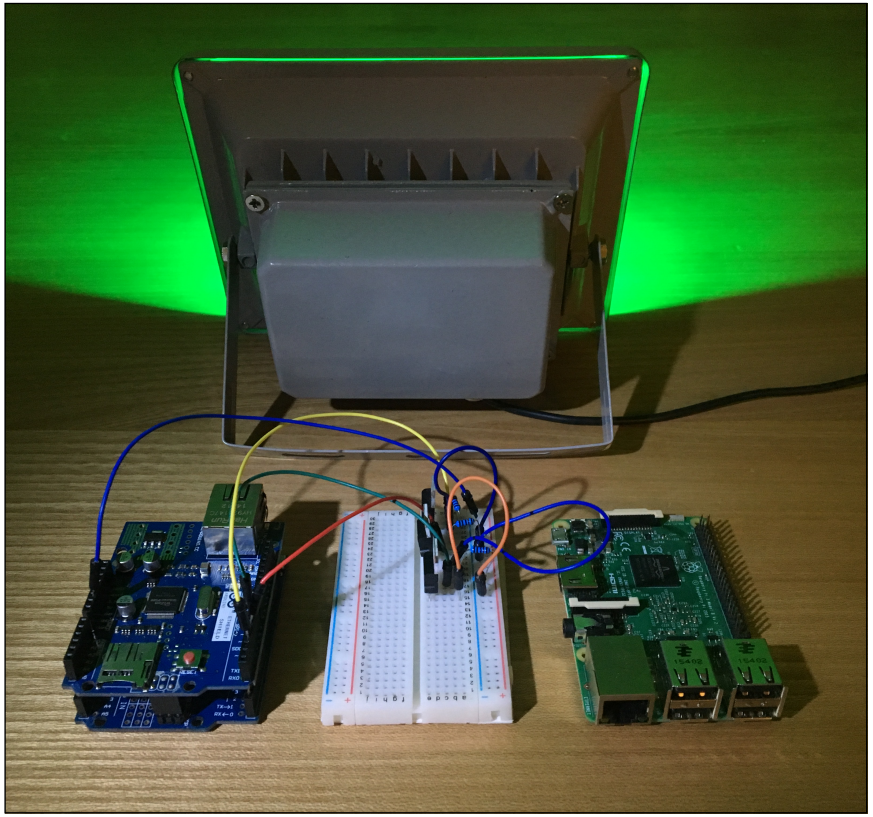
Es werde Licht. Mal so, mal so.



Bau einer Lichtsteuerung mit Arduino und Raspberry Pi

Schriftliche Maturaarbeit von Valentin Huber

Im Fachbereich Informatik

Betreut von Martin Schwarz

Im Dezember 2016

Kantonsschule Schaffhausen

Inhaltsverzeichnis

Fett gedruckte Wörter finden sich im Glossar.

1 Motivation

Elektronik hat mich schon immer fasziniert. Sei es, dass ich in der Unterstufe Stromkreise zusammensteckte und mit Elektromotoren experimentierte, oder dass ich begann, mich immer mehr mit elektronischen Geräten zu beschäftigen. Trotzdem war ich immer auf der Anwenderseite aktiv, habe Reviews von gewissen Produkten gelesen und mich mit Tricks und Kniffs bei der Bedienung meines Computers beschäftigt.
Irgendwann kam in mir der Wunsch auf, auch einmal zu erkunden, wie diese ganzen Geräte funktionieren und vielleicht sogar ein eigenes zu bauen und zu programmieren. Aber ich hatte nie eine konkrete Idee, die mich genug motiviert hätte, damit zu beginnen. Deshalb war für mich bald klar, dass eine Maturaarbeit vielleicht der richtige Anstoss dazu wäre, der auch verhindert, dass ich bald wieder aufgebe, weil mich die Motivation verlässt.
Allerdings kam ich nicht sofort auf ein Thema, das mir gefiel. Mir zu Hilfe kam, dass ich gerade in ein neues Zimmer zog und eine neue Deckenlampe brauchte. Ich hatte mir schon länger überlegt, LED-Glühbirnen zu verwenden, bei denen die Farbe über eine Handy-App gesteuert werden könnte. Allerdings waren mir die Produkte auf dem Markt zu teuer. Deshalb entschied ich mich, das Ganze selber zu bauen.

2 Aufgabenstellung

Die Idee war, irgendein farbiges LED-Leuchtmittel möglichst einfach ansteuerbar zu machen und damit die normale Deckenlampe in meinem Zimmer zu ersetzten. Dabei stellten sich verschiedene Fragen:

- Wie ist die Ansteuerung am einfachsten?
  - Die erste Idee war, das Licht über mein iPhone anzusteuern. Im Nachhinein merkte ich dann, dass das doch keine gute Idee war: Eine iOS-App zu programmieren ist sehr aufwendig und teuer. (Die Programmierung selbst ist gratis, doch eine Verteilung über den App Store, und damit die Möglichkeit der Weitergabe nach Abschluss des Projekts benötigt einen Developer-Account, der 99 US-Dollar im Jahr kostet.)[1] Die Programmierung der Ansteuerung über einen Webbrowser schien einfacher, da dieser schon vorliegt und die Ansteuerung über mehr Geräte möglich wäre. Allerdings dauert es doch einige Zeit, das Smartphone aus der Hosentasche zu nehmen, den Browser zu öffnen, die richtig Web-Adresse einzugeben und dann noch die Werte anzupassen.
  - Später kam noch die Idee dazu, das Licht über verschiedene physische Schalter und Regler zu steuern, die im ganzen Zimmer verteilt sind. Dabei können die verschiedensten Szenarien vorprogrammiert werden (zum Beispiel könnte ein spezieller Knopf an der Lampe neben dem Bett alles Licht ausser der Nachttischlampe ausschalten).
  - Noch einfacher für den Benutzer wäre, die Scheinwerfer automatisch ein- und auszuschalten wenn sie das Zimmer betreten bzw. vetrlassen. Dazu müsste aber jeder Benutzer (ohne dass die Programmierung extrem aufwendig würde) irgendeine Art von Funkmodul mit sich tragen.
- Wie kann man diese Ansteuerung möglichst einfach umsetzen?
  - Dazu gibt es verschiedene Ansätze, ich habe mich für einen serverbasierten entschieden. Dazu aber mehr im Abschnitt 3.
- Was gibt es für Erweiterungsmöglichkeiten?
  - Der grosse Vorteil daran, die ganze Steuerung selbst zu bauen, liegt darin, dass das Produkt auf eigene Bedürfnisse angepasst werden kann. Man kann also jene Erweiterungen einbauen, die man auch wirklich braucht. Dabei gäbe es verschiedenste Möglichkeiten, vor allem die Automatisation betreffend, dazu mehr im Abschnitt 9.

[1] https://developer.apple.com/programs/how-it-works/

3 Ansatz

Diese Ideen lassen sich auf verschiedene Wege umsetzten:
Zuerst muss man sich entscheiden, ob das Ganze zentral oder dezentral funktionieren soll, also mit oder ohne Server. Der dezentrale Weg hätte zwar den Vorteil, dass er viel fehlerresistenter wäre, da er nicht direkt ausfällt, wenn ein Bauteil der Steuerung ausfällt. Allerdings ist dies deutlich aufwendiger und schwieriger zu programmieren, da der Browser, wenn eine neue Eingabe erfolgt, diese an alle beteiligten Output-Clients verteilen muss. Das führt dazu, dass die Input-Seite programmiert werden muss, was den Nachteil hat, dass, wenn ganz verschiedene Geräte den Zustand verändern, der Code auf allen Geräte angepasst werden muss. Das ist sehr aufwändig.
Bei der Lösung mit Server hat man dieses Problem nicht. Dafür wird das Netzwerk viel stärker belastet, da die einzige Möglichkeit für einen Output-Client, an die Daten zu kommen ist, den Server permanent anzupingen und den aktuellen Status abzufragen. Ausserdem verlässt sich das ganze System auf einen Server und fällt in dem Moment komplett aus, in dem der Server nicht mehr verfügbar ist.
Ich habe mich schliesslich für den Ansatz mit Server entschieden, da es den Einstieg einfacher macht, und die einzelnen Erweiterungen unabhängig voneinander programmiert werden können.
Als Server fungiert ein Raspberry Pi, auf diesem läuft ein Webserver (Apache 2) und **PHP**. Ausser den beiden PHP-Skripts für die Ein- und Ausgabe kann über einen **HTTP-Request** auch die Datei *state.txt* direkt abgerufen werden.
Dies ist die zentrale Datei, in der alle Fäden zusammenlaufen. In ihr gespeichert ist eine einzige Zeile mit dem aktuellen Status, also der Sammlung aller Datensätze. Ein Datensatz sieht für eine Kennnummer K und untergeordnete Zahlen a, b und c folgendermassen aus:
#K/a/b/c.
Dabei werden verschiedene Datensätze einfach hintereinander abgespeichert, Trennzeichen sind die Rautezeichen. Die verschiedenen Clients verändern diese Datei oder lesen sie aus.

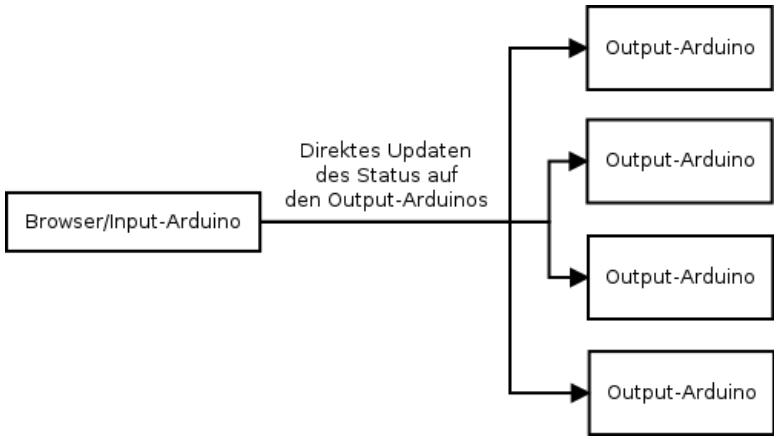


Abb. 2: Dezentralisierter Ansatz

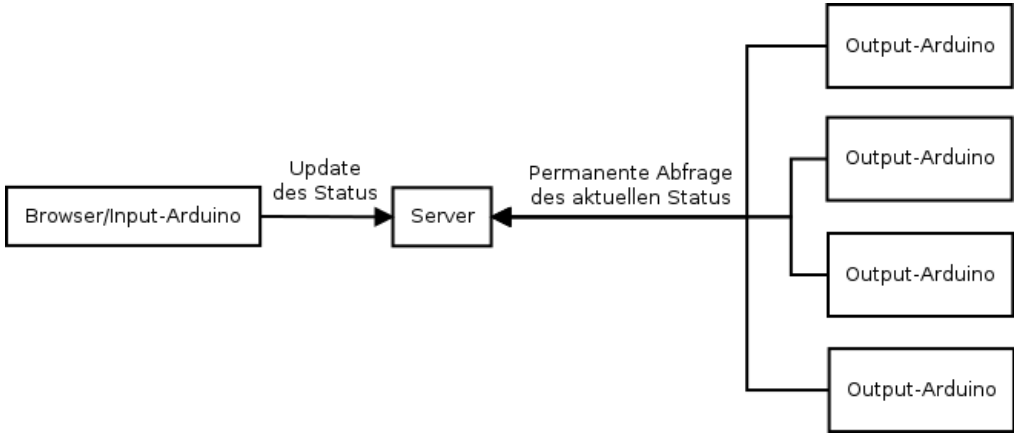


Abb. 3: Serverbasierter Ansatz

4 Bauteile

4.1 Raspberry Pi[2]

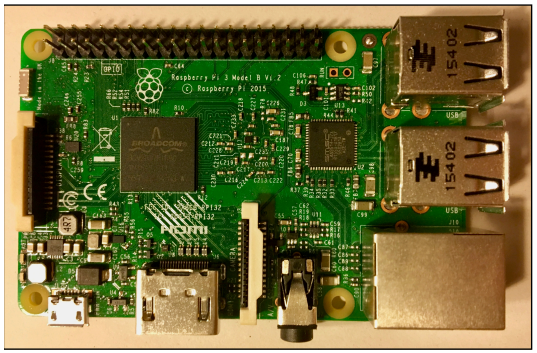


Abb. 4: Raspberry Pi

Der Raspberry Pi ist ein vollwertiger Computer in Grösse einer Kreditkarte. Das heisst, er ist eher leistungsschwach, braucht aber auch nur wenig Strom. Dadurch ist der Raspberry Pi bestens als Server geeignet.
Konfiguriert wird er dabei über eine angeschlossene Tastatur und Maus und einen externen Bildschirm oder über **SSH**. Dafür muss der Raspberry Pi nur am Strom angeschlossen sein und über eine Netzwerkverbindung verfügen. Für einen Server müssen diese beiden Bedingungen grundsätzlich erfüllt sein. Das hat den Vorteil, dass der Raspberry Pi irgendwo im Haus stehen kann, er braucht keinen speziellen Platz. Sobald der Server fertig konfiguriert ist, kann der Raspberry Pi ohne weitere Peripherie weiterlaufen.
Der Raspberry Pi 3, den ich verwende, ist ausgestattet mit einer 1.2GHz ARMv8 CPU, 1 GB RAM, 4 USB-Ports, 40 GPIO-Ports, je einem HDMI- und Ethernetstecker, einem 3.5mm-Klinkenstecker, und einem Slot für eine Micro-SD-Karte, auf welche das Betriebssystem gespeichert wird. Ausserdem besitzt der Raspberry Pi Chips für die Funkstandards 802.11n WLAN, Bluetooth 4.1 und Bluetooth Low Energy.

4.2 Arduino Uno[3]

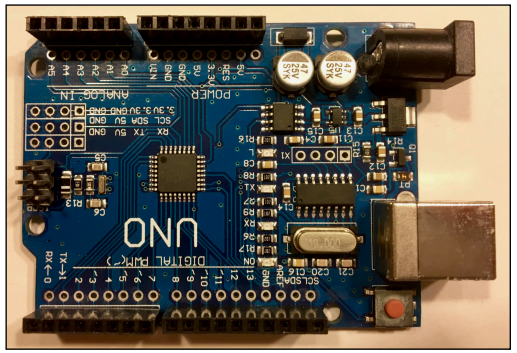


Abb. 5: Arduino Uno

Der Arduino Uno (bzw. ausserhalb der USA Genuino) ist ein Mikrocontroller auf Basis des Chips ATmega328P. Er besitzt 14 digitale Input-/Output-Pins, wovon 6 die sogenannte **Pulsweiten-Modulation** (kurz PWM) beherrschen. Neben diesen befinden sich auf dem Bord sechs analoge Inputs und ein USB-Port, um Daten zu übertragen. Ausserdem findet sich ein Stromanschluss, ein sogenannter ICSP-Header, um zum Beispiel die Firmware auf dem Chip zu aktualisieren, und ein Reset-Knopf.

[2] https://www.raspberrypi.org/products/raspberry-pi-3-model-b

[3] https://www.arduino.cc/en/Main/ArduinoBoardUno

4.3 Ethernet-Shield V1[4]

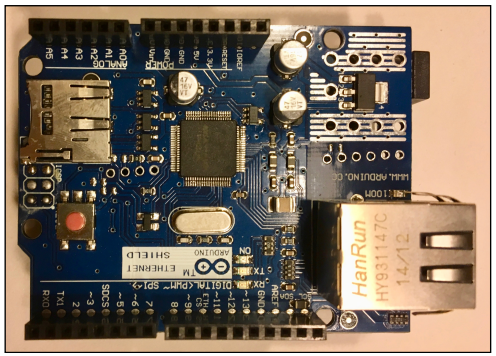


Abb. 6: Ethernet-Shield

Auf dem Arduino steckt ein sogenanntes Ethernet-Shield. Dieses steckt auf den Ports des Arduinos, belegt jedoch nur wenige davon (die sogenannten SPI-Ports), um mit dem Arduino zu kommunizieren. Die anderen stehen weiterhin zur Verfügung; das ist die grundsätzliche Funktionsweise eines Shields. Die Kommunikation wird durch eine **Library** deutlich vereinfacht.
Das Shield selbst besitzt einen W5100-Ethernet-Controller mit einem internen 16kB-Buffer. Es kommuniziert mit der Aussenwelt über ein Ethernet-Kabel. Das Shield beherrscht einen Datendurchsatz von höchstens 100 MB/s. Das ist jedoch kein Problem für mein Projekt, da nur sehr wenige Daten transportiert werden müssen und der Arduino für einen höheren Datendurchsatz zu langsam wäre.
Ausserdem bietet das Shield die Möglichkeit, eine SD-Karte anzusteuern. Das werde ich in meinem Projekt aber nicht brauchen.

4.4 Transistor TIP120[5]

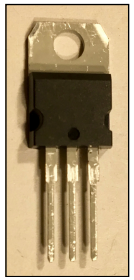


Abb. 7: TIP120

Da der Arduino nur auf 5V-Basis läuft, das Leuchtmittel aber meist eine höhere Spannung verlangt, wird ein Transistor in die Schaltung integriert. Ein Transistor funktioniert mit seinen 3 Pins wie ein automatischer Schalter: Wenn auf einem Pin eine Spannung herrscht, wird der andere Stromkreis ebenfalls geschlossen. Das kann mehrere tausend Mal pro Sekunde geschehen. Ein PWM-Signal kann deshalb ohne Probleme auf einen Stromkreis mit höherer Spannung übertragen werden. Damit wird vom Arduino eine variable Spannung auf einer anderen Skala ausgegeben.

4.5 LED-Scheinwerfer


Abb. 88: LED-Scheinwerfer

Das Leuchtmittel musste verschiedenen Kriterien entsprechen: Es sollte nicht zu teuer, hell und einfach ansteuerbar sein und alle Farben (RGB) ausstrahlen können. Da ich eine indirekte Beleuchtung bevorzuge, habe ich mich schliesslich für LED-Scheinwerfer entschieden. Diese werden mit einer Infrarot-Fernbedienung geliefert. Da diese aber nicht allzu zuverlässig und schwierig ansteuerbar ist, habe ich die Elektronik ausgebaut und den Scheinwerfer mit einer eigenen Ansteuerung versehen (siehe 6.4).

4.6 Potentiometer

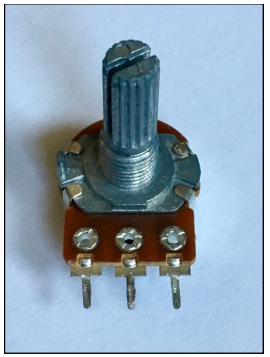


Abb. 9: Potentiometer

Ein Potentiometer ist ein variabler Widerstand. Damit kann er, sofern auf zwei seiner drei Pins eine Spannung liegt, auf den dritten eine variable Spannung ausgeben. Diese Spannung kann von einem Arduino ausgelesen und weiterverarbeitet werden.

[4] https://www.arduino.cc/en/Main/ArduinoEthernetShieldV1
[5] http://www.mikrocontroller.net/part/TIP120

5 Ansteuerung via Webbrowser auf dem Raspberry Pi

5.1 Kommentar zum Code

Dieser Code ist ein typisches Beispiel für ein PHP-Skript, das sich selbst aufruft, um gewisse Funktionen durchzuführen. Dabei wird zuerst die Seite geladen, angezeigt, und auf Befehl des Users neu geladen, wobei eine gewisse Aktion in Kraft tritt.
Am Anfang des Codes findet sich deshalb ein PHP-Skript, das zum Zweck hat, die über das **POST-Protokoll** übertragenen Daten in die Datei *state.txt* zu schreiben. Dabei werden diese Daten zuerst in einer Variable gespeichert. In diesem **String** befinden sich noch die Zeilenumbrüche. Diese sollen aber nicht in die Datei gespeichert werden und müssen deshalb zuerst herausgefiltert werden. Danach kann die Datei überschrieben werden.

Jetzt beginnt im Code die eigentliche **HTML**-Seite. In dieser befindet sich ein Formular, das ein Textfeld und einen Knopf enthält. Um den Inhalt des Textfeldes zu ermitteln, läuft zuerst ein PHP-Skript, das die Daten in der Datei *state.txt* ausliest, Zeilenumbrüche vor jedem Rautezeichen einfügt und das Resultat an den Browser weiterleitet.
Beim Betätigen des Knopfes wird die Datei *write.php* aufgerufen und ausgeführt. Weil dies dieselbe Datei ist, beginnt der Code wieder von vorne.

5.2 Variablen

Name	Funktion
$file_r	Handle, an den die Datei state.txt gebunden wird; für das Auslesen dieser verwendet
$file_w	Handle, an den die Datei state.txt gebunden wird; für das Überschreiben dieser verwendet
$post	Enthält Daten, die via POST-Protokoll mitgegeben wurden
$C	Enthält den aktuellen Inhalt von state.txt

- C für content
- _r für read
- _w für write

5.3 Code

```php
<?php
  if (isset($_POST['cont'])) { //Verhindern, dass die Datei beim ersten Aufrufen überschrieben wird, wenn noch keine POST-Variable definiert ist.
    $post = $_POST['cont'];
    $post = preg_replace("/\r|\n/", "", $post); //preg_replace(String1, String2, Variable) ersetzt String1 mit String2 in der Variable.
    $file_w = fopen('state.txt', 'w'); //fopen(Datei, Zugriffsart) bindet eine Datei mit der Zugriffsart (w für write) an eine Variable.
    fwrite($file_w, $post); //fwrite(Datei, String) schreibt den String in die Datei.
    fclose($file_w); //Die Verbindung lösen.
  }
?>
  <!DOCTYPE html>
  <html>
  <body>
    <form action="write.php" method="post"> //Ein Formular öffnen, dass bei Betätigung des Submit-Buttons die Datei write.php aufruft und dabei Daten via POST-Protokoll überträgt.
      <textarea rows="10" name="cont"> //Ein Textfeld mit dem Namen cont und 10 Zeilen öffnen.
        <?php
          $file_r = fopen("state.txt", "r");
          $C = fread($file_r, 1000); //Die ersten 1000 Stellen der Datei in Variable $C speichern.
          $C = str_replace("#", "&#10;#", $C); //Vor jedem Rautezeichen einen Zeilenumbruch einfügen.
          fclose($file_r);
          echo $C; //Den Inhalt der Variable $C an das HTML-Dokument geben.
        ?>
      </textarea> //Das Textfeld schliessen.
      <br> //Einen Zeilenumbruch einfügen.
      <input type="submit" value="save"> //Den Submit-Button einfügen.
    </form> //Das Formular schliessen.
  </body>
  </html>
```

6 Ansteuerung des LED-Scheinwerfers

6.1 Kommentar zum Code

Nach dem Definieren einiger Variablen und dem Initialisieren von Libraries werden die benötigten Ports als Outputs definiert und das Ethernet-Shield initialisiert.
Im Loop angekommen, soll als erstes eine Verbindung mit dem Server aufgebaut werden. Wenn das gelingt, wird der **HTTP-GET-Request** abgesendet und kurz auf den Server gewartet. Gelingt die Verbindung nicht, wird eine Sekunde gewartet und durch den Befehl *return* die Funktion *loop* neu gestartet, weil es keinen Sinn mehr hat, den Rest des Programms auszuführen.
Danach wird noch einmal überprüft, ob die Verbindung mit dem Server noch besteht. Ist dies der Fall, wird überprüft, ob sich Daten im Buffer befinden (die der Server an den Arduino senden will). Die allfälligen Daten werden **char** für char übertragen und dann im **Array** *S[]* gespeichert. Danach wird die Verbindung zum Server geschlossen. Die Daten sind jetzt noch in keinem verwertbaren Format und müssen deshalb erst weiterverarbeitet werden.
Dieses Array wird jetzt nach dem Anfang und Schluss des fraglichen Datensatzes durchsucht. Diese werden abgespeichert. Damit ist auch die Länge des fraglichen Datensatzes klar und das Array *thisS[]* kann definiert werden.
Damit kann nun der fragliche Datensatz aus *S[]* extrahiert und in *thisS[]* gespeichert werden. Ausserdem werden auch die Anzahl Slashes gezählt, die dieser Datensatz enthält. Dabei gibt es immer einen Slash weniger als es Zahlen gibt. Diese Anzahl Zahlen wird auch gespeichert.
Dank dem Wissen über die Anzahl Zahlen im Datensatz können nun die Anfangsstellen aller Zahlen bestimmt werden.
Jetzt wird, Zahl für Zahl, der Teil, welcher der fraglichen Zahl entspricht, aus *thisS[]* in ein Hilfs-Array kopiert. Dieses kann nun in eine ganze Zahl umgewandelt werden (aus einzelnen chars als Stellen zu einer mehrstelligen Zahl). Diese Zahlen werden abgespeichert, sie sind nun in einem Zustand, der die Ausgabe ermöglicht.
Die Ausgabe geschieht ebenfalls Zahl für Zahl via PWM an den am Anfang des Codes definierten Pins.
Danach werden einige Variablen wieder auf 0 gesetzt, weil diese im sonstigen Code nur grösser werden, bei jedem Durchlauf aber wieder bei 0 beginnen sollten.
Ausserdem wird noch ein Delay eingefügt, um den Server und das Netzwerk zu entlasten. Ohne dieses würde der Arduino das Programm jede Sekunde sehr oft ausführen.

6.2 Variablen

Name	Typ	Beschreibung
firstdigit	int	erste Ziffer, die zu thisS[] gehören soll
mac[]	byte	mac-Adresse des Ethernet-Shields
Ncombine	int	Counter für die Zahl, die gerade zusammengesetzt wird
nextH	int	Stelle in S[], die dem Rautezeichen des auf thisS[] folgenden Datensatzes gehört
Nofdigits[]	int	Enthält pro ganze Zahl in thisS[] die Anzahl Stellen dieser Zahl
NofNs	byte	Anzahl Zahlen
outputpins[]	byte	Welche Pins als Output verwendet werden sollen
readserver_c	int	Zählt mit beim Lesen der Daten vom Server, ist danach die Länge von S[]
S_l	int	Länge von S[], wird definiert, kann nicht automatisch bestimmt werden
S[]	char	der ganze Status auf dem Server
start_c	int	Counter, der beim Bestimmen von startpositions benötigt wird
startend_c	int	Counter zum bestimmen von firstdigit und nextH
startpositions[]	int	Stellen in thisS[], die dem char vor der ersten Ziffer der Zahlen entsprechen
thisclient	char	Nummer des Clients
thisN_c	int	Counter, der beim übertragen der aktuellen Zahl in thisN[] benötigt wird
thisN[]	char	Zwischenspeicher für die Stellen der aktuellen Zahl
thisS_c	byte	Counter, beim Extrahieren von thisS[] aus S[] benötigt wird
thisS_l	byte	Länge von thisS[]
thisS[]	char	Enthält Datensatz, den dieser Scheinwerfer anzeigen soll
thisstart_c	int	Counter, der beim Bestimmen von startpositions benötigt wird
wholeNs[]	int	Enthält die ganzen Zahlen des Datensatzes

- H für hashtag
- N für number
- S für state
- _c für counter
- _l für length

6.3 Code

```
const int S_l = 200;
byte outputpins[] = { 3, 5, 6 };
#define thisclient '1'

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte server[] = { 192, 168, 1, 69 };

#include <Ethernet.h> //Library für das Ethernet-Shield
#include <SPI.h> //Library für das Ethernet-Shield
EthernetClient client; //Versetzt das Ethernet-Shield in den Client-Modus.

char S[S_l];
int readserver_c = 0;
int firstdigit = 0;
int nextH = 0;
byte thisS_l;
byte NofNs = 0;

void setup() {
  for (byte i = 0; i < sizeof(outputpins); i++) pinMode(outputpins[i], OUTPUT); //Legt die benötigten
    Ports als Outputs fest
  Ethernet.begin(mac); //Das Ethernet-Shield initialisieren.
}
```

```
void loop() {

//Mit dem Server verbinden, HTTP-Request senden
  if (client.connect(server, 80)) { //Überprüfen, ob eine Verbindung mit dem Server möglich ist.
    client.println("GET /state.txt"); //Einen HTTP-GET-Request an den Server senden, der als Antwort den
    Inhalt der Datei State.txt verlangt.
    client.println(); //Den Request mit einer Leerzeile abschliessen.
  } else {
    delay(1000); //Eine Pause machen, wenn der Server nicht verfügbar ist.
    return; //Durch das Statement return wird void loop() abgebrochen und neu gestartet.
  }
  delay(100); //Dem Server etwas Zeit geben, auf den Request zu reagieren.

//Den Server auslesen
  while (client.connected()) { //Solange die Verbindung mit dem Server steht,
    while (client.available()) {//Und der Server eine Antwort bereithält, kann diese ausgelesen werden.
  S[readserver_c] = client.read(); //Die Antwort Zeichen für Zeichen in S[] zwischenspeichern.
  readserver_c++; //readserver_c wird um 1 grösser gemacht, damit beim nächsten Durchlauf die nächste
    Stelle im Array beschrieben wird.
    }
  }
  client.stop(); //Die Verbindung zum Server schliessen.
```

```
//firstdigit und nextH berechnen
  for (int startend_c = 0; startend_c < readserver_c; startend_c++) {
    if (S[startend_c] == '#' && nextH == 0) { //Solange nextH noch nicht belegt ist, wird nach einem
    Rautezeichen gesucht.
  if (S[startend_c + 1] == thisclient) { //Wenn der darauf folgende char die Nummer des Clients ist, ist
    der Beginn des Datensatzes erreicht.
    firstdigit = startend_c + 3;
  } else if ((S[startend_c + 1] != thisclient) && (firstdigit != 0)) { //Falls firstdigit belegt ist,
    aber die Zahl nach dem Rautezeichen nicht der Nummer des Clients entspricht, muss das das nächste
    Rautezeichen sein.
    nextH = startend_c + 1;
  }
    }
  }
  if (nextH == 0) nextH = readserver_c; //Weil am Schluss von S[] kein Rautezeichen steht, muss, sofern
    nextH noch nicht bekannt ist, der fragliche Datensatz der letzte sein.
  thisS_l = nextH - firstdigit;
  char thisS[thisS_l]; //Da jetzt die Länge von thisS[] (thisS_l) bekannt ist, kann thisS[] jetzt
    definiert werden.


//Extrahieren von thisS[] aus S[]
  for (byte thisS_c = 0; thisS_c < thisS_l; thisS_c++) {
    thisS[thisS_c] = S[thisS_c + firstdigit]; //Die chars aus S[] an die entsprechende Stelle in thisS[]
    schreiben.
    if (S[thisS_c + firstdigit] == '/') NofNs++; //Die Anzahl Slashes zählen.
  }
  NofNs++; //Die Anzahl Slashes noch einmal um 1 erhöhen, weil es eine Zahl mehr gibt als Slashes.
```

```
//startpositions ausrechnen
  byte start_c = 0;
  byte startpositions[NofNs + 2];
  for (byte thisstart_c = 0; thisstart_c < NofNs; thisstart_c++) {
    while (thisS[start_c] != '/' && thisS[start_c] != '#') { //Solange der aktuelle char weder ein Slash
    noch ein Rautezeichen ist, ist er uninteressant.
  start_c++;
    }
    startpositions[thisstart_c + 1] = start_c; //Wenn der aktuelle char ein Slash oder Rautezeichen ist,
    ist die richtige Position gefunden und wird abgespeichert.
    start_c++;
  }
  startpositions[0] = 0 - 1; //Die erste startpositions auf -1 setzen, weil die Schlaufen das nicht
    machen.


//In ganze Zahlen umrechnen
  byte Nofdigits[NofNs + 1];
  int wholeNs[NofNs + 1];
  for (byte Ncombine = 0; Ncombine < NofNs; Ncombine++) {
    Nofdigits[Ncombine] = startpositions[Ncombine + 1] - startpositions[Ncombine] - 1; //Die Länge der
    aktuellen Zahl als Differenz der Startwerte der nächsten und der aktuellen Zahl bestimmen.
    char thisN[Nofdigits[Ncombine] + 1];
    for (byte thisN_c = 0; thisN_c + startpositions[Ncombine] + 1 < startpositions[Ncombine + 1];
    thisN_c++) { //Durch die Zahl gehen und alle Stellen in thisN[] schreiben.
 thisN[thisN_c] = thisS[thisN_c + startpositions[Ncombine] + 1];
    }
    wholeNs[Ncombine] = atoi(thisN); //atoi(array) führt die einzelnen Elemente eines Arrays zu einer
    Zahl zusammen.
  }
```

```
//Ausgabe
  for (byte i = 0; i < sizeof(outputpins); i++) { //sizeof(Array) gibt die Menge an Bytes zurück, die
    dieses Array belegt (In diesem Fall die Anzahl Zahlen in diesem Array, weil outputpins[] bytes
    enthält).
    analogWrite(outputpins[i], wholeNs[i]); //analogWrite(Pin, Zahl) gibt via PWM eine Spannung zwischen
    0 und 5 V, proportional zur Zahl, aus (0 entspricht 0V, 255 5V).
  }


//resets     //Da diese Zahlen nicht jedes Mal neu definiert werden, müssen sie auf 0 gesetzt werden.
  NofNs = 0;
  firstdigit = 0;
  nextH = 0;
  readserver_c = 0;


//beenden
  delay(1000); //Ein Delay, um den Server bzw. das Netzwerk zu entlasten.
}
```

6.4 Schaltplan

Von den Arduino-Pins 3, 5 und 6 (PWM-fähige Pins) führt jeweils ein Stromkreis durch ein TIP120-Transistor zur gemeinsamen Masse. Ein jeweils paralleler Stromkreis geht von der externen Stromversorgung durch die Transistoren und das Leuchtmittel zur Masse. Ausserdem steckt ein **Pulldown-Widerstand** zwischen Output-Pin des Arduinos und Masse.

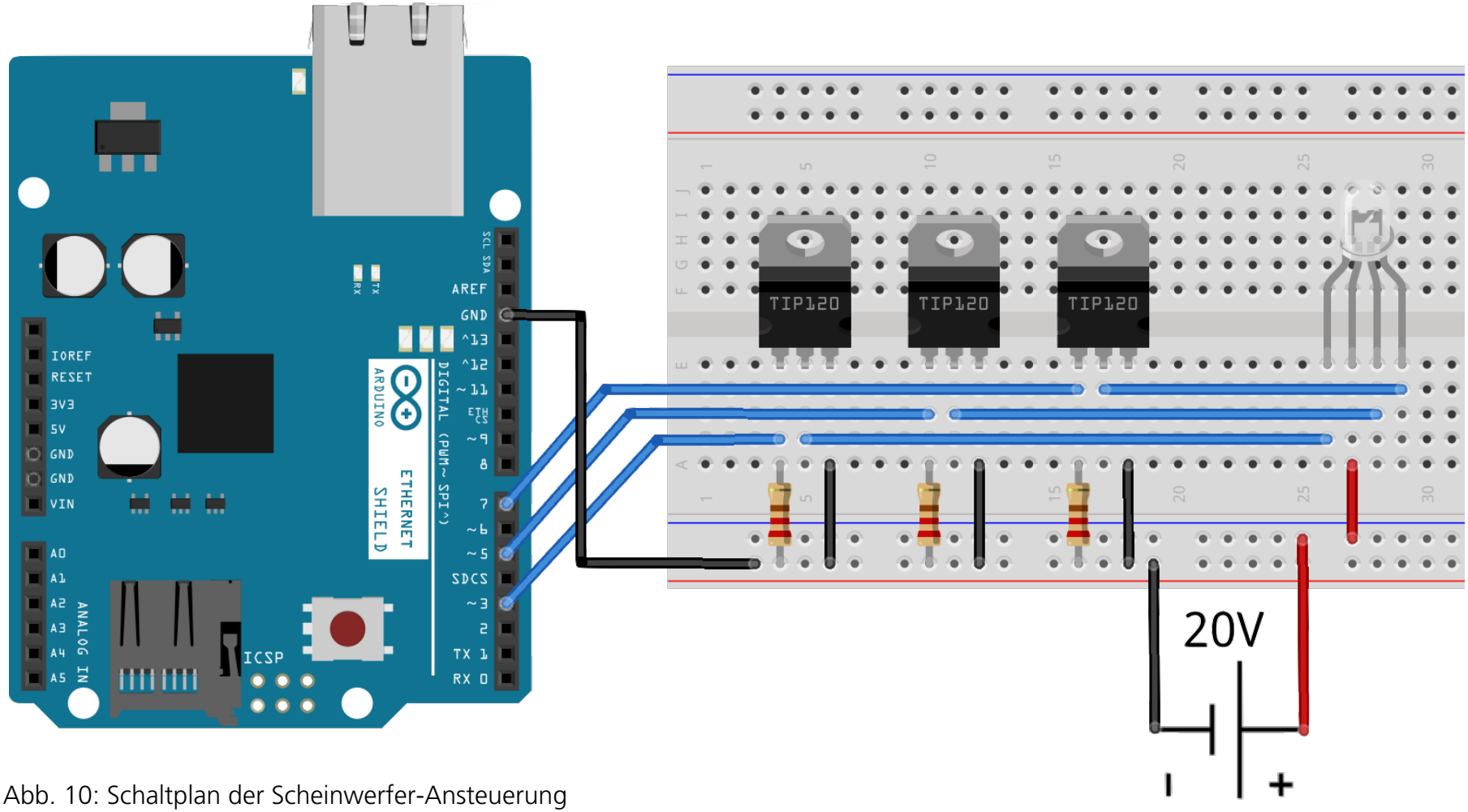


Abb. 10: Schaltplan der Scheinwerfer-Ansteuerung

7 Ansteuerung via Arduino

7.1 Kommentar zum PHP-Code

Dieser Code soll die Teile aus dem Inhalt von *state.txt* mit neuen austauschen, die per GET-Protokoll mitgeliefert werden.
Dazu wird zuerst die Datei ausgelesen und der Inhalt in einer Variablen gespeichert. Dann wird gezählt, wie viele Rautezeichen im Code vorhanden sind, wie viele Datensätze aktuell also vorhanden sind. Jetzt wird für jeden Datensatz überprüft, ob eine aktualisierte Variante davon verfügbar ist. Wenn das der Fall ist, wird diese Variante zu *$newS* hinzugefügt.

Wenn kein neuer Datensatz mitgeliefert wurde, muss der alte aus dem ganzen Inhalt der Datei ausgeschnitten werden. Dafür werden zuerst alle Datensätze vor dem fraglichen weggeschnitten, anschliessend die nachfolgenden. Der resultierende **String** wird dann auch zu *$newS* hinzugefügt.
Zum Schluss wird der Inhalt von *state.txt* durch *$newS* ersetzt.

7.2 Variablen

Name	Funktion
$C	Enthält alle alten Datensätze
$file_r	Handle, an den die Datei state.txt gebunden wird, für das Auslesen dieser verwendet
$file_w	Handle, an den die Datei state.txt gebunden wird. Für das Überschreiben dieser verwendet.
$HandN	Enthält ein Rautezeichen und die aktuelle bzw. die nächste Zahl
$newC	Enthält alle neuen Datensätze
$NofHs	Anzahl Rautezeichen, also Anzahl Datensätze in $C
$oldC	$C ohne die Datensätze vor dem aktuellen

- C für content
- H für hashtag
- N für number
- _r für read
- _w für write

7.3 PHP-Code

```php
<?php
  $file_r = fopen("state.txt", "r"); //fopen(Datei, Zugriffsart) bindet eine Datei mit der Zugriffsart (in diesem Fall r für read) an eine Variable.
  $C = fread($file_r, 1000); //Die ersten 1000 Stellen der Datei in der Variable $C speichern.
  fclose($file_r); //Die Verbindung lösen.
  $newC = ""; //newC definieren.
  $NofHs = substr_count($C,"#"); //substr_count(String1, String2) gibt zurück, wie oft String1 in String2 vorkommen.
  $C .= "#"; //Variable .= String fügt der Variable den String am Ende an.
  $C .= $NofHs; //Das Ende von $C definieren (durch das nächste Rautezeichen und die nächste Zahl).
  for ($i = 0; $i <= $NofHs; $i++) {
    if (isset($_GET[$i])) { //isset(Variable) überprüft, ob eine Variable definiert ist.
      $newC .= "#";
      $newC .= $i;
      $newC .= "/";
      $newC .= $_GET[$i]; //$newC den mit der $_GET-Methode mitgegebenen Wert anfügen.
    }
    else {
      $HandN = "#" . $i;
      $oldC = stristr($C, $HandN); //stristr(String1, String2) gibt den Rest von String1 nach dem ersten Vorkommen von String2 inklusive String2 zurück.
      $j = $i + 1;
      $HandN = "#" . $j;
      $oldC = stristr($oldC, $HandN, true); //Falls das dritte Argument true ist (default = false), wird der Teil vor dem ersten Vorkommen ohne dieses zurückgegeben.
      $newC .= $oldC;
    }
  }
  $file_w = fopen("state.txt", "w");
  fwrite($file_w, $newC); //fwrite(Datei, String) ersetzt den Inhalt der Datei mit dem String.
  fclose($file_w);
?>
```

7.4 Kommentar zum Arduino-Code

Nach dem Definieren einiger Variablen und dem Initialisieren von Libraries wird das Ethernet-Shield initialisiert.
Danach durchläuft der Arduino eine Schleife, in der permanent überprüft wird, ob der Knopf, der an Input A0 angeschlossen ist, gedrückt ist. Falls das der Fall ist, wird der benötigte HTTP-GET-Request zusammengesetzt. Dabei wird zuerst die URL und dann die Kennnummer des betreffenden Datensatzes angefügt. In einer Schleife werden die Pins ausgelesen, ins richtige Format (Zahlen zwischen 0 und 255, nicht zwischen 0 und 1023) gebracht. Ausserdem werden sie mit einem Slash voneinander getrennt und dem Request angefügt.
Darauf wird versucht, eine Verbindung zum Server aufzubauen. Wenn das gelingt, wird der Request übertragen. Ein kurzes Delay verhindert, dass der Server während der Dauer des Klicks permanent aufgerufen wird.

7.5 Variablen

Name	Typ	Beschreibung
changeHN	byte	Der Datensatz, der verändert werden soll • H für hashtag • N für number
Nofinputpins	byte	Die Anzahl Zahlen, die diesem Datensatz zuzuordnen sind
mac[]	byte	mac-Adresse des Ethernet-Shields
server[]	byte	IP-Adresse des Servers
request	String	Der HTTP-Request wird in diesem String zusammengefügt.
inputN	byte	Counter, der beim Anfügen der verschiedenen Inputs an request benötigt wird

7.6 Arduino-Code

```
byte changeHN = 1;
byte Nofinputpins = 3;
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte server[] = { 192, 168, 1, 69 };
#include <Ethernet.h> //Library für das Ethernet-Shield.
#include <SPI.h> //Library für das Ethernet-Shield
EthernetClient client; //Versetzt das Ethernet-Shield in den Client-Modus.
void setup() {
  Ethernet.begin(mac); //Das Ethernet-Shield initialisieren.
  delay(1000); //Das Ethernet-Shield starten lassen.
}

void loop() {
  if (analogRead(0) < 500) {
    String request = "GET /writearduino.php?"; //Den Anfang des GET-Befehls in content speichern.
    request += changeHN; //Die Nummer des zu ändernden Datensatzes zum request hinzufügen.
    request += "="; //Das "=" zeigt dem Server, dass der Inhalt der zuvor genannten Variable folgt.
    for (byte inputN = 1; inputN <= Nofinputpins; inputN++) {
      request += analogRead(inputN) / 4; //analogRead(pin) gibt eine Zahl zwischen 0 und 1023 aus,
    abhängig von der angelegten Spannung.
      request += "%2F"; //"%2F" entspricht einem Slash in einem HTTP-Request.
    }
    request += analogRead(inputN) / 4;

    if (client.connect(server, 80)) { //Überprüfen, ob eine Verbindung mit dem Server möglich ist.
      client.println(request); //Den Request an den Server übertragen.
      client.println(); //Der Request mit einer Leerzeile abschliessen.
    }
    client.stop(); //Die Verbindung mit dem Server schliessen.
    delay(1000); //Verhindern, dass während der Klickdauer der Server permanent aufgerufen wird.
  }
}
```

7.7 Schaltplan

Die Plus- und Minus-Pole der Potentiometer werden mit den entsprechenden Ports des Arduinos verbunden. Ihre mittleren Pins werden mit den Input-Pins des Arduinos verbunden. Der Knopf wird zwischen Pin A0 und Plus-Pin geschalten, mit einem Pulldown-Widerstand zur Masse.

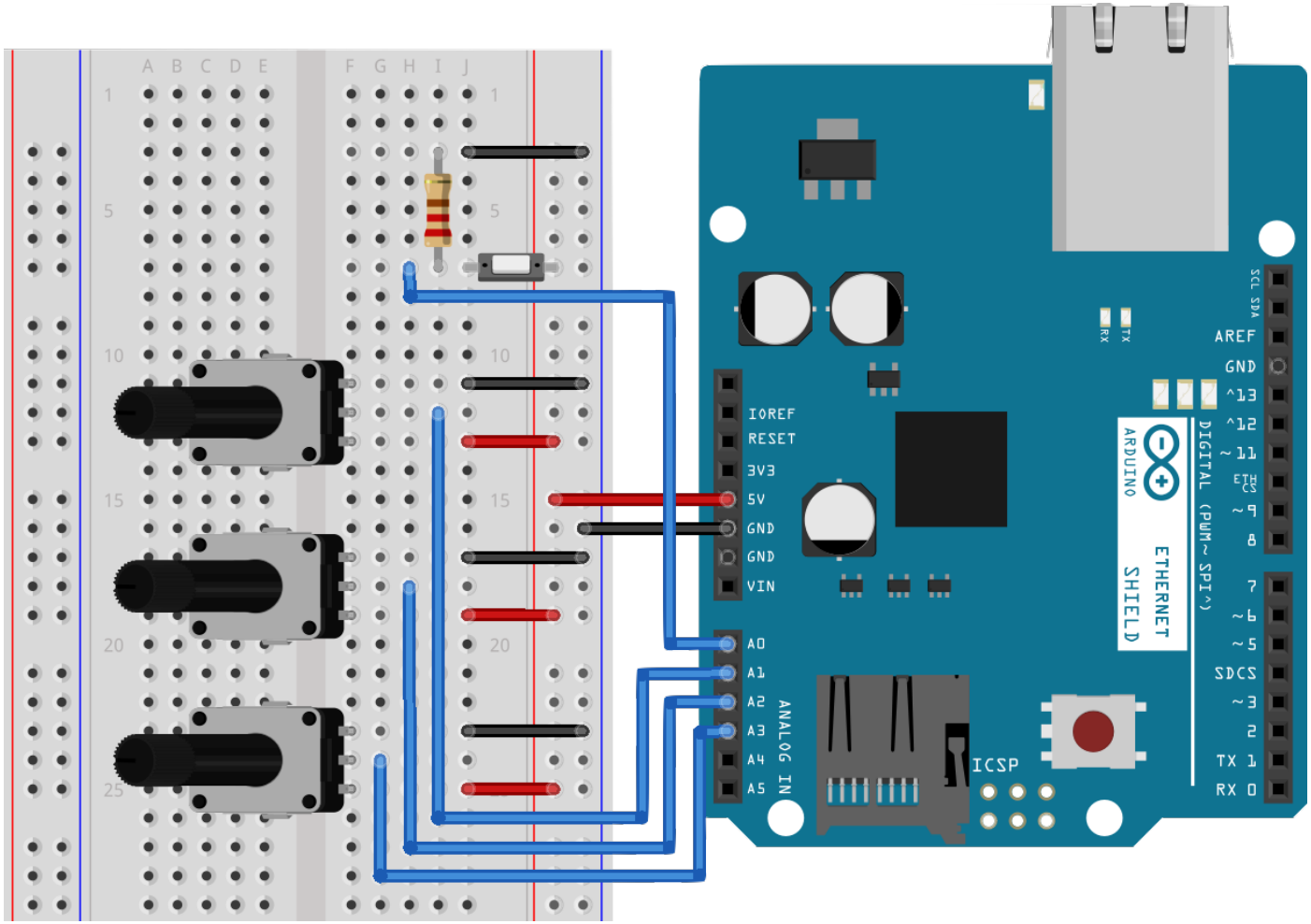


Abb. 11: Schaltplan der Ansteuerung über Arduino

8 Erweiterungsmöglichkeiten

8.1 WLAN

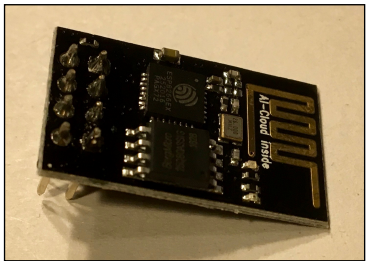


Abb. 12: ESP8266 ESP-01

Im aktuellen Stadium des Projektes muss zu jedem Arduino ein Ethernet-Kabel gezogen werden, was sehr aufwändig sein kann. Ausserdem wird ein Ethernet-Switch benötigt. Das Ganze liesse sich deutlich vereinfachen, wenn ein WLAN-Modul eingesetzt werden könnte. Dafür gibt es verschiedene Möglichkeiten. Es gibt, ähnlich wie das Ethernet-Shield, auch ein WLAN-Shield, welches aber relativ teuer ist. Billiger, kleiner und universaler einsetzbar (dank externer Antenne) wäre ein dediziertes WLAN-Modul, wie zum Beispiel das ESP8266 ESP-01.

8.2 Automatisation

Ausserdem muss aktuell jedes Mal ein Knopf gedrückt werden, wenn das Licht ein- bzw. ausgeschalten werden soll. Einfacher wäre es, irgendwie herauszufinden, wann jemand das Zimmer betritt, bzw. verlässt, und das Licht automatisch danach zu schalten. Das ist aber relativ kompliziert und aufwändig und sprengt den Rahmen meiner Maturaarbeit.

8.3 Wecker

Eine weitere angenehme Funktion wäre ein Wecker, der zu einer bestimmten Zeit das Licht einschaltet. Dazu müsste ein Arduino die Zeit über eine Internet-API bestimmen, diese mit der Weckzeit vergleichen, und zum richtigen Zeitpunkt das Licht einschalten. Dieses Einschalten könnte auch ein benutzerdefiniertes Einblenden sein oder verschiedene Stadien beinhalten.

8.4 Einbindung externer Dienste

Die Einbindung externer Dienste, wie IFTTT (www.ifttt.com, ifttt für «if this than that»), ein Automatisations-Dienst, der verschiedene andere Dienste verbindet, würde ganz neue Wege der Ansteuerung öffnen, weil auf die verschiedensten Ereignisse reagiert werden kann. So kann dieser Dienst auf die Position meines iPhones zugreifen und das Licht einschalten, sobald ich nach Hause komme.

8.5 Einbindung anderer Funktionen

Durch die universale Struktur des ganzen Projektes ist dieses nicht auf die Lichtsteuerung limitiert. Auch ein automatisches Öffnen und Schliessen von Vorhängen oder die Steuerung der Heizung wäre denkbar.

9 Fazit

Im aktuellen Stadium ist das Ergebnis meiner Arbeit nur im kleinen Rahmen alltagstauglich, weil zu jedem Arduino ein Ethernet-Kabel gezogen werden muss. Ich werde wohl noch einige Zeit investieren, den Arduino über WLAN mit dem Netzwerk zu verbinden (siehe 8.1).
Ich persönlich habe beim Erarbeiten der Kenntnisse und beim Schreiben des Codes und der Arbeit sehr viel gelernt. Ich habe zum ersten Mal Kabel zusammengelötet, mit Code experimentiert und mich mit Netzwerken und ihren Funktionsweisen beschäftigt. Ich habe gelernt, was Pulsweiten-Modulation ist und wie LEDs, Mikrocontroller, Transistoren und Potentiometer funktioniert.

10 Danksagung

An dieser Stelle möchte ich Allen danken, die zu dieser Arbeit beigetragen haben. Danke an die Mitglieder des Hackerspaces Odenwilusenz in Beringen für ihre materielle und inhaltliche Unterstützung[6]. Danke an meine Eltern für das Wohnzimmer, das ich tagelang mit meinen technischen Geräten belegen durfte. Und ausserdem Danke an meinen Betreuer Martin Schwarz für seine Unterstützung und Wegweisung im Entstehungsprozess.

11 Glossar

PHP: Eine Programmiersprache, die es erlaubt, über einen Browser Skripte auf dem Server laufen zu lassen.
SSH: Secure Shell. Ein verschlüsseltes Netzwerkprotokoll, über das sich ein Computer auf einem anderen Computer (meist einem Server) im selben Netzwerk einloggen kann.
Library: (dt. Programmbibliothek) Eine Art Wörterbuch, das komplizierte Abläufe mit einfachen Befehlen und Parametern verfügbar macht.
HTTP-Request: Anfrage eines Clients an einen Server. Dabei werden Daten an diesen übermittelt, diese verarbeitet und allenfalls eine Antwort zurückgegeben. Die einfachste Form davon ist eine URL.
Pulsweiten-Modulation: Ein technisches Verfahren, das es erlaubt, durch sehr schnelles Schliessen und Öffnen eines Stromkreises eine variable Spannung zu erzeugen, abhängig vom Verhältnis der Zeit, während der eine Spannung besteht gegenüber jener Zeit, in der keine vorliegt.
GET: HTTP-Request, der die zu übermittelnden Daten an die URL anhängt. Dabei sollten nur 256 Bytes an Daten übermittelt werden.
POST: HTTP-Request, der die zu übermittelnden Daten in der Nachricht überträgt. Damit können unbegrenzt viele Daten versendet werden.
HTML: Sprache, aus der normale Websites bestehen.
Array: Sammlung an gleichen Variablen.
char: Kurz für Charakter. Variablen-Typ, enthält Zeichen.
String: Variablen-Typ, enthält eine Folge von Zeichen.
Pulldown-Widerstand: Hochohmiger Widerstand zwischen Output und Masse, der das Potential eines Ausgangs gleich der Masse setzt, wenn dieser keins bestimmt (also eine Spannung ausgibt).

[6] www.odenwilusenz.ch

12 Quellen

12.1 Quellen zum Ansatz und zu Bauteilen

- www.apple.com
- www.raspberrypi.org
- www.arduino.cc
- www.microcontroller.net

(Alle abgerufen am 31. November 2016)

12.2 Inhaltliche Quellen, Referenzen

- www.php.net
- www.arduino.cc
- www.wikipedia.org
- www.instrucables.com

12.3 Bildquellen

- Abbildungen 1-12: Valentin Huber