Universelle Lichtsteuerung mit Arduino

Der Arduino ist die ideale Hard- und Softwareplattform für alle, die recht unaufwendig eine bestimmte, von einem Mikrocontroller gesteuerte Anwendung aufbauen wollen – ganz im Sinne der Grundidee. Unser Leser Michael Gaus hat auf dieser Basis eine interessante universelle Lichtsteuerung entwickelt, die mehrere Lichteffekte gleichzeitig steuert, die Möglichkeiten des Arduino umfassend nutzt und aufgrund der leicht beherrschbaren Arduino-Programmierumgebung einfach individuell anpassbar ist.

Vielseitig

Ein Mikrocontroller bietet sich förmlich an, will man mehrere voneinander unabhängige Lichteffekte generieren, z. B. für die Modelleisenbahnanlage.

Mit der hier beschriebenen universellen Lichteffektsteuerung können solche unabhängig voneinander anzusteuernden Lichteffekte relativ einfach erzeugt werden. Für jeden Lichteffekt wird eine eigene Funktion mit einem bestimmten Aufbauschema erstellt, um die Ansteuerung einer Teilaufgabe zu übernehmen. Die einzelnen Lichteffektfunktionen laufen dann interruptgesteuert durch einen Timer alle „quasi-parallel" nebeneinander.

In der Beispielanwendung sollen folgende Lichteffekte realisiert werden:

- Baustellenleitlicht mit acht LEDs:
  Alle LEDs werden der Reihe nach eingeschaltet, bis die ganze LED-Kette leuchtet. Nach einer kurzen Pause werden alle LEDs ausgeschaltet, und nach einer weiteren kurzen Pause wiederholt sich das Ganze von vorn.
- Baustellenblitzer:
  Ein Blitzlicht, das z. B. an Warnschildern angebracht werden kann.
- Baustellenampel:
  Zwei Ampeln zur Verkehrsregelung bei einer aufgrund einer Baustelle verengten Fahrbahn.
- Andreaskreuz:
  Warnblinklicht bei unbeschrankten Bahnübergängen. Dieses wird nach Auslösung eines Kontakts für eine gewisse Zeit eingeschaltet und anschließend automatisch wieder abgeschaltet.
- Erweiterungsmöglichkeit per I²C-Bus:
  Über Portbausteine des Typs PCF8574 können weitere Portausgänge realisiert werden. Hier wird über einen solchen Portbaustein ein 8-Kanal-Lauflicht gesteuert.

Bild 1 zeigt den Musteraufbau der Lichteffektsteuerung mit Baustellenleitlicht, Baustellenampeln und Andreaskreuzen inklusive eines Reedkontakts als magnetischen Auslösekontakt. Zusätzlich ist noch eine I²C-Erweiterungsplatine mit einem PCF8574 angeschlossen, sodass acht zusätzliche LEDs beliebig steuerbar sind.

Hardware und Schaltung

Bild 2 zeigt die Schaltung der Steuerung. Als Steuerplatine kam ein Arduino Uno [1], [2] zum Einsatz, der kostengünstig und sofort einsetzbar erhältlich ist. Dieser stellt bis zu 20 I/O-Pins zur Verfügung, die von einem ATmega328-Mikrocontroller aus der AVR-Familie gesteuert werden.

Die Firmware (Sketch) kann über die kostenlos er-

hältliche Arduino-Entwicklungsumgebung erstellt und per USB dank vorinstalliertem Bootloader übertragen werden, ohne dass ein spezielles Programmiergerät erforderlich ist.

Durch die Verwendung des Arduino-Uno-Boards ist der Hardwareaufwand somit relativ gering.

Der High-Pegel der Ausgänge beträgt ca. 5 V. LEDs mit einer Stromaufnahme von bis zu 20 mA können über entsprechende Vorwiderstände direkt an die Port-Pins angeschlossen werden. Dies dürfte für die meisten Lichteffekte ausreichend sein. Wenn höhere Ströme erforderlich sein sollten, müssen zusätzliche Treibertransistoren verwendet werden.

Der Arduino Uno stellt 14 Digitalpins über Buchsen zur Verfügung, die von 0 bis 13 durchnummeriert sind und jeweils als Eingang oder Ausgang konfiguriert werden können. Zusätzlich gibt es noch sechs Analogpins, die mit A0 bis A5 bezeichnet und als A/D-Eingänge verwendbar sind. Diese Analogpins können jedoch auch als digitale Ein- oder Ausgänge genutzt werden.

Insgesamt sind also 20 I/O-Pins verfügbar. Über die Pins A4 und A5 kann außerdem ein I²C-Bus angesteuert werden. So kann man z. B. über PCF8574-Bausteine weitere I/O-Pins generieren.

Beim PCF8574 ist zu beachten, dass die LEDs gegen

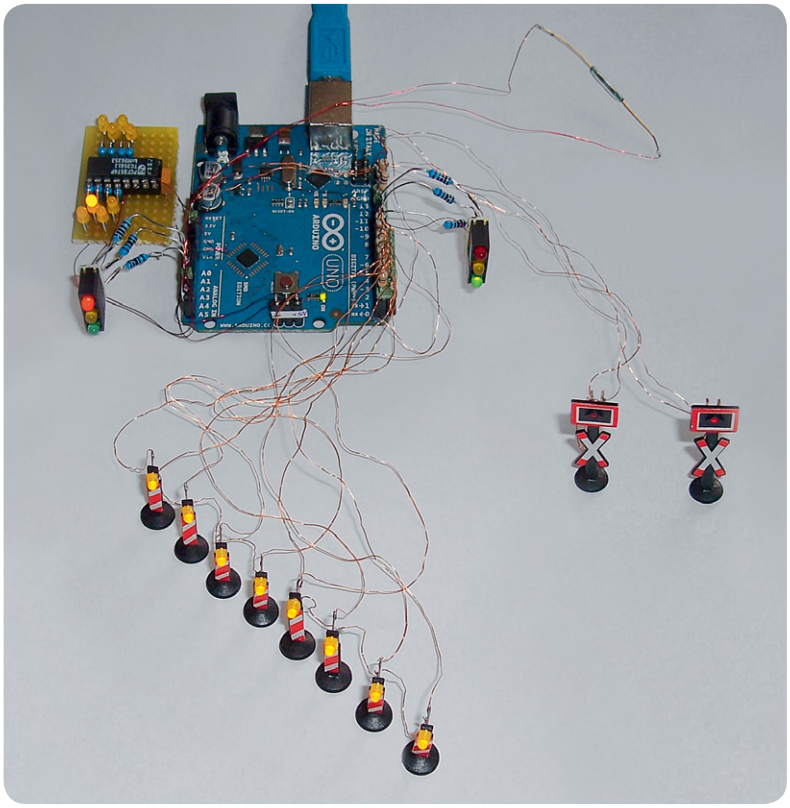

Bild 1: Die fertig aufgebaute Lichtsteuerung mit Port-Expander und Lichteffekt-Applikationen

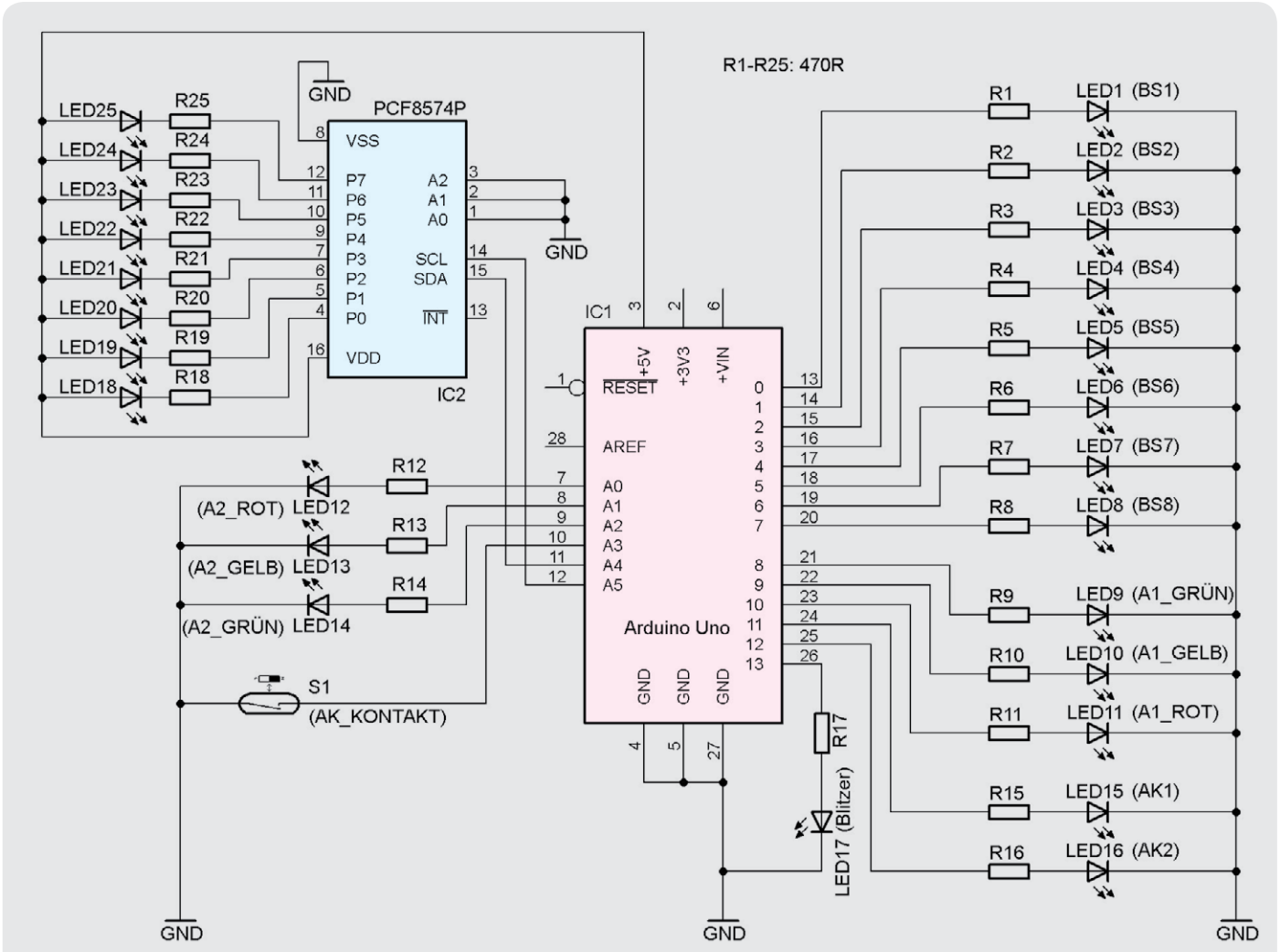

Bild 2: Die Gesamtschaltung der Lichteffektsteuerung

```
void init_lichteffekte(void)
{  // Initialisierung der Lichteffekte, z.B. Portpins als Ausgang konfigurieren.

  init_ampel();
  init_andreaskreuz();
  init_blitz();
  init_baustellen_leitlicht();
  init_i2c_lauflicht();
}


void lichteffektsteuerung(void)
{  // Zyklischer Aufruf der einzelnen Lichteffekte jede Millisekunde, gesteuert per Timerinterrupt.
   // Die hier aufgerufenen Funktionen sollten eine möglichst kurze Laufzeit haben, um das Timing nicht zu blockieren.
   // In der Praxis normalerweise kein Problem, da meist nur einige Portpins umgeschaltet werden müssen.

  ampel();
  andreaskreuz();
  blitz();
  baustellen_leitlicht();
  i2c_lauflicht();
}


void setup()
{
  init_lichteffekte();  // Initialisierung der Lichteffekte

  init_system();  // Diesen Aufruf nicht entfernen, da er für die Initialisierung der Steuerung unbedingt benötigt wird.
                  // Er sollte ganz am Ende der setup-Funktion stehen.
}


void loop()
{  // Hauptschleife, kann für nicht zeitkritische Dinge verwendet werden.

}
```

Bild 3: Das Hauptprogramm für die Lichteffektsteuerung

+5 V geschaltet werden müssen, da dieser Baustein nur bei Low-Pegel genügend Strom liefern kann. In der Firmware muss dann ein Low-Pegel ausgegeben werden, um die LED einzuschalten, die Ansteuerung muss also invertiert erfolgen. Über die Adresspins A0 bis A2 wird ein Teil der I²C-Adresse des Bausteins festgelegt. Wenn mehrere Bausteine angeschlossen werden, müssen die Adresspins so beschaltet werden, dass jeder Baustein eine eindeutige Adresse hat.

In der in Bild 2 gezeigten Beispielanwendung werden die I/Os als 17 Digitalausgänge zur Steuerung von LEDs, ein Digitaleingang zur Abfrage eines Reedkontakts sowie zwei Steuerpins für den I²C-Bus als Porterweiterung konfiguriert.

Die Buchsen mit den Ausgangspins auf dem Arduino Uno können über einreihige Stiftleisten mit einem 2,54-mm-Raster sehr einfach kontaktiert werden.

Die Spannungsversorgung kann entweder über die USB-Buchse (5 V) oder über die Netzteilbuchse mit einem zusätzlichen Steckernetzteil (7–12 V) erfolgen.

Die Firmware (Sketch)

Die Firmware wird als sogenannter Sketch mit der Arduino-Entwicklungsumgebung [3] erstellt, die kostenlos von der Arduino-Homepage heruntergeladen werden kann.

Die hier erstellte Beispielanwendung wurde mit der Arduino-Version 1.0.1 erstellt und hat den Namen „lichteffektsteuerung.ino“ (Bild 3). Dieser Sketch kann über „Datei“ => „Öffnen“ geladen werden.

Der Sketch besteht aus mehreren Dateien, die alle die Dateiendung „.ino“ haben. Diese werden als Tabs alle nebeneinander aufgelistet. Die beiden Tabs „_i2c“ und „_system“ sollten nicht verändert werden, da diese für die eigentliche Systemsteuerung zuständig sind. Zur Kennzeichnung wurden diese mit einem vorangestellten Unterstrich im Namen bezeichnet.

Pro Lichteffekt gibt es einen eigenen Tab, der mit dem Namen des Lichteffekts bezeichnet ist, z. B. „ampel“.

In diesem Tab werden zunächst die verwendeten Port-Pins definiert, sodass sie später über einen entsprechenden Namen angesprochen werden können. Bei evtl. Änderungen des Port-Pins reicht es aus, diese an dieser einen zentralen Stelle durchzuführen. Die rote LED der Ampel 1 ist beispielsweise an Pin 10 angeschlossen und die rote LED der Ampel 2 an Pin A0.

Anschließend gibt es eine Init-Funktion, die einmalig beim Programmstart aufgerufen wird und die Konfiguration der benötigten Port-Pins für den Lichteffekt vornimmt. Nach einem Reset sind alle Pins des Arduino als Eingang konfiguriert, bei Verwendung als LED-Steuerausgang müssen diese also zunächst als Ausgang konfiguriert werden. Hier heißt diese Init-Funktion „init_ampel“. Dort werden alle 6 Pins für die Ampel als Ausgänge konfiguriert (über die Arduino-Funktion „pinMode“) und die beiden roten LEDs von Ampel 1 und 2 ein- sowie alle anderen 4 LEDs der Ampeln ausgeschaltet (über die Arduino-Funktion „digitalWrite“).

Danach kommt die eigentliche Steuerfunktion für den Lichteffekt, hier im Beispiel mit „ampel“ (Bild 4) bezeichnet. Diese Steuerfunktion wird interruptgesteuert über einen Timer zyklisch jede Millisekunde aufgerufen. Über eine Zustandsmaschine wird dann der entsprechende, gerade aktive Zustand des Lichteffekts abgearbeitet. Durch die Verwendung von Makros geschieht dies für den Anwender sozusagen unbemerkt im Hintergrund. Es muss lediglich ein bestimmtes Grundgerüst erstellt werden, damit die Steuerung korrekt erfolgt.

Als Einstiegspunkt muss das Makro „STATE_ENTRY“ und als Ausstiegspunkt das Makro „STATE_EXIT“ aufgerufen werden. Dazwischen werden die einzelnen Zustände des Lichteffekts definiert. Mit dem Makro „STATE_DURATION(x)“ beginnt ein Zustand, der eine Zeitdauer von x Millisekunden hat. Dahinter werden die auszuführenden Befehle geschrieben, z. B. Setzen der gewünschten Port-Pins. Die hier aufgerufenen Funktionen sollten eine möglichst kurze Laufzeit haben, um das Timing nicht zu blockieren. In der Praxis ist dies normalerweise kein Problem, da bei den Lichteffektsteuerungen meist nur einige Port-Pins umgeschaltet werden müssen. Mit dem Makro „STATE_END“ wird das Ende des Zustands markiert.

Auf diese Art und Weise können die Zustände sequenziell nacheinander definiert werden. Die Steuerung durchläuft sozusagen im Hintergrund diese Zustände nacheinander, wobei jeweils zunächst die nach „STATE_DURATION“ angegebenen Befehle ausgeführt werden und nach Ablauf der angegebenen Zustandsdauer auf den nächsten Zustand umgeschaltet wird.

Wenn von einem Zustand nach Ablauf der Zustandsdauer nicht in den Folgezustand, sondern in einen anderen Zustand gewechselt werden soll, kann dies mit dem Makro „STATE_SET(x)“ erfolgen. „x“ ist die Nummer des gewünschten Zustands, beginnend mit 0. Dies wird z. B. bei der Andreaskreuz-Steuerung verwendet. Falls Variablen verwendet werden sollen, müssen diese vor „STATE_ENTRY“ definiert werden. Wenn diese über die verschiedenen Zustände der Statemaschine erhalten bleiben sollen, müssen sie mit „static“ deklariert werden.

```
#define LED_A1_ROT 10
#define LED_A1_ROT_AUS digitalWrite(LED_A1_ROT, LOW)
#define LED_A1_ROT_EIN digitalWrite(LED_A1_ROT, HIGH)
#define LED_A1_ROT_DIR_OUT pinMode(LED_A1_ROT, OUTPUT)

#define LED_A1_GELB 9
#define LED_A1_GELB_AUS digitalWrite(LED_A1_GELB, LOW)
#define LED_A1_GELB_EIN digitalWrite(LED_A1_GELB, HIGH)
#define LED_A1_GELB_DIR_OUT pinMode(LED_A1_GELB, OUTPUT)

#define LED_A1_GRUEN 8
#define LED_A1_GRUEN_AUS digitalWrite(LED_A1_GRUEN, LOW)
#define LED_A1_GRUEN_EIN digitalWrite(LED_A1_GRUEN, HIGH)
#define LED_A1_GRUEN_DIR_OUT pinMode(LED_A1_GRUEN, OUTPUT)

#define LED_A2_ROT A0
#define LED_A2_ROT_AUS digitalWrite(LED_A2_ROT, LOW)
#define LED_A2_ROT_EIN digitalWrite(LED_A2_ROT, HIGH)
#define LED_A2_ROT_DIR_OUT pinMode(LED_A2_ROT, OUTPUT)

#define LED_A2_GELB A1
#define LED_A2_GELB_AUS digitalWrite(LED_A2_GELB, LOW)
#define LED_A2_GELB_EIN digitalWrite(LED_A2_GELB, HIGH)
#define LED_A2_GELB_DIR_OUT pinMode(LED_A2_GELB, OUTPUT)

#define LED_A2_GRUEN A2
#define LED_A2_GRUEN_AUS digitalWrite(LED_A2_GRUEN, LOW)
#define LED_A2_GRUEN_EIN digitalWrite(LED_A2_GRUEN, HIGH)
#define LED_A2_GRUEN_DIR_OUT pinMode(LED_A2_GRUEN, OUTPUT)


void init_ampel(void)
{
  LED_A1_ROT_EIN;
  LED_A1_ROT_DIR_OUT;
  LED_A1_GELB_AUS;
  LED_A1_GELB_DIR_OUT;
  LED_A1_GRUEN_AUS;
  LED_A1_GRUEN_DIR_OUT;
  LED_A2_ROT_EIN;
  LED_A2_ROT_DIR_OUT;
  LED_A2_GELB_AUS;
  LED_A2_GELB_DIR_OUT;
  LED_A2_GRUEN_AUS;
  LED_A2_GRUEN_DIR_OUT;
}


void ampel(void)
{
  STATE_ENTRY;

  STATE_DURATION(3000); // A1: rot A2: rot
  LED_A2_GELB_AUS;
  LED_A2_ROT_EIN;
  STATE_END;

  STATE_DURATION(1000); // A1: rot/gelb A2: rot
  LED_A1_GELB_EIN;
  STATE_END;

  STATE_DURATION(5000); // A1: grün A2: rot
  LED_A1_ROT_AUS;
  LED_A1_GELB_AUS;
  LED_A1_GRUEN_EIN;
  STATE_END;

  STATE_DURATION(1000); // A1: gelb A2: rot
  LED_A1_GRUEN_AUS;
  LED_A1_GELB_EIN;
  STATE_END;

  STATE_DURATION(3000); // A1: rot A2: rot
  LED_A1_GELB_AUS;
  LED_A1_ROT_EIN;
  STATE_END;

  STATE_DURATION(1000); // A1: rot A2: rot/gelb
  LED_A2_GELB_EIN;
  STATE_END;

  STATE_DURATION(5000); // A1: rot A2: grün
  LED_A2_ROT_AUS;
  LED_A2_GELB_AUS;
  LED_A2_GRUEN_EIN;
  STATE_END;

  STATE_DURATION(1000); // A1: rot A2: gelb
  LED_A2_GRUEN_AUS;
  LED_A2_GELB_EIN;
  STATE_END;

  STATE_EXIT;
}
```

Bild 4: Beispiel für die Programmierung eines Lichteffekts, hier der Ampelsteuerung

Die Erstellung der Lichteffekte

Die Erstellung von Lichteffekten läuft in Kurzform folgendermaßen ab:

- Tab erstellen mit dem entsprechenden Namen des Lichteffekts, z. B. „ampel“
- Port-Pins definieren mit entsprechenden Namen, z. B. „LED_A1_ROT“
- Init-Funktion mit Konfiguration der Port-Pins erstellen, z. B. „init_ampel“
- Aufruf der Init-Funktion im Tab „lichteffektsteuerung“ in der Funktion „init_lichteffekte“
- Steuerfunktion für Lichteffekt erstellen, z. B. „ampel“

Der Aufbau:

```
STATE_ENTRY;

STATE_DURATION (Zeitdauer in Millisekunden bis zum nächsten State);
... // Port-Pins entsprechend steuern
STATE_END;

STATE_DURATION (Zeitdauer in Millisekunden bis zum nächsten State);
... // Port-Pins entsprechend steuern
STATE_END;

... // weitere States

STATE_EXIT;
```

- Aufruf der Steuerfunktion im Tab „lichteffektsteuerung“ in der Funktion „lichteffektsteuerung“

Weitere Infos:

[1] Beschreibung des Arduino Uno: arduino.cc/en/Main/ArduinoBoardUno

[2] Arduino Uno bei ELV: Best.-Nr. JY-10 29 70

[3] Arduino: www.arduino.cc

[4] Download der Beispieldateien: www.elv.de: Web-Code #1258

Übertragen der Firmware auf das Arduino-Board

Zunächst müssen die Einstellungen passend zum verwendeten Arduino vorgenommen werden.
Bei „Tools“ => „Board“ wird „Arduino Uno“ ausgewählt. Bei „Tools“ => „Serieller Port“ wird der entsprechende COM-Port ausgewählt, unter dem sich der Arduino am PC angemeldet hat. Falls mehrere COM-Ports in der Liste sind, kann in der Systemsteuerung nachgesehen werden, welches der passende virtuelle Port ist.

Anschließend muss der Sketch kompiliert werden mit „Sketch“ => „Überprüfen/Kompilieren“. Wenn keine Fehler aufgetreten sind, kann der Sketch auf den Arduino übertragen werden mit „Datei“ => „Upload“.

Nun sollten die Lichteffekte entsprechend ablaufen. Falls Änderungen an der Firmware vorgenommen werden sollen, können diese jederzeit wieder an den Arduino übertragen werden.

Der komplette Sketch kann mit „Datei“ => „Speichern“ abgespeichert werden.

Alle Beispieldateien des Projekts stehen unter [4] zum Download zur Verfügung.

ELV

Leserwettbewerb

Wir wollen es wissen – Ihre Anwendungen und Applikationen!

Jede veröffentlichte Anwendung wird mit einem Warengutschein in Höhe von 200 Euro belohnt.

Welche eigenen kreativen Anwendungen und Applikationen haben Sie mit den ELV-Haustechnik-Systemen, aber auch anderen Produkten und Bausätzen realisiert? Ob mit Standard-Bausteinen oder eingebunden in eigene Applikationen: Alles, was nicht gegen Gesetze oder Vorschriften, z. B. VDE-Vorschriften, verstößt, ist interessant. Denn viele Applikationen verhelfen sicher anderen zum Aha-Erlebnis und zur eigenen Lösung.

Schreiben Sie uns, fotografieren Sie Ihre Applikation, berichten Sie uns von Ihren Erfahrungen und Lösungen. Die interessantesten Anwendungen werden redaktionell bearbeitet und im ELVjournal mit Nennung des Namens vorgestellt.

Die Auswahl der Veröffentlichungen wird allein durch die ELV-Redaktion ausschließlich nach Originalität, praktischem Nutzen und realisierter bzw. dokumentierter Ausführung vorgenommen, es besteht kein Anspruch auf Veröffentlichung, auch bei themengleichen Lösungen. **Der Rechtsweg ist ausgeschlossen.** Für Ansprüche Dritter, Beschädigung und Verlust der Einsendungen wird keine Haftung übernommen. Alle Rechte an Fotos, Unterlagen usw. müssen beim Einsender liegen. Die eingesandten Unterlagen und Aufnahmen verbleiben bei der ELV Elektronik AG und können von dieser für Veröffentlichungen und zu Werbezwecken genutzt werden. Ihre Einsendungen senden Sie per Brief oder Mail mit Stichwort „Leserwettbewerb“ an:

ELV Elektronik AG, Leserwettbewerb, 26787 Leer
bzw. leserwettbewerb@elvjournal.de