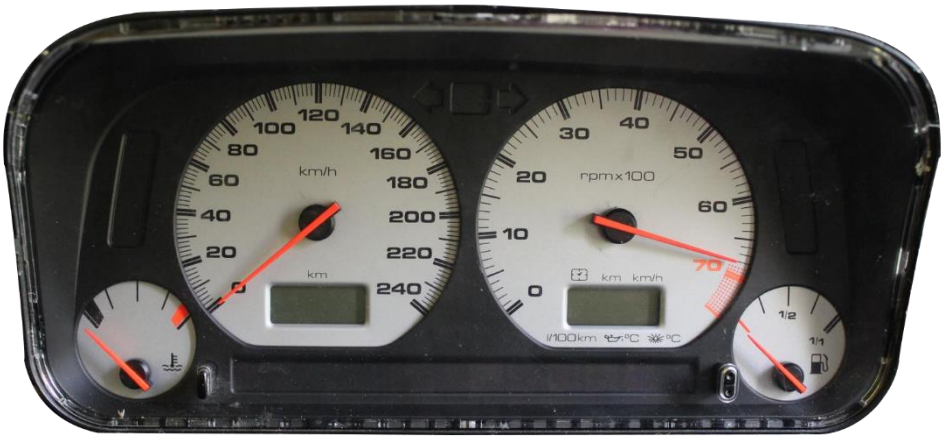

Abbildung 13 Foto: Kombiinstrument Volkswagen Golf 3

4.5 Herstellung des Controllers

4.5.1 Grundlegendes

Auf dem Controller müssen verschiedene Funktionen, Anschlüsse und der Mikrocontroller untergebracht werden. In der Entwicklungsphase wurden zunächst die einzelnen Funktionen auf einem Steckbrett aufgebaut und getestet. Nachdem die Funktionen erfolgreich getestet wurden, wurden der Mikrocontroller sowie die Funktionen auf einer Lochrasterplatine untergebracht. Um die Unterseite der Platine elektrisch zu isolieren, diese mechanisch zu schützen sowie eine Möglichkeit zur Zugentlastung der Leitungen zu schaffen, wurde die Platine auf einer Platte aus Acrylglas montiert.

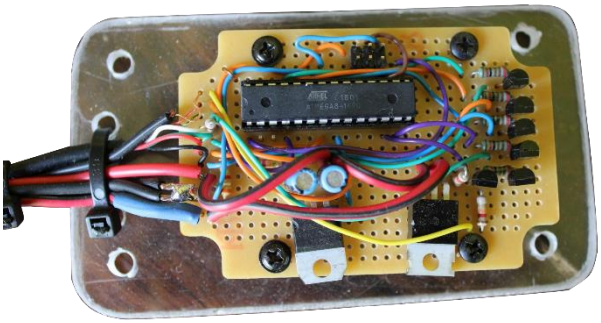

Abbildung 14 Foto: Controller Oberseite

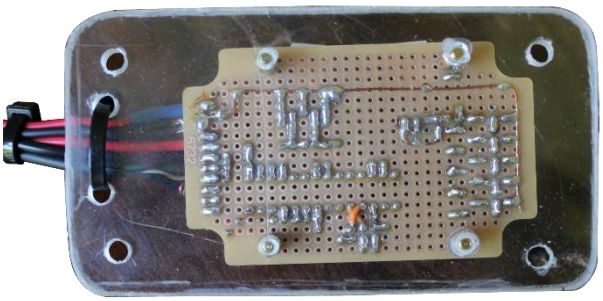

Abbildung 15 Foto: Controller Unterseite

4.5.2 Mikrocontroller

Als Mikrocontroller kommt ein Atmel ATmega8 zum Einsatz, welcher mit einer Taktfrequenz von 8 MHz betrieben wird und über 8 kB Programmspeicher verfügt. Seine Betriebsspannung beträgt 4.5-5.5 V (vgl. [ATM13], S. 1). Dieser Mikrocontroller wurde ausgewählt, weil er zur Verfügung stand, über die notwendigen Schnittstellen verfügt und vermutet wurde, dass dessen Leistung für die gestellten Aufgaben ausreicht, was sich im Rahmen des Projektes bestätigt hat.

4.5.3 Spannungsversorgung

Da die Bordspannung 12-14 Volt beträgt (siehe Abschnitt 3.6), während der verwendete Mikrocontroller eine Betriebsspannung von 5 Volt benötigt (siehe Abschnitt 4.5.2), muss diese in geeigneter Weise herabgesetzt werden. Zu diesem Zweck wird ein Festspannungsregler vom Typ L7805CV eingesetzt. Neben dem Spannungsregler selbst werden in der Minimalbeschaltung lediglich zwei zusätzliche Kondensatoren zur Spannungsstabilisierung benötigt (vgl. [STM18], S. 3).

4.5.4 Ansteuerung RGB-LED-Streifen

Wie bereits in Abschnitt 4.1 beschrieben, teilen sich die verschiedenen Farben der RGB-LEDs die positive Spannungsversorgung (gemeinsame Anode), während die Kathoden für jede Farbe einzeln herausgeführt sind. Möchte man also die Farben unabhängig voneinander steuern, so müssen dafür die Kathoden verwendet werden. Zu diesem Zweck werden drei NPN-Transistoren eingesetzt, welche die Kathode jeweils einer Farbe mit der Masse verbinden. Die Transistoren werden wiederum über Vorwiderstände an jeweils einen Ausgang des Mikrocontrollers angeschlossen, wobei die Vorwiderstände die Ausgänge vor Überlastung schützen sollen. Die Variation der jeweiligen Helligkeit wird dabei über ein PWM-Signal (siehe Abschnitt 3.5) realisiert. Das PWM-Signal wird dabei Softwareseitig erzeugt (siehe Abschnitt 5.4.5). Durch ein höheres bzw. niedrigeres Tastverhältnis wird der Mittelwert der angelegten Spannung erhöht bzw. verringert und die LED scheint heller bzw. dunkler zu leuchten. Es ist zu beachten, dass die PWM-Frequenz höher liegen muss als die Geschwindigkeit, mit der das menschliche Auge Bilder verarbeiten kann, da sonst ein Flackern wahrgenommen wird.

4.5.5 Pegelanpassung Drehzahlsignal

Ebenso wie die Bordspannung des PKW liegt das Pegelniveau des Drehzahlsignals bei 12V. Es handelt sich dabei um ein Rechtecksignal, dessen Frequenz mit der Drehzahl zunimmt (siehe Abschnitt 3.8) und es müssen somit nur die beiden Spannungen 0V und 12V verarbeitet werden. Aus diesem Grund gestaltet sich die Schaltung zur Pegelanpassung sehr übersichtlich: Es handelt sich dabei um einen Spannungsteiler (siehe Abschnitt 3.9), welcher die am Mikrocontroller anliegende Spannung auf einen Bereich zwischen 2V und 3V wandelt. Es wird absichtlich ein gewisser Abstand zu den vom Mikrocontroller erwarteten 5V eingerichtet, um eventuelle Spannungsspitzen auszugleichen. Mit der Spannung zwischen 2V und 3V funktioniert die Drehzahlmessung dennoch störungsfrei.

4.5.6 Ansteuerung Starter

Wie bereits in Abschnitt 2 beschrieben, erfolgt die Ansteuerung des Starters indirekt über ein Relais, welches sich nicht auf dem Controller befindet. Auf dem Controller selbst befindet sich lediglich die Ansteuerung des Relais. Realisiert wird diese primär über einen P-Channel MosFET, welcher zwischen positiver Spannung und der Last geschaltet wird. Der MosFET wird leitend, wenn die am Gate anliegende Spannung um ein bestimmtes Niveau unterhalb der an Source anliegenden Spannung fällt. Das Gate wird über einen Vorwiderstand genauso wie Source des MosFETs an 12V gelegt. Da das Gate isoliert ist und somit kein Strom durch den Widerstand fließt, liegen an Gate und Source dieselbe Spannung an, womit der MosFET im Normalfall sperrt und das Relais nicht ansteuert. Zur Ansteuerung des MosFETs wird das Gate mit Hilfe von zwei in Reihe geschalteten NPN-Transistoren mit der Masse verbunden, was einem UND-Gatter entspricht. Während der eine Transistor vom Mikrocontroller angesteuert wird, erfolgt die Ansteuerung des zweiten Transistors über den Starterknopf. Werden beide

Transistoren angesteuert, fließt ein Strom durch den Vorwiderstand am MosFET. Da die Widerstände der Transistoren nahe null sind, fällt (beinahe) die gesamte Spannung am Widerstand ab und die Spannung am Gate entspricht (beinahe) 0V (vgl. Prinzip Spannungsteiler, Abschnitt 3.9). Der MosFET wird leitend und das Relais wird angesteuert. Die Verwendung eines UND-Gatters soll die Wahrscheinlichkeit verringern, dass der Starter ungewollt betätigt wird. So müssen sowohl der Freigabeausgang des Mikrocontrollers, als auch der Kontakt des Starterknopfes Spannung führen, um den Starter zu betätigen.

4.5.7 Diagnose-Anschlüsse

Neben den für die dauerhafte Verwendung vorgesehenen, fest angebrachten Kabelverbindungen benötigt der Controller noch zwei weitere Anschlüsse zur Verwendung von SPI (siehe Abschnitt 3.3) und UART (siehe Abschnitt 3.2). SPI wird verwendet, um den Mikrocontroller zu programmieren. Durch die Implementierung der Schnittstelle auf dem Controller ist es möglich, den Mikrocontroller im aufgebauten Testsystem zu programmieren, was eine deutliche Zeitersparnis bei der Entwicklung mit sich bringt. UART ist die Schnittstelle für die serielle Kommunikation, über welche (auch im eingebauten Zustand) der Datensatz manipuliert werden kann. Diese Möglichkeit ist besonders wichtig, um die Einstellungen später im Fahrzeug genau an die Ansprüche anpassen zu können. Da diese beiden Schnittstellen nur selten genutzt werden, sind diese lediglich auf Stiftleisten herausgeführt.

4.5.8 Schaltplan

Im folgenden Abschnitt ist der Schaltplan dargestellt, nach dem der Controller aufgebaut ist. Nicht dargestellt sind die Verbindungen zu den Diagnoseanschlüssen, da es sich dabei lediglich um direkte Verbindungen zwischen den Stiften und den Pins des Mikrocontrollers handelt. Stattdessen wurde versucht, den Schaltplan übersichtlicher zu gestalten.

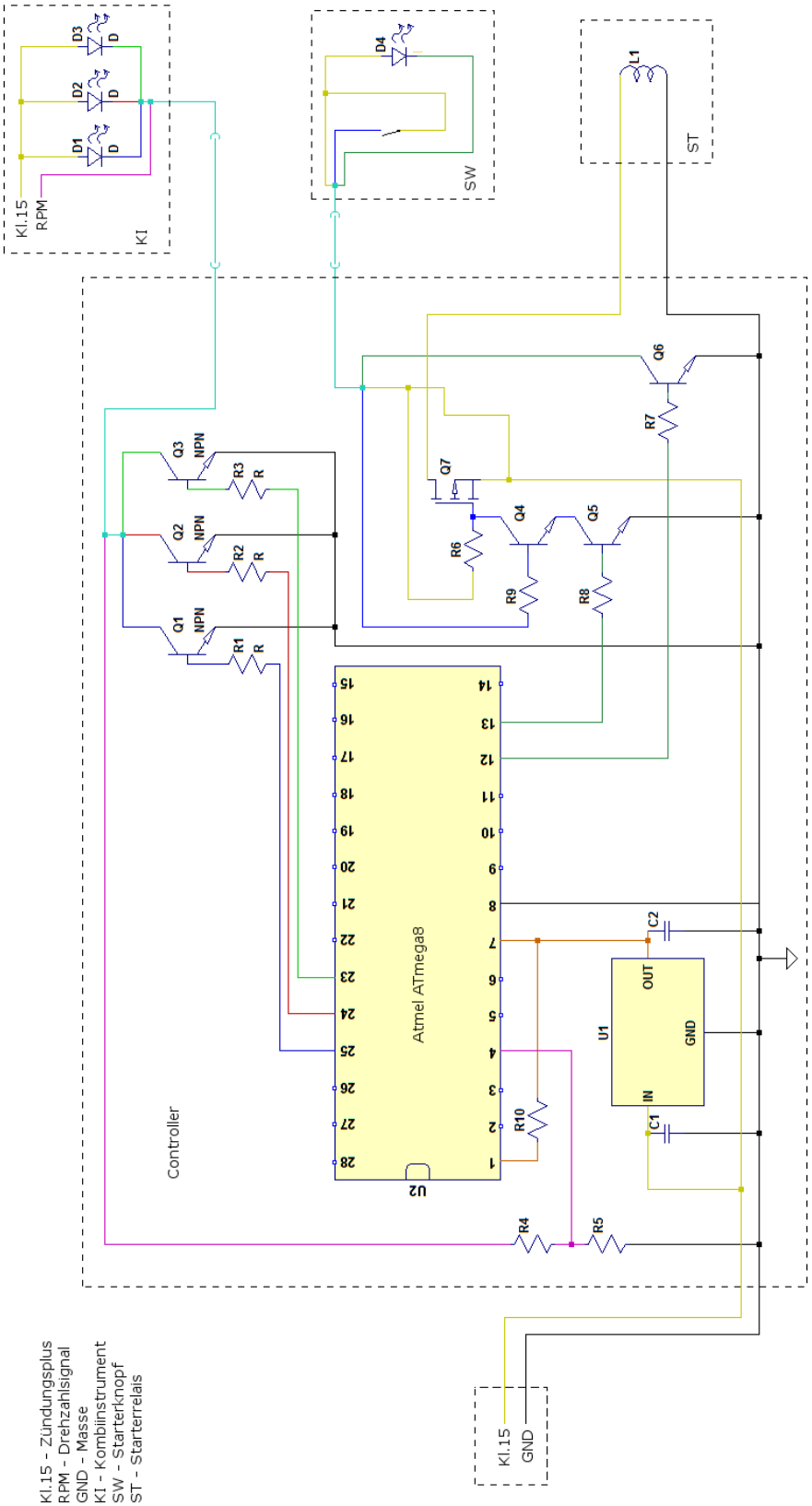

Abbildung 16 Schaltplan des Controllers

5 Software

5.1 Einleitung

Wie bereits in Abschnitt 2 beschrieben, sind im Rahmen des Projektes drei Softwarelösungen entstanden: Der Frequenzgenerator, der Controller und das Interface. Neben diesen Hauptlösungen sind noch einige Bibliotheken entstanden, welche zuerst vorgestellt werden. Anschließend erfolgt eine genauere Beschreibung der drei Hauptlösungen.

5.2 Bibliotheken

5.2.1 bitOperation

Bei dieser Bibliothek handelt es sich um eine sehr kompakte Sammlung von Funktionen, um einzelne Bits in einem Register auf einen Wert zu setzen oder den Wert eines Bits auszulesen. Der Sinn dahinter ist, die zum Teil unübersichtlichen Bit-Operationen als übersichtliche Funktionsaufrufe darzustellen.

5.2.2 charBuffer

Diese Bibliothek implementiert einen leichtgewichtigen, zirkulären Puffer zum Verwalten von Variablen vom Typ uint8_t und wird auf dem Controller verwendet, um die Ein- und Ausgabe der seriellen Kommunikation zu puffern. Ein Puffer besteht dabei aus einem Struct, welches den Zeiger auf das Datenarray enthält, genauso wie die Länge desselben und einen Lese- sowie Schreibezeiger. Bei der Initialisierung des Puffers liegen der Lese- sowie Schreibezeiger auf dem Anfang des Arrays.

Soll ein Zeichen in den Puffer geschrieben werden, so wird zunächst der nächste Schreibezeiger berechnet, wobei ein eventueller Überlauf des Arrays berücksichtigt und in dem Fall wieder von vorne begonnen wird. Zeigt der nächste Schreibezeiger auf dieselbe Stelle wie der Lesezeiger, gilt der Puffer als voll und es wird nichts durchgeführt. Andernfalls wird an der Stelle des aktuellen Schreibezeigers der Wert hineingeschrieben und der nächste Schreibezeiger als aktueller übernommen. Zusätzlich wird eine Zählervariable der gespeicherten Zeichen inkrementiert. Auf diese Weise wird zwar einerseits ein Speicherplatz verschwendet, es ist aber andererseits gewährleistet, dass keine Zeichen überschrieben werden, während die Kontrollstrukturen sehr übersichtlich bleiben.

Beim Lesen eines Zeichens wird überprüft, ob die Anzahl gespeicherter Zeichen größer null ist (Zählervariable). Trifft das zu, so wird das Zeichen an der Stelle des Lesezeigers zurückgegeben, andernfalls der Wert null. Ansonsten wird nichts getan, um das Zeichen weiterhin verfügbar zu halten. Soll es gelöscht werden, so muss das mit der entsprechenden Funktion durchgeführt werden. Diese Arbeitsweise wird gewählt, da der Puffer regelmäßig auf einen Befehl durchsucht und dieser erst durchgeführt wird, wenn er vollständig vorliegt (siehe Abschnitt 5.4.2).

5.2.3 kombiData

Mit „kombiData“ wird der Datensatz bezeichnet, welcher auf dem Controller austauschbar ist und das Verhalten dessen bestimmt. Es handelt sich dabei um ein Struct, welches einige Variablen und wiederum weitere Structs enthält, welche in den nachfolgenden Abschnitten genauer vorgestellt werden. Da dieses Struct sowohl im Controller, als auch im Interface Anwendung findet, wurde es in einer Bibliothek ausgelagert, welche beiden Softwarelösungen

zur Verfügung steht. Ferner sind aus denselben Gründen in dieser Bibliothek noch einige Statusmeldungen definiert, welche der Controller versendet.

5.2.4 serialCommunication

Diese Bibliothek ist Teil des Interfaces und beinhaltet sämtliche Funktionen zur Steuerung der seriellen Schnittstelle, welche Betriebssystemabhängig sind. Auf diese Weise ist es möglich, mehrere Varianten dieser Bibliothek für verschieden Betriebssysteme zu erstellen und das Interface später durch simples Austauschen der Bibliothek auf verschiedenen Betriebssystemen lauffähig zu machen. Der genauere Aufbau wird später in Abschnitt 5.5.3 beschrieben.

5.3 Frequenzgenerator

Der Frequenzgenerator wurde entwickelt, um den Controller in einer geeigneten Testumgebung testen zu können. Zum einen ist der Aufbau des Systems für die Entwicklung auf einem Tisch deutlich komfortabler um Änderungen vorzunehmen und zu programmieren, als im Fahrzeug. Zum anderen werden Motor und Umwelt in der Entwicklungsphase nicht belastet, was den Grundsätzen ökonomischer und ökologischer Arbeitsweise entspricht. Die Entwicklung des Frequenzgenerators sollte jedoch außerdem dazu dienen, Teilaspekte der Software des Controllers bereits im Vorfeld zu entwickeln und zu testen, sodass diese dort einfach übernommen werden können. Zu diesem Zweck wurde der Frequenzgenerator als Erstes entwickelt und als Basis derselbe Mikrocontroller verwendet werden, der auch auf dem Controller zum Einsatz kommt.

Die Teilaspekte, die im Frequenzgenerator realisiert wurden, sind die Bibliotheken „bitOperation“ und „charBuffer“ ebenso wie der generelle Aufbau der Kommunikation (siehe Abschnitt 5.4.2). Außerdem wurden die Implementierung des Zeitgebers (siehe Abschnitt 5.4.1) sowie der Software-PWM im Frequenzgenerator entwickelt.

Der Frequenzgenerator unterstützt insgesamt vier Betriebsmodi, welche über die serielle Schnittstelle eingestellt werden können. Neben den beiden Modi „dauerhaft aus“ und „dauerhaft an“ gibt es einen Modus für feste Frequenzen sowie einen Modus, in dem die Frequenz mit Hilfe eines analogen Eingangs (Potentiometer) zwischen einem oberen Grenzwert und 1 Hz variiert werden kann. Letzterer ist der für Testzwecke wichtigste Modus, da hier kontinuierlich die Drehzahl verändert und dabei die Veränderung der Farbgebung beobachtet werden kann. Da der Frequenzgenerator im Rahmen dieses Projektes zur Simulation der Drehzahl in 1/min genutzt werden sollte, wurden zwei verschiedene Möglichkeiten implementiert, die erzeugte Frequenz mittels serieller Kommunikation einzustellen: entweder erfolgt die Vorgabe einer Frequenz [1/s], oder einer Drehzahl [1/min], wobei die Drehzahl im Frequenzgenerator in die passende Frequenz umgerechnet wird.

Damit ist der Frequenzgenerator in ausreichendem Umfang beschrieben und es folgt die Beschreibung des Controllers.

5.4 Controller

5.4.1 Zeitgeber

Eine der wichtigsten Funktionen des Controllers ist der Zeitgeber. Dieser wird für die Realisierung der Software-PWM sowie für die Berechnung zeitabhängiger Farbeffekte benötigt. Die Funktion wird erreicht, indem ein Hardware-Timer des Mikrocontrollers so konfiguriert wird, dass er alle 100µs einen Interrupt erzeugt. In diesem Interrupt wird eine

Zählervariable inkrementiert, welche somit die Systemlaufzeit in 100µs repräsentiert. Als Datentyp wird ein uint32_t verwendet, welcher Zahlenwerte bis $4.295*10^9$ aufnehmen kann.

$$(2^{32} - 1) * 100\mu s \approx 429497s \approx 4.97d$$

Der Controller müsste also etwas weniger als fünf Tage durchgängig betrieben werden, bevor ein Überlauf stattfindet. Es ist unwahrscheinlich, dass dieser Fall eintritt, da der Controller nur bei eingeschalteter Zündung aktiv ist, womit dieser nicht behandelt werden muss. Mit Hilfe des Zeitgebers können mehrere im Projekt Timer genannte Stoppuhren realisiert werden, um zyklische Funktionen zu realisieren. Zu diesem Zweck wird ein Array des Typs uint32_t und der Größe entsprechend der Anzahl der gewünschten Timer erstellt. Mit Hilfe der Funktion *resetTimer(...)* wird der aktuelle Wert des Zeitgebers in den angegebenen Timer geschrieben. Anschließend kann mit Hilfe der Funktion *getTimeDiff(...)* die seit dem letzten Reset vergangene Zeit des jeweiligen Timers ermittelt werden. Die Timer werden für verschiedene Zwecke eingesetzt, wie z.B. der Messung der Drehzahl, der Realisierung der Dimmer (siehe Abschnitt 5.4.7) oder der zyklischen Überprüfung der aktuell anzuwendenden Effekte.

5.4.2 Kommunikation

Als Basis für die serielle Kommunikation dient die UART-Schnittstelle des Mikrocontrollers (siehe Abschnitt 3.2). Hierbei ist zu beachten, dass der UART des Mikrocontrollers immer nur ein Zeichen empfangen und eines zwischenspeichern kann. Aus diesem Grund wird das Empfangen interrupt-gesteuert durchgeführt. Der Empfängerteil des UARTs erzeugt immer einen Interrupt, sobald ein Zeichen empfangen wird. Innerhalb der Interrupt-Routine wird das Zeichen zunächst nur gelesen und in den Eingangspuffer geschrieben, um das Hauptprogramm und andere Interrupts nicht unnötig lange zu unterbrechen. Das Auswerten der Zeichen erfolgt in einer separaten Funktion, welche zyklisch im Hauptprogramm aufgerufen wird. Das Versenden von Daten erfolgt ebenfalls interrupt-gesteuert über einen Ausgangspuffer. Soll eine Nachricht versendet werden, so wird diese zunächst in den Ausgangspuffer geschrieben. Anschließend wird das erste Zeichen aus dem Ausgabepuffer in das Ein-/Ausgaberegister des UARTs geschrieben und aus dem Ausgabepuffer wieder gelöscht, woraufhin der UART beginnt, das erste Zeichen zu senden. Auch das Senden eines Zeichens generiert anschließend einen Interrupt, in welchem zunächst geprüft wird, ob im Ausgabepuffer ein weiteres Zeichen vorhanden ist. Trifft das zu, so wird dieses Zeichen wieder in das Ein-/Ausgaberegister des UARTs geschrieben und aus dem Ausgabepuffer gelöscht. So wird im Hauptprogramm die zu sendende Zeichenkette nur in den Ausgabepuffer geschrieben, deren Versenden anschließend quasiparallel zum Hauptprogramm stattfindet, sodass das Hauptprogramm währenddessen bereits weiterlaufen kann.

Um die Übertragung von Befehlen relativ sicher zu gestalten, wurde ein Nachrichtensystem entworfen, sämtliche Kommunikation findet in Form von definierten Nachrichten statt. Jede Nachricht beginnt dabei mit einem für diese Nachricht exklusiven Zeichen und endet mit dem Zeichen ‚e'. Zwischen diesen beiden Zeichen muss eine fest definierte Menge von Zeichen übertragen werden, welche die Nutzdaten darstellen. Bei einfachen Befehlen beträgt die Menge null. In der zyklisch im Hauptprogramm aufgerufenen Funktion *handleData()* wird stets mit Hilfe der Funktion *hasNextCommand()* abgefragt, ob aktuell ein gültiger Befehl vorliegt. Liefert die Funktion 1 zurück, so wird der aktuell vorliegende Befehl ausgewertet. Die Funktion *hasNextCommand()* geht folgendermaßen vor: Zunächst wird geprüft, ob im Eingangspuffer ein Zeichen vorliegt. Trifft das zu, so wird geprüft, ob es sich dabei um einen bekannten Befehl

handelt. Trifft das nicht zu, so wird das Zeichen gelöscht und der Status „Unknown Command“ zurückgesendet. Andernfalls wird als nächstes geprüft, ob bereits (mindestens) so viele Zeichen im Eingangspuffer vorliegen, wie für den Befehl definiert sind. Trifft das zu, wird bei vorhandenem Terminator ‚e‘ von der Funktion eine 1 zurückgegeben, andernfalls wird die vorgegebene Datenmenge aus dem Eingangspuffer gelöscht und der Status „Invalid Command“ zurückgesendet.

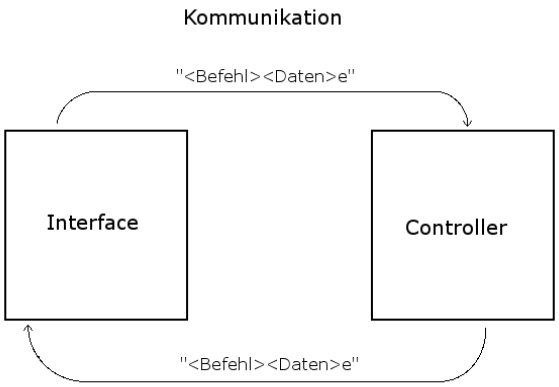

Abbildung 17 Schaubild: Nachrichtensystem

Durch das Nachrichtensystem wird sichergestellt, dass nur vollständig übertragene Befehle verarbeitet werden. Ohne diese Maßnahme wären Anfang und Ende eines Befehls nicht mehr bestimmt, falls durch einen Fehler einmal ein Zeichen zu viel oder zu wenig übertragen werden sollte und das Verhalten des Controllers wäre nicht mehr vorhersagbar. Mit Hilfe des Nachrichtensystems bleiben der Controller und das Interface jedoch synchronisiert.

5.4.3 Datenverwaltung

Bei dem austauschbaren Datensatz handelt es sich um das Struct *kombiData*, welches die für das Verhalten des Controllers wesentlichen Daten enthält. Im Arbeitsspeicher werden stets zwei Datensätze gespeichert: *kdActive* und *kdCache*. Während *kdActive* dem aktiven Datensatz entspricht, dient *kdCache* als Zwischenspeicher für Daten, die manipuliert werden. Erst wenn alle Daten manipuliert sind, werden in einem separaten Befehl alle Interrupts deaktiviert, *kdCache* in *kdActive* übertragen, und anschließend alle Interrupts wieder aktiviert. Auf diese Weise wird sichergestellt, dass die Farbgebung kein undefiniertes Verhalten aufweist, während Daten manipuliert werden. Zusätzlich kann *kdCache* im EEPROM abgelegt und daraus wieder geladen werden, um den Datensatz dauerhaft zu speichern. Das ist vom Programm auch so vorgesehen, da der Controller bei jedem Start den Datensatz aus dem EEPROM lädt und aktiviert. Andernfalls müsste man bei jedem Einschalten der Zündung erneut den gewünschten Datensatz in den Controller laden.

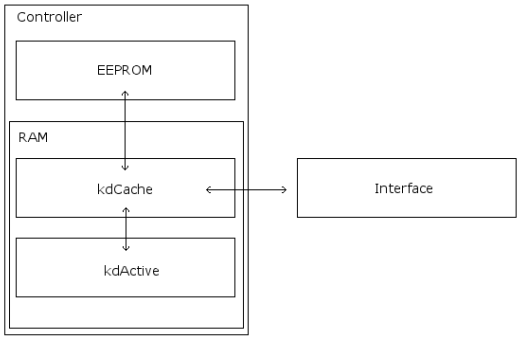

Abbildung 18 Schaubild: Speicherverwaltung

5.4.4 Drehzahlmessung

Bei dem Drehzahlsignal, das ausgewertet werden muss, handelt es sich um ein Rechtecksignal mit variabler Frequenz (vgl. Abschnitt 3.8), sodass die Frequenz bzw. Periodendauer des Signals die benötigte Information trägt. Die Messung der Drehzahl stellt gewisse Echtzeit-Anforderungen. Würde man die Drehzahl im Hauptprogramm auswerten, so wäre das Ergebnis nicht nur abhängig vom Signal selbst, sondern auch von der Programmlaufzeit, die je nach auftretenden Interrupts und Programmverzweigungen variabel ist. Dieser Umstand wäre inakzeptabel. Stattdessen wird zur Messung einer der Interrupt-fähigen Pins des Mikrocontrollers verwendet, um mit Hilfe eines Interrupts sofort auf einen Impuls des Drehzahlsignals zu reagieren. Dazu wird der Pin so konfiguriert, dass er bei Auftreten einer positiven Flanke einen Interrupt erzeugt, wobei die Wahl der positiven Flanke zufällig ist. Man könnte genauso gut auf die negative Flanke reagieren oder auf positive und negative gleichermaßen mit Anpassung der Auswertung. Bei Auftreten des Interrupts wird mittels eines Timers die Zeitdifferenz zum letzten Auftreten des Interrupts bestimmt und eine vordefinierte Konstante durch diese Zeitdifferenz geteilt, wodurch direkt die Drehzahl in Umdrehungen pro Minute berechnet wird. Wie bereits in Abschnitt 3.8 erklärt wurde, erfolgt bei einer Umdrehung alle 30 Sekunden eine Flanke. Beachtet man die Basis von 100µs, so erklärt sich daraus die Konstante von 300000, welche durch die Zeitdifferenz geteilt wird. Da sowohl der Zeitgeber, als auch die Reaktion auf eine Flanke Interruptgesteuert sind, während die in den Interrupts aufgerufenen Funktionen sehr kurz sind, kann eine hohe Messgenauigkeit erreicht werden. Jedoch weist dieser Aufbau eine Schwäche auf: Bleibt der Motor stehen oder tritt ein Kabelbruch auf, so erfolgt kein Impuls mehr und der Interrupt zur Auswertung wird nicht mehr erzeugt. Die letzte ermittelte Drehzahl würde also gespeichert bleiben. Um dieses Problem zu lösen, wird im Hauptprogramm zyklisch geprüft, ob die Zeitdifferenz des Timers einen gewissen Wert überschritten hat. Wenn dem so ist, wird die Drehzahl im Hauptprogramm manuell auf den Wert null gesetzt.

Zur Verifikation der korrekten Auswertung wird die Testumgebung genutzt: Entspricht das Verhalten der Farbgebung bei bestimmten, auf dem Kombiinstrument abgelesenen Drehzahlen dem gewünschten Verhalten, so ist die Auswertung korrekt. Dabei spielt es keine Rolle, ob die gemessene Drehzahl genau der reellen Drehzahl entspricht, da die optische Wahrnehmung entscheidend ist. Aus diesem Grund muss die durch den Controller gemessene Drehzahl mit den Werten übereinstimmen, die auf dem Kombiinstrument angezeigt werden, was in der Testumgebung als erfüllt bewertet wurde.

5.4.5 PWM-Erzeugung

Um die Helligkeit bzw. Intensität der verschiedenen Farben der RGB-LEDs zu verändern, werden diese mit Hilfe von PWM-Signalen angesteuert. Da die drei Signale in Phase sein sollen und der Mikrocontroller über keinen Hardware-Timer mit drei Ausgängen verfügt, werden die Signale softwareseitig erzeugt. Zu diesem Zweck wurde die Funktion *handlePWM()* geschrieben, die die PWM-Signale mit Hilfe eines Timers realisiert. In der Funktion wird zunächst geprüft, ob die Zeitdifferenz die definierte Periodendauer erreicht hat. Ist dem so, werden der Timer zurückgesetzt und alle drei Ausgänge eingeschaltet, falls deren jeweiliges Tastverhältnis größer null ist. Danach wird für jede Farbe einzeln geprüft, ob die Zeitdifferenz das jeweilige Tastverhältnis überschritten hat und in dem Fall der jeweilige Ausgang wieder ausgeschaltet. Die Funktion ist so entworfen, dass sie zyklisch aufgerufen wird, und zwar in demselben Interrupt, den auch der Zeitgeber verwendet.

5.4.6 Breakpoints

Beim *Breakpoint* handelt es sich um ein untergeordnetes Struct von *kombiData*, welches maßgeblich das Farbverhalten abhängig von der Drehzahl bestimmt. Der Name leitet sich dabei von der Realisierung der Funktion als Lookup-Tabelle ab, deren Grundlage die Stützstellen (engl. Breakpoints) sind. Ein *Breakpoint* beinhaltet die Drehzahl, an der er die Funktion stützt, sowie die Tastverhältnisse der Farben, die an dieser Stelle vorliegen sollen. Zwischen zwei Breakpoints wird stets linear interpoliert, wodurch ein Verlauf der Farben realisiert werden kann. Die folgende Abbildung stellt eine beispielhafte Konfiguration der *Breakpoints* dar:

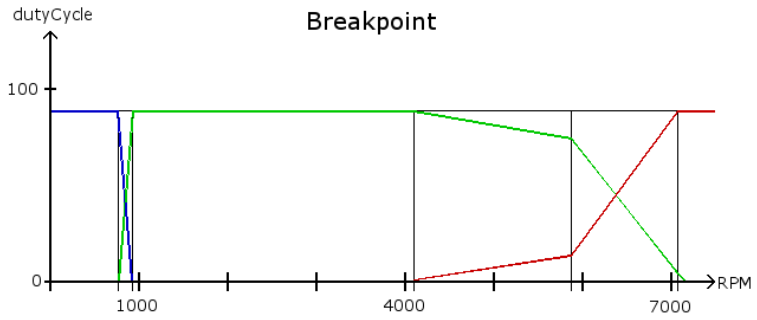

Abbildung 19 Schaubild: Funktion der Breakpoints

Dass die verschiedenen Farben sich dieselben Stützstellen teilen, hat den simplen Grund, dass so Speicher gespart werden kann und sich die Auswertungsfunktionen kompakter gestalten lassen. Zur Auswertung der Farben wird in der Funktion *determineActiveEffects()* zyklisch bestimmt, welcher *Breakpoint* entsprechend der aktuellen Drehzahl aktiv sein soll. Weicht der so bestimmte *Breakpoint* vom aktuellen ab, so werden der neue Breakpoint übernommen und die Parameter für die drei Geraden neu bestimmt. Die Funktion *calculateEffects()* zur Berechnung, die dauerhaft im Hauptprogramm aufgerufen wird, muss also nur noch drei Geradengleichungen auswerten.

5.4.7 Dimmer

Die *Dimmer* dienen dazu, ein gepulstes Signal zu erzeugen und werden ebenfalls durch ein untergeordnetes Struct von *kombiData* parametriert. Grund für die Entwicklung waren zwei Situationen: Gilt der Motor als aus, so soll ein langsames Pulsieren der Beleuchtung diesen Zustand signalisieren. Ist die aktuelle Drehzahl hingegen sehr hoch, so soll ein schnelles

Flackern den Fahrer dazu auffordern, einen höheren Gang einzulegen. Das folgende Schaubild soll die vier Phasen veranschaulichen, die der Dimmer zyklisch durchläuft:

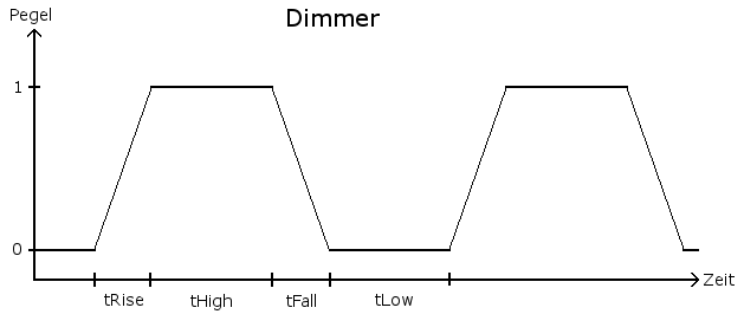

Abbildung 20 Schaubild: Funktion Dimmer

Neben den hier dargestellten Zeitdauern der verschiedenen Phasen werden in dem jeweiligen *Dimmer* eine minimale und eine maximale Drehzahl gespeichert, zwischen denen dieser aktiv wird. Die Bestimmung des aktiven *Dimmers* erfolgt ebenfalls in der Funktion *determineActiveEffects()*, wobei zu beachten ist, dass stets nur der erste Dimmer aktiv ist, dessen Anforderungen an die Drehzahl erfüllt sind. Je nach Phase und Dauer der aktiven Phase wird in der Funktion *calculateEffects()* ein Wert zwischen 0 und 1 bestimmt. Dieser Wert wird anschließend mit dem Ergebnis des zuvor berechneten *Breakpoints* multipliziert, um so die Helligkeit zu manipulieren.

5.4.8 Hysterese

Nach der Implementierung der *Breakpoints* und *Dimmer* hat sich in der Testumgebung gezeigt, dass der Controller permanent zwischen zwei Effekten hin- und herschaltet, wenn die Drehzahl, die gewissen Schwankungen unterliegt, sich im Grenzbereich dazwischen findet. Dieser Effekt ist wahrnehmbar und wirkt störend, weswegen die Hysterese eingeführt wurde. Dabei handelt es sich um zwei unabhängige Werte für *Breakpoints* und *Dimmer*, die einer Drehzahldifferenz entsprechen. In jede Richtung muss die Drehzahl erst die Summe aus Grenzwert und Drehzahldifferenz überschreiten, bevor der nächste/vorherige Effekt aktiv wird. So konnte das unerwünschte Verhalten stark vermindert werden.

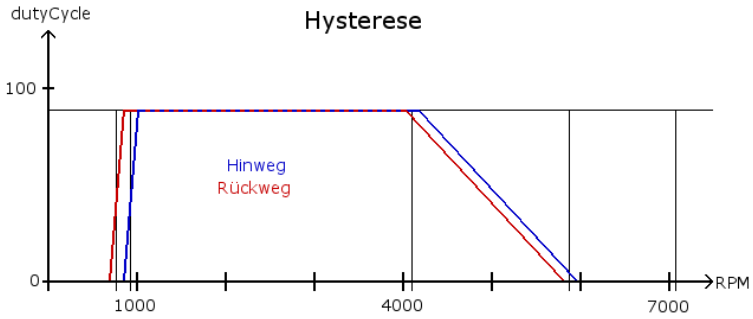

Abbildung 21 Schaubild: Funktion der Hysterese

5.4.9 Starterfreigabe

Wie zuvor beschrieben, hat der Controller auch die Aufgabe, die Freigabe zur Betätigung des Starters zu erteilen bzw. die Freigabe wieder zurückzuziehen, wenn der Motor als laufend bewertet wird. Zu diesem Zweck werden lediglich die zwei Werte *rpmStarterOff* und

rpmStarterOn in *kombiData* abgelegt. Steigt die Drehzahl über den in *rpmStarterOff* gespeicherten Wert, so wird die Freigabe zurückgenommen. Fällt die Drehzahl wieder unter den in *rpmStarterOn* gespeicherten Wert, so wird die Freigabe wieder erteilt. Die Auswertung sowie das Ein- und Ausschalten der Ausgänge sind dabei mit in die Funktionen *calculateEffects()* und *handlePWM()* eingebettet. Durch die Starterfreigabe wird sichergestellt, dass der Starter nicht betätigt werden kann, wenn der Motor bereits läuft, da dies Schäden am Starter sowie auch an der Schwungscheibe verursachen kann.

5.4.10 Programmstruktur (Echtzeit)

In diesem Abschnitt soll zusammengefasst dargestellt werden, wie den verschiedenen Funktionen die zur einwandfreien Funktion benötigte Rechenzeit zugesichert wird. Wichtig dafür ist das Konzept des Interrupts. Während die weniger wichtigen Funktionen in der Dauerschleife im Hauptprogramm aufgerufen werden, werden die zeitkritischen Funktionen in Interrupts aufgerufen. Die Interrupts reagieren sofort auf bestimmte Hardware-Ereignisse und unterbrechen zu deren Abarbeitung das Hauptprogramm, wodurch die zuvor genannte Aufteilung begründet wird. Nach der Abarbeitung eines Interrupts wird das Hauptprogramm an der unterbrochenen Stelle wieder fortgeführt. Es ist bei der Gestaltung der Interrupts jedoch darauf zu achten, dass deren Aufrufhäufigkeit in Verbindung mit der Abarbeitungszeit der aufgerufenen Funktionen nicht dazu führt, dass das Hauptprogramm kaum noch Rechenzeit erhält oder gar die Interrupts beginnen, sich gegenseitig zu stören. Je nach Schwere ist ein korrektes Ablaufen des Programms nicht mehr sichergestellt oder sogar ausgeschlossen. Aus diesem Grund ist darauf zu achten, dass nur die wirklich wichtigen Funktionen innerhalb der Interrupts bearbeitet werden und dabei möglichst wenig Rechenzeit benötigen. Ebenso sollten die Interrupts nach Möglichkeit nur so häufig generiert werden, wie es nötig ist.

Zu den zeitkritischen Aufgaben wird zunächst der Zeitgeber gezählt, gefolgt von der Messung der Drehzahl sowie der Erzeugung der PWM-Signale. Variieren die hierfür aufgewendeten Zykluszeiten, so wird dies schnell in der Farbwiedergabe sichtbar. Weiterhin ist es sehr wichtig, dass die über UART empfangenen Zeichen möglichst Zeitnah in den Eingangspuffer übertragen werden, da der Mikrocontroller selbst nur einen Zwischenpuffer für ein weiteres Zeichen bereithält. Das Versenden von Zeichen erfolgt ebenfalls interruptgesteuert, da das Hauptprogramm sonst darauf warten muss, bis alle Zeichen versendet wurden. Das Bestimmen der aktiven Effekte sowie das Berechnen der aktuellen Farbwerte sind zwar ebenfalls als relativ zeitkritisch zu betrachten, aber bei Weitem nicht so kritisch wie die zuvor genannten Funktionen, weshalb diese Funktionen lediglich dauerhaft im Hauptprogramm aufgerufen werden. Das Auswerten empfangener Befehle ist am wenigsten kritisch, da das Interface nach dem Senden eines Befehls stets auf eine Antwort des Controllers wartet, bevor es weitere Befehle sendet. Der Eingangspuffer ist dabei so dimensioniert, dass jeder bekannte Befehl ohne Schwierigkeiten hineinpasst.

Durch die beschriebene Struktur konnte eine störungsfreie Farbwiedergabe erreicht werden, womit die Anforderungen an Echtzeit erfüllt werden konnten.

5.5 Interface

5.5.1 Aufbau

Das Interface dient dazu, Datensätze für den Controller an einem Computer zu manipulieren und an diesen zu übermitteln. Bei der Entwicklung einer solchen Software muss man die

Entscheidung treffen, ob man die Bedienung konsolenbasiert entwirft, oder ein aufwendiges GUI entwickelt. Gemessen an der Tatsache, dass man in dem Interface im Wesentlichen einige Zahlenwerte anzeigen lassen und verändern kann, schien der Aufwand für die Entwicklung eines GUI als nicht verhältnismäßig und es wurde beschlossen, die Bedienung über die Konsole zu realisieren. Das Interface stellt somit einige Befehle bereit, mit denen man die zuvor vorgestellten Parameter des Structs *kombiData* verändern und sich übersichtlich anzeigen lassen kann.

```
Dimmer mit ID 0 erfolgreich angepasst.
--> dimmer 1 7000 15000 200 1100 200 1100
Dimmer mit ID 1 erfolgreich angepasst.
--> hysteresis 50 50
Hysterese-Parameter erfolgreich angepasst.
--> starter 300 650
Starter-Parameter erfolgreich angepasst.
--> listall
============[breakpoints]========
<ID> <rpm> <red> <green> <blue>
  0      0     0     0    80
  1    800     0     0    80
  2    820     0    80     0
  3   4000     0    80     0
  4   6000   100    60     0
  5   6500   100     5     0
  6   7000   100     0     0
  7   9000   100     0     0
  8  15000   100     0     0
  9      0     0     0     0
---------------------------------
```

Abbildung 22 Screenshot: Interface

5.5.2 Manipulation des Datensatzes

Aus den vorherigen Abschnitten ergibt sich, dass es insgesamt vier Typen von Daten gibt, die sich manipulieren lassen: *Breakpoints*, *Dimmer*, die Hysterese-Parameter sowie die Parameter der Starterfreigabe. Die ersten beiden Typen unterscheiden sich von den letzten Beiden darin, dass davon jeweils mehrere existieren, sodass bei den Befehlen für diese Typen zusätzlich zu den Parametern selbst noch die entsprechende ID angegeben werden muss. Der Befehl für die Manipulation eines *Breakpoints* lautet beispielsweise:

breakpoint <ID> <rpm> <red> <green> <blue>

Für jeden dieser Typen gibt es jeweils auch einen Befehl zur Auflistung der Parameter, z.B. *listbreakpoints*.

5.5.3 Serielle Kommunikation (Linux)

Für die serielle Kommunikation wird eine Betriebssystem-spezifische Bibliothek verwendet, welche im aktuellen Status nur für Linux entwickelt wurde. Sie stellt Funktionen bereit, um die verfügbaren Ports aufzulisten, sowie einen Port zu öffnen und wieder zu schließen. Diese Funktionen sind direkt über Befehle in der Konsole aufrufbar. Die zur tatsächlichen Kommunikation verfügbaren Funktionen der Bibliothek werden indirekt von den Befehlen *loaddata* sowie *getdata* aufgerufen, welche zum Laden eines Datensatzes in den Controller oder von dem Controller dienen und die richtige Kommunikation mit dem Controller sicherstellen.

5.5.4 Übertragen von Daten

Das Programm wendet zur Übertragung der Daten die auf dem Controller verfügbaren Befehle sowie die Bibliothek zur seriellen Kommunikation an. Um den vorhandenen Datensatz aus dem Controller zu erhalten, muss das Interface lediglich einen Steuerbefehl an diesen senden und die zurückgesendete Nachricht auswerten. Das Laden eines Datensatzes in den Controller ist

etwas umfangreicher. Zunächst wird der Datensatz in der entsprechenden Nachricht übertragen. Anschließend wird aus dem Controller der vorhandene Datensatz ausgelesen und mit dem Gesendeten verglichen. Stimmen beide überein, wird der Datensatz im Controller aktiviert und die Übertragung war erfolgreich. Ansonsten wird ein Fehler auf der Konsole ausgegeben. Sollen die aktuell im Controller gespeicherten Daten dauerhaft im EEPROM abgespeichert werden, so muss der Nutzer dies im Interface mit dem Befehl *savedata* durchführen. Das sollte er erst tun, wenn er sich sicher ist, dass die aktuellen Daten dem entsprechen, was er sich vorstellt. Dieses Verfahren dient dazu, den EEPROM zu schonen, dessen Schreibzyklen generell begrenzt sind.

5.5.5 Speichern und Laden von Datensätzen

Um mit verschiedenen Datensätzen arbeiten zu können bzw. nicht beim Programmstart immer wieder von vorne beginnen zu müssen, lassen sich die Datensätze in Dateien ablegen bzw. wieder aus Dateien einlesen. Ferner ermöglicht diese Funktionalität, dass man theoretisch mit anderen Personen Datensätze austauschen könnte, die ebenfalls dieses Projekt nutzen. Um eine gewisse Sicherheit zu erzeugen, dass die eingelesenen Daten auch aus dem Programm stammen, werden nur Dateien eingelesen, deren Länge exakt der Größe von *kombiData* entsprechen.

5.5.6 Ausführen von Skripten

Da sich das wiederholte Eintippen der Manipulationsbefehle zur Anpassung von Parametern als sehr mühselig herausstellte (es werden in dem jeweiligen Befehl immer alle Parameter angegeben), wurde noch die Möglichkeit geschaffen, Skripte zu laden, die man zuvor mit einem Texteditor erstellt hat. Ein solches Skript wird einfach Zeilenweise eingelesen und jede Zeile wie eine in die Konsole eingegebene Zeile behandelt. Auf diese Weise muss man nur noch in dem Skript den einzelnen Zahlenwert ändern und dieses anschließend erneut ausführen.

5.6 Dokumentation

Um die Software im Detail zu verstehen und deren Funktion nachvollziehen zu können, muss man in den beigefügten Quellcodes recherchieren. Es wurde versucht, möglichst aussagekräftige Variablen-, Konstanten- und Funktionsnamen zu vergeben. Des Weiteren wurden an allen Stellen, an denen es angebracht erschien, Kommentare hinzugefügt. Interessiert man sich nur für die Anwendung der Software, so ist die dem Quellcode beigefügte Readme-Datei zu Rate zu ziehen, in welcher die Anwendung beschrieben wird. Ferner verfügt das Interface über den Befehl „help“, welcher alle verfügbaren Befehle zusammen mit einer Beschreibung auflistet. Zu beachten ist, dass der Quellcode sowie dessen Kommentare vollständig in Englisch verfasst sind.

6 Probleme und Lösungen

6.1 Einleitung

Dieser Abschnitt dient dazu, die während des Entwicklungsprozesses entstandenen Probleme sowie die entsprechenden Lösungsansätze vorzustellen. Es ist zwar wünschenswert, dass ein Projekt auf Anhieb funktioniert, als wertvoller wird jedoch die Erfahrung eingeschätzt, welche Ansätze nicht funktionieren, und vor allem aus welchem Grund. Deswegen soll hier vor allem auf Probleme eingegangen werden, welche nicht erwartet wurden.

6.2 Schreiben in geteilte Register

Das erste Problem entstand indirekt durch das Einsetzen der Bibliothek *bitOperation*, welche übersichtliche Funktionen bereitstellt, um einzelne Bits zu manipulieren. Eine Eigenheit des ATmega8 ist, dass die beiden Register UCSRC und UBRRH, welche zur Kontrolle des UART dienen, sich dieselbe Adresse teilen. Um nun auf UCSRC statt UBRRH zuzugreifen, muss während des Schreibevorgangs das Bit 7 (URSEL) den Wert 1 tragen, andernfalls wird in das Register UBRRH geschrieben, welches der Baudrateneinstellung dient. Dadurch, dass dieses Verhalten durch Nutzen der Bibliothek *bitOperation* nicht berücksichtigt wurde (es wird stets nur ein einzelnes Bit gesetzt), stimmte die Baudrate nicht und es wurden 5 Datenbits statt 8 sowie eine falsche Baudrate eingestellt, sodass die Kommunikation nicht funktionierte. Durch manuelles Schreiben des kompletten Bytes in einer Zuweisung konnte das Problem behoben werden.

6.3 PWM-Störungen

Beim ersten Probelauf mit implementiertem PWM zeigte sich gelegentliches Flackern der LEDs, was als störend empfunden wurde. Um dieses Verhalten zu unterbinden, wurden verschiedene Maßnahmen ergriffen. Zunächst wurde vermutet, dass es daran lag, dass die Funktion *handlePWM()* im Hauptprogramm aufgerufen wurde, welches durch unterschiedliche Verzweigungen und Interrupts über keine konstante Zykluszeit verfügt. Aus diesem Grund wurde das Aufrufen der Funktion *handlePWM()* vom Hauptprogramm in den Interrupt des Zeitgebers verschoben, wodurch das Flackern bereits gemindert, jedoch nicht vollständig verhindert werden konnte. Ein weiteres Problem wurde darin vermutet, dass der Zeitpunkt, in dem die Tastverhältnisse verändert werden, nicht den PWM-Zyklus berücksichtigte (die Werte wurden irgendwo während eines Zyklus geändert). Aus diesem Grund wurden Puffervariablen eingeführt, in die das Hauptprogramm die neuen Tastverhältnisse einträgt. Die Funktion *handlePWM()* überträgt diese nun immer zwischen zwei Zyklen in die tatsächlich verwendeten Variablen. Mit diesen beiden Maßnahmen konnte schließlich erreicht werden, dass die PWM-Generierung störungsfrei erfolgt.

6.4 Padding in Structs

Ein Problem, das erst erkannt wurde, als versucht wurde, zwischen Controller und Interface einen Datensatz zu übermitteln, ist das in Structs angewandte Padding. Das Padding dient dazu, Daten in Structs gleichmäßig anzuordnen. Liegen in einem Struct bspw. eine 16-Bit Variable, eine 8-Bit Variable sowie wieder eine 16-Bit Variable hintereinander, so wird hinter der 8-Bit-Variable ein 8-Bit Leerplatz eingefügt, damit die zweite 16-Bit Variable nicht aus dem Muster fällt, siehe dazu z.B. [AJA10], S. 549.

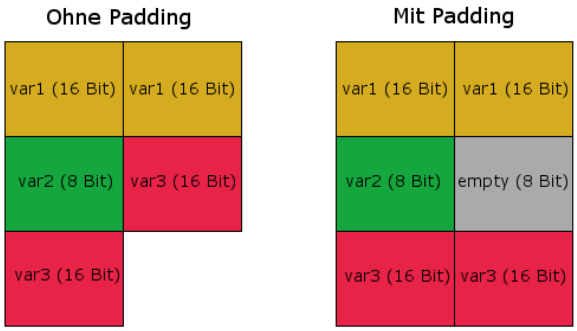

Abbildung 23 Schaubild: Padding in Structs

Jedoch liegt dieses Verhalten nur auf dem Computer vor, nicht jedoch auf dem Mikrocontroller, auf dem das Sparen von Speicherplatz im Vordergrund steht. Dieser Effekt führte dazu, dass das Struct *kombiData* auf dem Computer 10 Bytes größer war, als auf dem Mikrocontroller. Durch das zuvor beschriebene Nachrichtensystem ist dieser Fehler glücklicherweise bereits dadurch aufgefallen, dass die Anzahl der übermittelten Bytes nicht mit der erwarteten übereinstimmte. Andernfalls wären falsche Werte übermittelt worden, was vermutlich schwieriger zu analysieren gewesen wäre. Durch eine genauere Betrachtung von *kombiData* konnte darauf geschlossen werden, dass für die Differenz die 10 *Breakpoints* verantwortlich waren, in welchen nach der 16-Bit Variable *rpm* die drei 8-Bit Variablen für die jeweilige Farbe folgen. Es wird also in jedem *Breakpoint* auf dem Computer ein Leerplatz eingefügt, während der Controller dies nicht tut. Um das Problem zu lösen, wurde einfach eine *dummy* genannte 8-Bit Variable eingefügt. Nun ist *kombiData* auf dem Computer und dem Controller gleich groß und der Austausch funktioniert ohne Störungen.

6.5 Häufiger Wechsel zwischen Effekten

Ein weiteres Problem trat auf, wenn sich die Drehzahl im Grenzbereich zwischen zwei *Breakpoints* oder am Rand des Wirkungsbereichs eines *Dimmers* befand. Durch leichte Schwankungen der erzeugten Drehzahl und bei der Messung kam es dazu, dass der Controller häufig zwischen zwei Effekten hin- und herschaltete, was in einem sichtbaren Flackern resultierte. Aus diesem Grund wurde die bereits beschrieben Hysterese eingeführt, welche vor ihrer korrekten Wirkungsweise jedoch ein neues Problem generierte. Das Problem trat dabei bei der Bestimmung des aktuellen *Breakpoints* auf. Nach Einführung der Hysterese wurde dabei stets überprüft, ob die aktuelle Drehzahl niedriger liegt als die Drehzahl des vorherigen *Breakpoints* abzüglich der Hysterese. Wenn nun der zweite *Breakpoint* aktiv war, wurde also geprüft, ob die aktuelle Drehzahl niedriger liegt als die Drehzahl des ersten *Breakpoints* abzüglich der Hysterese. Die Drehzahl des ersten *Breakpoints* beträgt jedoch 0, sodass die Subtraktion zu einem Unterlauf führte und somit permanent zwischen dem ersten und dem zweiten *Breakpoint* hin- und hergeschaltet wurde. Nachdem die Hysterese so angepasst wurde, dass sie Über- und Unterläufe berücksichtigt, konnte die Funktion der Hysterese zum Tragen kommen und das Flackern durch häufige Wechsel der Effekte wurde beseitigt.

6.6 Schreiben in EEPROM

Beim Schreiben in den EEPROM verursachte ebenso wie schon in Abschnitt 6.2 wieder die Bibliothek *bitOperation* ein Problem. Das Schreiben in den EEPROM sieht nämlich vor, dass nach dem Laden des Bytes in das entsprechende Register zunächst das *Master Write Enable Bit* und anschließend das *Write Enable Bit* auf 1 gesetzt werden muss. Das *Master Write Enable Bit* wird jedoch entweder zurückgesetzt, nachdem ein Schreibvorgang erfolgte, oder aber nach 4 Prozessor-Zyklen (vgl. [ATM13], S. 20f). Der Aufruf der verwendeten Funktion *setBit(...)* jedoch benötigt länger als 4 Prozessor-Zyklen, sodass kein Schreibevorgang stattfinden konnte. Nachdem das Bit manuell gesetzt wurde, funktionierte auch das Schreiben in den EEPROM.

6.7 Serielle Kommunikation unter Linux

Bei der Implementierung der seriellen Kommunikation kam es zu einem Berechtigungsfehler bei dem Versuch, auf den entsprechenden Hardwareport zuzugreifen. Grund dafür ist, dass der Zugriff auf Hardware-Dateien unter Linux Berechtigungen erfordert, die ein normaler Nutzer zunächst nicht hat. Eine schnelle Lösung, um das Problem zu lösen, ist es, das Programm mit Root-Rechten auszuführen, jedoch scheinen Root-Rechte für diesen Zweck als zu großes Zugeständnis an das Programm in Anbetracht der Funktion, die es damit erfüllen soll. Eine weitere Lösung, die deutlich angebrachter erscheint, ist die Tatsache, dass unter Linux die Nutzergruppe *dialout* existiert, welche speziell dazu dient, auf serielle Schnittstellen zuzugreifen. Nachdem man seinen Nutzer dieser Gruppe hinzugefügt hat, kann man ohne Probleme auf die seriellen Schnittstellen zugreifen.

7 Zusammenfassung und Ausblick

Das Hauptziel war es, die Beleuchtung eines Kombiinstruments in Abhängigkeit der Drehzahl farblich zu gestalten, wobei die Abhängigkeit durch einen austauschbaren Datensatz vorgegeben werden sollte. Außerdem sollte mit demselben Controller auch eine Freigabe für die Betätigung des Starterknopfes realisiert werden. Ferner wollte der Autor mit Hilfe des Projektes neue Programmierfähigkeiten erlangen, darunter die softwarebasierte PWM-Generierung, das Messen von Frequenzen bzw. Periodendauern, der Umgang mit dem EEPROM eines Mikrocontrollers sowie die Implementierung einer seriellen Schnittstelle in einem C-Programm am Computer.

Alle diese Ziele konnten erfolgreich umgesetzt werden und es ist ein funktionierendes System entsprechend des zu Beginn entworfenen Konzeptes entstanden, welches den Anforderungen des Autors genügt. Entsprechend der Drehzahl ändert sich die Farbe und sowohl ein stehender Motor, als auch eine Aufforderung, den nächsthöheren Gang zu wählen, werden visuell besonders durch die implementierten *Dimmer* hervorgehoben. Auch die Freigabe des Starterknopfes funktioniert, sodass der Starter sich nur betätigen lässt, wenn der Motor steht. Auf diese Weise werden Schäden durch eine unbeabsichtigte Betätigung vermieden. Der Datensatz, der das Verhalten des Controllers bestimmt, lässt sich am Computer mit Hilfe des entwickelten *Interfaces* manipulieren, abspeichern und auf den Controller übertragen, wobei Wert auf eine gewisse Übertragungssicherheit gelegt wurde, damit der Controller sich nicht unerwartet verhält, falls die vollständig korrekte Datenübertragung fehlschlägt. Die Hardware ist so aufgebaut, dass alle relevanten Komponenten über Steckverbindungen verfügen und so leicht getrennt voneinander ein- bzw. ausgebaut werden können, falls dies nötig wird. Als Nebenprodukt wurde noch ein Frequenzgenerator entwickelt, welcher zum Entwickeln einiger Grundfunktionen Verwendung fand und vor allem die Entwicklung des Systems in einer Testumgebung ermöglichte – so konnte an einem geeigneten Arbeitsplatz agiert werden, während der Motor sowie die Umwelt nicht für Testzwecke belastet werden mussten. Neue Ansätze sowie verschiedene, aufgetretene Probleme haben die Erfahrung des Autors erweitert, was ebenfalls durch das Projekt erreicht werden sollte.

Für die Zukunft stehen bereits einige Ideen bereit, die im Projekt noch umgesetzt werden können: Eine Funktion, die noch realisiert werden soll, ist eine Implementierung der seriellen Kommunikation unter Windows. Eine weitere Funktion, die für die Zukunft angedacht ist, ist das Plotten der Farbwiedergabe direkt auf dem Monitor des Computers, auf dem der Datensatz manipuliert wird. Auf diese Weise bräuchte man für eine Erstabstimmung überhaupt keine Hardware. Eine weitere Idee, die noch aufgekommen ist, ist das Implementieren eines PT1-Gliedes für die Messung der Drehzahl, was die zeitliche Änderungsrate der gemessenen Drehzahl beschränkt. Damit könnte man die Trägheit der Nadel des Drehzahlmessers abbilden, damit die Farbwiedergabe auch dann noch mit der Anzeige übereinstimmt, wenn sich die Drehzahl schneller ändert, als der Drehzahlmesser es anzeigen kann. Ob dieser Fall eintritt, kann noch nicht beurteilt werden, da zu diesem Zeitpunkt noch kein Test im Fahrzeug stattgefunden hat. Sollte dem so sein, so wird diese Funktion noch implementiert.

Zusammenfassend wird das Projekt als erfolgreich abgeschlossen und lehrreich bewertet.

8 Literaturverzeichnis

[ATM13] Atmel Corporation Datenblatt Atmel ATmega8 [Buch]. - San Jose, USA : Atmel Corporation, 2013.

[BRI02] Brinkschulte Uwe und Ungerer Theo Mikrocontroller und Mikroprozessoren [Buch]. - Berlin Heidelberg, Deutschland : Springer, 2002.

[HAG01] Hagmann Gert Grundlagen der Elektrotechnik [Buch]. - Wiebelsheim : Aula-Verlag, 2001.

[HEI] Heinen Patrick VW Golf 3 Typ 1H (95-99) Stromlaufplan [Buch]. - Nettetal, Deutschland : KFZ-Verlag GmbH, 2018.

[KOR06] Korp Dieter Jetzt helfe ich mir selbst [Buch]. - Stuttgart : Motorbuch Verlag, 2006. - Bd. 154.

[AJA10] Mittal Ajay Programming in C: A Practical Approach [Buch]. - Indien : Dorling Kindersley, 2010.

[STM18] STMicroelectronics Positve voltage regulator ICs [Buch]. - [s.l.] : STMicroelectronics, 2018.