

Lernkarten-Set

Jan Thar

Mikrocontroller Arduino & Co

Impressum

Die Informationen in diesem Lernkarten-Set wurden mit größter Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Verlag und Autoren übernehmen keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene Fehler und deren Folgen.

Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen. Der Verlag richtet sich im wesentlichen nach den Schreibweisen der Hersteller. Das Werk steht unter der CC-BY-NC 3.0-Lizenz.

Kommentare und Fragen können Sie gerne an uns richten:

Bombini Verlags GmbH

Kaiserstraße 235

53113 Bonn

www.bombini-verlag.de

E-Mail: service@bombini-verlag.de

Copyright: © 2020 by Bombini Verlag

Bibliografische Information Der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-946496-19-9

Inhalt



Inhalt

Mikrocontroller sind kleine, kostengünstige Computer, die zur An- und Fernsteuerung aller möglicher Sensoren und Aktuatoren verwendet werden können. Dies erlaubt einen einfachen, modularen Schaltungsaufbau mit flexibler Programmierung, was auch für Einsteiger einfach aufzubauen ist. Insbesondere grafische Programmierung erlaubt dabei einen kinderleichten Zugang. Hier zeigen wir anhand der Arduino-IDE Programm- und Anwendungsbeispiele für die verschiedenen Basisfunktionen eines Controllers, hier der Atmega 32U4 als Micro Pro Modul.

Mit diesen Grundlagen können danach eigene Anwendungen erstellt werden - die Nutzung und Anpassung bereits vorhandener Module und Programme vereinfacht schließlich das Leben.

Mikrocontroller



Warum Mikrocontroller nutzen?

Microcontroller

Ein kleiner Computer für alle Lebenslagen: Mikrocontroller lassen sich flexibel für unterschiedliche Aufgaben programmieren - ob es nun darum geht, eine LED blinken zu lassen oder komplexe Sensorsysteme abzufragen, auszuwerten und miteinander zu kommunizieren. Auch wenn der Controller alleine manchmal etwas kompliziert erscheint, Modulplattformen wie Arduino erleichtern die Nutzung enorm.

Indem nur ein Programm durchgeführt wird, ist er auch echtzeitfähig (bei den Computern wird zwischen den einzelnen laufenden Aufgaben umgeschaltet), was einfache korrekte Zeitabläufe ermöglicht und die Programmierung vereinfacht.

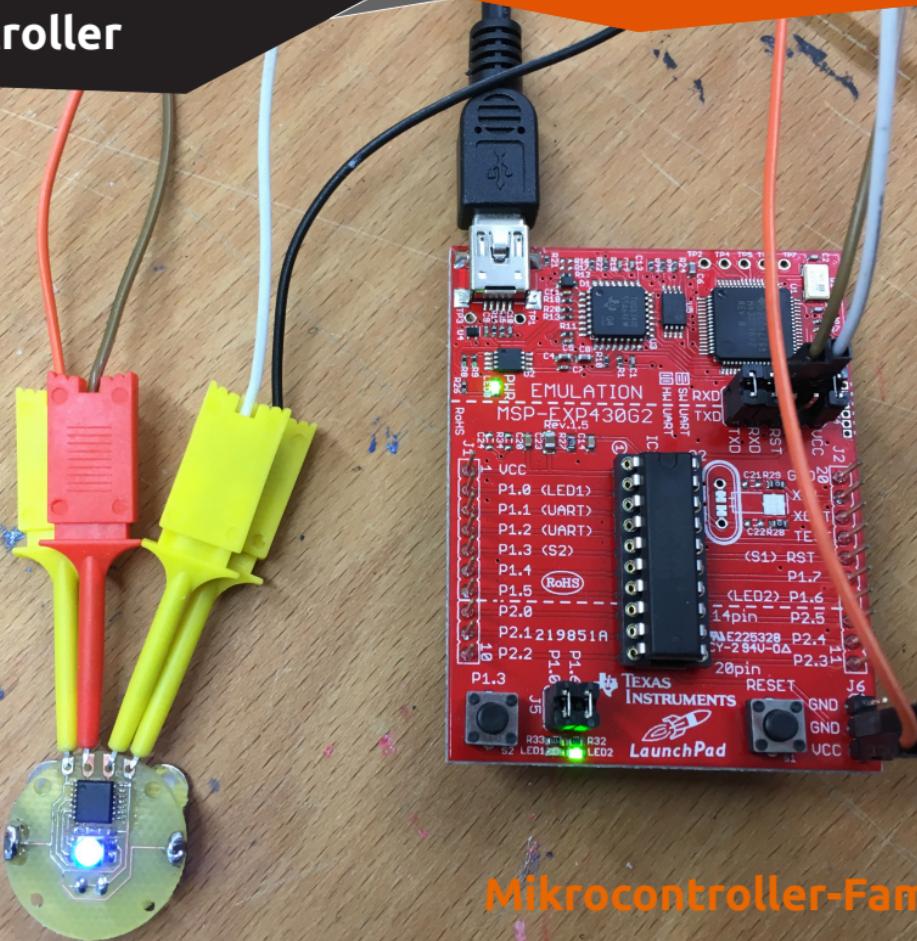
Er ermöglicht die direkte Ansteuerung von Elektronikkomponenten, Sensoren als Eingabemittel über LEDs bis hin zu komplexen Anzeigen sowie Motoren. Kommunikationschnittstellen ermöglichen die Steuerung komplexer Erweiterungen.

Dadurch kann im Vergleich zum früheren Hardwareaufbau die Komplexität des Elektronikaufbaus reduziert werden und in die einfachere und flexiblere Programmierung des Controllers verlagert werden.

Dies erlaubt einen einfachen Einstieg und ermöglicht durch Hard- und Softwaremodule auch komplexe Projekte für Personen, die sich nicht mit Elektrotechnik auskennen.

Warum Mikrocontroller nutzen?

Mikrocontroller



Mikrocontroller-Familien

Mikrocontroller

Es gibt eine Vielzahl von unterschiedlichen Mikrocontrollern. Zunächst einmal unterscheiden sie sich nach Herstellern in verschiedene Familien, die ähnlich aufgebaut sind (und programmiert werden), sich aber hinsichtlich Leistung (Speicher, Geschwindigkeit) und Anzahl sowie Funktionalität der Eingabe-/Ausgabe-pins (an denen die externe Elektronik angeschlossen wird) unterscheiden.

Eine für Einsteiger populäre Baureihe sind die 8-Bit-Mikrocontroller Atmega bzw. die kleineren Attinys und größeren Megas.

MSP430 existieren in ähnlichen Leistungsklassen, mit den Launchpads gibt es auch hier eine Boardreihe für Einsteiger, die mit der Energia-IDE genauso wie die Atmegas mit der Arduino-IDE programmiert werden können.

Die Cortex M3/M4 ist eine deutlich leistungsfähigere 32-Bit-Controllerklasse; auch hier gibt es passende Boards, die Arduino-IDE zu programmieren.

Als momentan extrem kostengünstiger 32-Bit-Controller, mit Bluetooth und WLAN ausgestattet, ist der ESP32 zu nennen, welcher somit ideal für vernetzte Projekte ist.

Eine andere Art, Intelligenz einzubauen, sind weniger weit verbreitete FPGAs, bei denen statt des Ablaufs eine elektrische Schaltung einprogrammiert wird.

Arten von Mikrocontrollern

Mikrocontroller



Zehn Regeln

Mikrocontroller

- Controllerauswahl Teil 1: Ein Controller hat eine Speichergröße und eine Anzahl I/O-Pins mit bestimmten Sonderfunktionen. Wähle einen Controller passend zu deinem Bedarf (mit Reserve).
- Controllerauswahl Teil 2: Mikrocontroller sind für bestimmte Spannungen (5V oder 3,3V) ausgelegt, wird eine höhere angelegt, führt dies zur Zerstörung oder Beschädigung. Magic Smoke lässt sich nicht nachfüllen.
- Immer eine Funktion/ein Modul nach der/dem anderen integrieren.
- Serielle Ausgaben zum Debugging verwenden (stimmen die Werte mit den erwarteten überein?)
- Programmierpins und serielle Schnittstelle wegen Programmieren des Controllers möglichst frei von externer Elektronik lassen.
- Vorhandenen Programmcode weiterverwenden - man muss das Rad nicht neu erfinden, kann es aber natürlich verwenden und verbessern (GitHub ist prima für Versionierung; im Internet findet man viele nützliche Infos).
- Kommentare reichlich verwenden und sinnvolle Bezeichnungen verwenden, damit man es selber später noch versteht.
- Lesbarerer Code: Einrückungen, Module, Klassen, Funktionen, Libraries.
- Elektronikmodule, Breakoutboard verwenden.
- (Später:) Umbau auf eigene PCB (ist kleiner und zuverlässiger).

Zehn Regeln

Mikrocontroller



Tipps und Tricks

Microcontroller

- Für fast jede Teilaufgabe wird es eine ähnliche Sache geben, die bereits jemand mit dem Arduino durchgespielt hat. Der Vorteil von Open-Source-Systemen ist ja, dass man auf den Schultern seiner Vorgänger stehen darf (und deren Entwicklungen weiterführt).
- Bluetooth-Debugging statt serieller Schnittstelle? Ein HC05-Modul einfach an die serielle Schnittstelle hängen und mit dem Rechner koppeln, danach kann man wie zuvor kabelgebunden kommunizieren.
- Spannungslevel der Controller: Wenn ein 3,3V-Controller mit einem 5V-System zu kommunizieren versucht, können Verständnisschwierigkeiten auftreten.
- Verbindungen prüfen - Wackelkontakte machen keine Freude.
- Spannungen und Stromverbrauch überprüfen.
- Digitale Werte, analoge Werte und auch serielle Protokolle lassen sich messen und ausgeben.

Tipps und Tricks

Arduino

Blink | Arduino 1.8.10

Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to the correct LED pin independent of which board is used.
If you want to know what pin the on-board LED is connected to on your Arduino model, check the Technical Specs of your board at:
<https://www.arduino.cc/en/Main/Products>

modified 8 May 2014
by Scott Fitzgerald
modified 2 Sep 2016
by Arturo Guadalupi
modified 8 Sep 2016
by Colby Newman

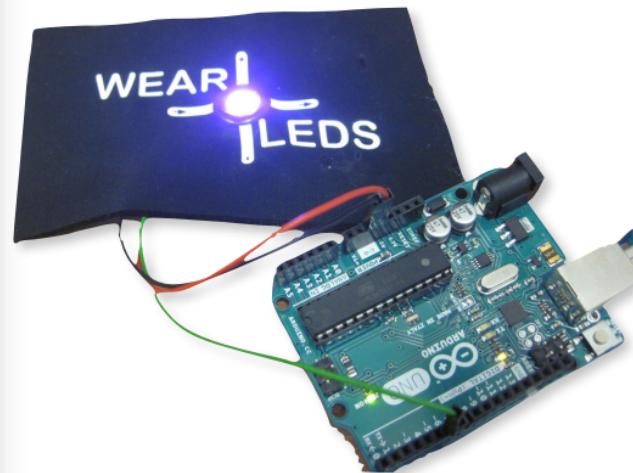
This example code is in the public domain.

<http://www.arduino.cc/en/Tutorial/Blink>

```
/*
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None auf /dev/cu.SLAB_USBtoUART



Warum Arduino?

Arduino

Obwohl eine Vielzahl von Controllern und Programmierumgebungen existiert, ist für Einsteiger die Arduino-Programmierung und die dazu passenden Controller immer noch am besten geeignet.

Zur Arduino-IDE existiert auch eine grafische Variante, für etwas komplexere Systeme muss man allerdings irgendwann auf die Textform umsteigen.

In der Arduino-IDE wird dabei die hardware-spezifische Seite noch möglichst weitgehend vereinfacht, wobei hier aber trotzdem notfalls auch auf die direkten hardwarespezifischen Funktionen zurückgegriffen werden kann, sollte man es mal brauchen.

Vor allem existiert aber insbesondere für den Arduino ein sehr großes Ökosystem an Modulen und Libraries, auf die man zurückgreifen kann, gepflegt durch eine riesige Online-Community, die auch für Fragen zur Verfügung steht.

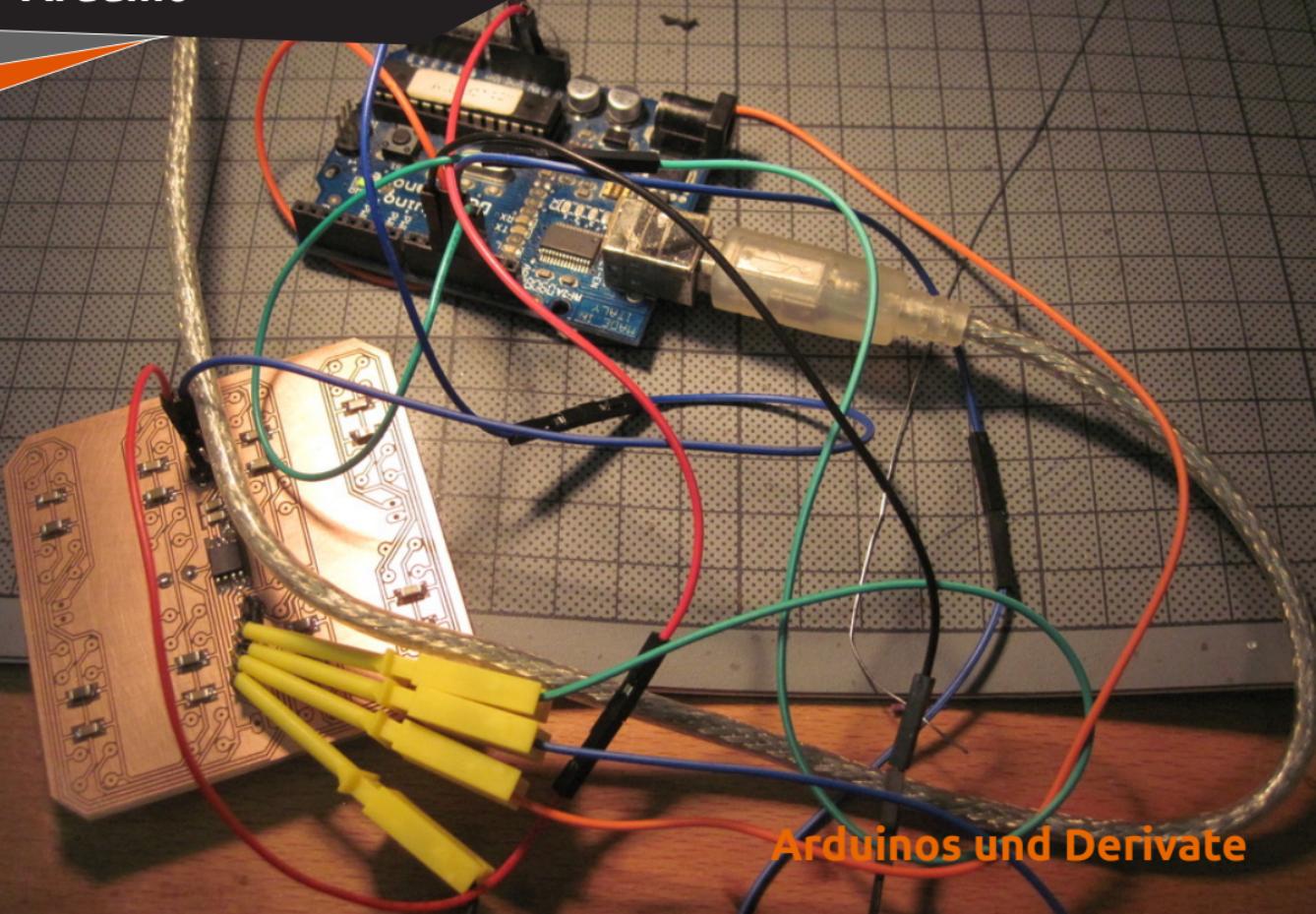
So sind aufgrund vorhandener Aufsteckmodule - sogenannte Shields - fast keine Elektronik-kenntnisse mehr notwendig. Diese sind der Hauptgrund, warum immer noch der Arduino UNO - trotz eines relativ schlechten Preis-/Leistungsverhältnisses, zum Beispiel im Vergleich zum ESP32 - eine der besten Einstiegsmöglichkeiten darstellt. Auch dieser kann mit der Arduino-IDE programmiert werden, dem Standardtool für Nicht-Hardcore-programmierer.

Auch sind viele Module aufgrund seiner Verbreitung auf die 5V Signalspannung des Arduino UNO ausgelegt.

Da die Arduino-IDE auf Processing aufbaut, mit welchem einsteigerfreundlich auch Programme für den Computer geschrieben werden können, kann auch leicht über eine serielle Schnittstelle eine Verbindung zwischen Rechner und Mikrocontroller erzeugt werden.

Warum Arduino?

Arduino



Arduinos und Derivate

Arduino

Die beiden klassischen Arbeitspferde der Arduino-Familie ist der Arduino UNO sowie der größere Mega.

Nachteil des UNO war vor allem die teure USB-zu-serielle Schnittstelle, welche inzwischen beim echten Arduino durch einen weiteren Atmega, bei chinesischen, billigeren Klonen durch einen anderen Chip (für den ein Treiber installiert werden muss) ersetzt worden ist. Bei letzteren wird natürlich nicht die Entwicklung unterstützt, und Sicherheit und Zuverlässigkeit ist nicht unbedingt gewährleistet. Der dazu verwendete 32U4 ist leistungsfähiger als der zu programmierende Atmega328. Deswegen wurden mit dem Arduino Leonardo und dem Arduino Pro Micro zwei Boards geschaffen, die diesen Controller nutzen. Da hier aber eine etwas andere Pinbelegung des Boards verwendet werden muss, sind diese nicht kompatibel mit einigen Shields, was bei Einsteigern zu Verwirrung führte.

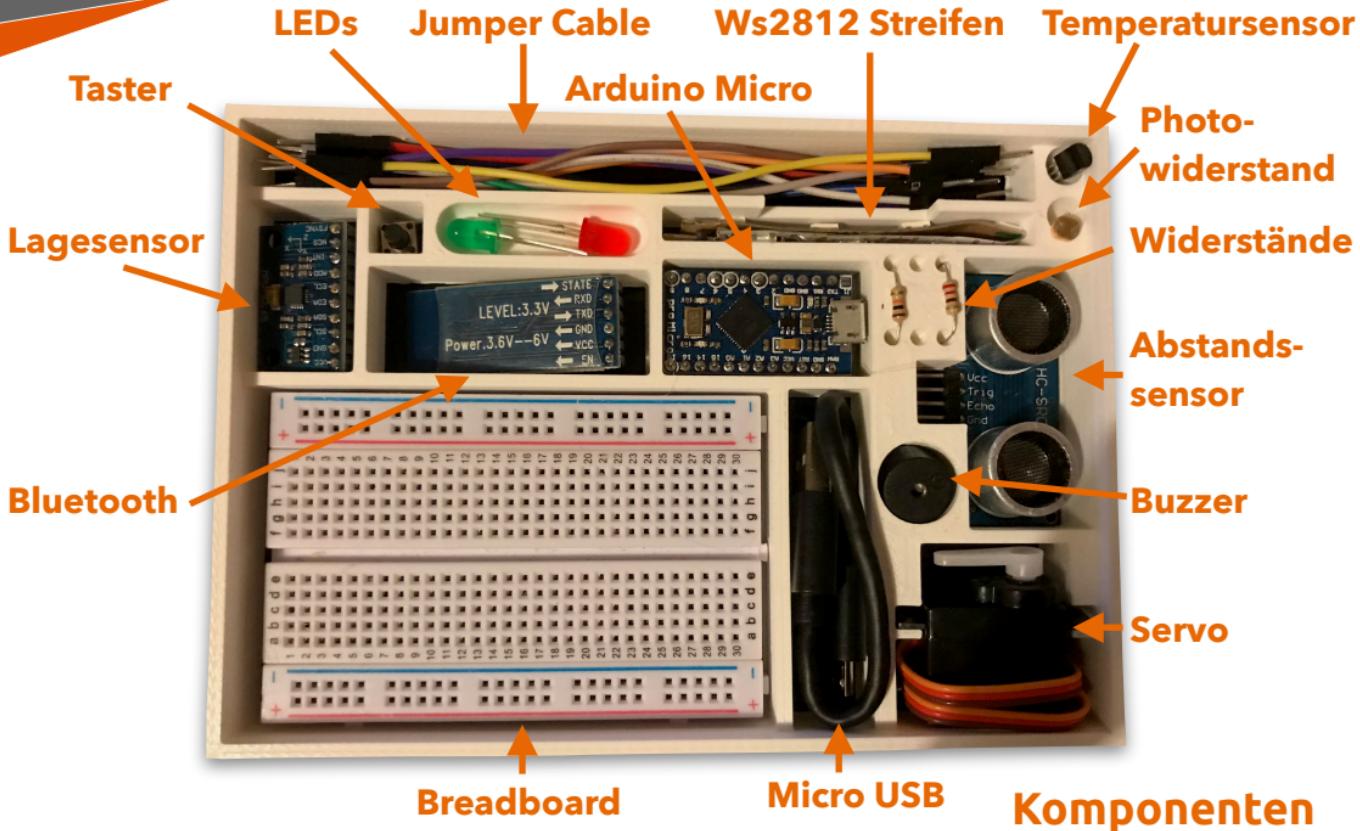
Teurere Module liefern weitere Funktionen wie WLAN oder Bluetooth. Durch einen Boardmanager kann Unterstützung von weiteren Boards geladen werden. So kann z.B auch der ESP32 mit der Arduino-IDE angesteuert werden.

Um von möglichst vielen Libraries und Wissen aus der Community zu profitieren, bleiben wir aber bei einer der klassischen Varianten.

Da wir hier keine Shields verwenden, sondern mit einem Steckbrett arbeiten, benutzen wir eine dafür passendere Variante, den Arduino Micro Pro. Er arbeitet auch mit 5V, seine 32U4 kann aber auch als Eingabegerät wie eine Tastatur am Rechner verwendet werden.

Arduinos und Derivate

Etwas Elektronik



Etwas Elektronik

Für die hier vorgestellten Beispiele werden abseits des Mikrocontrollers noch ein paar andere Elektronikbauteile verwendet:

- Breadboard
- Arduino Pro Micro
- ein 15cm langes Micro-USB-Kabel
- 10* JumperKabel
- Taster
- 2* LEDs (rot, grün)
- Photowiderstand
- Buzzer
- 1* 10-kOhm-Widerstand
- 1* 150-Ohm-Widerstand
- WS2812-Streifen
- Bluetooth-Modul HC 06
- Ultraschallabstandssensor HC-SR04
- Lagesensor MPU6050
- Servomotor
- Temperatursensor DS18B20

Komponenten

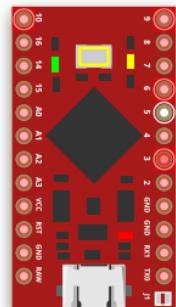
Beispiele

Digitale_Ausgabe | Arduino 1.8.10

```
// Kommentar: Wird einmal zu Beginn durchgeführt
void setup() {
    pinMode(13, OUTPUT);      // Pin 13 als Ausgang nutzen
    digitalWrite(13, HIGH);   // ... und diesen auf 5V legen
}

/* Mehrzeiliger Kommentar: Wird immer wieder wiederholt
*/
void loop() {
    // Hier machen wir noch nichts
}
```

Speichern abgeschlossen.



Digitale Ausgabe

Beispiele

Lass es leuchten!

Zunächst starten wir die Arduino-IDE und verbinden den Arduino Pro Micro und den Computer mit einem USB-Kabel. Unter Werkzeuge sollten wir dann unter Boards dieses Board finden (Arduino/Genuino Micro) und auswählen, unter Ports sollte auch schon der passende Port sichtbar sein (auch auswählen).

Danach schauen wir uns den bereits gezeigten oder durch **Datei -> Neu** erzeugten Sketch an: Anfangs ist hier nur ein **setup(){}** und **loop(){}** mit ein paar Kommentaren zu sehen (zwei Schrägstriche // bewirken, dass der dahinter stehende Text nicht ausgeführt wird).

Ein Programm wird dabei chronologisch ausgeführt, wobei das Setup einmal zu Beginn durchgeführt und der Teil innerhalb der loop-Schleife ständig wiederholt wird.

Da in beiden Fällen nichts drin steht, passiert nachdem das Programm auf den Controller geladen wurde (-> **Taste**): Nichts.

Das ist natürlich etwas langweilig. Deswegen schalten wir einmal die integrierte LED an. An Pin 13 des Controllers ist bei vielen Arduinos eine kleine SMD-LED auf dem Controller.

Dazu müssen wir Pin 13 als Ausgang definieren und diesen dann auf 5V legen, damit die LED leuchtet.

Da wir das nur einmal durchführen, ist dies am besten als Initialisierung in den geschweiften Klammern des Setups aufgehoben:

Mit **pinMode(13, OUTPUT);** wird der Pin als Ausgang festgelegt. Mit **digitalWrite(13, HIGH);** kann die LED angeschaltet werden.

Wie man sieht, sind die Kommandos sogar relativ gut verständlich – ein großer Vorteil gegenüber dadurch vertuschten Registermanipulationen.

Ein wichtiger Punkt sind die Semikolons am Ende, da diese gerne vergessen werden und damit Probleme auftauchen.

Digitale Ausgabe

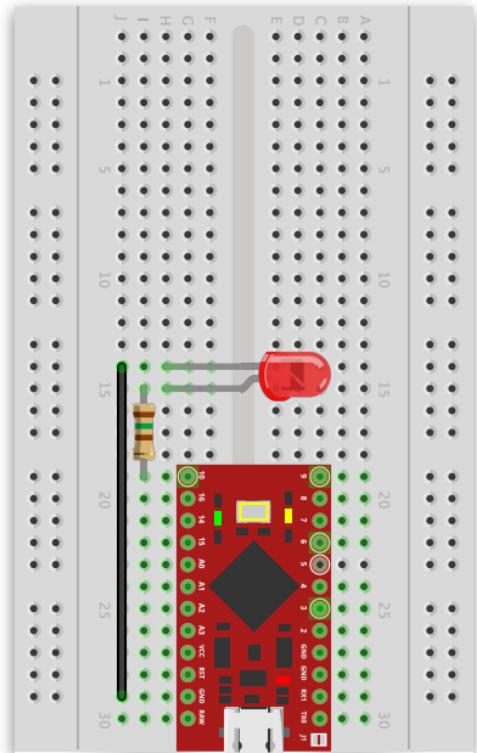
Beispiele

The screenshot shows the Arduino IDE interface with the title "Variablen | Arduino 1.8.10". The code editor contains the following sketch:

```
int led = 10; // Initialisieren einer Variablen

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

void loop() {
  // Hier machen wir immer noch nichts
}
```



Variablen

Beispiele

Natürlich ist die Verwendung nur der integrierten LED etwas langweilig. Man kann auch eine externe LED anschließen.

Dazu stecken wir den Arduino Micro auf ein Breadboard und verbinden ihn mit einem 150-Ohm-Widerstand an Pin 10 mit einer LED und dann über ein Kabel mit der Erde (Ground) des Controllers.

Der Widerstand ergibt sich dabei aus der maximalen Stromstärke von 20mA, den die LED verträgt, der LED-Spannung von (etwas über) 2V und den 5V als Betriebsspannung. Damit sollten über den Widerstand 3V abfallen (5V-2V). Mit $U = I \cdot R$ ergibt sich damit der Widerstand $R = 3V / 20mA = 150\Omega$.

Weiterhin ist die Polung der LED wichtig, da der Strom nur in eine Richtung fließen kann: Das lange Bein gehört an + (bzw. die abgeflachte Seite des Gehäuses an -). Ansonsten leuchtet sie einfach nicht.

Für die Verkabelung wird oft schwarz als GND und rot für 4V (VCC) genommen.

Wichtig hierbei ist: Die LED hängt jetzt an einem anderen Pin als vorher. Man kann natürlich jetzt alle 13 durch 10 ersetzen, was aber, wenn öfter etwas geändert wird, etwas lästig werden kann.

Stattdessen kann man Variablen verwenden, zum Beispiel noch vor dem Setup int led = 10; einsetzen und dann 13 durch led ersetzen. int ist dabei ein Zahlentyp (Integer), led der Variablenname, welcher hier mit 10 initialisiert wird.

Der Vorteil von Variablen ist, dass sie auch durch das Programm modifiziert werden können.

Eine andere Möglichkeit wäre eine #define led 10. Hier wird festgelegt, dass das Programm jede LED automatisch vor dem Hochladen des Programms durch 10 ersetzt.

Variablen

Beispiele

Verzoegerung | Arduino 1.8.10

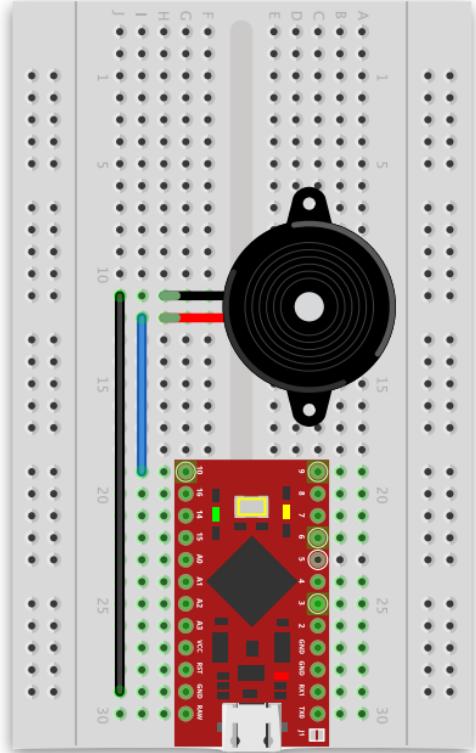
```
int buzzer = 10; // Initialisieren einer Variablen

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

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

Speichern abgeschlossen.
gefolgt von Buchstaben, Zahlen, Bindestrichen, Punkten und Unterstrichen.
Die maximale Länge beträgt 63 Zeichen.

12 Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



Verzögerung

Beispiele

Momentan ist der Mikrocontroller natürlich immer noch recht unnütz, immerhin könnte man die LED auch direkt an eine Spannungsquelle hängen. Deswegen füllen wir jetzt einmal die loop-Schleife und nutzen statt der LED einen Piezobuzzer.

Dazu müssen wir aber vorher noch eine neue Funktion kennenlernen. **delay(Millisekunden)** lässt den Controller eine Weile faul herumhängen und einfach mal nichts tun, mit den Millisekunden als die angestrebte Urlaubszeit. Des Weiteren können wir ja auch in **digitalWrite(led, HIGH)** das **HIGH (5V)** durch **LOW (0V)** ersetzen. In diesem Fall liegt an dem Buzzer an beiden Pins auf Erde (Ground, LOW, 0V) - da wird also nicht viel passieren ohne Spannungsdifferenz.

Wir schreiben also nun in geschweiften Klammern des loop():

```
digitalWrite(buzzer, HIGH); delay(1);
digitalWrite(buzzer, LOW); delay(1);
```

Damit wird abwechselnd der Buzzer ein- und ausgeschaltet und der Ton generiert.

Durch unterschiedliche Zeiten in den Delays kann dabei die Tonfrequenz angepasst werden.

Und wer zu faul zu tippen ist: In **Beispiel -> Basics -> Blink** gibt es das passende Programm als Beispiel. Dies führt zurück zu einem der wichtigsten Hinweise vom Anfang: So gut wie alles ist bereits einmal implementiert worden und kann genutzt und angepasst werden. Ähnliches gilt natürlich auch für die fortgeschrittenen Programme in Zukunft. Und ist es da nicht zu finden, findet man es im Internet.

Verzögerung

Beispiele

Schleifen | Arduino 1.8.10

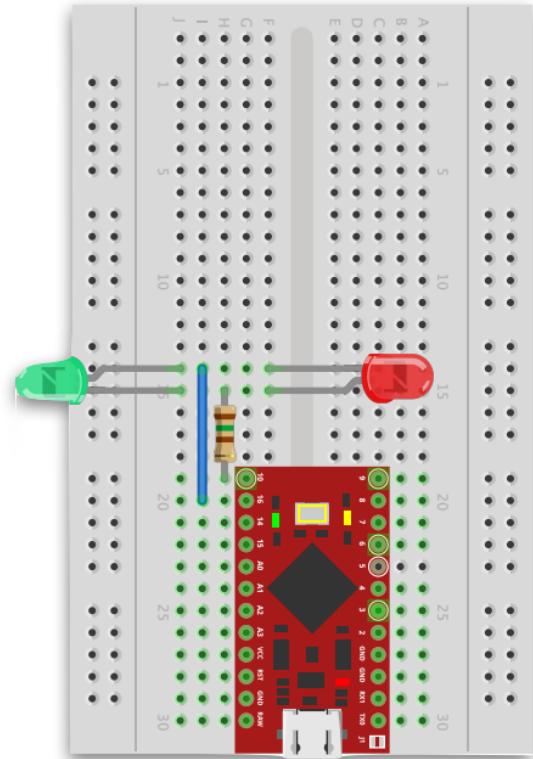
```
int led1 = 10;
int led2 = 16;

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  digitalWrite(led1, HIGH);
  digitalWrite(led2, HIGH);
}

void loop() {
  for(int i=0; i<10, i++) {
    digitalWrite(led1, LOW);
    delay(1000);
    digitalWrite(led2, HIGH);
    delay(1000);
  }
  digitalWrite(led1, HIGH);
  digitalWrite(led2, LOW);
  delay(1000);
}
```

Speichern abgeschlossen.
gefolgt von Buchstaben, Zahlen, Bindestrichen, Punkten und Unterstrichen.
Die maximale Länge beträgt 63 Zeichen.

19 Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



Schleifen: For, While

Beispiele

Eine blinkende LED für sich genommen ist noch immer nicht so interessant – solange es nicht auf eine besondere Frequenz ankommt, kann man auch einfach eine LED nutzen, die von sich aus schon blinkt (wo also eine kleine Elektronikschaltung schon integriert ist).

Daher machen wir es jetzt etwas komplizierter und nutzen zwei LEDs, die wir mit einer Schleife in einem etwas komplizierteren Leuchtmuster ansteuern wollen. Da wir ja schon wissen, dass die LED nur in einer Richtung leuchtet, schalten wir einfach eine zweite LED parallel, aber in einer unterschiedlichen Richtung (so dass das lange Bein der einen mit dem kurzen der anderen verbunden ist). Der Rückleiter geht auch nicht mehr auf GND, sondern an einen zweiten Pin. Werden damit beide Pins auf dasselbe Spannungslevel gelegt, leuchtet keine. Wenn beide auf unterschiedlichen liegen, leuchtet entweder die eine oder andere.

Einfaches Wechselblinken kann man also direkt wie mit der einzelnen LED programmieren. Hier wollen wir aber jede einzelne LED in einer bestimmten Anzahl aufblinken lassen, bevor die nächste blinkt. Dazu benutzen wir Schleifen. Im Grunde kennen wir schon eine: **loop** ist im Grunde eine while-Schleife, die für immer durchgeführt wird. In der regulären Variante wird in den runden Klammern die Bedingung eingeführt, bis zu welchem Ereignis die Schleife durchgeführt wird. Solange es wahr ist, geht sie immer weiter, **while(TRUE)** also für immer.

Eine andere Variante ist eine for-Schleife:
for(int i=0; i<10, i++) {... blink Programm...}
lässt damit eine LED zehnmal aufblinken. i ist dabei der Zähler, der als Integer mit 0 initialisiert wird, und mit **i++** (kurz für **i=i+1**) bei jeder Ausführung um **1** erhöht wird. **i<10** ist dabei die Bedingung, wie lange die Schleife ausgeführt wird.

Schleifen: For, While

Beispiele

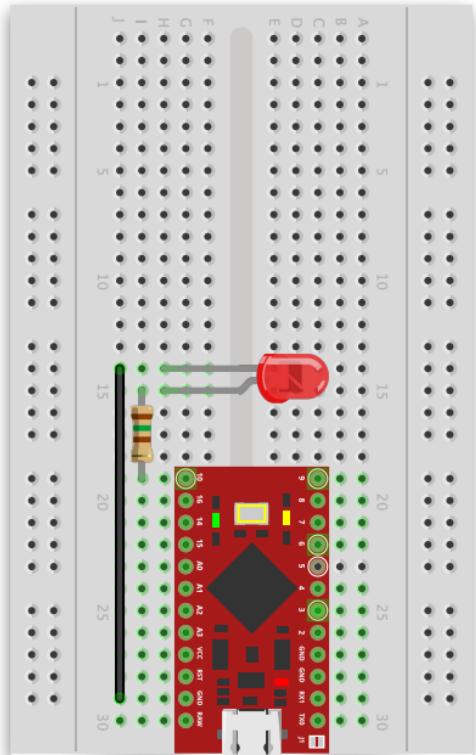
The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Analogue_Ausgabe | Arduino 1.8.10
- Toolbar:** Standard icons for file operations (New, Open, Save, Print, etc.) and a help icon.
- Sketch Name:** The current sketch is titled "Analogue_Ausgabe".
- Code Area:** The code is as follows:

```
int led = 10; // Initialisieren einer Variablen

void setup() {
  pinMode(led, OUTPUT);
  digitalWrite(Led, HIGH);
}

void loop() {
  for (int i=0, i<1023, i++) {
    analogWrite(Led, i);
    delay(100);
  }
  for (int i=1023, i>0, i-) {
    analogWrite(Led, i);
    delay(100);
  }
}
```
- Status Bar:** Speichern abgeschlossen.
gefolgt von Buchstaben, Zahlen, Bindestrichen, Punkten und Unterstrichen
Die maximale Länge beträgt 63 Zeichen.
- Bottom Progress Bar:** A horizontal progress bar indicating the status of the build process.



Analoge Ausgabe

Beispiele

Pulsierendes Licht

Beim Herumspielen mit den Delays könnte es euch schon aufgefallen sein, dass bei sehr kurzen Zeiten das Blinken gar nicht mehr zu sehen ist und sich stattdessen die Helligkeit der LED ändert. Damit habt ihr schon die Grundlagen der sogenannten Pulsweitenmodulation als (pseudo-)analoger Ausgang herausgefunden: Durch sehr schnelles Umschalten zwischen 0V und 5V wird im Durchschnitt ein Analogwert zwischen diesen Werten erzeugt, der sich aus dem Verhältnis zwischen Ein- und Ausschaltzeit ergibt.

Anstatt dies manuell tun zu müssen, wird dies netterweise auch hardwaremäßig unterstützt. Bei den zum Beispiel beim Arduino durch das ~-Symbol gekennzeichneten Pins kann man anstelle des **digitalWrite()** auch ein **analogWrite(pin, Wert);** verwenden.

Da Pin 10 zu den analogfähigen Pins gehört, können wir einfach frühere LED-Schaltung benutzen und müssen nur den Programmcode ändern. So wird einfach anstelle des **digitalWrite(led, HIGH)** ein **analogWrite(led, value)** verwendet, wobei **value** zu Beginn als Integerwert zwischen **0** und **1023** initialisiert wird, z.B. mit **int value = 300;**

Dies kann dann auch mit der for-Schleife kombiniert werden:

```
for (int i=0, i<1023, i++) {  
    analogWrite(led, i);  
    delay(100);  
}  
for (int i=1023, i>0, i--) {  
    analogWrite(led, i);  
    delay(100);  
}
```

Damit wird die LED abwechselnd langsam heller und dann wieder dunkler werden.

Analoge Ausgabe

Beispiele

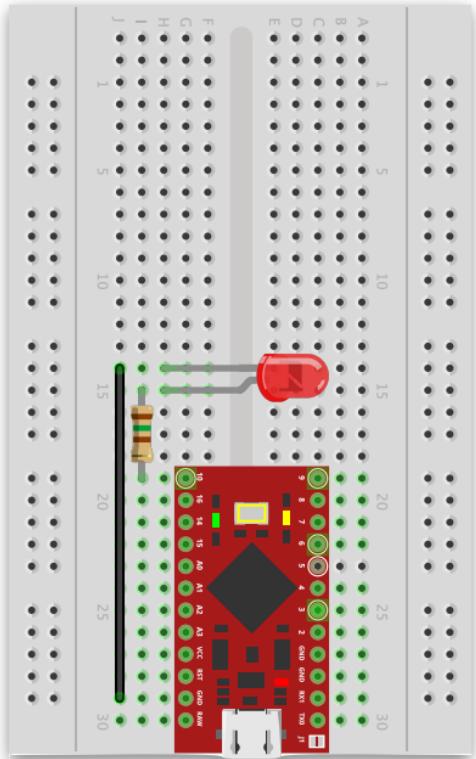
Logik | Arduino 1.8.10

```
Int led = 10;
bool state = true;
long timer = 0;
long maxTime = 1000000;

void setup() {
  pinMode(led, OUTPUT);
  digitalWrite(led, state);
}

void loop() {
  timer++;
  if(timer > maxTime) {
    state = !state;
    digitalWrite(led, state);
    timer = 0;
  }
}
```

1 Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



Logik: If, Bool

Beispiele

Im Andenken an Mr. Spock: Logik

Um dem Rechner etwas Intelligenz beizubringen, müssen wir ihm etwas Logik zum Entscheiden beibringen.

Zum einen brauchen wir dazu eine sogenannte boolesche Wahrheitsvariable:

```
bool logikVariable = 0;
```

Diese kann entweder **0 (oder False)** oder **1 (True)** werden. Für die richtige Logik benutzen wir dann noch eine Wenn-Dann-Schleife, durch die wir je nach Ergebnis wahr oder falsch unterschiedliche Funktionen ausführen:

```
if(a){tue etwas}  
else if(b){...tue etwas anderes}  
else {...ansonsten mache...}
```

In die geschweiften Klammern kommt somit unser ganz normaler Programmcode, während in die runden Klammern die Wenn-Bedingungen kommen. Dies kann einfach eine boolesche Variable sein (ein ! davor invertiert diese) oder ein Vergleich.

Wir können eine boolesche Variable aber auch ganz einfach mit anderen vergleichen und abfragen, ob sie grösser (>), kleiner (<) oder gleich (==, um es vom Zuweisungsoperator = zu unterscheiden) sind.

Hier bauen wir uns unser eigenes Delay: Wir zählen stupide eine Zahl hoch; wenn sie dann größer als ein bestimmter Wert wird, schalten wir die LED um und fangen mit dem Zählen von vorne an. Im Grunde programmieren wir hier ein energieverschwendendes Däumchen-drehen für den Mikrocontroller.

Logik: If, Bool

Beispiele

Digitale_Eingabe | Arduino 1.8.10

```
Digitale_Eingabe
int led = 10;
bool state = false;

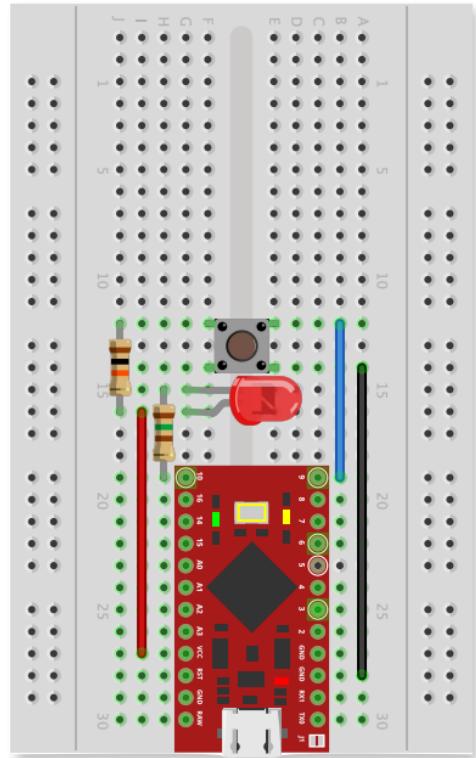
void setup() {
  pinMode(button, INPUT);
  pinMode(led, OUTPUT);
  digitalWrite(led, HIGH);
}

void loop() {
  if(digitalRead(taster)==0) {
    state = !state;
    delay(100);
    digitalWrite(led, state);
}
}

Speichern abgeschlossen.

gefolgt von Buchstaben, Zahlen, Bindestrichen, Punkten und Unterstrichen.
Die maximale Länge beträgt 63 Zeichen.
```

15 Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



Digitale Eingabe

Beispiele

Gehorcht auf Knopfdruck

Als Nächstes lernen wir die Eingabemöglichkeiten des Mikrocontrollers kennen. Hierzu erweitern wir unsere LED-Schaltung um einen Taster, der an einen anderen Pin des Controllers angeschlossen wird. Zu beachten ist dabei, dass jeweils zwei Pins des hier verwendeten Tasters schon intern miteinander verbunden sind. Werden dann alle Pins per Tastendruck elektrisch verbunden, in der dargestellten Schaltung ist dann Pin 9 mit GND verbunden. Der 10-k Ω -Widerstand an Taster und Pin 9 sorgt dabei dafür, dass auch im geöffneten Zustand ein definierter Spannungspegel (5V) vorliegt. Würde dies nicht gemacht, würde irgendeine zufällige Spannung anliegen und damit keine klare Fallunterscheidung zwischen gedrückt und geöffnet möglich. Der Widerstand begrenzt dabei den Strom im gedrückten Zustand.

Bei 10 k Ω ist dieser $I = U/R = 5V/10k\Omega = 0,5$ mA; würde man stattdessen direkt mit 5V verbinden, wird es unangenehm warm: $>5V/0\Omega$...

Vom Programm her setzen wir für unseren Schaltkreis zunächst den entsprechenden Pin mit **pinMode(taster, INPUT);** als Eingang, innerhalb der loop-Schleife kann dieser dann mit **digitalRead(taster);** abgefragt werden.

Zum Ein- und Ausschalten der LED brauchen wir jetzt noch eine Zustandsvariable:

bool state = false;

Diese schalten wir bei jedem Tastendruck um. Dazu wird bei jedem Kontakt die Zustandsvariable mittels ! invertiert. Allerdings gibt es während des Tastendrückens kurz eine Menge Wackelkontakte - das sogenannte Prellen. Man macht daher besser noch eine kleine Pause, bevor man die LED umschaltet:

```
if(digitalRead(taster)==0){  
    state = !state;  
    delay(100);  
    digitalWrite(led, state);  
}
```

Digitale Eingabe

Beispiele

Analoge_Eingabe | Arduino 1.8.10

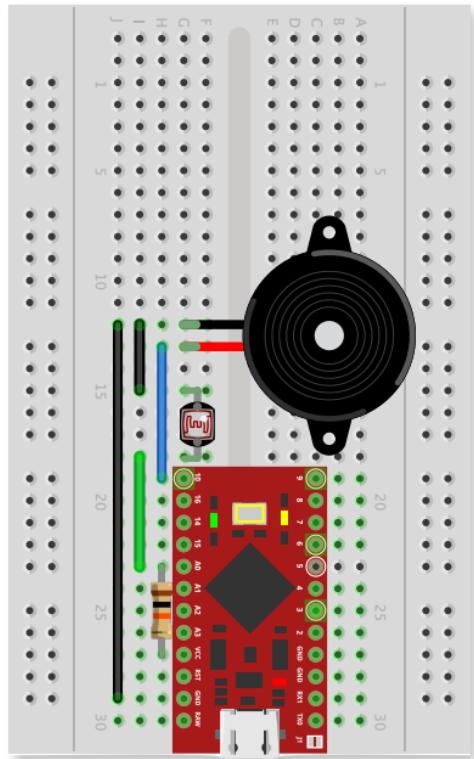
```
Analoge_Eingabe
int buzzer = 10;
int ldr = A0;

void setup() {
  pinMode(ldr, INPUT);
  pinMode(buzzer, OUTPUT);
}

void loop() {
  digitalWrite(buzzer,TRUE);
  delayMicroseconds(analogRead(ldr));
  digitalWrite(buzzer, FALSE);
  delayMicroseconds(1024-analogRead(ldr));
}
```

Speichern abgeschlossen.
gefolgt von Buchstaben, Zahlen, Bindestrichen, Punkten und Unterstrichen
Die maximale Länge beträgt 63 Zeichen.

2 Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



Analoge Eingabe

Beispiele

Theremin

Genauso wie es zur digitalen Ausgabe eine Eingabe gibt, existiert auch zur analogen Ausgabe eine entsprechende analoge Eingabe.

Der Aufbau ist natürlich fast genauso:

pinMode(analogPin, INPUT); würde einen Eingang deklarieren (was man sich sogar sparen könnte, es sieht aber hübscher aus, wenn man es trotzdem benutzt).

analogRead(analogPin); gibt dann einen Integerwert zwischen **0** und **1023** zurück: Üblicherweise sind beim Arduino die mit Präfix A durchnummerierten Pins analoge Eingabepins. (Als digitaler Ein- und Ausgang kann aber jeder genutzt werden). Die Vergleichsspannung kann dabei durch den AREF-Pin vorgegeben werden. Standardmäßig ist 5V (die Versorgungsspannung) die Referenzspannung, die Analogspannung ist Integerwert/1023 * 5V.

Durch **analogReference(EXTERNAL)** wird zum Beispiel die an AREF anliegende Spannung genutzt. Eine andere Referenzspannung kann zum Beispiel **INTERNAL** sein (1,1V).

In unserem Beispiel haben wir also wieder unseren Buzzer an einem Analogausgang, dessen Lautstärke wir durch einen Photowiderstand (LDR) einstellen:

analogWrite(buzzer, analogRead(ldr));

Für ein richtiges Theremin müsste allerdings nicht die Lautstärke, sondern die Frequenz verändert werden. Somit würde es eher passen, ein extrem kurzes Delay durch den Analogwert zu verändern:

digitalWrite(buzzer, TRUE);

delayMicroseconds(analogRead(ldr));

digitalWrite(buzzer, FALSE);

delayMicroseconds(1024-analogRead(ldr));

Das sollte reichen, um andere Personen in den Wahnsinn zu treiben.

Analoge Eingabe

Beispiele

Seriell | Arduino 1.8.10

Serial

```
int button = 9;
int ldr = A0;
int led = 13;

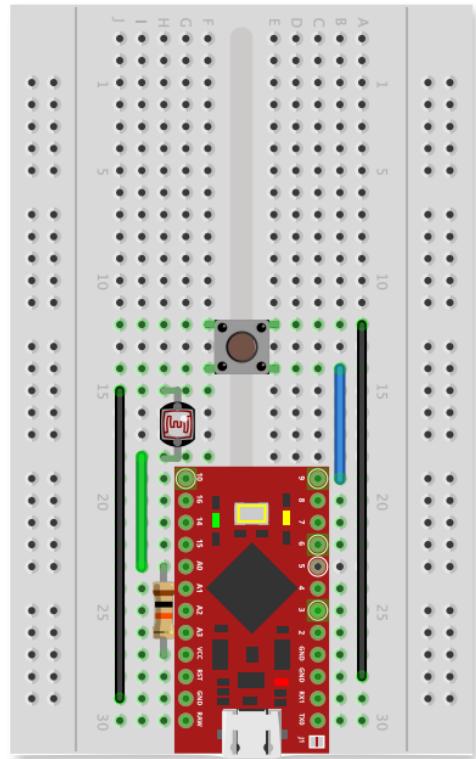
void setup() {
  pinMode(ldr, INPUT);
  pinMode(button, INPUT);
  pinMode(led, OUTPUT);

  Serial.begin(9600);
  Serial.println("Start");
}

void loop() {
  Serial.print("Button: ");
  Serial.print(digitalRead(button));
  Serial.print(", LDR: ");
  Serial.println(analogRead(ldr));
  if (Serial.available()) {
    analogWrite(led, Serial.read());
  }
  delay(100);
}
```

Speichern abgeschlossen.
gefolgt von Buchstaben, Zahlen, Bindestrichen, Punkten und Unterstrichen
Die maximale Länge beträgt 63 Zeichen.

18 Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



Kommunikation: Seriell

Beispiele

Es wäre natürlich schön, wenn Informationen nicht nur auf einem Controller bleiben würden, sondern dieser mit anderen Systemen kommunizieren könnte.

Eine einfache Möglichkeit dazu ist die serielle Schnittstelle, die über USB mit dem Computer kommunizieren kann. Im Grunde nutzt man diese ja bereits zum Programmieren des Arduino.

Soll dieser damit Daten ausgeben, so muss man zunächst einmal die serielle Schnittstelle mit **Serial.begin(9600);** initialisieren. Die **9600** sind dabei die Baudrate, wie viele Bits pro Sekunde übertragen werden sollen.

Mittels **Serial.print(Daten);** können dann Daten übertragen werden. Dies können zum Beispiel die Werte des Tasters oder des Fotowiderstands sein, aber auch reiner Text, gekennzeichnet durch Anführungsstriche vor und nach dem Text. Zur besseren Strukturierung kann man einen Zeilenumbruch nutzen.

Dies kann entweder durch ein Kommando \n im Text oder auch durch Ändern des **Serial.print(...)** in **Serial.println(...)** geschehen.

Was der Arduino sendet, kann man sich dann entweder als Text im seriellen Monitor oder grafisch im seriellen Plotter anschauen (unter Werkzeuge).

Es geht aber auch anders herum:

Mit **Serial.read();** bekommt man zum Arduino gesendete Bytes abgefragt. Da man ja nicht weiß, ob überhaupt Daten gesendet wurden, schreibt man meist noch davor:

if (Serial.available()) byte = Serial.read();

So schreibt man nur, wenn auch Daten vorhanden sind, diese in den (zum Beispiel) int byte.

Kommunikation: Seriell

Beispiele

simple | Arduino 1.8.10

```
// NeoPixel Ring simple sketch (c) 2013 Shae Erisson
// released under the GPLv3 license to match the rest of the Adafruit

#include <Adafruit_NeoPixel.h>
#ifndef __AVR__
    #include <avr/power.h>
#endif

// Which pin on the Arduino is connected to the NeoPixels?
// On a Trinket or Gemma we suggest changing this to 1
#define PIN          10

// How many NeoPixels are attached to the Arduino?
#define NUMPIXELS    16

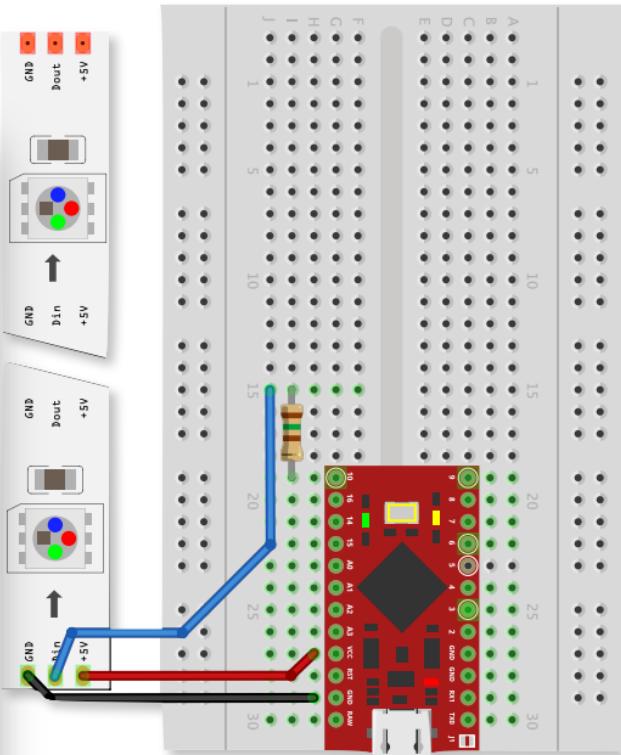
// When we setup the NeoPixel library, we tell it how many pixels, and
// Note that for older NeoPixel strips you might need to change the 1
// example for more information on possible values.
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB

int delayval = 500; // delay for half a second

void setup() {
    // This is for Trinket 5V 16MHz, you can remove these three lines if
    // you're using a Leonardo or Gemma
    Serial.begin(9600);
    pixels.begin();
    pixels.setBrightness(100);
}
```

10

Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



WS2812

Beispiele

Es wird bunt: Die intelligente LED

Natürlich braucht man nicht nur mit dem Computer zu kommunizieren, sondern kann dies auch mit anderen Controllern. So existieren zum Beispiel auch intelligente LEDs (WS2812). Hier nimmt ein kleiner Controller in der LED eine Datenkommunikation vor. Er empfängt einen Datenfluss vom Mikrocontroller, indem einfach immer weiter rote, gelbe und blaue Farbwerte (RGB) hintereinander gesendet werden. Der integrierte Controller schnappt sich drei, zeigt sie an und gibt den Rest an die LEDs dahinter über den Datenausgang weiter. Man braucht also nur eine Signalleitung und 5V/GND um eine Menge LEDs anzusteuern. Dies ist von der Verkabelung natürlich einfacher als jede LED einzeln zu verdrahten (und eine RGB-LED besteht ja schon aus drei Einzel-LEDs - mit der Lupe kann man diese und den Controller sogar innerhalb der WS2812 sehen).

Natürlich muss der Controller dazu das richtige Kommunikationsprotokoll kennen. Netterweise können wir wie beim Arduino üblich auf bereits vorhandenes Wissen zurückgreifen: Es gibt zwei relativ populäre Libraries dazu: Neopixel und FastLED.

Unter Bibliotheken verwalten kann man diese installieren. Anschließend sind diese auch unter Beispiele zu finden. Schaut man sich das simple-Beispiel der Neopixel-Library an, so sieht man:

- Ganz am Anfang wird mit **#include <Adafruit_NeoPixel.h>** die Library eingebunden (das Programm kennt sie also).
- Mit ein paar **#define** werden dann ein paar (nicht variable) Grundwerte eingefügt.
- **Adafruit_NeoPixel pixels = Adafruit_NeoPixel(...);** initialisierte ein Objekt namens pixels mit bestimmten Werten.
- **pixels.begin();** initialisiert die LEDs.
- Mit **pixels.setPixelColor(...);** und **pixels.show();** kann man dann einzelne Farben setzen.

WS2812

Beispiele

ColorPalette | Arduino 1.8.10

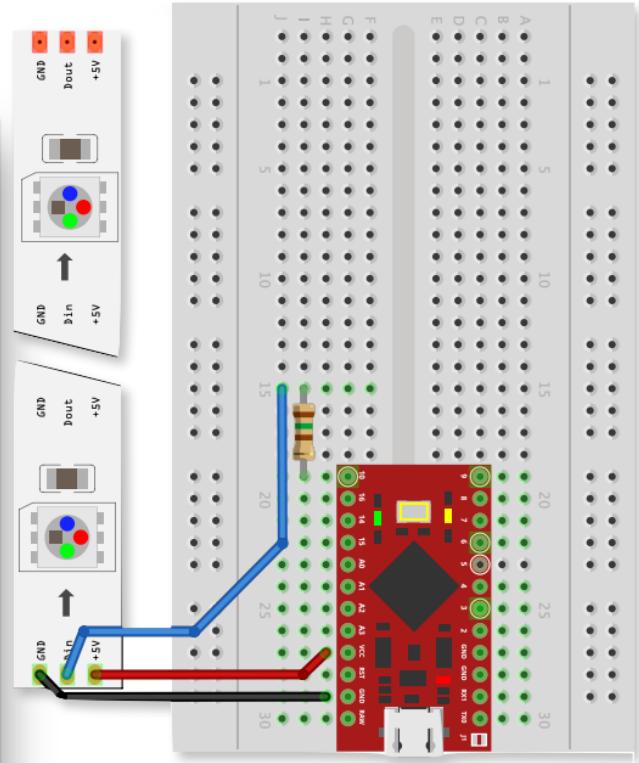
```
ColorPalette §
}

// This function fills the palette with totally random colors.
void SetupTotallyRandomPalette()
{
    for( int i = 0; i < 16; i++) {
        currentPalette[i] = CHSV( random8(), 255, random8());
    }
}

// This function sets up a palette of black and white stripes,
// using code. Since the palette is effectively an array of
// sixteen CRGB colors, the various fill_* functions can be used
// to set them up.
void SetupBlackAndWhiteStripedPalette()
{
    // 'black out' all 16 palette entries...
    fill_solid( currentPalette, 16, CRGB::Black);
    // and set every fourth one to white.
    currentPalette[0] = CRGB::White;
    currentPalette[4] = CRGB::White;
    currentPalette[8] = CRGB::White;
    currentPalette[12] = CRGB::White;
```

3

Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



Arrays

Beispiele

Bei einer Menge LEDs wird es natürlich irgendwann etwas kompliziert, wenn man jeden Wert und jede Variable einzeln abspeichert.

Man kann sie stattdessen gut zu sogenannten Arrays zusammenfassen.

Ein **int leds[7]** würde zum Beispiel ein Array von 7 Werten namens leds ergeben. Einzelne Werte können dann mit **leds[i]** abgefragt oder mit **leds[i] = x** auf einen bestimmten Wert gesetzt werden.

Dabei ist zu beachten, dass i die Position/Nummer innerhalb des Arrays angibt und diese mit 0 startet.

Damit kann man über eine for-Schleife beispielsweise die LEDs viel besser ansteuern.

Dann kann man beispielsweise bei einer Menge LEDs diese in einer for-Schleife ansteuern:

```
for (int i=0; i<numberOfLEDs, i++) {  
    digitalWrite(leds[i], HIGH);  
}
```

Aber da wir ja für viele LEDs die intelligenten nutzen, brauchen wir das dafür nicht so dringend (diese werden ohnehin einfach mit ihren Nummern durchgezählt). Aber man kann in einem Array beispielsweise die anzuzeigenden Farben festlegen. So ähnlich werden im Colorpalette-Beispiel der FastLED-Library die einzelnen Farbpaletten hergestellt und abgerufen.

Abseits davon sieht man in diesen Beispielen auch etwas anderes: Mit **Rückgabewert Funktionsname (int Eingabewerte) {...}** werden Unterfunktionen definiert.

```
int meineSumme(int wert1, init wert2){  
    return wert1+wert2;}
```

Dies gibt die Summe von zwei Werten zurück. So brauchen oft genutzte Funktionen nur einmal geschrieben zu werden und können gut wiederverwertet werden.

Arrays

Beispiele

Bluetooth | Arduino 1.8.10

Bluetooth §

```
#include <Adafruit_NeoPixel.h>
#include <SoftwareSerial.h>

int pixel = 10;
int rgb[] = {0,0,0};
Adafruit_NeoPixel strip = Adafruit_NeoPixel(pixel, 9, NEO_GRB + NEO_I
SoftwareSerial mySerial(2, 3); // RX, TX

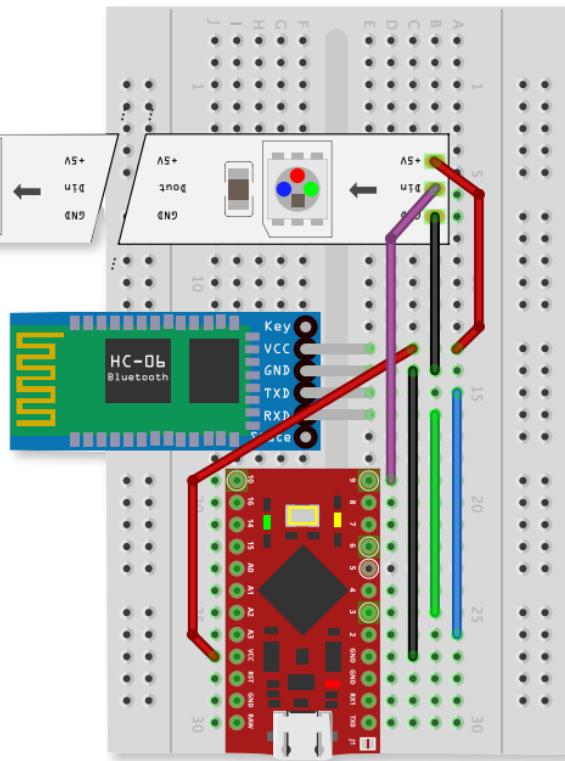
void setup() {
  strip.begin();
  mySerial.begin(4800);
}

void loop() {
  if (mySerial.available()){
    rgb[0]=mySerial.read();
    for(int i=0;i<pixel;i++){
      strip.setPixelColor(i, pixels.Color(rgb[0],rgb[1],rgb[2]));
      strip.show();
    }
    delay(100);
}
}
```

Kompilieren abgeschlossen.

Der Sketch verwendet 6796 Bytes (23%) des Programmsspeicherplatzes. Da
Globale Variablen verwenden 297 Bytes (11%) des dynamischen Speichers

16 Arduino Leonardo auf /dev/cu.SLAB_USARTtoUART



Kommunikation: Bluetooth

Beispiele

Fernsteuerung

Auf Dauer sind Taster und die kabelgebundene Verbindung zum Computer ja etwas almodisch und wir wissen ja alle, dass alles besser mit Bluetooth ist. Dazu benutzen wir das relativ günstige HC 06-Modul. Dieses wir einfach mit der Spannungsversorgung verbunden und dazu auch mit der seriellen Schnittstelle (Pin0 und Pin1). Dieses läuft dann parallel zu der Verbindung zum Computer.

Eine andere Möglichkeit ist es, die SoftwareSerial-Library zu verwenden, um jeden beliebigen Pin nutzen zu können (und die serielle Schnittstelle der Hardware weiter zum Debugging mit dem Computer nutzen zu können). Dazu fügen wir am Programmanfang Folgendes ein:

```
#include <SoftwareSerial.h>
```

Danach können wir bei dem hier verwendeten Arduino Micro die Pins 8, 9, 10, 11, 14, 15, 16 auch als zusätzliche serielle Ports nutzen.

Dazu wird zunächst ein SoftwareSerial **addonSerial(pin1, pin2);** erzeugt. Der Name ist natürlich relativ beliebig und muss nicht addonSerial heißen. Danach werden sowohl in der setup-Routine als auch in der loop-Schleife die Standardkommandos der seriellen Schnittstelle der Hardware genutzt, nur dass Serial durch den eigenen Namen ersetzt wird, also um Beispiel:

```
addonSerial.begin(9600); und so weiter...
```

Danach kann man seine Daten über Bluetooth übertragen. Dazu muss gegebenenfalls am Rechner noch in der Bluetooth-Konfiguration das HC 06-Modul ausgewählt und eine Verbindung hergestellt werden. Das dazu benötigte Passwort lautet übrigens 1234.

Danach kann man ganz normal in der Arduino-IDE unter **ports** die entsprechende Bluetooth-Schnittstelle auswählen und wie gehabt den seriellen Monitor oder Plotter nutzen.

Kommunikation: Bluetooth

Beispiele

LightballController | Processing 3.5.4

Java ▾

```
LightballController
```

```
30
31 void setup() {
32   size(displayWidth, displayHeight, P3D);
33   smooth();
34   background(255);
35   translate(140, 0);
36   for (int i=0; i< Serial.list().length; i++) {
37     if (Serial.list()[i].length() > 17) {
38       if(Serial.list()[i].substring(0,18).equals("/dev/cu.HC-06-DevB")) {
39         portNumber = i;
40       }
41     }
42   }
43   println(Serial.list());
44
45   String portName = Serial.list()[portNumber];
46   myPort = new Serial(this, portName, 9600);
47
48   myPort.write("POWER9\r\n");
49
50   for (int i=0; i<6; i++) {
51     int offset = (i<3) ? 0 : height/15;
52     slider[i].setWidth(width/4);
53     slider[i].setHeight(height/40);
54     slider[i].setX(width*2/3);
55     slider[i].setY((i+0.5+1)*height/15+offset);
56     slider[i].setColor(sliderColor[i%3]);
57     slider[i].makeValue();
58   }
59 }
```

Konsole Fehler Updates 3



Processing

Beispiele

Die GUI der Fernbedienung

Für den Rechner gibt es ein Programm namens Processing. Wenn es herunter geladen und gestartet ist, stellt man fest, dass es der Arduino-IDE sehr ähnlich sieht. Tatsächlich haben die Macher der Arduino-IDE Processing als Vorbild und Grundlage genutzt. Daher kann es auch prima mit dem Arduino kommunizieren. Probieren wir einmal aus:

```
import processing.serial.*;
```

```
for (int i=0; i< Serial.list().length; i++) {  
    print(i + " : ");  
    println(Serial.list()[i]);  
}
```

Dies gibt die Liste der seriellen Ports aus (die man auch in der Arduino-IDE sieht). Damit haben wir die Nummer i unserer Bluetooth-Schnittstelle.

Besser, aber etwas komplizierter, ist ein Vergleich des Listeneintrags mit einem entsprechenden Teilstring:

```
if(Serial.list()[i].substring(0,17).equals("//  
dev/cu.usbmodem1")) oder so ähnlich.
```

Danach merkt man sich den Namen des seriellen Ports und nutzt ihn mit:

```
Serial myPort;
```

```
String portName = Serial.list()[portNumber];  
myPort = new Serial(this, portName, 9600);
```

Ab da können wir über diese serielle Schnittstelle Daten senden und empfangen, und damit unseren Mikrocontroller über Funk fernsteuern. Da Processing selber schon ein Thema für sich ist, nutzen wir hier einfach eins der Beispiele - unter **Topics -> GUI -> Buttons** und fügen hier die serielle Schnittstelle ein sowie unter **mousePressed()** jeweils ein

```
Serial.print("N");
```

Wird die linke Maustaste also über einer der beiden Figuren gedrückt, sendet Processing ein Signal an den Arduino. Empfängt dieser etwas **Serial.available()**), kann man damit die LEDs ein- und ausschalten.

Processing

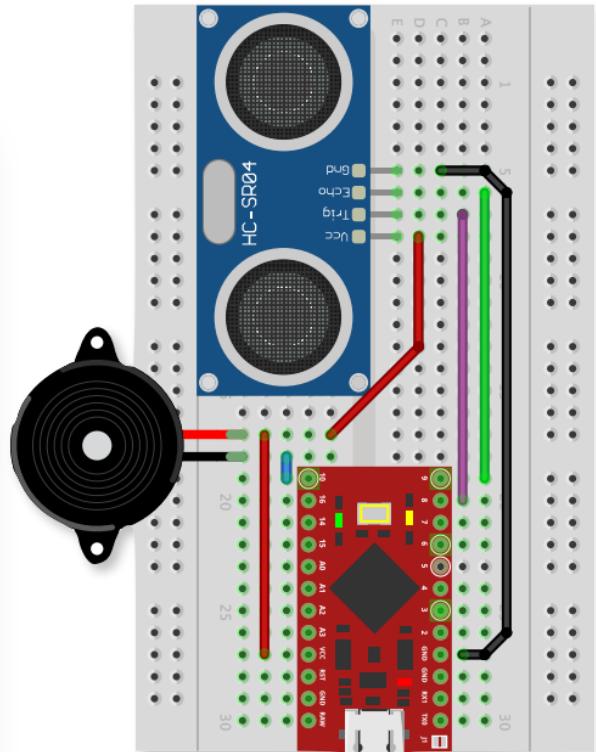
Beispiele

```
int trigger=8;
int echo=9;
int buzzer = 10;

void setup() {
  pinMode(trigger, OUTPUT);
  pinMode(echo, INPUT);
  pinMode(buzzer, OUTPUT);
}

void loop() {
  digitalWrite(trigger, LOW);
  delay(5);
  digitalWrite(trigger, HIGH);
  delay(10);
  digitalWrite(trigger, LOW);
  tone[buzzer, pulseIn(echo, HIGH) * 0.034/2];
}
```

Speichern abgeschlossen.



Ultraschallabstandssensor

Beispiele

Theremin Nr 2: Jetzt mit Abstand

In unserem ersten Theremin-Bau haben wir ja nicht ganz korrekt die Abschaltung des Photowiderstands zum Verändern der Frequenz bzw. der Lautstärke genutzt. Korrekterweise nutzt man dafür aber den Abstand. Hier lernen wir einen Abstandssensor kennen. Dieser sendet einen Ultraschallimpuls aus und misst die Zeit bis zur Rückkehr. So etwas Ähnliches gibt es auch optisch als Time-of-Flight-Sensor als Modul zu kaufen, welcher allerdings etwas teurer ist.

Hier wird durch den Trigger-Pin ein Impuls ausgesendet und die Antwort am Echo-Pin gemessen. Der Trigger-Pin wird daher als Ausgang, Echo-Pin als Eingang konfiguriert. Dann wird der Trigger-Pin normalerweise auf GND gelegt und nur zur Messung kurz für `delayMicroseconds(10);` auf **HIGH** gelegt. Danach kann die Impulslänge an dem Echo-Pin ausgelesen werden.

Diese kann durch den `pulseIn`-Befehl ausgeleren werden, wobei als Parameter der auszulösende Pin und der Zustand, dessen Dauer bestimmt werden soll, angegeben werden.

Der Abstand wird dann berechnet mit:

`pulseIn(echoPin, HIGH)*0.034/2;`

Die Hälfte verwendet man aufgrund des hin und wieder Zurücklaufens des Ultraschallimpulses, welcher damit die doppelte Distanz zurücklegt. Außerdem breitet er sich mit Schallgeschwindigkeit aus, also mit 340 m/s.

Damit können wir wieder wie gehabt den Buzzer ansteuern und mit dem Abstand zur Hand über dem Distanzsensor den Ton variieren. Oder wir nutzen mal wieder eine weitere Vereinfachung: **`tone(pin, frequenz)`** gibt einen Ton in der gewünschten Frequenz aus.

Ultraschallabstandssensor

Beispiele

HID | Arduino 1.8.10

HID

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include "Mouse.h"

Adafruit_MPU6050 mpu;

void setup(void) {
  mpu.begin();
  Mouse.begin();
}

void loop() {
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

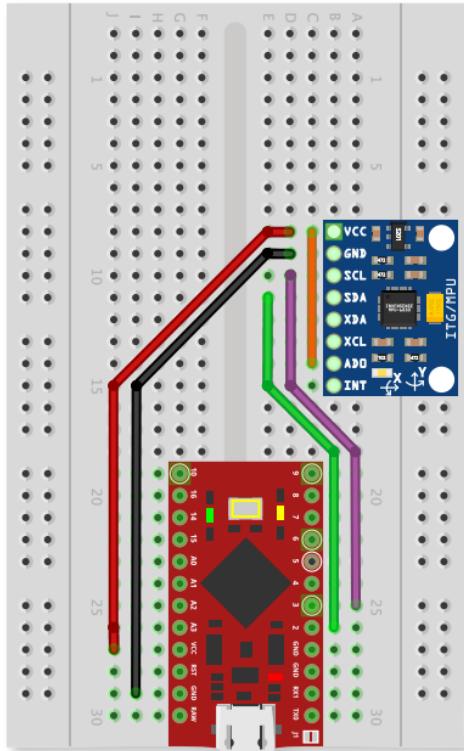
  Mouse.move(g.gyro.x, g.gyro.y, 0);

  delay(100);
}
```

Speichern abgeschlossen.

Der Sketch verwendet 13344 Bytes (46%) des Programmspeicherplatzes. Die Globale Variablen verwenden 558 Bytes (21%) des dynamischen Speichers

12 Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



I2C - Lagesensor, HID

Beispiele

Wir programmieren ein 3D-Eingabegerät

Eine nette zusätzliche Eigenschaft des Arduino Micro bzw. genauer des Atmega 32U4 ist, dass man diesen als HID-Gerät am Computer, also als Maus- oder Tastatursatz verwenden kann. Damit ist der Weg zum eigenen kabellosen Eingabegerät sehr kurz - Taster und Bluetooth kennen wir ja schon.

Dazu wird im einfachsten Fall eine Tastatur oder Maus simuliert. Hier werden wir die Maus nutzen:

Mit **Mouse.begin();** wird die Mausfunktionalität gestartet. Danach kann man mit **Mouse.move(x,y, wheel);** die zwei Richtungen und ein Mausrad zum Computer senden, **Mouse.click();** sendet einen Mausklick, wie der Name schon vermuten lässt. Für letzteres wäre ein Taster ausreichend, für das Rad könnte man den Photowiderstand als veränderbaren Widerstand benutzen.

Es gibt eine Vielzahl an Sensoren und anderen Erweiterungen, die mit einem sogenannten I2C-Protokoll kommunizieren. Jedes Modul hat dabei eine bestimmte Adresse, unter dem es von seinem Master, unserem Mikrocontroller, erreicht werden kann. Hier nutzen wir einen einfachen Lagesensor GY-521 bzw. MPU6050. Netterweise gibt es auch da wieder die passenden Libraries, beispielsweise von Adafruit: Im Beispiel **basicreadings** werden die notwendigen Dateien (Wire.h ist dabei die I2C-Kommunikation) eingebunden:

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
Adafruit_MPU6050 mpu; erzeugt dann das
passende Objekt, das mit mpu.begin(); ge-
startet wird. Dann holen wir uns die Daten, die
wir mit Mouse.move(g.gyro.x, g.gyro.y,
g.gyro.z); zur Maussteuerung verwenden
können, hiermit:
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);
```

I2C - Lagesensor, HID

Beispiele

Umwelt | Arduino 1.8.10

```
Umwelt

#include <OneWire.h>
#include <DS18B20.h>

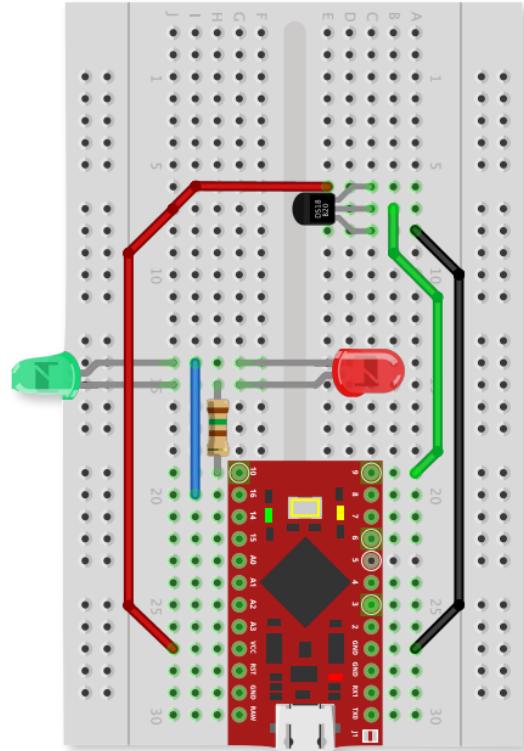
OneWire oneWire(9);
DS18B20 sensor(&oneWire);

void setup(void) {
  pinMode(10, OUTPUT);
  pinMode(16, OUTPUT);
  sensor.begin();
}

void loop(void) {
  sensor.requestTemperatures();
  if (sensor.getTempCByIndex(0) > maxTemperatur) {
    digitalWrite (16, LOW);
    digitalWrite (10, HIGH);
  } else {
    digitalWrite (16, HIGH);
    digitalWrite (10, LOW);
  }
}

Speichern abgeschlossen.
```

18 Arduino Leonardo auf /dev/cu.SLAB_USBtoUART



I2C - Umwelt

Beispiele

Temperaturwarner

Ein weiteres Kommunikationsprotokoll nutzt nur eine Datenleitung: 1-Wire. Dieses wird zum Beispiel in einem Temperatursensor wie dem hier enthaltenen DS18B20 genutzt. Das Vorgehen ist wie immer gleich: Zunächst suchen und installieren wir die passenden Libraries, in diesem Fall **OneWire.h**, für das Kommunikationsprotokoll und dann noch Dallas-Temperature.h für den konkreten Sensor. Anschließend schaut man sich die Beispiele zu letzterer an und sucht sich ein möglichst einfaches heraus. In diesem Fall klingt das Beispiel **Simple** gut.

Hier werden die Temperaturdaten über die serielle Verbindung an den PC geschickt, was man sich daher prima mit dem seriellen Monitor/Plotter mal eine Weile anschauen kann. Danach passen wir das Programm einfach an unsere Bedürfnisse an, hier also an die schon vorher einmal genutzten zwei LEDs, jetzt als Temperaturwarner.

Beim Überschreiten einer bestimmten Temperatur soll die rote LED leuchten, sonst die grüne. Wie gesagt, man kann immer wieder Code recyceln. Wir kopieren die Initialisierung der LEDs (Pins als Ausgang, Startwerte) einfach aus dem alten Beispiel und steuern sie dann den Temperaturwerten entsprechend an:

```
If (sensor.getTempCByIndex(0) >
maxTemperatur) {
    digitalWrite (16, LOW);
    digitalWrite (10, HIGH);
} else {
    digitalWrite (16, HIGH);
    digitalWrite (10, LOW);
}
```

Einen passenden Wert für **maxTemperatur** kann man sich vorher aus den mit dem seriellen Monitor angeschauten Daten aussuchen.

Beispiele

Servo | Arduino 1.8.10

Servo

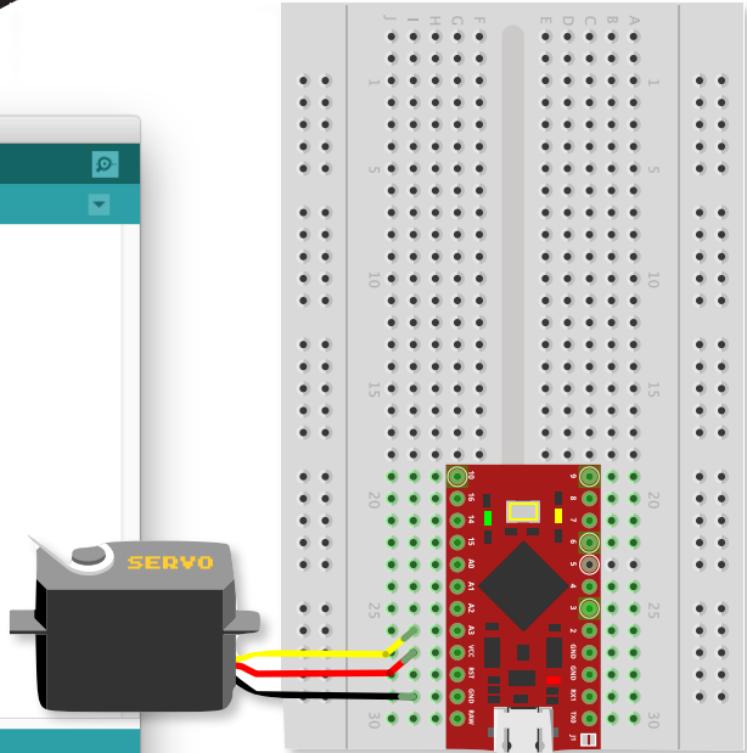
```
#include <Servo.h>

Servo myservo;

void setup() {
  myservo.attach(A3);
}

void loop() {
  myservo.write(((millis()/1000)%60)*3);
  delay(10);
}
```

Speichern abgeschlossen.



Servomotor, Zeit

Beispiele

Bewegung ist gesund

Von den unterschiedlichen Motoren nutzen wir hier einen Servomotor. Wenn man ihn aufschraubt, merkt man, dass er aus einem kleinen Gleichstrommotor zusammen mit einem veränderbaren Widerstand (Potentiometer) zur Positionserkennung und einem Controller (also dem, was wir in letzter Zeit gelernt haben##dem Inhalt dieser Lernkarten##) in einem kleinen Gehäuse besteht. Auch hier gibt's wieder die passende Library und Beispiele, allerdings schon direkt mit der IDE installiert: Unter Beispiel findet man direkt einen Unterpunkt **Servo** mit zwei Beispielen, mit denen man herumspielen kann.

In unserem Fall wollen wir am Ende eine Art Uhr bauen. Dazu müssen wir natürlich irgendwie die Zeit messen. Delays kennen wir ja schon, hier werden wir aber **millis();** nutzen. Damit bekommen wir die Zeit in Millisekunden seit Programmstart des Arduino.

Nun ist zu überlegen, ob unser Servo als Uhrzeiger Stunden, Minuten oder Sekunden anzeigen soll: Zunächst wandeln wir die **millis();** in einen passenden Wert um, indem wir sie durch einen entsprechenden Faktor teilen. Dummerweise wird **millis();** ja nur durch einen Neustart zurückgesetzt, sonst wird immer weiter hoch gezählt. Glücklicherweise gibt es dazu einen modulo-Operator %, der den Rest beim Teilen angibt.

$(\text{millis()}/1000)\%60$ gibt die Sekunden an, wobei nach je sechzig Sekunden wieder die 0 durch den Modulooperator erreicht wird.

Da das Servo zum Beispiel zwischen 0 und 180 Grad angesteuert wird, brauchen wir das Ergebnis also nur mit 3 zu multiplizieren; unser Sekundenzeiger wird so dargestellt:

```
myservo.write(((millis()/1000)\%60)*3);
```

Servomotor, Zeit

Ausblick



... es gibt zu fast allem passende Module

Ausblick

Hier haben wir nur einen kleinen ersten Einblick in die Möglichkeiten gegeben, wichtig ist es, neugierig zu bleiben und das Wissen in der Welt zu nutzen und zu etwas Neuem zu kombinieren. Es gibt zu allen Bereichen die passenden Elektronikmodule und auch Software, die man benutzen kann.

Ein paar Stichworte ohne Anspruch auf Vollständigkeit, was es noch so alles gibt (für die Internetsuche):

- SD-Leser
- MP3-Module
- Verstärker, Lautsprecher
- GPS
- Staubsensor
- Time-of-Flight-Abstandssensor
- Farbsensor
- Spektrum
- Real Time Clock
- WLAN, Ethernet
- Elektrolumineszenz
- Kamera

- TFT-, OLED-Displays jeder Größe
- Touchscreen
- E-Paper
- Text- oder Grafik-LCD-Displays
- Thermodrucker
- Relais
- Leistungselektronik
- Biometrische Sensoren
- Gassensoren
- Kompass
- Schrittmotoren und Motortreiber
- Brushless-Motoren
- Elektromagnete
- Flip-Dot-Displays
- Röhren
- RFID
-

... es gibt zu fast allem passende Module

Die Lernkarten-Reihe

Bombini Verlags GmbH
Kaiserstraße 235
53113 Bonn
www.bombini-verlag.de

Die Lernkarten im Bombini-Verlag

Die Lernkarten-Reihe im Bombini-Verlag behandelt die wichtigsten Werkzeuge einer modernen Maker-Werkstatt: der 3D-Drucker, der Lasercutter, der 3D-Scanner, der Mikrocontroller, die Fräse und die Nähmaschine.

Jedes Lernkarten-Set vermittelt vierfarbig die Grundkenntnisse im Umgang mit diesen digitalen Werkzeugen. Neben Basics werden praxisnahe Tipps und Tricks beschrieben. Attraktive Praxisbeispiele geben Anregungen für eigene Projekte.



9 783946 496175



9 783946 496182



9 783946 496199

Mikrocontroller

Mikrocontroller dienen als Gehirn von Bastelprojekten. Das Lernkartenset führt in die Grundlagen dieser kleinen und kostengünstigen programmierbaren Rechner ein. So wird gezeigt wie eine Vielzahl von Sensoren und Aktuatoren angeschlossen werden können, außerdem die Grundlagen der Kommunikation, ob nun mit Peripherie oder mit einem grossen Rechner. Dies geschieht mit der Arduino IDE und einem Pro Micro Modul mit einem ATMEGA 32U4 und verschwenden Sensoren und Aktuatoren. Mit den hier vermittelten Grundlagen stehen eigenen Projekten nichts mehr im Wege.

Das Lernkarten-Set *Laserschneiden* besteht aus 28 doppelseitigen, farbigen Lernkarten. Jede Lernkarte behandelt ein abgeschlossenes Thema.



Erasmus+

Dieses Lernkartenset wurde im Rahmen des Erasmus+ Projekts „Maker+“ erstellt und mit freundlicher Unterstützung des Bundesministeriums für Bildung und Forschung.



9 783946 496199

ISBN 978-3-946496-19-9
12,95 € [D]
www.bombini-verlag.de

