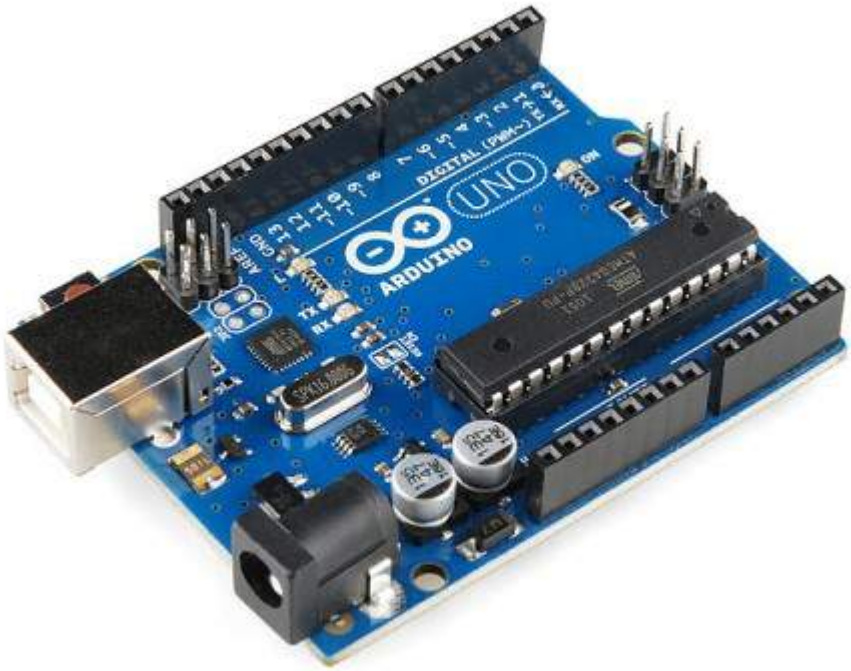
So einfach kann programmieren sein!

Arduino Uno Tutorial für die erste Benutzung



von
Christian Sommer
Heinrich-Emanuel-Merck-Schule

Die Ziele...

1. **Was brauchen wir?**
2. **Was ist ein Arduino? - Begriffserklärung**
3. **Programmieren?! - Grundlagen für Einsteiger**
4. **Beispiele für Programmierungen**
5. **Die Arduino (Programmier-) Software erklärt**
6. **Ein Arduino in Betrieb nehmen**
7. **Programme übertragen und selbst schreiben**

Der Anfang...

wirkt schwerer, als er wirklich ist. Diese Anleitung soll helfen und vor allem zeigen, dass man selbst mit wenig Erfahrung in Sachen Programmieren und Technik ganz interessante Dinge hinbekommen kann. Man braucht daher genau genommen wenig Vorkenntnisse, um mit Arduino arbeiten zu können. Jedoch sollte man schon ein wenig Zeit dafür zur Verfügung stellen. Der Spaß daran darf natürlich auch nicht fehlen.
Keine Angst davor, etwas neues auszuprobieren. So kompliziert und gefährlich ist es nicht. Eigentlich kann man auch nichts kaputt machen. Einfach genau das Tutorial lesen und dann wird schon nichts schief gehen.

1. Was braucht man...

- Ein Arduino Mikrocontrollerboard (Tutorial benutzt das Arduino Uno Rev. 3,
  Hier erhältlich: http://www.amazon.de/dp/B008GRTSV6)
- USB Verbindungskabel A auf B Kabel (Druckerkabel, USB auf abgeschrägt quadratisch,
  Hier erhältlich: http://www.amazon.de/dp/B0013F8R8I)
- Laptop oder PC mit USB-Anschluss (Für die Programmierung)
- Arduino-Software (Kostenlos verfügbar auf: http://www.arduino.cc bei „Download“)
- Ein paar LEDs, ein 220 Ohm Vorwiderstand und evtl. ein 10K Ohm Potentiometer
- Freiwillig (für spätere Versuche): LCD Anzeige
  (Hier erhältlich: http://www.amazon.de/dp/B009GEPZRE)
- Empfehlenswert für jeden, der gerne mit Arduinos mehr machen will. Ein Set mit einer Steckplatine (Hier erhältlich: http://www.amazon.de/dp/B0051QHPJM)

2. Was ist ein Arduino…

In diesem Tutorial werden öfters die Begriffe „Arduino“ oder „Mikrocontroller“ fallen. Dieser Lernschritt ist dafür da, dass man sich mit diesen Begriffen vertraut machen kann.

Ein **„Mikrocontroller“** ist im Prinzip ein kleiner Prozessor. Für Leser, die darunter nichts verstehen, kann man Mikrocontroller auch als Computer bezeichnen. Der enthaltene Chip (ein kleiner Baustein mit vielen Beinchen) hat analoge und digitale Ein- und Ausgänge. Er besitzt zusätzlich einen kleinen Timer. Das Wichtigste ist der kleine Speicher, den er besitzt. Dort werden die Programmierungen abgespeichert, welche später ausgeführt werden. Eine Speicherzahl ist nicht fest, da man verschiedene Größen bekommen kann, sie bewegen sich aber ungefähr bei ein paar kB. Ein kleiner Festplattenspeicher ist auch noch integriert, damit das Programm auch nach mehrmaligem Einschalten immer noch vorhanden ist.

Einen Nachteil hat der Mikrocontroller jedoch: Er kann immer nur ein Programm ausführen. Man gibt ein Programm darauf, das immer und immer wieder abgespielt wird.

Nur wozu ist er denn dann eigentlich gut? Aufgaben, für die der Mikrocontroller unter anderem verwendet wird, sind Ladegeräte, Motorsteuerungen, Roboter, Messwerterfassungen, MP3-Player, Temperaturregelung (Kühlschränke) und viele Bereiche im Auto (Airbags, u.a.).

Der Vorteil an Mikrocontrollern ist, dass sie sehr preiswert sind und sich einfach programmieren lassen. Es gibt viele unterschiedliche Hersteller der Mikrocontroller.

http://arduino.cc/en/Main/Products

Jetzt zu dem Begriff **„Arduino“**. Auf dem Bild rechts erkennt man das Arduino Uno. **Das Arduino ist also eine kleine Hardwareplatte, auf der ein Mikrocontroller sitzt.** In diesem Fall ist es der „AtMega328“. Wie oben beschrieben, verrichtet der Mikrocontroller die komplette Arbeit. Das Board ist dazu da, die einzelnen Pins der analogen und digitalen Ein- und Ausgänge ansteuern zu können. Das Arduino Uno ist nur eines von vielen Arduinos. Jedes hat einen anderen Mikrocontroller als Chip. Je nachdem, ob man mehr Speicher haben will, kann man bessere nehmen. Für den „normalen und spielerischen“ Gebrauch, reicht das Arduino Uno vollkommen.

Um noch einmal auf die Daten des Arduinos zurückzukommen, sollte man kurz einen Blick auf die Werte werfen. Das Arduino Uno besitzt einen Flashspeicher von 32Kbytes. Das ist eine normale Größe. Das Arduino kann mit einer maximalen Frequenz von 20MHz arbeiten. Das reicht auf jeden Fall für den normalen Gebrauch. Zum Vergleich: das leistungsstärkste Arduino ist das „Arduino Due“. Es besitzt einen ARM Cortex-M3 32-Bit Prozessor und kann mit einer Taktfrequenz von bis zu 84MHz arbeiten.

Die Arduino Boards werden entweder über **USB mit Spannung versorgt oder mit einem externen Netzteil, welches auf 5V eingestellt ist.** Unter Umständen kann es passieren, dass man die Volt bei dem Netzteil auf 7,5V stellen muss, da sonst nicht ausreichend Spannung anliegt. Wie man sehen kann, sind auf dem Arduino Board schon schwarze Pinleisten an den Außenseiten, an denen man Bauteile, wie zum Beispiel LEDs, hineinstecken kann. Pins nennt man Anschlüsse, welche das Arduino besitzt. Man steckt in die Pins die Bauteile, welche man benutzen will. Wahlweise benutzt man für größere Projekte Steckplatinen. Dort kann man Schaltungen einfacher aufbauen und man bekommt sie für einen günstigen Preis.

3. Programmieren?! - Grundlagen für Einsteiger...

Da Hardware und Software zwei komplett verschiedene Dinge sind, darf man beide nicht aus den Augen verlieren. Im oberen Thema haben wir zumindest schon einmal dir Grundbegriffe für das Verständnis der Hardware geklärt. Jetzt müssen wir uns noch der Software widmen.

In diesem Tutorial arbeite ich mit der offiziellen Arduino Software. Später mehr dazu. Das Arduino ist „Open Source“ und damit frei beschreibbar. **Die Programmiersprache ist C und C++.** Für die Neulinge, welche komplett neu in das Thema „Programmieren“ einsteigen, gibt es in diesem Lernschritt ein kleines C Tutorial für den einfachen Gebrauch. Ich will nicht zu sehr auf das Programmieren eingehen, da später noch einige Programme genauer beschrieben werden. Dies dient lediglich zur groben Übersicht, was man beim Programmieren wissen sollte. Wer Genaueres über das Programmmieren herausfinden will, kann sich im Internet schlauer machen.

Fangen wir an. Die grundlegende Struktur einer Programmierung ist eigentlich gar nicht so schwer. Zuerst beachten wir drei Teile, welche man alle als „Blöcke“ deuten kann. Alle drei haben ihre Aufgabe.

```
int x = 13;    // deklariert Variable 'x' als Integer mit Wert 13

void setup()
{
  anweisungen;
}

void loop()
{
  anweisungen;
}
```

Hier sieht man einen „Rohaufbau“ eines Programmiercodes.

Int legt fest, welche Variablen denn gebraucht werden. Das **void setup( )** ist eine Vorbereitung. Das **void loop( )** hingegen ist für die Ausführung verantwortlich.

Der **„int“** Befehl legt eine Variable fest. Eine Variable kann eine LED sein, welche am Pin 13 ihren Pluspol hat. Alle Bauteile oder Werte, die festgelegt werden müssen, werden hier als Wert gespeichert. Natürlich kann man jetzt genauer auf Variablen eingehen. Es gibt noch Input Variablen, die mit „int“ festgelegt werden. Werte werden hier definiert und später dann im Programmcode abgefragt, welchen Wert sie denn jetzt besitzen. Später mehr zum Thema In- und Output. Wenn man sich merken kann, dass „int“ den Pin-Ort für unsere Bauteile speichert, ist man auf dem richtigen Weg, der fürs erste reichen sollte.

Als nächstes der **„setup“** Befehl. Das „setup“ ist die Vorbereitung und läuft einmal am Anfang durch und setzt die Pins fest, ob sie Output sind, also etwas herauskommen muss, oder Input. Input brauchen wir eher nicht im Moment. Besonderheit: Selbst wenn man kein „setup“ festlegt, was eigentlich nie vorkommt, muss man ihn trotzdem erwähnen. Er ist Grundbestandteil einer Programmierung.

```
void setup()
{
  pinMode(ledPin, OUTPUT);          // Setzt den OUTPUT Pin
}
```

Darauf folgend der **„loop“** Befehl. Wie schon das Wort sagt, wird etwas wiederholt. Wenn die Pins mit dem „int“ und dem „setup“ Befehl festgelegt wurden, kommt der nächste Baustein, der sich „loop“ nennt. Er wiederholt das dort hineingeschriebene Programm immer und immer wieder. Der Code befindet sich in einer Dauerschleife.

In dem gezeigten Bild eines „loop“ Befehls sieht man deutlich, dass ein Pin (wahrscheinlich eine LED) eine Sekunde ein- und dann eine Sekunde ausgeschaltet wird. Dieser Befehl wird in einer Dauerschleife ausgeführt.

```
void loop()
{
  digitalWrite(pin, HIGH);   // schaltet 'pin' ein
  delay(1000);               // Pause für eine Sekunde
  digitalWrite(pin, LOW);    // schaltet 'pin' aus
  delay(1000);               // Pause für eine Sekunde
}
```

Der „loop“ Befehl ist daher so wichtig, weil er immer wieder mit dem Arduino Board interagiert. Wenn sich jetzt ein Wert irgendwann verändert, beispielsweise mit einem Potentiometer, wird der Wert immer und immer wieder neu ausgelesen und somit jede Veränderung bemerkt.

Vielleicht ist es auch noch wichtig zu wissen, dass man manchmal **„Libraries“** einfügen muss. Es gibt verschiedene „Libraries“ und übersetzt könnte man „Bibliotheken“ dazu sagen. Dort drin befinden sich die jeweiligen Befehle je nach Library. Das heißt, wenn man Befehle benutzt, aber die Library nicht eingebunden hat, welche den Befehl kennt, wird der Befehl nicht erkannt. Sie befinden sich über dem kompletten Programmcode. Bekannte Libraries wären „conio.h“, „stdio.h“ oder „sdtlib.h“.

Noch ein paar Tipps, die man eventuell wissen sollte...

- Hinter jede fertige Zeile kommt ein Semikolon.

```
int x = 13;
```

- Wenn man sich Notizen hinter eine fertige Zeile machen will, kann man zwei Querstriche benutzen „//“.

```
int x = 13;   // deklariert Variable 'x'
```

- Block Kommentare über mehrere Zeilen .

```
/*  Dies ist eine eingefügter Block Kommentar
    bitte den schliessenden Kommentar nicht vergessen -
    Diese müssen ausgeglichen sein
*/
```

- Geschweifte Klammern {} umschließen eine Funktion. Also beispielsweise alles, was in die Funktionen void loop oder in die void setup gehört.

```
Typ FunktionsName(parameter)
{
  anweisungen;
}
```

- **Gibt es auch Verzögerungen?** Ja! Wenn jetzt etwas beispielsweise länger warten soll, benutzt man einen „delay (Zeit in ms)“, um etwas zu verzögern.

```
delay(1000);   // wartet für eine Sekunde
```

- Verschiedene Datentypen (für Zahlen): „int“, „long“, „float“ und „array's“. Hier werden wir sie nicht unbedingt brauchen, da die „int“ Variablen völlig ausreichend sind bei dem, was wir machen.
- Etwas auf einer LCD Anzeige schreiben, bzw. anzeigen lassen? Kein Problem „print(Text)“

Vielleicht sollte man noch etwas über die Ablaufsteuerungen wissen. Diese Befehle sind auch ein Faktor für Programmcodes, die man **später** vielleicht einmal gebrauchen könnte

```
if (inputPin == HIGH)
{
  doThingA;
}
```

Die **„if“-**Abfrage testet, ob etwas bestätigt wird oder falsch ist. Je nach dem handelt diese Abfrage dann. Wenn dieser Fall zutrifft, so wird das in der geschweiften Klammer ausgeführt. Ansonsten passiert nichts.

```
if (inputPin == HIGH)
{
  doThingA;
}
else
{
  doThingB;
}
```

Die **„if...else“-Abfrage** ist die Erweiterung zu der normalen „if“-Abfrage. Wenn die „if“-Abfrage nicht erfüllt wird, wird der „else“-Weg gewählt und der Code in der geschweiften Klammer nach dem „else“ ausgeführt.

```
for (Initialisierung; Bedingung; Ausdruck)
{
  doSomething;
}
```

Die **„for“** Schleife hat einen lustigen Effekt. Alles, was sich in der geschweiften Klammer befindet, wird solange wiederholt, wie man es oben bei der Initialisierung, der Bedingung und dem Ausdruck, eingegeben hat. Beispielsweise kann man von int = 0 bis int = 20 in Einer-Schritten zählen. Hinweis: Falls man Schleifen oder Abfragen benutzen sollte, empfiehlt es sich, die „for“-Schleife zu nehmen, da sie relativ einfach ist.

```
while (someVariable ?? value)
{
  doSomething;
}
```

Die **„while“-**Schleife hat auch einen einfach verständlichen Ablauf. Sie wiederholt so lange etwas in der geschweiften Klammer, bis die „someVariable“ nicht mehr dem „value“ entsprechend ist. Dann hört sie auf. Beispielsweise soll die „while“-Schleife immer arbeiten, wenn der someVariable-Wert unter 50 ist. Sobald er die 50 erreicht hat, funktioniert sie nicht mehr und der Befehl hört auf zu arbeiten.

Digitaler In- und Output? Was bedeuten sie?

Mit dem „void setup“ werden Pins entweder als Ouput – Ausgang oder als Input – Eingang festgelegt. Standardmäßig sind sie alle digitalen Pins Eingänge, also Inputs. So muss man also nur einen digitalen Output festlegen, wenn er gewünscht ist.

Beispielsweise kann man LED's mit einem digitalen Output Befehl aufleuchten lassen. Da der maximale Strom leider nur bei 40mA liegt, kann man andere Geräte, wie Motoren, wahrscheinlich nicht über einen Pin laufen lassen, da er mehr benötigt.

```
pinMode(pin, OUTPUT);     // setzt 'pin' als Ausgang
```

Bei einem digitalen Input ist das Bild, welches oben gezeigt ist, eigentlich das gleiche, außer, dass

bei Output der Input steht. Digitale Inputs werden normalerweise für Schalter benutzt. Die Pins lesen also aus, auf was der Schalter gerade steht. Ein Input „kontrolliert“ sozusagen, was dort gerade passiert.

Jetzt zu den **digitalen Befehlen**, die letztendlich auch was mit den Bausteinen machen, die vorher als Output oder Input festgelegt wurden.

```
digitalWrite(pin, HIGH);   // setzt 'pin' auf high (an)
```

Dieser „digitalWrite“-Befehl setzt den pin auf High. Das bedeutet, dass z. B. eine angeschlossene LED an geht. Das geht nur, wenn der Pin als Output festgelegt wurde.

Jetzt noch der Aspekt, wenn ein Wert ausgelesen werden soll und dann den „value“ festlegt. Dieser „value“ wird dann benutzt für andere Progammcodes. Wie gesagt, zuerst jedoch muss ein Bauteil ausgelesen werden, um diesen Wert zu ermitteln

```
value = digitalRead(Pin);   // setzt 'value' gleich mit
                            // dem Eingangspin
```

Es gibt auch noch einen **analogen In- und Output**. Bei dem Befehl „analogRead(pin)“ wird ein Wert eingelesen mit einer Auflösung von 10 Bit. Dies funktioniert nur bei den Pins A0-A5 am Arduino Uno Board. Bei dem Befehl „analogWrite(pin,value), wird der Wert dann auf den Pin geschrieben. Input und Output funktionieren ähnlich wie die digitalen. Ein analoger Wert beschreibt jedoch nur 0-255. Wenn jetzt 5V die obere Grenze ist, werden sie mit 255 festgelegt. 0V wäre der Wert 0. Über ein Potentiometer kann man dann die Zwischenwerte gut einstellen. Man braucht lediglich einen Regler. Durch die Hardware, die hier benutzt wird, werden Änderungen mit den analogen Pins festgelegt.

```
value = analogRead(pin);  // setzt 'value' gleich mit 'pin'
analogWrite(pin, value);  // schreibt 'value' auf den analogen 'pin'
```

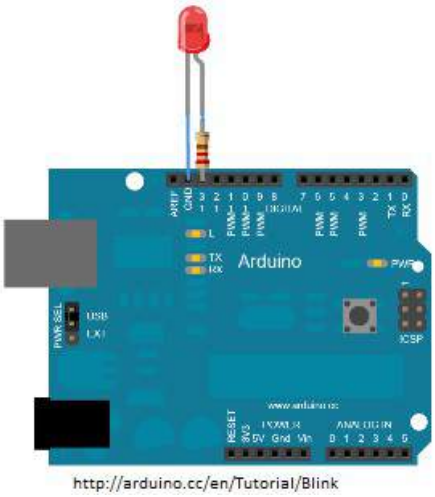
So weit, so gut. Dies sind die wichtigsten Befehle, um kleinere Programme schreiben zu können. Natürlich gibt es noch einige Befehle mehr, aber die werden wir hier nicht benötigen. Für den „einfachen“ Gebrauch sind unsere ausgewählten Befehle vollkommen ausreichend.

Wem das jetzt zu schnell ging, keine Angst, jetzt kommen noch ein paar Beispiele von mir im nächsten Lernschritt, damit ihr diese Befehle noch einmal am Stück seht, um ein Gefühl dafür zu bekommen, wie so ein kompletter Programmcode den funktioniert und aussieht.

4. Beispiele für Programmierungen...

Dieser Lernschritt dient als eine Art Übersicht über das komplette Thema „Programmieren“.

Fangen wir mit etwas ganz leichtem an. Eine blinkende LED. Dieser Versuch kann auch auf der Homepage nachgelesen werden unter http://arduino.cc/en/Tutorial/Blink.



http://arduino.cc/en/Tutorial/Blink

Auf dem Bild links erkennt man das Arduino Uno Board mit einer LED, welche einen kleinen Vorwiderstand im Pin 13 hat (220Ohm beispielsweise) und anschließend eine LED angeschlossen ist, die ihren Minuspol am Ground besitzt.

Hier findet sich eine genaue Erklärung zum Programmcode:

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

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

LED wurde auf den Pin 13 festgelegt

Da es sich hier um eine LED handelt, wird der Pin als ein Ausgang bestimmt.

Dieser Part wird im Programm nur einmal am Anfang durchlaufen

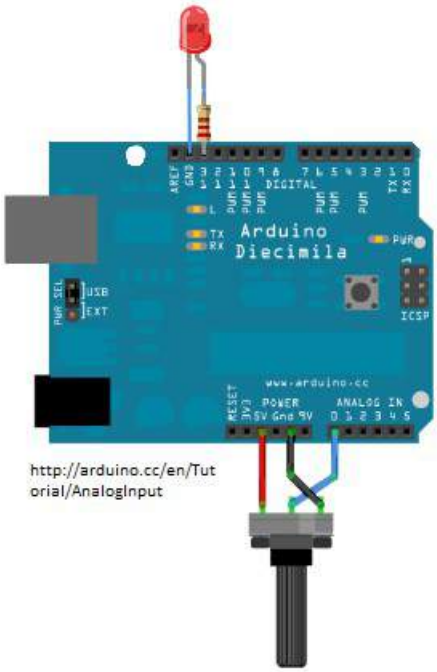
Die LED wird mit "HIGH" beschrieben. Sie wird eingeschaltet.

Dieser Befehl ist nur eine Zeitverzögerung. Für eine Sekunde passiert hier gar nichts mehr

Die LED wird neu beschrieben mit "LOW". Sie wird ausgeschaltet.

Alles, was im "void loop" steht wird in einer Dauerschleife ausgeführt. Was heißt das? Eine LED wird angemacht, leuchtet eine Sekunde. Dann wird eine LED ausgemacht und wartet eine Sekunde. Das Programm ist hier fertig, jedoch durch die Dauerschleife fängt das Programm wieder an die LED einzuschalten. Das Blinken fängt von vorne an

Nehmen wir noch ein schwereres Beispiel.



Diesmal vielleicht mit Werten, die zwecks einem Potentiometers ausgelesen werden. Je nach dem, wie das Potentiometer eingestellt ist, blinkt die LED schneller oder langsamer. Das Bild links zeigt, welche Pins am Arduino benötigt werden. Diese Schaltung nennt sich „Analog Input“ und lässt sich auch unter den Examples auf der offiziellen Arduino Seite finden. Die Seite dazu ist hier http://arduino.cc/en/Tutorial/AnalogInput.

Am Pin 13 ist erneut der Vorwiderstand angeschlossen, der dann zum Pluspol der LED wird. Die LED geht mit ihrem Minuspol zum Ground zurück.

Das Potentiometer ist mit dem Schleifer am A0 (Analog 0) angeschlossen. Der Pluspol geht auf den 5V Pin und der Minuspol geht zum Ground.

Mit dem passenden Programmcode kann man jetzt die beiden Bauteile miteinander interagieren lassen.

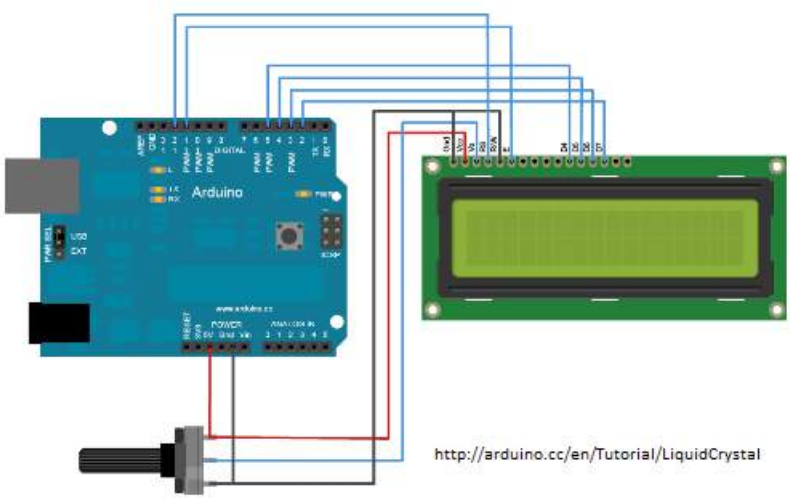
Das ist der Progammcode, damit man die Schnelligkeit der blinkenden LED einstellen kann mit einem Potentiometer. Hinzugefügt sind einige Erklärungen.

```
int sensorPin = A0;                 // Der Schleifer vom Potentiometer wird am A0 Pin befestigt. Hier wird er festgelegt
int ledPin = 13;                    // Unsere LED ist wieder am Pin 13 festgelegt mit ihrem Pluspol
int sensorValue = 0;                // Der Wert, welcher vom Potentiometer eingestellt wird ist hier zwischengespeichert

void setup() {                      // Wird einmal gelesen und festgelegt
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);          // Der LED Pin ist wieder als ein Ausgang (Ouput) festgelegt
}

void loop() {                       // Alle Befehle werden in einer Dauerschleife ausgeführt
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);   // Der vom Potentiometer eingestellte Wert wird hier gelesen
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);       // Hier wird unsere LED eingeschaltet
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);               // Das Programm macht jetzt je nach eingstelltem Wert eine Pause (Zeitverzögerung)
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);        // Jetzt wird die LED wieder ausgeschaltet
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);               // Jetzt bleibt die ausgeschaltete LED je nach Wert aus.
}
```

Noch ein Beispiel mit einem Liquid Crystal Display (Kurz LCD).

Hinweis direkt am Anfang!! Es gibt 2 Arten von LCD-Anzeigen. Die einen haben eine Beleuchtung, die anderen nicht. Je nach dem besitzen die mit Beleuchtung insgesamt 16 Anschlüsse. Pin 15 muss die 5V und Pin 16 an den Ground. Benutzer mit Anzeigen ohne Beleuchtung müssen das hier nicht beachten, da sie nur 14 Anschlüsse insgesamt besitzen.

Das Potentiometer in diesen folgenden beiden Programmcodes hat keinen Einfluss auf den Versuch. Es ist nur da, um die Helligkeit, bzw. die Intensität der LCD-Anzeige einzustellen.



Wenn die Anzeige einmal an das Arduino Uno angeschlossen wird, hat man viele Möglichkeiten über Funktionen, welche man sich anzeigen lassen kann auf dem Display. In diesem Beispiel wird zum einen „hello, world“ angezeigt (der Standardgruß von dem Projekt, welches getestet wird) und der Sekunden, welche hoch zählen seit des letzten Resets des Programms durch den Resetknopf des Arduino Unos.

Hier ist der dazugehörige Programmcode und deren Erklärung zu den einzelnen Befehlen. Das Beispiel lässt sich auch auf der Homepage finden: http://www.arduino.cc/en/Tutorial/LiquidCrystal.

```
// include the library code:
#include <LiquidCrystal.h>
```
Da die LCD Library jetzt hinzugefügt wurde können wir auch mit einer Anzeige arbeiten und sie ansteuern.

```
// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```
Hier werden die Pins festgelegt, an denen die LCD Anzeige angebracht wird

```
void setup() {
```
Das hier passiert einmal

```
  lcd.begin(16, 2);
```
Diese LCD Anzeige besitzt 2 Reihen mit jeweils 16 Zeichen

```
  lcd.print("hello, world!");
}
```
Per "print" Befehl wird in die erste Reihe "hello, world!" geschrieben

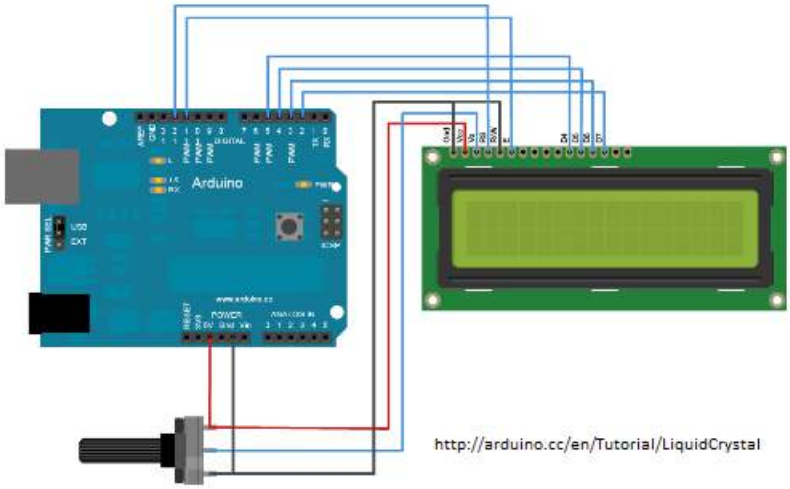
```
void loop() {
```
Das hier ist in einer Dauerschleife und läuft durchgehend

```
  // set the cursor to column 0, line 1
  lcd.setCursor(0, 1);
```
Hier wird festgelegt, dass der folgende Befehl in der "0." Spalte anfängt in der Reihe 2! Da von 0 gezählt wird ist die 1. Reihe die zweite!

```
  // print the number of seconds since reset:
  lcd.print(millis()/1000);
}
```
Das ist eine lustige Spielerei. Im Sekundentakt wird hier hochgezählt bis man das Arduino zurücksetzt.

Jetzt kommt ein Programmcode, indem ich euch nicht erkläre, was passiert. Das ist jetzt zum eigenen Testen, wie gut man durch einen Programmcode kommt und die einzelnen Befehle versteht bzw. nachvollziehen kann. Der Aufbau der LCD-Anzeige am Arduino ist dieselbe, wie im Beispiel davor. Die Lösung kann man auf der offiziellen Arduino Homepage finden unter: http://arduino.cc/en/Tutorial/LiquidCrystalDisplay.

```
#include <LiquidCrystal.h>


LiquidCrystal lcd(12, 11, 5, 4,

void setup() {

  lcd.begin(16, 2);

  lcd.print("hello, world!");
}

void loop() {

  lcd.noDisplay();
  delay(500);

  lcd.display();
  delay(500);
}
```

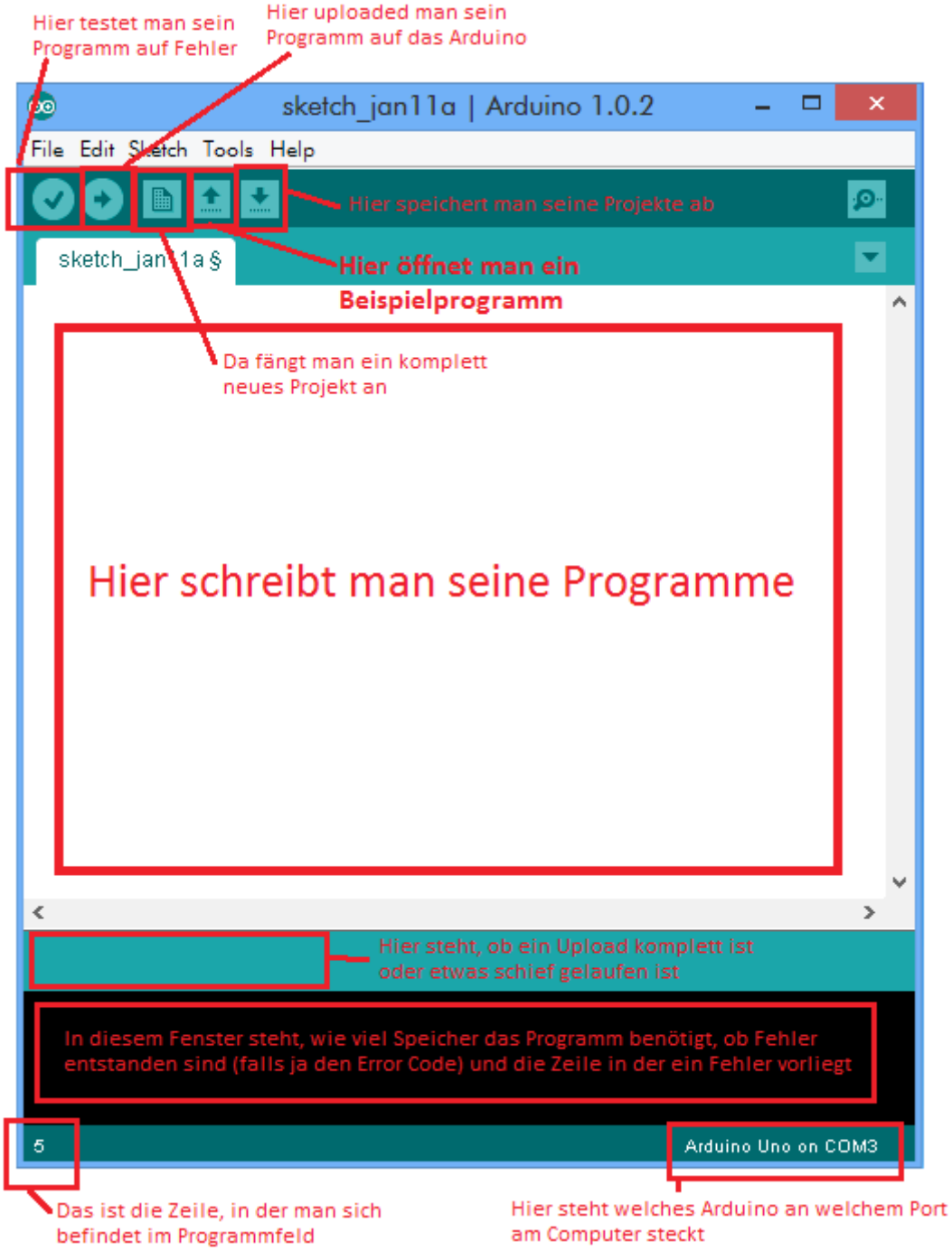


Ihr werdet sehen, dass man eigentlich relativ schnell die Begriffe und die einzelnen Befehle versteht. Wer diesen Lernschritt einigermaßen verstanden hat, kann schon die ersten Versuche am Arduino Uno Board vornehmen. In den nächsten Schritten wird euch das auch gezeigt.

5. Die Arduino (Programmier-) Software erklärt...

Dieser Lernschritt bringt euch das Arbeiten mit der Programmier-Software für das Arduino bei. Ihr könnt dieses Programm kostenlos herunterladen auf der offiziellen Homepage für Arduinos. Den Link dazu findet ihr hier: **http://arduino.cc/en/Main/Software.** Das Programm läuft auf allen bekannten Betriebssystemen. Windows, Mac OS X und Linux (32bit und 64bit) werden unterstützt. Es gibt auch noch andere Programme, mit denen man für das Arduino Programmieren kann, aber für mich ist das ein flexibler Weg, da dieses Programm für Arduinos gemacht ist.

Wenn ihr euch die Software heruntergeladen und geöffnet habt, schauen wir uns erst einmal die Oberfläche an. Eine kleine Übersicht über die Symbole und deren Bedeutung:



Das ist die Oberfläche, die man zum Programmieren verwendet. **Hinweis: Programme, bzw. Projekte nennen sich hier „Sketch“.** Wie beschrieben, werden Fehler genau angezeigt. Falls das Arduino nicht gefunden wird, erhält man einen Error Code, den man im Internet finden kann. Da es eine Vielzahl von Fehler-Codes gibt, kann ich jetzt keine genaueren Anweisungen dazu geben.

6. Ein Arduino in Betrieb nehmen…

Hier erkläre ich euch, wie man ein Arduino das erste Mal in Betrieb nehmen kann. Fassen wir zusammen: Wir haben jetzt das Arduino Board, das A auf B (Druckerkabel)-Kabel und die offene Software auf eurem Computer.

Da ich selber nur Windows benutze, kann ich nicht die Arduino-Installation unter Linux und MAC OS X zeigen. Wenn ihr dort das Arduino Board erfolgreich zum Laufen gebracht habt, könnt ihr meinem Tutorial wieder folgen. Überspringt dazu „Für Windows 7/8“. Windows-Benutzer starten bitte hier:

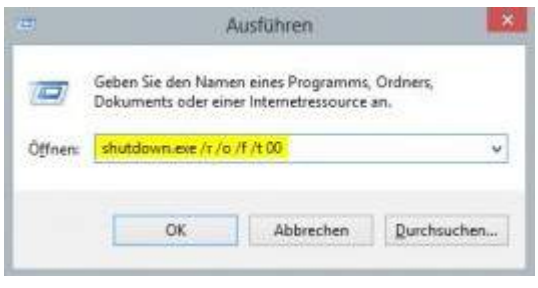
Für Windows 7:

1. Verbindet das Arduino Uno mit dem Druckerkabel und steckt es anschließend in einen USB Port.
2. Windows wird erkennen, dass ihr ein Gerät eingesteckt habt und sucht den Treiber.
3. Windows findet nicht immer den gewünschten Treiber auf dem Computer
4. Sucht manuell auf dem Computer. Da ihr die Arduino Software heruntergeladen habt, müsste sich in dem heruntergeladenen Arduino-Ordner ein anderer Ordner namens „drivers“ befinden
5. Jetzt müsste das Arduino korrekt erkannt werden.
6. Bitte „Für Windows 8“ überspringen

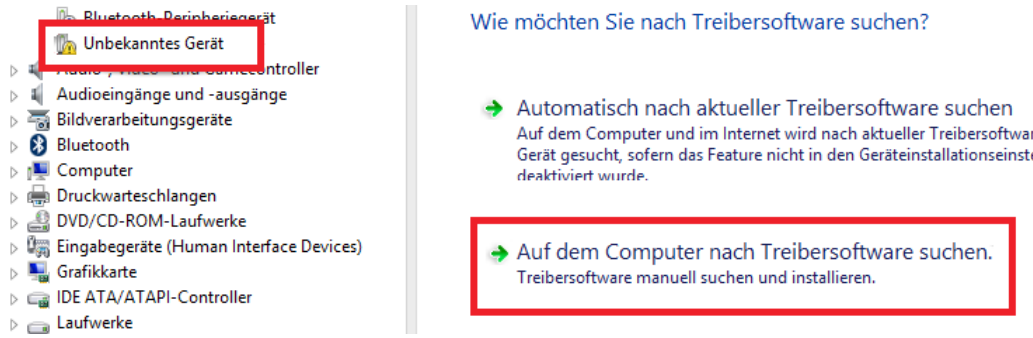
Für Windows 8:

Windows 8 hat ein paar Probleme mehr das Arduino anzunehmen und es erfordert ein paar Zwischenschritte um es erfolgreich in den Betrieb zu nehmen.

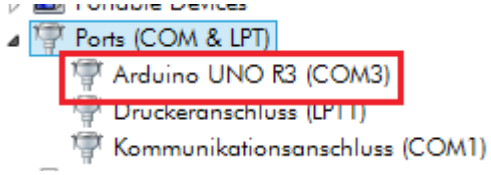
1. Da es sich hier um einen unsignierten Treiber handelt, den ihr herunterladen wollt, müssen wir das in Windows 8 erst erlauben.
2. Öffnet mit Rechtsklick unten links „Ausführen“ und gebt den Befehl `shutdown.exe /r /o /f /t 00` ein. Schreibt euch aber vorher Schritt 3 auf.



3. Jetzt kommt von Windows 8 das Options-Menü.
    1. Geht auf „Problembehandlung (Troubleshoot)“
    2. Geht dann auf „Erweiterte Optionen (Advanced options)“
    3. Als nächstes auf „Starteinstellungen (Startup Settings)“
    4. Dann mit einem Druck auf die Taste 7 für „Erzwingen der Treibersignatur deaktivieren (Disable Driver Signature Enforcement)
4. Wenn der Computer neu hochgefahren ist, steckt ihr das Arduino Uno in einen USB-Port per Kabel und öffnet anschließend den „Gerätemanager“
5. Ihr müsstet ein nicht installiertes System sehen, welches das Arduino ist. Sucht manuell auf dem Computer nach einem Treiber. Da ihr schon die Arduino Software heruntergeladen habt, befindet sich in diesem Ordner ein weiterer Ordner namens „drivers“. Den lasst ihr durchsuchen.

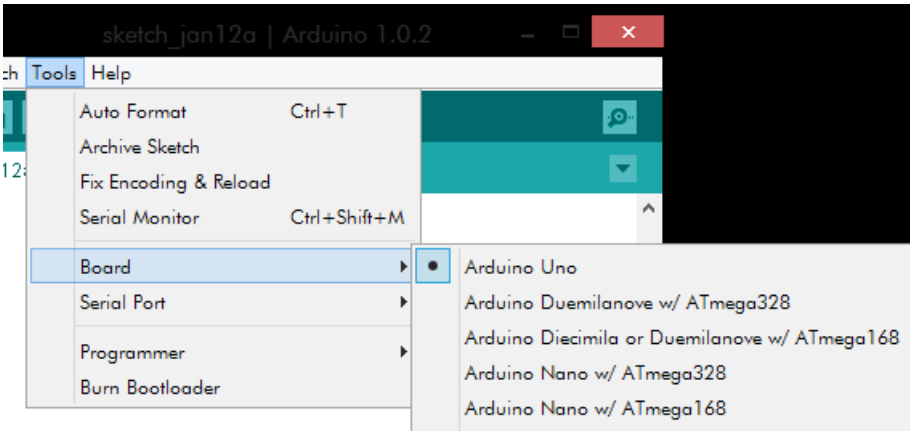


6. In diesem Schritt müsste jetzt euer Arduino korrekt installiert sein und im Gerätemanager mit passendem Port angezeigt werden. Hinweis: Merkt euch die „COM..“ Zahl eures Arduinos.

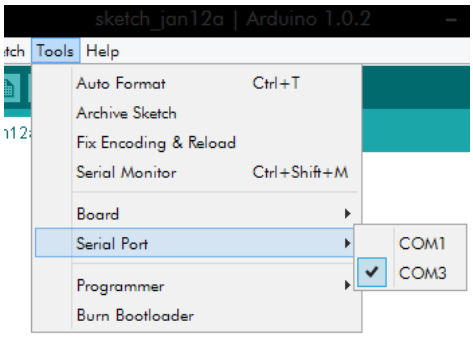


Ab hier geht es wieder für Windows 7,8 (und auch Mac OS X und Linux, wenn das Arduino dort installiert wurde) weiter:

1. Öffnet die Arduino Software
2. Stellt unter „Tools“ → „Board“ euer Arduino ein (Wahrscheinlich Arduino Uno)



3. Jetzt müsst ihr nur noch den „Serial Port“ einstellen, den ihr auch unter „Tools“ findet. Der Port ist bei jeder Person ein anderer, daher müsst ihr im „Gerätemanager“ nachschauen. Bei mir ist es „COM3“



4. Herzlichen Glückwunsch!! Das Arduino wurde erfolgreich installiert und in der Software eingestellt. Jetzt kann man das Arduino aus dem USB-Port nehmen, wenn man es nicht mehr benötigt und wieder einstecken, wenn man daran weiter arbeiten möchte. Zukünftig müsste es sofort erkannt werden und betriebsbereit sein.

7. Programme übertragen und selbst schreiben

Dieser Lernschritt ist noch einmal eine Zusammenfassung und kurze Darstellung, wie ihr auf euer Arduino nach dem erfolgreichen installieren Programme schreiben oder laufen lassen könnt.

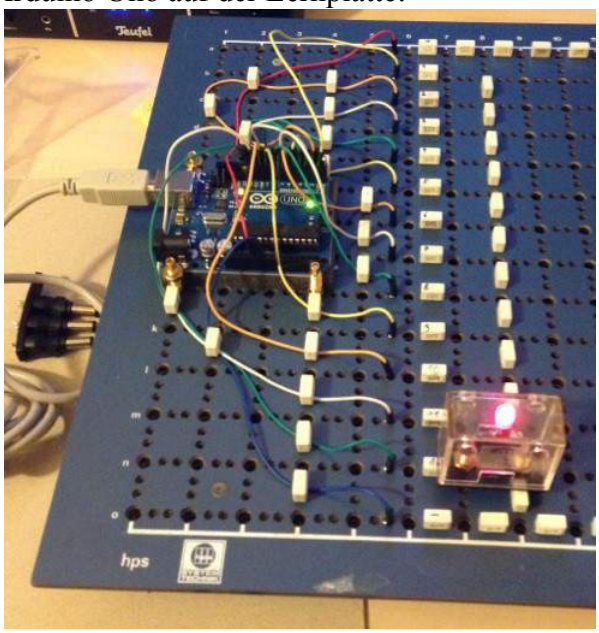
1. Öffnet die Arduino Software und steckt das Arduino Uno in einen USB-Port mit dem Kabel
2. Wenn ihr den Button „Open“ drückt, findet ihr fast jedes Beispiel, welches auch auf der Arduino-Hompage verzeichnet ist. Dort sind einfache Beispiele für den Anfang, z. B.: Unter „01. Basics“ → „Blink“
3. Wenn ihr ein Programm ausgewählt habt und das Arduino angeschossen ist, könnt ihr das Programm über „Upload“ auf euer Arduino laden. **Je nach Computerleistung kann es etwas länger dauern (bis zu 5 Minuten) oder auch schnell gehen (ungefähr 5 Sekunden)**
4. Für den Anfang empfehle ich nur Beispielprogramme auszuwählen und diese dann mit eigenen Bauteilen zu stecken (wenn es nicht viele Bauteile sind)
5. Ich würde auch eine Steckplatine empfehlen, wenn man Spaß am Programmieren mit einem Arduino hat. Eine Steckplatine kann man günstig in einem Set bekommen, bei dem auch die Arduino-Anschlusskabel dabei sind. Den Link findet ihr oben bei den Materialien in der Beschreibung. Man hat dann mehr Möglichkeiten, da man eine komplette Fläche hat, auf der man Bauteile stecken kann und diese nachher wieder abzutrennen sind.

Ich bedanke mich an dieser Stelle bei euch und hoffe, dass dieses Tutorial dem Einen oder Anderen helfen konnte. Das komplette Arduino-Projekt ist ziemlich interessant und nicht schwer zu bedienen. Ihr werdet sehen, schon nach kurzer Zeit bekommt man außergewöhnliche Sachen hin und ihr möchtet wahrscheinlich nicht mehr aufhören.

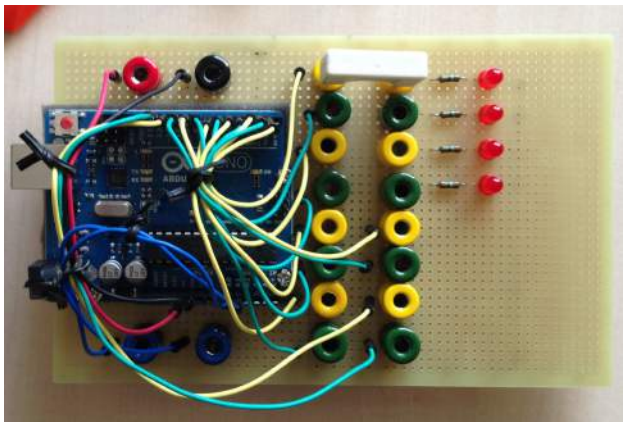
Mein Name ist Christian Sommer und ich bin Schüler auf der Heinrich-Emanuel-Merck-Schule in der 13. Klasse in der Fachrichtung Elektrotechnik im Beruflichen Gymnasium. Meine Aufgabe war es, ein Projekt mit einem Arduino Uno zu machen. Ich sollte mich näher damit auseinandersetzen und zeigen, welche Funktionen es besitzt und was man damit alles machen kann. Da es aber unzählige Funktionen hat und keine großen Grundkenntnisse erfordert, kann es in den späteren Klassen eingesetzt werden, da man wirklich viel lernt.

Hier sind noch ein paar Bilder zu meinem „Projekt“ für das erste Halbjahr der 13. Klasse. Meine Aufgabe war es, zuerst ein Arduino Board auf eine große Lernplatte zu stecken, um mit ein paar Bauteilen die Funktionen zu testen. Da dies so gut funktioniert hat, war meine neue Aufgabe, das Arduino Uno auf eine eigene Platine mit Buchsen zu befestigen. Auf dieser Platine sind 12 normale Anschlüsse, zwei analoge Pins, ein 5V-Anschluss und ein Ground. Vier weitere Stecker kann man mittels einer genormten Brücke direkt an eine LED mit Vorwiderstand verbinden. Es gibt noch ein Potentiometer, welches die Intensität der LCD-Anzeige einstellt und eine LCD Anzeige, welche beleuchtet ist und 2x16 Zeichen besitzt. Wenn man die LCD Anzeige benutzen will, muss man dementsprechend die Pins umstecken.

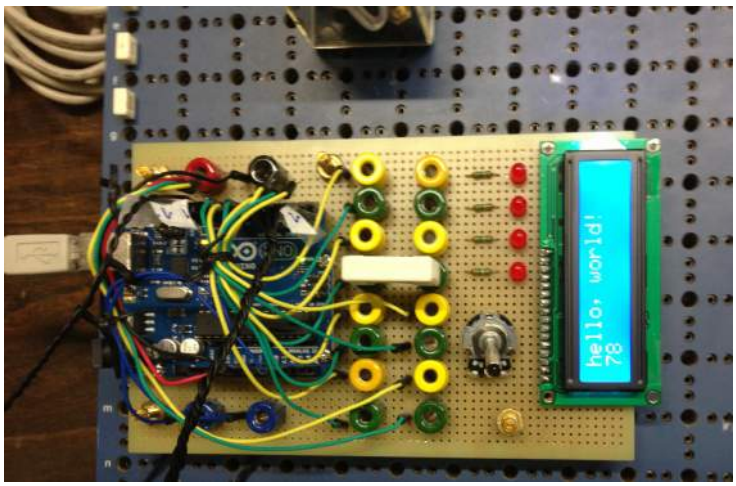
Arduino Uno auf der Lernplatte:



Das Arduino Uno ohne LCD Anzeige:



Das Arduino Uno mit LCD Anzeige und im Betrieb:



Literatur...

- Bild auf dem Cover:
  - http://static3.watterott.com/arduino_uno-r3_1.jpg

- Die Arduino Seite:
  - http://arduino.cc/en/

- Was ist ein Mikrocontroller:
  - http://www.asklubo.com/de/elektronik/was-ist-ein-mikrocontroller
  - http://fynns-programming.de/index.php/was-ist-ein-mikrocontroller
  - http://www.mikrocontroller.net/articles/Mikrocontroller
  - http://de.wikipedia.org/wiki/Mikrocontroller

- Was ist ein Arduino:
  - http://www.atmel.com/devices/atmega328.aspx?tab=parameters
  - http://de.wikipedia.org/w/index.php?title=Arduino-Plattform&stable=1

- Programmieren:
  - http://www.netzmafia.de/skripten/hardware/Arduino/Arduino_Programmierhandbuch.pdf

- Beispiele beim Programmieren:
  - http://arduino.cc/en/Tutorial/Blink
  - http://arduino.cc/en/Tutorial/AnalogInput
  - http://arduino.cc/en/Tutorial/LiquidCrystal
  - http://arduino.cc/en/Tutorial/LiquidCrystalDisplay

- In Betrieb nehmen:
  - http://code-bude.net/2012/11/18/arduino-treiber-unter-windows-8-installieren/
  - http://www.bryonconnolly.com/windows-8-arduino-driver-install

- Bilder des Arduinos:
  - Entstanden in Darmstadt, Heinrich-Emanuel-Merck-Schule

- Seiten wurden das letzte mal am 12.1.13 aufgerufen

- Bericht von 10.1.13

Christian Sommer, Datum

_____