7.3.2021

Grundlagen im Umgang mit dem Arduino Nano

Der Versuch einer Anleitung für Einsteiger, Anfänger und alle, die es ausprobieren wollen

speziell für Freunde des ZFI

Bei Fragen oder Hinweisen bitte an:

hage_mueller@gmx.de

Schorsch Müller
begeisterter Zetti

Inhaltsverzeichnis

Vorwort

Diese kurze Einführung in die Grundlagen des Umgangs mit dem Arduino Nano soll nur Anregung sein. Ich bin selbst Anfänger, also bitte keine Doktorarbeit erwarten. Fehler gibt es bestimmt auch mal. Geplant ist auch ein zweiter Teil, in dem es um Erweiterungen gehen soll (Bluetooth, Motoren, WLAN etc.)

1. Arduino-Grundlagen

Um mit Arduino zu programmieren werden einige Komponenten benötigt. Na klar, Soft- und Hardware. Ich beziehe mich bei meinen laienhaften Ausführungen auf den Nano, er ist schön klein, preiswert und reicht meistens aus. Alles lässt sich auch auf den Uno anwenden.

1.1. Software

1.1.1. *Arduino IDE, die Programmierumgebung für C/C++*

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

1.1.2. *Fritzing*

ein Programm mit grafischer Benutzeroberfläche, um Schaltpläne zu erzeugen

1.1.3. *Treiber für Nano board*

abhängig vom Board wird evt. noch ein USB-Treiber benötigt. Eine gute Seite zum Download inkl. Installationsanleitung gibt's hier: https://www.makershop.de/ch340-341-usb-installieren/

1.1.4. *Anleitungen, Hilfe, Beispiele*

Es gibt unzählige Seiten, einige habe ich hier gefunden. Ich empfehle Einsteigern auf jeden Fall sich mit den Grundlagen vertraut zu machen, ohne ein wenig Theorie geht gar nichts, leider.
Sehr schön, habe sogar ich verstanden:
https://funduino.de/anleitung#211_Beschreibung_Controller
Hier gibt's viel Theorie und Praxis mit Beispielen:
https://starthardware.org/category/projekte/arduino-projekte/
Befehlsreferenz Arduino, kommt in engl., kann auf deutsch umgestellt werden. Ich habe einen Link gespeichert, greife oft darauf zu, die meisten Befehle etc. sind gut erklärt und mit Beispielen hinterlegt:
https://www.arduino.cc/reference/en/
Und einige dicke Wälzer habe ich auch gefunden, als PDF verfügbar:
Fritzing Creator Kit Anleitung, 68 Seiten, bunte Bilder will ich sehen (und verstehen)
Arduino Praxisbuch 528 Seiten, für Nerds
Schnupperkurs 85 Seiten, Grundlagen und erste Programme in Wort und Bild
Und noch viel mehr. Wer will, über PN wird's verschickt.
Natürlich gibt's noch viel mehr, das wird hier zu viel, glaube ich.

1.2. Hardware

1.2.1. Board

Nanos gibt's überall, aber… Nicht alle funzen gleich gut. Das liegt am verbauten Prozessor. Es sollte ein Atmega 328p Version 3.1 sein, das gibt's schnell und günstig hier:
https://www.ebay.de/itm/Nano-Atmega328P-V3-1-BEREITS-GEL%C3%96TET-verl%C3%B6tet-fertig-aufgebaut-Arduino/253638131454?ssPageName=STRK%3AMEBIDX%3AIT&_trksid=p2057872.m2749.l2649
Andere Boards brauchen dann evt. den alten Bootloader, macht Ärger

1.2.2.Zubehör

Shield zum festen verschrauben, praktisch https://www.ebay.de/itm/HIMALAYA-basic-Nano-Expansion-Terminal-Board-IO-Shield-Adapter-for-Arduino-Nano/283712174549?ssPageName=STRK%3AMEBIDX%3AIT&_trksid=p2057872.m2749.l2648

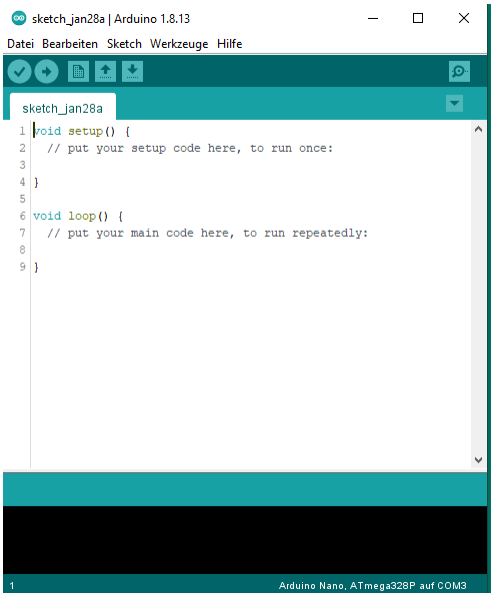
1.2.3.*Breadboards, Steckbrett, Steckbrücken (flexibel), Netzteil, Servos, Steppermotoren,* LEDs, Widerstände und, und,…
Alles nach Bedarf, es gibt Unmassen an Zubehör, wir werden schon merken, was wir noch brauchen

Ich empfehle allen Mitlesern und Einsteigern erst mal alles zu installieren und zu probieren, ob die IDE funzt und der Nano erkannt wird. Wenn nicht, gleich fragen, sonst geht's nicht weiter

Ok, da es keine Rückmeldungen gab, mache ich mal weiter. Immer noch Grundlagen, muss wohl sein.

1.3. Installation und erster Test

Wenn die IDE installiert ist, sollte man die Übertragung überprüfen. Nach dem Start der Arduino DIE sollte es so aussehen. Voraussetzung: der Nano ist über USB-Mikrokabel angeschlossen ;)



Über das Menü Werkzeuge-Prozessor muss Atmega 328P gewählt werden.
Dann bei Werkzeuge-Board den Nano, und
Über Werkzeuge-Port den entsprechende COM-Port.
Achtung. Wer einen alten Arduino hat, sollte bei Prozessor den Bootloader old wählen, evt. ausprobieren.
Wenn alles funktioniert, leuchtet die 2. LED von unten dauerhaft rot.
Sollte es bis dahin Probleme geben, „Bitte melde dich".

Und jetzt probieren wir mal ein erstes Programm (hier heißen die Dinger Sketch) aus. Die IDE bietet eine ganze Menge färdsches Zeuch an. Zu finden unter Datei-Beispiele und wir nutzen Basics. Dort greifen wir uns ein einfaches BLINK. Dieser Sketch nutzt die auf dem Board eingebaute LED und lässt sie blinken und zwar im Sekundentakt. Mit Klick auf Hochladen wird die Übertragung gestartet, es blinkt eine (die unterste) LED im Sekundentakt.

Hurra (oder auch nicht), ich habe ein Programm gemacht (Tom Hanks).

1.4. Fragen, Hinweise, Kritiken oder nur weitermachen???

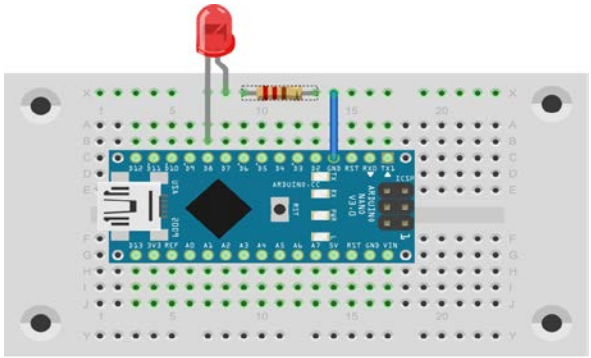
Ok, kurz vor dem Fußball, mache ich mal weiter. Immer noch Grundlagen, muss wohl sein.
Den Punkt 1.4. Fragen lasse ich offen und ergänze, falls notwendig

1.5. Ein erster eigener Sketch und Erklärungen

Nehmen wir und den Beispielsketch „Blink“ mal vor

```
/*        Das ist das Einleiten-Symbol für mehrzeiligen Kommentar
  Blink   Alles zwischen diesen Zeilen ist wie Notizen, ich schreibe mir immer dazu, worum es geht, man wird ja nicht jünger ;)

  Turns an LED on for one second, then off for one second, repeatedly.
Lässt eine LED im Sekundentakt blinken


  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/ Abschlusssymbol für mehrzeiligen Kommentar, der hier abgeschlossen wird

// the setup function runs once when you press reset or power the board = einzeiliger Kommentar
void setup() {    diese Funktion steht am Anfang jedes Programms, wird 1 mal ausgeführt
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT); legt den Pin der eingebauten LED als Ausgangsmodus fest
}
// the loop function runs over and over again forever
void loop() { die Loop-Funktion (Schleife)wird immer wieder durchlaufen
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second) Pausen-Befehl warte 1 Sekunde (1000 millisek
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW Setzt Pin auf aus
  delay(1000);                       // wait for a second warte 1 Sekunde (1000 millisek
}
```

Funktionen werden mit void aufgerufen und haben immer (), was in der Funktion passiert, wird in {} eingeschlossen. Einzelne Befehle müssen mit ; abgeschlossen werden.
Soweit, so gut. Machen wir den Test und schreiben mal einen eigenen Sketch, der eine externe LED blinken lässt. So siehts aus:

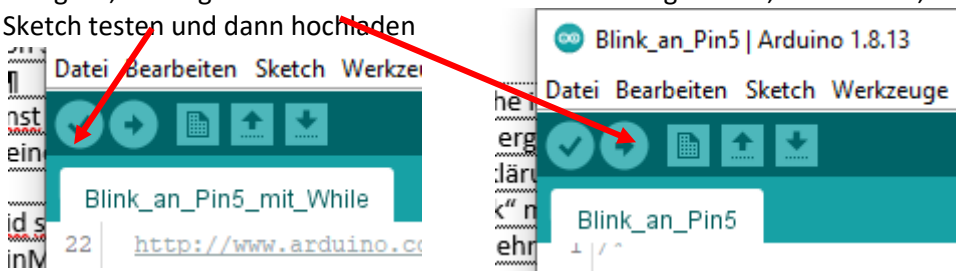


Um nun einen neuen Sketch zu schreiben: DATEI-NEU und es öffnet sich das schon bekannte Fenster.
Jetzt was Neues. Über das Setup legt man in der Regel Variablen und Konstanten an, auf die im Loop zugegriffen wird. Die sind vom Typ global globale und gelten im gesamten Programm. Andere – später.

Ich zeige den fertigen Sketch bezogen auf obigen Schaltplan. Ihr solltet versuchen den mal selber zu schreiben, nicht alles geht mit STRG-C und STRG-V. Außerdem entwickelt man ein Gefühl für die Programmiersprache.

```
/*
  BlinkDings mit 1 LED
  Für ZFI
  von Schorsch
*/
const int LED = 5;
// eine Konstante mit dem Namen LED und dem Wert 5 festlegen

void setup() {
  pinMode(LED, OUTPUT);
  // Den Pin der LED auf AUSGANGs-Modus schalten
}

void loop() {
    digitalWrite(LED, HIGH);   // schreibt auf die LED Ausgangsimpuls AN (1)
    delay(1000);               // wartet (leuchtet)1 Sekunde
    digitalWrite(LED, LOW);    // schreibt auf die LED Ausgangsimpuls  AUS (0)
    delay(1000);               // wartet 1 Sekunde , also 1 Sek lang aus
  }
```

Übrigens, Zeitangaben werden immer in Millisekunden gemacht, z.B. 500 = 0,5 Sekunde
Sketch testen und dann hochladen



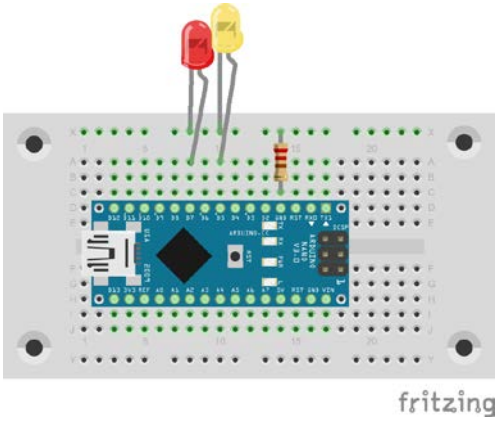
Wer jetzt mehr zu den Befehlen wissen will, dem empfehle ich die Referenz zu lesen.
Und dann mal etwas basteln im Programm. Mit dem Teil lässt sich auf einfache Weise ein Blitzer/Stroboskop machen.
Genug für heute.

1.6. Verschiedene Schaltungen mit LEDs

Da keine Reaktionen eingegangen sind, werde ich noch dieses Kapitel machen. Nur für mich selbst treibe ich den Aufwand nicht weiter.
Erklärungen gibt es nur noch wenige.
Wozu mit LEDs? Es sind Verbraucher, die durch andere (Relais, Motoren, Signale…) ersetzt werden können. Hier geht es ums Prinzip. Also los.

1.6.1 Einfacher Wechselblinker

Schaltplan und Sketch

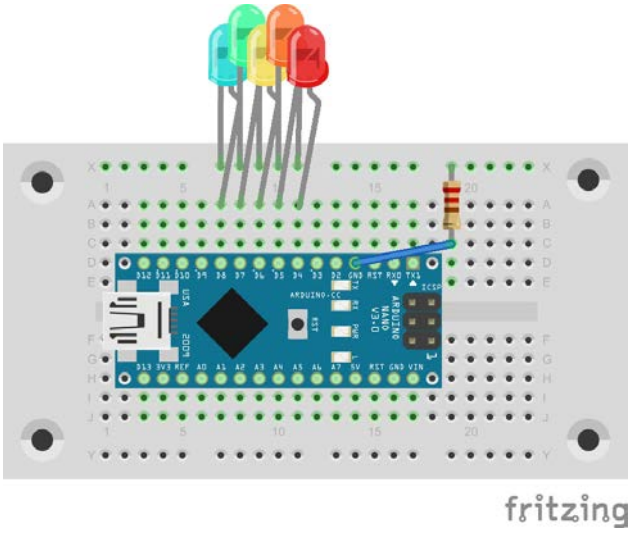



```
/*
 * Wechselblinker, Dioden an D5 und D7
 * für ZFI
 * von Schorsch
 */
int ledGelb = 5;
int ledRot = 7;
int pause = 500;            //Ändert man die Zeit hier,
//ändern sie sich gleichzeitig für beide LEDs

void setup () {
  pinMode (ledGelb , OUTPUT);
  pinMode (ledRot , OUTPUT);
}
void loop () {
  digitalWrite (ledGelb , HIGH);
  digitalWrite (ledRot , LOW);
  delay(pause);             //statt dessen delay(1000);
  digitalWrite (ledGelb , LOW);
  digitalWrite (ledRot , HIGH);
  delay(pause);             // delay(200);
}
```

1.6.2 Einfaches Lauflicht

5 LEDs als Lauflicht, könnte später eine Steuerung für Beleuchtung werden. Das Ganze kann erweitert werden auf bis zu 18 LEDs, wenn man auch die analogen Aus/Eingänge 0-5 nutzt. Dazu später.

Der Sketch:



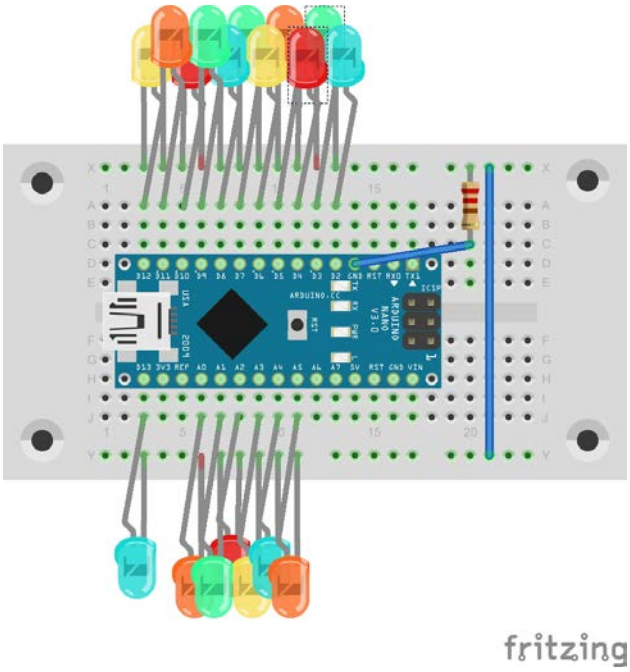

```
/*
  5-Kanal-Lauflicht
  LEDs an den Ausgängen: 4 bis 8
  LED Strombegrenzung: R1 = 220 Ohm
  von Schorsch für ZFI
*/

void setup() {
  pinMode( 4, OUTPUT);
  pinMode( 5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
}

void loop() {
  digitalWrite(4, HIGH);
  delay(200);
  digitalWrite(4, LOW);
  digitalWrite(5, HIGH);
  delay(200);
```

```
 digitalWrite(5, LOW);
 digitalWrite(6, HIGH);
 delay(200);
 digitalWrite(6, LOW);
 digitalWrite(7, HIGH);
 delay(200);
 digitalWrite(7, LOW);
 digitalWrite(8, HIGH);
 delay(200);
 digitalWrite(8, LOW);
}
```

Das ist die umständliche, aber gut lesbare Variante. Man kann die ersten 3 Zeilen im Loop auch einzeilig schreiben (und dann für jeden Pin kopieren und entsprechend 5, 6…8 ersetzen: digitalWrite(4, HIGH);delay(200); digitalWrite(LOW); //damit wird der Sketch kürzer ☺
Legt man vor dem Setup noch eine Variable für delay an, kann man sehr schnell die Lauflichtgeschwindigkeit steuern
z.B.
int zeit = 100
digitalWrite(4, HIGH); delay(zeit); digitalWrite(LOW);

1.6.2 Lauflicht mit 20 LEDs, Schleife und Nutzung analoger Ausgänge

18 LEDs als Lauflicht unter Einbindung der analogen Ausgänge A0 bis A5 und Nutzung einer FOR-Schleife.
Analoge Ausgänge können auch als “digitale“ genutzt werden. Hier wird auch die For-Schleife mit „Hochzähler“ genutzt.




```
/*
 20-Kanal-Lauflicht
 LEDs an den Ausgängen: Digital_0 bis Digital_13 und Analog A0 bis A5
 LED Strombegrenzung: R1 = 220 Ohm
 von Schorsch für ZFI
 */
int i;  //Zählervariable festlegen

void setup() {
 for(i=0; i<20; i++)  pinMode( i, OUTPUT);
 // Schleife beginnt bei 0, zählt im 1-er-Schritt hoch, solange i<20
}

void loop() {
 for(i=0; i<20; i++) { digitalWrite(i, HIGH); delay(100); digitalWrite(i, LOW);
// hier wird der Inhalt von i in die Befehle eingesetzt, rennt sehr schnell
 }
}
```

So kurz kanns werden mit Schleifen.

1.6.3 Lauflicht mal mit hin und zurück, Nutzung vom Zähler negativ

Ganz schnell ein Lauflicht „Knight-Rider-Prinzip“
Was hochzählt,

```
 for(i=0; i<20; i++) {
digitalWrite(i, HIGH); delay(100); digitalWrite(i, LOW); }
```

kann auch runter:
```
for(i=20; i>0; i--) {
   digitalWrite(i, HIGH); delay(100); digitalWrite(i, LOW);  }
```
Also beides im Loop:
```
void loop() {

 for(i=0; i<20; i++) {
   digitalWrite(i, HIGH); delay(pause); digitalWrite(i, LOW);
 }

 for(i=20; i>0; i--) {
   digitalWrite(i, HIGH); delay(pause); digitalWrite(i, LOW);
 }
```
Lässt unser Lauflicht hin und zurück laufen, wobei die Geschwindigkeit als globale Variable in den Kopf-(Deklarierungs)bereich ausgelagert wurde.
```
}
```

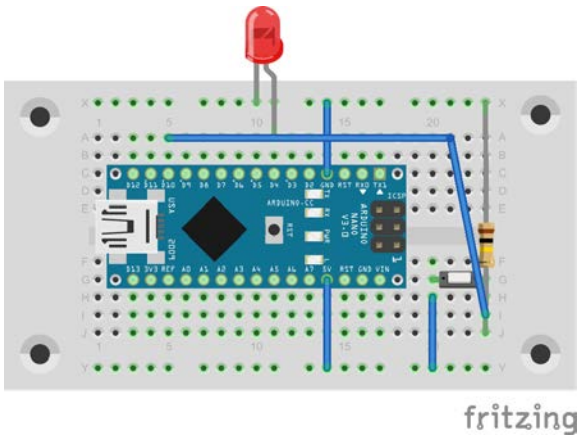
1.7. Schalten und Walten

Alle Sketche rennen sofort los, wenn der Ardu Strom bekommt. Was im Loop() steht, wird dauerhaft wiederholt. Aber der Chef bin ich, also will ich bestimmen, wann was passiert. Dazu braucht man Auslöser, Schalter, Knöpfe, Hebel, also etwas zum Draufditschen(sächs.) oder Ähnliches. Alle Beispiele dienen nur zur Einführung. Es wird im Weiteren auf „Schalter“ vom Typ Reed, Lichtschranke und meinem Favoriten Hallsensor eingegangen. Anwendungen folgen im Kapitel 2 ☺

1.7.1. Taster

Die wohl einfachste Variante

Hier geht es darum, den Zustand eines Tasters auszulesen. Wenn Taste gedrückt, dann LED an. Scheint einfach, aber es gibt dabei zu beachten, dass der Taster „entprellt“ werden muss. Deshalb der Widerstand (kann 10k bis 100K sein) parallel zum Schalter. Den obligatorische LED-Widerstand habe ich weggelassen, ich verwende zum Testen LEDs mit eingebautem Widerstand.




```
// LED wird an Pin 4 angeschlossen .
int led = 4;
// Der Taster wird an Pin 10 angeschlossen .
int taster = 10;

void setup ()
{
// Pin 8 wird als Ausgang deklariert .
pinMode(led , OUTPUT);
// Pin 10 wird als Eingang deklariert .
pinMode(taster , INPUT);
}
void loop ()
{
// Wenn der Taster gedrückt wird:
if( digitalRead (taster) == HIGH)
{
// Schalte die LED an.
digitalWrite (led , HIGH);
}
// Ansonsten:
else{
```

```
// Schalte die LED aus.
digitalWrite (led , LOW);
}
}
```

1.7.2 „magnetische“ Schalter- der Hallsensor

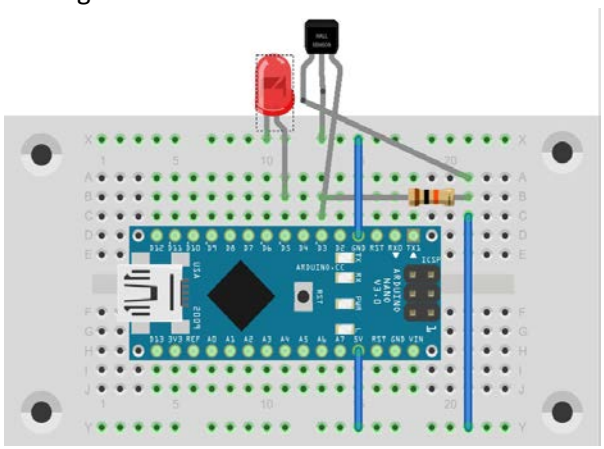
Da sie immer wieder zur Anwendung kommen: Reedschalter oder-kontakte. Klein und relativ zuverlässig. Der Sketch ist der, wie beim Taster, es wird einfach nur statt des Tasters ein Reedkontakt verbaut.

Jetzt kommt mein Liebling, der Hallsensor. Noch kleiner, kann fast unsichtbar verbaut werden und ist unkompliziert. Wer sich nicht sicher ist, welchen Sensor man wohl nehme, der sollte evt. hier schauen. Es gibt 4 Arten und entsprechend unterschiedlich ist der Einsatz. Ich habe in meinem Beispiel einen unipolaren, non-latching

https://funduino.de/wp-content/uploads/2021/01/Licht-schalten_Hall-Sensor.pdf

Der Sketch dazu:

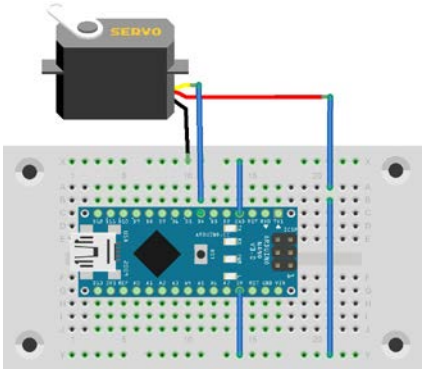
```
const int hallSensorPin = 3; // Pin an dem der Hall-Sensor angeschlossen ist

int hallSensorStatus = 0;
int LED = 5;

void setup() {
  // Pin fuer die interne LED als Ausgang definieren
  pinMode(LED, OUTPUT);
  // Pin fuer den Hall-Sensor als Eingang definieren
  pinMode(hallSensorPin, INPUT);
}

void loop() {
  // Status des Hall-Sensors auslesen
  hallSensorStatus = digitalRead(hallSensorPin);

  if (hallSensorStatus == LOW) {
    // LED einschalten
    digitalWrite(LED, HIGH);
  } else {
    // LED ausschalten
    digitalWrite(LED, LOW);
  }
}
```



1.8 Servos

Servos bringen Leben in die verschiedensten Anwendungen und sind sehr beliebt als Weichensteller. Anleitungen dafür gibt es sehr viele. Ich habe mich an dieses Thema „von klein an“ herangetastet, um dann im Bedarfsfall verschiedenste, leicht anpassbare Sketche zu haben. Ich beziehe mich hier immer auf den einfachen Turmservo:

1.8.1 Einen Servo pendeln lassen



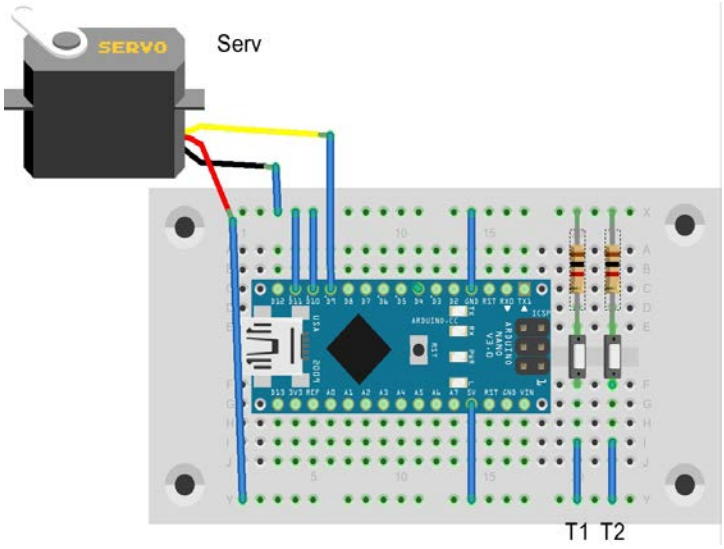
Mit dem nachfolgenden Sketch und diesem einfachen Aufbau habe ich begonnen mir das Verhalten von Servos anzueignen. Er eignet sich hervorragend, um verschiedene Einstellungen auszuprobieren.

```
/* Pendeln eines Servos
 *  verschiedene Positionen und Geschwindigkeiten einstellbar
 *  von Schorsch
 *  für ZFI
 *  08.02.2021
*/

#include <Servo.h>
// einbinden der Servo-Bibliothek

Servo serv;
// legt das Objekt serv an

int pos = 0;    // variable zum Speichern der Servo-Position mit Zuweisung der Position 0

void setup() {
  serv.attach(4);  // gibt den Pin an, mit dem die Steuerleitung des Servos verbunden ist
}

void loop() {
  for (pos = 0; pos <= 90; pos += 1) {
    // zählt den Stellwinkel von 0 auf 90 Grad im 1 Grad-Schritt

    serv.write(pos);
    // sagt dem Sevo die Position aus der variablen 'pos'
    delay(10);
    // wartet 10ms zum Erreichen der Servo-Position
  }
  for (pos = 90; pos >= 0; pos -= 1) {
    // zählt den Stellwinkel von 90 auf 0 Grad im 1 Grad-Schritt
   serv.write(pos);
    delay(10);
  }
}
```

Das sollte klappen. Vorteil dieses Sketches: Man kann die Stellgeschwindigkeit an 2 Stellen variieren: Einmal mal hier:
for (pos = 0; pos <= 90; pos += 1) ändert man pos+=5 Wird der Stellwinkel im 5 Grad-Schritt verändert, d.h. es geht schneller
Oder hier:
   delay(10);   ändert man die delay Zeit z.B. auf 100ms delay(100) kann man die Servogeschwindigkeit ebenfalls beeinflussen. Und das für die Vorwärts- und Rückwärtsbewegung getrennt!!
Jetzt heißt es ausprobieren.
Nachteile: Die Servoposition vor- und rückwärts müssen übereinstimmen (von 0 auf 90 und 90 auf 0. Und: Wenn der Sketch Strom bekommt, kann der Servo „springen“, da die letzte Position bestimmt nicht 0 war beim Ausschalten. Außerdem möchten wir ja vielleicht selbst bestimmen, wann der Servo losgeht (Schalter). Kommt noch.

1.8.2 Servo mit 2 Tastern langsam stellen (Weichensteuerung)

Ein der häufigsten Anwendungen ist der Einsatz eines Servos als Weichenschalter. Auch hier gibt es sehr viel Fertiges, aber wir wollen ja selbst tun. Außerdem kommt jetzt eine erste eigene Methode zum Einsatz, die ich mal näher erkläre. 2 Taster sollen also unseren Servo auf definierte Positionen stellen und das schööön langsam, leicht einstellbar.



Der Sketch:

```
#include <Servo.h>
int W1 = 0;
Servo Serv;
int T1 = 10;
int T2 = 11;

void setup() {
 Serial.begin(9600);
 pinMode(T1, INPUT);
 pinMode(T2, INPUT);
}

int schalten (int PIN, int soll, int ist, int Pause){
 Serv.attach(PIN);
 if (soll>ist){
  for(ist; ist<=soll; ist++){
   Serv.write(ist);
   delay(Pause);
  }
 }
 if (soll<ist){
  for(ist; ist>=soll; ist--){
   Serv.write(ist);
   delay(Pause);
  }
 }
 Serv.detach();
 return soll;
}


void loop() {
 if ((digitalRead(T1)) == HIGH){
  W1 = schalten(9, 40, W1, 50);
 }
 if ((digitalRead(T2)) == HIGH){
  W1 = schalten(9, 0, W1, 200);
 }
}
```

So, jetzt doch mal ein paar Erklärungen.

```
int schalten (int PIN, int soll, int ist, int Pause){
```

Das ist die erwähnte Selbstbaufunktion. Bietet sich an, wenn man sich wiederholende Befehlsfolgen vereinfachen will. Die sollte oberhalb vom Loop stehen.
Hier werden 4 Werte übergeben: Pin des Servos, auf wieviel Grad soll er fahren, bei wieviel Grad ist er aktuell, Pause zwischen den einzelnen Schritten. Die Funktion wird im Loop aufgerufen und dort die Werte übergeben für jeden Taster, also hier für Taster 1

```
if ((digitalRead(T1)) == HIGH){
   W1 = schalten(9, 40, W1, 50);
```

Ich finde das Anlegen eigener Funktionen elegant, um Sketche kurz zu halten.
Damit wäre ein Weiche schaltbar.
Mit einem Nano ließen sich bei Anpassung des Sketches also bis zu 4 Weichen schalten unter Nutzung der D2-D13 Eingänge.
Für noch mehr Weichen (bis zu 16) gibt es eine Erweiterungsplatine PCA9685 von Adafruit
Das soll dann aber im „fortgeschritten“ Teil eingebaut werden.
Wenn‘s noch mehr werden sollen – Fortsetzung folgt.
Viel Spaß bis dahin. Über Rückmeldungen freue ich mich natürlich, Vorschläge und Beispiel eingeschlossen.

Zettige Grüße
Schorsch