Table of Contents

Anhang

senseBox:edu

Die senseBox:edu ist ein Werkzeugkasten, der Schülerinnen und Schülern sowie Nachwuchsforscherinnen und -forschern das Programmieren spielerisch und greifbar vermitteln soll. Dazu werden einfache Schaltungen aufgebaut, die mit Hilfe eines Mikrokontrollers programmiert und gesteuert werden. Die verwendeten Mikrokontroller basieren auf der einsteigerfreundlichen Arduino Plattform, die mit einer C-ähnlichen Sprache über eine simple Programmieroberfläche (Abkürzung IDE, engl. für Integrated Development Environment) gesteuert werden.

Zusammen mit dieser Anleitung und dem Bauteilsortiment vermittelt die senseBox:edu einen praxisnahen Einstieg in die Programmierung sowie eine schrittweise Einführung zur Gestaltung eigener Technikprojekte aus den Bereichen Geoinformatik, Sensorik und Photonik, wie zum Beispiel Projekte zur Thematik Zukunftsstadt.

Über dieses Buch

Auf diesen Seiten sind alle Unterlagen zur senseBox:edu zu finden. Dies umfasst Anleitungen, Aufgaben, Beispiel-Projekte, sowie Downloadlinks und ein FAQ.

Diese Seiten sind auch als PDF Dokument verfügbar, besuche dazu den Downloadbereich.

Eigene Projekte beitragen

Wenn ihr ein eigenes Projekt selbst entwickelt und die Entwicklung dokumentiert habt, fügen wir eure Anleitung gerne zu diesem Buch hinzu!

Die Dokumentation sollte einen Überblick über das Projekt geben, und eine Schritt-für-Schritt Anleitung zum Nachbau enthalten. Fotos und Programmcode können ebenfalls hinzugefügt werden!

Für die Bearbeitung und Verwaltung dieses Buches verwenden wir github.com und das Tool GitBook, alle Inhalte werden in Markdown geschrieben.

Nicht nur eigene Inhalte, auch Korrekturen & Verbesserungen am bestehenden Inhalt sind gern gesehen ;)

Schreiben der Dokumentation

Die Dokumentation sollte im Markdown-Format verfasst sein, sodass der Inhalt direkt in das Buch eingearbeitet werden kann. Falls ihr nicht vertraut mit Markdown seid, findet ihr hier eine Hilfestellung und Syntaxerläuterung.

Um das Schreiben mit Markdown zu vereinfachen gibt es übrigens auch online-Editoren wie stackedit.

Inhalt

Sofern sie für euer Projekt geeignet ist, könnt ihr euch an dieser **Vorlage** für Anleitungen orientieren.

In jedem Fall sollten alle Informationen enthalten sein, sodass das Projekt von anderen wiederholt werden kann!

Dateinamen

Achtet darauf, dass in den Dateinamen keine Leerzeichen enthalten sind! Wenn Bilder oder andere Dateien (zB. Arduino-Sketches) eingebunden werden sollen, legt diese einfach im in einem gleichnamigen Unterordner ab:

```
mobile-wetterstation.md
mobile-wetterstation/uebersicht.jpg
mobile-wetterstation/mobile-wetterstation.ino
```

Lizenz

Euer Beitrag wird - wie der Rest des Buchs - unter der CC BY-SA 4.0 Lizenz eingefügt. Das bedeutet, dass der Beitrag von anderen frei weiterverwendet werden darf, solange der Autor zusammen mit dem Inhalt genannt wird.

Hochladen der Dokumentation

Um uns die Dokumentation zuzusenden, fügt ihr diese am besten in das Buch selber ein.

Der Quellcode dieses Buches ist hier auf GitHub zu finden. **Forkt dazu dieses Repository, fügt eure Inhalte dort ein, und erstellt dann einen Pullrequest.**

Falls ihr mit git nicht vertraut seid, erfahrt ihr nun im Detail wie das alles funktioniert. Wenn Fragen offen bleiben, stehen wir euch auch gerne per Mail zur Verfügung.

GitHub Account erstellen

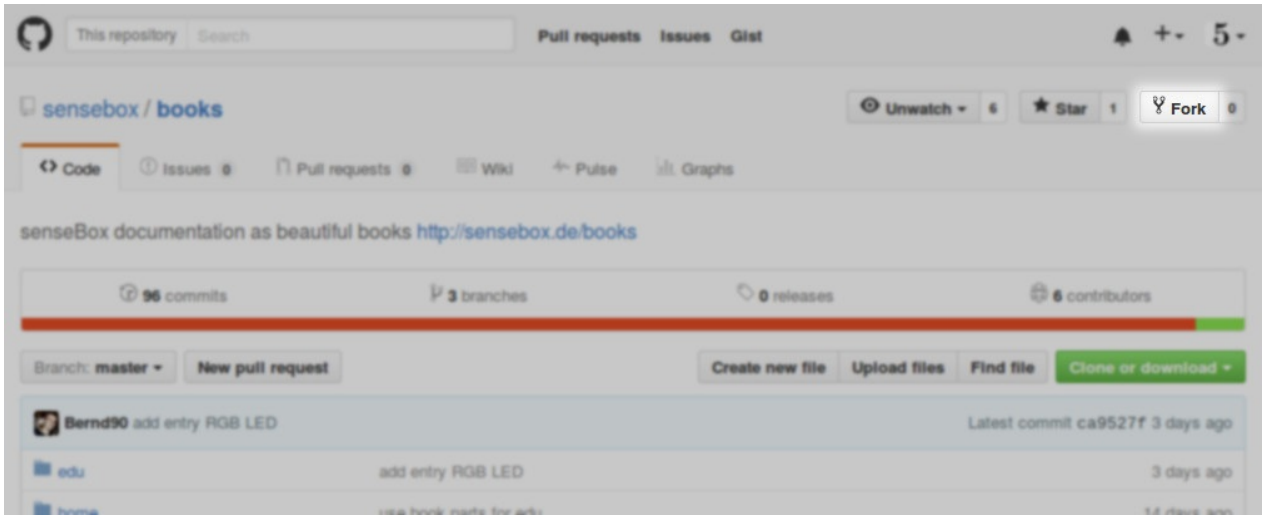
Um eigene Beiträge auf GitHub zu erstellen wird dort ein Account benötigt. Erstellt diesen hier. Das kostenlose Konto ist vollkommen ausreichend, da wir keine nicht-öffentlichen Repositories erstellen wollen.

GitHub ist ein Dienst, welcher git-Repositories online bereitstellt. Ein *Repository* ist ein Verzeichnis für Code und Daten, welche inhaltlich zusammengehören. Zu jedem Repository gehört eine Versions-Geschichte, über welche vorherige Versionen wiederhergestellt werden können. Jede Version wird als *Commit* bezeichnet.

Repository forken

Durch diese Versionsverwaltung können gleichzeitige Entwicklungsstränge dezentral entwickelt und wieder zusammengeführt werden; einen abgezweigten Entwicklungsstrang bezeichnet man *Fork* (engl. Gabelung).

Um unser Buch also weiter zu entwickeln, müsst ihr das Quelltext-Repository forken. Besucht dazu die Seite des Repositories, und klickt den Button "Fork" oben rechts.

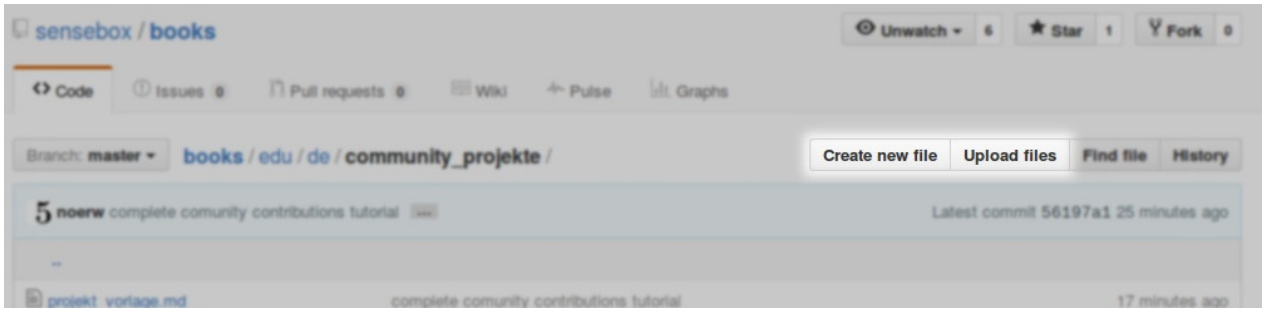


Das Repository wurde mit dem aktuellen Zustand zu eurem Account kopiert, und ist unter `https://github.com/<dein nutzername>/books` zu finden. Auf dieser Kopie habt ihr Schreibrechte, und könnt euren Inhalt einfügen!

Inhalt einfügen

Fügt eure Dateien in den Order `edu/de/community_projekte/` ein.

Besucht dazu die Adresse `https://github.com/<dein nutzername>/books/tree/master/edu/de/community_projekte`, wo ihr über die Buttons "Upload files" und "Create New File" Bilder und Textdateien hochladen könnt:

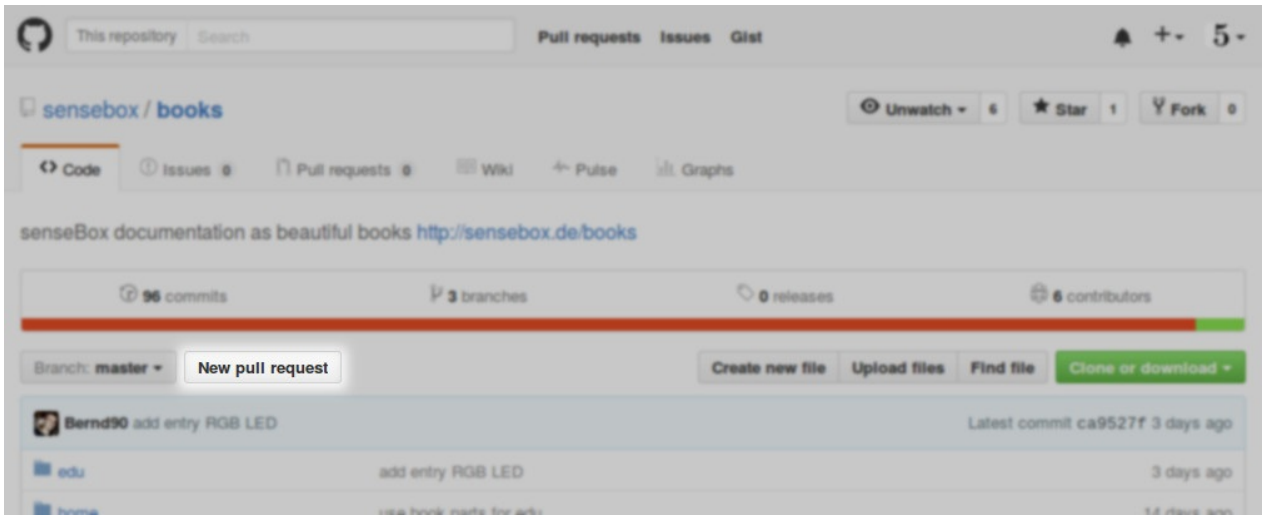


Zusätzlich könnt ihr die Projekt-Anleitung im Inhaltsverzeichnis (`edu/de/SUMMARY.md`) verlinken, hier ist ein Beispiel:

```
...
* [Community-Projekte](community_projekte.md)
    * ...
    * [Mobile Wetterstation](community_projekte/mobile_wetterstation.md)

...
```

Pullrequest erstellen

Nun da euer Inhalt online ist, könnt ihr uns benachrichtigen, dass ihr eure Änderung in unser Ursprungs-Repository einfügen wollt. Erstellt dazu auf der GitHub-Weboberfläche von eurem geforkten Repository einen Pull Request.



Im Erscheinenden Fenster könnt ihr eine kurze Beschreibung eurer Änderung und weitere Kommentare angeben.

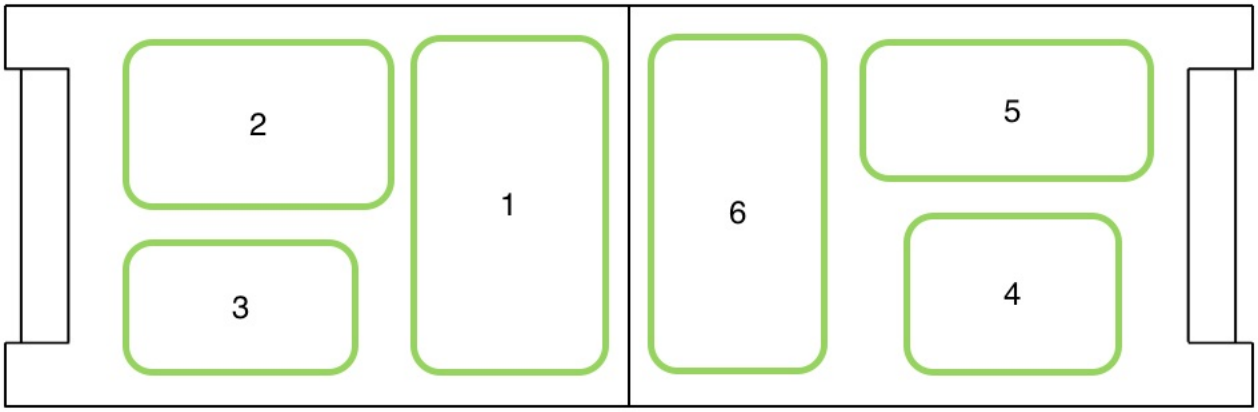
Wir werden nun über euren Beitrag informiert und uns eure Änderung ansehen. In den nächsten Tagen solltet ihr eure Anleitung in diesem Buch im Kapitel Community-Projekte finden!

Alternativ zu dem Prozess auf GitHub gibt es auch den Gitbook.com Editor, welcher euch einen Teil der Arbeit auf Github erspart. Auch hier wird allerdings ein Github Account benötigt.

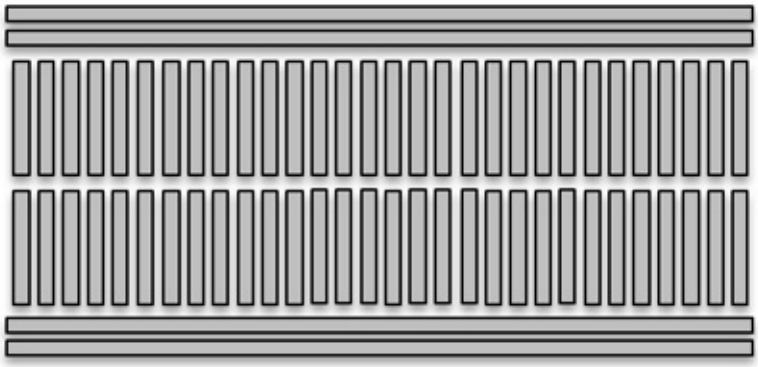
Falls all dies nicht funktioniert, könnt ihr die Anleitung im Markdown-Format gezippt an uns per Mail senden.

Vielen Dank für euren Beitrag!

Inhalt der Kiste



1. **Basisstation:** Die Grundlage für unsere Experimente bildet die Basisstation. Sie besteht aus einem Arduino Uno Mikrokontroller, dem senseBox Shield und einem Steckbrett. Letzteres besteht aus jeweils zwei vertikalen Reihen für die Plus- und Minusanschlüsse sowie zweimal 30 horizontalen Reihen mit je fünf Bohrungen, die mit `a` bis `e` bzw. `f` bis `j` beschriftet sind. Die Plus- und Minusanschlüsse sowie die fünf horizontalen Bohrungen einer Reihe sind, wie unten dargestellt, leitend miteinander verbunden.

   Das **senseBox Shield** ist eine Erweiterungsplatine für den Mikrokontroller, und wird einfach auf ihn drauf gesteckt (daher die Bezeichnung Shield). Es bietet die Möglichkeit Messungen auf einer microSD Karte abzuspeichern und zukünftige über Steckverbindungen auf der Oberseite sensoren schnell anzuschließen. Des Weiteren ist in der Platine eine Uhr eingebaut (Abkürzung RTC, engl. für Real-Time Clock), die aktiviert wird, sobald die Batterie aus dem Kleinteileset eingebaut wurde.

2. **Netzteil:** Normalerweise wird der Mikrokontroller über ein USB Kabel mit einem Computer verbunden. Hast du ihn einmal programmiert, ist diese Verbindung nicht mehr nötig, da das aktuelle Programm fortlaufend auf ihm ausgeführt wird. Das einzige was dann benötigt wird, ist eine konstante Stromversorgung.

3. **Batterypack:** Durch das Batterypack lassen sich deine Experimente mobil nutzen! Das ist vor allem sinnvoll, wenn du beispielsweise Temperaturwerte an unterschiedlichen Orten messen willst.

4. **Ethernet Shield:** Hiermit lassen sich Messwerte direkt ins Internet übertragen und beispielsweise auf einer Kartenanwendung wie der OpenSenseMap darstellen.

5. **Kabelsortiment:** Neben dem bereits erwähnten USB Kabel, beinhaltet das Sortiment noch ein Verbindungskabel für die Batteriehalterung sowie die Steckkabel die zur Verbindung zwischen Steckbrett und Mikrokontroller notwendig sind.

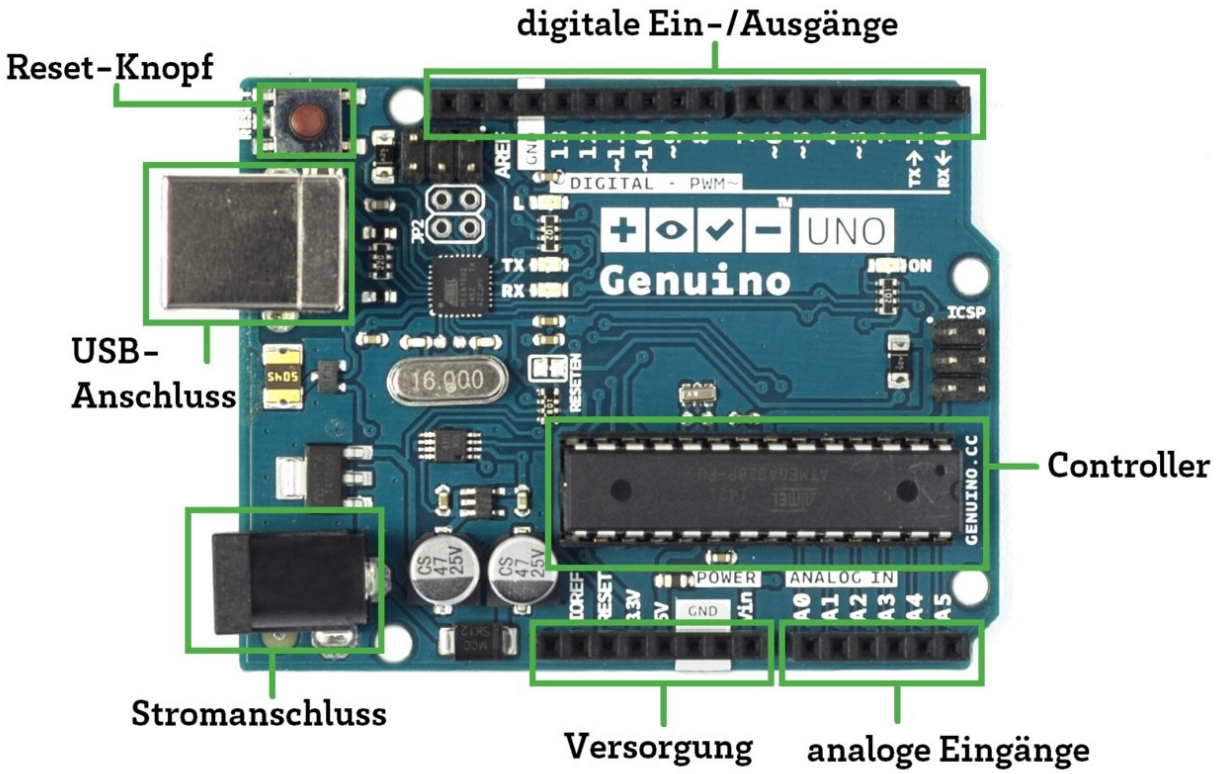
6. **Sensoren und Kleinteile:** Hier haben wir Sensoren und andere Komponenten zusammengestellt, die für deine Experimente notwendig sind. Das Kleinteilset beinhaltet die folgenden Teile:

   - Leuchtdioden (Abkürzung LED, engl. für Light-Emitting Diode)
   - Druckknöpfe
   - 470 Ω und 10 kΩ Widerstände

- RGB-LED (BL-L515)
- Lichtabhängiger Widerstand (Abkürzung LDR.engl. für Light-Dependant Resistor)
- Potentiometer
- Mikrofon (CEM-C9745JAD462P2.54R)
- Summer
- Kombinierter Temperatur- und Luftfeuchtigkeitssensor (HDC1008)
- Luftdrucksensor (BMP280)
- Infrarot-Distanzsensor (GP2YA)
- Ultraschall-Distanzsensor (HC-SR04)
- Lichtsensor (TSL45315)
- UV-Sensor (VEML6070)
- microSD-Karte mit Adapter
- Batterie für das senseBox Shield

Das Genuino Board

Der Genuino UNO ist ein Mikrocontroller-Board welches speziell für das Prototyping, also das schnelle Entwickeln von ersten funktionsfähigen Schaltungen, gedacht ist. Neben dem Genuino UNO gibt es noch eine Menge anderer Genuino Mikrocontroller-Boards, mit denen wir uns hier aber nicht weiter beschäftigen möchten.



Oben siehst du die für den Anfang wichtigsten Bauteile des Genuino UNO auf die wir nun auch kurz eigehen werden. Alle anderen Bauteile sind natürlich ebenso wichtig, aber für uns erst einmal uninteressant.

USB-Anschluss

Die selbst geschriebenen Programme müssen natürlich irgendwie auf den Genuino übertragen werden, dies funktioniert über den USB-Anschluss. Neben der Datenübertragung übernimmt der USB-Anschluss auch die Stromversorgung des Genuino, wenn keine andere Stromquelle angeschlossen ist.

Stromanschluss

Wie der Name schon sagt, lässt sich der Genuino über diesen Anschluss mit Strom versorgen. Der Genuino hat eine Betriebsspannung von 5V, er sollte allerdings bei externer Stromversorgung eine Spannung von mindestens 6 bis höchstens 20V erhalten.

Reset-Knopf

Durch das Drücken des Reset-Knopfes startet der Genuino neu. In der Praxis bedeutet dies, dass der zur Zeit gespeicherte Sketch neu gestartet wird. Der Reset-Knopf kann dann Abhilfe schaffen, wenn du dir nicht mehr sicher bist, ob dein Genuino noch ordnungsgemäß arbeitet.

Digitale Ein- und Ausgänge

Die Pins des Genuino sind quasi seine Verbindung zur Außenwelt. Die digitalen lassen sich als Ein- und Ausgänge verwenden, d.h. man kann sie zum Anschließen von digitalen Sensoren und Aktoren verwenden. Zusätzlich bieten die digitalen Pins die Möglichkeit analoge Aktoren anzuschließen. Was die Unterschiede zwischen digitalen und analogen Signalen sind, wirst du im nächsten Kapitel erfahren.

Analoge Eingänge

Genau wie die digitalen Pins dienen auch die analogen zur Kommunikation mit der Außenwelt. Sie können verwendet werden um analoge Sensoren anzuschließen und den Genuino sozusagen mit Daten zu "füttern". Der Genuino verfügt nicht über analoge Ausgänge, da diese Funktion von den digitalen Ein- und Ausgängen übernommen wird.

Versorgungs-Pins

Die Versogungs-Pins sind weder digitale noch analoge Ein- bzw. Ausgänge. Sie dienen zur Versorgung von angeschlossenen Sensoren oder Aktoren mit Strom. Weiterhin lässt sich der Gernuino auch durch spezielle Pins mit Strom versorgen.

Controller

Der Controller ist der Gehirn des Genuino, hier werden die einzelnen Prozesse ausgeführt.

Installation der Software

Zur Programmierung des Arduinos muss auf dem Computer eine spezielle Software und ein Treiber installiert werden. Je nach Betriebssystem müssen unterschiedliche Schritte zur Installation durchgeführt werden.

Die jeweils aktuellen Installationsdateien finden sich hier zum Download.

Anleitung zur Installation der Software/Treiber unter Windows

Nach dem Download müssen die Dateien nur ausgeführt und den Anweisungen gefolgt werden:

Installation der Treiber und Windows XP/Vista/7

Durch die Installation der Arduino-Software wurden bereits der Treiber auf dem Computer installiert. Allerdings kann Windows die Treiber oftmals nicht automatisch dem Arduino zuordnen, sodass wir diesen Schritt manuell durchführen müssen. Dazu sind folgende Schritte nötig:

1. Schließt den Arduino Uno über ein USB Kabel an den Computer an.
2. Der Computer wird ein neues Gerät erkennen und versuchen automatisch einen Treiber zu installieren. Diese automatische Installation ist meistens nicht erfolgreich. Wir wollen den Treiber manuell installieren.
3. Öffnet dazu die Systemsteuerung (Start --> Systemsteuerung)
4. Anschließend über System und Sicherheit --> System --> in den Geräte-Manager
5. Im Geräte-Manager sucht nach dem Arduino Uno oder unbekanntes Gerät. Mit der rechten Maustaste auf das Gerät und Treibersoftware aktualisieren auswählen.
6. Anschließend die Option Auf dem Computer nach einer Treibersoftware suchen und den Computer nach dem Ordner der Arduino-Software durchsuchen. In dem Arduino-Software Ordner liegt ein weitere Ordner mit dem Namen Driver. Wählt diesen Ordner aus und bestätigt die Auswahl.
7. Windows wird die Treiberinstallation vervollständigen und der Arduino-Uno sollte funktionieren.

Installation der Software/Treiber unter Mac OS X

Nach dem Download (s.o.) muss Datenabbild eingebunden werden und die Arduino IDE kann in den Programmordner kopiert werden. Eine Treiberinstallation unter Mac OS X ist nicht nötig.

Installation der Software unter Linux

Nach dem Download (s.o.) muss das heruntergeladene `.tar.xz` -Archiv entpackt werden. Darin liegt ein Installationskript `install.sh` , welches am einfachsten in einem Terminal über den Befehl

```
cd <pfad zum entpackten ordner>
./install.sh
```

ausgeführt wird. Alternativ ist es unter einzelnen Desktopumgebungen möglich, das Skript über einen Doppelklick im Dateimanager zu starten.

Anschließend sollte eine Verknüpfung in der jeweiligen Desktopumgebung angelegt worden sein, über welche die IDE zu starten ist.

Arduino IDE

Die Arduino IDE ist die Programmieroberfläche, die wir für unseren Mikrokontroller verwenden. Im weißen Codefenster wird der Code programmiert und im schwarzen Bereich werden dir Status- und Fehlermeldungen angezeigt. Die Arduino IDE ist in der Lage Rechtschreibfehler sowie Fehler in der Syntax zu erkennen, Fehler in der Logik des Codes werden allerdings nicht erkannt. Im oberen Bereich der Software findest du die wichtigsten Bedienelemente, die im folgenden Kurz erklärt werden.

- Mit dem Häkchen kannst du Dein Arduino Programm (auch Sketch genannt) auf Fehler überprüfen lassen. Diesen Vorgang nennt man auch kompilieren
- Mit dem Pfeil wird der Sketch auf den Arduino-Mikrokontroller hochgeladen
- Hier kannst du einen neuen Sketch anlegen, einen bereits vorhandenn Sketch öffnen oder deinen aktuellen Sketch speichern
- Mit der Lupe öffnest du den seriellen Monitor. Weitere Informationen zum seriellen Monitor findest du hier

Digitale Signale

Digitale Signale können lediglich die Werte 1 oder 0 bzw. High oder Low annehmen. Sie verwenden also nur abzählbare Elemente wie zum Beispiel Finger. Daher auch der Begriff digital, der auf das lateinische digitus, der Finger zurück geht.

Ein Arduino Programm (auch "Sketch" genannt) hat einen sehr einfachen Aufbau, der aus zwei Hauptbestandteilen besteht. Diese zwei benötigten Funktionen enthalten Blöcke von Anweisungen, welche den Programmablauf beschreiben:

```
void setup(){
  // Anweisungen
}
void loop(){
  // Anweisungen
}
```

Die `setup` -Funktion wird nur einmal beim Start des Programmes ausgeführt. In der `loop` -Funktion werden hingegen alle Anweisungen fortlaufend in einer endlosen Schleife wiederholt. Beide Funktionen sind zwingend notwendig um das Programm erfolgreich kompilieren und ausführen zu können. "Kompilieren" bezeichnet die Übersetzung des Programms in Maschinencode, welcher vom Arduino-Prozessor verstanden werden kann; dies Übernimmt die Arduino-IDE für uns.

Mit einem doppelten Schrägstrich (`//`) lassen sich Kommentare zum Programmcode hinzufügen. Es ist immer wichtig seinen Programmcode zu kommentieren, damit auch andere nachvollziehen können, was an einer bestimmten Stelle passiert.

Digitale Aktoren ansteuern

Um nun einen digitalen Aktor - beispielsweise eine LED - anzusteuern, benötigt man zwei Befehle: Der Erste steht im `setup()` , der Zweite im `loop()` . In der `setup` -Funktion wird mit dem Befehl `pinMode(13, OUTPUT);` festgelegt, dass an Pin Nummer 13 etwas angeschlossen ist, was als Ausgang (oder OUTPUT) benutzt werden soll. Die 13 kann hier durch jede andere Pin-Nummer ersetzt werden, je nachdem an welchen Arduino-Pin man den Aktor angeschlossen hat. Die zweite Funktion im `loop()` lautet `digitalWrite(13, HIGH);` . Damit wird der an Pin 13 angeschlossene Aktor mit Strom versorgt, also angeschaltet. Das Gegenstück zu diesen Befehl wäre `digital´Write(13, LOW);` um die Stromversorgung wieder zu beenden. Auch hier ist die 13 wieder durch jede andere Pinnummer ersetzbar. Der Sketch sollte also wie folgt aussehen:

```
void setup() {
  pinMode(13, OUTPUT); // Deklariere den Pin, an dem die LED
                       // angeschlossen ist, als Ausgang
}

void loop() {
  digitalWrite(13, HIGH); // Schalte die LED an
}
```

Digitale Sensoren auslesen

Dieselben Pins die wir zum Ansteuern von digitalen Aktoren genutzt haben, lassen sich auch zur Registrierung von Eingangssignalen verwenden. Digitale Eingänge können dabei genau wie digitale Ausgänge zwei Zustände annehmen; `HIGH` oder `LOW` . Damit eingehende Signale verarbeitet werden können, müssen diese in Variablen gespeichert werden.

Um digitale Signale zu speichern, eignet sich besonders eine boolesche Variable (auch `boolean` genannt), welche nur zwei Werte annehmen kann. Um nun einen digitalen Sensor auszulesen, werden ähnlich wie beim Ansteuern digitaler Sensoren zwei Befehle benötigt. Im `loop()` wird durch den Befehl `pinMode(13, INPUT);` Pin 13 des Arduino als Eingang festgelegt. Im `setup()` kann durch den Befehl `TestVariable = digitalRead(13);` ein an Pin 13 angeschlossener Sensor ausgelesen und der Wert in der zuvor angelegten Testvariable gespeichert werden. Genau wie beim Ansteuern von digitalen Aktoren steht die 13 für den verwendeten Pin und kann durch jeden anderen digitalen Pin ersetzt werden. Der Sketch sollte also wie folgt aussehen:

```
boolean TestVariable = 0;          // deklariere eine neue boolean Variable

void setup() {
  pinMode(13,INPUT);
}
void loop() {
  TestVariable = digitalRead (13); // schreibe den gelesenen Wert in die Variable
}
```

Den Inhalt der angelegten Variable kannst du dir im seriellen Monitor anzeigen lassen.

Analoge Signale

Im Gegensatz zu digitalen Signalen können analoge Signale sehr viele Werte zwischen hohem und niedrigen Pegel annehmen. Die genaue Anzahl der Werte - die Auflösung des digitalen Eingangs - liegt beim Arduino UNO bei 1024 Werten (10 bit). Beim Arduino UNO entspricht der maximale Pegel 5 V und der niedrige 0 V. Diese individuellen Spannungswerte können mit den sechs analogen Pins (`A0` - `A5`) des Arduino gemessen werden.

Aktoren analog ansteuern

Der Befehl `analogWrite()` gibt eine Spannung auf einen angegeben Pin aus. Er kann benutzt werden um beispielsweise einen Motor in verschiedenen Geschwindigkeiten laufen zu lassen.

Es ist auch möglich an auf digitalen Pins mehrere Spannungswerte auszugeben. Hierzu generiert der Arduino eine stetige Rechteckwelle mit der gewünschten Einschaltdauer und simuliert so ein analoges Signal (Pulsweitenmodulation oder PWM). Am Arduino UNO können die Pins 3, 5, 6, 9, 10 und 11 diese Funktion übernehmen. Diese sind jeweils mit einer Tilde gekennzeichnet. Die Syntax für den Befehl lautet ähnlich wie beim digitalen Gegenstück `analogWrite(pin, <Wert>)` . Der Wert kann zwischen `0` (immer aus) und `255` (immer an) liegen.

Auch hier ist darauf zu achten, den Pin zuvor als `OUTPUT` zu deklarieren.

Ein Beispielhaftes Programm könnte so aussehen:

```
int ledPin = 9;              // LED an Pin 9

void setup() {
  pinMode(ledPin, OUTPUT);   // deklariere den Pin als Ausgang
}

void loop() {
  analogWrite(ledPin, 60);  // Ansteuern der LED
}
```

Analoge Sensoren auslesen

Der Befehl `analogRead()` liest den Wert von einem analogen Pin. Die so gemessenen Werte zwischen 0 V und 5 V werden vom eingebauten 10-bit analog zu digital Konverter (ADC) in `integer` Werte zwischen 0 und 1023 umgewandelt, d.h. das Signal verfügt über eine Auflösung von 0.0049 Volt pro Wert.

Das auslesen eines Eingangs dauert etwa 0,0001 Sekunden, es können also etwa 10.000 Messungen pro Sekunde aufgenommen werden. Es bietet sich an die gemessenen Daten im seriellen Monitor anzuzeigen.

Ein beispielhaftes Programm könnte so aussehen:

```
int analogPin = 3;     // Potentiometer an Pin 3
int val = 0;           // Variable um die Messwerte zu speichern

void setup() {
  Serial.begin(9600);  //  Start des seriellen Monitors
}

void loop() {
  val = analogRead(analogPin);  // Auslesen des Sensors
  Serial.println(val);          // Ausgeben der Messerwerte
}
```

Variablen und Datentypen

Um Daten in Programmen festzuhalten, verwendet man Variablen. Eine Variable ist ein Speichercontainer, der über seinen Namen angesprochen werden kann, und in dem Daten abgelegt werden können. Auf Variablen lässt sich sowohl lesend als auch schreibend zugreifen, der Wert ist also variabel. Eine Variable hat immer einen zugeordneten Datentyp, folgende Typen sind für die Arduinoprogrammierung wichtig:

Datentypen	Bedeutung	Beschreibung
boolean	wahr o. falsch	Kann nur zwei Werte annehmen, 1 oder 0.
char	Character	Alphanumerische Zeichen (Buchstaben, Zahlen, Sonderzeichen)
byte	ganze Zahlen	ganze Zahlen von 0 bis 255
int	ganze Zahlen	ganze Zahlen von -32.758 bis 32.767
long	ganze Zahlen	ganze Zahlen von - 2 Milliarden bis 2 Milliarden
float	Fließkommazahlen	gebrochene Zahlen
String	Zeichenkette	Zeichenkette bestehend aus ASCII Zeichen
array	Variablenfeld	mehrere Werte des selben Variablentyps können gespeichert werden

Hinweis: *Beim Programmieren gibt es einige Konventionen, das heißt einige Regeln, auf die man sich geeinigt hat, um die Lesbarkeit von Programmcode zu verbessern. Eine davon ist, dass Name von Variablen immer klein geschrieben werden.*

So verwendet man die verschiedenen Datentypen

boolean

Ein Boolean kann nur zwei Werte annehmen, wahr oder falsch (`true` or `false`).

```
boolean testWert = false;
```

Die Zuweisung `= false` steht in diesem Fall für den Startwert der Variable.

char

Um beispielsweise einen Buchstaben zu speichern benötigt man den Datentyp `char` . Der Wert wird in einfachen Anführungszeichen (`'`) übergeben.

```
char testWert = 'a';
```

byte

Ein Byte speichert eine 8-bit große, vorzeichenlose Zahl von 0 bis 255.

```
byte testWert  = 18;
byte testWert2 = B10010;
```

Das `B` kennzeichnet, dass die Folgende Zahlenfolge im Binärcode geschrieben ist. `B10010` entspricht 18 im Dezimalsystem, beide Variablen enthalten also den selben Wert mit unterschiedlicher schreibweise.

int

Der Datentyp `int` speichert ganze Zahlen in einem Wertebereich von -32.768 bis 32.767.

```
int testWert = 99;
```

long

Der Datentyp `long` wird dann benötigt, wenn der Wertebereich eines Integer nicht mehr ausreicht. Es können ganze Zahlen von -2 Milliarden bis 2 Milliarden gespeichert werden.

```
long testWert = 9999999;
```

float

Um gebrochene Zahlen zu speichern benötigt man den Datentyp `float`.

```
float testWert = 2.4476;
```

String

Ein String wird folgendermaßen definiert:

```
String testWert = "Hallo Welt";
```

Im Gegensatz zu den Datentypen die ihr zuvor kennen gelernt habt, wird der Bezeichner `String` groß geschrieben. Darauf müsst ihr achten, sonst erkennt das Programm den Datentyp nicht. In den meisten Programmiersprachen gibt es primitive Datentypen und höhere Datentypen. Du erkennst sie daran ob ihre Bezeichner klein (primitiver Datentyp) oder groß (höherer Datentyp) geschrieben werden. Für unsere Anwendungen in der senseBox:edu ist es nicht notwendig zwischen primitiven und höheren Datentypen zu unterscheiden; wenn du später komplexere Anwendungen programmierst, wirst du mehr darüber lernen. Möchtest du jetzt schon mehr darüber erfahren, dann schaue doch hier nach.

array

Ein Array ist kein eigentlicher Datentyp, sondern viel mehr eine Sammlung mehrerer Varibalen des selben Typs.

```
int testArray[5] = {5, 10, 15, 20, 15};
```

Im Beispiel wird ein Array vom Typ `int` angelegt, da ganze Zahlen gespeichert werden sollen. Die 5 in eckigen Klammern hinter dem Namen der Variable legt die Anzahl an Speicherplätzen fest. Arrays auf dem Arduino haben eine feste Größe, und können nicht nachträglich vergrößert werden.

Die Speicherplätze eines Arrays werden bei 0 beginnend durchnummeriert. In einem Programm lässt sich auf die verschiedenen Speicherplätze des Arrays zugreifen, indem der Index des Speicherplatzes in eckigen Klammern hinter den Variablennamen gestellt wird:

```
Serial.print(testarray[0]); // gibt 5 aus
Serial.print(testarray[4]); // gibt 15 aus
Serial.print(testarray[5]); // erzeugt einen Fehler!
```

Lebensdauer von Variablen

Eine Variable ist immer in dem Block (innerhalb der geschweifeten Klammern) für das Programm sichtbar, in welchem die Variable deklariert wurde. Man unterscheidet zwischen globalen und lokalen Variablen. Lokale Variablen sind all diejenigen, welche innerhalb geschweifeter Klammern (meist innerhalb einer Funktion) deklariert wurden. Globale Variablen werden üblicherweise vor der `setup` -Funktion definiert und sind für das gesamte Programm sichtbar.

Hinweis: *Da globale Variablen immer sichtbar sind, verbrauchen sie auch für die gesamte Programmlaufzeit Speicherplatz. Willst Speicherplatz sparen, definiere Variablen nur dort wo du sie benötigst. Wenn du mehr über die Lebensdauer von Variablen erfahren willst, schaue [hier](https://de.wikipedia.org/wiki/Variable%28Programmierung%29#Lebensdauervon_Variablen "Wikipedia") nach.*

if/else-Bedingung

Mit `if` ist es möglich in einem Programm Entscheidungen zu fällen und den Arduino, jenachdem wie die Entscheidung ausfällt, unterschiedlichen Code ausführen zu lassen. Wenn du beispielsweise eine LED in Abhängigkeit von einem Schalter leuchten lassen möchtest würde der Code wie folgt aussehen:

```
if (digitalRead(BUTTON_PIN) == HIGH) {
  digitalWrite(LED_PIN, HIGH);
}
```

Die erste Codezeile beginnt mit einem `if` . Innerhalb der darauffolgenden Klammern wird die zu prüfende Bedingung angegeben, also in diesem Fall ob der Taster gedrückt ist. Ist diese Bedingung wahr (gibt `true` zurück), wird der in den geschweiften Klammern eingetragene Code ausgeführt.

Wie dir bestimmt aufgefallen ist, wird in der Bedingung ein Vergleichsoperator verwendet, nämlich ein doppeltes Gleichzeichen (`==`). Ein häufiger Fehler besteht darin, dass nur ein einzelnes Gleichzeichen verwendet wird. Für den Arduino steht ein einzelnes Gleichzeichen jedoch nicht "prüfe, ob gleich" sondern für "setze linken gleich rechten Wert".

Verwendung von else

Mit `else` kannst du deiner `if` -Anweisung noch eine zusätzliche Aktion hinzufügen, welche alternativ ausgeführt wird, falls die Bedingung nicht wahr ist. Wenn du also den obenstehenden Code um ein `else` ergänzt, würde der komplette Sketch so aussehen:

```
#define LED_PIN 13
#define BUTTON_PIN 7

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
}
void loop() {
  if (digitalRead(BUTTON_PIN)==HIGH){
    digitalWrite(LED_PIN, HIGH);
  } else {
    digitalWrite(LED_PIN, LOW);
  }
}
```

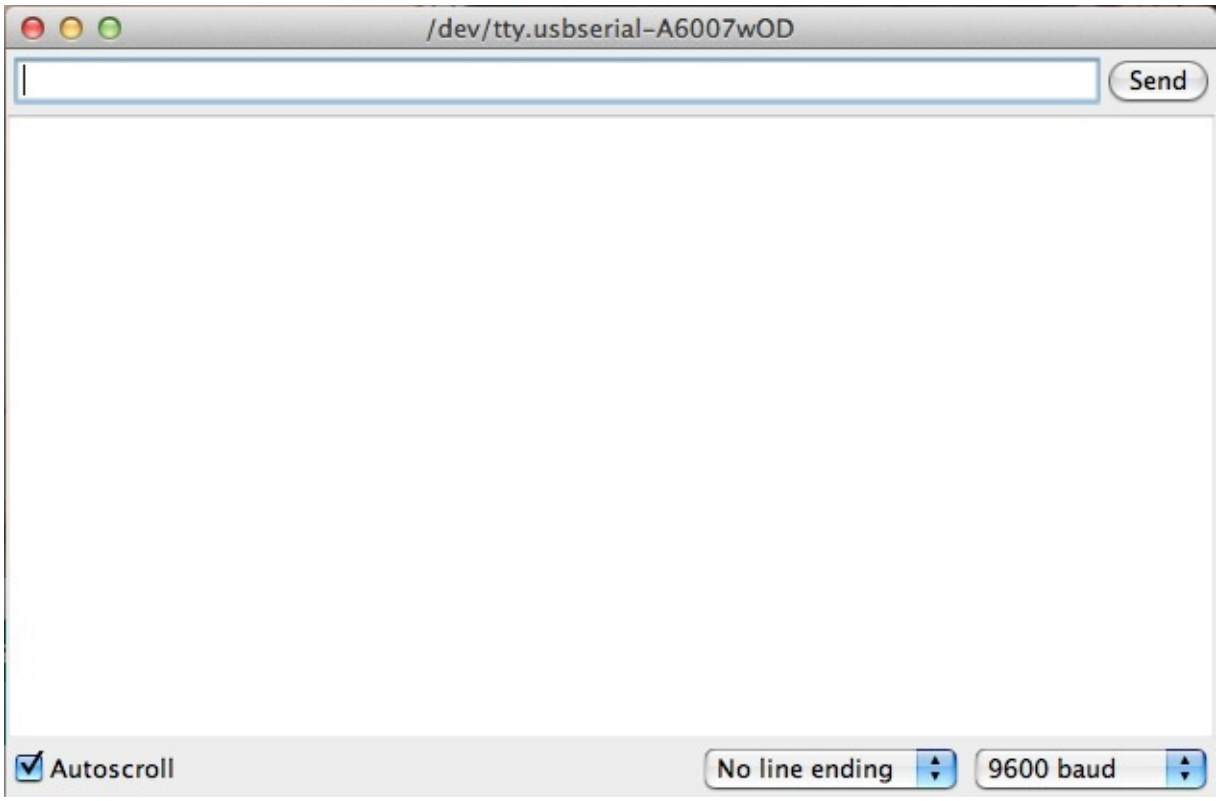
Der Serielle Monitor

Der serielle Monitor ist ein Werkzeug um Daten über die USB-Verbindung des Arduino direkt in der IDE anzeigen zu lassen und Daten von der Computertastatur an den Arduino zu übertragen.

Den seriellen Monitor starten

Um den seriellen Monitor zu starten, musst du zuerst die IDE öffnen und dann in der Symbolleiste auf das Symbol mit der kleinen Lupe klicken.

Das nun geöffnete Fenster hat oben eine Eingabezeile mit "Senden"-Schaltfläche und darunter ein Ausgabefenster. Im Ausgabefenster werden fortlaufend die neusten Ausgaben angezeigt. Wenn das Häkchen bei Autoscroll gesetzt ist, werden nur die aktuellsten Ausgaben angezeigt. Das heißt, wenn das Ausgabefenster voll ist, werden ältere Daten nach oben aus dem sichtbaren Bereich des Bildschirms geschoben um Platz für die aktuellen Ausgaben zu schaffen. Deaktiviert man die Autscroll Funktion, muss manuell über den Scrollbalken am rechten Rand gescrollt werden.



Werte auf dem seriellen Monitor ausgeben

Um sich Daten im seriellen Monitor anzeigen lassen zu können, muss dieser zuerst initialisiert werden. Dies passiert über die Funktion `Serial.begin(9600)` in der `setup()` Funktion. Der Wert `9600` definiert die Baud-Rate, also die Geschwindigkeit mit der Daten zwischen Computer und Arduino übertragen werden. Der eingetragene Wert muss immer der im seriellen Monitor unten rechts ausgewählten Geschwindigkeit entsprechen.

Um Daten an den seriellen Monitor zu senden, verwendet man die Funktionen `Serial.print()` und `Serial.println()`. Die erste Variante der Funktion gibt einfach die Daten aus, während die zweite Variante einen Zeilenumbruch am Ende einfügt.

Als ersten Versuch sollst du jetzt Text im Ausgabefenster anzeigen lassen. Um Text anzeigen zu lassen, muss dieser in Anführungszeichen in den Klammern der Funktion stehen:

```
Serial.println("senseBox rocks!");
Serial.print("senseBox ");
Serial.println("rocks!");
```

Das Beispiel sollte in je einer Zeile den Text "senseBox rocks!" ausgeben. Beachte die Verwendung von `print` und `println` !

Neben Text kann man sich im seriellen Monitor auch die Inhalte von Variablen anzeigen lassen. Dazu muss statt dem gewünschten Text der Name der jeweiligen Variable eingetragen werden:

```
String beispielvariable = "hallo welt!";
Serial.println(beispielvariable);
```

Die Verwendung von Software-Bibliotheken

Ähnlich wie mit einem Shield die Hardware des Arduino erweitert werden kann, lassen sich auch deine Sketches mit Hilfe von Software-Bibliotheken um nützliche Funktionen erweitern. Eine solche Bibliothek (engl. Library) kann prinzipiell von jedem erstellt werden. Im Normalfall stellen die Entwickler der Hardwarekomponenten Bibliotheken für ihre Produkte bereit, sodass der Nutzer sich diese Arbeit sparen kann.

In der Arduino-IDE sind bereits einige Software-Bibliotheken enthalten. Um diese in deinen Sketch einzubinden musst du auf *Sketch -> Library importieren* klicken, woraufhin sich eine Liste aller verfügbarer Bibliotheken öffnet. Wenn du in einem unserer Projekte eine Bibliothek benötigst, wird dir genau erklärt werden, welche du einbinden musst.

Fremde Bibliotheken importieren

Wenn du eigene Sensoren benutzen möchtest, kann es sein dass du die dazugehörigen Bibliotheken selbst herunterladen und importieren musst.

Als erstes muss die gewünschte Bibliothek heruntergeladen werden. Die Arduino-IDE bietet eine Funktion um externe Bibliotheken zu importieren. Dazu musst du auf *Sketch -> Library importieren -> Add Library...* klicken. Es öffnet sich ein Fenstern in welchem du zum Speicherort der heruntergeladen .zip-Datei navigieren und diese dann auswählen kannst.

Einfügen von Softwarebibliotheken in deinen Sketch

Nachdem du die Softwarebibliotheken der Arduino IDE hinzugefügt hast musst du diese noch in deinen Sketch einbinden. Das Einbinden erfolgt als erstes im Programmablauf, noch bevor die `setup` -Funktion aufgerufen wird. Mithilfe der `#include` Direktive wird eine Bibliothek geladen. Für das Ethernet-Shield wird zum Beispiel eine Library benötigt, die über `#include <Ethernet.h>` eingebunden wird.

Zum besseren Verständis kannst du dir einmal folgenden Beispielcode anschauen:

```
#include <Ethernet.h> // Einbinden der Bibliothek

int Sensor;           // Deklaration von Variablen


void setup() {
  Ethernet.begin();   // diese Funktion ist nur durch das Laden der
                      // Ethernet.h Bibliothek verfügbar!
}

void loop() {
  //Anweisungen, die fortlaufend ausgeführt werden
}
```

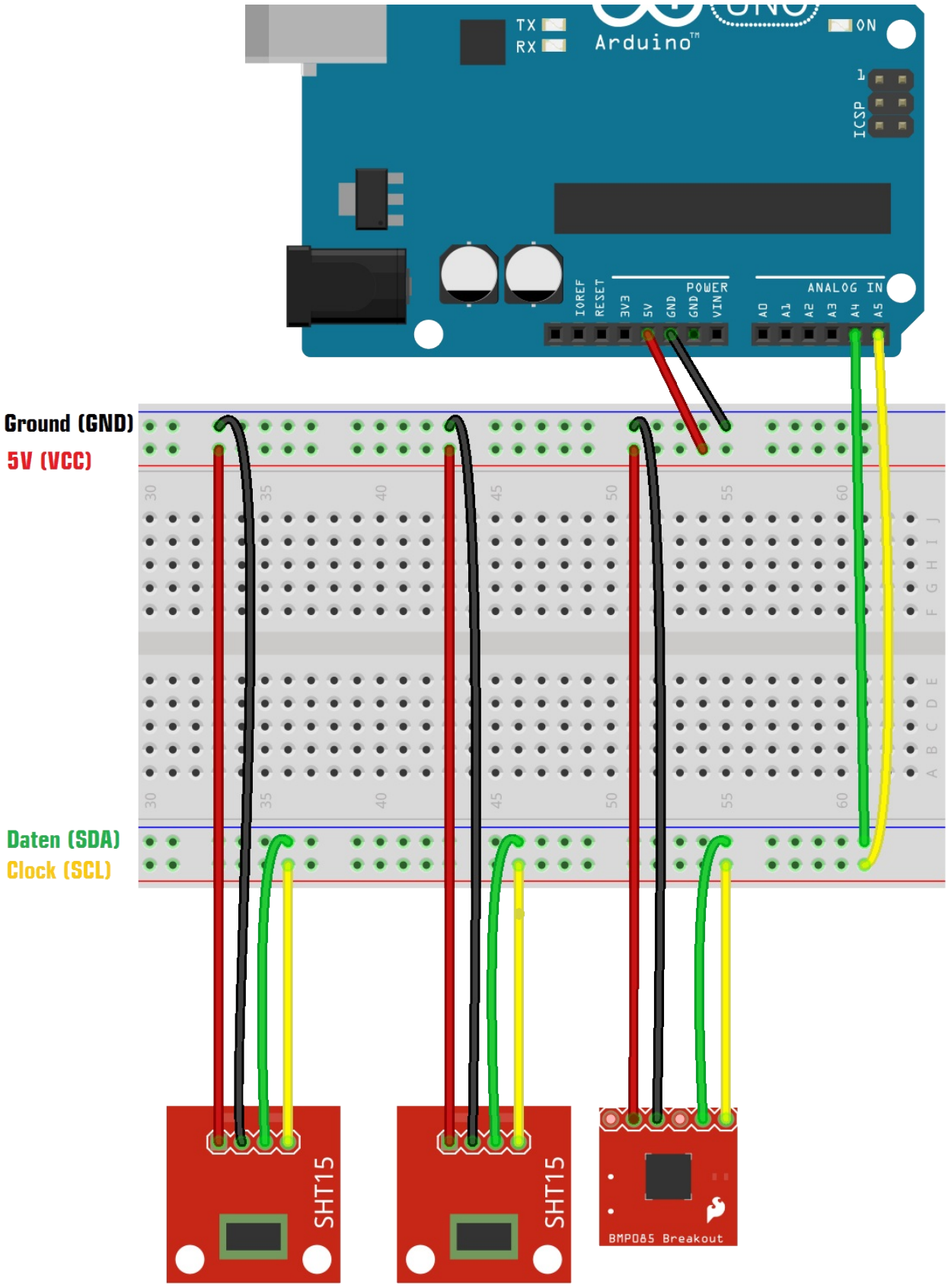
Der serielle Datenbus

Der Arduino kann über einen Datenbus mit anderen Geräten kommunizieren. Ein Datenbus beschreibt ein System, über das zwei oder mehr Geräte Daten auf eine geordnete Art und Weise austauschen können. Bei unserem Arduino wäre das zweite Gerät fast immer ein Sensor, bzw ein Aktor.

Der I²C-Bus

Der I²C-Bus ist ein einfach zu verwendender Datenbus um Daten zu übermitteln. Hierbei werden die Daten zwischen dem Arduino und dem anderen Gerät durch zwei Kabel übertragen, die als `SDA` und `SCL` bezeichnet werden. Die als `SDA` (serial data) bezeichnete Leitung ist die Datenleitung, über welche die eigentlichen Daten übermittelt werden. Die `SCL` (serial clock) Leitung wird auch Taktleitung genannt und gibt die Taktfrequenz vor. Am Arduino findest du die beiden Anschlüsse als `A4` (SDA) und `A5` (SDC).

Wenn mehrere I²C Geräte an den Arduino angeschlossen werden sollen, wird dies über eine Reihenschaltung umgesetzt. Das SDA Kabel am ersten Sensor würde also auf der selben Reihe des Breadboards zum nächsten Sensor verlängert:



Benutzt man den I²C-Bus auf dem Arduino, gilt der Arduino immer als Master-Gerät und alle anderen Geräte am Bus als Slave. Jeder Slave hat seine eigene Adresse in Form einer Hexadezimalzahl, mit welcher er eindeutig angesprochen werden kann. Für gewöhnlich bringt jedes Gerät einen Bereich von Busadressen mit, welche man verwenden kann. Die jeweiligen Adressen können im Datenblatt des Herstellers nachgeschaut werden, und stehen auch bei uns im Glossar.

Die `Wire.h` Bibliothek

Um den I²C-Bus verwenden zu können muss die Wire-Bibliothek (welche in der Arduino IDE enthalten ist) verwendet werden. Die Bibliothek wird durch den Befehl `#include <Wire.h>` noch vor dem `setup()` eingebunden und im `setup()` mit dem Befehl `Wire.begin();` gestartet.

Die Daten werden über den I²C-Bus Byte für Byte übertragen. Um ein Byte zu senden sind drei Befehle erforderlich:

1. `Wire.beginTransmission(Adresse);`

   Mit diesem Befehl wird die Kommunikation eingeleitet. Adresse steht hier für die jeweilige hexadezimale Bus-Adresse des Slave-Gerätes.

2. `Wire.write(Daten);`

   Dieser Befehl sendet ein Byte Daten vom Arduino an der zuvor angesprochene Gerät.

3. `Wire.endTransmission();`

   Mit diesem Befehl wird die Übertragung beendet.

Um nun Daten von einem I²C-Gerät anzufordern benötigt man vier Befehle:

1. `Wire.beginTransmission(Adresse);`
2. `Wire.requestFrom(Adresse, X);`

   Bei diesem Befehl steht das X für die Anzahl an Bytes, die angefordert werden.

3. `incoming = Wire.read();`

   Mit diesem Befehl speichert man die ankommenden Daten in einer Variable.

4. `Wire.endTransmission();`

   Mit diesem Befehl wird die Übertragung beendet.

Kommentare im Quelltext

Das Kommentieren von Quellcode ist leider ein Thema das stiefmütterlich behandelt wird. Den Nutzen von guten Kommentaren erkennt man häufig erst dann, wenn man versucht sich durch fremden Quelltext durchzuwuseln oder wenn man ein eigenes Programm nach langer Zeit wieder 'ausgräbt'. Kommentare werden vom Compiler nicht ausgewertet und beeinflussen den Ablauf des Programms damit nicht. Text und Programmteile die auskommentiert sind, erkennt man daran, dass sie grau gefärbt sind.

Einzeilig

Einzeilige Kommentare finden sich oft im Quelltext. Sie dienen dazu bestimmte Befehle oder Konstrukte zu erklären. Ein einzeiliger Kommentar wird durch zwei `//` gekennzeichnet.

```
// Ich bin ein Kommentar
int led = 13;  // Die variable LED bekommt den Wert 12
```

Mehrzeilige Kommentare

Mehrzeilige Kommentare stehen oft zu beginn eines Programms oder vor einer Methode. Sie beginnen mit `/*` und enden mit `*/`. Man kann sie außerdem dazu verwenden Teile eines Programms auszukommentieren. Etwa wenn man einen Fehler hat und überprüfen möchten in welchem Programmteil er liegt.

```
/*
 *
 * Ich bin ein mehrzeiliger Kommentar
 * Ich kann zum Beispiel beschreiben, welchen Zweck ein ganzes Programm
 * oder eine einzelne Methode hat.
 *
 * übrigens:
 * <- diese Sterne werden zwar automatisch erzeugt, sind aber nicht unbedingt notwendig.
 *
 */
```

Wie viele Kommentare braucht ein Quellcode?

Das ist eine Frage auf die es keine klare Antwort gibt. Es gibt Programmierer, die erwarten, dass jede Zeile Quelltext kommentiert wird. Dies ist bei unseren einfachen Programmen nicht notwendig. Grundsätzlich sollten mindestens folgende Programmteile Kommentiert sein:

- Ein Kommentar zu Beginn, welcher den Zweck des Programms beschreibt.
- Jede Methode muss kommentiert sein und hier insbesondere die Eingabeparameter und eventuelle Rückgaben.
- Mathematisch oder logisch anspruchsvolle Befehle oder besondere 'Kniffe' die sich der Programmierer überlegt hat.

Tipp: *Wenn du dich weiter mit dem Thema beschäftigen möchstest, dann schau doch mal hier*

DIY Umweltstation

In diesem Projekt erfahrt ihr, wie man eine senseBox Umweltstation aufbaut. Am Ende wird die Messung diverser Umweltphänomene wie Temperatur, Luftfeuchte, Helligkeit und Luftdruck, sowie die Veröffentlichung der Daten auf der openSenseMap möglich sein!

Dieses Projekt ist das umfangreichste, weshalbt es in mehrere Kapitel aufgeteilt wurde. In jedem neuen Kapitel wird ein zusätzlicher Baustein eingeführt, bis eine vollwertige - den Funktionen der senseBox:home ähnliche - Umweltstation gebaut wurde!

DIY - Experimente mit Licht

Wenn du fern siehst, das Radio anschaltest, mit deinem Smartphone eine Nachricht schreibst oder Essen in der Mikrowelle warm machst, nutzt du dabei elektromagnetische Energie. Heutzutage sind alle Menschen ständig auf diese Energie angewiesen. Ohne sie würde das Leben in modernen Städten völlig anders sein, als du es kennst.

Ziele der Station

In dieser Station verwendest du einen Lichtsensor, um die Beleuchtungsstärke des sichtbaren Lichts in Lux zu erfassen.

Materialien

- Lichtsensor `TSL 45315`

Grundlagen

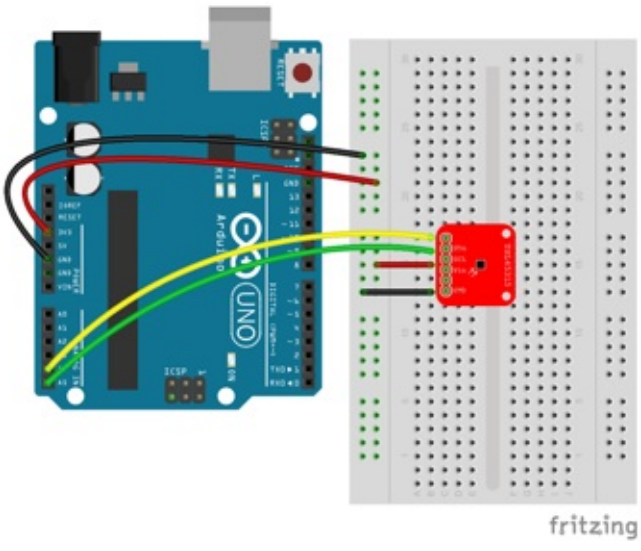
Elektromagnetische Energie bewegt sich in Wellen durch den Raum. Ihr Spektrum reicht von sehr langen Radiowellen bis hin zur sehr kurzwelligen Gammastrahlung. Das menschliche Auge kann dabei nur einen sehr kleinen Teil dieses Spektrums wahrnehmen: das sichtbare Licht. Unsere Sonne ist dabei die Quelle der Energie über das gesamte Spektrum hinweg. Die Atmosphäre der Erde schützt uns davor, einem zu hohen Maß an Strahlung ausgesetzt zu werden, die für uns lebensgefährlich werden könnte.

Für uns ist die Intensität des sichtbaren Lichts besonders interessant. Um die sog. Beleuchtungsstärke des einfallenden Lichts im sichtbaren Teil des Spektrums zu messen, wird die Einheit Lux verwendet. Sie gibt das Verhältnis der Helligkeit in Lumen pro Quadratmeter an. Bei einem hellen Sonnentag beträgt sie über 100.000 Lux, in einer Vollmondnacht hingegen nur etwa 1 Lux.

Für diese Messung benutzt du im Folgenden den I2C Sensor TSL45315 von AMS-TAOS. Im Datenblatt des Sensors sieht man, dass seine Empfindlichkeit auf den sichtbaren Teil des Lichtspektrums angeglichen ist, der ungefähr zwischen 400 und 700 nm liegt.

Laut dem Datenblatt hat dieser Sensor eine Reichweite von 2 bis 200.000 Lux, bei einer Auflösung von 3 Lux. Des Weiteren muss der Sensor mit 3,3V betrieben werden.

Aufbau



Grundlagen

Zur manuellen Programmierung benutzt du dieses Mal nur die Wire-Bibliothek. Am Anfang brauchst du ein paar Konstanten, die mit der Direktive `#define` definiert werden. Anders als bei Variablen belegen sie einen festen Platz im Speicher, der sich nur auslesen, aber nicht beschreiben lässt. In unserem Falle soll die Busadresse sowie die folgenden Registeradressen des Sensors gespeichert werden.

ADDRESS	RESISTER NAME	R/W	REGISTER FUNCTION	RESET VALUE
—	COMMAND	W	Specifies register address	0x00
0x00	CONTROL	R/W	Power on/off and single cycle	0x00
0x01	CONFIG	R/W	Powersave Enable / Integration Time	0x00
0x04	DATALOW	R	ALS Data LOW Register	0x00
0x05	DATAHIGH	R	ALS Data HIGH Register	0x00
0x0A	ID	R	Device ID	ID

Diese Register werden zur Konfiguration und Kommunikation benötigt:

```
#include <Wire.h>
#define I2C_ADDR     (0x29)
#define REG_CONTROL   0x00
#define REG_CONFIG    0x01
#define REG_DATALOW   0x04
#define REG_DATAHIGH  0x05
#define REG_ID        0x0A
```

In der Setup-Funktion soll nun eine Verbindung zu dem Sensor hergestellt werden und dem Kontrollregister der Befehl zum Hochfahren gegeben werden:

```
Wire.begin();
Wire.beginTransmission(I2C_ADDR);
Wire.write(0x80 | REG_CONTROL);
Wire.write(0x03); //Power on
Wire.endTransmission();
```

Als nächstes legen wir eine Belichtungszeit von 400 ms fest:

```
Wire.beginTransmisson(I2C_ADDR);
Wire.write(0x80 | REG_CONFIG);
Wire.write(0x00); //400 ms
Wire.endTransmission();
```

Um die Belichtungszeit zu verändern, kann man den entsprechenden Wert von `0x00` in `0x01` oder `0x02` ändern, um die Belichtungszeit auf 200 bzw. 100 ms im Konfigurationsregister des Sensors zu reduzieren. In der Loop-Funktion geben wir nun den Befehl zum Start der Messroutine und lassen uns vom Sensor die Daten senden, die für die Berechnung der Beleuchtungsstärke benötigt werden:

```
Wire.beginTransmisson(I2C_ADDR);
Wire.write(0x80 | REG_DATALOW);
Wire.endTransmission();
Wire.requestFrom(I2C_ADDR, 2); //2 Bytes anfordern
uint16_t low = Wire.read();
uint16_t high = Wire.read();
```

Falls der Sensor noch Daten sendet, sollten diese danach abgefangen werden, um Fehler im nächsten Durchgang zu vermeiden.

```
while(Wire.available()){
    Wire.read();
}
```

Hinweis: *Ähnlich wie bei unserer loop-Funktion führt eine while-Schleife die Anweisungen in den geschweiften Klammern immer wieder aufs Neue aus. Abgebrochen wird sie erst dann, wenn die Bedingung nicht mehr erfüllt wird.*

Zu guter Letzt nutzt du die ausgelesenen Datenbytes, um Beleuchtungsstärke in Lux auszurechnen. Im Datenblatt findet sich die dazu passende Formel:

```
uint32_t lux;
lux = (high << 8) | (low << 0);
lux = lux * 1; //Multiplikator für 400ms
```

Um diese Formel auf eine Belichtungszeit von 200 oder 100ms anzupassen, musst du nur den Multiplikator auf 2 bzw. 4 erhöhen.

Aufgabe 1

Füge den Code aus dieser Lektion zusammen und ergänze eine Funktion um die Daten im Seriellen Monitor ausgeben zu lassen.

Aufgabe 2

Ändere die Belichtungszeit des Sensors und vergleiche danach die Ergebnisse der Messungen.

Tipp: *Vergiss nicht, neben der Belichtungszeit im Konfigurationsregister auch die Berechnung des Lux-Wertes entsprechend anzupassen.*

DIY - UV-Licht Sensor

Ziele der Station

In dieser Station verwenden wir einen UV-Lichtsensor, um die Intensität des UV-Lichts in Mikrowatt je Quadratzentimeter (µW / cm²) zu erfassen. Anschließend wollen wir den Wert in den UV-Index umrechnen.

Materialien

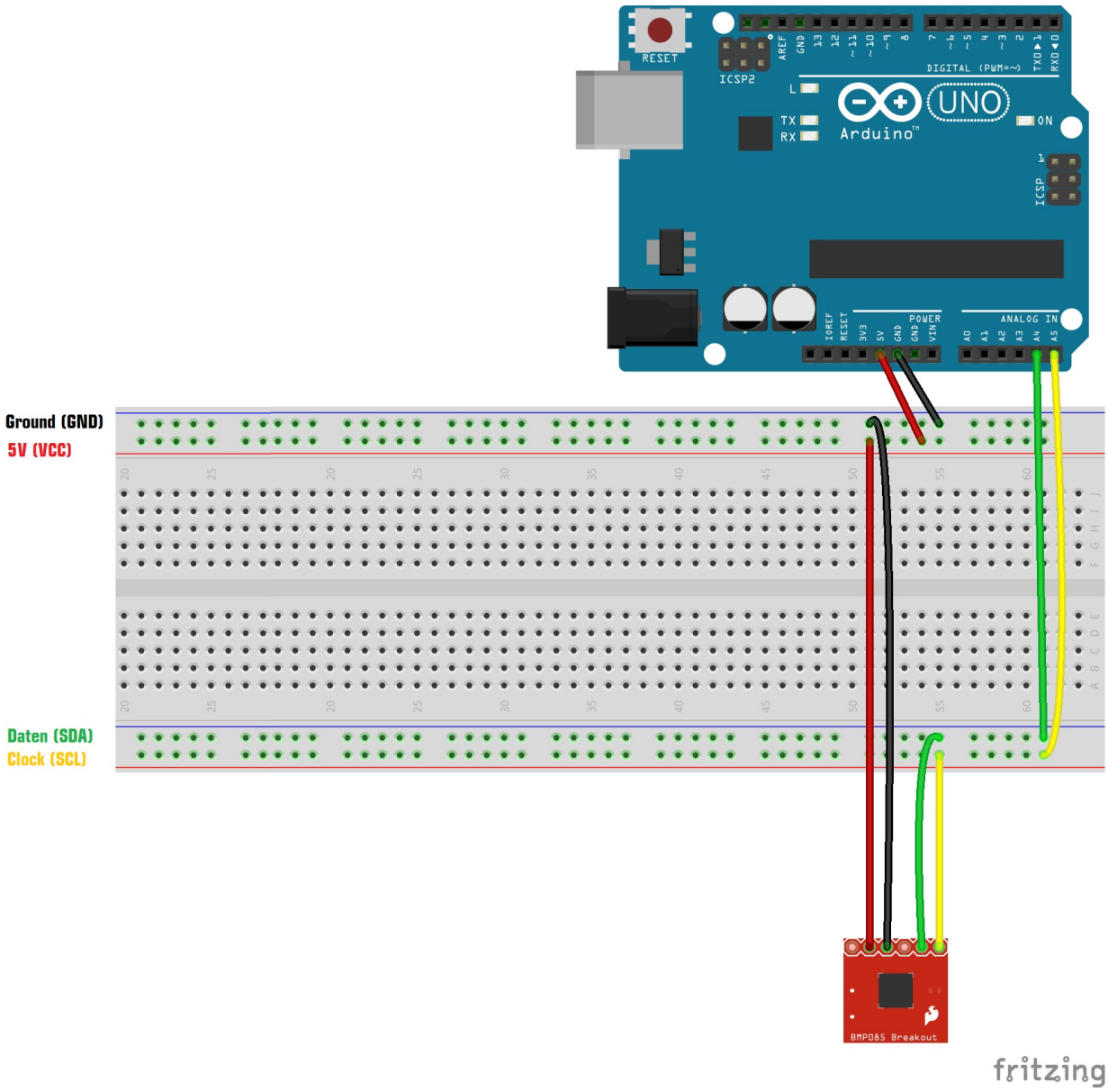
- UV-Licht Sensor `VEML6070`

Grundlagen

Ultraviolettstrahlung (UV-Licht) ist für den Menschen unsichtbare elektromagnetische Strahlung mit einer Wellenlänge, die kürzer ist als die sichtbaren Lichtes, aber länger als die der Röntgenstrahlung. UV-Licht umfasst die Wellenlängen von 100 nm bis 380 nm. Wegen der Absorption in der Erdatmosphäre - insbesondere in der Ozonschicht - dringt nur wenig UV-B-Strahlung (100 - 300 nm) bis zur Erdoberfläche vor. UV-A-Strahlung (300 - 380 nm), welche weniger gefährlich für die menschliche Haut ist, wird weniger durch die Atmosphäre absorbiert.

UV-Lichtintensität wird in Mikrowatt je Quadratzentimeter (µW / cm²) gemessen. Unser Sensor misst im Bereich von ca. 300 - 400 nm, nimmt also nur UV-A Strahlung auf (für genauere Angaben beachte das Datenblatt).

Aufbau



Schließe den Sensor an den Arduino an, wie es in der Grafik dargestellt ist.

Um über den I²C Bus auf den Sensor zugreifen zu können, müssen wir die Bibliothek `Wire.h` importieren und die Adresse des Sensors als Konstante definieren. Wir benötigen weitere Konstanten um das Auslesen des Sensors zu ermöglichen, sowie einen Referenzwert um die Messwerte in den UV-Index umrechnen zu können:

```
#include <Wire.h>

#define I2C_ADDR_UV 0x38
// Integrationszeit
#define IT_1_2 0x0 //1/2T
#define IT_1   0x1 //1T
#define IT_2   0x2 //2T
#define IT_4   0x3 //4T

// Referenzwert: 0,01 W/m2 ist äquivalent zu 0.4 als UV-Index
float refVal = 0.4;
```

Jetzt können wir unser Programm in der `setup` -Funktion konfigurieren.

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

  Wire.begin();
  Wire.beginTransmission(I2C_ADDR_UV);
  Wire.write((IT_1<<2) | 0x02);
  Wire.endTransmission();
  delay(500);
}
```

In der `loop` -Funktion schreiben wir unser Hauptprogramm:

```
void loop() {
  byte msb=0, lsb=0;
  uint16_t uv;

  Wire.requestFrom(I2C_ADDR_UV+1, 1); // MSB (erstes byte am sensor lesen)
  delay(1);
  if(Wire.available())
    msb = Wire.read();

  Wire.requestFrom(I2C_ADDR_UV+0, 1); // LSB (zweites byte am sensor lesen)
  delay(1);
  if(Wire.available())
    lsb = Wire.read();

  uv = (msb<<8) | lsb;   // bytes durch Bitshift zu einer Zahl verbinden

  Serial.print("uW je cm²: ");
  Serial.println(uv, DEC);            // Ausgabe als 16bit integer
  Serial.print("UV-Index: ");
  Serial.println(getUVI(uv));

  delay(1000);
}
```

> ***Achtung:*** *Falls ihr das Programm kompiliert bevor ihr die Methode* `getUVI()` *programmiert (siehe unten), wird euch eine Fehlermeldung ausgegeben.*

Da im Alltag häufig mit dem UV-Index gearbeitet wird, wollen wir nun eine Methode schreiben, welche uns den Messwert in einen UV-Index umrechnet:

```
/*
 * getUVI()
 * erwartet den Messert des UV-Sensors als Eingabeparameter
 * und gibt den entsprechenden Wert auf dem UV-Index zurück
 */
float getUVI(int uv){
  float uvi = refVal*(uv*5.625)/1000;
  return uvi;
}
```

Aufgabe

DIY - Temperatur und Luftfeuchtigkeit

Damit wir täglich den Wetterbericht im Internet, im Fernsehen, in der Zeitung oder in Apps sehen können, werden nicht nur Satellitendaten ausgewertet. Auch Daten von Wetterstationen am Boden spielen eine wichtige Rolle bei der Vorhersage. Aber wie funktioniert die Messung und Darstellung von Temperatur- und Luftfeuchtigkeitswerten?

Voraussetzungen

- Die Verwendung von Software-Bibliotheken
- Der seriellen Datenbus
- Der serielle Monitor

Ziele der Station

In dieser Station beschäftigen wir uns mit dem Temperatur- und Luftfeuchtigkeitssensor der senseBox, dem HDC1008.

Materialien

- kombinierter Temperatur und Luftfeuchtigkeitssensor `HDC1008`

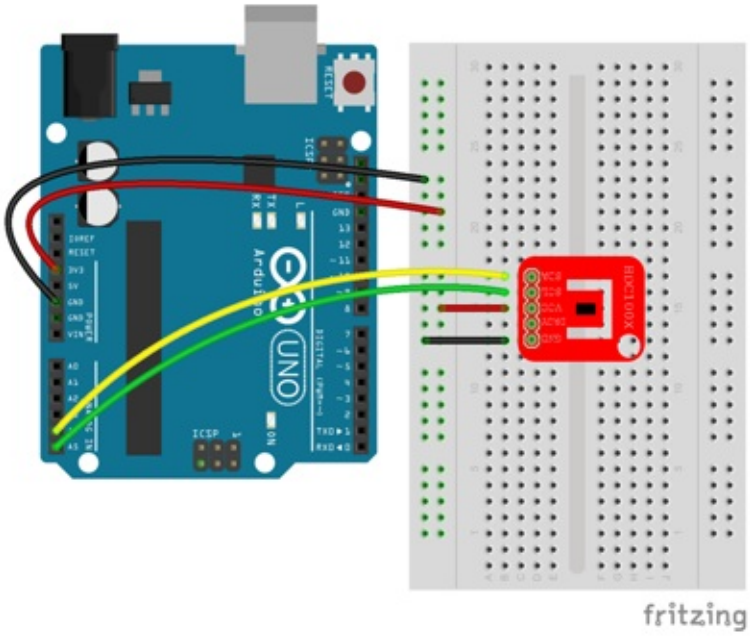
Grundlagen

Der `HDC1008`, aus der Serie HDX100X von Texas Instruments, ist ein kombinierter Temperatur- und Luftfeuchtigkeitssensor. Der Sensor kann die Luftfeuchtigkeit von 0% bis 100%, sowie die Temperatur von -40°C bis 125°C bei einer Genauigkeit von ±4% bzw. von ±0,2°C messen.

Die Kommunikation des Sensors mit dem Mikrokontroller läuft über den seriellen Datenbus I²C. Anders als bei einfachen digitalen oder analogen Eingängen, können an den Datenbus mehrere I²C-Geräte (wie z.B. Sensoren oder Displays) parallel geschaltet werden. Jedes Gerät hat dabei eine eindeutige Kennung, damit der Datenbus jedes Einzelne davon zuordnen und separat ansprechen kann.

Aufbau

Steckt den Schaltkreis wie ihr ihn unten in der Grafik seht.



Programmierung

Bevor wir mit der Programmierung starten können, müssen wir die HDC100X Bibliothek hinzufügen. Dies solltest du eigentlich bereits mit den ersten Schritten getan haben, falls nicht kannst du im Kapitel zu Softwarebibliotheken nachlesen.

Um alle zusätzlichen Funktionen der Bibliothek nutzen zu können, muss sie an oberster Stelle in eurem Programm mit der Direktive `#include` eingebunden werden. In unserem Fall brauchen wir neben der HDC100X Bibliothek noch die Wire-Bibliothek (Standarderweiterung für I²C Geräte), also steht in den ersten beiden Zeilen:

```
#include <Wire.h>
#include <HDC100X.h>
```

Hinweis: *Anders als bei regulären Befehlen, steht am Ende der include Direktive* ***KEIN*** *Semikolon.*

In allen darauf folgenden Zeilen können nun die Funktionen der Bibliotheken genutzt werden. Als erstes muss eine Verbindung zur Sensoradresse angegeben werden. Bei diesem Sensor lautet die Adresse `0x43` (s. Datenblatt).

```
HDC100X HDC(0x43);
```

Durch diesen Befehl hast du nun eine Instanz `HDC` des Sensors vom Typ `HDC100X` angelegt. Jetzt muss dieser Sensor in der `setup` -Funktion wie folgt gestartet werden:

```
HDC.begin(HDC100X_TEMP_HUMI, HDC100X_14BIT, DISABLE);
```

Die Argumente der `begin` -Funktion geben dabei an, dass Temperatur und Luftfeuchte gemessen werden, jeweils in einer Auflösung von 14 Bit, und dass das Heizelement des Sensors deaktiviert werden soll. Nachdem du den Sensor, wie oben beschrieben, initialisiert hast, kannst du zwei Befehle in der loop-Funktion nutzen, um einen Temperatur- bzw. Feuchtigkeitswert ausgeben zu lassen:

```
HDC.getHumi();
HDC.getTemp();
```

Hinweis: *Beim Speichern der Messwerte sollten die Variablen den gleichen Datentypen haben wie die Rückgabewerte der Messfunktionen. In unserem Fall sind das beides float Werte.*

Aufgabe 1

Baue die oben beschriebene Schaltung nach und versuche den HDC1008 auszulesen und Dir die gemessenen Daten im seriellen Monitor anzeigen zu lassen.

Hinweis: *Orientiere dich an dem Beispiel aus der HDC100X-Bibliothek.*

Datenlogger

In der folgenden kurzen Anleitung wird beschrieben, wie du Messwerte verschiedener Sensoren auslesen und auf SD-Karte speichern kannst.

Materialien

- Arduino Uno
- senseBox-Shield
- microSD-Karte
- Sensor(en) (nach Wunsch)

Grundlagen

Um Daten auf SD-Karte zu speichern stellt das senseBox-Shield einen microSD-Karten Slot bereit. Die Sensordaten werden ausgelesen und anschließend als .csv (comma-seperated-value) Datei gespeichert.

Vorlage

Die Folgende Code Vorlage beinhaltet alle Bestandteile um Daten auf SD-Karte zu speichern. Nun muss nur das Auslesen des Sensors in die Code-Vorlage integriert werden und schon können die Messwerte gespeichert und ausgelesen werden.

```
/*
  senseBox Datenlogger Vorlage
  Das Auslesen der jeweiligen Sensoren muss in der loop() stattfinden und individuell angepasst werden
*/

#include <SPI.h> // wichtige Libraries für das Speichern von Daten auf SD-Karte
#include <SD.h>

const int chipSelect = 4;

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ;
  }


  Serial.print("Initialisiere SD-Karte");

  // Überprüfe ob eine SD-Karte eingelegt ist.
  if (!SD.begin(chipSelect)) {
    Serial.println("Karte nicht gefunden");
    return;
  }
  Serial.println("Karte erfolgreich initialisiert");

}

void loop() {

  //Auslesen der Sensoren und schreiben der Daten auf SD-Karte
  File dataFile = SD.open("datalog.txt", FILE_WRITE);

    float Messwert = // auslesen des Sensors
    dataFile.print(Messwert); //Speichern der Messwerte in der geöffneten Datei
    dataFile.println(";"); //Trennzeichen für die .csv Datei

  // Wenn die Datei geöffnet ist
  if (dataFile) {
    dataFile.close();
  }
  // Falls die Datei nicht geöffnet werden kann, soll eine Fehlermeldung ausgegeben werden
  else {
    Serial.println("Fehler beim Öffnen!");
  }
  delay(1000);
}
```

Das Auslesen des Sensors sollte in der void loop () nach dem öffnen der Datei stattfinden. Die Informationen zum Auslesen der Sensoren finden sich in den Datenblättern und Beispielcodes der Hersteller und müssen individuell angepasst werden. Die jeweiligen Messwerte werden mit dem Befehl `dataFile.print(Messwert);` in eine Zeile der CSV-Datei geschrieben. Der Zusatz `ln` erzeugt einen Zeilenumbruch. Über den Befehl `delay(Mikrosekunden)` kann das Mess- und Speicherintervall angepasst werden.

Zur Veranschaulichung findest du unten einen Sketch, der den HDC100x ausliest und die Messwerte auf SD-Karte speichert.

```
/*
  senseBox HDC100x Datenlogger
  Anschluss des HDC100x über I2C an den Arduino
  SDA - A4, SCL - A5, VCC - 5V, GND - GND
*/

#include <SPI.h> //wichtige Libraries für das Speichern von Daten auf SD-Karte
#include <SD.h>
#include <Wire.h> //I2C Library
#include <HDC1000.h> //Library für den HDC100x

HDC1000 mySensor(0x43); //IC2 Adresse des HDC1000

const int chipSelect = 4; //

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ;
  }

  Serial.print("Initialisiere SD-Karte");

  // Überprüfe ob eine SD-Karte eingelegt ist.
  if (!SD.begin(chipSelect)) {
    Serial.println("Karte nicht gefunden");
    return;
  }
  Serial.println("Karte erfolgreich initialisiert");

 mySensor.begin();

}

void loop() {

  //Auslesen der Sensoren und schreiben der Daten auf SD-Karte
  File dataFile = SD.open("datalog.txt", FILE_WRITE);

    float Temp = mySensor.getTemp(); //Auslesen der Temperatur
    dataFile.print(Temp); //Speichern der Temperatur
    dataFile.print(";");
    Serial.println(mySensor.getTemp()); //Anzeige der Temperatur im seriellen Monitor
    float Humi = mySensor.getHumi();
    dataFile.print(Humi);
    dataFile.println(";");
    Serial.println(mySensor.getHumi());

  // Wenn die Datei geöffnet ist, speichere die Änderungen
  if (dataFile) {
    dataFile.close();
  }
  // Falls die Datei nicht geöffnet werden kann, soll eine Fehlermeldung ausgegeben werden
  else {
    Serial.println("Fehler beim Öffnen!");
  }
  delay(1000); //Intervall des Speichern und Auslesen
}
```

Messgerät zur Erfassung der Temperatur, Luftfeuchtigkeit und Luftdruck

Bei diesem Messgerät werden die Temperatur, Luftfeuchtigkeit und der Luftdruck als CSV-Datei auf SD-Karte gespeichert. Die Sensoren werden über I²C mit dem Arduino verbunden. Der Code befindet sich hier zum Download.

DIY - Datenupload von zwei Sensoren zur OpenSenseMap

Wenn wir eine senseBox aufgebaut haben, wäre es doch schön die gewonnenen Daten von jedem Ort aus abrufen zu können. Dazu gibt es die OpenSenseMap (OSeM), welche diverse Sensordaten sammelt und auf einer Karte visualisiert. Über den Arduino Ethernet-Shield können wir unsere senseBox ans Internet anbinden und die Daten zur OSeM hochladen.

Ziele der Station

In dieser Station wird beispielhaft die Integration eines Sensors in die OpenSenseMap gezeigt, sodass die gewonnenen Daten online verfügbar sind.

Materialien

- Luftdruck- & Temperatursensor `BMP280`
- Ethernet Shield
- Ethernetkabel

Grundlagen

Ethernet-Shield

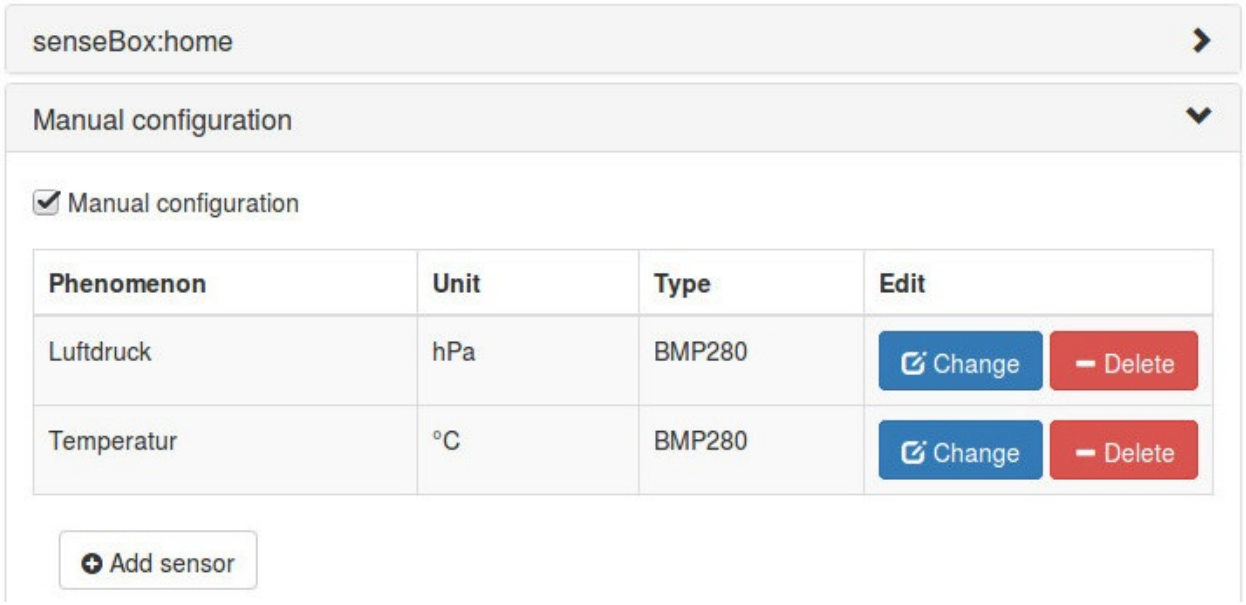
Ein Shield bezeichnet eine aufsteckbare Platine für den Arduino, welche den Microcomputer kompakt um Funktionalität erweitert. Shields werden einfach in die Pins des Arduino aufgesteckt und stellen diese Pinbelegung wieder selbst bereit.

Im Kasten der senseBox:edu findest du einen *Ethernet-Shield* (rote Platine), welcher den Arduino mit einem Netzwerkanschluss austattet, sowie einen *senseBox-Shield* (grüne Platine), auf welchem eine Echtzeituhr (RTC) und weitere Anschlüsse verbaut sind.

Da unser Ethernet-Shield eine modifizierte Variante des offiziellen Arduino-Shields ist, funktioniert die mit der Arduino-IDE mitgelieferte `Ethernet.h` Bibliothek nicht. Für dieses Projekt solltest du unsere Versionen der Bibliotheken verwenden. Wenn ihr diese Bibliothek installiert habt, wird die Arduino-IDE möglicherweise die Ethernet-Bibliothek updaten wollen, dies solltet ihr ablehnen.

OpenSenseMap

Um Daten auf die OSeM hochzuladen, muss dort zuerst eine senseBox registriert werden. Achte bei der Registrierung darauf, die manuelle Konfiguration zu wählen, und dort 2 Sensoren hinzuzufügen (siehe Screenshot).



Nach der Registrierung erhältst du per Email einen Schlüssel, über den die Messungen deiner senseBox zugeordnet werden. Zusätzlich erhältst du einen Arduino-Sketch, in welchem die Übermittelung der Messdaten schon implementiert ist, aber noch mit der Ansteuerung deines Sensors erweitert werden muss.

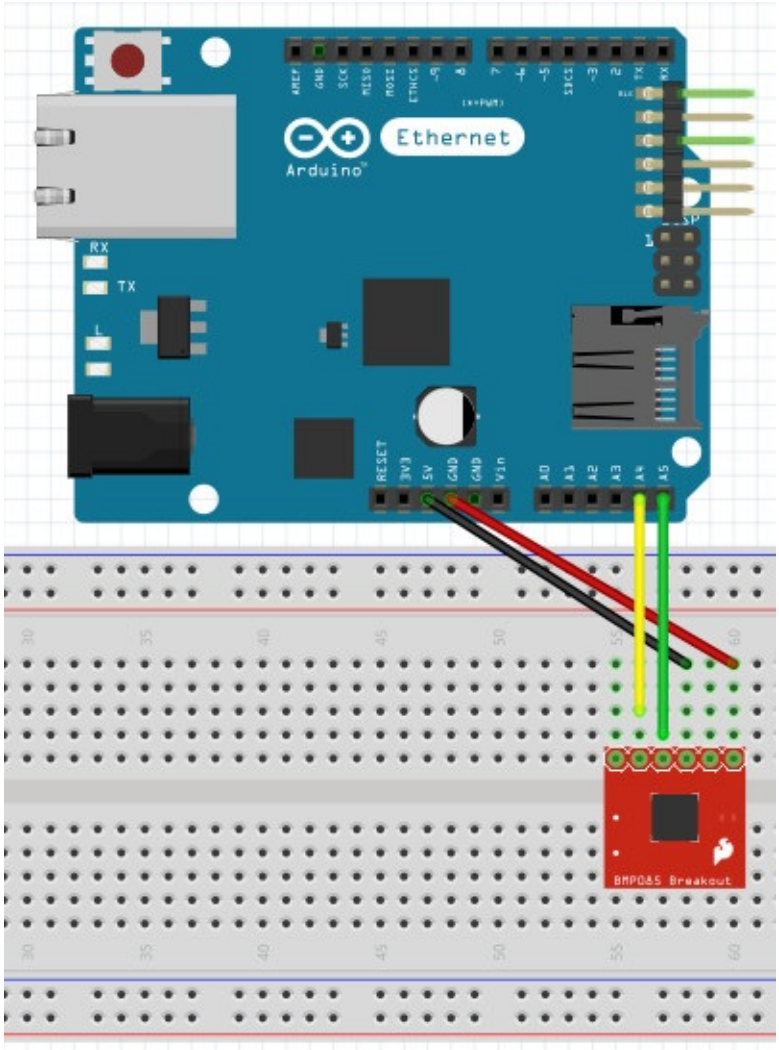
Sensor

Der `BMP280` misst Luftdruck (hPa) und Temperatur (°C). Dieser Sensor wird über das I²C Protokoll angesteuert, und benötigt eine Betriebsspannung von 3 bis 5 Volt. I²C-Geräte werden an den Arduino Uno über die Pins `A4` (SDA), und `A5` (SCL) angeschlossen, und so digital ausgelesen (siehe auch Der serielle Datenbus).

Da der Luftdruck von der Höhe über dem Meeresspiegel abhängt, kann über den `BMP280` auch die Aufbau-Höhe der senseBox bestimmt werden. Dazu wird ein Referenzdruck `P0` benötigt, dessen Höhe bekannt ist. Üblicherweise wird dazu der aktuelle Luftdruck auf Meerespiegelniveau verwendet. Da der Luftdruck in Abhängigkeit vom aktuellen Wetter stark schwanken kann, ist diese "Höhenmessung" aber nicht sehr akkurat, und muss immer wieder neu kalibriert werden.

Aufbau

Auf den Arduino muss zunächst der Ethernet-Shield gesteckt werden, um es mit einem Netzwerkanschluss auszustatten. Die Betriebsspannung des Sensors wird über die Verbindung des Arduino Pins `3.3V -> VCC` und `GND -> GND` realisiert.



Erweiterung des OSeM-Sketches

In dem Arduino-Sketch, den du nach der Registrierung erhalten hast, fehlt noch die Integration des Sensors.

Füge dazu die zugehörige Bibliothek zu dem Projekt hinzu, und erstelle eine Instanz `bmp` davon. Auf diesem Objekt werden alle Funktionen der Bibliothek aufgerufen:

```
#include <BMP280.h>
BMP280 bmp;
```

In der `setup()` -Funktion muss der Sensor initialisiert werden. Verwende dazu die folgenden Zeilen:

```
if (!bmp.begin()) Serial.println("BMP init failed!");
bmp.setOversampling(4);
```

Nun muss der Sensor in der `loop()` -Funktion innerhalb der `if` -Abfrage ausgelesen werden. In den Variablen `temp` und `pressure` stehen dann jeweils die aktuellen Messwerte. Über die bereits vorhandene Funktion `postFloatValue()` wird ein Messwert zur OSeM geschickt.

```
double temp, pressure;
char bmpStatus = bmp.startMeasurment();

// if an error occured on the sensor: stop
if (bmpStatus == 0) {
  Serial.println(bmp.getError());
  return;
}

delay(bmpStatus); // wait for duration of the measurement
bmpStatus = bmp.getTemperatureAndPressure(temp, pressure);

postFloatValue((float)temp, 4, TEMPSENSOR_ID);
postFloatValue((float)pressure, 4, PRESSURESENSOR_ID);
```

Netzwerkverbindung

Nachdem du den Arduino über ein Netwerkkabel mit dem Internet verbunden hast, kannst du den Sketch mit der IDE auf den Arduino hochladen. Im Seriellen Monitor (kann über `ctrl + shift + m` geöffnet werden), wird dir nun angezeigt ob die Verbindung zum Internet funktioniert.

Aufgabe 1

Registriere deine senseBox auf der OSeM und konfiguriere dort den korrekten Sensor.

Aufgabe 2

Erweitere deinen in Aufgabe 1 erhaltenen Arduino Sketch, sodass deine senseBox die Sensordaten zur OSeM überträgt.

> *Hinweis: Falls du die `BMP280.h`-Bibliothek nicht in der **Arduino** IDE findest, kannst du sie **hier** herunterladen und in den `Arduino/libraries` Ordner kopieren.*

Aufgabe 3

Berechne aus der Luftdruck-Messung die Aufbauhöhe über NN, und gebe diese über den seriellen Port aus.

> *Tipp: sieh dir das bei der **BMP280**-Bibliothek beiliegende Beispiel an. Der Referenzdruck P0 muss an die derzeitige Wetterlage angepasst werden.*

Verkehrszähler

Ziele der Station

Ziel ist es, einen Verkehrs- oder Personenzähler zu entwickeln. Dazu verwenden wir einen Ultraschall-Distanzsensor. Die so aufgenommenen Werte sollen im seriellen Monitor ausgegeben werden.

Materialien

- Ultraschall-Distanzsensor

Zusätzliche Materialien

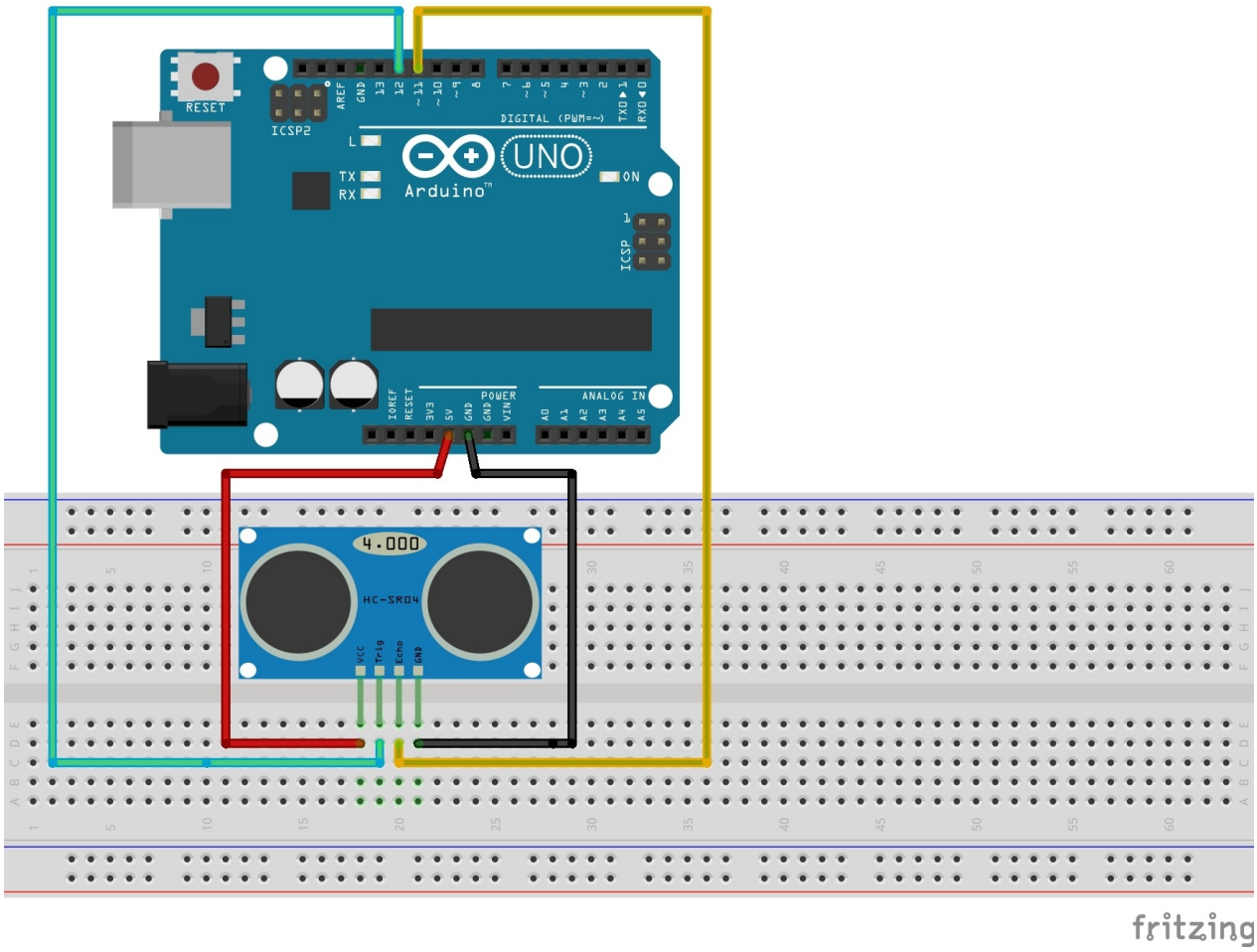
- Ihr benötigt keine zusätzlichen Materialien.

Grundlagen

Der Ultraschall-Distanzsensor nutzt den Schall um die Entfernung von Objekten zu bestimmen. Der Sensor sendet einen Impuls aus und misst die Zeit, bis er das Echo des Impulses wieder empfängt. Aus dieser Zeit errechnet man mit Hilfe der Schallgeschwindigkeit die Entfernung.

Aufbau

Der Ultraschallsensor wird mit vier verschiedenen Ports des Arduino verbunden. Zur Stromversorgung wird der VCC-Pin des Sensors mit dem 5V-Port des Arduino verbunden. Um den Stromkreis zu schließen wird der GND-Pin des Sensors mit einem GND-Port des Arduino verbunden. Als Letztes werden der Echo- und der Trig-Pin des Sensors jeweils mit digitalen Ports des Arduino verbunden.



Hinweis: *Ihr könnt natürlich jeden digitalen Port verwenden, denkt aber daran den Code anzupassen.*

Programmierung

Definiert die Pins an dem ihr den Sensor angeschlossen habt wie üblich. Außerdem werden zwei Variablen angelegt in der die gemessene Zeit und die errechnete Distanz gespeichert werden.

```
int trig = 12;  // Trig-Pin des Sensors ist an Pin 12 angeschlossen.
int echo = 11;  // Echo-Pin des Sensors ist an Pin 11 angeschlossen.
unsigned int time=0;
unsigned int distance=0;
```

Im `void setup` müsst ihr nun den Seriellen Monitor starten und die Ports an denen der Sensor angeschlossen ist als Ein- bzw. Ausgang definieren. Der Trigger-Pin des Sensors muss als Ausgang und der Echo-Pin als Eingang definiert werden.

```
Serial.begin(9600);
pinMode(trig, OUTPUT);
pinMode(echo, INPUT);
```

Im `void loop` wird mit den Befehlen

```
digitalWrite(trig, HIGH);
delayMicroseconds(10);
digitalWrite(trig, LOW);
```

ein 10 Microsekunden langer Ultraschallimpuls ausgesendet. Der darauffolgende Befehl `time=pulseIn(echo, HIGH);` speichert die Zeit bis zum Empfang des Echos in der Variable time. Zum Schluss muss noch die Distanz aus der Zeit errechnet werden, sowie die Werte auf dem Seriellen Monitor angezeigt werden.

```
distance=time/58
Serial.println(distace);
```

Aufgabe 1

Versucht mit Hilfe bekannter Befehle und dem oben angegebenen Sketch zum Ultraschallsensor einen Personen- bzw. Verkehrszähler zu entwickeln.

Beachtet dabei folgende Hinweise:

- Versucht nur einen bestimmten Entfernungsbereich auszuwerten, damit es nicht zu Störungen durch Bewegungen im Hintergrund kommt. Praktisch bedeutet solltet ihr also eure Fahrbahnen klar definieren.
- Um Mehrfachzählungen eines stehenden Fahrzeuges zu vermeiden solltet ihr eine Bedingung programmieren, der den Zählvorgang stoppt bis die Spur wieder frei ist, der Sensor also eine vorher festegelegte Maximaldistanz für die Spur misst.

Ampel

Es soll eine Ampel simuliert werden. Mit einem Button kann man die Ampel umschalten.

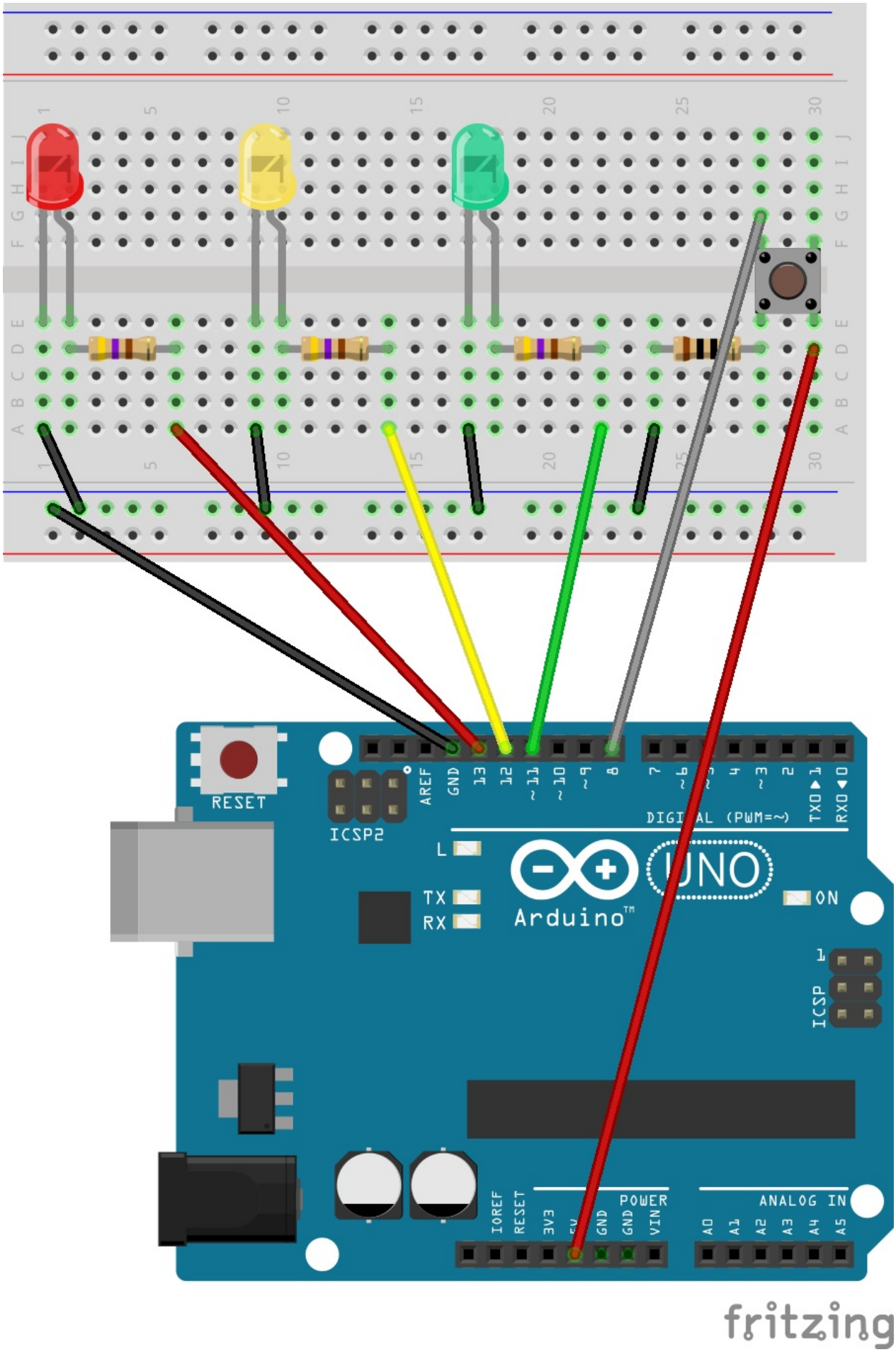
Materialien

Aus der senseBox:edu

- Genuino UNO
- rote LED
- gelbe LED
- grüne LED
- 3x 470Ω Wiederstand
- Button
- 10Ω Wiederstand

Setup Beschreibung

Hardwarekonfiguration


RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
Arduino™
UNO
ON
ICSP
POWER
ANALOG IN
IOREF
RESET
3V3
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5
fritzing

Softwaresketch

```
int rot = 13;
int gelb = 12;
int gruen = 11;

int button = 8;

void setup() {
  pinMode(rot, OUTPUT);
  pinMode(gelb, OUTPUT);
  pinMode(gruen, OUTPUT);

  // Der Button soll Signale messen, also INPUT
  pinMode(button, INPUT);

  // Ampel zuerst auf ROT setzen
  digitalWrite(rot, HIGH);
  digitalWrite(gelb, LOW);
  digitalWrite(gruen, LOW);
}

void loop() {

  // Hier wird geprüft ob der Button gedrückt wird
  if(digitalRead(button) == HIGH) {

    delay(5000);

    // ROT zu GRUEN
    digitalWrite(rot, HIGH);
    digitalWrite(gelb, HIGH);
    digitalWrite(gruen, LOW);

    delay(1000);

    digitalWrite(rot, LOW);
    digitalWrite(gelb, LOW);
    digitalWrite(gruen, HIGH);

    delay(5000);


    // GRUEN zu ROT
    digitalWrite(rot, LOW);
    digitalWrite(gelb, HIGH);
    digitalWrite(gruen, LOW);

    delay(1000);


    digitalWrite(rot, HIGH);
    digitalWrite(gelb, LOW);
    digitalWrite(gruen, LOW);
  }
}
```

- Am Anfang der `loop()` Funktion wird jedesmal abgefragt ob der Button gedrückt wird.
- `digitalRead(button)` liest den aktuellen Zustand des Buttons aus. Wird er gedrückt, liefert die Funktion `HIGH` aus, ansonsten `LOW`.
- Um zu Prüfen ob der Button gedrückt wurde muss `digitalRead(button)` mit `HIGH` verglichen werden. Der Vergleich geschieht mit **zwei** Gleichzeichen `==` (Vergleichsoperator). **Ein** Gleichzeichen `=` ist eine Zuweisung, wie etwa `int rot = 13`.

Die ersten Töne - Nutzung eines Summers

Bis jetzt ist unsere senseBox noch recht schweigsam, aber das wollen wir in dieser Station ändern.

Ziele der Station

In einem ersten Schritt wollen wir den Summer zum Piepen bringen. In einem zweiten Schritt wollen wir die Lautstärke des Summers verändern und in einem dritten Schritt soll der Summer eine kurze Melodie abspielen. Während die ersten beiden Ziele recht schnell erreicht werden, ist der dritte Teil ein kleines bisschen kniffeliger.

Materialien

- Summer
- Potentiometer

Grundlagen

Ein Summer oder Piezo ist ein Bauteil, welches elektrische Signale in Töne umwandelt. Die Lautstärke beträgt bis zu 80dB. Der Summer hat zwei Pins, mit denen er auf das Steckboard gesteckt werden kann. Die Betriebsspannung des Summers liegt zwischen 1V und 12V, wobei er bis zu 19mA verbraucht. Wie bei der LED kann der Strom nur in eine Richtung fließen. Der kürzer Pin muss mit Ground (GND) verbunden werden und der längere mit einer Spannungsquelle.

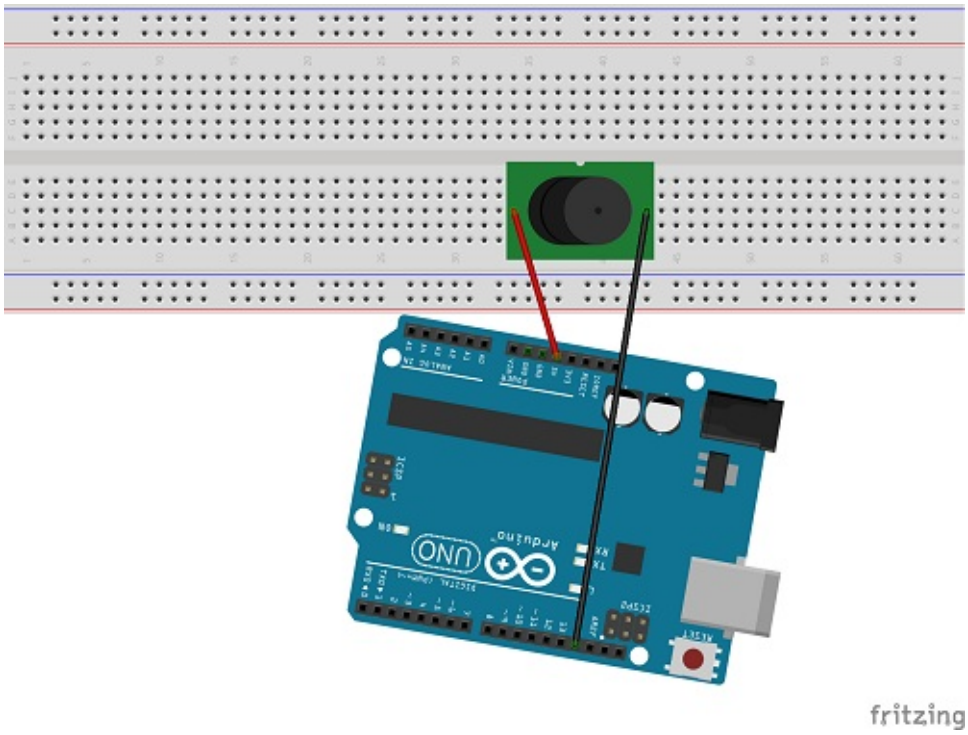
Das Potentiometer ist ein elektrisches Bauelement dessen Widerstandswert sich stufenlos einstellen lässt. Über einen Widerstandskörper wird der sogenannte Schleifer bewegt. Die Position des Schleifers bestimmten den Widerstand. Gewöhnlich hat ein Potentiometer drei Anschlüsse: Zwei für den Widerstand und einen dritten für den Abgriff. Unser Potentiometer hat einen maximalen Widerstand von 10k Ohm.

Achtung: *Kleine Potentiometer sind nur für einen relativ geringen Stromfluss konstruiert. Für die elektrischen Bauteile in der senseBox reicht das Potentiometer aus, doch wenn du Bauteile mit einer größeren Stromaufnahme anschließen möchtest (z.B. einen Servomotor), benötigst du ein größeres Potentiometer.*

Aufbau

Schritt 1:

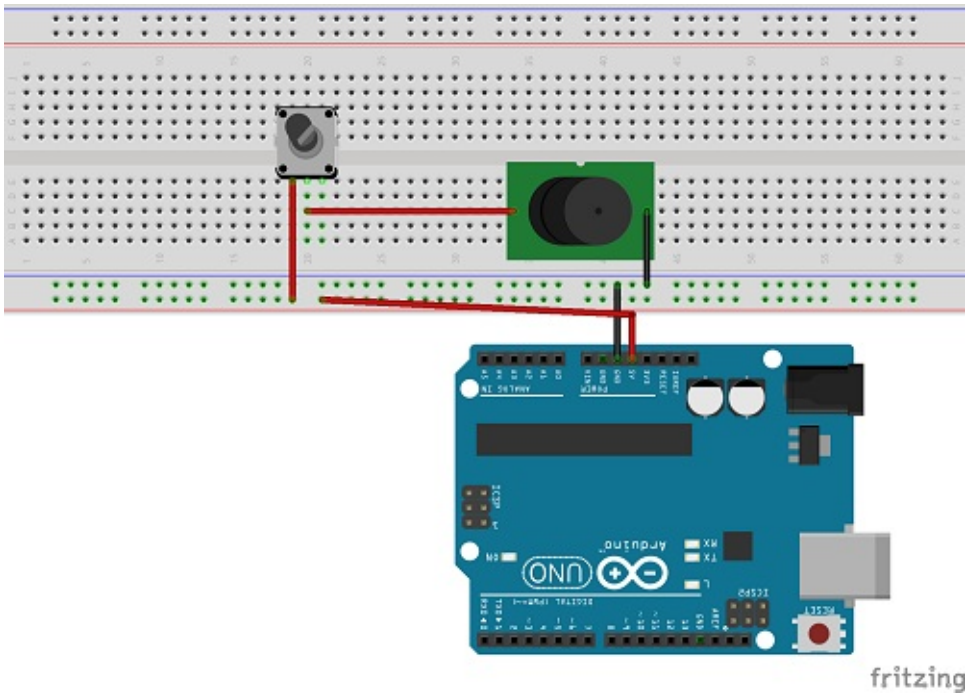
Wenn ihr den Schaltkreis wie in der Grafik steckt und den Arduino mit dem Netzteil verbindet, sollte der Summer einen lauten Dauerton erzeugen. Damit haben wir unsere erste Aufgabe bereits erledigt.



Schritt 2:

Nun wollen wir ein weiteres Bauteil in unseren Schaltkreis integrieren, mit dessen Hilfe sich die Lautstärke des Summers verändern lässt. Wie in älteren Radios wollen wir dazu ein Potentiometer verwenden.

Verbinde dazu den `5V` Ausgang des Arduinos mit dem Potentiometer, und dieses mit dem längeren Pin des Summers. Nun musst du den kurzen Pin des Summers noch mit `GND` verbinden und schon kannst du über das Potentiometer die Lautstärke verändern.



Tipp: *Falls ihr ein paar Informationen zur Funktionsweise eines Potentiometers lesen möchtet, schaut euch den Eintrag im Glossar an.*

Idee: *Falls ihr nicht jedes Mal das Kabel ziehen wollt um den Summer auszuschalten, könnt ihr noch einen Drucktaster einbauen.*

Schritt 3:

Ein einzelner durchgehender Ton ist nicht besonders spannend; unser Summer kann mehr. Um dem Summer verschiedene Töne zu entlocken, müssen wir spezielle Ausgänge des Arduinos nutzen, welche dazu in der Lage sind Pulsweiten auszugeben. Für nähere Information zu Pulsweitenmodulation (PWM) kannst du hier mehr erfahren. Für dieses Projekt ist das aber nicht unbedingt nötig.

Diese Pins sind mit dem Zeichen ~ gekennzeichnet: Das sind unter anderem die Pins 4, 5, 6, 9, 10 und 11. Ein Summer setzt jede Pulsweite in einen spezifischen Ton um. Wir wollen unser Programm so schreiben, dass eine Note (Tonleiter: c, d, e, f, g, a, h, c) für eine Pulsweite steht. Dazu benutzen wird das Konstrukt `#define` wie folgt:

```
#define  h     4064    // 246 Hz
#define  c     3830    // 261 Hz
#define  d     3400    // 294 Hz
#define  e     3038    // 329 Hz
#define  f     2864    // 349 Hz
#define  g     2550    // 392 Hz
#define  a     2272    // 440 Hz
#define  h     2028    // 493 Hz
#define  C     1912    // 523 Hz
#define  E     1518    // 659 Hz
#define  F     1432    // 698 Hz
#define  R     0       // Definiere eine Note als Ersatz für eine Pause
```

Nun benötigen wir einige Variablen um das spätere Abspielverhalten des Arduinos zu steuern. Die Werte könnt ihr für den Anfang so übernehmen. Wenn ihr später ein fertiges Programm habt, dann könnt ihr verschiedene Werte einsetzen und prüfen wie sich das auf die Melodie auswirkt:

```
// Set overall tempo
long tempo = 26000;
// Set length of pause between notes
int pause = 1000;
// Loop variable to increase Rest length
int rest_count = 50;
```

Zusätzlich benötigen wir noch einige globale Variablen welche von den Abspielfunktionen intern genutzt werden, und definieren im `setup` unseren Ausgangs-Pin:

```
// Initialize core variables
int tone = 0;
int beat = 0;
long duration  = 0;
int speakerOut = 9;

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

Jetzt könne wir unsere Melodie in ein Array schreiben. Ein weiteres Array `beats` definiert, wie lange die entsprechende Note in `melody` gespielt werden soll:

```
int melody[] = { g, e, R, R, R, e, f, g, E,  E,  C }; //example melody
int beats[]  = { 8, 8, 8, 8, 8, 8, 8, 8, 16, 16, 32 };
```

Natürlich dürft ihr hier später eure eigene Melodie einfügen. Dazu könnt ihr im Internet einmal die Noten eures Lieblingssongs suchen (siehe unten).

Wir schreiben uns eine Hilfsmethode, welche genau einen Ton unserer Melodie abspielt. Dazu überprüft sie in der ersten `if`-Anweisung ob es sich um eine Note oder eine Pause handelt. Falls es sich um eine Note handelt, wird die Note in einer Schleife für `duration` Millisekunden gespielt:

```
void playTone() {
  long elapsed_time = 0;
  if (tone > 0) { // if this isn't a Rest beat
    while (elapsed_time < duration) {

      digitalWrite(speakerOut,HIGH);
      delayMicroseconds(tone / 2);

      // DOWN
      digitalWrite(speakerOut, LOW);
      delayMicroseconds(tone / 2);

      // Keep track of how long we pulsed
      elapsed_time += (tone);
    }
  }
  else { // Rest beat;
    for (int j = 0; j < rest_count; j++) {
      delayMicroseconds(duration/2);
    }
  }
}
```

Nach dem wir jetzt einen Ton abspielen können, soll eine weitere Hilfsmethode die gesamte Melodie abspielen. Dazu geht eine `for`-Schleife unser Array `melody` durch und ruft für jeden Eintrag die Hilfsfunktion `playTone()` auf, die wir weiter oben definiert haben. Zusätzlich wird nach jeder Note eine kurze Pause eingefügt.

```
int MAX_COUNT = sizeof(melody) / 2; // number of tones

void playMelody(){

  for (int i=0; i<MAX_COUNT; i++) {
    tone = melody[i];
    beat = beats[i];

    duration = beat * tempo; // Set up timing

    playTone();

    delayMicroseconds(pause);
  }
}
```

Es fehlt uns nur noch die Hauptschleife, welche den Ablauf des Programms steuert:

```
void loop() {
  playMelody();
}
```

Idee: *Falls ihr für eure Melodie höhere oder tiefere Töne benötigt, so könnt ihr diese wie im Beispiel oben definieren. Wie viel Hertz ein Ton hat könnt ihr hier nachschauen. Diesen Wert müsst ihr dann mit einem antiproportionalen Dreisatz umrechnen.*

Achtung: *Zwei Variablen im Programm dürfen nicht den gleichen Namen haben!*

Lauschangriff

Ziele der Station

In dieser Station wollen wir lernen, wie wir das Mikrofon mit dem Arduino nutzen können.

Materialien

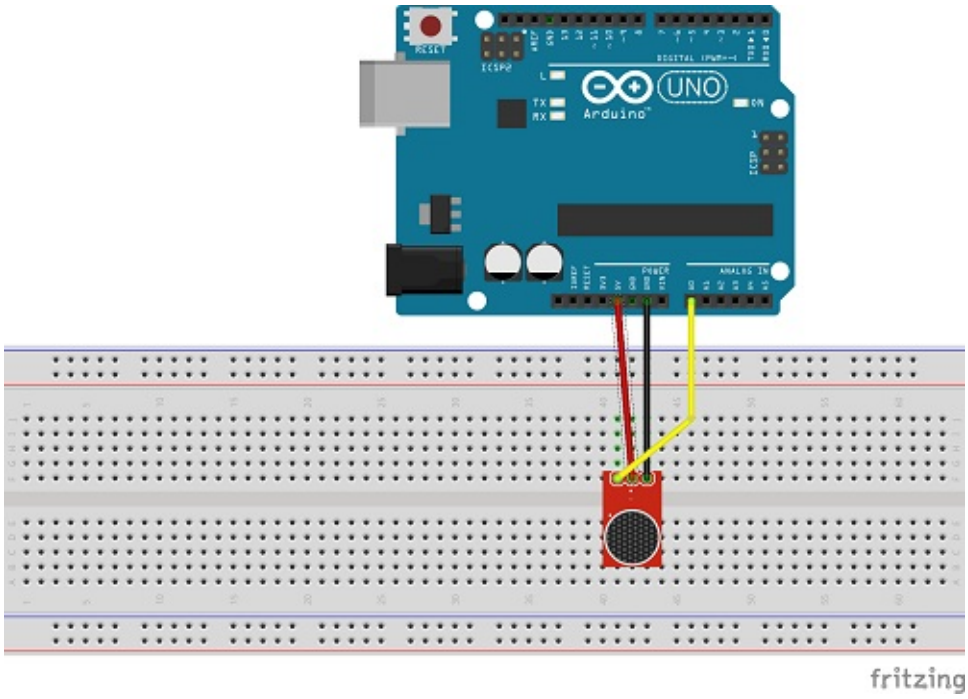
- Mic-Breakout

Grundlagen

Das Mikrofon ist zusammen mit einem Verstärker (100x) auf dem Board verbaut. Es benötigt eine Betriebspannung von 2.7V - 5.5V und ist in der Lage Geräusche zwischen 58dB und 110dB wahrzunehmen.

Aufbau

Steckt den Schaltkreis wie ihr ihn unten in der Grafik seht.



Hinweis: *Falls euer analoger Pin* `A0` *bereits belegt ist, könnt ihr natürlich auch einen anderen Pin verwenden. Vergesst nicht, dies auch im Code zu ändern.*

Aufgabe 1

Definiert den Pin an dem der Ausgang von eurem Mikrofon anliegt wie üblich. Außerdem muss eine Variable angelegt werden, in der die Werte des Mikrofons gespeichert werden:

```
int mic = A0;
long micVal = 0; //speichert den Wert des Mikrofons
```

Nun muss die serielle Ausgabe initialisieren werden und dem Pin `mic` der Modus `INPUT` zuordnen. Das machen wir im `setup`:

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

Wir schreiben uns eine Funktion, die den Mikrofonwert ausließt:

```
long getMicVal() {
  micVal = analogRead(mic);
  return micVal;
}
```

Wenn ihr nun in eurem `loop()` den Wert auslest könnt ihr ihn euch über den seriellen Monitor ausgeben lassen.

```
void loop() {
  Serial.print(getMicVal());
}
```

Ihr werdet sehen, dass die Ausgabe in einem leisen Raum um den Wert 510 schwankt. Bei lauten Geräuschen können auch negative Werte zurück gegeben werden. Um die Lesbarkeit der erhaltenen Werte zu verbessern führen wir einige Berechnungen in der Methode `getMicVal()` durch:

```
long getMicVal(){
  int period = 3; // mittelt drei Werte um 'Ausreißer' abzufangen
  int correction_value = 510;
  for(int i = 0; i < period; i++){
    // berechnet den Absolutbetrag des Wertes um negative Ausschläge abzufangen
    micVal = micVal + abs(analogRead(mic)-correction_value);
    delay(5);
  }
  micVal = constrain(abs(micVal/period),1,500);
  return(micVal);
}
```

Jetzt könnt ihr ausprobieren Welche Geräusche welche Ausschläge verursachen:

- Wie stark ist der Ausschlag bei Gesprächen?
- Was passiert wenn du den Summer vor das Mikrofon hältst?
- Und was, wenn du hinein pustest?

Aufgabe 2

Bau eine Geräuschampel mit drei LED's, welche in einem leisen Raum grün leuchtet, bei Zimmerlautstärke orange und bei Lärm rot.

Kleiner Webserver

Ziele der Station

In dieser Station wollen wir lernen, wie wir eine Webseite aufrufen die auf der SD Karte des Arduinos gespeichert ist. Über diese Seite sollen dann einzelne Pins des Arduinos geschaltet werden.

Das Projekt ist aufwändiger als die Einstiegsprojekte; du solltest also schon ein bisschen Erfahrung aus anderen Projekten gesammelt haben bevor du mit dem Webserver beginnst. Die Themen Netzwerkzugriff und HTML sind so umfangreich, dass wir sie im Rahmen dieses Projekts nur anschneiden können. Weiterführende Informationen sind im Internet aber leicht zu finden!

Materialien

- Ethernet-Shield

Materialien die nicht in der senseBox enthalten sind

- LAN Kabel

Grundlagen

Ethernet-Shield

In der senseBox ist ein Ethernetshield enthalten. Über dieses kann man den Arduino mit Hilfe des LAN-Kabels mit dem Netzwerk verbinden.

Da es meist sehr schwer ist, über das Internet auf ein Schulnetzwerk zu zugreifen, werden wir nur über das lokale Netzwerk auf den Arduino zugreifen.

Falls ihr dennoch den Arduino über das Internet erreichbar machen wollt und Zugang zu eurem Router habt, gibt es die Möglichkeit dies über *Portforwarding* umzusetzen. Hier findet ihr weitere Informationen dazu.

Webserver

Als Webserver (englisch server ‚Diener') bezeichnet man einen Computer mit Webserver-Software (in unserem Fall den Arduino), der Dokumente an Clients wie z.B. Webbrowser überträgt. Das Dokument ist bei uns der Inhalt von index.htm.

Aufbau

Aufgabe 1

Für den Server werden drei Bibliotheken benötigt:

```
#include <Ethernet.h>
#include <TextFinder.h>
#include <SD.h>
```

Achtung *Es ist wichtig, dass ihr die Ethernet-Bibliothek von unserer Website benutzt. Mit der Standartbibliothek vom Arduino funktioniert unser Ethernet-Shield nicht. Die Arduinoumgebung wird die Ethernet-Bibliothek immer 'aktualisieren' wollen; das dürft ihr nicht machen.*

Anschließend müssen in zwei Variablen die MAC-Adresse und die IP-Adresse des Arduino-Servers fest legen. Zusätzlich muss der Pin 4 als SD-Karten Pin festgelegt werden. Außerdem definieren wir uns ein `File` -Objekt `webFile` in dem die HTML Seite abgelegt wird.

```
// MAC-Adresse des Ethernet-Shield
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// Lokale IP zum aufrufen des Webservers
IPAddress ip(192, 168, 0, 42);
// Pin der SD-Karte
byte sdPin = 4;
File webFile;
```

Dem Server müssen wir jetzt noch ein Port zuweisen. Außerdem führen wir eine Variable ein, über die definiert wird ob wir uns im Testmodus befinden. In diesem Fall werden über den seriellen Port zusätzliche Statusinformationen ausgegeben, die das System allerdings verlangsamen.

```
// Server port
EthernetServer server(80);
boolean testmode = false;// Auf true setzen falls etwas nicht funktioniert.
Dann werden Informationen über die serielle Schnittstelle ausgegeben.
```

In der setup-Methode wird im ersten Schritt die mac-Adresse und die IP-Adresse an den Server übergeben. Anschließend wird überprüft ob ein Zugriff auf die SD Karte besteht und ob auf der Karte ein Datei mit dem Namen `index.htm` existiert. Falls die Variable `testmode` auf `true` gesetzt wurde, werden über die serielle Schnittstelle Statusinformationen ausgegeben. Das kann dir bei der Suche von Fehlern helfen.

```
void setup() {
  if(testmode) Serial.begin(9600);
  Ethernet.begin(mac, ip); // Client starten
  server.begin();          // Server starten

  if(testmode) Serial.println("Initialisiere SD-Karte...");
  if(!SD.begin(sdPin)) {
    if(testmode)Serial.println("Initialisierung der SD-Karte fehlgeschlagen!");
    return;
  }
  if(testmode) Serial.println("SD-Karte erfolgreich initialisiert.");

  if(!SD.exists("index.htm")) {
    if(testmode) Serial.println("Datei (index.htm) wurde nicht gefunden!");
    return;
  }
  if(testmode) {
    Serial.println("Datei (index.htm) wurde gefunden.");
    Serial.println("Verbraucher schalten");
  }
}
```

Die `loop` -Methode ist in zwei Bereiche unterteilt. Im ersten Teil werden Aktionen auf der Website ausgewertet, im zweiten Teil wird die Website von der SD-Karte ausgelesen und an den Browser gesendet.

Auswertung von Anfragen

Wenn der client verfügbar ist, wartet der `finder` auf eine Anfrage vom client. Bekommt er diese, sucht er das Schlüsselwort "pin" speichert er die nächsten beiden Zeichen in den Variablen `typ` , `pin` und `val` . Je nach Typ wird dann eine Aktion ausgeführt. Standartmäßig sind die Typen D, A und a definiert. D = digitalen Pin schalten A = analogen Pin schalten a = analogen Pin auslesen

```
void loop() {
  /*********************
   * Anfragen auswerten *
   *********************/
  EthernetClient client = server.available(); // Auf Anfrage warten
  if(client) {
    TextFinder finder(client);

    if(finder.find("GET")) { //erkennt Aktion auf der Website
      while(finder.findUntil("pin", "\n\r")) {// Bis das Schlüsselwort "pin" erkannt wird
        char typ = char(client.read());
        int  pin = int(finder.getValue());
        int  val = int(finder.getValue());

        if(typ == 'D') {
          pinMode(pin, OUTPUT);
          digitalWrite(pin, val);
          if(testmode)Serial.print(" - D"+String(pin));
        }
        else if(typ == 'A') {
          analogWrite(pin, val);
          if(testmode)Serial.print(" - A"+String(pin));
        }
        else if(typ == 'a') { // a -> Sensorwert auslesen und ausgeben
          val = analogRead(pin);
          if(testmode)Serial.print(" - a"+String(pin));
        }
        else {
          if(testmode) Serial.print(" - Falscher Typ");
        }
        //Hier können neue Befehle definiert werden die aus dem Browser gestartet werden.
        if(testmode) {
          if(val==1) Serial.println(" ein");
          else if(val==0) Serial.println(" aus");
          else Serial.println(" "+ val);
        }
      }
    }
```

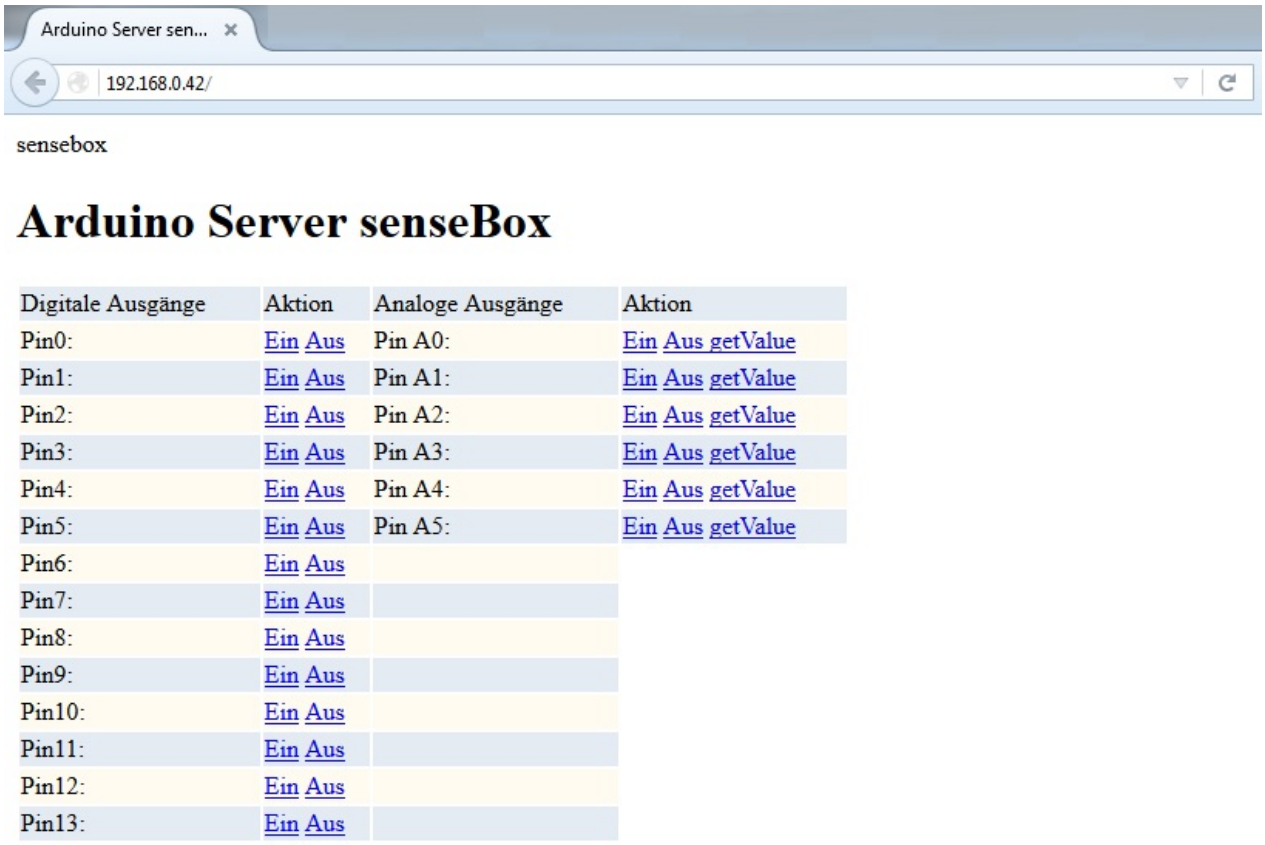
Webformular anzeigen

Jede Zeile des HTML Dokuments wird eingelesen und an den Browser geschickt. Da der Speicherplatz für Variablen sehr begrenzt ist, wird die Zeichenkette (String) bei jedem Leerzeichen oder Zeilenumbruch geteilt. Anschließend wird der String nach bestimmten Schlüsselwörtern durchsucht. Diese kann man nach bedarf in der Methode `filter` nach bedarf definieren. Diese Schlüsselwörter können dann durch Sensorwerte ersetzt werden.

```
    /************************
    * Webformular anzeigen  *
    ************************/
    boolean current_line_is_blank = true;         // eine HTTP-Anfrage endet mit einer Leerzeile und einer neuen Zeile

    String htmlline = "";
    while (client.connected()) {
      if (client.available()) {                   // Wenn Daten vom Server empfangen werden

        char c = client.read();                   // empfangene Zeichen einlesen
        if (c == '\n' && current_line_is_blank) { // wenn neue Zeile und Leerzeile empfangen
          // Standard HTTP Header senden
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          client.println();

          // Website von SD-Karte laden
          webFile = SD.open("index.htm");  // Website laden
          if (webFile) {
            while(webFile.available()) {
              char temp = char(webFile.read());
              //Bei jedem Leerzeichen wird ein neuer String generiert
              if((int(temp) != 10)&&(int(temp) != 32)&&(int(temp) != 13)){
                htmlline = htmlline + char(temp);
              }
              else {
                htmlline = filter(htmlline); //Ersetzt Schlüsselwörter zum Beispiel durch Sensorwerte
                client.println(htmlline);
                if(testmode) Serial.println(htmlline);
                htmlline = "";
              }
            }
            webFile.close();
          }
          break;
        }
        if (c == '\n') {
          current_line_is_blank = true;
        }
        else if (c != '\r') {
          current_line_is_blank = false;
        }
      }
    }
    delay(1);
    client.stop();
  }
}

//Ersetzt Schlüsselwörter zum Beispiel durch Sensorwerte
// Es können nach belieben neue Schlüsselwörter definerit und durch andern Text ersätzt werden
String filter(String htmlline){
  htmlline.replace("sa0en",String(analogRead(A0)));
  htmlline.replace("sa1en","No sensor");
  htmlline.replace("sa2en","No sensor");
  htmlline.replace("sa3en","No sensor");
  htmlline.replace("sa4en","No sensor");
  htmlline.replace("sa5en","No sensor");
  return htmlline;
}
```

HTML Dokument

Unsere erste kleine Webseite sieht so aus:



Diesen Code musst du kopieren und in einer Textdatei abspeichern. Den Namen der Textdatei musst du anschließend in index.htm ändern. Das HTML Dokument muss anschließend auf der SD Karte gespeichert und diese in das Ethernetshield gesteckt werden.

```
<!DOCTYPE html>
<html>
<head>
  <title>Arduino Server senseBox</title>
  <style type="text/css">
    h2 { margin-bottom:5px; }
    table {width: 40%;} th { background-color: #666; color: #fff; } tr { background-color: #fffbf0; color: #000; } tr
:nth-child(odd) { background-color: #e4ebf2 ; } </style>
    </style>
</head>
<body>
  <img src="sensebox_logo_neu.png" alt="sensebox" width="700" height="150"/>
  <h1>Arduino Server senseBox</h1>
  <!-- D for digitalWrite, A for analogWrite, d for digitalRead, a for analogRead -->
  <table>
    <tr> <td>Digitale Ausg&auml;nge</td> <td> Aktion </td><td>Analoge Ausg&auml;nge</td><td>Aktion</td></td></tr>
    <tr> <td>Pin0:</td> <td> <a href="/?pinD0=1" target="ifr">Ein</a> <a href="/?pinD0=0" target="ifr">Aus</a> </td>
<td> Pin A0:</td> <td><a href="/?pinA0=1" target="ifr">Ein</a> <a href="/?pinA0=0" target="ifr">Aus </a> <a href="/?p
ina0=2" target="ifr"> getValue </a></td></tr>
    <tr> <td>Pin1:</td> <td> <a href="/?pinD1=1" target="ifr">Ein</a> <a href="/?pinD1=0" target="ifr">Aus</a> </td>
<td> Pin A1:</td> <td> <a href="/?pinA1=1" target="ifr">Ein</a> <a href="/?pinA1=0" target="ifr">Aus</a> <a href="/?p
ina1=2" target="ifr"> getValue </a></td></tr>
    <tr> <td>Pin2:</td> <td> <a href="/?pinD2=1" target="ifr">Ein</a> <a href="/?pinD2=0" target="ifr">Aus</a> </td>
<td> Pin A2:</td> <td> <a href="/?pinA2=1" target="ifr">Ein</a> <a href="/?pinA2=0" target="ifr">Aus</a> <a href="/?p
ina2=2" target="ifr"> getValue </a></td></tr>
    <tr> <td>Pin3:</td> <td> <a href="/?pinD3=1" target="ifr">Ein</a> <a href="/?pinD3=0" target="ifr">Aus</a> </td>
<td> Pin A3:</td> <td> <a href="/?pinA3=1" target="ifr">Ein</a> <a href="/?pinA3=0" target="ifr">Aus</a> <a href="/?p
ina3=2" target="ifr"> getValue </a></td></tr>
    <tr> <td>Pin4:</td> <td> <a href="/?pinD4=1" target="ifr">Ein</a> <a href="/?pinD4=0" target="ifr">Aus</a> </td>
<td> Pin A4:</td> <td> <a href="/?pinA4=1" target="ifr">Ein</a> <a href="/?pinA4=0" target="ifr">Aus</a> <a href="/?p
ina4=2" target="ifr"> getValue </a></td></tr>
    <tr> <td>Pin5:</td> <td> <a href="/?pinD5=1" target="ifr">Ein</a> <a href="/?pinD5=0" target="ifr">Aus</a> </td>
<td> Pin A5:</td> <td> <a href="/?pinA5=1" target="ifr">Ein</a> <a href="/?pinA5=0" target="ifr">Aus</a> <a href="/?p
ina5=2" target="ifr"> getValue </a></td></tr>
    <tr> <td>Pin6:</td> <td> <a href="/?pinD6=1" target="ifr">Ein</a> <a href="/?pinD6=0" target="ifr">Aus</a> </td>
<td></td></tr>
    <tr> <td>Pin7:</td> <td> <a href="/?pinD7=1" target="ifr">Ein</a> <a href="/?pinD7=0" target="ifr">Aus</a> </td>
<td></td></tr>
    <tr> <td>Pin8:</td> <td> <a href="/?pinD8=1" target="ifr">Ein</a> <a href="/?pinD8=0" target="ifr">Aus</a> </td>
<td></td></tr>
    <tr> <td>Pin9:</td> <td> <a href="/?pinD9=1" target="ifr">Ein</a> <a href="/?pinD9=0" target="ifr">Aus</a> </td>
<td></td></tr>
    <tr> <td>Pin10:</td> <td> <a href="/?pinD10=1" target="ifr">Ein</a> <a href="/?pinD10=0" target="ifr">Aus</a> </t
d> <td></td></tr>
    <tr> <td>Pin11:</td> <td> <a href="/?pinD11=1" target="ifr">Ein</a> <a href="/?pinD11=0" target="ifr">Aus</a> </t
d> <td></td></tr>
    <tr> <td>Pin12:</td> <td> <a href="/?pinD11=1" target="ifr">Ein</a> <a href="/?pinD12=0" target="ifr">Aus</a> </t
d> <td></td></tr>
    <tr> <td>Pin13:</td> <td> <a href="/?pinD13=1" target="ifr">Ein</a> <a href="/?pinD13=0" target="ifr">Aus</a> </t
d> <td></td></tr>
  </table>
  <iframe name="ifr" style="display:none;" width="0" height="0"></iframe>
</body>
</html>
```

Kaminfeuer

Es soll ein Kaminfeuer simuliert werden. Dazu wird eine rote LED zum flackern gebracht.

Materialien

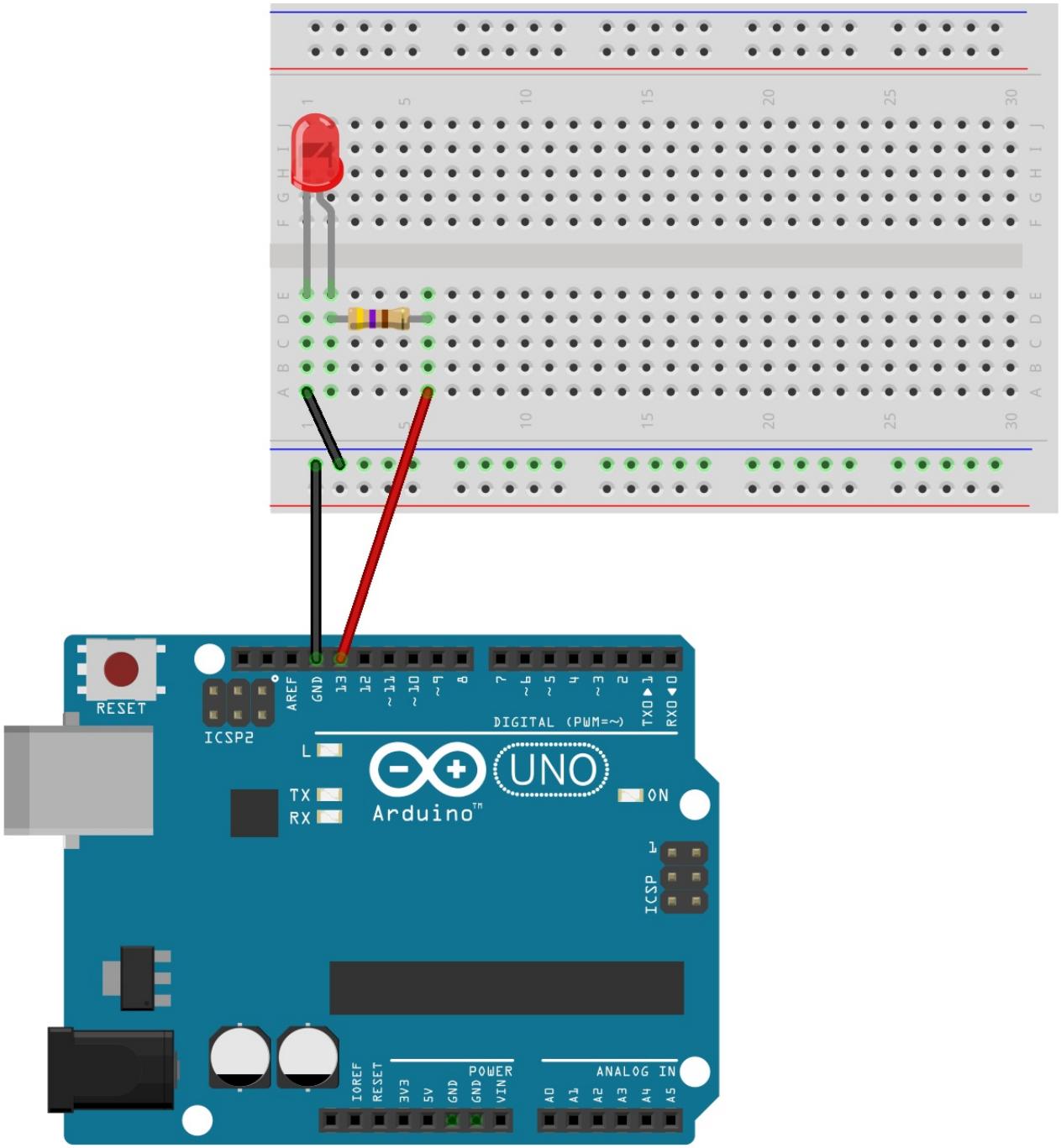
Aus der senseBox:edu

- Genuino UNO
- LED
- 470Ω Wiederstand

Setup Beschreibung

Hardwarekonfiguration

Es wird nur die LED angeschlossen. Diese wird am langen Beinchen mit einem 470Ω Wiederstand mit dem digitalen Port 13 verbunden. Das kurze Beinchen wird mit GND verbunden.



Softwaresketch

```
// die LED ist an den digitalen Port 13 angeschlossen
int led = 13;

void setup() {
  // der digitale Port 13 wird als OUTPUT definiert
  // d.h.: es werden Signale herausgeschickt
  pinMode(led, OUTPUT);
}

void loop() {
  // generiert Zufallswert zwischen 0 und 1000 und speichert ihn in randomDelayAn
  int randomDelayAn = random(1000);
  digitalWrite(led, HIGH);

  // Zufallswert wird eingesetzt, LED bleibt entsprechend lange an (in Millisekunden)
  delay(randomDelayAn);

  // generiert Zufallswert zwischen 0 und 500 und speichert ihn in randomDelayAus
  int randomDelayAus = random(500);
  digitalWrite(led, LOW);

  // Zufallswert wird eingesetzt, LED bleibt entsprechend lange aus (in Millisekunden)
  delay(randomDelayAus);
}
```

- Wir speichern den digitalen Port 13 in einer Variable, damit wir uns nur noch den aussagekräftigen Variablennamen merken müssen und nicht die Portnummer. Das ist vorallem bei mehreren angeschlossenen LEDs hilfreich.
- Die Funktion `random(max)` generiert Zufallszahlen von 0 bis `max`. Falls man ebenfalls ein Minimum angeben will kann man die Funktion `random(min, max)` benutzen.

Community Projekte

Hier sind verschiedene Projekte zu finden, welche von der Community erstellt wurden!

Ihr könnt auch einen Blick auf die englischsprachigen Projekte in der englischen Version dieses Buchs werfen.

Wenn ihr selber ein Projekt hinzufügen wollt, schaut euch die Anleitung an.

Projekt Titel	Autor	Datum	zusätzliche Materialien

Glossar

Arduino

Der Arduino ist ein quelloffenes Mikrocontroller-Board, auf welchem ein Microcomputer sowie mehrere digitale und analoge Eingänge integriert sind.

Wattuino

Der Wattuino UNO wurde in älteren Versionen der senseBox verwendet und entspricht weitgehend dem Arduino UNO, wird aber in Deutschland von Watterott electronic hergestellt.

Genuino

Genuino ist der aufgrund von Lizenzverfahren in Europa verwendete Vertriebsname von Arduino.

BMP280

Der BMP280 ist ein Luftdruck- und Temperatursensor, welcher in der senseBox auf einem Breakoutboard installiert ist. Er wird über I²C angesteuert und hat die dort die Adresse 0x77. Betriebsspannung 3 - 5.5V. Datenblatt

VEML6070

Der VEML6070 ist ein UV-Intensitäts-Sensor, welcher in der senseBox auf einem Breakoutboard installiert ist. Er wird über I²C angesteuert und hat die dort die Adressen 0x38 und 0x39. Betriebsspannung 2.7 - 5V. Datenblatt

HDC1008

Der HDC1008 ist ein Luftfeuchte- und Temperatursensor, welcher in der senseBox auf einem Breakoutboard installiert ist. Er wird über I²C angesteuert und hat die dort die Adresse 0x43. Betriebsspannung 3.3 - 5V. Datenblatt

RV8523

Die RV8523 ist eine Realtimeclock (RTC), welche durch eine separate Stromversorgung (Knopfbatterie) dem Arduino immer die aktuelle Zeit angibt. Die RTC ist auf dem senseBox-Shield integriert und wird über I²C unter der Adresse 0x68 angesteuert.

I²C

I²C (Inter-Integrated Circuit) ist ein Protokoll zur digitalen Kommunikation von Geräten auf einem Bus. Das Protokoll wurde 1982 entwickelt und darauf optimiert, mehrere Geräte mit möglichst wenigen Kanälen ansteuern zu können. Geräte haben dabei eine bestimmte Adresse, über die sie auf zwei Kanälen SDA (serial data) und SDC (serial clock) angesteuert werden. Mehrere Geräte können so in Reihe hintereinander an ein Masterdevice (in unserem Fall den Arduino) angeschlossen werden.

Ethernet-Shield

Das Ethernet-Shield basiert auf dem Wiznet W5500 Ethernet Chip und ermöglicht es den Geduino über ein LAN-Kabel an ein Netzwerk anzuschließen. Es läuft mit einer Betriebsspannung von 3.3V - 5V. Mit dem Reset-Knopf wird sowohl das Shield als auch der Arduino neugestartet. Datenblatt

senseBox-Shield

Das senseBox-Shield ist eine Steckplatine, die vom senseBox-Team zusammen mit Watterott electronic entwickelt wurde. Auf dem Shield ist ein SD-Kartenslot (über Pin 4) und eine Realtimeclock RV8523 mit Batterie verbaut. Außerdem sind I2C-Steckplätze für die senseBox:home-Sensoren verlötet, welche für die senseBox:edu nicht benötigt werden. Da der SD-Kartenslot Pin 4 und 10 belegt, sind diese standardmäßig nicht mehr verwendbar. Benötigt man den SD-Kartenslot nicht, so kann man die Pins mit den Befehlen pinMode(4, INPUT); digitalWrite(4, HIGH); freigeben. Schemazeichnung

FAQ

Downloads

In diesem Bereich findest du verschiedene Downloads, die dir bei der Verwendung der senseBox helfen können.

Libraries

Ein vollständiges Library Paket für die senseBox:edu findest du hier. Beachte, dass wir eine modifizierte Version der `Ethernet.h` -Bibliothek verwenden, da die Variante der Arduino-IDE inkompatibel mit unserem Ethernet-Shield ist.

Datenblätter

Viele Hersteller bieten für ihre Sensoren und andere Bauteile Datenblätter an, aus denen ihr verschiedenen elektronische Spezifikationen ablesen könnt:

Sensor	Beschreibung	Hersteller	Download
BMP280	Luftdrucksensor	Bosch	Datenblatt
HC-SR04	Ultraschall Distanzsensor	KT-Electronic	Datenblatt
HDC1008	Kombinierter Temperatur und Luftfeuchtigkeitssensor	Texas Instruments	Datenblatt
TSL4531	Digitaler Lichtsensor	TAOS Texas Advanced Optoelectronics Solutions	Datenblatt
VEML6070	UV-Lichtsensor	VISHAY	Datenblatt
GP2Y0A21YK0F	IR-Distanzsensor	Sharp	Datenblatt

Dokumentation als PDF

Dieses Buch ist auch als PDF zum ausdrucken verfügbar!