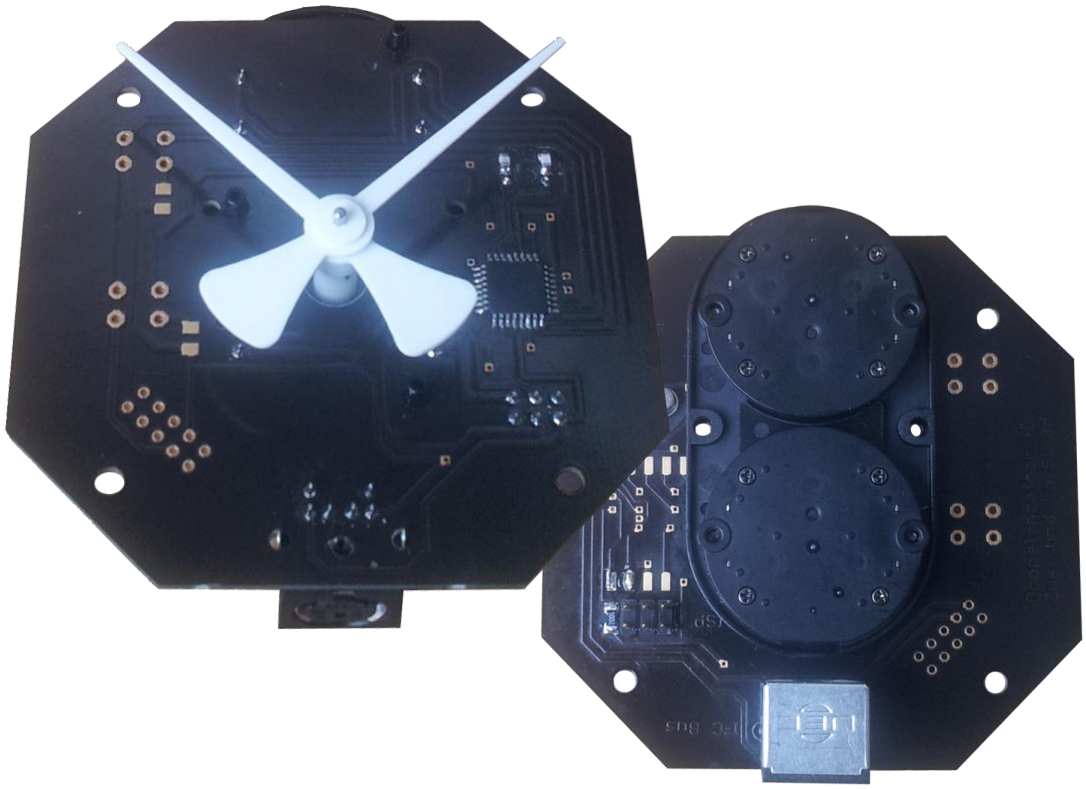
Schrittmotoranzeiger für Simulatoren mit I²C-Anschluss



Funktionsumfang

- Ansteuerung von zwei Schrittmotoren
- Einbau verschiedener Motoren möglich
- Speicherung der Zeigerposition auch nach Abschalten
- Mechanische Zeigersynchronisation durch Anschlag
- Interne Festlegung des Maximalwertes
- Automatische Umrechnung von Anzeigewerten in die Schrittposition
- Konfigurierbare Dämpfung der Zeigerbewegung
- Änderung der I²C-Adresse

Idee

Innerhalb der Simulatorgemeinde ist der Antrieb von Originalen Manometern aus Führerständen immer wieder ein Thema. Hierbei werden häufig Servos aus dem Modellbaubereich bevorzugt. Manchmal kamen auch Schrittmotoren zum Einsatz. In der Regel entwickelt jeder Fahrpultbauer eine eigene Lösung für die gleiche Anwendung.

Beide Ansätze haben zunächst den Nachteil, dass ein Getriebe entworfen werden muss, das die Kraft beider nebeneinander angeordneter Motoren auf einer Doppelwelle zusammen führt. Außerdem sind diese Motoren so groß, dass sie außerhalb des Gehäuses angebracht werden müssen.

Servos sind prinzipiell gut hierfür geeignet. Ihr größter Nachteil besteht aus dem kleinen Winkel (180°), den sie bedienen können. Werden sie mit einem Arduino angesteuert, kann der Servo auch nur in 1° Schritten bewegt werden. Möchte man einen größeren Winkel anfahren, braucht man ein Getriebe. Die Winkelschritte würden dann noch größer.

Viele Schrittmotoren, die für unsere Anwendung zum Einsatz kommen, haben eine große Stromaufnahme, weil sie eigentlich für den Antrieb von schweren Maschinenteilen ausgelegt sind. Das macht eine Leistungselektronik erforderlich.

Meine Idee war eine universelle Lösung zu schaffen, die klein, günstig, direkt am Arduino zu betreiben und universell ist, zu schaffen.
Für den Antrieb von Zeigern gibt es bereits spezielle Schrittmotoren, mit integriertem Getriebe und TTL-Pegel als Versorgung. Diese Motoren kommen im Bahn-, wie im KFZ-Bereich zum Einsatz.
Ich fand den VID28-05, ein Doppelschaftmotor des Chinesischen Herstellers VID.

Abb. 1: VID28-05 (Quelle: http://www.vid.wellgain.com/product.aspx?sortid=26)

Ich entwarf eine Platine, auf der dieser, sowie pinkompatible Motoren mit nur einer Welle anderer Hersteller eingebaut werden können. Z.B. Juken (ehm. Switec) oder MCR.

Um den Arduino nicht, der den Motor betreiben soll, nicht zu viele Anschlusspins abnehmen zu müssen, baute ich auf der Platine einen Kontroller ein, der via I²C-Bus mit Informationen versorgt wird. Diesen Kontroller stattete ich mit einigen Features aus, die weiter unten beschrieben werden.

Elektrischer Anschluss

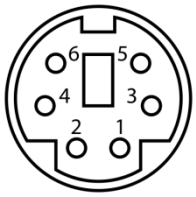
Auf der Platine sind drei Anschlussmöglichkeiten vorgesehen. Zwei für den I²C-Bus und eine ISP-Schnittstelle zum Programmieren des Kontrollers.
Alle beinhalten auch die Spannungsversorgung.

Da der I²C-Bus eigentlich nur für Verbindungen innerhalb eines Designs vorgesehen ist, gibt es keine genormte Schnittstelle. Üblich ist jedoch die Verwendung eines 6-Poligen Mini-DIN Steckers oder ein 10 Poliger Pfostenstecker.

Beide sind im Layout vorgesehen. Mini-Din hat den Vorteil dass dieser Stecker dem PS2-Stecker entspricht und man so sehr gut alte Tastatur- und Mauskabel verwenden kann. Diese Kabel sind sehr Flexibel.

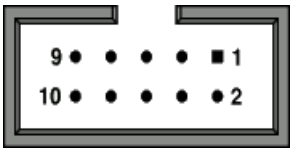
Der Pfostenstecker hat den Vorteil, dass er wenig Platz einnimmt und man mit Flachbandkabeln mehrere Geräte leicht hintereinander setzen lassen können.

Mini-DIN Stecker (I²C)



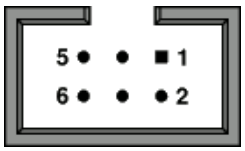
Pin	Belegung
1	SDA
2	*nc*
3	GND
4	+5V
5	SCL
6	*nc*

Pfostenstecker (I²C)

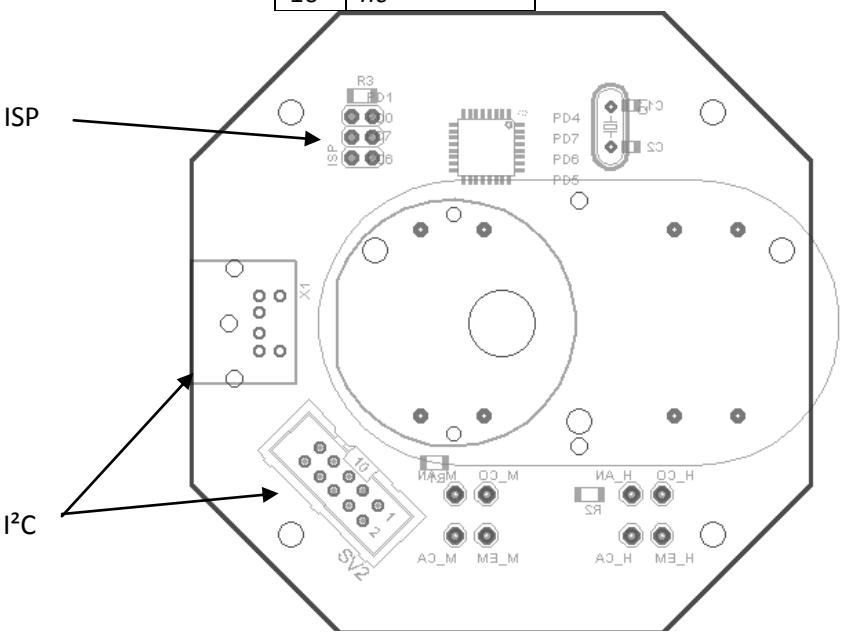


Pin	Belegung
1	SCL
2	*nc*
3	SDA
4	*nc*
5	*nc*
6	*nc*
7	+5V
8	GND
9	*nc*
10	*nc*

Pfostenstecker (ISP)



Pin	Belegung
1	MISO
2	+5V
3	SCK
4	MOSI
5	Reset
6	GND

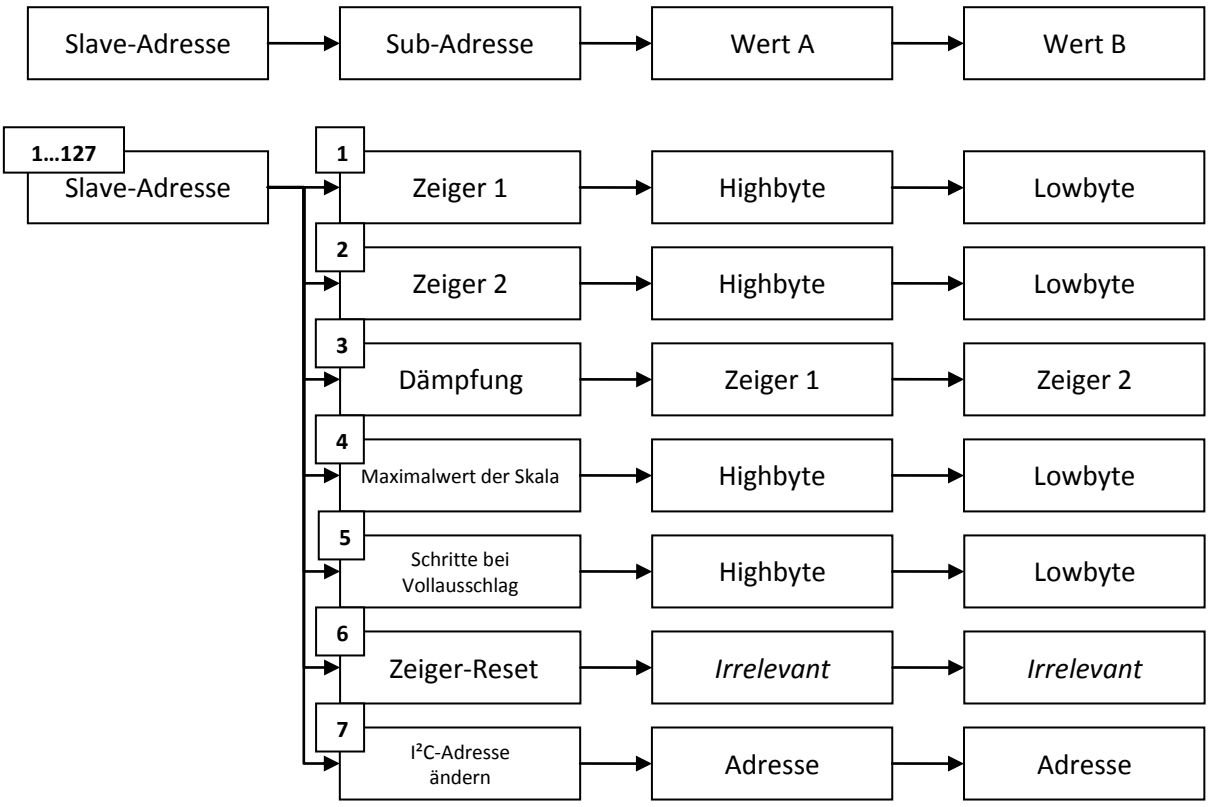


I²C-Protokoll

Das Datenformat besteht aus zwei Teilen. Der Subadresse und der Information.

Die Subadresse besteht aus einem Byte und legt fest, wofür die folgenden Informationen genutzt werden sollen.

Die Informationen bestehen aus zwei Bytes. Entweder zusammenhängend als 16-Bit-Wert oder einzeln.



Setzvorgang für einen Zeiger

An Controller wird der anzuzeigende Wert, multipliziert mit einhundert, übertragen. Hierfür stehen 2 Byte zu Verfügung.

Soll z.B. der Wert 120km/h angezeigt werden, muss 1200 übertragen werden. Diese Zahl muss zur Übertragung in das sog. Hig-Byte und das Low-Byte zerteilt werden.

Hierfür rechnen wir 1200 zunächst in seinen Hexadezimalen Wert um:

$$\mathbf{1200}_{Dec} = \mathbf{4B0}_{Hex}$$

Von unten an, teilen wir den Wert alle zwei Stellen:

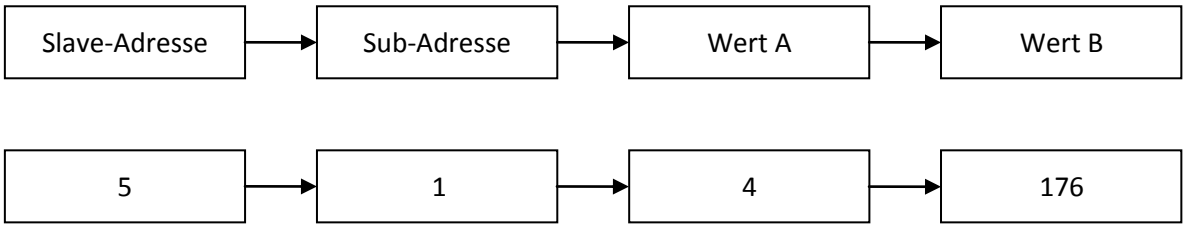
$$\mathbf{04}_{Hex} \mid \mathbf{B0}_{Hex} \qquad \text{bzw.} \qquad \mathbf{4}_{Dec} \mid \mathbf{176}_{Dec}$$

04 Stellt nun das High-Byte dar und B0 stellt das Low-Byte dar.

In diesem Beispiel soll der Zeiger **1** diesen Wert erhalten.

Selbstverständlich müssen wir auch die I²C-Adresse des Mikrocontrollers kennen. Standartmäßig ist dies die 5.

Jetzt haben wir alles, was wir brauchen, um den Zeiger zu setzten:



Typischer Arduino-Code

- Slave-Adresse des Anzeigers: 5
- Höchstwert der Skala: 180km/h (1800 -> 07_{Hex} | 08_{Hex})
- Die Zeiger sollen den Dämpfungsfaktor 100 erhalten
- Vollausschlag, also 180km/h wird nach 920 Schritten erreicht (Typ. 230°)

Festlegen Grundlegender Dinge im Setup:

```
void setup()
{
   Wire.begin();
   Wire.beginTransmission(5);     //Unser Anzeiger hat die Adresse 5
   Wire.write(4);                 //Sub-Adresse 4 (Skalen-Max.-Wert)
   Wire.write(7);                 //Highbyte
   Wire.write(8);                 //Lowbyte
   Wire.endTransmission();

   Wire.beginTransmission(5);     //Unser Anzeiger hat die Adresse 5
   Wire.write(3);                 //Sub-Adresse 3 (Dämpfung)
   Wire.write(100);               //Dämpfung für Zeiger 1
   Wire.write(100);               //Dämpfung für Zeiger 2
   Wire.endTransmission();

   Wire.beginTransmission(5);     //Unser Anzeiger hat die Adresse 5
   Wire.write(5);                 //Sub-Adresse 5 (Schritte bis Vollausschlag)
   Wire.write(3);                 //Highbyte für 920
   Wire.write(98);                // Lowbyte für 920
   Wire.endTransmission();
}
```

Beispiel für eine Funktion zur Übertragung eines Anzeigewertes

```
void SetPointer(word PointerValue)
{
                                  //High- und Lowbyte ermitteln
   byte HighByte = (PointerValue *10) >> 8;
   byte LowByte = (PointerValue *10)&0xFF;

   Wire.beginTransmission(5);     //Unser Anzeiger hat die Adresse 5
   Wire.write(1);                 // Sub-Adresse 1 Anzeigewert 1
   Wire.write(HighByte);          //Highbyte
   Wire.write(LowByte);           //Lowbyte
   Wire.endTransmission();
}
```