Der Arduino im Physikunterricht

Cheat-Sheet

Christopher Kommetter

Der Arduino - Cheat-Sheet

Was ist ein Arduino?

Um den Einstieg in die Mikrokontrollerprogrammierung zu vereinfachen, wurde das Arduino-Projekt ins Leben gerufen. Es gibt fertige Platinen, die sogenannten Arduino-Boards oder einfach Arduino, sowie eine eigene Software zum Programmieren, die Entwicklungsumgebung. (Vgl. Schernich 2014)

Das Board besteht aus mehreren digitalen und analogen Ein- und Ausgängen (I/O Ports). Programmiert wird der Arduino in der Sprache C bzw. C++. Das Arduino-Board wird in verschiedenen Versionen angeboten, diese unterscheiden sich hauptsächlich in der Anzahl der Ports und der Bauform. Am gängigsten ist der so genannte Arduino UNO. Dazu gibt es bereits viele verschiedene Klone, die komplett ident und kompatibel zum originalen UNO und Zubehör sind.

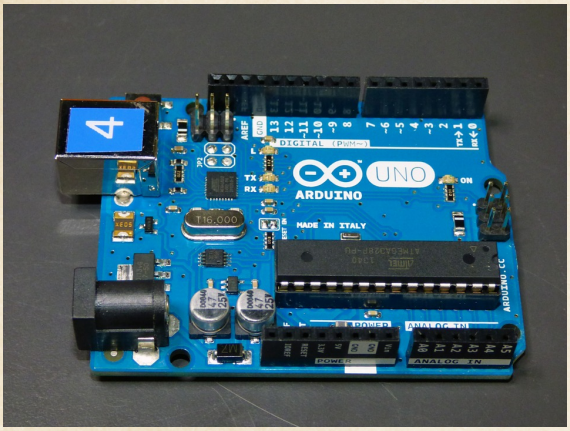

Abbildung 1: Ein Arduino UNO Board

Mikrocontroller

Ein `Mikrocontroller`, oft auch als `Mikroprozessor` oder μC bezeichnet, ist ein Mikrochip der alle Komponenten, wie den Arbeitsspeicher und Prozessor, bereits integriert hat.

Pin's

Der Arduino hat eine Reihe von Ein- und Ausgängen, den so genannten Pin's. Diese sind in Gruppen angeordnet: die digitalen Ein- und Ausgänge, die analogen Eingänge, Power-Pin's zur Spannungsversorgung und Schnittstellen für die Programmierung.

Digital vs. Analog

Ein `analoges Signal` wird durch eine kontinuierliche Funktion beschrieben, beispielsweise die Höhe einer Quecksilbersäule eines Thermometers: hier entspricht eine bestimmte Höhe genau einem Temperaturwert. `Digitale Werte` werden durch Zeichen dargestellt. Analoge Signale werden mithilfe eines AD-Wandlers in digitale umgewandelt, damit ein Computer sie verarbeiten kann. Die Genauigkeit der Umwandlung nennt man die Auflösung.

Analoge Eingänge

Der Arduino UNO verfügt über 6 analoge Eingänge. Im Gegensatz zu den digitalen PIN's, sind die analogen PIN's NUR Eingänge und können nicht als Ausgänge genutzt werden. Legt man an einem dieser Eingänge ein Signal (eine Spannung) an, wandelt der Arduino dies in ein digitales Signal um (AD - Wandler). Der Spannungsbereich ist auf 0 - 5 V beschränkt, möchte man dennoch größere Spannungen messen, muss ein Spannungsteiler vorgeschaltet werden. Die Auflösung der AD-Wandlung beträgt 10 Bit - das entspricht hier einer Genauigkeit von 0,005 V - das heißt der Arduino kann zwischen 3,01 und 3,02 V unterscheiden, nicht aber zwischen 3,0101 V und 3,0109 V.

Diverse Modelle neben dem UNO

Boards	Besonderheiten
Arduino Mega	54 digitale I/O's
Arduino Zero	32-Bit-μC
Arduino Mini	ATmega328 in kleiner Bauform
Arduino MKR1000	Arduino Zero mit WLAN

Quelle: Brühlmann 2017

Technische Daten

Modell: Arduino UNO

Prozessor: ATmega328
Taktfrequenz: 16 MHz
Betriebsspannung: 6–20 VDC

Speicher:	Flash	SRAM	EEPROM
	32 KB	2 KB	1 KB

digitale Ein-/Ausgänge: 14
Analoge Eingänge: 6
Auflösung analoge Eingänge: 10 Bit

Arduino-Clones. Neben den originalen Arduino-Boards existiert eine große Anzahl an Arduino kompatiblen Boards. Die technischen Daten sind dabei größten Teils ident.

Shields

Shields sind Erweiterungsplatinen, die direkt auf das Arduino-Board gesteckt werden können und dessen Funktionsumfang erweitern. zB.:

LCD Shield. Dies existiert in verschiedenen Formen, wie etwa einem 2 zweizeiligen LCD-Display mit Tasten oder als Farb-Touchscreen - Damit lassen sich Ausgaben und Interaktionen direkt am Arduino durchführen.

Ethernetshield. Damit wird dem Arduino die Anbindung an ein Netzwerk (gibt es auch als WLAN-Version) ermöglicht, um beispielsweise Messwerte direkt ins Internet zu übertragen.

Programmierung des Arduino's

Die Arduino Plattform stellt neben den Board's auch eine Entwicklungsumgebung zur Verfügung.

Die IDE

Als Integrierte Entwicklungsumgebung (kurz: **IDE** - *integrated development environment*) wird ein Programm, bzw. eine Sammlung von Programmen, bezeichnet, mithilfe derer Softwareentwickler programmieren. Die IDE's stellen den Entwicklern viele nützliche Tools (Werkzeuge) zur Verfügung.

Zum Arduino-Board (die Hardware) gehört auch die IDE zum Arduino-Projekt. Die IDE erlaubt das Erstellen, Testen (Debuggen) und Hochladen des erstellten Programms auf den Arduino. Die Entwicklungsumgebung (IDE) ist für alle gängigen Betriebssysteme verfügbar und steht kostenlos zum Download zur verfügung: **https://www.arduino.cc/en/Main/Software**. (Vgl. Brühlmann 2017)

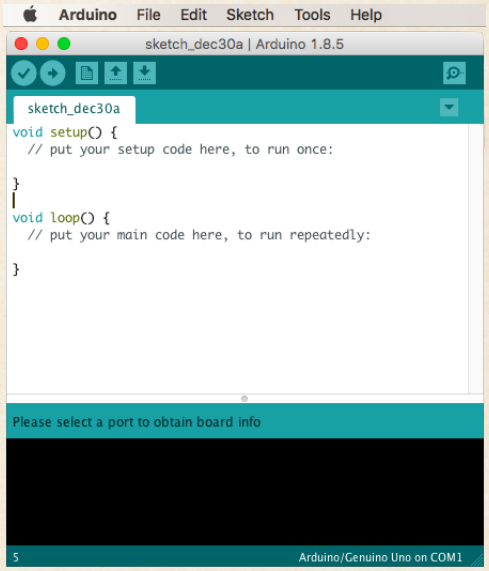

Abbildung 2: Arduino IDE unter MacOS X

Die Programmiersprache C

Jedes Arduino-Programm besteht mindestens aus den beiden Funktionen `setup` und `loop`.

Listing 1: Programmstruktur

```
1 void setup() {
2    // wird nur 1 mal ausgefuehrt
3 }
4
5 void loop() {
6
7 }
```

Diese beiden Funktionen werden von der IDE automatisch erstellt. In der ersten Funktion (**`void`** `setup()`) werden Grundeinstellungen (zB. ob ein Kanal ein In- oder Output ist) definiert und es können **Bibliotheken** eingebunden werden. Diese Funktion wird nur 1 mal ausgeführt, nämlich sobald der Arduino das erste Mal mit Strom versorgt wird.

Die **`void`** `loop()` Funktion wird ständig

wiederholt. In diesen Block schreibt man das eigentliche Programm.

Bibliotheken

Eine `Bibliothek` ist eine Ansammlung von Funktionen. Diese ermöglichen es, dass komplizierte und immer wiederkehrende Abläufe nicht immer neu geschrieben werden müssen. Viele Arduino-Shields und Sensoren bringen solche Bibliotheken mit und ermöglichen die einfache Benutzung dieser.

Infos zum Listing 1

Zeile	Bedeutung
1,5	**void**: diese Funktion gibt keinen Wert zurück
1,5	Die Klammern `()` bilden die so genannte Parameterliste, hier kann man der Funktion Werte übergeben
1,3,5,7	Die geschwungenen Klammern bilden den Funktionsblock: innerhalb dieser werden alle Argumente geschrieben
2	ein Kommentar: dies hat keine Auswirkungen auf das Programm

Variablen

Um Werte zu speichern und um sie bearbeiten zu können, benötigen wir Variablen. Variablen sind Speicherplätze, denen man einen beliebigen Namen zuweisen kann. Der Arduino muss aber wissen, um welche Art von Werten es sich handelt, dafür gibt es mehrere verschiedene Datentypen.

Variablendatentypen

Typ	Beschreibung
int	ganzzahlige Zahlen von -32.768 bis 32.767
long	ganze Zahlen, größer als 32.767, zB. um Millisekunden zu stoppen
float	Kommazahlen
char	Buchstaben

Hier ein kurzes Beispiel, wie man mit Variablen in der Programmiersprache *C* arbeitet:

Listing 2: verschiedene Variablen

```
1 int eineZahl = 123;
2 float komma = 3.14;
3 char buchstabe = 'c';
4
5 int andereZahl = eineZahl * 5;
6 int nochEine = andereZahl + 3;
7 //die Variable nochEine hat nun den
8 //Wert 618
```

Das Semikolon ;

Jedes Argument, außer Kommentare und Funktionsköpfe (zB. **void** `setup()`), werden mit einem Strichpunkt, den so genannten Semikolon, abgeschlossen

Arbeiten mit den digitalen Pin's

Damit der Arduino weiß, ob ein Kanal (Pin) als Eingang oder Ausgang genutzt wird, muss man dies in der Funktion **void** `setup()` für jeden genutzten Pin festlegen. Dies gilt nur für die digitalen Pin's, analoge müssen nicht definiert werden, da sie nur als Eingänge genutzt werden können.

Listing 3: Definition der Ein- und Ausgänge

```
1 pinMode(7, OUTPUT); //definiert PIN 7
2                     //als Ausgang
3 pinMode(2, INPUT);  //PIN 2 als Eingang
```

PWM-Pin's

Die analogen Pin's des Arduino's können im Gegensatz zu den digitalen nur als Eingänge genutzt werden. Nutzt man die digitalen Pin's als Ausgänge, kann man diese entweder EIN (1) oder AUS (0) schalten. Dies würde einer Spannung von +5V (EIN) oder 0V (AUS) entsprechen. Möchte man nun aber eine andere Spannung (oder eben ein analoges Signal) ausgeben, nutzt man dazu die PWM-Pin's. Diese Ausgänge sind in der Lage, neben 0V und 5V, auch alle Spannungen inzwischen (zB. +4V) auszugeben. Die digitalen Pin's 3, 5, 6, 9, 10, 11 sind dazu in der Lage.

Mithilfe des Befehls `digitalWrite(Pin,Wert)` schaltet man einen, zuvor mit `pinMode()` als *OUTPUT* definierten, digitalen Kanal auf HIGH (+5V) oder LOW (0V). ZB.: `digitalWrite(7,HIGH);` schaltet +5V auf den Pin mit der Nummer 7. An die als PWM bezeichneten digitalen Pin's können auch beliebige Spannungen geschalten werden: `analogWrite(3,200);` legt an den digitalen Pin 3 eine Spannung von 4 Volt. Die Spannung wird als digitaler Wert zwischen 0 (0V) und 255 (+5V) angegeben. 200 entspricht dabei +4V.

Was ist eigentlich ein `Listing`? Als Listing wird ein Auszug aus einem Quellcode (auch Sourcecode genannt) bezeichnet. Der Quellcode ist wie ein Kochbuch: er schreibt den Computer, in unserem Fall der Arduino, vor, welche Schritte er nach der Reihe ausführen soll.

Digitale Eingänge können beispielsweise genutzt werden um festzustellen ob ein Taster gedrückt wird. Digitale Inputs können nur zwischen 2 Werten unterscheiden: HIGH (+5V) und LOW (0V) - es kann also nur festgestellt werden, ob eine Spannung anliegt oder nicht. Nicht festgestellt werden kann, welche Spannung genau anliegt - dafür nutzt man die analogen Eingänge. Der Befehl `digitalRead(2);` liefert HIGH oder LOW an dem zuvor als INPUT festgelegten digitalen Pin mit der Nummer 2.

Spannungen messen mithilfe der analogen Pin's

Im Gegensatz zu den den digitalen Pin's müssen wir die analogen nicht als INPUT definieren. Man kann auf diese direkt zugreifen. Die Eingänge sind von A0 bis A5 nummeriert.

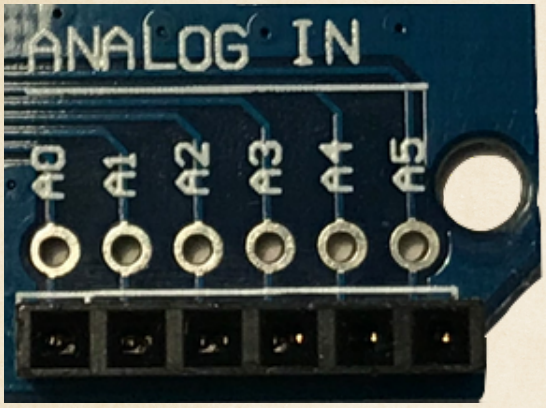

Abbildung 3: Die 6 analogen PIN's eines Arduino UNO Klons

Der Arduino benötigt zirka 100 Mikrosekunden (0.0001 s) um einen analogen Wert zu lesen, also kann man 10000 Werte pro Sekunde messen. (*Arduino Projekt* 2017)

Mithilfe des Befehls `analogRead(1);` liest man den Wert des Pin's A1 aus. Diese Funktion gibt einen Wert zwischen 0 (entspricht 0 Volt) und 1023 (entspricht +5V) zurück. Diese Zahl muss nun in Volt umgerechnet werden:

Listing 4: Umrechnen der analogen Werte in Spannungswerte

```
1 int wert;
2 float volt;
3 wert = analogRead(1);
4
5 //Umwandeln des Datenwertes (0-1023)
6 //in lesbare Spannungswerte (0 - 5 V):
7 volt = wert * (5.0 / 1024.0);
```

Wie kann ich etwas sehen?

Am Einfachsten lässt man sich alles am PC darstellen, dazu muss aber der Arduino ständig per USB mit dem Rechner verbunden sein. Die Arduino IDE bietet dafür den so genannten *Serial Monitor*. Möchte man mit

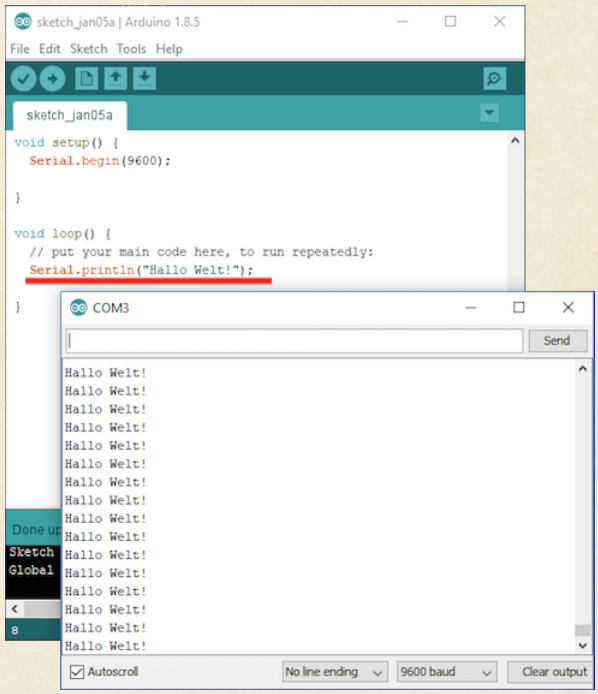

Abbildung 4: Ausgabe am Seriellen Monitor

dem PC kommunizieren, ist es erforderlich die Schnittstelle zu *initialisieren*. Das heißt man muss diese aktivieren (in der Funktion **void** `setup()`, da dies nur 1 mal notwendig ist).

`Serial.begin(9600);` die Zahl 9600 definiert die *Baudrate*, die Übertragungsgeschwindigkeit

Zum Ausgeben verwendet man eine der 2 Funktionen: `Serial.print();` bzw. `Serial.println();` (diese macht einen Zeilenumbruch [ENTER] nach der Ausgabe).

VERSCHIEDENE TYPEN AUSGEBEN

Ausdruck	Beschreibung
`Serial.println(123);`	gibt die Zahl 123 aus
`Serial.println("Hallo");`	gibt den Text *Hallo* aus
`Serial.println(volt);`	gibt den Wert, der in der Variable *volt* gespeichert ist, aus

Das erste Programm

Das erste Programm soll eine LED blinken lassen. Oder als **Algorithmus** ausgedrückt: Der Arduino soll eine LED einschalten, dann warten, danach die LED ausschalten und wieder warten. Diesen Vorgang soll der Arduino unendlich oft wiederholen.

Abbildung 5: Eine LED

Der Aufbau

Für den schnellen Aufbau elektronischer Schaltungen benutzt man oft **Steckplatinen**. Hier können elektronische Bauteile direkt in ein Lochraster eingesteckt werden, ohne dass man löten muss. Bei solch Platinen sind die Kontaktreihen quer miteinander verbunden (siehe Markierung in Abbildung 7)

Die am Arduino als **GND** bezeichneten Pin's stellen die Masse dar, also den Minuspol. Eine LED darf nicht direkt am Arduino angeschlossen werden, es muss zwingend ein Widerstand in Serie geschalten werden, um den Strom zu begrenzen. Ansonsten könnte die LED und der Arduino zerstört werden. Idealerweise sollte ein 220Ω Widerstand eingesetzt werden. Der Widerstand kann auch einen größeren Wert haben. Die LED wird auch bei einem 1kΩ (kilo Ohm = 1000Ω) noch leuchten, allerdings etwas schwächer.

Abbildung 6: Sammlung verschiedener Widerstände

Die Anode (Pluspol) der LED wird mit einen der digitalen Pin's verbunden (mit einem Widerstand inzwischen), die Kathode an einem der GND-Pin's.

LED IN WELCHE RICHTUNG ANSCHLIESSEN?

Die beiden Anschlussdrähte einer LED sind unterschiedlich lang. Der Längere von beiden ist der Pluspol, die Anode, der Kürzere die Kathode. Einfach zu merken: das Pluszeichen hat einen Strich mehr als das Minuszeichen und macht damit den Draht etwas länger. Außerdem: die meisten LED's sind auf der Minus-Seite etwas abgeflacht, wie ein Minuszeichen. *Kathode = kurz = Kante*

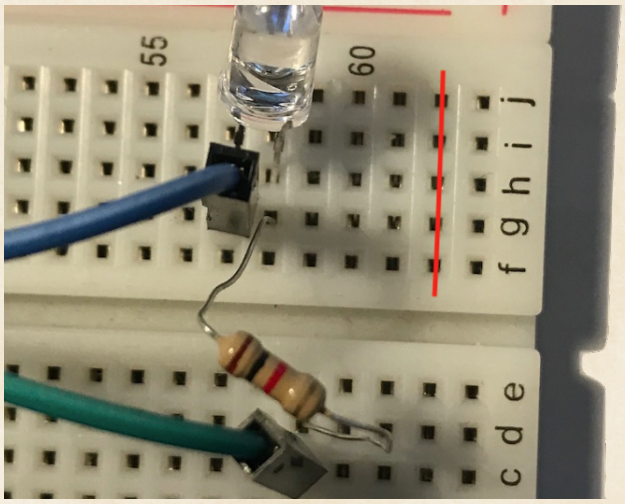

Abbildung 7: Komponenten auf Steckplatine

Der Quellcode

Wenn alles fertig aufgebaut wurde, geht es zur Programmierung mithilfe der IDE. In diesem Beispiel wurde die LED am Pin 13 angeschlossen.

Listing 5: LED blinken lassen

```
1  void setup() {
2    //Pin 13 als OUTPUT definieren
3    pinMode(13, OUTPUT);
4  }
5
6  void loop() {
7    digitalWrite(13, HIGH); // LED ein
8    delay(1000);  //1 Sekunde warten
9    digitalWrite(13, LOW);  // LED aus
10   delay(1000);
11 }
```

Der Befehl `delay(1000);` in Zeile 8 und 10 haltet den Ablauf für 1 Sekunde an, bevor die LED wieder ausgeschaltet wird. Die Zahl `1000` gibt die Zeit in Millisekunden an.

Kommentare werden durch `//` eingeleitet und haben keinerlei Auswirkung auf das Programm, diese dienen nur der Beschreibung.

Hochladen des Programms

Um den Quellcode auf den Arduino zu laden, muss dieser per USB mit dem PC verbunden sein. Die IDE übersetzt (kompiliert) den Quellcode in Maschinensprache, die der Mikrocontroller versteht und lädt dies direkt auf den Arduino. Sobald der Upload abgeschlossen ist, beginnt der Arduino mit der Ausführung des Programms. Gibt es im Quellcode Fehler, so schreibt die IDE im unteren Bereich des Fensters, dass es Fehler gibt und markiert im Quellcode auch die entsprechende Stelle.

Abbildung 8: durch klick auf Upload wird der gesamte Prozess in die Wege geleitet

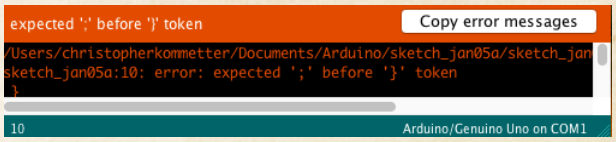

Abbildung 9: Der Quellcode konnte nicht kompiliert werden: ein Strichpunkt wurde vergessen

Bonus: Die dimmbare LED

Die LED soll nun nicht mehr blinken (EIN, Warten, AUS, Warten,...), sondern sie soll immer heller werden und wenn sie am Hellsten leuchtet, soll sie wieder immer dunkler werden. Dieser Vorgang soll unendlich oft wiederholt werden.

Tipp: es muss die Spannung kontinuierlich erhöht/verringert werden. Welche Pin's sind dazu im Stande?

Listing 6: if-Block (Wenn Abfrage)

```
1  if(wert == 123) {
2        wert = 0;
3  }
```

Listing 6 zeigt einen Bedingungsblock (if-Block). Wenn der Ausdruck in den Klammern erfüllt wird, werden alle Argumente im Block ausgeführt. In unserem Beispiel: Wenn die Variable `wert` den Wert 123 hat, wird die Variable wieder auf 0 gesetzt.

Literatur

Arduino Projekt (2017). URL: https://arduino.cc.

Brühlmann, Thomas (2017). *Sensoren im Einsatz mit Arduino*. ISBN-13: 978-3-95845-152-0. mitp Verlag.

Schernich, Erik (2014). *Arduino für Kids*. ISBN-13: 978-3-8266-9470-7. mitp Verlag.