Wasserkocher-Steuerung mit Arduino

Tags:	Hardware, Software
Autor/Impressum:	Dr. Michael Neher
Geeignet für Klassenstufe:	12 (evtl. auch 11)
Zeitdauer:	Ab 12 Doppelstunden
Pädagogische Ziele:	Lesen von Schaltplänen, Funktion elektronischer Bauteile, praktische Erfahrung mit Schaltungsaufbau, erste Erfahrungen mit der Programmierung eines Mikrocontrollers (in c), Verständnis für die Denkweise von Informatikern und ihrer Illusion, dass die Welt kontrollierbar ist ;-)
Pädagogischer Hintergrund:	
Nötige Vorbereitungen:	Als Vorbereitung wünschenswert ist ein Praktikum in Klasse 10 oder 11 zur Vermittlung von Grundkenntnissen in Boolescher Algebra, Dualzahlen und dem Aufbau eines Addierers. Anregungen dazu findet man auf der Waldorf-IT-Webseite oder auf Nachfrage beim Autor. Aufgrund des erhöhten Anspruchs des Labors empfiehlt es sich sehr, die Teilnahme auf freiwilliger Basis zu gestalten.
Hilfsmittel:	Siehe Text
Involvierbare Fächer:	Mathematik, Physik
Erscheinungsdatum:	15.10.2017
Letzte Überarbeitung:	12.12.2018 durch Robert Neumann

Inhaltsverzeichnis

Kurzbeschreibung

Ein Mikrocontroller enthält neben dem Prozessor auch Peripheriefunktionen sowie meist auch Arbeits- und Programmspeicher. Es ist also ein kompletter Computer auf einem einzigen Chip! In diesem Praktikum wollen wir den bekanntesten einfachen Mikrocontroller, den Arduino, kennenlernen. Wir können damit mit wenigen zusätzlichen Bauelementen bereits Lichtsteuerungen bauen, elektrische Geräte mit einem Relais ansteuern, Temperaturen messen und anzeigen, aber auch Primzahlen berechnen. Zum Abschluss verwenden wir diese Teilschaltungen, um damit eine Steuerung für einen Wasserkocher zu bauen. Natürlich nicht so ein simples Gerät wie man es beim ALDIDL bekommt, sondern eines mit allen Schikanen: Man kann die Temperatur nicht nur anzeigen, sondern auch einstellen. Damit lässt sich also nicht nur kochendes Wasser bereiten, sondern auch eine Babyflasche auf eine verträgliche Temperatur erwärmen: Zwei Geräte in einem! Ziel des Praktikums ist es nicht, fundierte Hardware- und Softwarekenntnisse zu vermitteln, sondern in einem Schnelldurchgang mit teilweise vorgegebenen Beispielen die Möglichkeiten eines Mikrocontrollers und seiner Programmierung aufzuzeigen.

1 Einleitung

Mikrocontroller sind im Alltag allgegenwärtig, obwohl sie kaum auffallen. In einer modernen Oberklasse-Limousine sind etwa 50 Mikrocontroller verbaut. Die meisten elektrischen Haushaltsgeräte werden von ihnen gesteuert. Sie halten Einzug im Smart Home und dem Internet der Dinge. In der Industrie werden sie als Maschinensteuerungen verwendet. Man sieht also, dass sie rein zahlenmäßig eine viel größere Bedeutung haben als Notebooks und Smartphones. Dermaßen motiviert, geht es an die Auswahl eines Mikrocontroller-Systems. Zwei Systeme sind derzeit weit verbreitet, da Hardwareaufbau und Betriebssystem öffentlich sind und eine Unmenge von Applikationen, Schaltungen und Hilfen im Internet verfügbar sind: Arduino und Raspberry Pi. Der Arduino ist geeignet für die oben genannten mehr oder weniger einfachen Anwendungen und auch leichter zu verstehen. Der Raspberry Pi ist dagegen ein schon recht leistungsfähiger Computer, wie er auch in Smartphones verwendet wird; es fehlt eigentlich nur der Touchscreen, der aber auch angeschlossen werden kann. Er hat ein „richtiges“ Betriebssystem, meist Linux. Damit findet er z.B. Verwendung als Media-Box, Internet-Server, Navigationsgerät, etc. Beim Arduino gibt es wiederum einige Varianten. Für unsere Zwecke reicht der einfache, preiswerte und verbreitete Arduino Uno. Dass die Anzahl seiner Schnittstellen recht begrenzt ist, ist hier kein Nachteil, sondern eher Ansporn, diese durch Zusatzschaltungen „aufzubohren“.

Zunächst werden Grundschaltungen mit einzelnen Sensoren und Aktoren aufgebaut und ihre Funktion anhand simpler Programme demonstriert. Es werden Anregungen gegeben, wie die Schüler diese Programme modifizieren können. Im nächsten Schritt werden alle vorgestellten Grundschaltungen kombiniert zu einer Steuerung für einen Wasserkocher mit Temperaturvorgabe und -anzeige.

Abbildung 1: Der komplette Aufbau

2 Material und Hardware-Vorbereitungsarbeiten

2.1 Material für den Lehrer

- Notebook und leistungsfähiger Beamer zur Projektion der Arduino-Programme oder -Schaltpläne
- USB-Stick zur Installation der Software und Verteilung von Programmen (ein Internet-Anschluss der Schüler-PCs ist nicht nur unnötig; für den Praktikumsfortschritt ist er abträglich)
- Optional: Lötstation (z.B. von Weller; eine Temperaturanzeige ist überflüssig), bleifreies Lötzinn
- Nützlich: Multimeter (zur Fehlersuche)
- Arduino Starter Kit: Wenn man mit Arduino, Elektronik und/oder Programmieren noch wenig Erfahrung hat, erweist sich das offizielle „Arduino Starter Kit“ mit hervorragendem Handbuch als notwendig und hinreichend und sollte vorab durchgearbeitet werden: www.amazon.de/Franzis-Arduino-Starter-Kit/dp/3645651942. Ferner sollte man die Datenblätter der Bauteile entweder vom Internet herunterladen oder sich vom Autor zusenden lassen, allein schon wegen der Anschlussreihenfolge.

2.2 Material für die Schüler

Je nach Erfahrung und Stresstoleranz des Lehrers können drei bis sechs Arbeitsgruppen mit je zwei Schülern betreut werden. Ebenso viele Arduino-Bausätze und Host-Rechner (mit Windows, Mac OS oder Linux) sollten vorhanden sein (dazu evtl. ein weiterer Bausatz für den Lehrer oder als Reserve für zerschossene Bauteile). Ferner benötigt man ein paar weitere Kleinteile, die unten aufgeführt sind.

Es gibt keinerlei Anforderungen an die Rechengeschwindigkeit der Host-Rechner; daher sind auch ältere PCs und Betriebssysteme völlig ausreichend und verursachen lediglich Naserümpfen bei den Schülern.

2.3 Material, einmal pro Labor

- Wasserkocher: Das billigste Modell mit verdeckter Heizspirale ist ausreichend. Nett ist es, wenn er ein Sichtfenster enthält, denn dann kann man ihn von innen mit einer RGB-LED beleuchten. Außerdem sollte man den Einschalter z.B. mit einem Streichholz arretieren können.
- Switchbox-Relais: Das Relais dient zum Schalten großer Lasten an Netzspannung. Details siehe unten.

2.4 Arduino-Bausätze

Nach umfangreichem Vergleich von Preis-Leistungs-Verhältnissen wurde folgender Bausatz gewählt: „SunFounder Starter Learning Kit for Arduino Beginner, from Knowing to Utilizing“: www.amazon.de/SunFounder-Starter-Learning-Beginner-Utilizing/dp/B00E59L71S. Man sollte auf jeden Fall vermeiden, die teurere und schlechter ausgestattete Version 2.0 zu kaufen! Hauptvorteil dieses Bausatzes ist (neben dem günstigen Preis) das sog. Prototype Shield. Das ist eine kleine Aufsteckplatine für ein Breadboard zum Aufbau der eigenen

Schaltungen, die man direkt auf die Arduino-Hauptplatine stecken kann. Dadurch wird die Verdrahtung wesentlich vereinfacht und Fehlerquellen durch unterbrochene Drahtverbindungen vermieden. Das scheint aber in Version 2.0 nicht enthalten zu sein. Für umfangreichere Schaltungen ist in diesem Bausatz ein weiteres, größeres Breadboard enthalten. Man kann das Prototype Shield mit kleinem Breadboard allerdings auch separat bestellen: www.amazon.de/Arduino-ProtoShield-Prototyping-Shield-170-Mini-Steckplatine/dp/B00I0R9Z56 (derzeit ca. 5,- Euro)

Es gibt mittlerweile auch ein praktisches Verbundsystem von Arduino-Uno-Board und etwas größerem Breadboard, STEMTera: https://www.elektormagazine.de/news/review-stemtera-arduino-praktisch-gut. Der Nachteil ist der höhere Preis. Wenn ein Bauteil zerschossen wird, kann man das Board abschreiben.

Der Bausatz von Sunfounder enthält:

Anzahl	*	Bauteil	Kommentar
1		SunFounder UNO R3	Mikrocontroller-System, kompatibel zu Arduino Uno
1	*	IR-Fernbedienung	
			Aufbau und Verdrahtung:
1	*	AAA Batteriehalter	
1		Prototype-Shield	Aufsteckplatine
1		Breadboard (klein)	zum Aufkleben auf das Prototype Shield
1		Breadboard (groß)	
1		USB Kabel	zum Programm-Download, Spannungsversorgung und evtl. Datenaustausch mit dem Host
30		Jumper wire (male to male)	Verbindungsdrähte
30		DuPont wire (male to female)	Flachbandkabel
40		2.54 Pin Header	Steckerleiste fürs Breadboard zum Anlöten von Kabeln
			Passive Bauelemente:
8		Widerstand (220Ω)	Rot-rot-schwarz-schwarz—braun (1%)
5		Widerstand (1KΩ)	Braun-schwarz-schwarz-braun—braun (1%)
5		Widerstand (10KΩ)	Braun-schwarz-schwarz-rot—braun (1%)
			Aktive Bauelemente:
1		74HC595	Seriell-Parallel-Schieberegister; nützlich zur I/O-Erweiterung z.B. bei der Ansteuerung der 4x7-Segment-Anzeige

Anzahl	*	Bauteil	Kommentar
			Input Devices:
10		Drucktaster (groß)	
4		Drucktaster (klein)	
2	*	Neigungsschalter	
2		Potentiometer (50KΩ)	Regelbarer Widerstand
1	*	Joystick PS2	Für Leute, die auf die Reinheit der deutschen Sprache achten: dt. Luststengel. Enthält zwei kreuzweise angeordnete Potentiometer und einen Druckschalter
1	*	IR-Empfänger	Infrarot-Empfänger
1			
2-3		Fotowiderstände (LDR)	
1	*	Flammensensor-Modul	
			Output Devices:
8		LED (rot)	
5		LED (blau)	
5		LED (gelb)	
1		RGB Modul	Dreifach-LED inklusive Vorwiderständen
1	*	Aktiver Buzzer	Summer; nützlich um Lehrer in den Wahnsinn zu treiben
1	*	Passiver Buzzer	dito
1		4-stellige 7-Segment-Anzeige (common cathode)	
1		einstellige 7-Segment-Anzeige (common cathode)	
1		LCD-Display 1602	
1	*	Schrittmotor	
1	*	Schrittmotortreiber-Modul	
1	*	Servomotor	z.B. für ferngelenkte Fahrzeuge
1	*	Relais-Modul	nicht für netzbetriebene Geräte geeignet

An dieser Liste kann man sich auch orientieren, wenn man einen anderen Bausatz auswählen möchte. Die mit (*) markierten Bauteile sind „nice-to-haves“, der Rest ist „must-have“ (wenn auch nicht unbedingt in der angegebenen Stückzahl).

Nachteile des Sunfounder-Bausatzes (immerhin gute China-Qualität):

- Manche Drähte haben sich nach längerer Fehlersuche als defekt heraus gestellt. Außerdem wären mehr Drähte hilfreich.
- Das USB-Kabel ist ziemlich kurz.
- Bei alten Bausätzen war die Anschlussbeschriftung der RGB-LED falsch (vertauschte Farben).
- Das Sunfounder-Handbuch war in der alten, mir vorliegenden Version nicht sonderlich brauchbar; aber zumindest ist ein Beispielprogramm für die meisten Sensoren und Aktoren angeführt.
- Ich habe aber auf der Website neue, brauchbare Tutorials entdeckt: https://www.sunfounder.com/learn

2.5 Zusätzliche Bauteile

Folgende für das Praktikum nötige Bauteile muss man separat besorgen (jeweils pro Bausatz):

- Mindestens 4 Universal-NPN-Transistoren (z.B. BC547C; Anschlussreihenfolge, wenn man auf die abgeflachte Seite schaut: Kollektor, Basis, Emitter) (am besten 100er-Pack BC547C bei Ebay bestellen, z.B. bei Kessler-Elektronik: www.ebay.de/itm/100-x-BC547C-Transistor-npn-45V-0-1A-0-5W-TO92-/311252446290
- 3 Keramik-Kondensatoren 1nF http://www.ebay.de/itm/Stk-30-x-Keramik-Scheiben-Kondensator-Wahlbar-1pF-1uF-100V-Capacitor-RM5-/191608762057
- 1 Keramik-Kondensator 100nF http://www.ebay.de/itm/10-x-Multilayer-Keramik-Kondensator-100nF-0-1uF-THD-50V-/192315675048
- 4 Abstandsbolzen M3 15mm, mit 2 Innengewinden, Polyamid (Conrad Bestell-Nr. 534781–62, billiger bei Ebay) 4 Zylinderschrauben M3 10mm, Schlitz, Polyamid (Conrad Bestell-Nr. 815802–62, billiger bei Ebay) Zylinderschrauben und Abstandsbolzen verwendet man, um dem Arduino-Uno-Board Beinchen zu machen, damit es keine Kurzschlüsse gibt, wenn es z.B. auf dem PC-Gehäuse liegt. Evtl. muss man seitlich etwas von der Zylinderschraube abknipsen, damit sie trotz beengter Verhältnisse auf dem Board passt.
- Ggf. zusätzliche Drahtbrücken z.B. www.amazon.de/gp/product/B008U4ZOLI

Zur Aufbewahrung der ganzen Kleinteile ist ein Sortimentskasten nützlich, z.B. www.amazon.de/Sortimentskasten-schwarz-Doppelseitig-Kleinteile-Organizer/dp/B0053WO06U

All diese Bauteile, ferner größere Breadboards findet man bei Amazon, Ebay oder bei Conrad-Elektronik, wo man alles bekommt, aber dafür etwas teurer. Bei Bestellungen von Artikeln aus China muss man längere Versandzeiten beachten und rechtzeitig bestellen.

2.6 Relais

Will man größere 230V-Lasten ansteuern, wie z.B. einen Wasserkocher oder eine Lampe, braucht man ein Relais mit galvanischer Trennung durch einen Optokoppler (das Relais im Bausatz taugt dafür nicht). Gut geeignet ist das Switchbox-Relais von Antrax (www.antrax.de):
https://www.antrax.de/produkt/switchbox-relais-sb1105/ (50,30 Euro). Das Relais ist mit einer Schaltleistung von 2000W angegeben. Laut Herrn Bode von der Firma Antrax wird aber intern ein 16A-Relais verwendet, das bis zu 3650W schalten kann.

Dazu benötigt man noch ein Adapterkabel, das man entweder selber löten oder auch einfach kaufen kann:

https://www.antrax.de/produkt/anschlusskabel-laborstecker/ (7,20 Euro)
Die 4mm-Laborstecker des Adapterkabels kann man z.B. mit Krokodilklemmen an der Schaltung anbringen; oder man schneidet sie ab und lötet die Kabel an zwei kurze Stifte des 2.54 Pin Headers.

Aufgrund des Preises wird man wohl nur ein Relais für das Praktikum besorgen.

Man kann das Switchbox-Relais direkt durch einen Arduino-Ausgang ansteuern. Der Steuerstrom ist so niedrig, dass man parallel dazu noch eine Kontroll-LED (mit 220Ω-Vorwiderstand) schalten kann.

Bei der Verwendung des Relais ist darauf zu achten, dass die Einschalt- und Ausschaltdauer nicht zu kurz ist und damit das Relais „flattert“. Das beeinträchtigt die Lebensdauer des Relais.

2.7 Temperatur-Sensor

Mindestens einen der LM35-Temperatursensoren sollte man so präparieren, dass er sich in den Wasserkocher tauchen lässt (ohne dass er schmilzt oder es Kurzschlüsse gibt).

Zunächst kürzt man die Beinchen des Sensors auf unterschiedliche Längen und lötet daran ein höchstens 1m langes Flachbandkabel so an, dass jeweils eine Lötstelle noch auf der Höhe der Isolation der nächsten liegt.

Es hat sich gezeigt, dass das lange Kabel die Genauigkeit des Sensors beeinträchtigt. Daher schlage ich vor, noch einen möglichst kleinen 100nF-Keramikkondensator parallel zur Versorgungsspannung direkt an den Sensor zu löten (das sind bei dem TO92-Gehäuse die beiden Außenpins). Das andere Ende des Kabels lötet man später an die kurzen Pins einer Steckerleiste mit 3 Pins und vergisst nicht, sie zu beschriften!

Zuvor aber muss der Sensor wasserdicht isoliert werden. Die naheliegende Lösung mit der Klebepistole funktioniert nicht, da der Kleber bei der Siedetemperatur von Wasser weich wird. Ich habe Zweikomponenten-Epoxidkleber verwendet, um die Anschlüsse zu isolieren und anschließend die ganze Anordnung in einen Schrumpfschlauch eingeschrumpft (der Schlauch sollte bis zum Wasserkocher-Oberrand reichen). Auch die Schrumpfschlauch-Enden wurden nochmal verklebt (Uhu). Alternativ könnte man die Anordnung auch in ein Glasröhrchen einkleben. Das obere Ende des Sensors sollte jedenfalls frei bleiben, damit ein zügiger Wärmeaustausch gewährleistet ist. Der Sensor im Plastikgehäuse sollte nicht auf dem Metallboden des Wasserkochers aufliegen, damit er nicht schmilzt. Besser wäre es sicher, einen LM35 im Metallgehäuse zu kaufen.

Inzwischen habe ich eine vermutlich wesentlich bessere Lösung mit dem wasserdichten digitalen Temperatursensor DS18B20 gefunden, die ich allerdings noch nicht getestet habe:
https://blog.silvertech.at/arduino-temperatur-messen-mit-1wire-sensor-ds18s20ds18b20ds1820/ Damit sollten

sich auch Störungen auf der Leitung vermeiden lassen. Die Ansteuerung dieses Sensors ist allerdings wesentlich komplizierter und greift daher auf Bibliotheksfunktionen zurück.

2.8 Warnhinweise

Die Wasserkocher-Steuerung ist nur für den experimentellen Betrieb unter ständiger direkter Aufsicht eines Lehrers ausgelegt. Da der Wasserkocher-Schalter dauerhaft auf Einschalten festgeklemmt wird, kann eine Überhitzung mit Brandgefahr nicht ausgeschlossen werden, wenn das vorgeschaltete Relais nicht rechtzeitig ausschalten sollte.

Integrierte Schaltungen sind anfällig für statische Elektrizität. Wenn man das Labor unbedingt im Winter durchführen muss, wenn üblicherweise hohe statische Spannungen auftreten, sollte man entweder auf hohe Luftfeuchtigkeit achten oder mit Erdungsarmband arbeiten. Ferner kann man den Schülern raten, auf Kunststoffkleidung zu verzichten. Da die verwendeten Bauteile alle recht billig und einzeln erhältlich sind, ist weniger der Verlust von Bauteilen ein Problem, als der Zeitaufwand, der benötigt wird um herauszufinden, dass ein Bauteil nur noch unvollständig arbeitet.

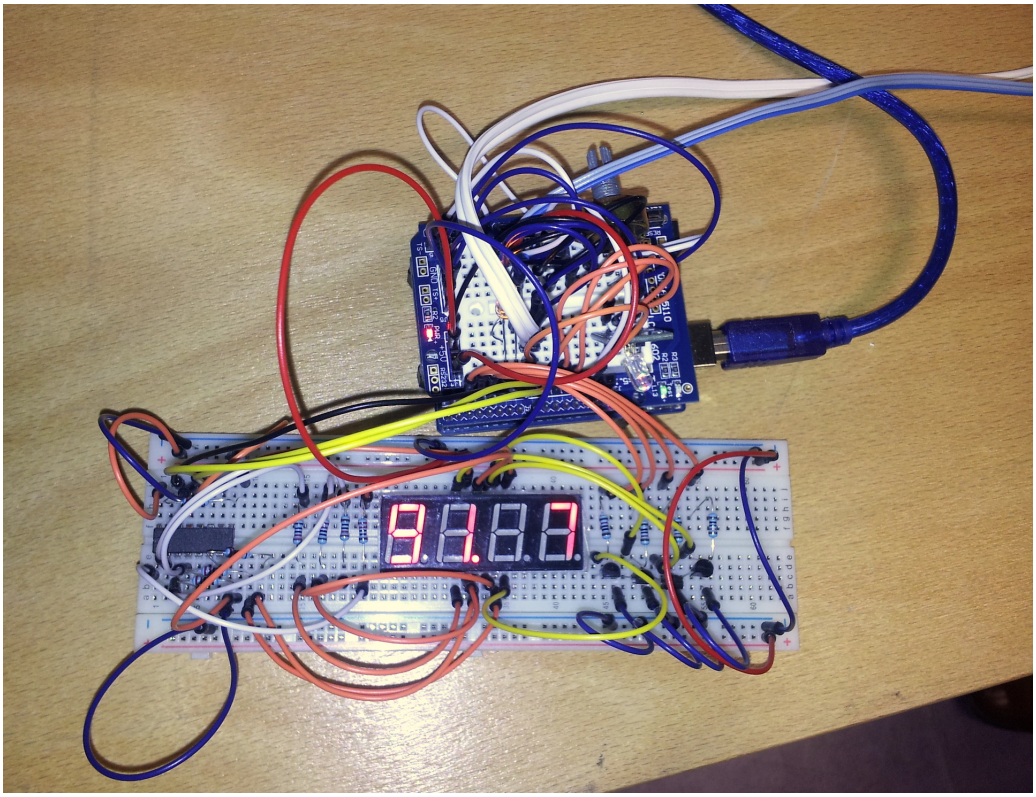

Abbildung 2: Der Aufbau im Detail

3 Installation der Software und erste Schritte

Man kann alle Programme von einem USB-Stick installieren. Eine Internet-Verbindung für die Schüler ist nicht erforderlich. Den Spieleordner kann man von den Schülerrechnern löschen.

Die Arduino-Entwicklungsumgebung findet man hier:
www.arduino.cc/en/Main/Software Oben rechts klickt man auf die zum Betriebssystem passende Version und lädt sie auf den USB-Stick. Die Installation ist problemlos; eine Anleitung wird eher nicht benötigt.

Das Arduino-Language-Reference-Manual findet man zum Online-Nachschlagen hier:
www.arduino.cc/en/Reference/HomePage Will man das ganze Manual auf den USB-Stick kopieren, googelt man nach: Arduino Reference PDF.[1]

Den Schaltplan-Editor Fritzing findet man hier: http://fritzing.org/download/?donation=0 Man lädt die zum Betriebssystem passende Version auf den USB-Stick und entpackt sie. Es gibt keinen Installer – zum Aufruf nur auf die Fritzing-exe-Datei klicken. Der Schaltplan-Editor hat drei Ansichten: Steckplatine (Breadboard) – Schaltplan – Leiterplatte (Platine). Wir verwenden nur die ersten beiden Ansichten. Man sollte sich durch Herumspielen etwas mit dem Editor vertraut machen.

Nun kann man die ersten Schritte wagen: Man schließt einen Arduino-Uno per USB-Kabel an den Host-Rechner an und ruft die zuvor installierte Entwicklungsumgebung auf. Sollte sich der Windows-Firewall beschweren, teilt man ihm mit, dass er das Programm zulassen soll. Es öffnet sich ein kleines Fensterlein mit der Entwicklungsumgebung, in dem schon ein kleines Programmgerüst steht (Programme heißen übrigens beim Arduino nicht Apps, sondern Sketches). Anstatt gleich selbst ein Programm zu schreiben, lädt man ein Beispiel-Programm: Datei → Beispiele → 01.Basics → Blink. Es öffnet sich ein neues Fensterlein (das alte kann man schließen). Dieses Prográmmchen muss nun zuerst übersetzt werden mit dem sog. Cross-Compiler, der ebenfalls in der Entwicklungsumgebung enthalten ist und wird dann per USB-Kabel auf den Arduino geladen. Es ist nun dauerhaft im Arduino gespeichert und man könnte das USB-Kabel entfernen, wenn es nicht auch als Stromversorgung für den Arduino dienen würde (man könnte aber auch eine Batterie oder ein Netzteil verwenden). Vor dem Compile-and-Upload sollte man aber (jedes Mal!) die Grundeinstellungen prüfen:
Werkzeuge (Tools) → Board sollte auf Arduino/Genuino Uno stehen.
Werkzeuge (Tools) → Port sollte eine sinnvolle serielle Schnittstelle anzeigen. Intelligenterweise wird jedes Mal, wenn man das USB-Kabel neu eingestöpselt hat, auch eine neue serielle Schnittstelle verwendet und man muss sie neu einstellen, sonst funktioniert der Upload nicht.
Übersetzen (compile, überprüfen) ohne Upload startet man mit dem Häkchen.
Übersetzen mit anschließenden Upload startet man mit dem Pfeil nach rechts. Falls das Programm Fehler enthält, tauchen im unteren Fenster kryptische Fehlermeldungen auf, die man dann mühsam entschlüsseln muss. Daher sollte der Lehrer schon etwas Erfahrung mit (fehlerhaften) Programmen haben, damit die Fehlersuche schneller geht.

In unserem Blink-Beispiel wird, wenn alles gut geht, nun eine kleine LED auf dem Arduino-Board anfangen zu blinken. Man kann nun z.B. die Blinkfrequenz und das Tastverhältnis verändern, indem man die delay-Werte ändert und wieder compile-and-upload drückt.

[1] Im Internet findet man an vielen Stellen (Unterrichts-) Material zum Arduino, z.B. https://funduino.de/

3.1 Upload-Probleme

Die häufigsten Probleme beim Upload entstehen, wie erwähnt, dadurch, dass der Arduino an der seriellen Schnittstelle nicht gefunden wird. Daher sollte man nach jedem Einstöpseln des Arduino-USB-Kabels verifizieren, ob unter „Tools → Serieller Port“ ein plausibler Portname für den Arduino erscheint. Ggf. einen anderen Port auswählen oder sogar den Treiber neu installieren. Unter Ubuntu-Linux mussten die Rechte auf den Port bei jedem USB-Kabel-Einstöpseln neu gesetzt werden, so dass nicht nur der Administrator Zugriff hat, sondern World. Dafür eine Beispiel-Befehlssequenz:

```
#!/bin/bash
ls -ls /dev/ttyACM*
sudo usermod -a -G dialout <USER-Soundso>
sudo chmod a+rw /dev/ttyACM*
```

3.2 Ein paar Programmierhinweise

- Im Programm-Kopf sollte ein Kommentar stehen mit: Name des Programms, Funktion, Autor, Erstellungsdatum, Versionsnummer.
- Auch sonst sollte mit Kommentaren nicht gespart werden.
- Code sollte leicht lesbar sein: Einrücken, nur eine Anweisung pro Zeile, sprechende Namen. Das Einrücken kann man auch automatisch machen lassen: Werkzeuge (Tools) → Automatische Formatierung.
- Deutsch-Englisch-Mischmasch vermeiden und konsistent in Englisch programmieren.

3.3 Beliebte Programmierfehler

Auf die folgenden üblichen Programmierfehler sollte man vorrangig achten:

- Die Arduino-Programmiersprache c bzw. c++ ist „case-sensitive“, d.h. es wird zwischen Groß- und Kleinschreibung unterschieden. Daher müssen Identifier (Variablen- und Unterprogrammnamen) immer gleich geschrieben werden.
- Sehr beliebt sind Klammer-Fehler: Jeder öffnenden runden oder geschweiften Klammer muss eine ebensolche schließende Klammer zugeordnet sein. Man kann sein Programm mit „Tools → Automatisch formatieren“ einrücken lassen, um die Struktur besser zu erkennen.
- Eine Anweisung endet mit einem Semikolon (außer nach }).

3.4 Verwendung von Programmbibliotheken

Die Wasserkocher-Steuerung wird im Folgenden systematisch aus einzelnen Programmblöcken aufgebaut, die jeweils eine Funktion erledigen. Würde man nun für die Steuerung alle Programmblöcke in eine große Programmdatei zusammenkopieren, wäre das ziemlich unübersichtlich. Daher wird eine Funktion oder eine zusammengehörende Gruppe von Funktionen in eine Bibliothek (library) gestellt, in der die Details verborgen sind. Diese Libraries sind in c++ geschrieben und nicht mehr Gegenstand des Unterrichts.

Wenn Sketches Libraries verwenden, müssen diese vor dem Übersetzen bekannt gemacht werden:

- In Datei → Voreinstellungen ganz oben den Sketches-Ordner eintragen, in dem der libraries-Ordner steht.
- In Sketch → Bibliothek einbinden (library import) die verwendeten Libraries hinzufügen.

4 Schaltungen

Die Beispielschaltungen des Labors (Dateinamenendung .fzz) öffnet man, indem man Fritzing ausführt und die Schaltung mit Datei → öffnen öffnet und zunächst auf die Steckplatinen-Ansicht wechselt. Später kann man auch den Schaltplan betrachten, um sich mit den Symbolen für die Bauelemente vertraut zu machen und den Schaltplan zu verstehen.

Bei LED.fzz sieht man ein kleines Breadboard, das neben der Arduino-Hauptplatine dargestellt ist. Dies soll die Aufsteckplatine mit Breadboard darstellen, die leider in der Bauteilbibliothek nicht vorhanden ist. Das angezeigte Breadboard hat ein paar Spalten mehr. Die rechteste Spalte wird später verwendet, um die schwarzen Spannungsversorgungsbuchsen auf der Aufsteckplatine darzustellen!

4.1 LED und LEDx3

LED.fzz ist die denkbar einfachste Schaltung, bestehend aus nur einer LED und dem Vorwiderstand. Man lernt hier:

Die jeweils fünf Kontakte der oberen und unteren Spalten des Breadboards sind miteinander elektrisch verbunden. Beim großen Breadboard gibt es darüber hinaus noch oben und unten je zwei Reihen für die Spannungsversorgung, die reihenweise ebenfalls verbunden sind. Beim Breadboard sollte man darauf achten, dass man Bauteile vorsichtig in die empfindlichen Kontakte einsteckt. Das ist besonders bei den fetten Anschlussbeinchen des Potentiometers (Potis) schwierig.

Per Konvention verwendet man rote (orange) Drähte für + (5V). (3,3V wird im Praktikum nicht verwendet). Schwarze oder blaue Drähte verwendet man für – (0V, GND = Ground). Für alle anderen Verbindungen verwendet man andere Farben. Die Beachtung dieser Konvention hilft später den Schaltungsaufbau nachzuvollziehen, wenn das Breadboard voll mit Drähten ist.

Die Leuchtdiode (LED) leitet, wie alle Dioden, den Strom nur in einer Richtung. Das kurze Beinchen kommt an Minus (GND). Wenn man sie falsch herum verbindet, bricht sie bei 5V noch nicht durch, sie leuchtet einfach nicht. Wenn man den roten Draht direkt mit 5V verbindet, sollte sie dauerhaft leuchten, ansonsten ist sie tot.

Eine Leuchtdiode braucht einen Vorwiderstand als Strombegrenzung. Wir verwenden 220Ω an 5V. Bei größeren Widerständen wird sie dunkler oder bleibt aus. Ohne Vorwiderstand wird der maximal zulässige Strom durch die LED überschritten (5mm-LED: 20mA). Sie brennt zunächst sehr hell und dann durch. Überschreitet der Strom 40mA, nimmt auf Dauer auch der Mikrocontroller Schaden.

Diese Schaltung verwenden wir für einfache Blinkschaltungen und demonstrieren an ihr auch die Helligkeitssteuerung einer LED. Wenn die LED mit Ausgang 13 betrieben wird, leuchtet sie parallel zu der auf dem Arduino-Board eingebauten LED.

Statt LED und Vorwiderstand kann man auch das Relais (in richtiger Polung!) anschließen und daran eine Glühlampe, die nun statt der LED blinkt. Die Schüler können die Schaltung nun selbst auf drei oder mehr LEDs aufbohren und dafür die Output Ports 9-11 verwenden. Zur Kontrolle ist diese Schaltung in LEDx3.fzz abgebildet. (Selbstverständlich erhält man Glühlampen noch bei Ebay. Und in den Bildekräfte-Seminaren von Ulrike Wendt und Markus Buchmann kann auch jeder nachvollziehen, dass man nichts anderes zur Beleuchtung von Wohn- und Klassenräumen verwenden sollte.)

Die drei LEDs samt ihren Vorwiderständen kann man ersetzen durch die RGB-LED, die man direkt an die Ausgänge 9-11 anschließt, den GND-Anschluss an GND. Sie enthält bereits winzige SMD-Vorwiderstände auf der Platine.

Die Schüler*innen sind meist recht kreativ im Erfinden von eigenen Blinkschaltungen, Ampelschaltungen und Lauflichtern und man sollte dafür ausreichend Zeit lassen.

4.2 IO_Testschaltung

Nachdem nun ausreichend mit LEDs gespielt wurde und auch der Schaltungsaufbau auf dem Breadboard kein Problem mehr darstellt, bauen wir eine Testschaltung, auf der (mit Ausnahme der Displays) alle im Praktikum verwendeten Input- und Output-Devices eingesetzt werden:

- Digitale Inputs: Taster
- Digitale Outputs: LED, Relais
- Analoge Inputs: Poti, Temperatur-Sensor
- Analoge Outputs: RGB-LED

Damit sind alle Anwendungs-Möglichkeiten der Arduino-Ports durch ein Beispiel vertreten. Die Teilschaltungen sind alle unabhängig voneinander und lassen sich daher auch einzeln testen und betreiben, was wir auch zunächst tun, um den Aufbau zu überprüfen. Danach fallen keine Umbauarbeiten mehr an und man kann sich auf die Software konzentrieren. Diese Schaltung ist auch die Schaltung, die für den Wasserkocher benötigt wird.

Der Schaltungsaufbau ist recht gedrängt. Man sollte daher von links anfangen und die Schaltung exakt so aufbauen wie abgebildet. Die rechteste Spalte sitzt außerhalb des Breadboards (schwarze Buchsenreihen) und ist die Spannungsversorgung.

Weiter ist zu beachten, dass die Anschlussreihenfolge der RGB-LED in Fritzing nicht mit der Reihenfolge unserer RGB-LED übereinstimmt. Man baut die RGB-LED so ein, dass sie nach außen strahlt. Dann sollte man auf die Schaltplanansicht wechseln und sie dem Schaltplan gemäß verdrahten.

Den Miniatur-Taster muss man wie dargestellt einbauen, die Beinchen vorne und hinten.

Besonders wichtig ist die Einbaurichtung des Temperatursensors. Sie sollte vor dem Einschalten des Stroms nochmal überprüft werden, sonst könnte er zerschossen werden.

Poti und Temperatursensor haben am Ausgang noch einen kleinen Glättungskondensator (z.B. 1nF) gegen Ground. Beim Poti soll er das Rauschen vermindern. Beim Temperatursensor soll er die Störeinstrahlung auf der Leitung glätten, wenn der Sensor später mit einem längeren Anschlusskabel angeschlossen wird.

Parallel zum LED-Ausgang 13 und GND wird eine 2-Pin-Buchsenleiste angebracht. Sie dient als Platzhalter, wenn dort später das Relais (in richtiger Polung!) angestöpselt wird.

Wenn man zusätzlich zur IO-Testschaltung noch die 4x7-Segment-Anzeige betreibt, sind die IO-Ports des Arduino ausgereizt. Wenn man zusätzlich noch einen Port benötigt, z.B. für einen Summer oder einen Feuchtesensor, könnte man dafür den G-Port der RGB-LED verwenden, die dann nur zweifarbig betrieben wird. Oder man verwendet statt der 4x7-Segment-Anzeige die LCD-Anzeige, die einen Port weniger benötigt. Der Umgang mit einer limitierten Port-Anzahl ist gewollt und führt zur nächsten Schaltung, die zeigt, wie man mit wenigen Ports viele LEDs betreiben kann.

4.3 4x7_Segment-Anzeige_mit_74595

Diese Schaltung ist die komplexeste Schaltung und ihr Aufbau daher optional und nur für die „Cracks“ geeignet. Sie verwendet keine gemeinsamen Ports mit der IO-Testschaltung und kann daher gemeinsam oder unabhängig von ihr betrieben werden. Sie wird auf dem großen Breadboard aufgebaut; aufgrund der gedrängten Bauweise möglichst auch wieder getreu der Vorgabe.

Beschreibung der 4x7-Segment-Anzeige

Jedes der sieben Ziffern-Segmente und der Dezimalpunkt bestehen aus Leuchtdioden. Sie werden von oben beginnend im Uhrzeigersinn mit a-f bezeichnet, der Mittelbalken ist g, der Dezimalpunkt dp. Wenn man diese einzeln ansteuern wollte, würde man $4 \cdot (7+1) = 32$ Anschlüsse für die Anoden (Plus-Pole) der Leuchtdioden benötigen und einen gemeinsamen Kathodenanschluss (Minus-Pol). Das wäre teuer und aufwändig zu verdrahten, daher behilft man sich mit dem sogenannten Multiplexverfahren. Dazu sind die entsprechenden Segmente aller vier Ziffern jeweils intern miteinander verbunden. Die Kathodenanschlüsse aller Segmente einer Ziffer sind ebenfalls miteinander verbunden (c1-c4). Siehe hierzu das Datenblatt: CPS05643AB (A-Version mit gemeinsamer Kathode). Zur Ansteuerung einer LED wählt man nun das entsprechende Segment und die entsprechende Ziffernkathode. Die Multiplexansteuerung erfolgt nun so, dass alle Segmente einer Ziffer gleichzeitig angesteuert werden, die der darzustellenden Zahl entsprechen, also z.B. a-b-c-d-g für 3, und dazu die gemeinsame Ziffernkathode der Ziffer. Danach werden die Segmente der nächsten Ziffer dargestellt, und das reihum für jede Ziffer und dann wieder von vorne. Dieser Vorgang wiederholt sich so schnell, dass für das Auge alle Ziffern gleichzeitig erscheinen. Da die Ziffern nur zeitweise angesteuert werden, haben sie nur ¼ ihrer Maximalhelligkeit. Man könnte also die Vorwiderstände reduzieren, um den Helligkeitsverlust auszugleichen, wobei man aber die Belastbarkeit der Treiber beachten muss. Die gemeinsame Kathode hat hier maximal den achtfachen Strom einer einzelnen LED und überschreitet auf jeden Fall die Fähigkeit des Treibers. Daher wird als Verstärker ein NPN-Transistor zur Ansteuerung der Kathoden verwendet. Wenn man mit diesem Multiplex-Verfahren das 4x7-Segment-Display ansteuert, wären $8+4 = 12$ Ports des Arduino belegt, also fast alle. Wenn man zusätzlich noch die IO-Testschaltung betreiben will, muss man sich mit einem weiteren Trick behelfen. Die vier Kathodentreiber werden ganz normal an den Arduino angeschlossen. Aber die 8 Segmente werden an

die Ausgänge eines 47HC595-ICs angeschlossen und dieser benötigt selbst nur 3 Ports vom Arduino, womit wir 5 Ports eingespart haben. Wie funktioniert das nun wieder? Zu diesem IC führen eine Datenleitung (DS), ein Schiebetakt (SH_CP) und ein Ausgangstakt (ST_CP). Das IC enthält einen Seriell-zu-Parallel-Wandler: Die Daten (an oder aus) für die einzelnen Segmente werden einzeln nacheinander mit jedem Schiebetakt über die Datenleitung in ein Schieberegister hineingeschoben. Anschließend erfolgt ein Ausgangstakt, mit dem sie parallel vom Schieberegister in ein Ausgangsregister übertragen werden und dort stabil am Display anliegen. Währenddessen können schon die seriellen Daten für die nächste Ziffer in das Schieberegister geschoben werden. Siehe hierzu auch das Datenblatt des 74HC595.
Beim Aufbau ist zu beachten, dass man die feinen Anschlussdrähte der 4x7-Segment-Anzeige nicht verbiegt.

Vor dem eigentlichen Aufbau empfiehlt es sich, die 7-Segment-Anzeige kennenzulernen. Dazu testet man zunächst die 1x7-Segment-Anzeige (Datenblatt: CL5611AH), indem man GND an Pin 3 oder 8 anschließt und dann die einzelnen Segmente nacheinander per Vorwiderstand an 5V anschließt. Das wiederholt man mit der 4x7-Segment-Anzeige (Datenblatt CPS05643A), wobei man jetzt zunächst nur jeweils eine der vier Kathoden c1-c4 an GND anschließt.

4.4 LCD1602

Anstelle der 4x7-Segmentanzeige könnte man auch das LCD-Display verwenden, das auch mehr Möglichkeiten bietet, da es zwei Reihen zu je 16 Zeichen (inkl. Buchstaben und Sonderzeichen) darstellen kann und sogar mit einem Port weniger auskommt.

Allerdings kann man daran wenig lernen. Es enthält selbst wieder einen eigenen Controller, der die Multiplexansteuerung übernimmt und mit einer eigenen Programmbibliothek angesteuert wird, so dass weder Hardware noch Software transparent sind. Es wird daher nicht direkt für unser Projekt verwendet, kann aber in weiterführenden Schülerprojekten eingesetzt werden. Der Aufbau kann auf dem kleinen oder dem großen Breadboard erfolgen. Die Schaltung kann mit dem Programm WK_LCD getestet werden.

5 Programme

Programme heißen beim Arduino eigentlich Sketches und müssen jeweils in einem eigenen Ordner stehen, der genauso benannt ist wie das Programm, aber ohne die Endung .ino.

5.1 Übersicht über die Sketches

WK_AllPinsToInput	Schaltet alle I/Os auf Input; Die Anwendung empfiehlt sich vor neuen Schaltungsaufbauten.
WK_Blink	LED 13 blinkt
WK_LED_sequencer	Drei einzelne LEDs als Lauflicht
WK_Morse	Morse-Code senden
WK_Pseudo-PWM	Wie WK_Blink, aber so schnell, dass man einen Dimmer hat.
WK_PWM_Fade	Hardware-PWM mit Fading
WK_AnalogReadSerial	Poti auslesen, in Spannung umwandeln, auf PWM-Wert mappen, LED-Helligkeit steuern, an seriellen Monitor ausgeben
WK_TEMP_Poti	Poti auslesen, in Temperaturwert umwandeln, Ausgabe am seriellen Monitor
WK_TEMP_SENSOR	Temperatur-Sensor auslesen, Ausgabe am seriellen Monitor
WK_RGB_Poti	Farbtemperatur der RGB-LED mit Poti einstellen (1 oder 3 Potis)
WK_RGB_TEMP_SIMPLE	Farbtemperatur der RGB-LED wechselt, mit Table-Lookup
WK_RGB_TEMP	dito, aber mit Interpolation und optionalem Pulsing
WK_RGB_TEMP_lib	WK_RGB_TEMP mit library
WK_RGB_TEMP_SENSOR	Temperatursensor oder Poti und RGB-LED
WK_button	Taster schaltet LED ein und aus
WK_button_millis	Taster mit anderem Entprellungsverfahren
WK_button_timeout	Zusätzlich: Ausschaltung nach Zeitablauf (Treppenhaus-Automat)
WK_SevenSeg_4_A	4x7-Segment-Anzeige, einfachste Ansteuerung
WK_SevenSeg_4_B	4x7-Segment-Anzeige mit Unterdrückung führender Nullen und einstellbarem Dezimalpunkt
WK_SevenSeg_4_C	Wie WK_SevenSeg_4_B, aber mit library
WK_SevenSeg_2x2	4x7-Segment-Anzeige mit Split der Anzeige in 2x2 Stellen (für Soll- und Ist-Temperatur)
WK_water_kettle	Kombination von WK_button_timeout, WK_RGB_TEMP_SENSOR, (optional) WK_SevenSeg_2x2
WK_FillLevel	Füllstandsmessung für Wasserkocher (optional, nicht integriert in die Wasserkocher-Software)
WK_LCD	LCD-Anzeige (optional, nicht integriert in die Wasserkocher-Software)
Math_Fakultaet	Programm zur Berechnung der Fakultät
Math_SiebErathosthenes	Programm zur Berechnung der Primzahlen

5.2 Zugehörige Breadboard-Aufbauten

Die ersten LED-Programme benötigen nur die LED oder LEDx3 –Breadboard-Aufbauten. Schaltungen nur mit LED 13 funktionieren auch ohne Aufbau einer Zusatzschaltung mit der LED13 auf dem Arduino-Board.
Spätestens für WK_AnalogReadSerial sollte man die IO-Testschaltung aufbauen und bis zum Schluss verwenden.
Die Schaltungen mit der Siebensegmentanzeige benötigen den Breadboard-Aufbau 4x7_Segment-Anzeige_mit_74595, die optionale Schaltung WK_LCD benötigt den LCD1602-Aufbau.

Für das Morse-Programm und beim Testen des Wasserkochers schließt man zusätzlich zur LED noch das Switchbox-Relais richtig gepolt an Pin 13 an.

Fakultaet und SiebErathosthenes benötigen nur den Arduino ohne Zusatzhardware. Das sind kleine Programme, die man an beliebiger Stelle zur Entspannung einschieben kann, wenn man mal keine Lust auf Hardware hat und demonstrieren will, dass der Arduino nicht nur LEDs blinken lassen kann, sondern ein vollwertiger Rechner ist.

5.3 Sicherheitsprogramm (WK_AllPinsToInput)

Dieses Prográmmchen setzt die Richtung aller bidirektionalen Ports des Arduino auf Input. Man kann es verwenden, bevor man ein neues Breadboard an den Arduino anschließt, das eine andere Anschlussbelegung hat als das zuvor verwendete. Damit vermeidet man, dass zwei Ausgänge mit verschiedenen Werten gegeneinander treiben.

Beispiel: Das Breadboard hat vielleicht einen Schalter, der in Richtung des Arduino-Ports 13 eine „1“ (5V) sendet, während dieser Port aber selbst als Ausgang geschaltet ist und in Richtung der eingebauten LED13 eine „0“ (0V) sendet. Es kommt an Port 13 zu einem Kurzschluss und der Arduino raucht ab. Indem man alle Ports als Eingänge schaltet, kann das vermieden werden.

5.4 Digitale Ausgabe (WK_Blink und WK_LED_sequencer)

Zunächst lassen wir eine oder mehrere LEDs blinken und lernen damit die digitale Ausgabefunktion des Arduino kennen.

WK_Blink ist das einfachste aller Arduino-Programme. Es bringt die auf dem Arduino-Board eingebaute LED und ggf. zusätzlich die LED auf dem Breadboard (LED.fzz) zum Blinken.

Aufgaben:

- Verschiebe den Programmteil, der im loop()-Block steht, in den setup()-Block und beachte die Veränderung.
  Erkläre die Bedeutung der setup()- und loop()-Blöcke.
- Schlage im Referenz-Manual die Bedeutung der Funktionen digitalWrite und delay nach.
- Verändere die Einschalt- und Ausschaltzeiten. Daran lässt sich der Begriff Tastverhältnis erklären.

- Erzeuge durch eine längere Sequenz von digitalWrite und delay einen (bekannten) Blink-Rhythmus und lasse den Rhythmus von Mitschülern erraten.
- Lauflicht:
  Erweitere die Schaltung auf drei LEDs (an Pins 9, 10, 11).
  Erweitere das Blink-Programm so auf drei LEDs, dass diese nacheinander aufleuchten.
  (Die LED an Pin 13 kann auch belassen werden für die nachfolgende Morse-Schaltung).
  (Lösung in LEDx3.fzz und WK_LED_sequencer)
- Mit weiteren LEDs und erweitertem Lauflicht-Programm lassen sich weitere interessante Lichteffekte erzeugen. Welche Gruppe hat die spektakulärste Idee?
- Programmiere eine Ampelschaltung (grün-gelb-rot-rot/gelb-grün). (Wenn keine grüne LED zur Hand ist, kann man stattdessen eine blaue LED verwenden).
- Ersetze die drei LEDs durch das RGB-LED-Modul. Die Vorwiderstände können ebenfalls entfallen, da auf dem LED-Modul bereits Vorwiderstände enthalten sind. Beobachte, wie sich die Farben mischen, wenn mehrere LEDs gleichzeitig aufleuchten.

Lernaufgaben: H/W: Einsatz der LED und des RGB-Moduls. S/W: setup, loop, digitalWrite, delay.

5.5 Morse-Sender (WK_Morse)

Mit unseren Kenntnissen können wir bereits einen Morse-Sender bauen.

Für den Morse-Sender wird wieder die Schaltung mit einer LED an Pin 13 benötigt (LED.fzz). WK_Morse sendet in der Loop Sequenzen von dots (Punkten), dashes (Strichen) und eols (End of Letter, Pause zwischen den Buchstaben). Ein Strich ist dreimal so lang wie ein Punkt. Zwischen Strichen und Punkten ist eine Pause von einer Punktlänge, zwischen den Buchstaben eine Pause von drei Punktlängen. Dot(), dash(), eol() sind selbst wieder kleine Unterprogramme, die die bereits gelernten Befehle zur Ansteuerung einer LED enthalten. Statt Unterprogramme zu verwenden, könnte man auch bei jedem Aufruf ihren Inhalt an ihre Stelle setzen, was aber sehr umständlich ist.

- Teste das Morse-Programm WK_Morse und versuche den gesendeten Morsespruch zu entschlüsseln. Die Code-Tabelle findet man auf Wikipedia.
- Ersetze den vorgegebenen Morsespruch durch deinen Namen.
- Schalte zusätzlich an Pin 13 und GND das Antrax-Switchbox-Relais (auf die richtige Polung achten) und schließe daran eine Glühlampe an (Quecksilberdampflampen sind ungeeignet).
- Sende einem Partner über eine Distanz eine Morsebotschaft, die er entschlüsseln muss.
- Alternativ kann man statt des Relais an Pin 13 und GND auch den aktiven Buzzer schalten (auf die richtige Polung achten; der aktive Buzzer ist unten schwarz).

- Für Durchblicker: Das Programm lässt sich komfortabler gestalten, indem man für jeden Buchstaben und jede Zahl wiederum ein Unterprogramm baut, das die zugehörige Sequenz aus dots, dashes und eol enthält.
  Beispiel: void Morse_A() {dot(); dash(); eol();}
  In der loop ruft man nun nur noch Morse_A() auf, um ein „A“ zu senden. Lernaufgaben: H/W: Einsatz des Switchbox-Relais. S/W: Unterprogramme.

5.6 Software-Pulsbreitenmodulation (Pseudo-PWM)

Verwende nochmals die Schaltung und das Programm für WK_Blink.

- Reduziere die Einschalt- und die Ausschaltdauer auf 1ms respektive 2ms.
  (Lösung: WK_Pseudo-PWM)
  Was passiert? Was passiert, wenn man die Ausschaltdauer weiter in 1ms-Schritten erhöht?
  Die Helligkeitssteuerung durch schnelles Ein- und Ausschalten nennt man Pulsbreitenmodulation (PWM). Das Tastverhältnis ist dabei der Quotient Einschaltdauer/Ausschaltdauer.
- Filme die LED mit dem Smartphone und lasse den Film in Slow-Motion ablaufen. Was sieht man?
  (iPhone5s: In der Kamera-App SLO-MO auswählen. Dadurch wird mit der vierfachen Geschwindigkeit von 120 Bildern/Sekunde aufgenommen und später mit Normalgeschwindigkeit abgespielt. Android-Smartphone: Die Hardware erlaubt derzeit keine Aufnahme mit höherer Geschwindigkeit. Man kann mit VivaVideo aber das Video beim Abspeichern verlangsamen [unter Capture – Slow Motion]).
- Lass Dir vom Lehrer mit dem Oszilloskop die Kurvenform an der LED zeigen. (Y: 1V/cm, X: 1ms/cm)

Lernaufgaben: S/W-Pulsbreitenmodulation und Tastverhältnis.

5.7 Hardware-Pulsbreitenmodulation (WK_PWM_Fade)

Die Helligkeitssteuerung, die mit WK_Pseudo-PWM per Software erreicht wurde, ist im Arduino hardwaremäßig an einigen, aber nicht an allen Ports implementiert (3,5,6,9,10,11). Statt der Funktion digitalWrite mit den Werten HIGH (an) und LOW (aus) verwendet man die Funktion analogWrite mit Werten zwischen 0 (dauerhaft aus) und 255 (dauerhaft ein). Das wird im Programm WK_PWM_Fade gezeigt. Hier wird die Helligkeit (brightness) der LED an Pin 9 bei jedem loop-Durchlauf um einen bestimmten Wert (fadeAmount) verändert. Wenn die Maximalhelligkeit erreicht wurde, wird wieder in Richtung Null gedimmt und umgekehrt. Das nennt man Fading.

Aufgaben:

- Beobachte wieder das schnelle Ein- und Ausschalten der LED mit der Kamera oder dem Oszilloskop.
- Modifiziere das Programm WK_PWM_Fade so, dass statt einer anschwellenden und abschwellenden Helligkeitssteuerung (Dreiecks-Kurve) eine ansteigende Helligkeit mit abruptem Übergang von maximaler Helligkeit auf Null stattfindet (Sägezahn-Kurve).

- Für Durchblicker: Erweitere das Programm so, dass die LED zusätzlich auch noch blinkt, wie bei WK_Blink.
- Erweitere Schaltung und Programm so, dass die RGB-LED angesteuert werden kann. Verwende dabei unterschiedliche Startwerte für die jeweiligen Farben.

Lernaufgaben: H/W-Pulsbreitenmodulation mit analogWrite.

5.8 Analog-Eingabe

(WK_AnalogReadSerial, WK_TEMP_Poti, WK_TEMP_SENSOR, WK_RGB_Poti)

Nun wird erstmals die IO-Testschaltung benötigt, die spätestens jetzt aufzubauen ist. Davon wird zunächst nur das Poti, der Temperatursensor und eine LED der RGB-LED benötigt. Das Poti vorsichtig auf das Breadboard stecken, damit die Kontakte des Boards nicht beschädigt werden!

Ferner wird hier der Host-Rechner zur Ausgabe von Messwerten benötigt.
Dazu wird eine serielle Kommunikationsverbindung über das USB-Kabel aufgebaut, das sonst nur zum Laden des Programms und zur Stromversorgung dient. Die Verbindung wird beim Arduino initialisiert, indem man ihm einmalig mit Serial.begin(9600) mitteilt, dass er eine serielle Verbindung mit 9600 Baud (Bits pro Sekunde) aufbauen soll. Mit Serial.print-Befehlen kann man nun Zahlenwerte oder Text zum Host senden. Eine Zeile wird abgeschlossen, indem man den Serial.println-Befehl verwendet. Damit der Host-Rechner die Werte auch anzeigt, muss man jedes Mal nach dem Upload des Programms ein Monitor-Fenster aufmachen: Werkzeuge (Tools) → Serieller Monitor.

Das hier verwendete Potentiometer ist ein Kohlewiderstand, der an den beiden Enden an Plus und Minus angeschlossen ist. Der Mittelanschluss liegt an einem Schleifer, der sich zwischen Plus und Minus bewegen lässt. Damit lassen sich alle Spannungswerte zwischen 0V und 5V einstellen.

Die Poti-Spannung liegt an dem Arduino-Analog-Port A0 an. Da der Arduino allerdings intern nicht mit beliebig vielen Spannungswerten rechnen kann, sondern nur mit Einsen (dargestellt als 5V) oder Nullen (0V), wird die Spannung zunächst in eine zehnstellige Dualzahl umgewandelt, die also 1024 verschiedene Werte annehmen kann (0 für 0V bis 1023 für 5V). Die Umwandlung macht ein Analog-zu-Digital-Wandler (A/D-Wandler), eine komplizierte Schaltung, deren Aufbau wir nicht verstehen müssen. Hauptsache sie tut, was sie soll.

Ferner wird der Temperatursensor LM35 verwendet. Das ist eine kleine integrierte Schaltung mit drei Beinchen, die neben den Versorgungsspannungen an einem weiteren Pin die Temperatur in Grad Celsius als Spannungswert ausgibt. Dabei entsprechen 0°C 0V und 100°C 1V, je 1°C verändert sich die Spannung also um 10mV.

Programmschritte von WK_AnalogReadSerial:

Zunächst wird der binär-gewandelte Spannungswert vom A/D-Wandler gelesen. Er wird dann von einer Ganzzahl (Integer, abgekürzt int) im Bereich 0-1023 in einen Wert von 0.0 bis 5.0 als Gleitpunktzahl (float) umgewandelt, so dass man die gemessene Spannung am Monitor beobachten kann. Eine Gleitpunktzahl wird im Rechner mit Mantisse und Exponent dargestellt. Eine Ganzzahl kann keine „ungeraden“ Spannungen von z.B. 2.7V darstellen, wohl aber eine Gleitpunktzahl. Wenn eine Konstante eine Nachkommastelle hat, erkennt der Rechner sie als Gleitpunktzahl (z.B. 1023.0). Daher sollten alle Zahlen in Gleitpunktberechnungen auch als Gleitpunktzahlen dargestellt sein (z.B. 5.0 und nicht 5).

Ferner wird der Wert vom A/D-Wandler vom Bereich 0-1023 auf den Bereich 0-255 abgebildet. Das könnte man machen, indem man ihn einfach durch 4 teilt oder die beiden letzten Stellen abschneidet. Eleganter geht das mit der map-Funktion (siehe Referenz-Manual). Die Bereichstransformation ist nötig, weil wir den Wert zur Helligkeitssteuerung einer LED mit analogWrite ausgeben wollen, diese Funktion aber ungeschickterweise nur den Bereich von 0-255 zulässt.

Zur Kontrolle senden wir den Wert vom A/D-Wandler (0-1023), die errechneten Spannungswert (0.0-5.0) und den Wert zur Helligkeitssteuerung der LED (0-255) auf den seriellen Monitor. Wir haben damit nebenbei ein Voltmeter gebaut!

Aufgaben:

- Ersetze die Map-Funktion durch eine Division durch 4.
- Was passiert, wenn man den Sensorwert direkt wieder zur Helligkeitssteuerung ausgibt, ohne Map oder Division?
- Gebe zu den reinen Werten, die an den Monitor geschickt werden, noch Kommentare aus, z.B.: Sensor: xxx, Spannung: yy.yV, Helligkeit: zzz.
- Miss die Spannung am Poti-Schleifer und vergleiche sie mit dem Spannungswert im Monitor.
- Temperatureinstellung:
  Mit dem Poti soll nun eine Soll-Temperatur für eine Heizung eingestellt werden können.
  Die Werte vom A/D-Wandler müssen daher auf den Temperaturbereich von 0-100 (in °C) abgebildet werden. Verwende dazu die map-Funktion. Eine Umrechnung in Spannung ist nun unnötig. Sende den A/D-Wandlerwert und den Temperaturwert an den Monitor.
  (Lösung: WK_TEMP_Poti).
  Dieses Programm wird als Teil unserer Wasserkocher-Steuerung benötigt. Es wurde zum einfacheren Gebrauch in der ReadTemp-Bibliothek versteckt und heißt ReadTempCPoti.
- Temperaturmessung:
  Mit dem Temperatursensor am Analog-Port A5 wird nun eine Ist-Temperatur gemessen. Das kann die Raumtemperatur oder eine Heizungstemperatur sein. Der Wert vom A/D-Wandler soll zunächst wieder in eine der Spannung entsprechende Gleitpunktzahl umgewandelt werden.
  Von diesem Spannungswert wird eine Offset-Spannung subtrahiert (diese ist allerdings bei dem verwendeten Sensortyp 0.0V).
  Da bei diesem Sensor 10mV = 0.01V einem Grad Celsius entsprechen, muss die Spannung mit 100.0 multipliziert werden, um den Temperaturwert in Grad zu erhalten.
  Die Temperatur soll nun durch Rundung in eine Ganzzahl verwandelt werden.
  Dazu addiert man 0.5 und schneidet anschließend die Nachkommstellen ab.
  Das Abschneiden funktioniert hier so, dass man durch sogenanntes type casting mit vorangestelltem int in Klammern die Umwandlung einer Gleitpunktzahl in eine Ganzzahl erzwingt:
  int Temp_int = (int) Temp_float;

Anschließend sendet man zur Kontrolle alle Zwischenergebnisse an den Monitor. Versuche mit dem Programm soweit zu kommen wie möglich und schaue dir erst dann das Programmierbeispiel in WK_TEMP_SENSOR an.
Teste die Schaltung, indem Du die angezeigte Temperatur mit einem Thermometer für die Raumtemperatur vergleichst, dann den Sensor erhitzt (Fön) oder abkühlst (anblasen).
Dieses Programm wird als Teil unserer Wasserkocher-Steuerung benötigt. Es wurde zum einfacheren Gebrauch in der ReadTemp-Bibliothek versteckt und heißt ReadTempC.

- Optional: Baue ein Temperaturmessgerät, indem Du die Temperatur über das LCD-Display ausgibst. Verwende dazu zusätzlich die Schaltung LCD1602 und das Programm WK_LCD.

- Verwende das große Breadboard und baue eine Schaltung mit drei Potis und der RGB-LED auf, so dass sich die einzelnen Farben individuell steuern lassen. Oder verwende ein Poti und steuere damit zwei oder drei Farben der RGB-LED (dazu ist kein neuer Aufbau nötig). (Beispielprogramm in WK_RGB_Poti). Die Bedeutung der Direktiven zur bedingten Übersetzung (#define, #ifdef, #ifndef, #endif findet man im Internet. Je nachdem, ob die Zeile #define THREE_POTIS vorhanden ist, werden unterschiedliche Code-Abschnitte übersetzt, andere ignoriert. Man kann dadurch ein Programm für beide Versionen (mit einem oder mit drei Potis) verwenden.

Lernaufgaben: H/W: Poti, Temperatursensor LM35. S/W: Datentypen int und float, analogRead, map, Serial.*, Umrechnung, bedingte Übersetzung.

5.9 RGB-LED-Spielereien

(WK_RGB_TEMP_SIMPLE, WK_RGB_TEMP, WK_RGB_TEMP_lib, WK_RGB_TEMP_SENSOR)

Schön wäre es doch, wenn man die Farbtemperatur der RGB-LED in Abhängigkeit von der Temperatur ändern könnte. Dazu werden hier einige etwas aufwändigere Programme vorgestellt, die wir jetzt analysieren.

WK_RGB_TEMP_SIMPLE

Nachdem wie üblich dem Arduino mitgeteilt wird, an welchen Pins die RGB-LED sitzt, folgt eine Tabelle aus drei sogenannten Arrays. In dieser Tabelle sind Helligkeitswerte (0-255) jeweils für die drei Grundfarben gespeichert, und zwar ein Wert für jeweils einen Zehn-Grad-Bereich. Selbstverständlich wäre auch eine Tabelle mit einer individuellen Farbeinstellung für jedes einzelne Grad möglich, aber eine Tabelle mit 3x101 Werten zu erstellen und einzutippen ist nicht gerade spaßig.

In der loop passiert nichts anderes, als dass in einer weiteren sogenannten for-Schleife alle Temperaturwerte von -10 bis 110 Grad in 5-Grad-Schritten ausprobiert werden. Die Beschreibung der for-Schleife findet man im Referenz-Manual. Nun wird das Unterprogramm RGB_LED mit der momentanen Temperatur aufgerufen.

Dieses Programm beschneidet zunächst die vorgegebene Temperatur auf einen Wert zwischen 0 und 100, denn für andere Werte ist unsere Tabelle gar nicht vorgesehen. Die if-else-Anweisung findet man im Referenz-Manual. Da unsere Tabelle die Temperatur für 0 Grad an Stelle 0 enthält, für 10 Grad an Stelle 1, für 100 Grad an Stelle 10, teilen wir die Temperatur durch 10 und können damit direkt in der Tabelle die Helligkeitswerte

für die drei Farben auslesen. Anschließend senden wir die Werte zur Kontrolle an den Monitor und übergeben die Helligkeitswerte an analogWrite zur PWM-Ansteuerung der LEDs.

Wenn wir das Programm austesten, zeigt sich, dass unsere Faulheit dadurch bestraft wird, dass die Farbtemperatur sich doch in unangenehmen Farbsprüngen ändert. Wir können zwar noch die einzelnen Farben an unseren Geschmack durch Änderungen in der Tabelle anpassen, die Sprünge jedoch nicht vermeiden.

Also streichen wir SIMPLE und schauen uns WK_RGB_TEMP an:

Ganz oben haben wir einen neuen Datentyp: bool. Dieser kann nur zwei Werte annehmen, wahr oder falsch (1 oder 0).

Im Setup-Teil finden wir ein Unterprogramm RGB_LED_setup. Darin wird wie üblich die Port-Direction als Output festgelegt.

Im Loop-Teil werden wieder alle Temperaturen durchprobiert, diesmal in einzelnen Grad-Schritten. Als Besonderheit findet man eine weitere for-Schleife, die einfach RGB_TEMP viermal hintereinander aufruft (warum zeigt sich später).

Das eigentliche Geschehen findet in RGB_TEMP statt.
Den ersten Teil mit der Puls-Erzeugung ignorieren wir zunächst.
Die Temperatur wird diesmal auf 0 bis 99 Grad begrenzt.
Da wir wieder eine Tabelle mit 11 Einträgen verwenden, wird die Temperatur wieder durch 10 geteilt. Nun machen wir eine Farbinterpolation: Wir holen uns aus der Tabelle den Farbwert, der (in Zehnerschritten) der Temperatur kleiner gleich der aktuellen Temperatur entspricht und den, der (in Zehnerschritten) größer der aktuellen Temperatur ist. Ferner berechnen wir noch die Einerstelle der Temperatur (T_rest). Mit den unteren und oberen Farbwerten und T_rest berechnen wir nun einen Farbwert, der entsprechend T_rest dazwischen liegt. Dieser Farbwert wird nun noch mit einem Gesamthelligkeitswert (pulse) multipliziert und anschließend in eine Ganzzahl verwandelt und per analogWrite an die LEDs geschickt.

Als weiteres Gimmick haben wir die (optionale) Puls-Erzeugung, die wir bisher ignoriert hatten. Die macht nichts anderes als den Farbwert noch mit einer pulsierenden Helligkeit zu überlagern, um etwas Leben in die Anzeige zu bringen. Dazu wird bei jedem Aufruf von RGB_TEMP ein Programmaufrufzähler (program_call_incrementer) hochgezählt. Aus diesem Wert erhalten wir mit dem Modulo-Operator (%) den Rest von einer Division durch 128. Dieser Rest (timeindex) bewegt sich also zunächst zwischen 0 und 127 und wird mit jedem Aufruf hochgezählt, kippt aber nach 127 wieder auf 0. Wir operieren aber noch ein bisschen an unserem timeindex herum, so dass er nur aufwärts bis 64 zählt und anschließend abwärts bis 0 und dann wieder von vorne. Damit erhalten wir also einen dreieckförmigen Puls. Wir teilen ihn durch 64, so dass er immer kleiner gleich 1.0 ist. Mit diesem Puls steuern wir die Gesamthelligkeit aller LEDs.

So, dass war doch schon mal ein kleines Programm, das sich nicht „Prográmmchen“ schimpfen lassen muss! Wenn man es aber als Teil eines noch umfangreicheren Programms verwenden will, wird es doch schnell unübersichtlich. Daher verstecken wir die Funktion RGB_TEMP in einer Bibliothek namens RGBLED im libraries Ordner und stellen sie dem übergeordneten Programm zur Verfügung. Das wird dadurch wesentlich einfacher, siehe WK_RGB_TEMP_lib.

Nun können wir Temperatureinstellung oder –messung aus dem vorherigen Kapitel mit der Temperaturanzeige via RGB-LED zusammenpacken. Das Ergebnis findet man in WK_RGB_TEMP_SENSOR. In der loop wird die Soll-Temperatur vom Poti gelesen (Temp_Nom) und die Ist-Temperatur vom Temperatursensor (Temp_Act). Eine der beiden Temperaturen kann man mit der RGB-LED anzeigen (derzeit ist Temp_Nom voreingestellt, da sie sich leichter verändern lässt als die gemessene Temperatur Temp_Act).

5.10 Digitale Eingabe mit einem Taster

(WK_button, WK_button_millis, WK_button_timeout)

Um eine Maschine in Gang zu setzen, bedient man sich meist eines Tasters, der an den Arduino nur zwei Informationen sendet: gedrückt („1“) oder nicht gedrückt („0“). Den aktuellen Zustand des Tasters kann man mit der Funktion digitalRead() lesen. Zum Einschalten der Maschine drückt man einmal den Taster, zum Ausschalten noch einmal. Die Dauer, wie lange man drückt, ist egal.

In der Programm-Loop muss also festgestellt werden, ob der Taster erstmals gedrückt wurde, während er im vorhergehenden Durchlauf nicht gedrückt war. Man nennt das: Erkennung der steigenden Flanke. Mit der steigenden Flanke wird die Maschine eingeschaltet. Darauf folgt logischerweise eine fallende Flanke. Mit der nächsten steigenden Flanke wird die Maschine wieder ausgeschaltet.

Exakt das macht das Programm WK_button:

In jedem Schleifendurchlauf wird der aktuelle Zustand des Schalters in current_button_state mit digitalRead eingelesen. Wenn dieser Zustand nicht dem vorherigen entspricht, dann wurde offensichtlich der Schalter betätigt, wenn er dann noch „1“ ist, haben wir eine steigende Flanke. Dann wird die Maschine eingeschaltet, wenn sie zuvor nicht eingeschaltet war oder ausgeschaltet, wenn sie zuvor eingeschaltet war (d.h. der Wert von running wird gerade umgekehrt, was der !-Operator macht). Nach jedem Schleifendurchlauf wird dann der aktuelle Schalterzustand abgespeichert, damit er zu Beginn des nächsten Durchlaufs mit dem neuen Wert verglichen werden kann.
Nun gibt es noch eine mechanische Besonderheit der Taster: Da die Kontaktfläche eines Tasters aus Metall mit winzigen Rauhigkeiten besteht, kommt es nicht selten vor, dass der Taster beim Einschalten mehrfach zwischen 0 und 1 hin- und herschaltet. Das Programm würde dann mehrfache steigende Flanken erkennen, wo nur eine erkannt werden sollte. Man nennt das Kontaktprellen (Bouncing).
Wenn man z.B. einen Zähler bauen möchte, der mit jedem Tastendruck um eins erhöht wird, könnte es passieren, dass mit jedem Tastendruck gleich mehrere steigende Flanken ausgelöst werden und der Zähler um ein unbestimmtes Inkrement erhöht wird. Eine Maschine würde evtl. eingeschaltet und sofort wieder ausgeschaltet.
In WK_button wird das verhindert, indem man nach jeder Flanke einige Millisekunden wartet, bis sich der Taster-Zustand stabilisiert hat und während dieser Zeit weitere Flanken ignoriert.

Aufgabe:

- Kommentiere das delay-Statement aus und überprüfe die Schaltung erneut. Es könnte nun manchmal passieren, dass die LED mit einem Tastendruck eingeschaltet und sofort wieder ausgeschaltet wird, so dass man sie nicht einmal aufblitzen sieht.

Ein Nachteil des WK_button-Programms ist die Verwendung eines delays. Wenn weitere Programmteile delays benötigen, dann lässt sich die Gesamtverzögerung nicht mehr kontrollieren.

Im Programm WK_button_millis verwenden wir daher im Prinzip das gleiche Debouncing-Verfahren, aber statt eines Delays verwenden wir eine Zeitmessung. Auf eine Flankenänderung wird jetzt nur reagiert, wenn die vorherige Flankenänderung 50ms vergangen war.

Das Programm enthält aber nun einen Fehler ähnlich dem des Jahr-2000-Problems: Die Zeitmessung erfolgt in Millisekunden seit dem letzten Upload oder Reset. Zum Speichern der Zeit verwenden wir zwar einen unsigned long int Wert mit 32 bits, der bis zu 4,294,967,295 ms zählen kann. Nach knapp 50 Tagen läuft aber dieser Zähler über und fängt wieder bei 0 an. Zuvor sollten wir also unsere Schaltung neu in Betrieb nehmen, sonst funktioniert sie nicht zuverlässig.

Das WK_button_millis-Programm wird nun noch etwas aufgebohrt, so dass es noch verschiedene Aktionen ausführen könnte zum Startzeitpunkt und zum Endzeitpunkt.
Ferner enthält es noch einen Timeout-Mechanismus, der ein laufendes Gerät ausschaltet, wenn eine gewisse Zeit überschritten ist. Damit können wir z.B. einen komfortablen Treppenhausautomaten bauen, der auf einen Tastendruck das Licht einschaltet, auf einen weiteren Tastendruck das Licht wieder ausschaltet oder es ausschaltet, wenn eine voreingestellte Zeit verstrichen ist.
Für einen Wasserkocher wird ebenfalls dieser Timeout-Mechanismus benötigt: Es könnte ja sein, dass der Temperatursensor defekt ist und nie die voreingestellte Soll-Temperatur erreicht wird. Dann verdampft das ganze Wasser und der Kocher überhitzt. Um die Brandgefahr zu reduzieren, wird er von unserem Programm nach einer gewissen Zeit wieder ausgeschaltet. (In einem käuflichen Wasserkocher ist ein Übertemperaturschutz eingebaut).

Aufgabe:

- Teste das WK_button_timeout-Programm und verfolge am Monitor die einzelnen Werte, um die Funktionsweise zu verstehen.
- Versuche den Überlauffehler des Zählers abzufangen.

Wie man sieht, kann die Bedienung eines einfachen Tasters ganz schön kompliziert werden!

5.11 4x7-Segment-Anzeige

(WK_SevenSeg_4_A/B/C, WK_SevenSeg_2x2)

Die 4x7-Segmentanzeige kann bis zu vierstellige Dezimalzahlen anzeigen. Dafür muss die Schaltung 4x7_Segment-Anzeige_mit_74595 aufgebaut und verstanden werden. Die Schaltung kann als Erweiterung der Wasserkocher-steuerung dienen, ist aber nicht unbedingt erforderlich.

WK_Sevenseg_4_A

Das Programm basiert auf dem Code von Paul Jenkins.
Zunächst werden in einem Array für jede Ziffer von 0 bis 9 und A bis F (zur Darstellung von Hexadezimalzahlen) die Segmente a-g definiert, die bei dieser Ziffer leuchten. Eine Binärzahl 11111100 bedeutet beispielsweise, dass alle Segmente außer dem Mittelbalken g leuchten (die letzte Stelle ist immer 0).
Danach werden wie üblich die Pinnummern festgelegt und die Port-Directions im setup-Block gesetzt.
Die Loop enthält einen Zähler, der alle Zahlen von 0-9999 anzeigt. Dazu wird das Unterprogramm SevenSeg-Display mehrfach aufgerufen, denn in einem Durchgang werden die vier Ziffern nur jeweils einmal angezeigt, danach bleibt die letzte Ziffer stehen.

SevenSegDisplay beschränkt zunächst die Zahl auf vierstellige Werte, berechnet dann die einzelnen Ziffern und stellt sie mit DisplayADigit dar. Diesem Unterprogramm wird nicht die Dezimalziffer übergeben, sondern bereits der zughörige Array-Wert mit den anzuzeigenden Segmenten.

DisplayADigit blockiert im 74595 zunächst die Datenübertragung vom Schieberegister ins Ausgangsregister. Anschließend werden alle Ziffern ausgeschaltet, indem die Kathoden abgeschaltet werden (AllDispOff, könnte auch nach dem Schieben erfolgen). Nun werden die Segmentbits durch das Unterprogramm shiftOut nacheinander in das Schieberegister des 74595 hineingeschoben. Danach kann die Datenübertragung ins Ausgangsregister wieder eingeschaltet werden. Die aktuelle Ziffern-Kathode wird ebenfalls eingeschaltet.

WK_Sevenseg_4_B

Hier gibt es ein paar kleine Modifikationen in SevenSegDisplay: Führende Nullen werden unterdrückt, indem alle Segmente der Ziffer ausgeschaltet werden. Ferner kann noch ein Dezimalpunkt gesetzt werden, indem das unterste Segmentbit der entsprechenden Ziffer auf 1 gesetzt wird (durch Addition von 1).

WK_Sevenseg_4_C

Entspricht der B-Version, nur sind die Details wieder in eine Bibliothek gewandert.

WK_SevenSeg_2x2

Diese Version ist ebenfalls ähnlich der B-Version, nur dass jetzt zwei zweistellige Ziffern (mit unterdrückten führenden Nullen) dargestellt werden, getrennt durch einen Dezimalpunkt in der Mitte. Diese Anzeige kann dazu dienen, bei einem Heizgerät Soll- und Ist-Temperatur (von 0-99 Grad) gleichzeitig darzustellen. Der eigentliche Code befindet sich wieder in der SevenSeg-Bibliothek.

5.12 Wasserkocher-Steuerung (WK_water_kettle)

Dieses Programm ist eine Kombination aus WK_button_timeout, WK_RGB_TEMP_SENSOR und (optional) WK_SevenSeg_2x2. Es dient zur Steuerung eines Wasserkochers.

Hardwareaufbau

Der Hardwareaufbau besteht aus der IO-Testschaltung und optional der 4x7-Segmentanzeige. Der Temperatursensor der IO-Testschaltung wird durch den Temperatursensor am Kabel ersetzt, der nun im Wasserkocher hängt (möglichst tief, aber ohne Heizplattenberührung).
Parallel zum LED-Ausgang 13 wird das Switchbox-Relais geschaltet, das den Wasserkocher schaltet. (Leider flackert LED13 auch während des Upload-Vorgangs, so dass sich das Relais zu Beginn unmotiviert ein- und ausschaltet). Der Schalter des Wasserkochers wird z.B. mit einem Streichholz blockiert, so dass er immer eingeschaltet ist.

Funktionsbeschreibung der Wasserkocher-Steuerung

Auf Tastendruck wird der Kocher über das Relais eingeschaltet, was auch an der LED angezeigt wird. Die RGB-LED pulsiert während der Heizphase und verändert mit der Temperatur ihre Farbe von einem kühlen Blau zu einem warmen Rot. Das 4x7-Segment-Display zeigt, wenn vorhanden, die Soll- und die Ist-Temperatur an. Nach Erreichen der Soll-Temperatur schaltet das Relais den Kocher aus. Ferner erfolgt nach einer festgelegten Timeout-Periode eine Sicherheitsabschaltung.

Programmbeschreibung

DEBUG-Modus: Wenn der DEBUG-Modus auskommentiert wird, werden am seriellen Monitor interne Statusmeldungen ausgegeben.
SEVENS-Modus: Diese Zeile wird auskommentiert, wenn eine Siebensegment-Anzeige vorhanden ist.

Nun werden die Bibliotheksdateien ins Programm eingebunden. Sie enthalten die Zuweisung der I/O-Pins, die Ansteuerung der RGB-LED, die Vorgabe der Soll-Temperatur und die Messung der Ist-Temperatur und die Ansteuerung der Siebensegmentanzeige.
Anschließend erfolgt die Deklaration von Konstanten und Variablen.
Danach werden alle Bibliotheken initialisiert und insbesondere die Pinnummern übergeben, um damit die Port-Richtung festzulegen.

Im setup-Teil wird ggf. der serielle Monitor gestartet und weitere Port-Richtungen festgelegt. Ein Array zur Mittelung der gemessenen Temperaturen wird initialisiert.

Der grobe Ablauf im loop-Teil entspricht im Wesentlichen dem Ablauf in WK_button_timeout, den man sich noch einmal vergegenwärtigen sollte.
Zunächst wird die Soll-Temperatur vom Poti abgelesen und auf maximal 99 Grad limitiert.
Außerdem wird die Ist-Temperatur vom Temperatursensor im Wasserkocher gelesen. Um Schwankungen und unplausible Werte durch Störungen auszumitteln, wird das Unterprogramm middle_Temp der Bibliothek ReadTemp aufgerufen.

Nun folgt das Einlesen des Taster-Zustands, Flankenerkennung, Debouncing und Timeout wie in WK_button_timeout. Zusätzlich zur Timeout-Bedingung gibt es noch eine Tempout-Bedingung zur Abschaltung des Wasserkochers, wenn die Ist-Temperatur über der Soll-Temperatur liegt.

Im Debug-Modus werden nun Temperatur- und Steuervariablen am Monitor ausgegeben.

Nun folgt der Aktionsteil, der Kocher und Anzeigen steuert: Beim Einschalten wird ein Runcounter auf 0 gesetzt, der dann während der Einschaltphase bei jedem Schleifendurchlauf inkrementiert wird.

Beim Abschalten werden RGB-LED und Siebensegmentanzeige ausgeschaltet.

Während der Einschaltphase werden Soll- und gemittelte Ist-Temperatur in regelmäßigen Runcounter-Abständen zwischengespeichert und an die Siebensegmentanzeige ausgegeben. Ferner wird die gemittelte Temperatur durch die RGB_LED dargestellt.

Am Ende der Loop wird der aktuelle running-Zustand (Einschaltphase) an die Zustands-LED und das Relais ausgegeben.

Aufgaben: Teste die Wasserkocher-Steuerung:

- Was passiert, wenn die Soll-Temperatur bereits beim Einschalten unterhalb der Ist-Temperatur liegt?
- Was passiert, wenn der Temperatur-Sensor defekt (also abgeklemmt) ist? W
- äre es sinnvoll, einen hochohmigen Widerstand vom Sensorausgang an 5V zu legen, um bei ausgefallenem Sensor eine definierte Spannung zu haben, oder sollte man den Widerstand besser an 0V legen?
- Lässt sich der Wasserkocher per Tastendruck ausschalten?
- Wie stark schwankt die Temperatur am Sensor?
- Verfolge die Temperatur am Monitor, nachdem der Kocher ausgeschaltet wurde. Steigt sie weiter?
- Warum? Was würdest Du an dieser Steuerung noch verbessern wollen?

5.13 Mathematische Programme (Math_Fakultaet, Math_SiebErathosthenes)

Um zu zeigen, dass der Arduino ein kompletter Rechner ist und z.B. in Taschenrechnern eingebaut werden könnte, sind ein paar kleine mathematische Programme angeführt, die man zwischendurch ins Praktikum einstreuen kann.

Math_Fakultaet berechnet die Fakultät, allerdings nur bis 12. Daran kann man auch zeigen, dass die Zahlenbereiche im Rechner durch die Anzahl der verwendeten Bytes beschränkt sind. Man könnte das Programm auch auf long integer umbauen.

Math_SiebErathosthenes berechnet Primzahlen mit dem Sieb des Erathosthenes.

6 Weitere Aufgaben

6.1 Schaltungsfehler

Erlaube Deinem Partner aus der Arbeitsgruppe, einen oder mehrere Fehler in die Schaltung einzubauen, die Du schnellstmöglich finden musst.

Nicht erlaubt sind: Kurzschlüsse eines Arduino-Ports zur Versorgungsspannung oder GND oder untereinander, Kurzschlüsse von 5V nach GND (das kann schlimmstenfalls den Host-Rechner zerstören), Entfernung des Vorwiderstands einer LED, Vertauschen der Anschlüsse eines Dreibeiners (z.B. Temperatursensor, Transistor), Manipulation des Arduino-Boards selbst; also alles, was zu Kurzschlussströmen und somit zur Zerstörung von Bauteilen führen kann.

Erlaubt sind: Aktoren (LEDs, Motoren) an andere Arduino-Ausgangs-Ports anschließen, Sensoren (LDR, Poti, Switch, etc.) an andere Arduino-Eingangs-Ports anschließen, Schaltungs-Unterbrechungen (z.B. Kabel in freie Breadboard-Zeile umstecken oder ganz entfernen), LED mit vertauschter Polung einsetzen.

Als Hilfsmittel zur Fehlersuche kann ein Vielfach-Messinstrument eingesetzt werden (Bereich DCV, eingestellter Spannungsbereich $\geqslant$ 5V). Man überprüft zunächst immer die Versorgungsspannung und sucht dann weiter, bis man keine Spannung mehr misst, wo eine anliegen sollte.

6.2 Programmierfehler

Erlaube Deinem Partner aus der Arbeitsgruppe, einen oder mehrere Fehler in das Programm einzubauen, die Du schnellstmöglich finden musst. Tipps dazu findet man unter „beliebte Programmierfehler“. Man kann auch ganze Anweisungen löschen oder modifizieren. Hier ist fast alles erlaubt, auch Mehrfachfehler. Nicht erlaubt sind Änderungen, die die Hardware schädigen: Die Port-Richtung ändern, ein Relais zu schnell hin- und herschalten (toggeln).

6.3 Elektrische Messungen und Berechnungen an LEDs

- Miss die Versorgungsspannung U_B gegen GND auf dem Arduino-Board.
- Miss die Spannung U_O an einem Arduino-Ausgang, der auf HIGH gesetzt ist und mit dauerleuchtender LED mit Vorwiderstand beschaltet wird. Sie ist etwas geringer als U_B.
- Miss die Schwellspannung U_S direkt an der LED für alle verfügbaren LED-Farben.
- Miss die Spannung über dem Widerstand U_R.
  Vergleiche das Ergebnis mit der Berechnung: $U_R = U_O - U_S$
- Berechne den Strom durch Widerstand und LED: $I = \frac{U_R}{R}$ $(R = 220\,\Omega)$.
- Berechne den minimalen Widerstand Rmin für den maximal erlaubten LED-Strom von $I_{max} = 20\,mA$ $\left(R_{min} = \frac{U_R}{I_{max}}\right)$.

- Was würde passieren, wenn man den Vorwiderstand weg lässt?
  (Die Differenz zwischen U_O und U_S fällt an dem Innenwiderstand des Arduino-Ausgangs ab. Dadurch kann der überlastet werden und bei einem höheren als dem zulässigen Strom von 40mA durchbrennen. Ebenso kann auch die LED bei einem höheren als dem zulässigen Strom von 20mA durchbrennen).
- Baue eine LED-Prüfschaltung auf: 5V-R(220Ω)-LED-GND.

7 Weiterführende Projekte

7.1 Wasserkocher: Fertig-Signal

Die Wasserkocher-Steuerung soll erweitert werden um einen Summer, der das Ende der Aufheizzeit signalisiert. Das könnte ein nerviger Summton sein, aber auch eine kleine Melodie.

7.2 Wasserkocher: Feuchtesensor

Da die vorgestellte Version des Wasserkochers keinen Sensor enthält, der anzeigt, ob Wasser im Kocher ist, kann der Kocher ohne Wasser überhitzen. Die Schaltung müsste daher um einen Sensor erweitert werden, der anzeigt, ob Wasser im Kocher vorhanden ist und den Temperatursensor bedeckt. Dazu könnte ein Feuchtesensor oder ein Drucksensor verwendet werden.

7.3 Wasserkocher: Temperatur-Regelung

Modifiziere das Programm zur Wasserkocher-Steuerung so, dass die Temperatur auch noch eine Minute nach Ende der Aufheizzeit angezeigt wird.
Man wird feststellen, dass sich die Temperatur über die eingestellte Soll-Temperatur hinaus erhöht! Das ist besonders beim Erwärmen von Babyflaschen unerwünscht.
Wie lässt sich das vermeiden?
Man könnte einerseits die Soll-Temperatur ganz langsam anfahren. Dadurch wird aber die Aufheizzeit unnötig verlängert.
Man könnte während des Aufheizens aber bereits messen, wie schnell die Temperaturerhöhung erfolgt und berechnen, wie stark die Heizung „überschwingt“. Aus einer simplen Steuerung wird dann eine komplizierte Regelung, die aber keine weitere teure Hardware benötigt, sondern nur mehr Überlegungen des Programmierers.
Im Rahmen eines Einführungspraktikums ist das zu kompliziert, aber vielleicht geeignet für eine Jahresarbeit.

7.4 Test von Leuchtstofflampen

Stiftung Warentest hat die Zyklenfestigkeit von Kompakt-Leuchtstofflampen (sog. Energiesparlampen, also den giftigen Quecksilberdampflampen) getestet. Ein Schaltzyklus dauerte 5 Minuten, davon ½ Minute ein, 4½ Minuten aus. Gemessen wurde die Zahl der Schaltzyklen. Eine Messvorrichtung könnte folgendermaßen aussehen: Die Messung wird durch Betätigung eines Tasters gestartet (und kann damit auch gestoppt werden). Der Arduino zählt die Zyklen und gibt die Zahl auf einem Siebensegment- oder LCD-Display oder auf dem

Hostrechner aus. Er steuert die Leuchtstofflampe über das Switchbox-Relais an. Ein Ausfall der Lampe wird über einen LDR detektiert. Billige Leuchten und Sofortzünder hielten diesen Testzyklus nur 30 Stunden durch, teure mehrere 1000 Stunden.

8 Zum Autor

Ich habe in Karlsruhe Elektrotechnik studiert und wurde in Informatik promoviert. Danach arbeitete ich viele Jahre als Chipentwickler in der Telekommunikationsbranche.

Nach meinem Besuch der Akademie für Waldorfpädagogik in Mannheim unterrichtete ich Physik und Informatik an der Freien Waldorfschule Rastatt und der Helmut-von-Kügelgen-Schule in Fellbach. An diesen Schulen wurde auch das Arduino-Praktikum entwickelt und erprobt. Die nötigen Bausätze sind dort vorhanden und können evtl. ausgeliehen werden. Derzeit stehe ich noch ab und an für Gastepochen in Physik und Informatik zur Verfügung und fertige physikalische Geräte für den Unterricht an. Ich habe ein eigenes, gut ausgestattetes Physiklabor mit Geräteverleih. Ferner kann ich beim Aufbau eines Arduino-Praktikums Hilfestellung leisten.

Das Skript entstand aufgrund einer Anregung von Robert Neumann, der darum bat, dieses Praktikum der Waldorf-Informatik-Gemeinde zur Verfügung zu stellen. Es hat ein paar Schuljahre gedauert, ist dadurch aber auch schon etwas gereift. Für seine freundliche Ermutigung sei ihm gedankt. Auch über die Mitteilung von Erfahrungen, Fehlermeldungen, Verbesserungsvorschlägen und Erweiterungen bin ich dankbar.

Kontakt zum Autor per Email an: waldit_mjn(at)gmx-topmail.de
Für Rückfragen am besten Name und Telefonnummer angeben.