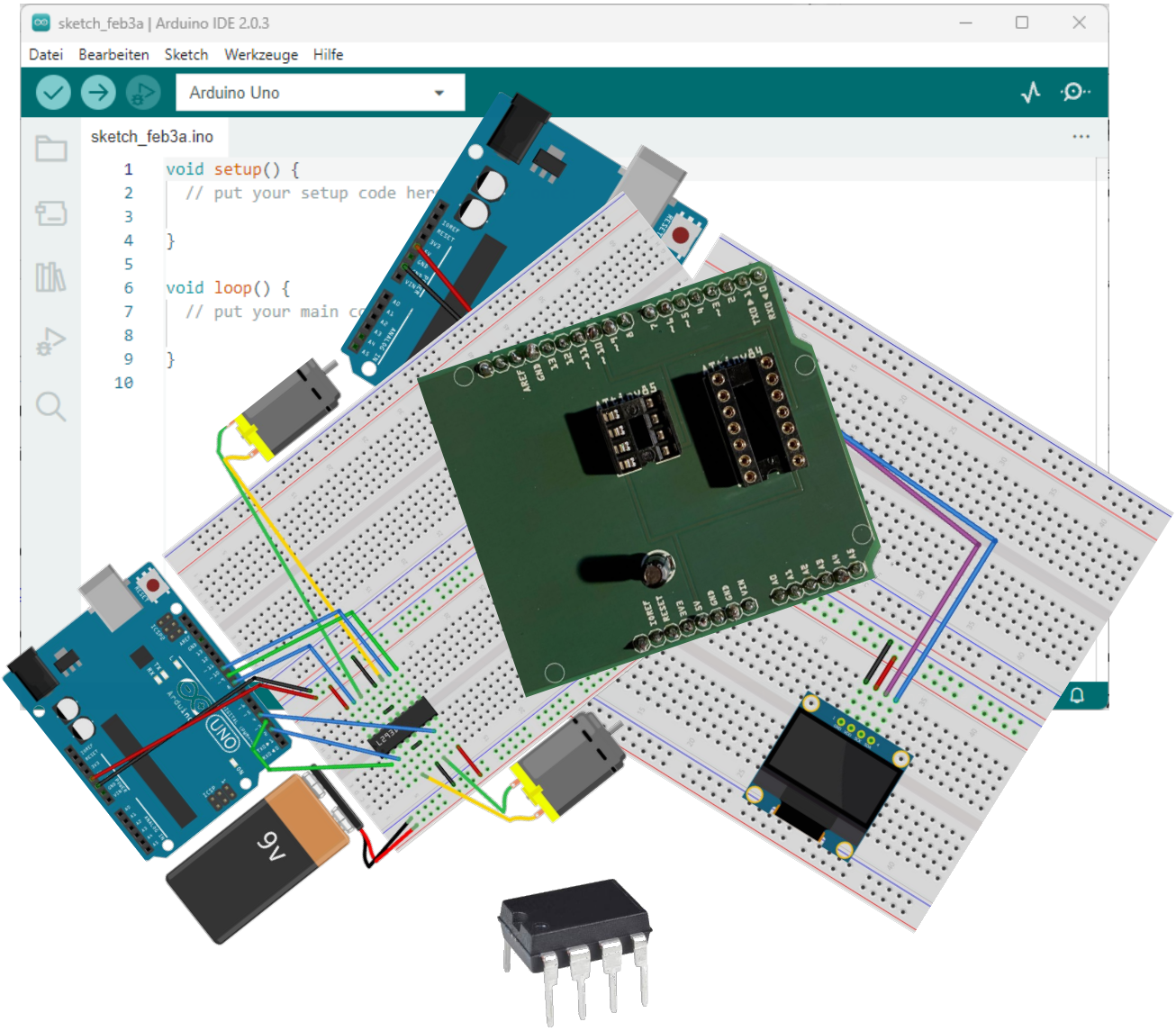
Werkstatt zu Mikrocontrollern



Verfasst von Beat Trachsler, KSK

Version 25.03.2023

Schulbereich, Stufe

Gymnasium

Vorkenntnisse

- Grundkenntnisse über Datenrepräsentation (z.B. ASCII)

Bearbeitungsdauer

6 - 10 Doppellektionen

Inhaltsverzeichnis

Einleitung

In diesem Skript geht es um Mikrocontroller. Diese sind im Gegensatz zu Mikroprozessoren so organisiert, dass sie Interrupt-Controller, Taktgeber usw. bereits enthalten. Man spricht in diesem Zusammenhang oft auch von System-on-a-Chip (SoC). Sie werden daher traditionellerweise in der Steuerungstechnik verwendet. Behandelt wird in dieser Werkstatt die Programmierung von Einplatinen-Mikrocontrollern wie dem Arduino Uno und von Mikrocontrollern der ATtiny Serie. Zur Programmierung wird die Arduino IDE verwendet. Als Grundlage dient die Programmiersprache C++, die im Rahmen dieser Werkstatt erlernt werden kann. Angewandt wird das neu erworbene Wissen auf Modelleisenbahnen (Signale, Weichen, etc.).

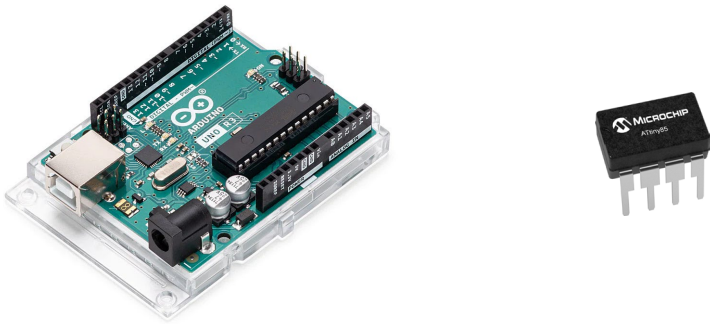


Abbildung 1: Arduino Uno und ATtiny85

Die Werkstatt enthält neben der aufbauenden Theorie die folgenden, immer wiederkehrenden Bausteine, welche farblich hervorgehoben sind:

Beispiele

Die Beispiele dienen zur Illustration der behandelten Theorie.

Tabellen

Tabellen dienen zur Erklärung von Datentypen und Operatoren.

Aufgaben

Die Aufgaben dienen zur schrittweisen Erarbeitung des Lernstoffs. Zu jeder Aufgabe existiert eine detaillierte Beispiellösung.

Anwendungsaufgaben

Die Anwendungsaufgaben sollen die Verbindung zur Praxis in der Informatik herstellen. Hier können Sie das Gelernte ausprobieren.

Posten I: Die Programmiersprache C++ und die Arduino IDE

C++ ist eine moderne Programmiersprache, welche es dem Programmierer erlaubt ein allgemeines Problem[1] so zu formulieren, dass ein Computer (= Rechner) das Problem bearbeiten kann. Die nachfolgende Abbildung zeigt die Geschichte einiger Programmiersprachen.

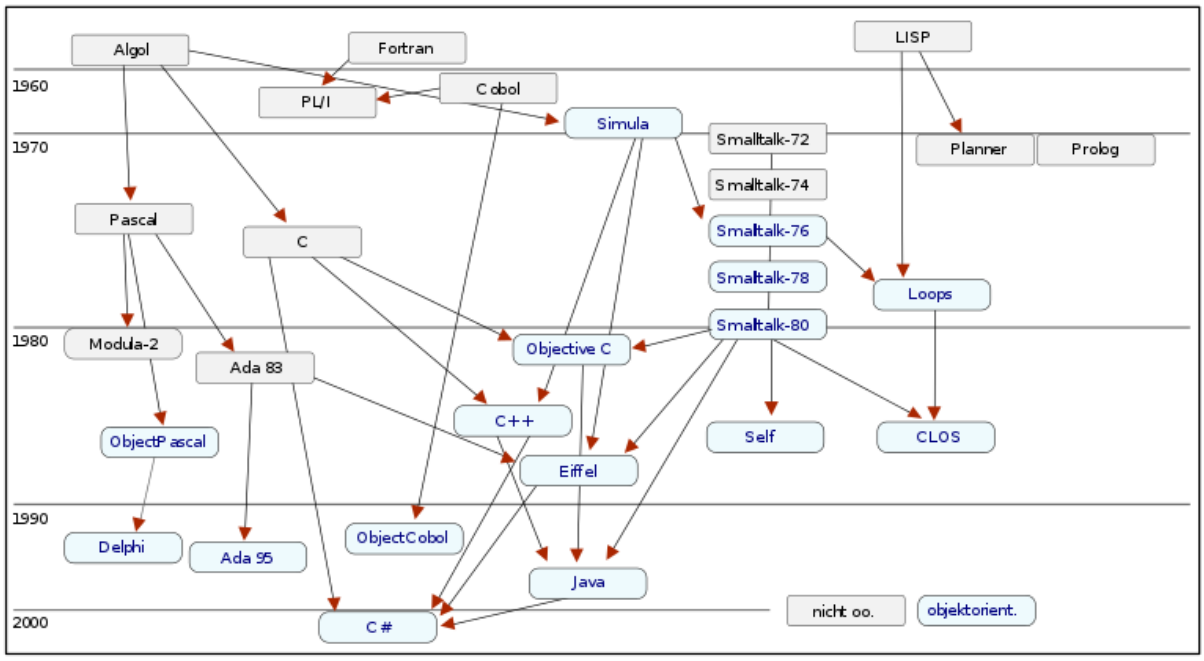


Abbildung 2: Geschichte der Programmiersprachen
Quelle: https://commons.wikimedia.org/wiki/File:History_of_object-oriented_programming_languages.svg

Die unter dem Stichwort **objektorientiert** zusammengefassten Programmiersprachen werden oft als **OOP**-Sprachen für **o**bjekt**o**rientierte **P**rogrammiersprachen bezeichnet. Dabei werden die Rechenanweisungen (= Befehle) mit den Daten (Variablen etc.) zu einem Objekt verbunden. Im Gegensatz dazu nennt man Sprachen wie Basic, Pascal oder C prozedural, weil dort die einzelnen Teilprogramme oder Prozeduren im Vordergrund stehen. C++ ist also genau wie Java und C# eine objektorientierte Programmiersprache. Im Gegensatz zu Java ist C++ aber nicht plattformunabhängig, was für unser Thema zum Glück keine Rolle spielt, da wir ausschliesslich Mikrocontroller programmieren. Ausserdem sind C++-Programme in der Regel schneller als Java Programme, da sie direkt für die jeweilige Hardware optimiert werden können.

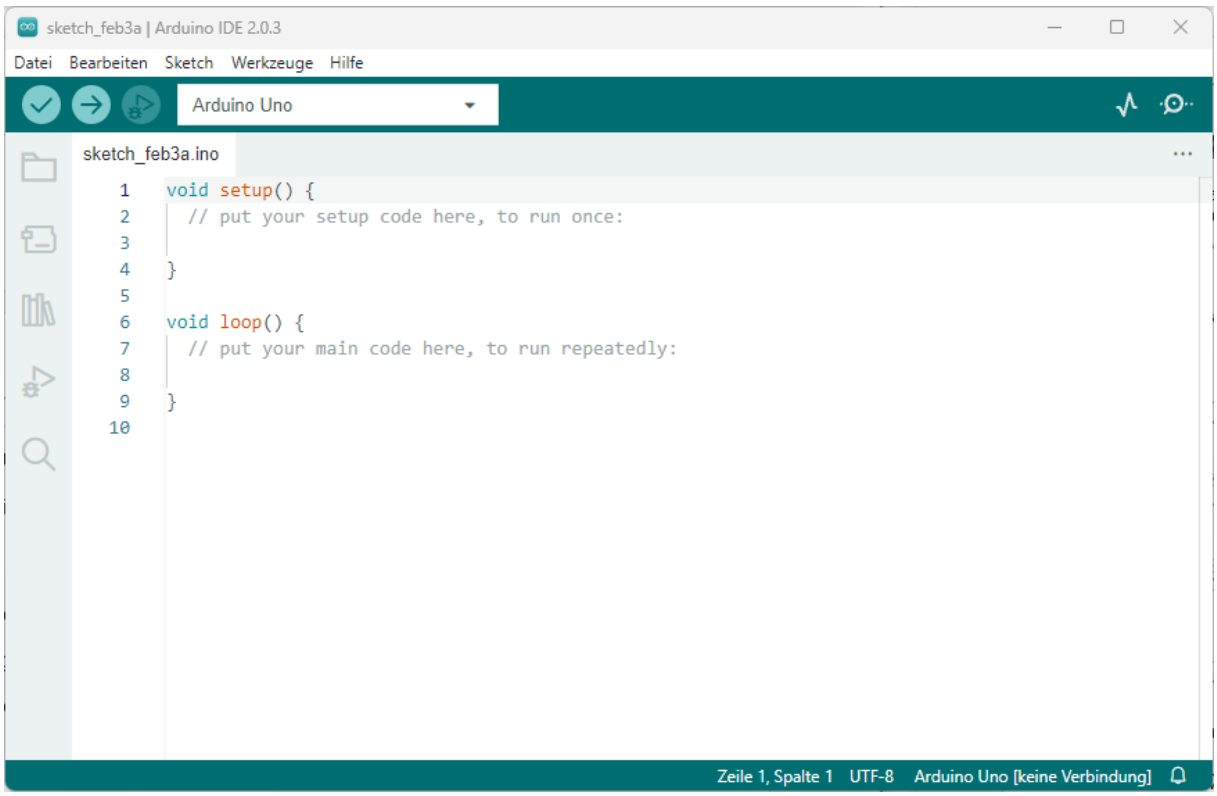
[1] Daneben gibt es für spezielle Aufgabenstellungen, wie z.B. das Erstellen einer Homepage für das Internet oder für das Abfragen von Datenbanken, eigene, speziell dafür entwickelte Sprachen wie HTML oder SQL.

Die Entwicklungsumgebung Arduino IDE

Die Arduino IDE[1] ist eine Entwicklungsumgebung zur Programmierung von Mikrocontrollern. Sie dient wegen ihrem einfachen, benutzerfreundlichen Aufbau oft als Einstiegsumgebung für Studierende, welche damit Mikrocontroller kennenlernen und C++ lernen wollen. Die Arduino IDE kann unter dem folgenden Link heruntergeladen werden:

https://www.arduino.cc/en/software

Nach dem Start der Software kann man direkt mit dem Coden beginnen. Es ist auch, wie die folgende Abbildung zeigt, bereits eine Art Programmgerüst dafür vorhanden. Ausserdem sind die Zeilen nummeriert.

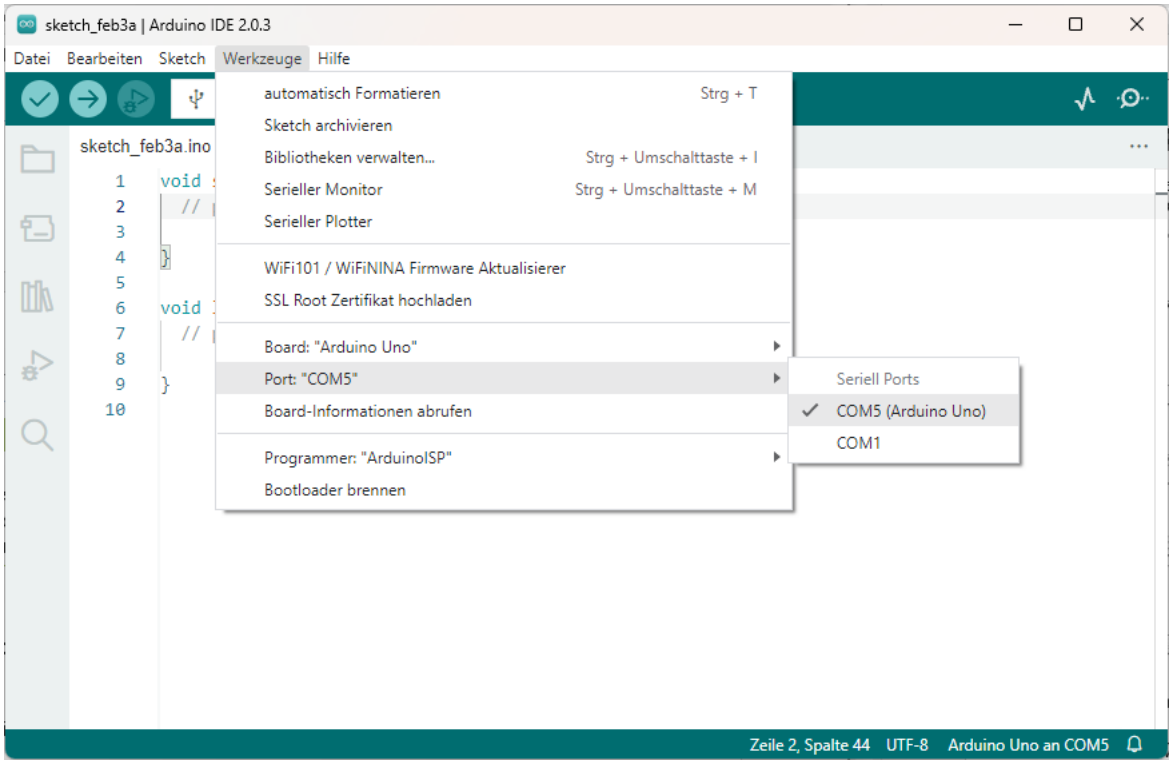


Sämtliche Programmierbefehle, welche auf Zeile 3 ergänzt werden, gehören zum setup[2]. Das bedeutet, dass sie nur einmal beim Start des Programms ausgeführt werden. Programmierbefehle, welche in die Lücke auf Zeile 8 eingefügt werden, gehören zum loop. Sie werden permanent wiederholt. Dies mag ungewohnt erscheinen, ist aber, wie wir sehen werden, für die Programmierung von Mikrocontrollern sehr praktisch.

[1] IDE steht für Integrated Development Environment, auf Deutsch integrierte Entwicklungsumgebung.
[2] Genauer gesagt, gehört alles zum setup, was sich zwischen der öffnenden geschweiften Klammer auf Zeile 1 nach «setup()» und der schliessenden geschweiften Klammer auf Zeile 4 befindet. Man nennt diese durch Klammern gebildete Einheit Block.

Beispiel 1

Der erste Schritt in einer neuen Programmiersprache ist traditionellerweise das «Hello World!». Gelingt die Ausgabe davon, ist der Anfang geschafft. Dafür müssen wir zuerst den Arduino Uno über ein USB-Kabel mit dem Computer verbinden, auf dem wir die IDE gestartet haben. Daraufhin können wir im Menü «Werkzeuge» den gewünschten Port auswählen. Er ist mit «(Arduino Uno)» gekennzeichnet (Vgl. Abbildung unten).



Schliesslich geben wir den folgenden Code ein, wobei wir die beiden mit «//» beginnenden Zeilen löschen können. Es handelt sich dabei lediglich um Kommentare.

```
void setup() {
  Serial.begin(9600);
  Serial.println("Hello World!");
}

void loop() {

}
```

Dabei ist zu beachten, dass in C++ jeder Programmierbefehl am Zeilenende mit «;» abgeschlossen werden muss. Mit der Schaltfläche «→» oder dem Menübefehl «Sketch > Hochladen» lässt sich das Programm auf den Arduino Uno hochladen und ausführen. Damit man die Rückmeldung an die IDE sehen kann, braucht man noch einen seriellen Monitor. Dieser lässt sich mit dem Menübefehl «Werkzeuge > Serieller Monitor» aufrufen. Daraufhin erscheint in der unteren Hälfte des Fensters der serielle Monitor, welcher unser «Hello World!» anzeigt.

Das obige Beispiel zeigt, wie man mit der Arduino IDE Programme schreiben und auf dem Arduino Uno ausführen kann. Aber was steckt hinter den Programmierbefehlen «Serial.begin» und «Serial.println»?

Mit «Serial» wird ein C++-Objekt aufgerufen, das die USB-Schnittstelle vom Arduino Uno zur Arduino IDE repräsentiert. Damit lassen sich Zeichenketten oder später auch Speicherinhalte vom Arduino Uno aus auf dem seriellen Monitor der IDE ausgeben. Damit dies klappt, muss die Verbindung zuerst mit «Serial.begin» und der gewünschten Übertragungsrate (in unserem Beispiel 9600 Bit pro Sekunde[1]) hergestellt werden. Der Befehl «println» gibt in der Folge die gewünschte Zeichenkette aus und wechselt auf eine neue Zeile («ln» für line). Die verwendete Schreibweise «Objekt.Methode» ist typisch für objektorientierte Programmiersprachen.

Aufgabe 1

Verändern Sie das obige Programm so, dass die Ausgabe von «Hello World!» beliebig oft wiederholt wird.

Zusatz: Verwenden Sie den Befehl «delay(<Anzahl Millisekunden>);» um die Ausgabe von «Hello World!» alle 5 Sekunden zu wiederholen.

Zusammenfassung

1. Arduino Programme enthalten immer ein **setup** und ein **loop**. Setup enthält dabei diejenigen Befehle, welche nur beim Programmstart ausgeführt werden, während die Befehle in loop permanent wiederholt werden.
2. Für die Ausgabe auf dem seriellen Monitor der Arduino IDE wird das **Serial**-Objekt verwendet, das die USB-Schnittstelle des Arduino Uno verkörpert. Mit Serial.begin(9600); wird eine Verbindung mit Übertragungsrate 9600 Bit pro Sekunde hergestellt. Die Ausgabe von Daten erfolgt mit dem Befehl Serial.println(…).
3. Mit dem Befehl **delay**(…) kann die Ausführung des Programms auf dem Arduino Uno um eine bestimmte Anzahl Millisekunden verzögert werden.

[1] Die Übertragungsrate entspricht hier gerade der Symbolrate, da die Symbolgrösse bei der seriellen Übertragung gerade ein Bit beträgt. Die Symbolrate wird in Baud gemessen. Die Symbolrate ist demnach 9600 Baud. Die Übertragungsrate kann bis 2 Mbit/s erhöht werden.

Posten II: Digitale Pins und das Breadboard

Der Arduino Uno ist ein Einplatinen-Mikrocontroller. Das bedeutet, dass sämtliche Bauteile auf einer Platine verbaut wurden. Dies ist zum einen die bereits verwendete USB-Schnittstelle. Daneben kann der Arduino Uno auch mit einem Netzteil verbunden werden, das den Arduino Uno mit einer Spannung von 7-12 V versorgt. Die Betriebsspannung des Mikrocontrollers beträgt jedoch standardmässig 5 V. Der Mikrocontroller ist derzeit ein ATmega328P der Firma Microchip mit einer Taktfrequenz von 16 MHz und einem Flash Speicher von 32 KB. Dieser verfügt über insgesamt 32 Beinchen oder Pins.

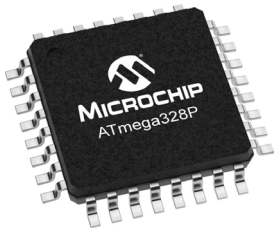


Abbildung 3: Mikrocontroller ATmega328P

Auf der Platine des Arduino Uno sind diesen Pins Steckplätze zugeordnet, auf denen man Steckbrücken einstecken kann. Die folgende Abbildung zeigt die Zuordnung dieser Steckplätze zu den sogenannten Arduino Pins, welche für die Programmierung wichtig sind.

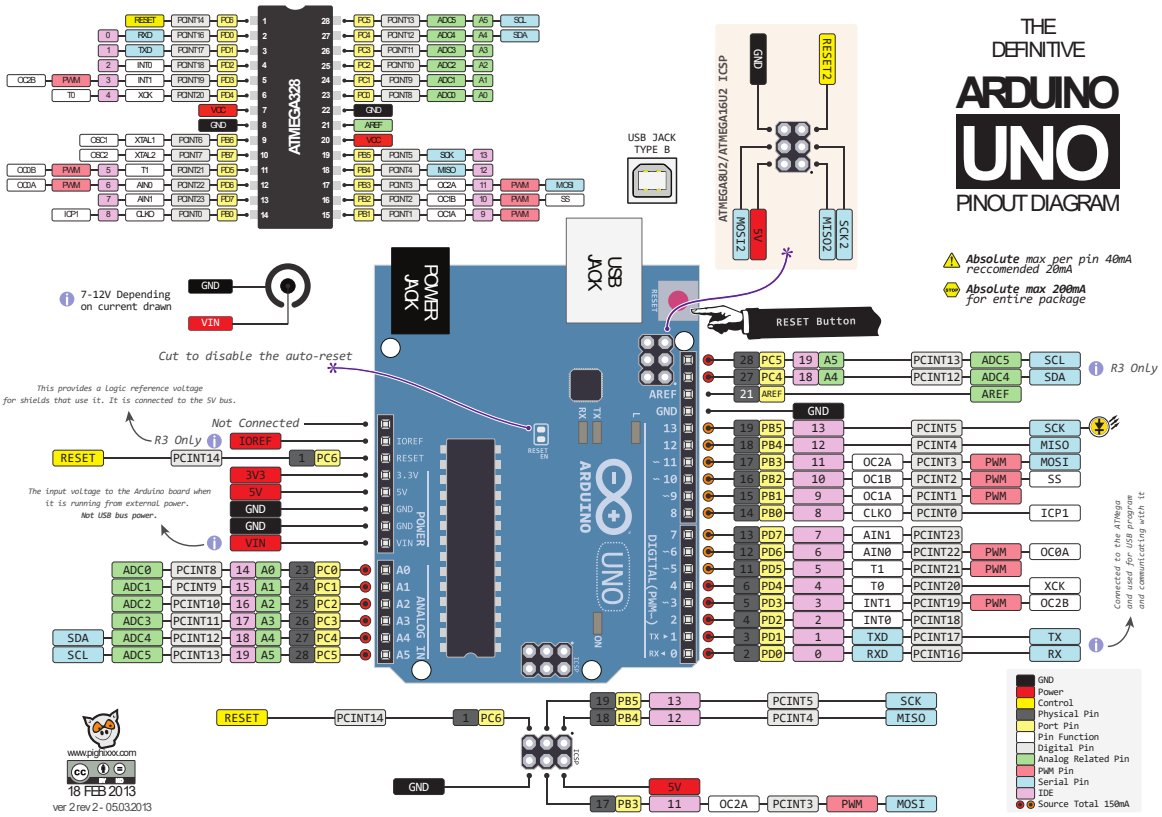
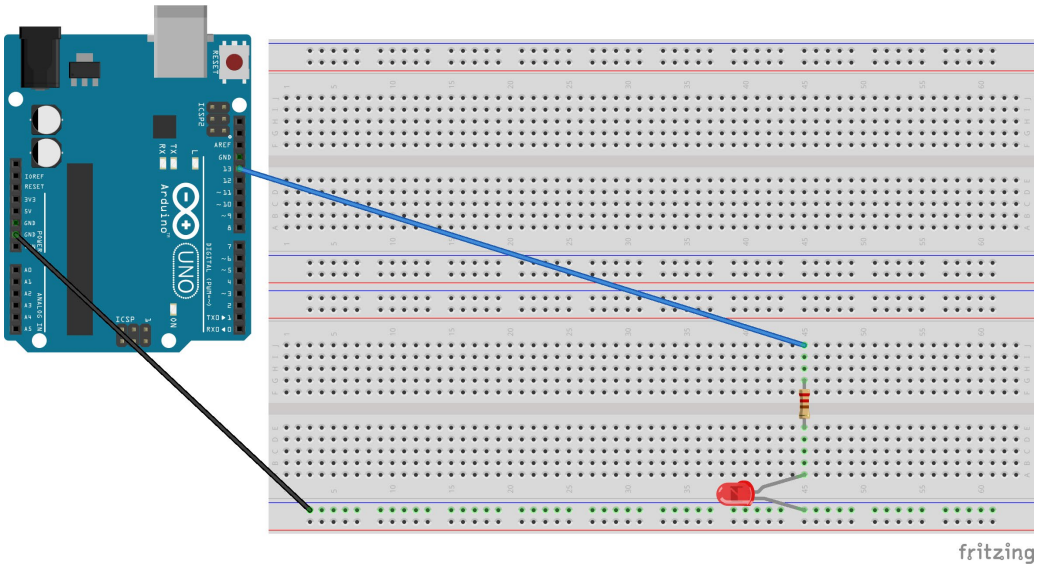


Abbildung 4: Beschriftung der Steckplätze beim Arduino Uno
Quelle: https://commons.wikimedia.org/wiki/File:Pinout_of_ARDUINO_Board_and_ATMega328PU.svg

Im aktuellen Posten geht es um die digitalen Pins (0-13), von denen vorerst nur die Pins 2-13 zur Verfügung stehen. Dies liegt daran, dass die Pins 0 und 1 als Empfänger (RX) und Sender (TX) für die serielle Schnittstelle reserviert sind. Dort können beim Gebrauch des Serial-Objektes in der Arduino-IDE dieselben Daten empfangen werden wie über den USB-Anschluss. Würden diese Pins umprogrammiert, müsste man auf die serielle Schnittstelle zum Arduino Uno verzichten, was zumindest am Anfang sehr nachteilig wäre.

Beispiel 2

In diesem Beispiel geht es darum, ein Programm zu schreiben, das eine LED, welche wir auf dem Breadboard[1] verdrahten, zum Leuchten bringt. Dafür benötigt man zunächst ein bisschen Physik. Würde die LED direkt mit den 5 V unseres Arduino Boards angesteuert, würde sie sofort verglühen. Daher wird ein Vorwiderstand[2] benötigt. Dieser beträgt bei einer LED und der Betriebsspannung von 5 V gerade 220 Ω (Farbcode Rot-Rot-Braun). Zunächst geht es somit darum, die nachfolgend abgebildete Schaltung auf dem Breadboard nachzubauen. GND bedeutet dabei Ground, übersetzt Boden. Das entspricht in der Regel dem Minuspol. Dort kommt immer das kurze Beinchen der LED hin.



Der folgende Programmcode lässt die LED permanent leuchten. Zu beachten ist dabei, dass auch die boardinterne LED am Steckplatz 13 leuchtet. Das ist eine Besonderheit dieses Steckplatzes.

```
void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
}

void loop() {

}
```

[1] Die internen Verbindungen des Breadboards werden im Anhang B1 erklärt.
[2] Hintergrundinfos zum Thema Widerstände sind im Anhang B2 zu finden.

Der Befehl «`pinMode`» im obigen Programm sorgt dafür, dass der digitale Pin 13 als OUTPUT- oder Ausgabe-Pin deklariert wird. Mit dem Befehl «`digitalWrite`» kann man beim Wert «HIGH» eine Spannung von 5 V anlegen oder diese mit dem Wert «LOW» zurück auf 0 V setzen. Andere Werte ausser «HIGH» und «LOW» ergeben bei einem digitalen Pin keinen Sinn.

Aufgabe 2

Verändern Sie das obige Programm so, dass die LED jeweils 3 Sekunden leuchtet und anschliessend 2 Sekunden ausgeschaltet bleibt.

Zusatz: Erstellen Sie einen Warnblinker mit einer für Sie passenden Frequenz.

Dimmen von LEDs mittels PWM

Mit Pulsweitenmodulation (PWM) bezeichnet man in der Elektronik ein Rechtecksignal mit konstanter Periodendauer, wie es in der nachfolgenden Abbildung dargestellt ist.

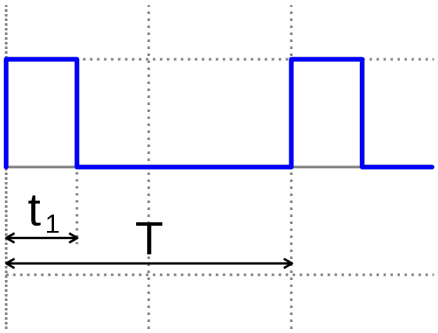


Abbildung 5 Pulsweitenmodulation mit konstanter Periode T
Quelle: https://commons.wikimedia.org/wiki/File:Pulse_wide_wave.svg

Wichtig ist dabei der prozentuale Anteil der Hochphase t_1 an der gesamten Periodendauer T. In der obigen Abbildung beträgt dieser Anteil 25%. Das Signal liefert somit im Mittel 1.25 V. Beim Arduino Uno sind diejenigen digitalen Pins, welche PWM unterstützen, mit «~» gekennzeichnet. Das folgende Beispiel zeigt, wie man mit PWM eine LED dimmen kann.

Beispiel 3

Damit PWM eingesetzt werden kann, muss der Pin für die LED von 13 auf 11 gewechselt werden. Damit dies nicht an diversen Stellen im Programmcode gemacht werden muss, verwendet man beim Programmieren Variablen. Der nachfolgende Programmcode lässt die LED am digitalen Pin 11 mit 25% ihrer Leuchtstärke leuchten.

```
int LEDPin = 11;

void setup() {
  pinMode(LEDPin, OUTPUT);
}
```

```
void loop() {
  digitalWrite(LEDPin, HIGH);
  delay(1);
  digitalWrite(LEDPin, LOW);
  delay(3);
}
```

Auf der ersten Zeile im obigen Programmcode wird noch vor dem `setup` die Variable LEDPin mit dem Wert 11 initialisiert. Vorsicht! Das ist keine Gleichung wie in der Mathematik, sondern eine Zuweisung. Da die Variable zuoberst steht, ist sie global und kann sowohl im setup wie auch im `loop` verwendet werden. Der Datentyp «`int`» bedeutet, dass es sich bei dieser Variablen um einen ganzzahligen Wert handelt, was vor allem für die Speicherbelegung im Mikrocontroller von Bedeutung ist (Vgl. Tabelle 1). Im `loop` ist das Verhältnis der HIGH-Phase zur LOW-Phase aufgrund der Delay-Werte gerade 1 : 3, was der gewünschten Leuchtstärke von 25% gleichkommt.

Dies lässt sich übrigens noch einfacher mit dem Befehl «analogWrite» erreichen, wie die folgenden Zeilen zeigen:

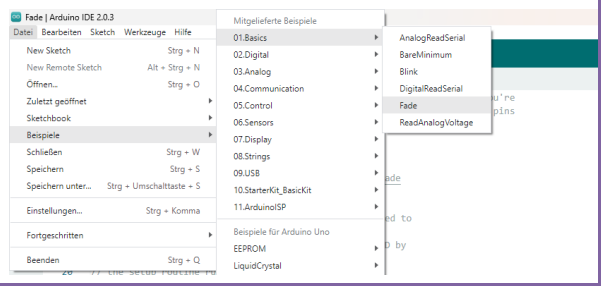
```
void loop() {
  analogWrite(LEDPin, 63);
}
```

Der Befehl «`analogWrite`» kann nur an den PWM-Ports verwendet werden. Der zweite Wert (hier 63) bezeichnet den Anteil t_1 an der gesamten Periodendauer T = 255.

Aufgabe 3

Verändern Sie das obige Programm so, dass die LED im Mittel mit 4 V versorgt wird.

Zusatz: Testen Sie das Beispielprogramm «Fade» aus Ihrer Arduino IDE. Versuchen Sie den Code zu verstehen.



Datentyp	Beschreibung
`int`	ganze Zahl von -32’768 bis 32’767[1]
`long`	ganze Zahl von -2'147'483’648 bis 2'147'483’647
`bool`	Wahrheitswert false oder true
`char`	128 Schrift- und Steuerzeichen gemäss ASCII-Zeichensatz
`byte`	nichtnegative ganze Zahl von 0 bis 255
`float`	Dezimalzahlen mit 7 Stellen bis zu Grössen von ca. 10^{38}

Tabelle 1: Basisdatentypen auf dem Arduino Uno

[1] Dieser 16 Bit Datentyp wird in C++ standardmässig mit «`short`» bezeichnet. Da der Arduino Uno kein 32 Bit «`int`» anbietet, wird hier wie üblich «`int`» statt «`short`» verwendet.

Posten III: Programmierung des ATtiny85

Da der ATtiny85 ein Mikrocontroller ohne Platine ist, gestaltet sich die Programmierung etwas komplizierter. Es braucht dafür einen so genannten Programmer. Dafür kann der Arduino Uno verwendet werden. Damit das klappt, müssen wir den Arduino Uno auf diese Aufgabe vorbereiten. Dazu öffnen wir mit dem Menübefehl «Datei > Beispiele > 11.ArduinoISP > ArduinoISP» das Programm für die In-System-Programmierung (ISP). Dieses laden wir direkt auf den Arduino Uno, so dass es dort ausgeführt wird. Nun kann die eigentliche ISP des ATtiny85 beginnen.

Für die ISP mit dem Arduino Uno brauchen wir zunächst eine passende Verdrahtung. Alternativ kann man das dafür erstellte Arduino Shield verwenden (Vgl. Anhang B3).

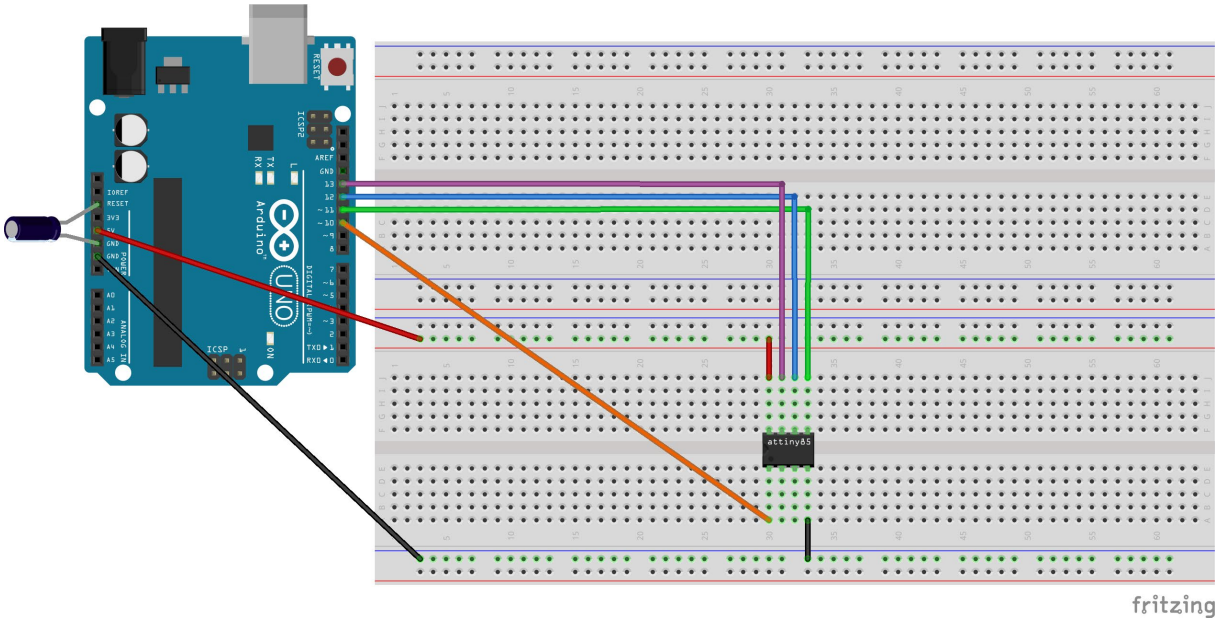


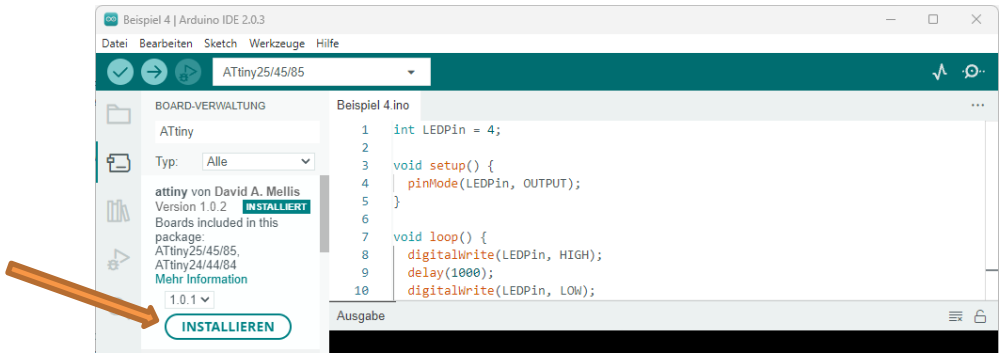
Abbildung 6 ISP des ATtiny85 mit dem Arduino Uno

Der Kondensator zwischen dem Reset-Pin und dem GND-Pin des Arduino Uno hat eine Kapazität von 10 µF und dient dazu, die Reset-Funktion des Arduino Uno zu übersteuern, damit er als Programmer verwendet werden kann. Bevor wir mit der ISP beginnen, sollten wir nochmals kontrollieren, ob der ATtiny85 richtig eingesetzt ist. Die halbkreisförmige Kerbe auf der Oberseite des ATtiny85 sollte dabei wie in der obigen Abbildung nach Links zeigen. Das orange Kabel sollte direkt zum ATtiny-Pin mit dem • führen.

Wenn diese Schaltung steht, kann man die Arduino IDE auf den ATtiny85 vorbereiten. Als erstes tragen wir mit dem Menübefehl «Datei > Einstellungen…» unter «Zusätzliche Boardverwalter-URLs» die folgende Adresse ein:

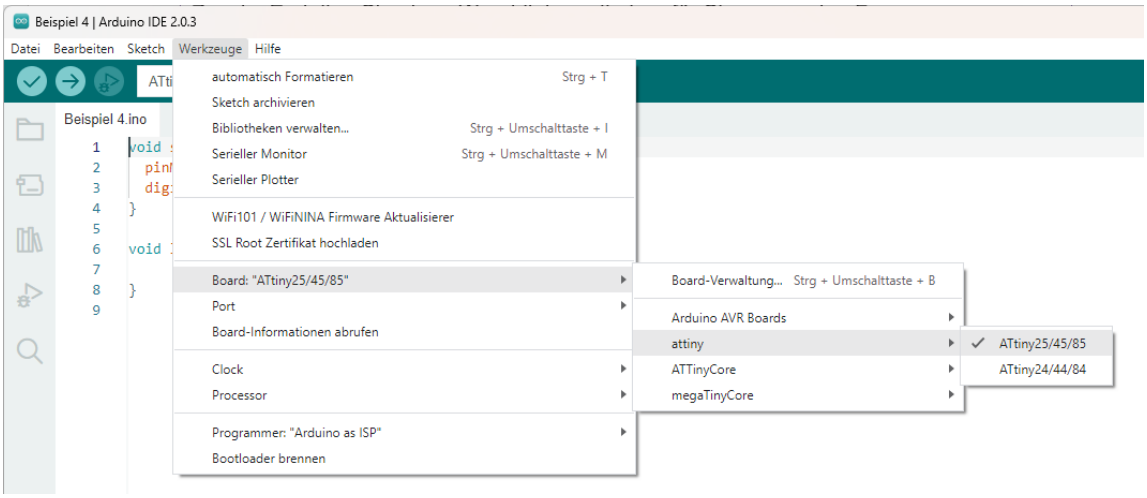
https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json

Dann lässt sich der Boardverwalter über die Seitenleiste installieren (Vgl. Abbildung unten).



Dieser Schritt ist nur einmal nötig, da unsere IDE diese URL in den Einstellungen abspeichert. Die folgenden Schritte müssen aber bei jeder Programmierung eines neuen Mikrocontrollers angepasst werden:

1. Mit dem Menübefehl «Werkzeuge > Board > attiny > ATtiny25/45/85» die korrekte Mikrocontroller-Familie auswählen (Vgl. Abbildung unten).
2. Mit dem Menübefehl «Werkzeuge > Port» den korrekten Port auswählen. Damit dies klappt, muss der Arduino Uno per USB-Kabel verbunden sein.
3. Mit dem Menübefehl «Werkzeuge > Clock > Internal 8 MHz» die gewünschte Taktfrequenz auswählen.
4. Mit dem Menübefehl «Werkzeuge > Processor > ATtiny85» den gewünschten Mikrocontroller auswählen.
5. Unter «Werkzeuge > Programmer» die Option «Arduino as ISP» auswählen (Vgl. Abbildung unten).

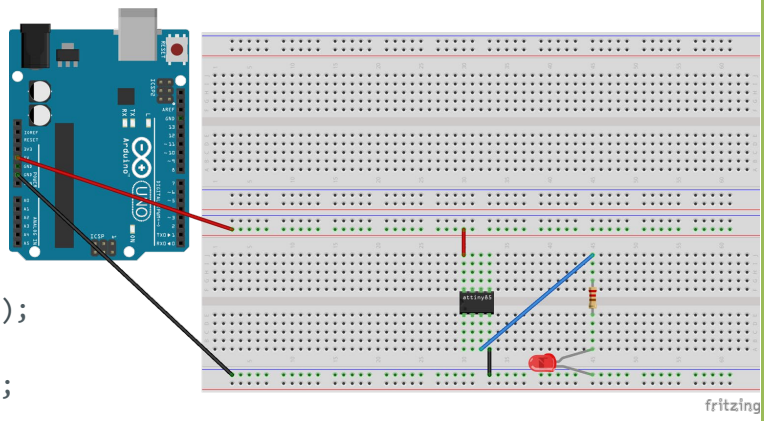


Wenn die Einstellungen gemacht wurden, können wir als erstes den Bootloader brennen. Dies ist nötig, weil der ATtiny85 standardmässig auf eine Taktfrequenz von 1 MHz eingestellt ist, wir aber eine Taktfrequenz von 8 MHz benötigen (Vgl. oben). Dieser Schritt ist nur einmal vor der ersten Verwendung notwendig. Dazu rufen wir den Menübefehl «Werkzeuge > Bootloader brennen» auf.

Beispiel 4

Das folgende Programm lässt die LED am Arduino Pin 4 des ATtiny85 leuchten. Dafür benötigen wir die Schaltung aus der untenstehenden Abbildung. Dabei ist zu beachten, dass der ATtiny85 über den Arduino Uno mit einer Betriebsspannung von 5 V versorgt werden muss.

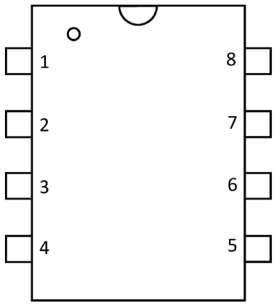
```
int LEDPin = 4;

void setup() {
  pinMode(LEDPin, OUTPUT);
}

void loop() {
  digitalWrite(LEDPin, HIGH);
  delay(1000);
  digitalWrite(LEDPin, LOW);
  delay(1000);
}
```



Wenn die Verkabelung steht und das Programm eingegeben wurde, kann es mit dem Menübefehl «Sketch > Mit Programmer hochladen» auf dem ATtiny85 ausgeführt werden.

Die nachfolgende Abbildung zeigt die Nummerierung der so genannten Arduino-Pins beim ATtiny85. Leider stimmt sie offensichtlich nicht mit der Nummerierung der ATtiny-Pins überein.

Funktion	Arduino
Reset	
	3
	4
GND	



Arduino	Funktion
	VCC
2	SCK
1	MISO
0	MOSI

Abbildung 7 Nummerierung der Pins beim ATtiny85

Aufgabe 4

Erstellen Sie mit dem ATtiny85 eine «Ampel», welche abwechselnd eine rote, eine gelbe und eine grüne LED ein- und wieder ausschaltet. Achten Sie dabei auf die Vorwiderstände.

Anwendungsaufgabe

Organisieren Sie ein Lichtsignal für eine Modelleisenbahn. Schalten Sie dieses mit dem ATtiny85. Informieren Sie sich dazu vorgängig über die Verdrahtung des Lichtsignals. Vorsicht! Auch die LEDs des Lichtsignals benötigen Vorwiderstände!

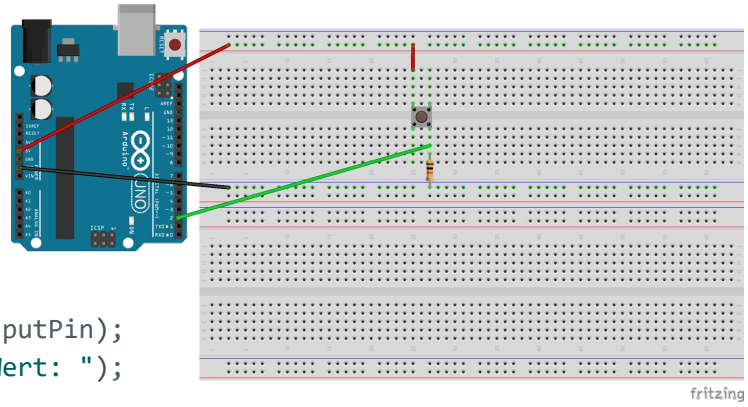
Posten IV: Digitale Werte erfassen und Bedingungen mit if

Selbstverständlich können die digitalen Steckplätze des Arduino Uno auch verwendet werden, um digitale Signale zu erfassen. Dafür wird der Befehl «digitalRead» verwendet, der entweder «HIGH» oder «LOW» zurückgibt. Wie man ihn einsetzt, zeigt das folgende Beispiel.

Beispiel 5

Wir bauen zunächst die folgende Schaltung nach mit einem Taster, den wir hinterher wahlweise gedrückt halten und loslassen können. Dabei möchten wir beobachten, welche Werte der Arduino am digitalen Pin 2 misst. Dazu dient das untenstehende Programm. Die Ausgabe erfolgt wieder auf dem seriellen Monitor. Beim Widerstand handelt es sich um einen 10 kΩ Widerstand mit dem Farbcode Braun-Schwarz-Orange.

```
int inputPin = 2;

void setup() {
  pinMode(inputPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  int value = digitalRead(inputPin);
  Serial.print("Gemessener Wert: ");
  Serial.println(value);
}
```



Der Befehl «`pinMode`» hat im obigen Code als zweiten Parameter den Wert INPUT. Damit wird der digitale Pin 2 auf Eingang geschaltet. Somit kann man weiter unten im `loop` mit dem Befehl «`digitalRead`» gefolgt vom passenden Pin als Parameter einen Wert einlesen. Da es sich um einen digitalen Pin handelt, ist der eingelesene Wert entweder 1 (HIGH) oder 0 (LOW). Dieser Wert wird der Variablen «value» vom Datentyp `int` zugewiesen. Der Befehl «`Serial.print`» gibt den Text in der Klammer auf dem seriellen Monitor aus, schaltet jedoch die Zeile noch nicht weiter.

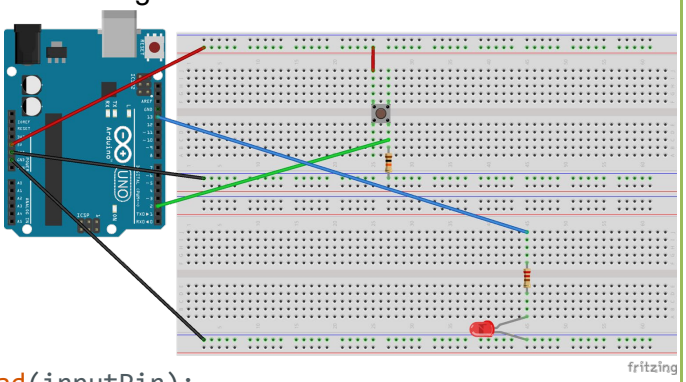
Aber wozu dient eigentlich der 10 kΩ Widerstand in der obigen Schaltung? Diese Frage lässt uns tief in die Elektronik eintauchen. Es handelt sich nämlich um einen so genannten Pull-down-Widerstand. Er wird gebraucht, damit der obige Schaltkreis definiert ist. Würde man ihn weglassen, würde der Arduino Uno am digitalen Pin 2 keine vernünftigen Messdaten erhalten. Der Wert würde permanent zwischen 0 und 1 wechseln.

Bedingungen mit if

Bedingungen erlauben uns, den Code zu gliedern. Abhängig von der Bedingung kann entweder die eine oder eine andere Befehlsfolge ausgeführt werden. Wie das funktioniert, zeigt das folgende Beispiel.

Beispiel 6

Wir erweitern die obige Schaltung, indem wir eine LED mit dem digitalen Pin 13 verbinden (Vgl. Abbildung). Anbei ausserdem der neue Programmcode.

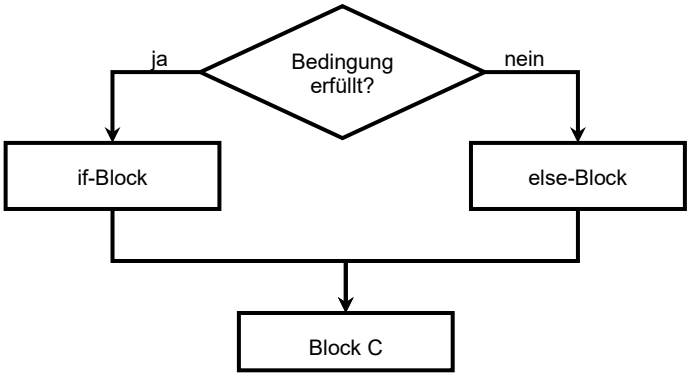



```
int inputPin = 2;
int LEDPin = 13;

void setup() {
  pinMode(inputPin, INPUT);
  pinMode(LEDPin, OUTPUT);
}

void loop() {
  bool buttonPressed = digitalRead(inputPin);
  if(buttonPressed) {
    digitalWrite(LEDPin, HIGH);
  }
  else {
    digitalWrite(LEDPin, LOW);
  }
}
```

Zu Beginn des `loop` wird der mit «`digitalRead`» eingelesene Wert unter der Variable «buttonPressed» abgespeichert. Diese Variable hat neu den Datentyp `bool`, da sie ja nur entweder true (HIGH) oder false (LOW) sein kann. Hier zeigt sich eine Besonderheit von C und C++. Die Grenzen zwischen den Basisdatentypen sind nicht so strikt wie beispielsweise in Java. Man könnte diesen Wert problemlos wie beim letzten Beispiel in einer Variablen des Datentyps `int` speichern. Das würde aber viel mehr Speicher belegen.

Auf der nächsten Zeile folgt die Bedingung mit `if`. Wenn die Variable «buttonPressed» den Wert true enthält, wird der Befehl auf der nächsten Zeile ausgeführt, der «LEDPin» wird somit auf HIGH gesetzt. Anderenfalls wird der Befehl nach dem `else` ausgeführt, der den «LEDPin» auf LOW setzt. Dies lässt sich wie folgt verallgemeinern.

```
if ( Bedingung )
{
      if-Block
}
else
{
      else-Block
}
Block C
```



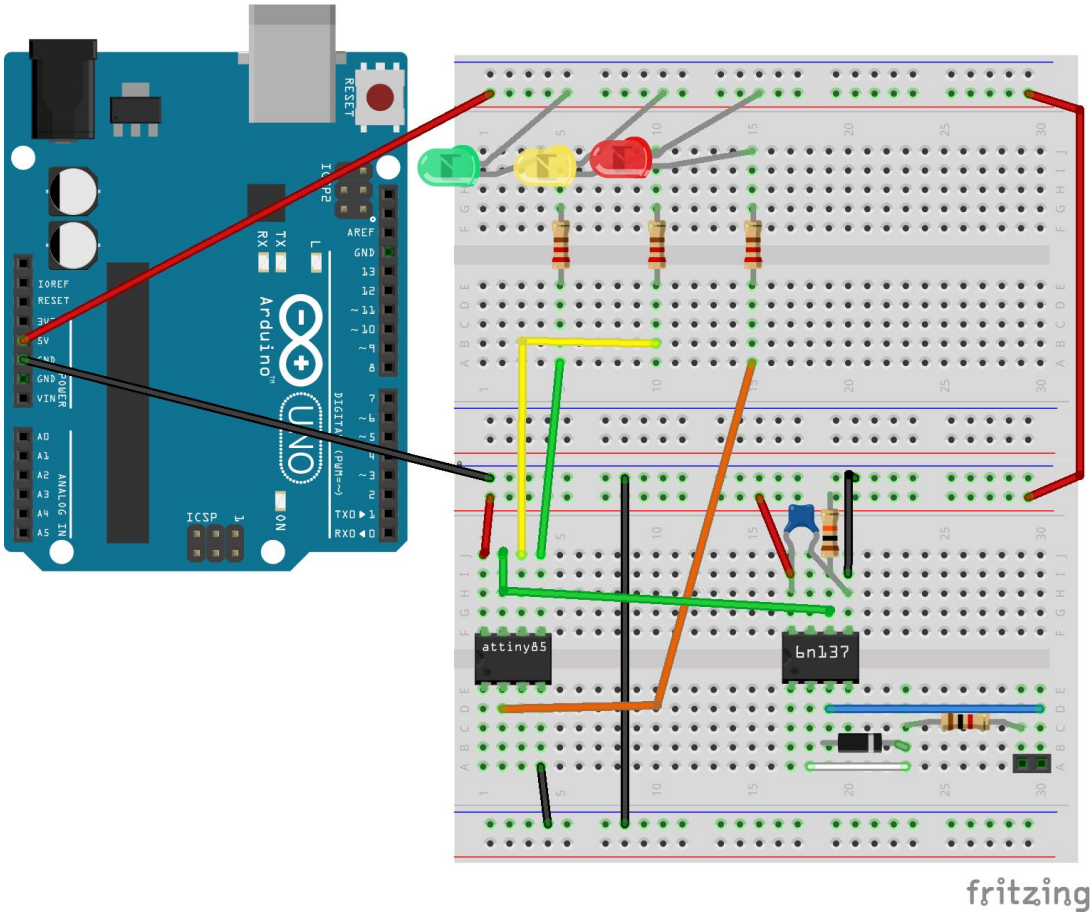
Aufgabe 5

Verändern Sie das Programm aus Beispiel 6 so, dass die LED leuchtet, wenn der Taster nicht gedrückt wird.

Anwendungsaufgabe

Erweitern Sie die Schaltung für Ihr Lichtsignal aus der letzten Anwendungsaufgabe so, dass Sie das Signal mit einem Taster umschalten können.

Zusatz: Wenn Sie über eine Z21 verfügen, können Sie das Lichtsignal über DCC steuern. Sie benötigen dazu einen Optokoppler 6N137, mit dem Sie das DCC-Signal dekodieren können. Die Schaltung dazu zeigt Ihnen die folgende Abbildung. Für die Dekodierung des DCC-Signals erhalten Sie von Ihrer Lehrperson eine passende Library.



Beachten Sie, dass die LEDs diesmal direkt mit 5 V verbunden sind. Die Widerstände liegen somit zwischen der LED und GND und die LEDs leuchten, wenn wir die entsprechenden Pins auf LOW setzen. Dies ist bei Modellbahnsignalen oft so. Diese Schaltung heisst in der Fachsprache «common anode».

Posten V: H-Brückenschaltung und Funktionen

H-Brückenschaltung

Eine H-Brückenschaltung ist eine H-förmige Anordnung von elektronischen Bauteilen, welche in der Elektronik an verschiedenen Stellen gewinnbringend eingesetzt wird. In diesem Posten geht es um H-Brückenschaltungen zur Motorsteuerung. Schematisch lassen sich diese wie folgt darstellen:

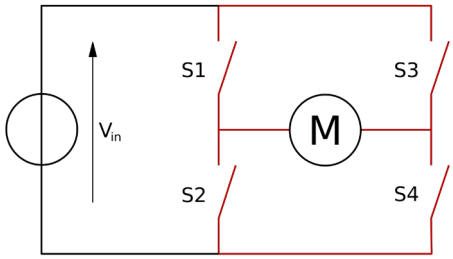


Abbildung 8 H-Brückenschaltung für einen Motor
Quelle: https://commons.wikimedia.org/wiki/File:H_bridge.svg

Mit einer H-Brückenschaltung lässt sich der Motor in zwei Richtungen (vorwärts und rückwärts) betreiben. Dies ist in der folgenden Abbildung schematisch dargestellt:

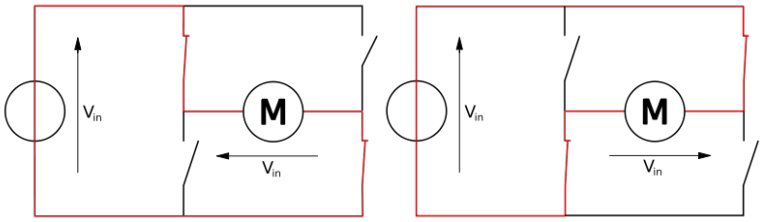


Abbildung 9 Zwei Schaltzustände der H-Brückenschaltung
Quelle: https://commons.wikimedia.org/wiki/File:H_bridge_operating.svg

Die öffnenden und schliessenden Schalter S1-S4 werden in der Elektronik durch einen integrierten Schaltkreis (IC) gesteuert. Man verwendet dazu oft das Bauteil L293D, weil es mit Stromstärken von bis zu 1.2 A klarkommt. Dies ist für unsere Zwecke ausreichend. Mit der L293D kann man sogar zwei Motoren gleichzeitig steuern (Vgl. Abbildung unten).

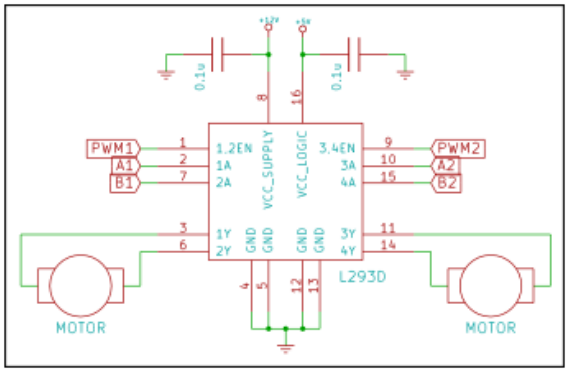
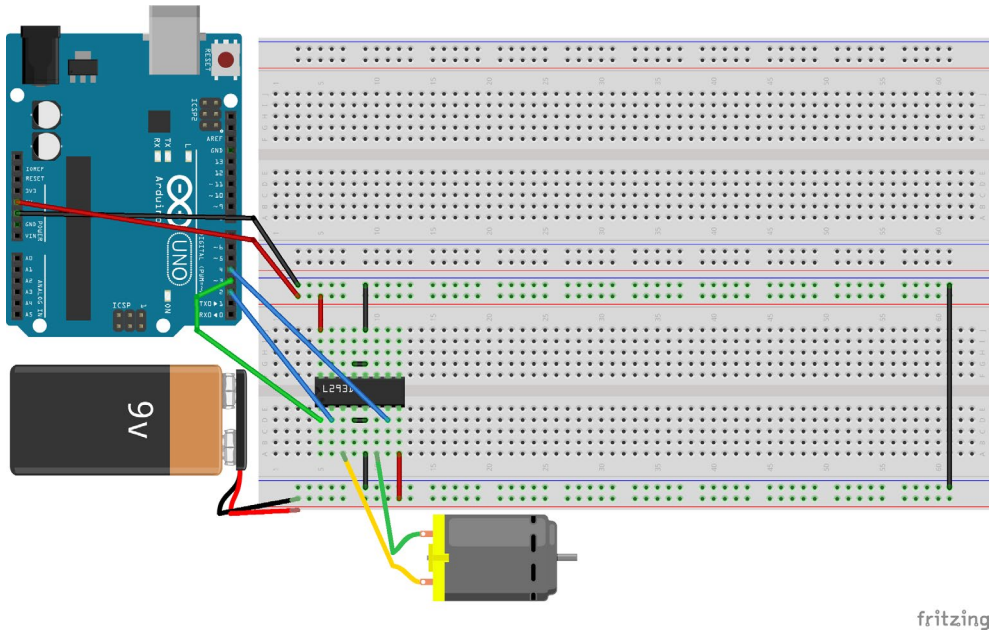


Abbildung 10 Schaltschema des Bauteils L293D
Quelle: https://nathandumont.com/blog/h-bridge-tutorial

Beispiel 7

Die nachfolgende Abbildung zeigt eine einfache Schaltung von einem Motor mit dem Bauteil L293D. Der Programmcode ist direkt darunter zu finden. Beachten Sie beim Nachbauen der Schaltung, dass die halbrunde Einbuchtung der L293D nach links in Richtung des Arduino Uno schaut. Setzt man dieses Bauteil verkehrt herum ein, kann es irreparabel beschädigt werden. Die externe Stromquelle wird benötigt, weil der Motor eine Spannung von 9 V benötigt. Idealerweise verwenden Sie eine 9 V Batterie.




```
int hBridge1A = 2;
int hBridge12Enable = 3;
int hBridge2A = 4;

void setup() {
  pinMode(hBridge1A, OUTPUT);
  pinMode(hBridge12Enable, OUTPUT);
  pinMode(hBridge2A, OUTPUT);
}

void loop() {
  digitalWrite(hBridge1A, HIGH);
  digitalWrite(hBridge2A, LOW);
  digitalWrite(hBridge12Enable, HIGH);
  delay(2000);
  digitalWrite(hBridge12Enable, LOW);
  delay(2000);
}
```

Das obige Beispiel zeigt, dass man mit einer H-Brücke einen Gleichstrommotor steuern kann. Die digitalen Pins 2 und 4 sind dabei zur Bestimmung der Richtung notwendig. Der PWM-Pin 3 kann zur Regulierung des Motors verwendet werden. Damit kann man die Geschwindigkeit steuern.

Aufgabe 6

Verändern Sie das Programm aus Beispiel 7 so, dass der Motor zuerst 3 Sekunden lang mit maximaler Geschwindigkeit in die eine Richtung dreht und anschliessend 4 Sekunden lang mit halber Geschwindigkeit in die Gegenrichtung. Verwenden Sie dazu den Befehl «analogWrite».

Funktionen in C++

Funktionen dienen in C++ dazu, oft verwendeten Code abzukürzen. Sie kennen das vielleicht von Makros aus Ihrem Tabellenkalkulationsprogramm. Das folgende Beispiel zeigt, wie man die Motorsteuerung aus dem letzten Abschnitt benutzerfreundlicher gestalten kann.

Beispiel 8

Die Schaltung aus dem letzten Beispiel wird auch für dieses Beispiel verwendet. Hier also das neue Programm mit zwei Funktionen «`forward`» und «`backward`» zur Steuerung des Motors.

```
int hBridge1A = 2;
int hBridge12Enable = 3;
int hBridge2A = 4;

void setup() {
  pinMode(hBridge1A, OUTPUT);
  pinMode(hBridge12Enable, OUTPUT);
  pinMode(hBridge2A, OUTPUT);
}

void forward(int speed){
  digitalWrite(hBridge1A, HIGH);
  digitalWrite(hBridge2A, LOW);
  analogWrite(hBridge12Enable, speed);
}

void backward(int speed){
  digitalWrite(hBridge1A, LOW);
  digitalWrite(hBridge2A, HIGH);
  analogWrite(hBridge12Enable, speed);
}

void loop() {
  forward(255);
  delay(3000);
  backward(128);
  delay(4000);
}
```

Die neuen Befehle «`forward`» und «`backward`» sehen auf den ersten Blick ganz ähnlich aus wie `setup` und `loop`. Dies liegt daran, dass auch `setup` und `loop` Funktionen sind. Der Unterschied besteht darin, dass `setup` und `loop` von der Arduino IDE vorgegeben sind[1]. Wenn wir eigene Funktionen programmieren, können wir die Namen dieser Funktionen (fast) frei wählen (Stichwort ASCII). Die Variablen in den runden Klammern nach dem Funktionsnamen (hier: speed) können ebenfalls beliebig benannt werden. Es handelt sich um so genannte Parameter, mit denen man Werte an die Funktion übergeben kann, z.B. 255 oder 128 als Geschwindigkeit. Der Ausdruck «`void`» bedeutet, dass die Funktionen keine Werte zurückgeben, wohingegen die Funktion `digitalRead` entweder HIGH oder LOW zurückgibt.

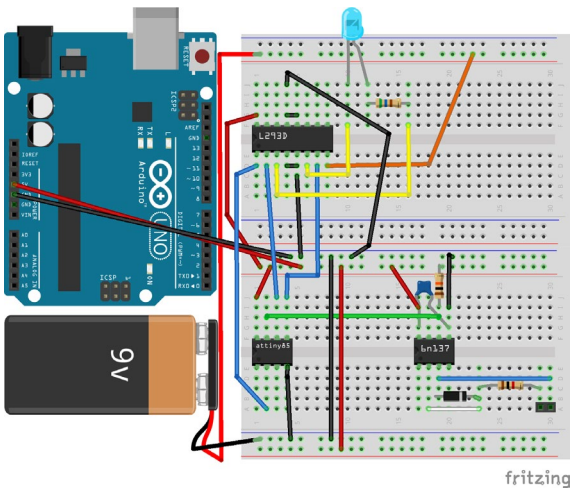
Aufgabe 7

Bauen Sie eine Schaltung mit zwei Motoren. Steuern Sie diese mit dem Bauteil L293D. Schreiben Sie dann ein Programm mit vier Funktionen motorAForward, motorABackward, motorBForward, motorBBackward. Überlegen Sie sich, wie man damit ein Fahrzeug steuern könnte. Vielleicht ergibt dies wieder neue Funktionen…

Anwendungsaufgabe

Bauen Sie eine Weichenschaltung für Ihre Modelleisenbahn. Verwenden Sie dazu das Bauteil L293D.

Zusatz: Wenn Sie über eine Z21 verfügen, können Sie die Weiche über DCC steuern. Sie benötigen dazu einen Optokoppler 6N137, mit dem Sie das DCC-Signal dekodieren können. Die Schaltung dazu zeigt Ihnen die folgende Abbildung. Für die Dekodierung des DCC-Signals erhalten Sie von Ihrer Lehrperson eine passende Library.



Die Weiche wird hier durch eine blaue LED dargestellt, daher der Vorwiderstand.

[1] Solche Funktionen werden callback-Funktionen genannt.

Posten VI: I²C und die Ausgabe auf einem Display

In diesem Posten geht es darum, Text und Grafik auf einem angehängten Display auszugeben. Dafür stehen viele Displays mit unterschiedlicher Auflösung und Technologie zur Verfügung. Eine relativ neue Technologie bieten die OLED-Displays. Es handelt sich dabei um organische LEDs, welche kostengünstiger hergestellt werden können als herkömmliche LED-Displays. Wir verwenden OLED Displays mit einer Diagonale von 0.96" und einer Auflösung von 128 x 64 Pixeln (Vgl. Abbildung unten).

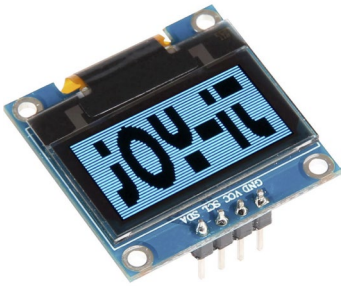
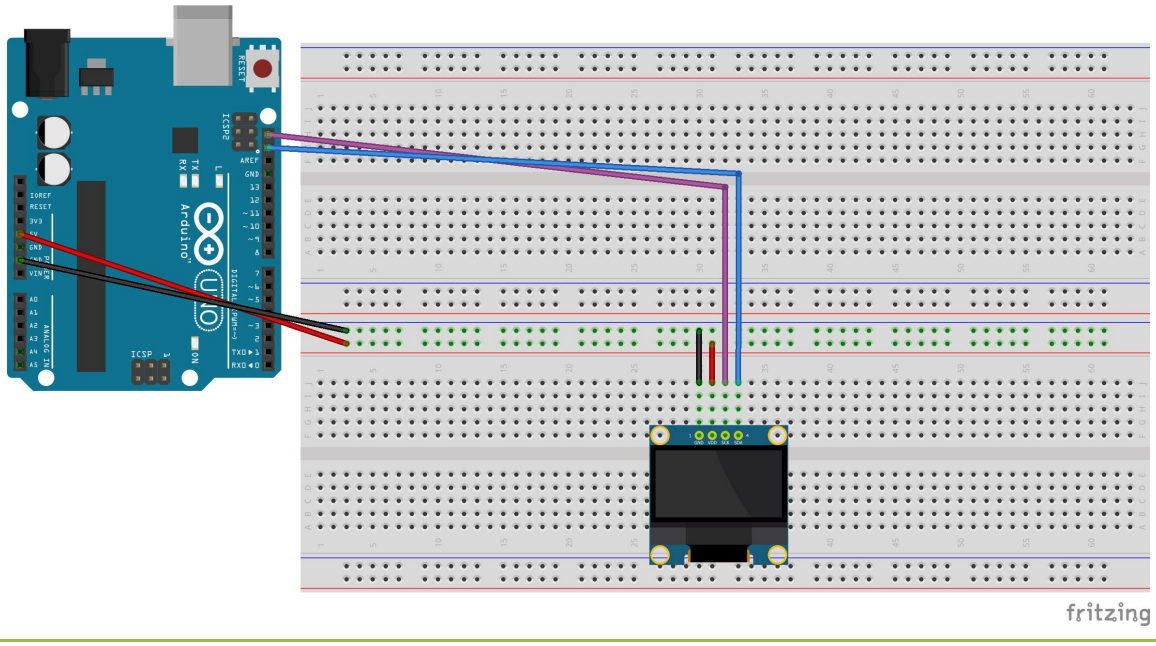


Abbildung 11 OLED Display mit 128 x 64 px, 0.96"

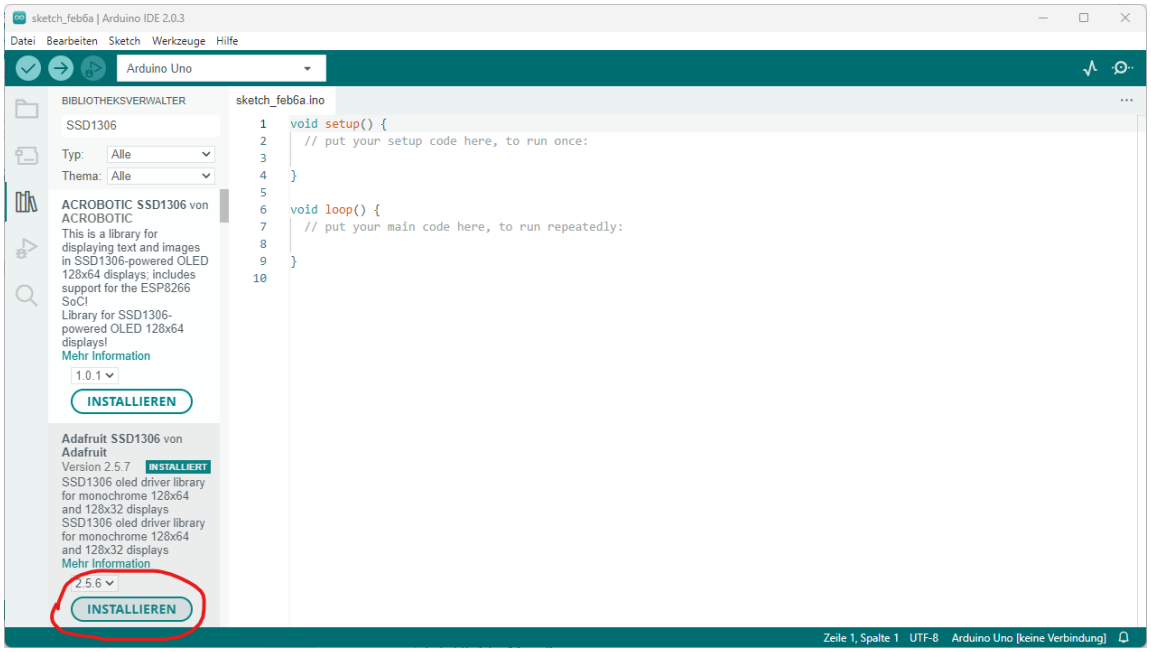
Ein solches Display lässt sich über I²C mit dem Arduino Uno verbinden. Die Abkürzung steht für **I**nter-**I**ntegrated **C**ircuit. Dabei handelt es sich um einen seriellen Datenbus zur Übermittlung von Informationen zwischen Bauteilen wie Sensoren, Displays, etc. und dem Mikrocontroller. Der Arduino unterstützt I²C standardmässig auf den analogen Pins A4 und A5 bzw. auf den eigens dafür markierten Steckplätzen SDA (**S**erial **Da**ta) und SCL (**S**erial **Cl**ock)

Beispiel 9

Die nachfolgende Abbildung zeigt, wie das Display angeschlossen wird.



Damit man das Display verwenden kann, braucht man eine Grafik-Library namens Adafruit SSD1306. Die folgende Abbildung zeigt, wie man diese installiert.



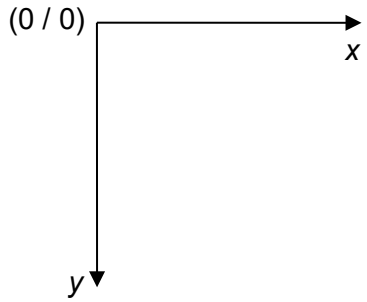
Danach testen wir das Display durch das Beispielprogramm zur Library. Dieses kann mit dem Menübefehl «Datei > Beispiele > Adafruit SSD1306 > ssd1306_128x64_i2c» geladen werden. Damit das Programm läuft, muss man eventuell noch auf Zeile 35 die Adresse 0x3D auf 0x3C ändern. Die I²C-Adresse ist jeweils abhängig vom Hersteller.

Wenn dies funktioniert hat, können Sie das folgende Programm testen. Es soll einen Bildpunkt von der Position (10, 10) aus zunächst nach rechts bis zum rechten Rand wandern und dort umkehren lassen. Beachten Sie dabei, dass der Nullpunkt in der Computergrafik in der linken oberen Ecke liegt (Vgl. Abbildung unten).

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET     -1 // Reset pin #
#define SCREEN_ADDRESS 0x3C // could also be 0x3D
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

int x = 10;  // x start
int y = 10;  // y start
int v = 2;   // speed in x-direction

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
}
```

```
void loop() {
  display.clearDisplay();
  display.drawPixel(x, y, SSD1306_WHITE);
  x = x + v;
  if (x > SCREEN_WIDTH) {  // right border
    v = - v;
  }
  display.display();
}
```

Die folgende Abbildung zeigt das Koordinatensystem in der Computergrafik:



Aufgabe 8

Schreiben Sie ein Billardprogramm für das OLED-Display. Das Pixel soll dabei an jeder Bande abprallen. Benützen Sie dafür den Code aus Beispiel 9 als Grundlage.

Tipp: Sie brauchen eine weitere Variable w für die Geschwindigkeit in y-Richtung.

Anwendungsaufgabe

Bauen Sie eine Schaltung zum Messen der Geschwindigkeit Ihrer Modelleisenbahn-Loks. Halten Sie sich dazu an die Anleitung aus dem folgenden YouTube-Video:

https://www.youtube.com/watch?v=Ywsuv9l1PBs

Ihre Lehrperson wird Ihnen die benötigten Bauteile zur Verfügung stellen.

Tipp: Achten Sie darauf, dass der korrekte Massstab eingestellt ist.

Operator	Beschreibung
&&	Logisches Und
||	Logisches Oder
!	Logisches Nicht, verneint den Ausdruck danach

Tabelle 2: Logische Operatoren in C++

Literaturverzeichnis

Arduino.cc. (Februar 2023). *Tutorials | Arduino Documentation*. Von Arduino: https://docs.arduino.cc/tutorials/ abgerufen

Brunner, B. (15. Juni 2018). Arduino Lehrgang. Kreuzlingen, Thurgau, Schweiz.

Mahn, J. (10. Mai 2019). *Erste Schritte mit den Mikrocontrollern ATtiny84 und 85*. Von Heise Online: https://www.heise.de/ratgeber/Erste-Schritte-mit-den-Mikrocontrollern-ATtiny84-und-85-4399393.html abgerufen

Popovic, F. (1. Januar 2016). *arduino - Eine Einführung*. Von Frerks kleine Welt: http://popovic.info/html/arduino/arduinoUno_1.html abgerufen

Wikimedia Foundation Inc. (30. 9 2020). *Wikipedia - Die freie Enzyklopädie*. Von http://de.wikipedia.org/ abgerufen

Anhang A: Lösungen

Lösungen zum Posten I

Aufgabe 1

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Hello World!");
}
```

Zusatz

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Hello World!");
  delay(5000);
}
```

Lösungen zum Posten II

Aufgabe 2

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(3000);
  digitalWrite(13, LOW);
  delay(2000);
}
```

Aufgabe 3

```
int LEDPin = 11;

void setup() {
  pinMode(LEDPin, OUTPUT);
}

void loop() {
  digitalWrite(LEDPin, HIGH);
  delay(4);
  digitalWrite(LEDPin, LOW);
  delay(1);
}
```

Lösungen zum Posten III

Aufgabe 4

```
int redLEDPin = 0;
int yellowLEDPin = 1;
int greenLEDPin = 2;

void setup() {
  pinMode(redLEDPin, OUTPUT);
  pinMode(yellowLEDPin, OUTPUT);
  pinMode(greenLEDPin, OUTPUT);
}

void loop() {
  digitalWrite(greenLEDPin, LOW);
  digitalWrite(redLEDPin, HIGH);
  delay(1000);
  digitalWrite(redLEDPin, LOW);
  digitalWrite(yellowLEDPin, HIGH);
```

```
  delay(1000);
  digitalWrite(yellowLEDPin, LOW);
  digitalWrite(greenLEDPin, HIGH);
  delay(1000);
}
```

Lösungen zum Posten IV

Aufgabe 5

```
int inputPin = 2;
int LEDPin = 13;

void setup() {
  pinMode(inputPin, INPUT);
  pinMode(LEDPin, OUTPUT);
}

void loop() {
  bool buttonPressed = digitalRead(inputPin);
  if(buttonPressed) {
    digitalWrite(LEDPin, LOW);
  }
  else {
    digitalWrite(LEDPin, HIGH);
  }
}
```

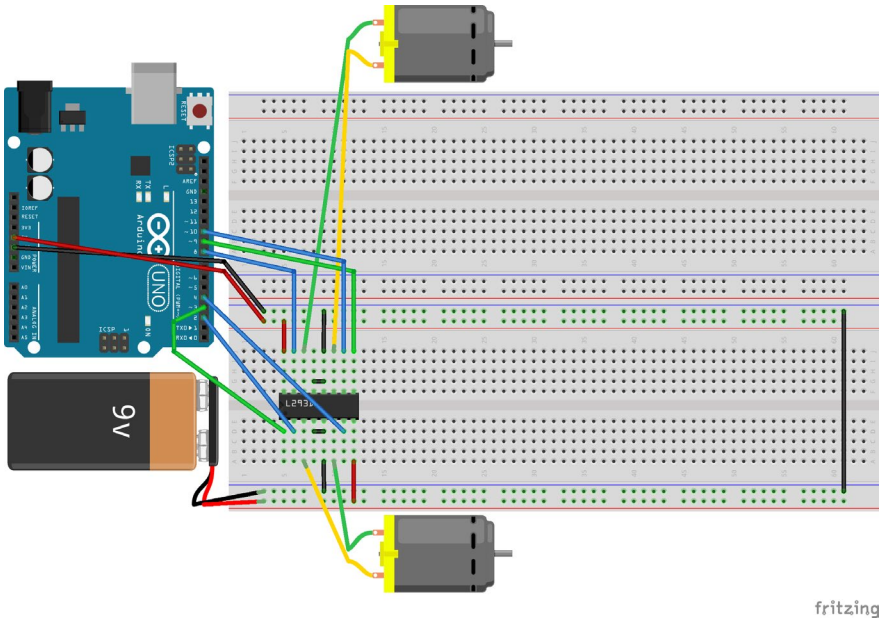
Lösungen zum Posten V

Aufgabe 6

```
int hBridge1A = 2;
int hBridge12Enable = 3;
int hBridge2A = 4;

void setup() {
  pinMode(hBridge1A, OUTPUT);
  pinMode(hBridge12Enable, OUTPUT);
  pinMode(hBridge2A, OUTPUT);
}

void loop() {
  digitalWrite(hBridge1A, HIGH);
  digitalWrite(hBridge2A, LOW);
  analogWrite(hBridge12Enable, 255);
  delay(3000);
  digitalWrite(hBridge1A, LOW);
  digitalWrite(hBridge2A, HIGH);
```

```
  analogWrite(hBridge12Enable, 127);
  delay(4000);
}
```

Aufgabe 7




```
int hBridge1A = 2;
int hBridge12Enable = 3;
int hBridge2A = 4;
int hBridge3A = 8;
int hBridge34Enable = 9;
int hBridge4A = 10;

void setup() {
  pinMode(hBridge1A, OUTPUT);
  pinMode(hBridge12Enable, OUTPUT);
  pinMode(hBridge2A, OUTPUT);
  pinMode(hBridge3A, OUTPUT);
  pinMode(hBridge34Enable, OUTPUT);
  pinMode(hBridge4A, OUTPUT);}

void motorAForward(int speed){
  digitalWrite(hBridge1A, HIGH);
  digitalWrite(hBridge2A, LOW);
  analogWrite(hBridge12Enable, speed);
}

void motorABackward(int speed){
  digitalWrite(hBridge1A, LOW);
  digitalWrite(hBridge2A, HIGH);
  analogWrite(hBridge12Enable, speed);
}
```

```
void motorBForward(int speed){
  digitalWrite(hBridge3A, HIGH);
  digitalWrite(hBridge4A, LOW);
  analogWrite(hBridge34Enable, speed);
}

void motorBBackward(int speed){
  digitalWrite(hBridge3A, LOW);
  digitalWrite(hBridge4A, HIGH);
  analogWrite(hBridge34Enable, speed);
}

void loop() {
  motorAForward(255);  // forward
  motorBForward(255);
  delay(2000);
  motorAForward(255);  // to the right
  motorBForward(127);
  delay(2000);
  motorABackward(255); // backward
  motorBBackward(255);
  delay(2000);
}
```

Lösungen zum Posten VI

Aufgabe 8

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64,
0x3C for 128x32
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

int x = 10;  // x start
int y = 10;  // y start
int v = 2;   // speed in x-direction
int w = 3;   // speed in y-direction

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
}
```

```
void loop() {
  display.clearDisplay();
  display.drawPixel(x, y, SSD1306_WHITE);
  x = x + v;
  y = y + w;
  if (x > SCREEN_WIDTH || x < 0) {
    v = - v;
  }
  if (y > SCREEN_HEIGHT || y < 0) {
    w = - w;
  }
  display.display();
}
```

Anhang B: Hilfsmittel

B1 Das Breadboard

Breadboards oder Steckplatinen dienen in der Elektronik als Grundlage für die Entwicklung von Schaltungen. Die folgende Abbildung zeigt die interne Verdrahtung eines Breadboards.

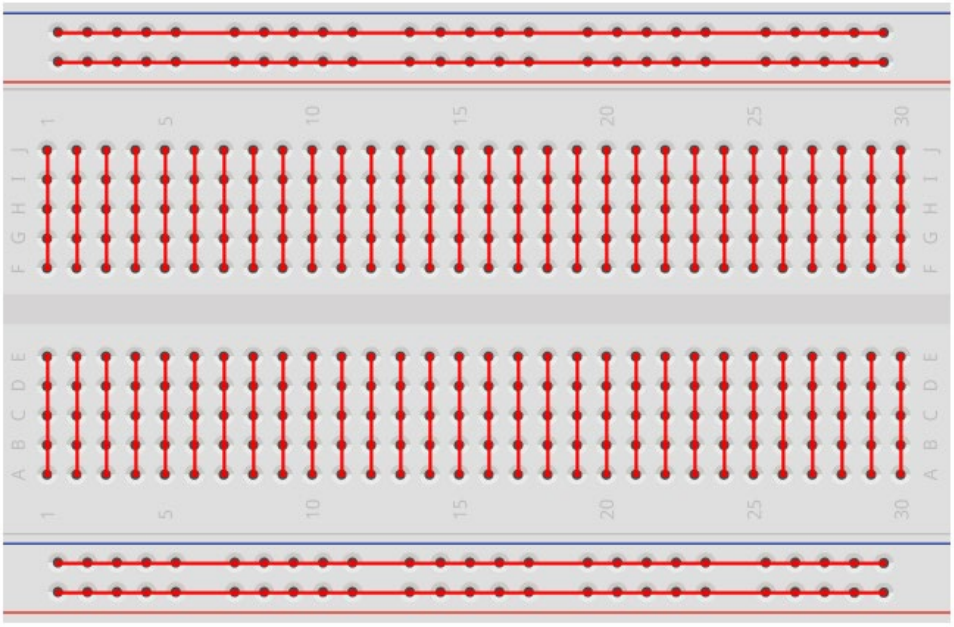
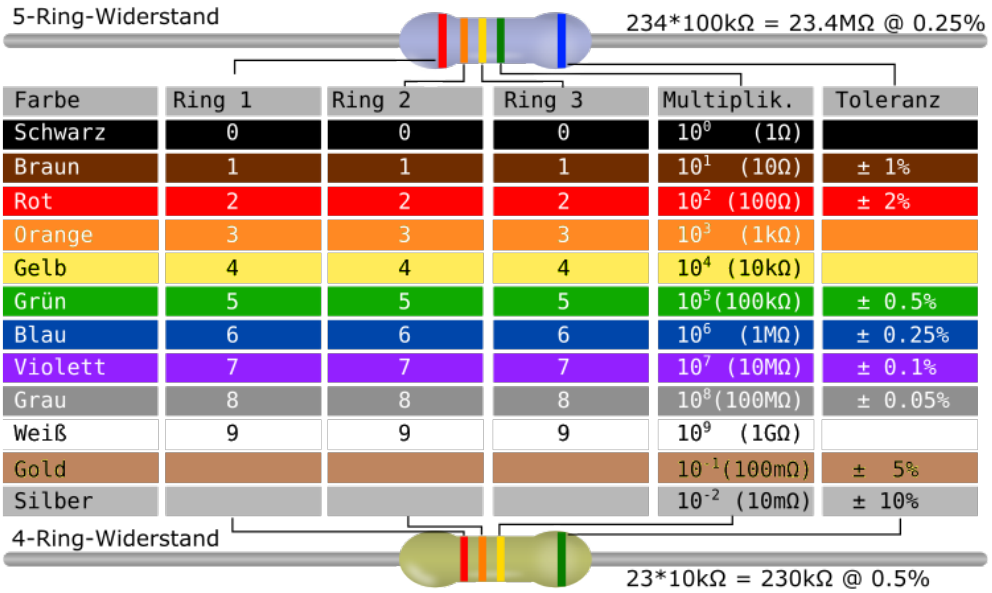


Abbildung 12 Interne Verdrahtung des Breadboards
Quelle: https://www.datenreise.de/raspberry-pi-wie-verwendet-man-ein-breadboard-steckplatine-anleitung/

B2 Widerstände

Die Stärke eines Widerstandes wird in der Masseinheit Ohm (Ω) angegeben. Statt eines Zahlenwertes werden aber Farbcodes zur Bestimmung von Widerständen eingesetzt. Wie das geht, zeigt die folgende Tabelle:



Farbe	Ring 1	Ring 2	Ring 3	Multiplik.	Toleranz
Schwarz	0	0	0	10^0 (1Ω)	
Braun	1	1	1	10^1 (10Ω)	± 1%
Rot	2	2	2	10^2 (100Ω)	± 2%
Orange	3	3	3	10^3 (1kΩ)	
Gelb	4	4	4	10^4 (10kΩ)	
Grün	5	5	5	10^5(100kΩ)	± 0.5%
Blau	6	6	6	10^6 (1MΩ)	± 0.25%
Violett	7	7	7	10^7 (10MΩ)	± 0.1%
Grau	8	8	8	10^8(100MΩ)	± 0.05%
Weiß	9	9	9	10^9 (1GΩ)	
Gold				10^{-1}(100mΩ)	± 5%
Silber				10^{-2} (10mΩ)	± 10%

Abbildung 13 Farbcodes von Widerständen
Quelle: https://openclipart.org/detail/250258/resistor-color-code-table-german

Widerstände treten in dieser Werkstatt hauptsächlich in zwei Situationen auf:

1. als Vorwiderstände von Bauteilen wie LEDs, Optokopplern, etc. In dieser Funktion werden sie meist so bemessen, dass die Stromstärke unter 0.02 A liegt ($U = R \cdot I$).
2. als Pull-down- oder Pull-up-Widerstände, um einen Schaltkreis zu schliessen. In dieser Funktion sind sie in der Regel hochohmig, z.B. 10 kΩ.

B3 Arduino-Shield für die ISP des ATtiny85

Das nachfolgend abgebildete Arduino-Shield wurde eigens für diesen Kurs konzipiert. Es erlaubt die ISP für den ATtiny85 und für den grösseren ATtiny84. Bei beiden Mikroprozessoren ist darauf zu achten, dass sie korrekt auf dem IC-Sockel platziert werden. Die halbkreisförmige Kerbe auf dem IC-Sockel muss mit der halbkreisförmigen Einbuchtung auf dem ATtiny85 übereinstimmen.

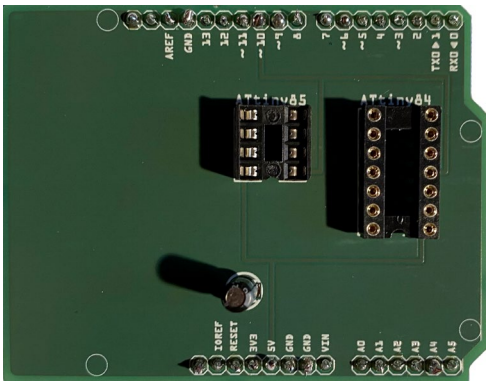


Abbildung 14 Arduino-Shield für ISP

Anhang C: Hintergrundinformationen für die Lehrperson

Wieso Modelleisenbahnen?

Nach Ansicht des Autors sind Modelleisenbahnen ein ideales Übungsfeld für den Umgang mit Mikrocontrollern. Zum einen liegt die Betriebsspannung gängiger Systeme im Mittel bei 20 V und somit in einem für die Schülerinnen und Schüler ungefährlichen Bereich. Zum anderen haben viele Schülerinnen und Schüler bereits einen Bezug zu Modelleisenbahnen, und sei es nur über die Brio-Bahn, mit der sie als Kleinkinder gespielt haben. Die Modellwelt ermöglicht ausserdem, Erfahrungen in einem Umfeld zu sammeln, das in direktem Bezug zur Realität steht. So sind mittlerweile täuschend echte Modellbahnsignale im Fachhandel erhältlich.

Digital Command Control (DCC)

Digital Command Control ist ein Übertragungsstandard für Steuersignale von Modell-Loks, Bahnsignalen, Weichen, etc. über die Betriebsspannung der Schienen. Es handelt sich dabei um eine einfache Form der seriellen Kommunikation mit Rechtecksignalen (Vgl. Abbildung).

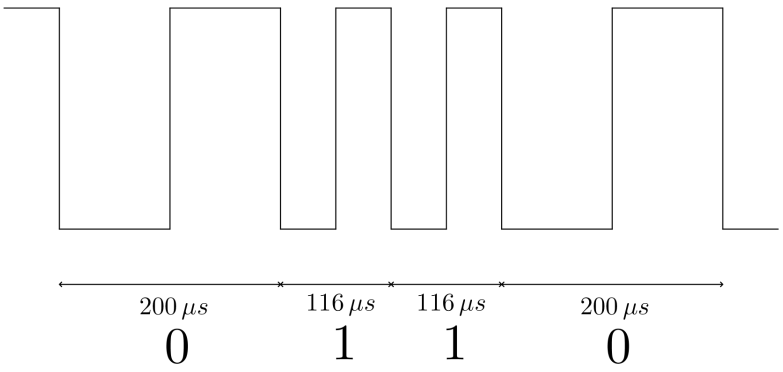


Abbildung 15 DCC-Signal

Die Codierung der Signale folgt einem relativ einfachen Muster mit einer Präambel von Eins-Bits gefolgt von mehreren Adress- und Datenbytes. Den Abschluss bildet ein Prüfbyte zur Fehlererkennung. Ergänzend zu dieser Arbeit wurde eine Arduino Library mit Beispielen für unsere Schülerinnen und Schüler erstellt. Sie finden sie auf SwissEduc als Begleitmaterial zu dieser Werkstatt.

Alternativ kann auch die DCC-Library von Alex Shepherd verwendet werden, welche in der Arduino-IDE verlinkt ist. Weitere Informationen zur Modelleisenbahnsteuerung mit DCC finden Sie unter dem folgenden Link:

https://www.nmra.org/sites/default/files/standards/sandrp/pdf/s-9.2.1_dcc_extended_packet_formats.pdf