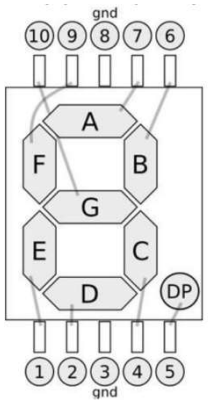
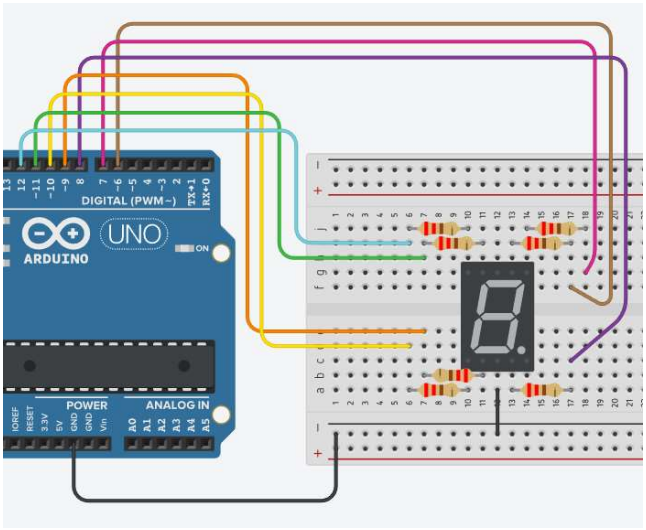
Kapitel B1 – 7-SegmDecoder

Neue Bauelemente: 7 Segment Display
Neue Befehle: Array[], bitRead(z,s);
Ziel: Ziffernanzeige (7 Segment Decoder) mit **Unterprogrammtechnik**



Info: 7 Segmentanzeige
Die 7 Segmentanzeige wird ebenso angesteuert, wie eine RGB-LED. Jedes LED-Segment benötigt einen eigenen 220 Ohm Vorwiderstand. Alle LEDs besitzen eine gemeinsame Kathode (GND).

Aufgabe B1.1:

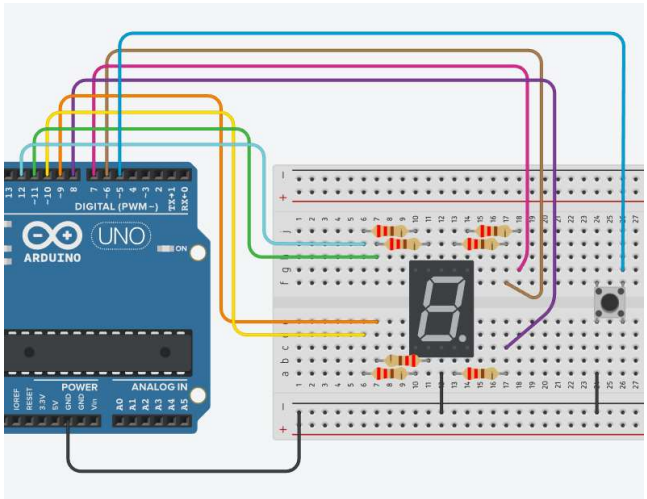


- Setze die 7-Segment-Anzeige, wie dargestellt, auf das Breadboard! Verbinde die LED-Segmente a bis g jeweils mit einem 220 Ohm Vorwiderstand. Den Dezimalpunkt benutzen wir nicht. Achte auf die richtige Reihenfolge bei dem Verbinden zum Arduino: Segment a an Pin6 .. Segment g an Pin 12!
- Schreibe eine Testroutine für die Anzeige, in der jedes Segment in der `loop()`-Schleife der Reihe nach einzeln **ein** und das Vorherige **aus**geschaltet wird. Setze ein `delay(1000);` zwischen die Schritte! Die Segmente sollten Reihum leuchten, zuletzt mittig.
- Speichere den Sketch unter „B1_7Segment“ ab!

Aufgabe B1.2:

- Überlege, welche LED für die Anzeige 0 bis 9 leuchten muss. Erstelle eine Tabelle mit a..g im Tabellenkopf und 0..9 in den Tabellenzeilen. Das Leuchten der LED wird durch 0 und 1 aus der Tabelle dargestellt. Für jede Ziffer lässt sich der Code in einer Hexadezimalzahl darstellen.
- Lösche den Inhalt deiner `loop()`-Schleife!
  Erstelle dafür eine `void decoder(int ziffer){..}` nach deiner loop-Schleife, in der die Segmente a..g mit `digitalWrite()` angezeigt werden! Die leuchtenden Segmente werden pro Ziffer in einem Byte-Array (Liste) bereitgestellt. Auslesen lässt sich die Liste mit dem Befehl `ziffer=digit[n];` und das betreffende Bit mit `bitRead(ziffer,segment);` In der `loop()`-Schleife ruft z.B. `decoder(1);` das Unterprogramm auf und lässt die Ziffer 1 leuchten. Teste andere Ziffern aus deiner Tabelle! Vielleicht mit einer Schleife.
- Speichere den Sketch unter „B1_7SegmentDecoder“ ab!

Aufgabe B1.3: Elektronischer Würfel



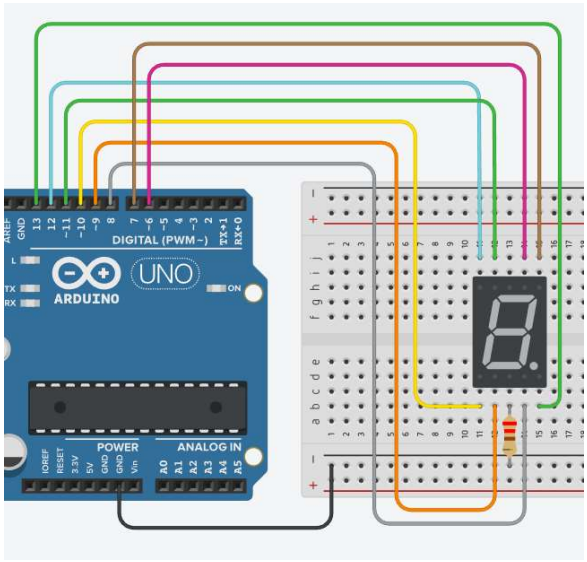
- Ergänze einen Taster am Pin 5 analog der Ampelschaltung!
- Der Zähler soll nur laufen, wenn die Taste betätigt ist und stehen bleiben, wenn sie los gelassen wurde. Wähle ein geeignetes `delay();`!
- Speichere den Sketch unter „B1_7SegmentWürfel“ ab!

Kapitel B2 – Multiplexer

Neue Bauelemente: 7 Segment Display
Neue Befehle: static, millis(), micros()
Ziel: Ziffernanzeige (7 Segment Decoder) mit **Multiplextechnik** und ohne delay()

Info: Multiplexer
Dieses Schaltungsprinzip kommt in technischen Geräten sehr häufig vor. (z.B. Computer, Displays, Telefon usw.) Es wird die Rechenleistung zum Einsparen von Bauelementen und Leitungen genutzt. Wie mit einem Drehschalter wird genügend schnell Bit für Bit zum Ausgang durchgeschaltet, so dass immer nur eine Leitung aktiv ist. (serielle Datenübertragung)

Aufgabe B2.1:



- Setze die 7-Segment-Anzeige, wie dargestellt, auf das Breadboard! Verbinde die LEDs direkt mit den Anschlüssen des Arduino. Nur ein Vorwiderstand 220 Ohm an der gemeinsamen Kathode ist nötig.
- Beachte, dass niemals mehr als ein Segment leuchten darf! Deshalb setzen wir im `setup()` alle Ausgänge auf LOW. Die `void mux()` als Multiplexer muss in der loop()-Schleife ständig aufgerufen und darf nicht durch delay oder andere Warteschleifen unterbrochen werden. Sie bildet faktisch ein Objekt. Ihr wird über eine globale Variable (Attribut) mitgeteilt, was anzuzeigen ist. Verzögerungen im Programm werden über `millis()` und `micros()` erreicht.
- Ersetze in deinem Sketch „7-SegmentDecoder“ die `void decoder()` durch void `mux()`! Ergänze im `mux()` die Verzögerung `micros()` (hier wird die Anzeigefrequenz einstellbar) und das Darstellverfahren der Segmente. Bevor ein neues Segment eingeschalten wird, ist das Vorherige zu löschen!
- Teste mit `mux(8, 1)`; in der loop-Schleife, sowie einem Wert von 100000 für `micros()` (=100ms) in der `void mux()`, um die Darstellung aller Segmente zu prüfen! Prüfe durch Reduzieren der `micros()`, ab wann die Anzeige nicht mehr flackert!
- Teste deine `mux()` mit verschiedenen Zahlen und schalte den Dezimalpunkt mit 0 und 1!
- Speichere den Sketch als „B2_7SegmMux“ ab!

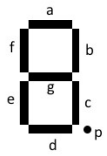
Aufgabe B2.2:

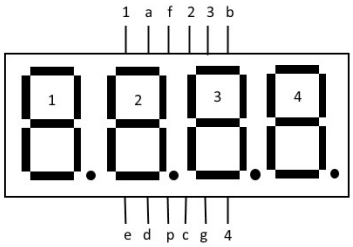
- Ergänze je eine globale Variable für die Ziffer und den Dezimalpunkt, desweiteren eine long-Variable für einen Timer. Wie bei dem elektronischen Würfel kann nun mit der neuen Verzögerungstechnik ein Zähler programmiert werden.
- Speichere den Sketch als „B2_7SegmMux2“ ab!

```
// B2 7 Segment Decoder und Multiplexer
//

void setup(){
  for (int n=6; n<14; n++) {
    pinMode(n,OUTPUT);
    digitalWrite(n,LOW); // alle Segmente aus!
  }
}
void loop(){
  mux(8, 1);
}

// 7 Segment Decoder und Multiplexer
void mux(int ziffer, boolean dezP) {
  static long mTimer;
  static int mSeg = 0;
  byte digit[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x58,0x5E,0x79,0x71};

  if (micros()>mTimer + 100000) { // Anzeigefrequenz
    mTimer = micros();

    digitalWrite(mSeg + 6,LOW); // altes Segm ausschalten

    mSeg++; // nächstes Segment
    if (mSeg>7) mSeg = 0; // von vorn beginnen

    if((ziffer > -1) && (ziffer < 16)) ziffer = digit[ziffer];
    else ziffer = 0; // ungültige Werte blocken
    if(dezP) ziffer = ziffer + 0x80;

    digitalWrite(mSeg + 6, bitRead(ziffer, mSeg)); // neues Segm
  }
}
```

Kapitel B3 – komplexe Anzeige

Neue Bauelemente: 4x7 Segment Display
Neue Befehle: modulo %
Ziel: Multiplex-Prinzip, Nutzung der Unterprogrammtechnik



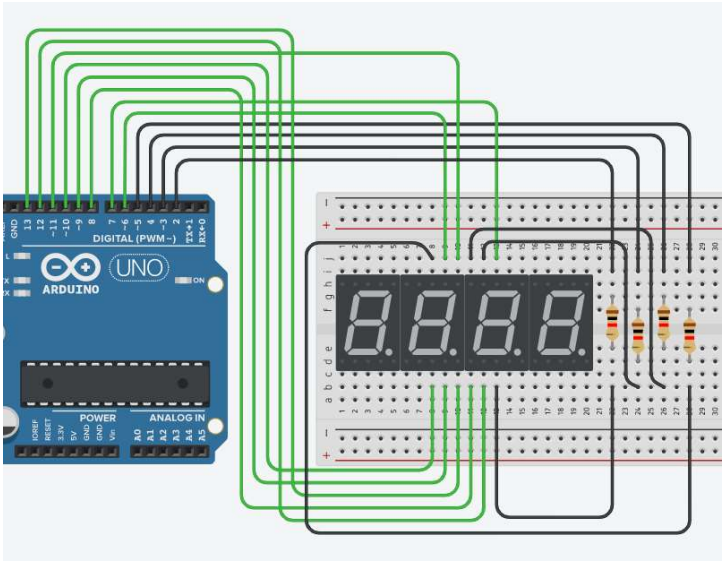


Segm	1	a	f	2	3	b
Ardu	D5	D6	D11	D4	D3	D7

Segm	e	d	p	c	g	4
Ardu	D10	D9	D13	D8	D12	D2

Info: 4x7 Segmentanzeige
Die Anzeige besteht aus 4 eigenständigen Digits. Alle Segmente a..g und der Dezimalpunkt p (Anoden) sind je miteinander verbunden. Jedes Digit hat eine gemeinsame Kathode (1..4). **Es darf stets nur ein Segment leuchten**, indem **einer** der Anschlüsse a..g oder p HIGH und eine der Kathoden 1..4 LOW führt. Würden mehrere Anoden eingeschalten, kann der entstehende Strom den Kathodenanschluss überlasten!

Aufgabe B3.1:



- Setze die 7-Segment-Anzeige auf das Breadboard und daneben 4 x 220 Ohm Vorwiderstände! Für eine bessere Übersicht belasse je eine Leerspalte zwischen den Widerständen. Verbinde zuerst die Kathoden 1 bis 4 über die Widerstände mit D2 bis D5! Verbinde anschließend die LED-Segmente a bis g und p mit dem Arduino von D6 bis D13!
- Überlege, welche Pins wie geschalten werden müssen, um den Dezimalpunkt von Digit1 zu Digit4 wandern zu lassen!
- Teste nun auch die anderen Zifferstellen. Beachte, dass stets nur eine Anode HIGH und eine Kathode LOW führt. Das lässt sich auch mit einer Schleife automatisieren.
- Speichere den Sketch unter „B31_4x7SegmentTest“ ab!

Aufgabe B3.2:

- Wir erweitern den Decoder/Multiplexer aus B2 um die Ansteuerung der Kathoden 1..4. Zusätzlich müssen wir aus einer 4 stelligen Zahl die jeweilige Ziffer gewinnen können. Interessant ist dafür die Nutzung der Modulo-Funktion %, die den Rest einer Division liefert. Nutze dazu das Listing auf der Rückseite!
- Programmiere in der loop-Schleife einen Zähler! Ersetze dafür delay() durch eine Polling-Variante mit millis() analog unserem Multiplexer, die die loop() Schleife nicht ausbremst. Speichere den Sketch unter „B32_4x7SegmentMux“ ab!

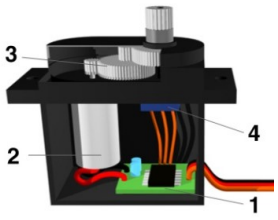
Aufgabe B3.3:

- Mit dem Wissen über die Verzögerungstechnik und der Modulo-Funktion kannst du in der loop()-Schleife eine elektronische Uhr programmieren.
- Speichere den Sketch unter „B33_4x7SegmentUhr“ ab!

```
// B3 4x7 Segment Decoder und Multiplexer
//

void setup(){
  for (int n=2; n<14; n++) pinMode(n,OUTPUT);
  for (int n=6; n<14; n++) digitalWrite(n,LOW); // alle Segmente aus!
  for (int n=2; n<16; n++) digitalWrite(n,HIGH); // alle Digits aus!
}

void loop(){
  mux(1234, 1);
}

// 7 Segment Decoder und Multiplexer
void mux(int zahl, int dezP) {
  static long mTimer;
  static int mSeg = 0; // Segmentzähler 0..7
  static int mDig = 0; // Digitzähler 0..3
  byte ziffer;
  byte digit[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

  if (micros() > mTimer + 100000) { // Anzeigefrequenz
    mTimer = (micros);

    digitalWrite(mSeg+6, LOW);     // altes Segm ausschalten

    mSeg++;                        // nächstes Segment wählen
    if (mSeg > 7) {
       mSeg = 0;                   // von vorn beginnen
       digitalWrite(mDig+2, HIGH); // altes Digit ausschalten
       mDig++;                     // nächstes Digit wählen
       if (mDig > 3) mDig=0;       // wieder 1. Digit
       digitalWrite(mDig+2, LOW);  // neues Digit einschalten
    }

    // zum Segment gehörige Ziffer ermitteln
    if(mDig==3) ziffer = digit[zahl/1000];
    if(mDig==2) ziffer = digit[zahl%1000/100];
    if(mDig==1) ziffer = digit[zahl%100/10];
    if(mDig==0) ziffer = digit[zahl%10];
    if(dezP==mDig+1) ziffer = ziffer + 0x80;

    digitalWrite(mSeg+6, bitRead(ziffer, mSeg)); // neues Segm einschalten
  }
}
```

Kapitel B4 - Servomotor

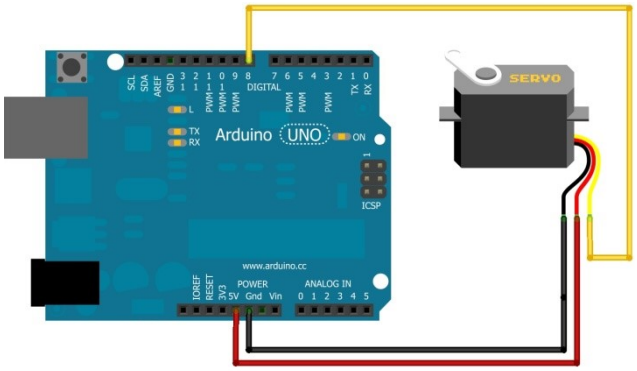
Neue Bauelemente: Servomotor
Neue Befehle: Nutzung von Bibliotheken
Ziel: Ein Servomotor soll sich drehen, wenn eine Taste gedrückt wird.



Info: Servomotor
Ein Servo besteht aus einer Motorsteuerung (1), einem Elektromotor (2), einem Getriebe (3) und einem Potentiometer zur Positionsbestimmung (4). Alle Komponenten sind in einem robusten Gehäuse untergebracht. Angesteuert wird er über einen alle 20ms zu sendenden Impuls von ca. 1-2ms Länge. Die Servo-Library erzeugt die richtige Impulsfolge automatisch, und steuert den Motor von 0 bis 180°.

Aufgabe B4.1:

- Baue die nebenstehende Schaltung auf, ergänze den zugehörigen Sketch und speichere ihn unter B41_Servo_(xx). Beobachte die Bewegung des Motors und vergleiche mit dem Sketch.
- Ergründe die Wirkung der Befehle servo1.attach(8); und servo1.write(0);
- Warum sind die Pausen delay(3000) notwendig?
- Nutze dein Wissen über Arrays, um den Code zu optimieren! Nutze einen Zähler int cnt sowie byte winkel[] = {0, 90, 180, 20}; und im loop() die Abfrage: servo1.write(winkel[cnt]);



Aufgabe B4.2:

- Nutze dein Wissen aus dem Projekt A2 (digitale Eingabe), indem du 2 Taster zum Steuern verwendest. Schließe sie an Digital-Pin 6 und 7 an. Ersetze den Inhalt der loop-Schleife durch die Tastenabfrage. Beachte, dass der Wert in servo1.write() nur zwischen 0 und 180 liegen darf. Dafür benötigst du eine int-Variable „winkel“, die mit den Tasten hoch und runter gezählt wird. Der Befehl winkel - - reduziert die Variable um Eins. Warum reicht ein delay(100)?
- Ergänze die unten stehenden Zeilen um die Taste2-Abfrage und das Hochzählen um Eins mit der Grenze 180. Speichere den Sketch unter B42_Servo_(xx).

```
if(digitalRead(ta1) == 0){
  if(winkel>0) winkel--;
}
servo1.write(winkel);
delay(100);
```

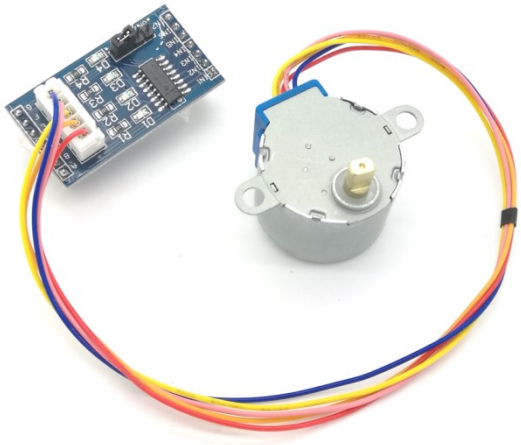
```
//Orange -> Digital 8
//Black -> GND
//Red -> +5V

#include <Servo.h> //Bibliothek
Servo servo1;

void setup(){
  servo1.attach(8);
}

void loop(){
  servo1.write(0); //Winkel 0°
  delay(3000);
  servo1.write(90); //Winkel 90°
  delay(3000);
  servo1.write(180);//Winkel 180°
  delay(3000);
  servo1.write(20); //Winkel 20°
  delay(3000);
}
```

Aufgabe B4.3:

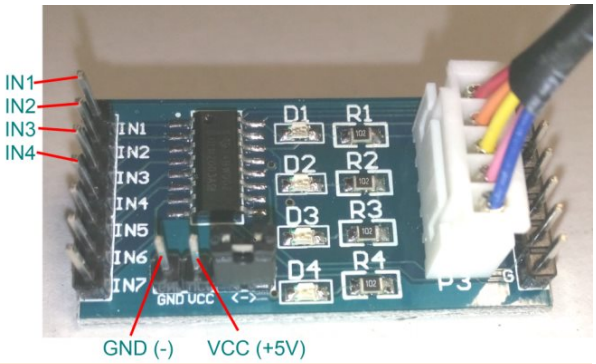
- Jetzt wird die Vorstufe eines Spurverfolgers möglich. Ersetze die Taster durch 2 LDR, wie in Kapitel B3.2 mit je einem Vorwiderstand 10 kΩ. Da für den Spannungsteiler auch die +5V notwendig sind, muss der 5V-Anschluss des Servo mit am Steckbrett angeschlossen werden. Nutze Analog0 und 1 des Arduino!

Kapitel B5 - Steppermotor

Neue Bauelemente: Steppermotor
Neue Befehle: Nutzung von Bibliotheken
Ziel: Ein Steppermotor soll sich drehen, wenn eine Taste gedrückt wird.



```
// In1 -> Digital 2
// In2 -> Digital 3
// In3 -> Digital 4
// In4 -> Digital 5
// -   -> GND
// +   -> +5V

#include <Stepper.h>

Stepper motor(2048,2,4,3,5);

void setup() {
motor.setSpeed(5);
}

void loop() {
motor.step(2048);
delay(1000);
motor.step(-2048);
delay(1000);
}
```



Info: Steppermotor
Ein Steppermotor arbeitet durch seine Impulsansteuerung in exakten Schritten. Der vorliegende Motor teilt eine Umdrehung in 2048 Schritte. Damit lassen sich Winkel oder Umdrehungen genau positionieren. Solche Motoren kommen im CD-Laufwerk, in Druckern, Festplatten, Robotern bzw. Werkzeugmaschinen zum Einsatz.

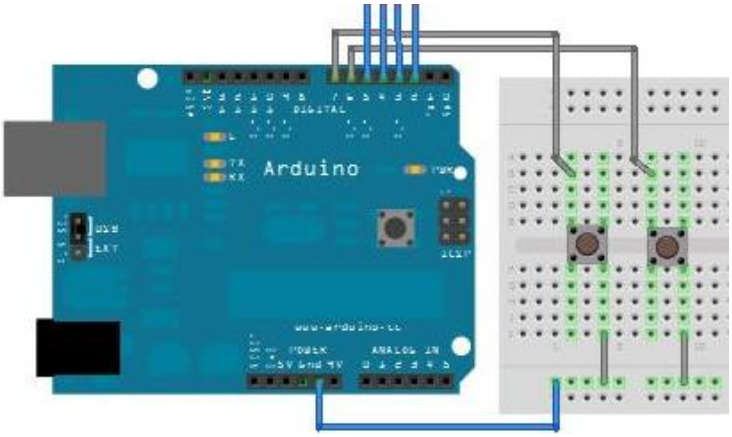
Die Bibliothek für die Ansteuerung von Steppermotoren:
#include <Name.h> bindet die bezeichnete Bibliothek mit ihren neuen Befehlen ein
.setSpeed() stellt die Drehzahl pro Minute ein. Der Wert darf normalerweise für diesen Motor 15 nicht überschreiten! **.step()** legt Anzahl und Richtung der Schritte fest

Aufgabe B5.1:

- Verbinde den Steppermotor mit der Treiberplatine. Nutze das Flachbandkabel zum verdrahten von In1..4 mit den Digitalports 2..5 des Arduino. Verbinde – mit GND und + mit +5V. Baue einen geeigneten Zeiger aus Papier, den du auf die Motorachse steckst.
- Übernehme die oben dargestellte Stepperroutine und speichere sie unter „B51_Stepper_(xx)“ ab. Beachte die Reihenfolge der Pins in der Zeile Stepper Motor(2048,2,4,3,5) !
- Ermittle die Drehrichtung für positive und negative Werte in .step()! Was passiert, wenn du den Wert in .step() veränderst?
- Untersuche die Wirkung im Befehl .setSpeed(). Beachte, dass du nur Werte <=15 verwendest

Aufgabe B5.2:

- Nutze dein Wissen aus dem Kapitel A4, indem du den Motor mit 2 Tastern in Bewegung bringst. Dabei sind die Taster an den DigitalPins 6 und 7 anzuschließen.
- Im Sketch sind im Setup der Befehl pinMode() mit der Eigenschaft INPUT_PULLUP und im Loop der Befehl digitalRead() zu verwenden. Benutze für die Tastenabfrage 2x (ta1 und ta2) if(digitalRead(taX) == 0), der den „Betätigt-Zustand“ anzeigt. Der Motor soll dann nur 5 Steps ausführen.
- Speichere den Sketch unter „B52_Stepper_Tasten_(xx)“.




```
// Taste1  -> Digital 6
// Taste2  -> Digital 7

#include <Stepper.h>

Stepper motor(2048,2,4,3,5);
const int ta1 = 6;
const int ta2 = 7;

void setup()   {
  motor.setSpeed(5);
  pinMode(ta1,INPUT_PULLUP);
  pinMode(ta2,INPUT_PULLUP);
}

void loop() {
  if(digitalRead(ta1) == 0){
    motor.step(5);
  }
  if(digitalRead(ta2) == 0){
    motor.step(-5);
  }
}
```

Aufgabe B5.3:

Damit der Schrittmotor in Maschinen keinen Schritte überspringt (Schlupf), wird er beschleunigt und gebremst, um seine Trägheit zu überwinden und Maximalgeschwindigkeit zu erreichen.

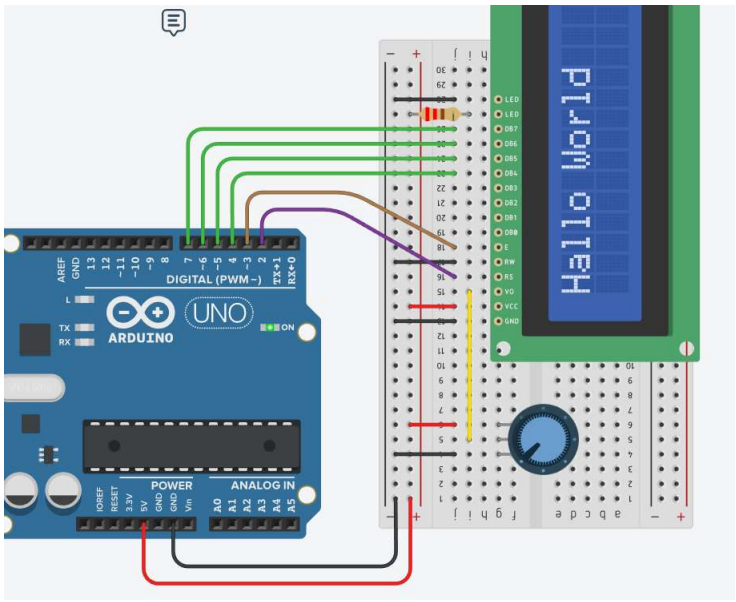
- Teste mit der unten stehenden Routine für eine Drehrichtung die maximal erreichbare Geschwindigkeit für deinen Steppermotor. Die Geschwindigkeit bestimmt setSpeed. Berechne unter Beachtung der Steps für die Beschleunigungen die Anzahl der Schritte mit Maximalgeschwindigkeit, um genau eine Umdrehung (2048 Steps) zu erreichen!
- Speichere deinen Sketch unter „B53_Stepper“ ab!

```
for(int n=8; n<21; n++){
  motor.setSpeed(n);
  motor.step(40);
}
motor.step(1000);
for(int n=20; n>7; n--){
  motor.setSpeed(n);
  motor.step(40);
}
```

Kapitel B6 – LCD Display

Neue Bauelemente: LCD-Display
Neue Befehle: Nutzung von Bibliotheken
Ziel: Das LCD-Display soll Messwerte von zwei LDR anzeigen.

Info: LCD-Display
Unser „**L**iquid **C**rystal **D**isplay“ ist in der Lage, 16 Zeichen in 2 Zeilen darzustellen. Die darstellbaren Zeichen entsprechen dem ASCII-Zeichensatz vom Computer, 8 sind selbst programmierbar. Wir benötigen 6 Leitungen, 4 davon für die Daten und 2 zum steuern. Zusätzlich steuert ein Potentiometer den Kontrast des Displays. Beachte den 220 Ohm Widerstand für die LED Beleuchtung! Die Bibliothek <LiquidCrystal.h> organisiert den kompletten Datenaustausch.



Aufgabe B6.1:

- Verbinde das Display mit dem Arduino gemäß dem Schaltplan.
- Teste den nebenstehenden Sketch, stelle am Potentiometer einen passenden Kontrast ein und speichere ihn unter B61_LCD!

```
// *** LCD ansteuern

#include <LiquidCrystal.h>

LiquidCrystal lcd(2, 3, 4, 5, 6, 7);

void setup() {
  lcd.begin(16,2); // Displaygröße festlegen
  lcd.setCursor(0,0); // Cursor Spalte/Zeile
  lcd.print("Hello, world!"); // Textausgabe
}

void loop() {
}
```

Aufgabe B6.2:

- Verändere die Position der Ausgabe im Display mit lcd.setCursor(s,z). Beachte, dass an erster Stelle die Spalte, beginnend mit 0, und an zweiter Stelle die Zeile kommt!

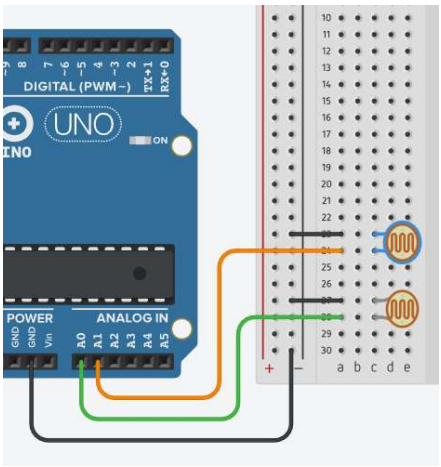
Aufgabe B6.3:

- Stelle in der zweiten Displayzeile eine laufende Uhrzeit dar. Benutze dazu 3 int-Variable für Stunde, Minute und Sekunde. Alle 1000 Millisekunden sollte die Uhr weiter gestellt werden, nutze das Pollingprinzip als „Wecker“ im loop(). Speichere den Sketch unter B63_LCD_Uhr ab!

Aufgabe B6.4:

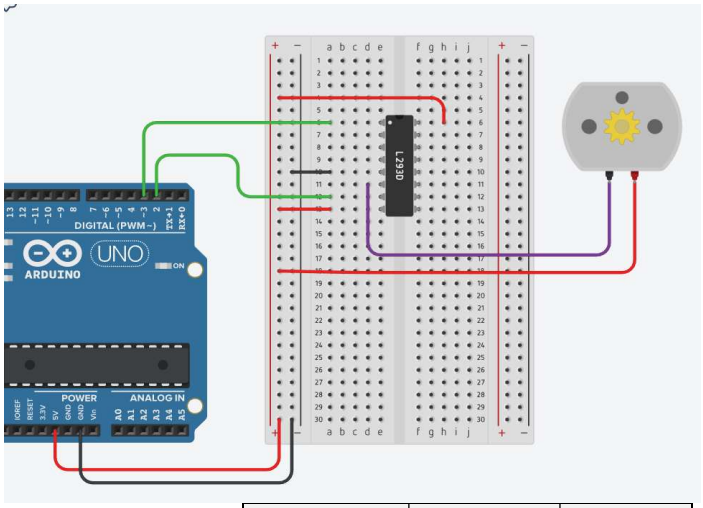
- Mit dem LCD-Display lassen sich leicht Messwerte darstellen. Ergänze deine Schaltung mit umseitig zu sehenden Änderungen. Eigentlich benötigen die LDR PullUp-Widerstände. Es können mit einem Programmiertrick aber auch die integrierten benutzt werden.
- Verändere den Sketch aus B6.3 so, dass im loop() die Messwerte für LDR1 und LDR2 an den Analogeingängen A0 und A1 mit analogRead(x) ermittelt und im Display in der 2. Zeile angezeigt werden. Dafür musst du zwei Variable LDR1 und LDR2 vereinbaren, die du dann mit lcd.print(LDR1) anzeigen lassen kannst. Beachte, dass jede halbe Sekunde eine Messung bzw. eine Ausgabe erfolgen soll. Speichere den Sketch unter B64_LCD_2LDR ab! Die Verwendung von analogRead(x) findest du u.a. in den letzten Arbeitsblättern.

```
void setup() {
  pinMode(14,INPUT_PULLUP); // eigentlich A0
  pinMode(15,INPUT_PULLUP); // eigentlich A1
  ...
}

void loop() {
  ldr1 = analogRead(A0);
  ldr2 = analogRead(A1);
  delay(500);
  ...
}
```



Kapitel B7 - Gleichstrommotor

Neue Bauelemente: L293D, Motor
Neue Befehle:
Ziel: Geschwindigkeit und Richtung eines Gleichstrommotors steuern



Info: L293D
Dieser Chip enthält zwei Motor-Brücken-Treiber. Er kann 2 Gleichstrommotoren unabhängig voneinander in Richtung und Geschwindigkeit steuern. Die Pins des Chips werden vom Punkt auf dem Gehäuse beginnend entgegen dem Uhrzeiger gezählt.

L293D	Brücke 1	Brücke 2
Enable	1	9
Vorwärts	2	15
Rückwärts	7	10
Motor	3, 6	11, 14
+5 V	16	
VCC	8	
GND	4, 5	12, 13

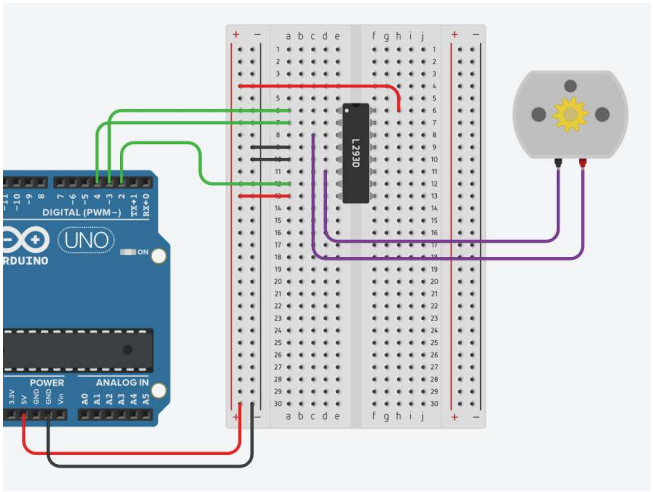
Aufgabe B7.1:

- Baue den Schaltplan entsprechend obiger Darstellung auf. Zur Orientierung nutze die nebenstehende Tabelle für die Anschlüsse des Chips.
- Am Enable-Anschluss, verbunden mit PIN3, sorgt ein HIGH für das Laufen des Motors. PIN2 wird den Motor Ein- (LOW) und Ausschalten (HIGH). Erstelle ein Skript, durch das der Motor 3s dreht und 1s ruht!

Aufgabe B7.2:

- Ersetze die Ansteuerung am PIN3 durch eine PWM (Puls-Weiten-Modulation) mit z.B. analogWrite(3,200); die die Geschwindigkeit des Motors steuert! Teste verschiedene Werte für den 2. Parameter im analogWrite(). Ab wann beginnt der Motor langsam zu drehen?
- Speichere den Skript unter B7_Motor1 ab!

Aufgabe B7.3:



- Ändere deine Schaltung wie nebenstehend ab! Der Motor wird nun durch die volle Brücke angesteuert. Ändere auch deinen Skript, indem du PIN4 zur Ansteuerung hinzufügst. PIN3 liefert nach wie vor per PWM die Geschwindigkeit, PIN2 und PIN4 steuern die Richtung über LOW-HIGH und HIGH-LOW. Der Motor soll 3s in die eine Richtung drehen, 1s ruhen und dann 3s in die andere Richtung drehen und 1s ruhen.
- Speichere deinen Skript unter B7_Motor2 ab.

Aufgabe B7.4:

- Lass den Motor in verschiedenen Geschwindigkeiten in die eine Richtung und nach 1s Pause mit verschiedenen Geschwindigkeiten in die andere Richtung drehen!
- Speichere deinen Skript unter B7_Motor3 ab!

Kapitel B7 – Gleichstrommotor Teil 2

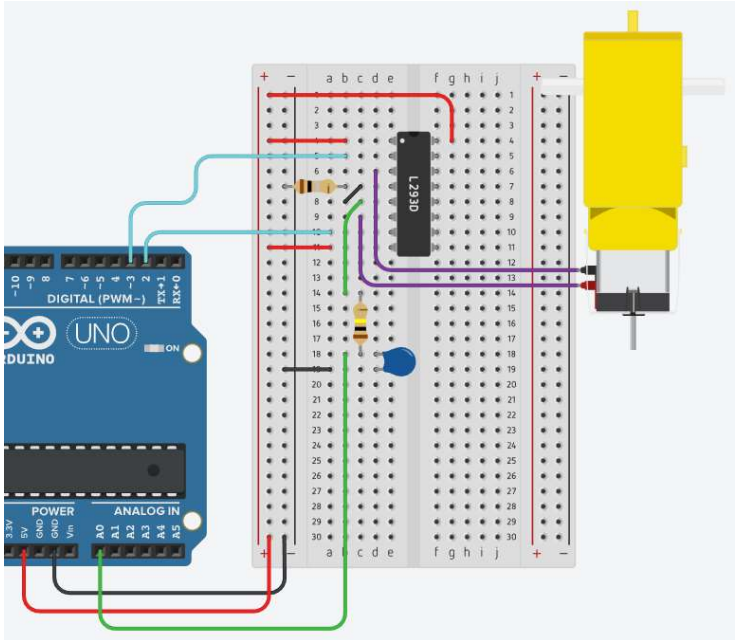
Neue Bauelemente: L293D, Motor
Neue Befehle:
Ziel: Steuern des Motors ohne delay(), eigene PWM, Beschleunigen, Bremsen und Lastmessung

Wir erzeugen unsere eigene Puls-Weiten-Modulation, um ein niederfrequentes analoges Signal für den Gleichstrommotor zu erzeugen. Wie in den anderen Kapiteln gesehen, bilden wir dazu ein über zwei Parameter steuerbares Objekt, das im loop() steht. Speed 0..255 und Richtung 0..1.

Aufgabe B7.5:

- Baue den Schaltplan entsprechend Kapitel 7.3 auf! Ergänze den neben stehenden Skript! Teste das Motorverhalten mit mehreren Geschwindigkeiten und Richtungen.
- Programmiere im loop() einen Zähler (auch mit timer), der für motLi() steigende und fallende Werte liefert. Bei 0 könntest du auch die Richtung umpolen, also aus 1 0 machen und umgekehrt.
- Speichere den Sketch unter B7_Motor4 ab!

```
#define MLV 2    // Motor links vorwärts
#define MLR 3    // Motor links rückwärts

const int pwmFq=100;  // PWM-Frequenz des Motors
                      // 100us*256 Stufen = 25,6 ms
void setup(){
  pinMode(MLV,OUTPUT); pinMode(MLR,OUTPUT);
  digitalWrite(MLV,LOW); digitalWrite(MLR,LOW);
}

void loop(){
  motLi(100,1); // Motor-Objekt mit Speed und Richtg
}

void motLi(byte sp, boolean ri){
  static long pwmTimer;
  static byte pwmSpeed;
  static boolean pwmPhase;
  if(micros()>pwmTimer+pwmSpeed*pwmFq){
    pwmTimer=micros();  // pwm-Wecker
    if(pwmPhase){ // HL-Flanke, Motor abschalten
      digitalWrite(MLV,LOW); digitalWrite(MLR,LOW);
      pwmPhase=false; // Phasenstatus "aus" merken
      pwmSpeed=255-sp; // Wecker für Auszeit stellen
    } else {
      if(ri && sp>0) digitalWrite(MLV,HIGH);
      if(!ri && sp>0) digitalWrite(MLR,HIGH);
      pwmPhase=true; // Phasenstatus "ein" merken
      pwmSpeed=sp; // Wecker für Einzeit stellen
    }
  }
}
```

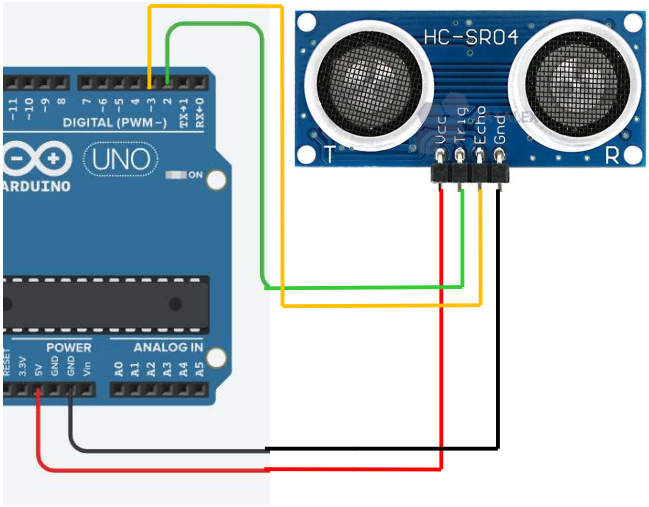


Aufgabe B7.6:

- Ergänze den Schaltplan für eine Strommessung an den GND-Anschlüssen des L293D um die Widerstände 4,7 (ersetzt die GND Verbindung) und 100k Ohm, sowie einen Kondensator 100nF. Die letzten beiden Bauteile bilden einen Tiefpass und verringern Störsignale bei der Messung, sowie schützen den Arduino.
- Es wird eine globale Variable „mess“ benötigt. Ergänze die Zeile mess=analogRead(A0); **vor dem Ausschalten der Motoranschlüsse** in der void motLi(). Im loop() kann nun ständig nach dem aktuellen Strom geschaut werden, z.B. mit einem eigenen Timer, der mess mit Serial.println(mess); zur Anzeige bringt.
- Beobachte den Stromverlauf über verschiedene Geschwindigkeitsstufen hinweg! Warum ist der Stromverbrauch in langsamen Drehzahlen höher? Was passiert bei einem vorsichtigen Bremsen des Motors mit der Hand? Ergänze eine Überlastabschaltung im loop()!
- Speichere den Sketch unter B7_Motor5 ab!

Kapitel B8 - Ultraschallsensor

Neue Bauelemente: HC-SR04
Neue Befehle: switch - case
Ziel: Entfernungen messen



Info: HC-SR04
Der Ultraschallsensor HC-SR04 kann alle 20ms Entfernungen zwischen 2cm .. 300cm messen. Über den TRG Pin wird die Messung gestartet und der ECH Pin liest eine Pulslänge, die direkt proportional zur Entfernung ist, ein. Die Schallgeschwindigkeit beträgt 34,3cm/ms. Daraus ist der Abstand berechenbar:
Distance = Pulslänge in ms * 34,3 / 2.

```
#define TRG 2  // HC-SR04 Trig
#define ECH 3  // HC-SR04 Echo

boolean startHC = false; // set true:
startet Messung, read false: Messung fertig
int readHC;    // Abstand auslesen in cm

void setup(){
  pinMode(TRG, OUTPUT);
  pinMode(ECH, INPUT);
  Serial.begin(115200);
}

void loop(){
  messHC(); // ganze Messung bearbeiten
            // Ergebnis in readHC
  messen();
}

void messen(){
  long timer;
  if(millis()>timer+500){ // Wecker 0.5s
    timer=millis();
    Serial.print(readHC); // Entf darstellen
    Serial.println(" cm");
    startHC = true;  // Messung neu starten
  }
}
```

Aufgabe B8.1: (Fledermaus)

- Schliesse das HC-Modul wie oben beschrieben an! Ergänze nebenstehenden und umseitigen Skript. Prüfe den Abstand zum Sensor mit dem seriellen Monitor. Beachte die hohe Übertragungsrate 115200Bd!
- Warum wird in der Berechnung im Treiber „29.1“ verwendet?
- Warum muss zwingend ein Delay umgangen werden?
- Überlege, weshalb readHC als globale Variable existieren muss.
- Wie verhält sich der Messwert bei schräg liegenden Hindernissen?

Aufgabe B8.2: (Abstandauswertung)

- Erweitere die void messen() um Auswertekriterien, die am seriellen Monitor anzuzeigen sind: „Halt“ bei <= 5cm Abstand, „Bremsen“ bei kleiner werdendem Abstand (20..5cm), „Beschleunigen“ bei größer werdendem Abstand (5..20cm) und „freie Fahrt“ bei Abstand > 20cm.
- Überlege, ob auch stark Bremsen, bzw. stark beschleunigen darstellbar ist.

Aufgabe B8.3: (Gleitender Durchschnitt)

- In der Messroutine wird unter case 2: der Messwert berechnet. Wie kann dort ein gleitender Durchschnitt erzeugt werden, der die Qualität der Messungen erhöht?
- Wie kann in unendlicher Reihe im Hintergrund die Messung automatisiert werden (alle 20ms), sodass wir mit messen() stets einen aktuellen Abstand erhalten?

```
/*************************************************************
*  Polling für HC-Sensor-Steuerung; der ganze Messprozess
*    beide Ports TRG und ECH müssen initialisiert sein
*    benötigt global startHC und readHC
*    startHC LH-Flanke startet Messzyklus
*    startHC HL-Flanke meldet Messung fertig
*    Messwert in cm steht in readHC
*************************************************************/
void messHC(){
  int triggerPin = TRG, echoPin = ECH;
  static long timer;
  static int phase;
  static boolean startOld = false;
  static boolean messOld = false;
  if(startHC>startOld){     // 1. Start der Messung erkennen: LH-Flanke
    timer=micros();                  // Zeit für Triggerpuls merken
    digitalWrite(triggerPin, HIGH);  // Trigger starten
    startOld = startHC;         // Messprozess nun aktiv, Startpegel merken
    phase = 0;
  }
  switch (phase){
    case 0: {
      if(micros()>timer+10){         // 2. Trigger-Ende 10us abwarten
        phase=1; digitalWrite(triggerPin, LOW);  // Trigger abschalten
      }
      break;
    }
    case 1: {
      if(digitalRead(echoPin)>messOld){  // 3. Echostart abwarten LH-Flanke
        phase=2; timer=micros();         // Zeit für Echo-Beginn merken
        messOld = HIGH;                  // Echopegel merken
      }
      break;
    }
    case 2: {
      if(digitalRead(echoPin)<messOld){  // 4. Echoende abwarten
        phase=3; messOld = LOW;          // Echo HL-Flanke erkannt
        long tHelp = (micros()-timer) / 2;
        if(tHelp<6000){ readHC = tHelp / 29.1;}  // Messwert berechnen
        startHC = false;                 // Messung fertig signalisieren
        startOld = startHC;              // Startpegel merken
      }
      break;
    }
  }
}

/* Gleitender Durchschnitt:
       static int messTab[8];
       ..
       int messWert = 0;
       for(int i=7; i>0; i--) messTab[i] = messTab[i-1];
       messTab[0] = tHelp / 29.1;
       for(int i=7; i>=0; i--) messWert = messWert + messTab[i];
       readHC = messWert / 8;
       ..
*/
```