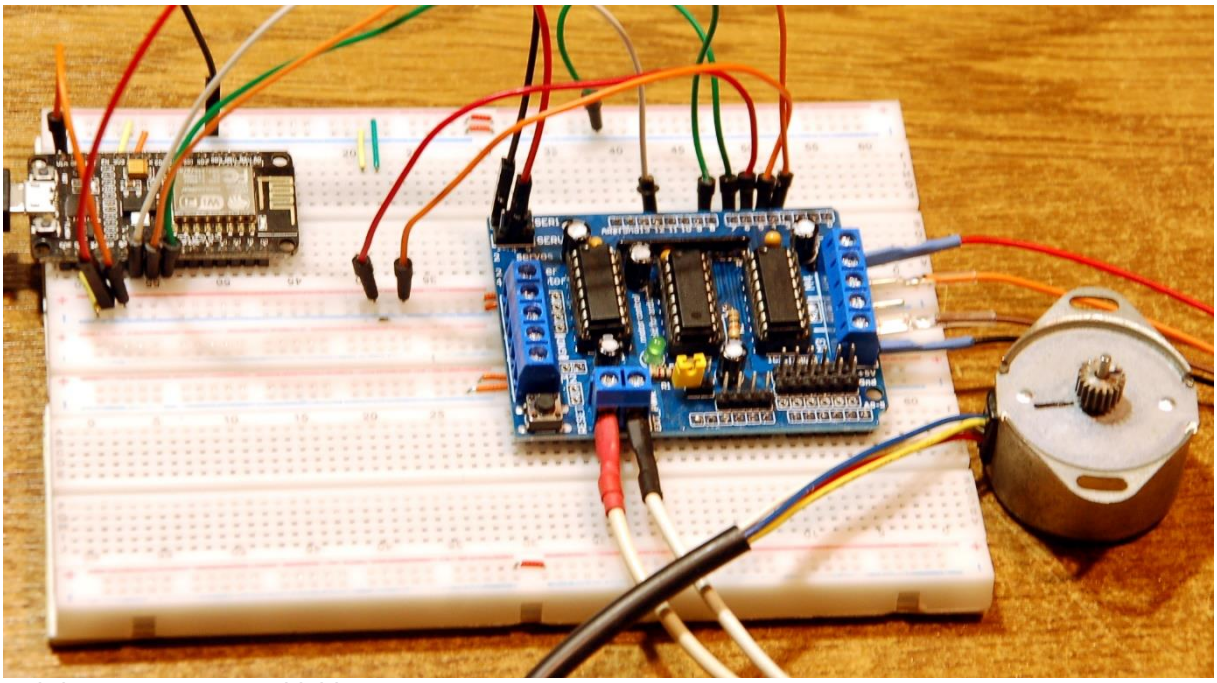

Schrittmotor am Motorshield

Diesen Beitrag gibt es auch als PDF-Dokument zum Download.

Zwei Arten von Elektromotoren habe ich bereits in anderen Blogfolgen vorgestellt. beim Robotcar waren es normale DC-Motoren, die mittels des Arduino-Motorshields über einen ESP32 durch PWM-Signale in der Drehzahl gesteuert wurden. Ebenfalls durch PWM-Signale wurden die Servomotoren beim Spiel Joyballwizzard angesteuert. Dort hatte ich das Treibermodul PCA9685 verwendet. Aber das Motorshield kann noch mehr. Die darauf verbauten Treiberstufen des L293 können auch uni- und bipolare Schrittmotoren mit einer Motorspannung im Bereich von 4,5 bis 36 V und einer maximalen Stromstärke von 600mA antreiben. Wie das funktioniert und welche Bauformen von Schrittmotoren, aka Steppermotoren, es gibt, das verrate ich in dieser neuen Folge von

MicroPython auf dem ESP32 und ESP8266

heute

Umdrehungen Schritt für Schritt

In einem normalen Elektromotor dreht sich ein Anker mit einer oder mehreren versetzten Wicklungen in einem von außen angelegten Magnetfeld. Der Strom wird über einen Kommutator den Ankerwicklungen zugeführt. Dieses Bauteil schaltet durch die Rotation des Ankers automatisch der Reihe nach die verschiedenen Spulenpaare über die Kohlebürsten durch. Mit so einem Motor kann man aber keine bestimmte Winkelposition anfahren, wenigstens nicht ohne weitere Hilfsmittel, wie zum Beispiel einem Winkelencoder. Die Position des Ankers ist nicht gerastert. Es ist nicht vorhersehbar, in welcher Winkelposition der Anker zum Stillstand kommt.

Wenn man nun umgekehrt einen Permanentmagneten in einem äußeren Feld von Magnetspulen platziert, die von einem Steuerteil in einem ganz bestimmten Rhythmus ein- und ausgeschaltet beziehungsweise umgepolt werden, dann lassen sich gezielt bestimmte Winkel der Rotation anfahren und halten. Bei dieser Bauform ist allerdings die Anzahl der Pole des Stators und damit die Feinheit der Winkelteilung räumlich begrenzt. In Abbildung 1 wird im Moment die rechte Spule so vom Strom durchflossen, dass gegenüber dem Anker ein Nordpol entsteht. Der Anker wird dadurch in dieser Position festgehalten. Aktiviert man als Nächstes die untere Spule, mach der Anker eine Vierteldrehung im Uhrzeigersinn und so weiter. Nach vier Schritten hat sich der Rotor um 360 Grad gedreht.

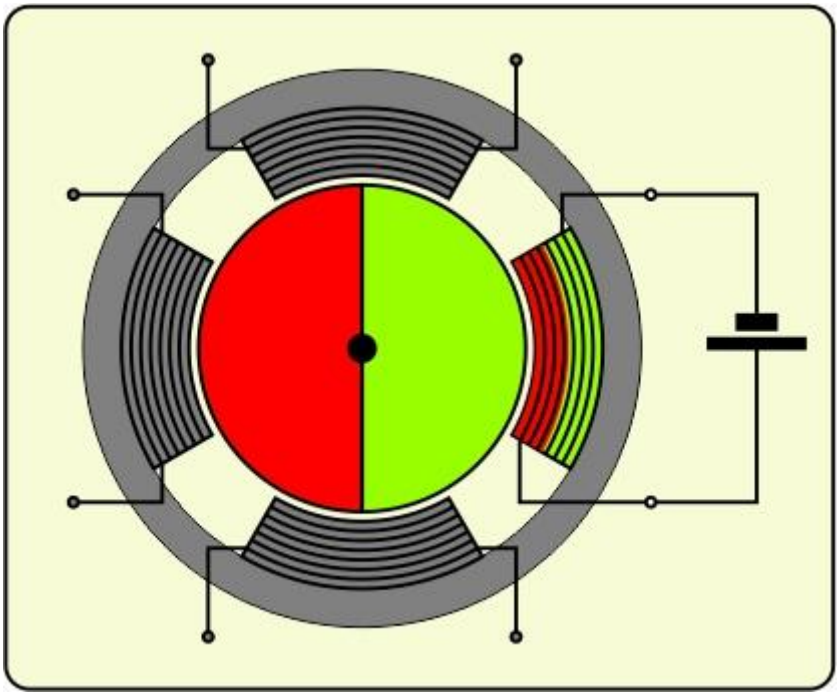

Abbildung 1: Permanentmagnet-Schrittmotor

Eine andere Bauweise verwendet einen gezackten **Weicheisenrotor**, dessen Zacken von den ebenfalls gezackten Feldmagnetpolen angezogen wird. Der Rotor bewegt sich so, dass seine Zacken den Zacken des Statorpols möglichst genau gegenüberstehen und so einen größtmöglichen magnetischen Fluss durch das Weicheisen des Ankers erlauben. Bei dieser Bauform (Reluktanzmotor) lässt sich eine feinere Winkelschrittweite erzielen. Der Begriff Reluktanz beschreibt den magnetischen Widerstand, den die Feldlinien auf ihrem Verlauf überwinden müssen. In Luft ist er sehr groß, in ferromagnetischen Stoffen wie Eisen, ist er sehr klein. Beim Aktivieren eines Statormagneten begibt sich der Rotor in eine Position, in der die Reluktanz minimiert wird. Das ist der Fall, wenn sich die Zacken von Stator und Rotor, nur durch einen schmalen Luftspalt getrennt, gegenüberstehen.

Dadurch, dass der Rotor aus Weicheisen ist, gibt es allerdings kein Haltemoment im stromlosen Zustand.

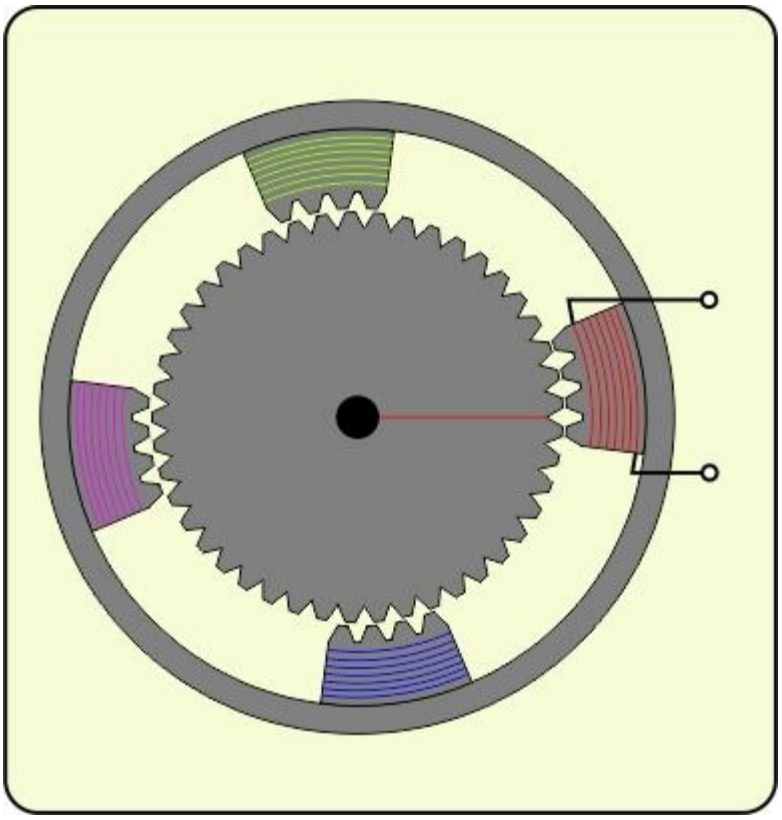

Abbildung 2: Reluktanzschrittmotor

Der Rotor in Abbildung 2 macht beim Aktivieren der nächsten Spule jeweils einen Winkelschritt von 2 Grad. Daraus ergeben sich 180 Schritte pro Umdrehung. Nach vier Schritten liegen wieder vier Zähne des Rotors wieder genau denen der roten Spule gegenüber.

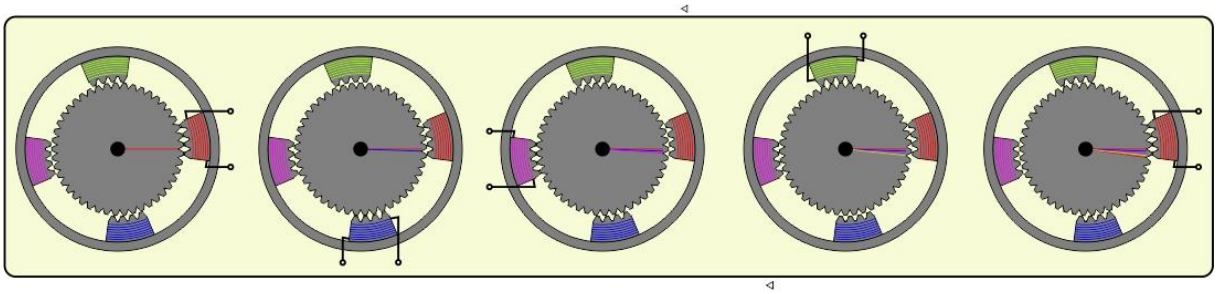

Abbildung 3: Reluktanzschrittmotor - Schrittfolge

Schaltung der Spulen

Unabhängig von der Hardwarebauform eines Schrittmotors können die Spulen unterschiedlich geschaltet und angesteuert werden.

Unipolarmotoren

In Abbildung 4 haben wir einen Unipolarmotor. Der heißt so, weil die Spulen alle nur ein Magnetfeld einer Ausrichtung erzeugen, sie werden nur ein- und ausgeschaltet.

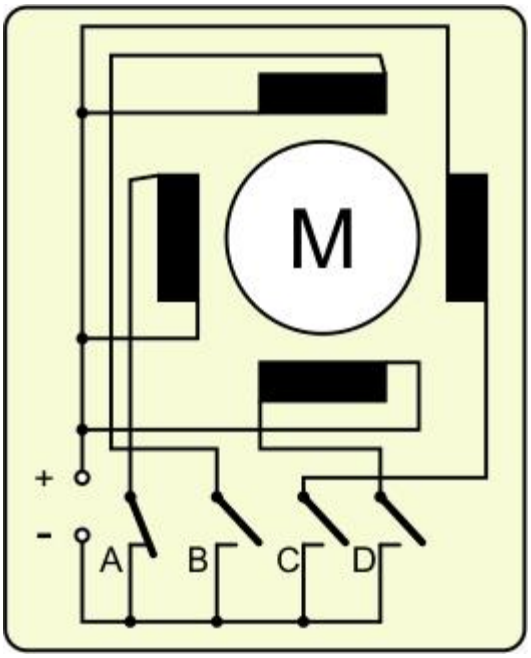

Abbildung 4: Unipolarmotor

Durch das Schließen und Öffnen der Schalter A bis D reihum führt der Rotor vier Schritte aus. Welche Winkel sich daraus ergeben, hängt von der Bauweise des Motors ab.

Die Schalter werden im echten Leben natürlich durch Transistoren ersetzt, die von einem Microcontroller angesteuert werden. Der Controller gibt den Takt vor und bestimmt durch die Abfolge der Steuersignale die Drehrichtung des Motors – A-B-C-D-A-B… im Uhrzeigersinn A-D-C-B-A-D… im Gegenuhrzeigersinn. Die Transistoren müssen lediglich die Motorspannung und den Spulenstrom vertragen, die Ansteuerung ist recht einfach. Zum Schutz des Transistors muss parallel zur Spule eine Freilaufdiode geschaltet werden, welche die Spannungsspitzen beim Ausschalten des Spulenstroms abfängt.

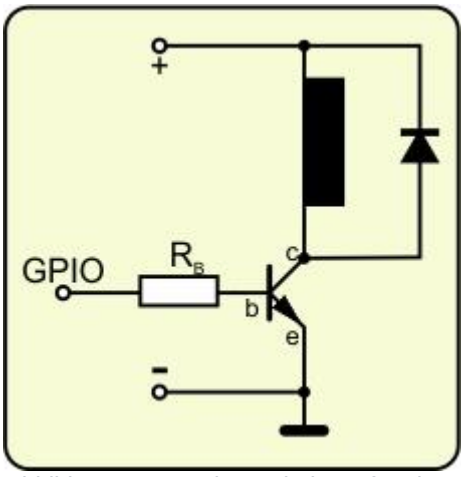

Abbildung 5: Transistorschaltung für eine Spule

Für die Motoren, die ich verwende (bis 15V und 300 bis 500mA), würde locker ein BC337 (45V, 800mA) oder ein Darlingtontyp, zum Beispiel BC517 (30V, 1A) genügen. Weil wir aber in der Hauptsache bipolare Motoren einsetzen werden, lösen wir das Ansteuerproblem anders, dazu gleich mehr.

Bipolare Motoren

Bei dieser Motorklasse werden die Spulen nicht nur ein- und ausgeschaltet, sondern auch noch umgepolt. Dazu braucht man noch einmal vier Schalter. Um die linke Spule auszuschalten genügt das Öffnen von Schalter A oder D, umgepolt wird das Magnetfeld der Spule, wenn A und D geöffnet und dafür B und C geschlossen werden. Ähnlich verhält sich das bei der zweiten Wicklung.

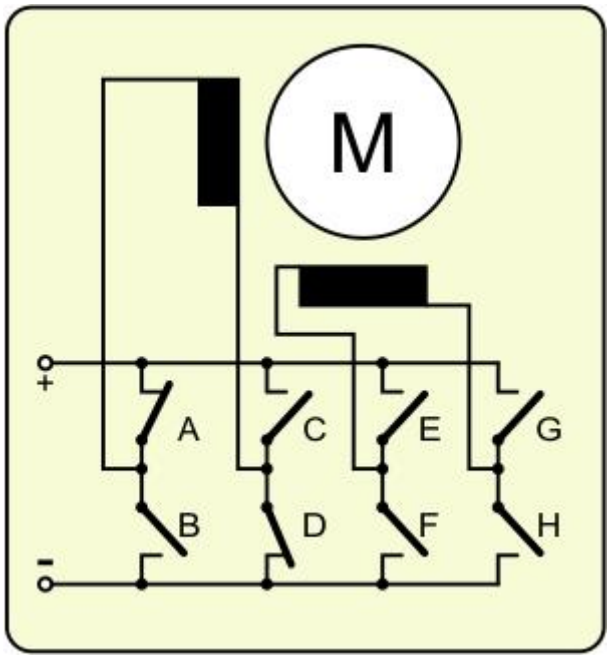

Abbildung 6: Bipolarmotor

Schwierig wird es beim Austausch der Schalter durch Transistoren. Für die Schalter A, C, E und G am heißen Ende, also gegen den Pluspol wird die Ansteuerung durch einen GPIO-Anschluss aufwendiger und komplizierter, gerade bei Motorspannungen von 5V und mehr. Ohne zusätzliche Vorstufe, welche die 0V bis 3,3V am Controllerausgang auf den Plus-Pegel der Motorspannung anhebt, geht da gar nix.

Ich wollte mir das Gepfriemel sparen, habe mich daher für das Arduino-Motor-Shield mit seinen beiden L293D-Chips entschieden. Ein L293D ist ein IC, das zwei Vollbrücken in Form von jeweils zwei getrennten Halbbrücken zur Verfügung stellt. Eine Halbbrücke sieht vom Prinzip her so aus, wie die oben dargestellten Schalter A und B, nur eben mit Transistoren. Wir schauen uns das jetzt genau an.

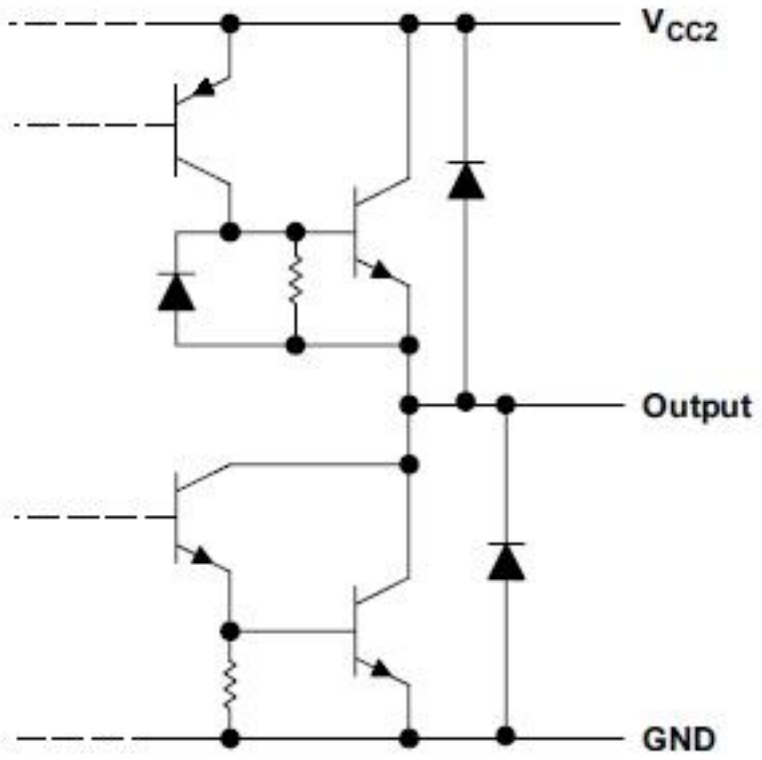

Abbildung 7: Ausgangsstufe (Datenblatt des L293D)

Wie die untere und die obere Stufe angesteuert werden, muss uns nicht interessieren, wichtig ist nur, dass die beiden NPN-Endstufentransistoren getrennt durchschalten und sperren können und, dass das über zwei Eingänge angesteuert wird, die Logikpegel bis fünf Volt benötigen. Das entspricht dem Schließen und Öffnen der Schalter. Wichtig ist auch, dass bereits jede H-Brücke des L293D mit Freilaufdioden abgesichert ist. Der L293 (ohne D) hat diese Dioden nicht auf dem Chip integriert. Daher kann man einen L293D nicht ohne weiteres mit einem L293 ersetzen.

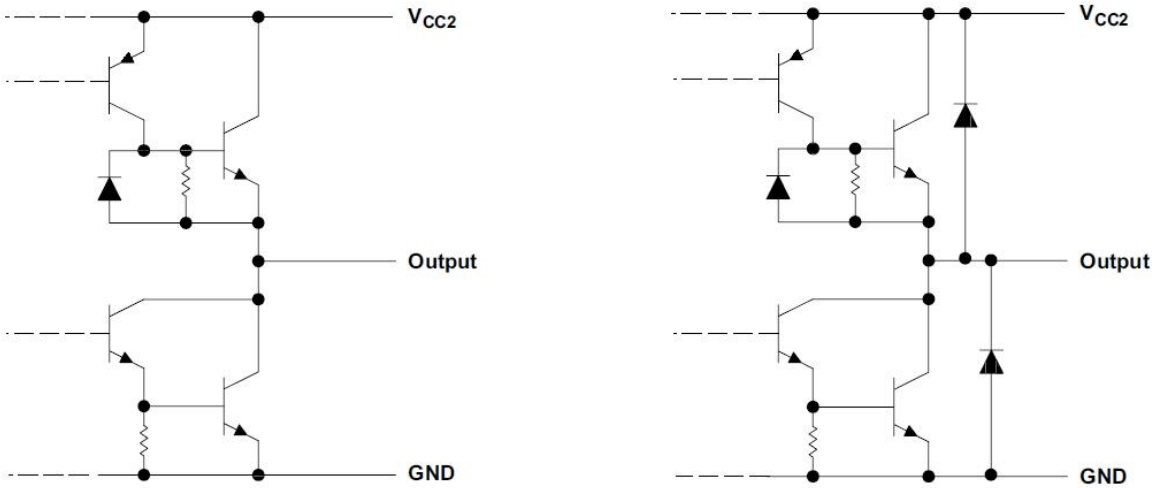

Abbildung 8: L293 - Endstufen H-Brücke - L293(links) und L293D (rechts) (Datenblatt des L293D)

Der Chip kann Motorspannungen bis 36V bei Stromstärken bis 600mA schalten und das bei normalen Gleichstrommotoren, Unipolar- und Bipolarschrittmotoren. Für die Schaltlogik ist dazu eine weitere Spannung Vcc1 von 4,5V bis 7V nötig. Bedeutsam für uns ist, dass die Steuereingänge eine Spannung von 2,3V bereits als logische 1 erkennen, wenn wir als Versorgungsspannung 5V anlegen. Wir könnten also einen

Steuereingang des L293D ohne weiteres mit einem GPIO-Ausgang des ESP32 oder ESP8266 bedienen.

"Könnten" deshalb, weil einerseits die beiden Halbbrücken gleichzeitig umgeschaltet werden müssen und andererseits auf dem Motorshield noch ein weiterer Chip verbaut ist, von dem die Halbbrücken letztlich angesteuert werden. Aber gerade dieser SN74HC595, ein Schieberegister, ermöglicht es uns, die erste Bedingung auf einfache Weise zu erfüllen. Ein ESP kann das direkt nicht bewerkstelligen, weil wir die GPIOs nur nacheinander schalten können. Die Abbildung 9 zeigt einen Ausweg, der aber ein zusätzliches IC mit Invertern am Eingang verwendet.

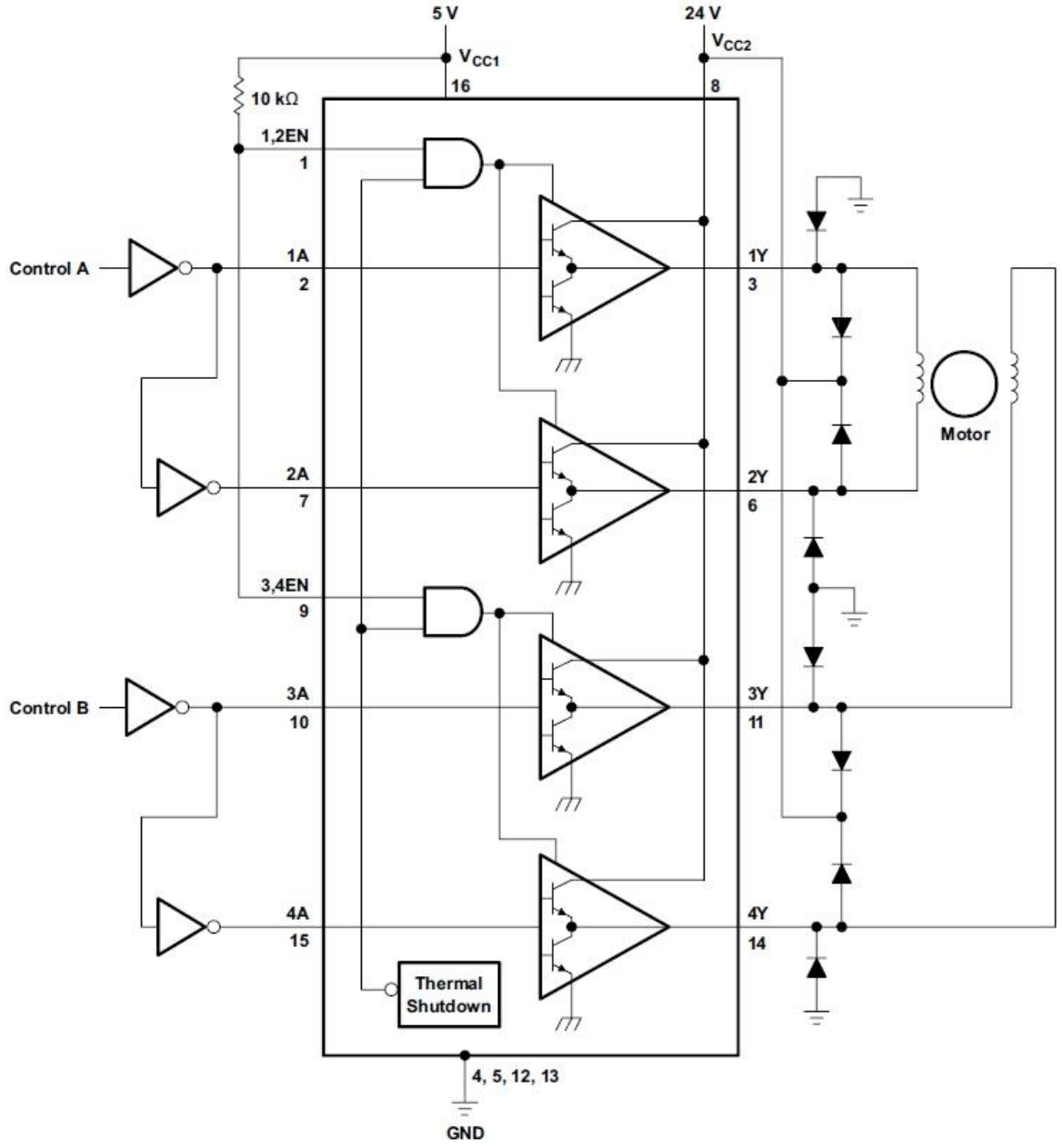

Abbildung 9: L293 - Bipolar Schrittmotor- Ansteuerung (Datenblatt des L293D)

Ein Nachteil dieser Schaltung ist, dass die Ausgänge stets unterschiedliches Potenzial haben. Keine der Spulen kann also stromlos geschaltet werden.

Abbildung 10 zeigt auch den Anschluss eines bipolaren Schrittmotors mit den Spulen L1 und L2 an den L293D.

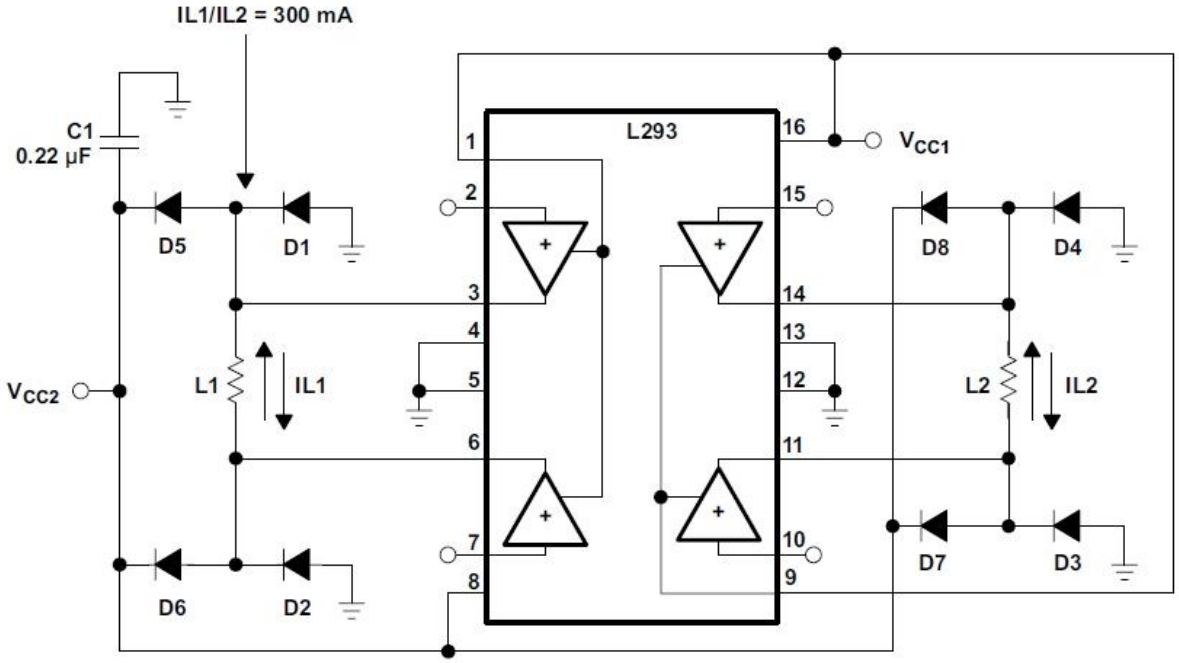

Abbildung 10: L293 - Anschluss eines bipolaren Schrittmotors (Datenblatt des L293D)

In diesem Fall können wir über die Eingänge an den Pins 2, 7, 10 und 15 selbst verfügen. Die Pins 1 und 9 sind Enable-Eingänge, das heißt, dass die Halbbrücken aktiviert werden, wenn die Pins auf logisch 1 gezogen werden. Auf dem Motorshield sind diese Eingänge getrennt herausgeführt. Die komplette Pinbelegung der ICs ist in Abbildung 11 ersichtlich.

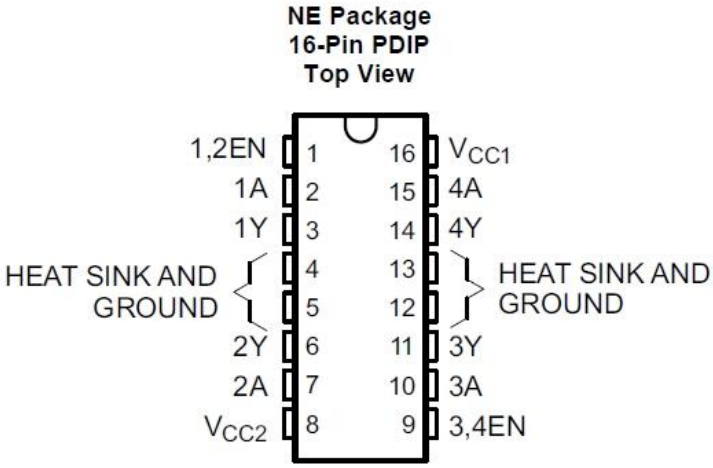

Abbildung 11: L293 – Pinbelegung (Datenblatt des L293D)

Die Eingänge 1A, 2A, 3A und 4A werden von den Pins QA (15), QG (6), QH (7) und QF (5) des SN74HC595 angesteuert. Das geht etwas durcheinander und hat wohl mit der Leiterbahnführung zu tun.

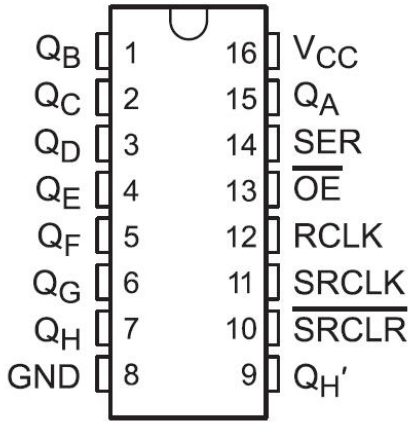

Abbildung 12: 74 HC 595 – Pinbelegung (Datenblatt des SN74HC595)

Für die Übertragung von Bytes an das Schieberegister brauchen wir die Eingänge SER (14), RCLK (12) und SRCLK (11). Die Datenbits werden an SER gelegt und mit der positiven Flanke an SRCLK im 74HC595 übernommen. Mit jedem Takt werden bereits empfangene Bits weitergeschoben. Sind alle acht Bits im Schieberegister angekommen, übernehmen wir alle simultan mit einem Puls an RCLK (Ripple Clock) in die Ausgangs-Flipflops. -OE (Output Enable) legen wir fest auf GND-Potenzial und damit erscheinen die entsprechenden Logikpegel auch an den Q-Ausgängen und werden gleichzeitig an die Eingänge der beiden L293D weitergeleitet. Jeder der beiden kann also einen Schrittmotor bedienen. Abbildung 13 gibt Auskunft über das Innenleben des 74HC595.

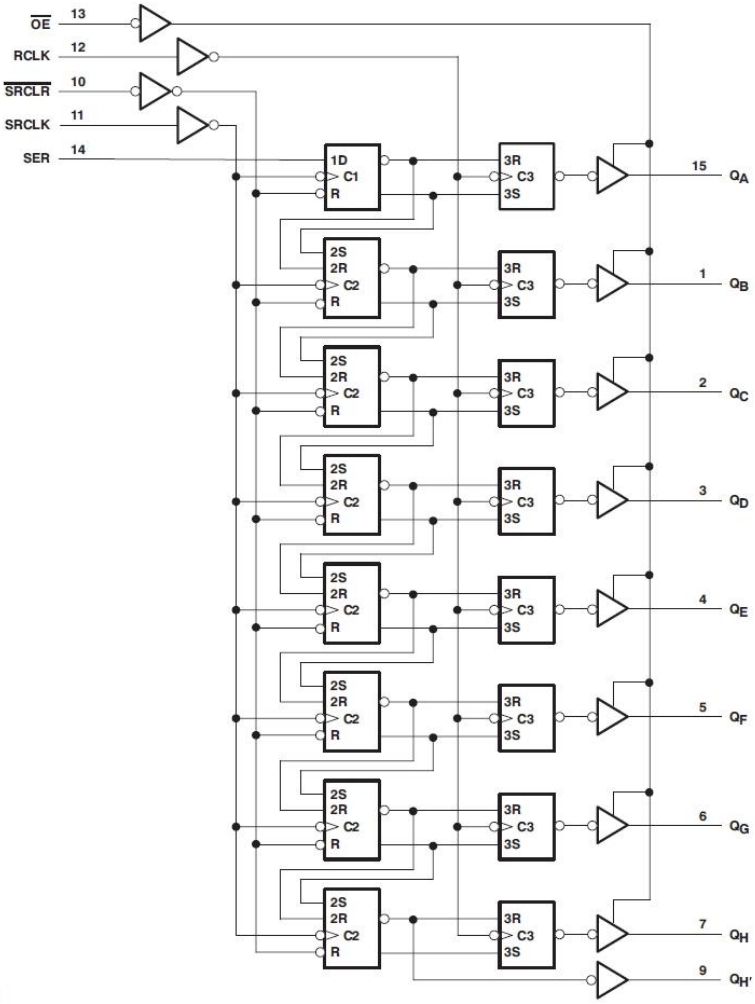

Abbildung 13: Schieberegister 74 HC 595 (Datenblatt des SN74HC595)

Hardware

1	ESP32 Dev Kit C unverlötet oder ESP32 NodeMCU Module WLAN WiFi Development Board oder NodeMCU-ESP-32S-Kit oder **NodeMCU Lua Amica Modul V2** oder ESP8266 ESP-01S WLAN WiFi Modul oder D1 Mini V3 NodeMCU mit ESP8266-12F
1	4-Kanal L293D Motortreiber-Shield
1	Breadboard Kit - 3x Jumper Wire m2m/f2m/f2f + 3er Set MB102 Breadbord kompatibel mit Arduino und Raspberry Pi - 1x Set
1	Schrittmotor uni- oder bipolar z.B. Pollin Best.-Nr. 310689 oder 310690
diverse	Jumperkabel
Optional	Logic Analyzer

Fügen wir die Teile jetzt zusammen. Abbildung 14 zeigt die Schaltung des Motor-Shields. Für uns sind die Positionen der Anschlüsse zu den einzelnen ICs wichtig. Die gestichelten Leitungen führen zum linken L293D, den wir nicht benutzen. Die Enable-Leitungen des rechten L293D sind an den Pins 5 und 6 am Shield herausgeführt, die legen wir an +Vcc1 also +5V. -OE des SN74HC595, an Pin 7 des Boards, verbinden wir mit GND.

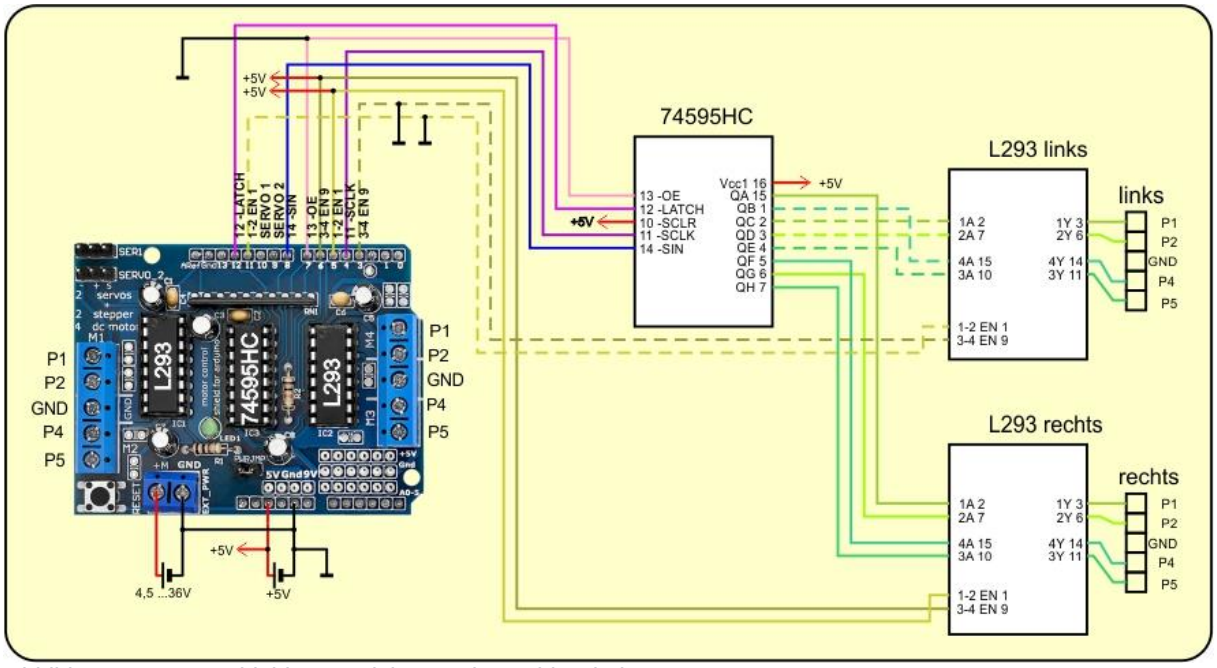

Abbildung 14: Motorshield - Innenleben und Anschlussbelegung

SER, SRCLK und LATCH=RCLK gehen an den ESP8266.

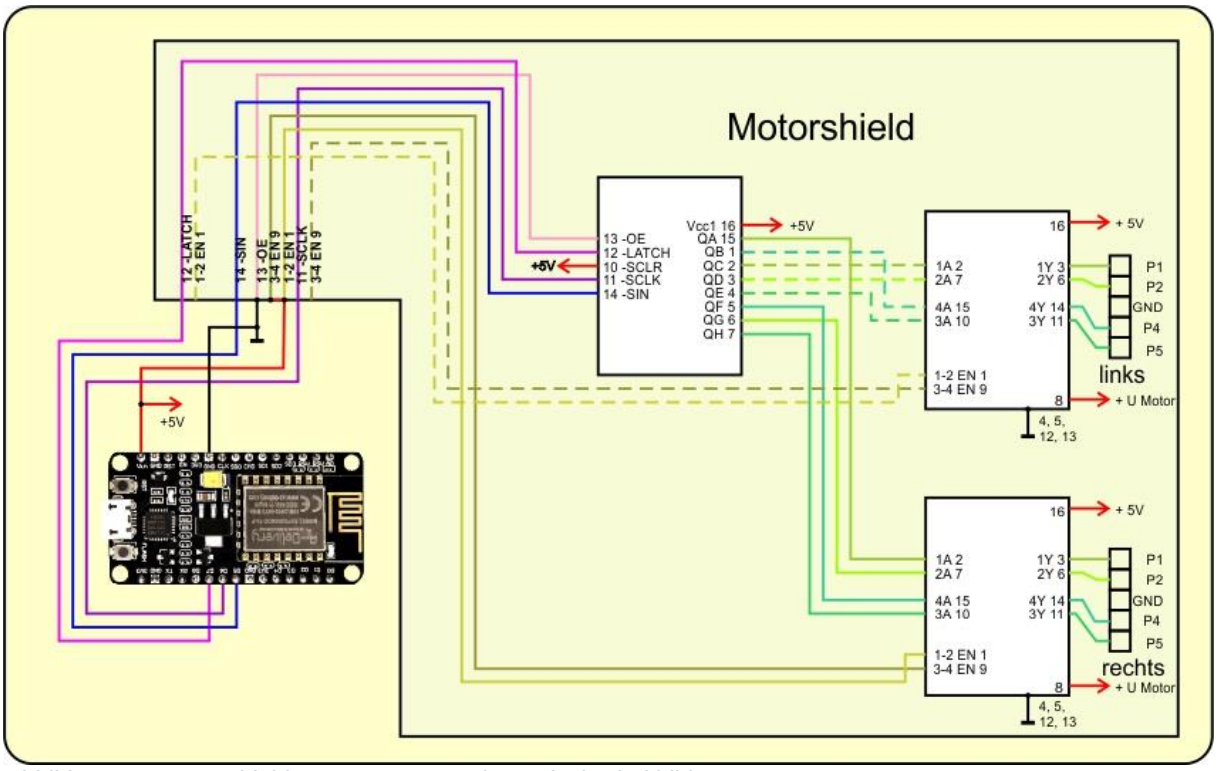

Abbildung 15: Motorshield am ESP8266-Amica - Blockschaltbild

Jetzt fehlt nur noch der Motor. Einige Exemplare sehen Sie in Abbildung 16. Die in der oberen Reihe sind, bis auf den ganz rechts, bipolare Typen von Pollin. Der rechte ist ein unipolarer Motor aus meinem Sammelsurium. In der unteren Reihe liegen Motoren, die ich aus alten Druckern oder Scannern ausgebaut habe.

Abbildung 16: Verschiedene Schrittmotoren

Und so wird der Motor mit dem Shield verbunden.

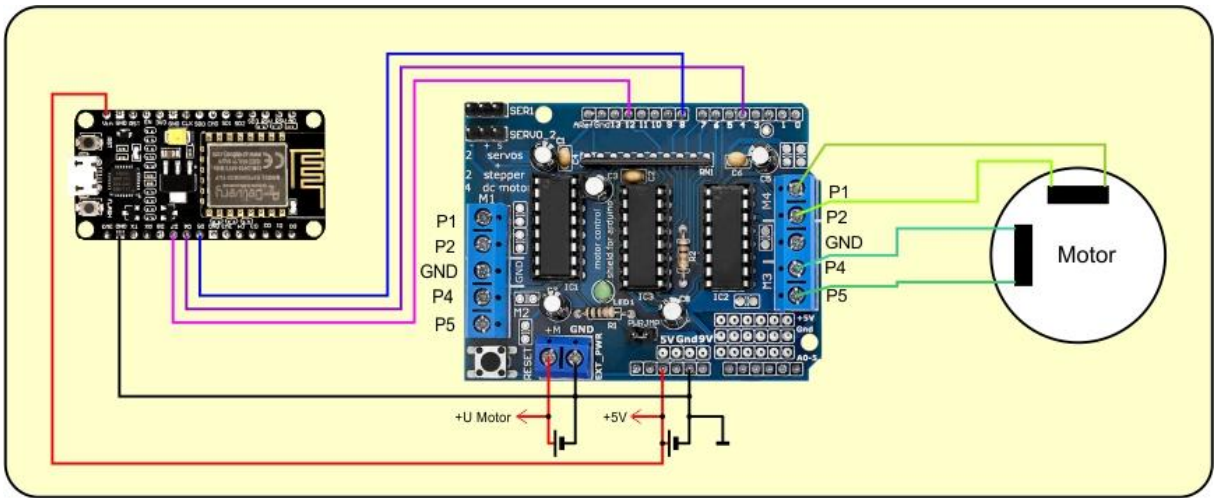

Abbildung 17: Steppermotor - Schaltung

Nach der Besprechung der Hardwaregrundlagen wenden wir uns jetzt der Programmierung zu.

Die Software

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft
packetsender zum Testen des ESP8266 als UDP-Client und -Server
SALEAE – Logic-Analyzer-Software (64 Bit) für Windows 8, 10, 11

Verwendete Firmware für einen ESP32:

MicropythonFirmware
v1.19.1 (2022-06-18) .bin

Verwendete Firmware für einen ESP8266:

v1.19.1 (2022-06-18) .bin

Die MicroPython-Programme zum Projekt:

shieldtest.py Betriebsprogramm

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung (english version). Darin gibt es auch eine Beschreibung, wie die Micropython-Firmware (Stand 05.02.2022) auf den ESP-Chip gebrannt wird.

MicroPython ist eine Interpretersprache. Der Hauptunterschied zur Arduino-IDE, wo Sie stets und ausschließlich ganze Programme flashen, ist der, dass Sie die MicroPython-Firmware nur einmal zu Beginn auf den ESP32 flashen müssen, damit

der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang hier beschrieben.

Sobald die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm kompilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Makro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

Autostart

Soll das Programm autonom mit dem Einschalten des Controllers starten, kopieren Sie den Programmtext in eine neu angelegte Blankodatei. Speichern Sie diese Datei unter boot.py im Workspace ab und laden Sie sie zum ESP-Chip hoch. Beim nächsten Reset oder Einschalten startet das Programm automatisch.

Programme testen

Manuell werden Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5 gestartet. Das geht schneller als der Mausklick auf den Startbutton, oder über das Menü **Run**. Lediglich die im Programm verwendeten Module müssen sich im Flash des ESP32 befinden.

Zwischendurch doch mal wieder Arduino-IDE?

Sollten Sie den Controller später wieder zusammen mit der Arduino-IDE verwenden wollen, flashen Sie das Programm einfach in gewohnter Weise. Allerdings hat der ESP32/ESP8266 dann vergessen, dass er jemals MicroPython gesprochen hat. Umgekehrt kann jeder Espressif-Chip, der ein kompiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer so, wie hier beschrieben.

Motortestbetrieb

Am ESP8266 (Amica), den ich hier einsetze, sind nicht alle GPIOs gleichermaßen verwendbar. Die Anschlüsse 12 (LATCH), 8 (SER) und 4 (SRCLK) werden durch Pulldownwiderstände auf GND-Potenzial gezogen. Damit der ESP8266 richtig starten kann, dürfen diese Pins also nicht mit GPIOs verbunden werden, die beim Start HIGH-Potenzial brauchen. Um sicher zu gehen, kommen daher nur D5, D6 und D7 in Frage.

ESP8266: Motorshield
D5 = GPIO14: 4 SRCLK
D6 = GPIO12: 12 RCLK
D7 = GPIO13: 8 SER

```
#LUA-Pins      D0 D1 D2 D3 D4 D5 D6 D7 D8 RX TX
#ESP8266 Pins 16  5  4  0  2 14 12 13 15  3  1
#Achtung       hi sc sd hi hi          lo hi hi
```

Abbildung 18: Besonderheiten beim ESP8266

In Libre Office Calc habe ich mir die Bytewerte für die Motoransteuerung berechnet.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
25	Schrittmotoren – Schaltmuster (unipolar)															
26	Motor 2 (rechts)						Bit-Muster									
27	Leitung	3a	3b	4a	4b		7	6	5	4	3	2	1	0		Bytewert
28	Anschluss	P5	P4	P2	P1		4b	3b	4a					3a		
29	Wertigkeit															
30	Schritt															
31	0	1	0	1	0		0	0	1					1		33
32	1	1	0	0	0		0	0	0					1		1
33	2	1	0	0	1		1	0	0					1		129
34	3	0	0	0	1		1	0	0					0		128
35	4	0	1	0	1		1	1	0					0		192
36	5	0	1	0	0		0	1	0					0		64
37	6	0	1	1	0		0	1	1					0		96
38	7	0	0	1	0		0	0	1					0		32
39																
40	Motor 1 (links)						Bit-Muster			2b	1b	1a	2a			
41	Schritt	1a	1b	2a	2b		7	6	5	4	3	2	1	0		Bytewert
42	0	1	0	1	0					0	0	1	1			6
43	1	1	0	0	0					0	0	1	0			4
44	2	1	0	0	1					1	0	1	0			20
45	3	0	0	0	1					1	0	0	0			16
46	4	0	1	0	1					1	1	0	0			24
47	5	0	1	0	0					0	1	0	0			8
48	6	0	1	1	0					0	1	0	1			10
49	7	0	0	1	0					0	0	0	1			2

Abbildung 19: Ansteuerung der Treiberstufen unipolar

Man unterscheidet beim Schrittmotor drei Betriebsmodi: Vollschritt, Halbschritt und Microschritt. Im Vollschrittmodus ist jeweils genau eine Spule aktiviert. Das entspricht den weißen Zeilen in obiger Tabelle.

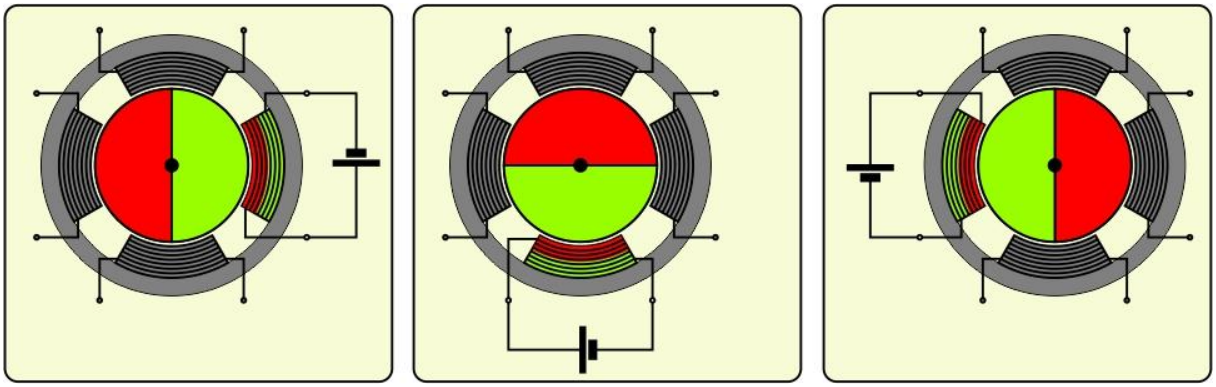

Abbildung 20: Vollschrittbetrieb

Im Halbschrittbetrieb wird die darauffolgende Spule als Zwischenstufe auch mit eingeschaltet. Das passiert in den rosa Zeilen.

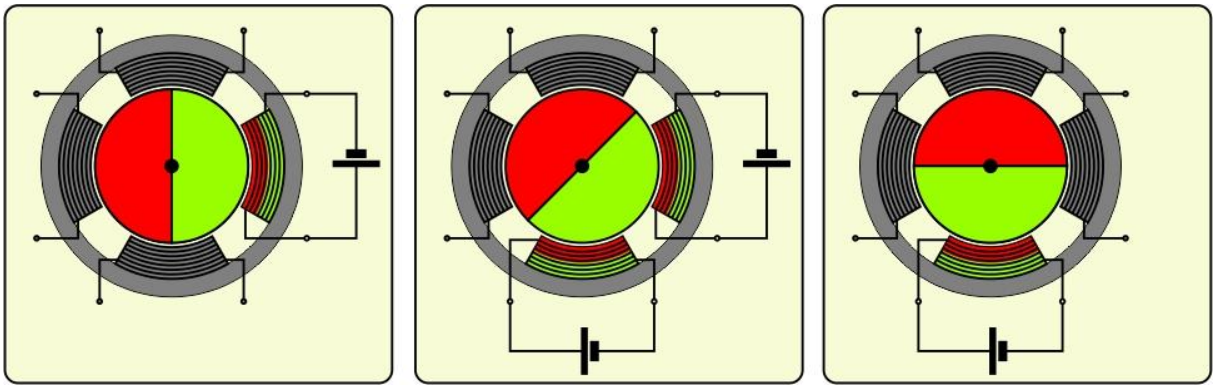

Abbildung 21: Halbschrittbetrieb

Die Abbildungen 19 und 20 zeigen das für einen unipolaren Motor. Im Zeitdiagramm sieht das dann so aus. Die Polung der Spulen ist stets gleich, die Aktvierung zeitversetzt.

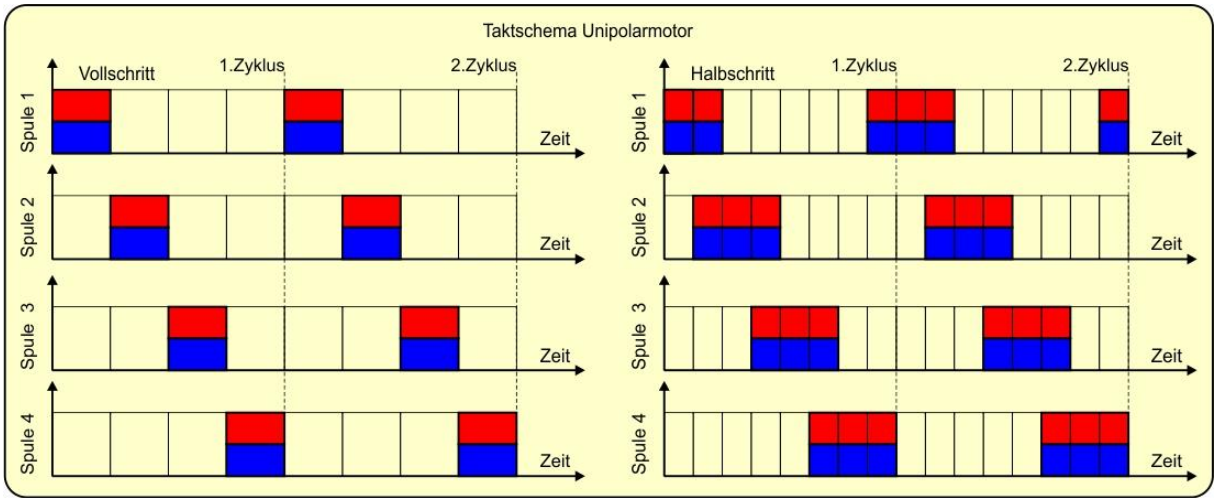

Abbildung 22: Spulenaktivieriung beim Unipolarmotor

Bei dem verwendeten Bipolarmotor werden die Spulen umgepolt.

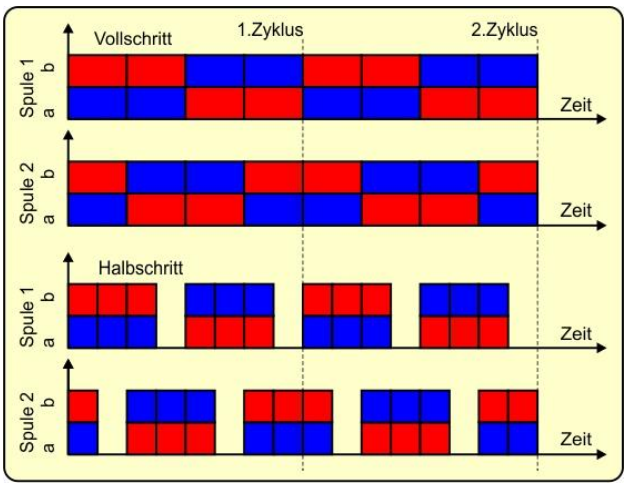

Abbildung 23: Polung der Spulen

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Schrittmotoren – Schaltmuster (bipolar)						Bit-Muster									
2	Motor 2 (rechts)						4b	3b	4a					3a		
3	Schritt	3a	3b	4a	4b		7	6	5	4	3	2	1	0		Bytewert
4	0	1	0	1	0		0	0	1					1		33
5	1	1	0	0	0		0	0	0					1		1
6	2	1	0	0	1		1	0	0					1		129
7	3	0	0	0	1		1	0	0					0		128
8	4	0	1	0	1		1	1	0					0		192
9	5	0	1	0	0		0	1	0					0		64
10	6	0	1	1	0		0	1	1					0		96
11	7	0	0	1	0		0	0	1					0		32
12																
13	Motor 1 (links)						Bit-Muster			2b	1b	1a	2a			
14	Schritt	1a	1b	2a	2b		7	6	5	4	3	2	1	0		Bytewert
15	0	1	0	1	0					0	0	1	1			6
16	1	1	0	0	0					0	0	1	0			4
17	2	1	0	0	1					1	0	1	0			20
18	3	0	0	0	1					1	0	0	0			16
19	4	0	1	0	1					1	1	0	0			24
20	5	0	1	0	0					0	1	0	0			8
21	6	0	1	1	0					0	1	0	1			10
22	7	0	0	1	0					0	0	0	1			2

Abbildung 24: Ansteuerung der Treiberstufen bipolar

Zur Berechnung des Bytewerts setze ich die Werte für die Halbbrücken auf 0 oder 1 und übertrage sie in den Bereich Bitmuster.
G31: =E31 …
Der Bytewert ergibt sich dann durch die Formel:
P31: =N31+32*I31+64*H31+128*G31…
Analog läuft es für den linken Motoranschluss.

Diese Werte brauche ich jetzt gleich in meinem Testprogramm. Das Rechenblatt können Sie herunterladen. Klicken Sie dazu einfach auf die Abbildungen.

Das Testprogramm

Die Importliste ist nicht lang. Wir müssen GPIO-Pins ansteuern, brauchen gelegentlich kurze Schlafpausen und sollten eine Möglichkeit für ein sauberes Beenden des Programms einplanen.

```
from machine import Pin
from time import sleep, sleep_us
from sys import exit
```

Die GPIOs für die Leitungen zum SN74HC595 legen wir auf die bereits besprochenen Pins fest.

```
serOut=Pin(14,Pin.OUT,value=0) # D5 gn
sClk=Pin(12,Pin.OUT,value=0)  # D6 or
sLatch=Pin(13,Pin.OUT,value=0)# D7 ws
```

Aus der Libre Office Tabelle übertragen wir die Bytewerte in Form einer Liste.

```
MotorX=[6,
        4,
        20,
        16,
        24,
        8,
        10,
        2,
        ]

# Minebea bipolar:
# ge, rt, or, bl
MotorY=[33,
        1,
        129,
        128,
        192,
        64,
        96,
        32,
        ]
```

Für die Ausgabe der Pulse an SRCLK deklarieren wir die Funktion **pulse**(). Der Parameter nimmt die Pulsdauer in Microsekunden. Der Defaultwert 2 gilt, wenn beim Aufruf keine Angabe erfolgt.

```
def pulse(delay=2):
    sClk.on()
    sleep_us(delay)
    sClk.off()
```

Ähnlich funktioniert die Ausgabe des Pulses zur Übernahme der Bits aus dem Schieberegister in die Ausgangs-RS-Flipflops.

```
def latch(delay=2):
    sLatch.on()
    sleep_us(delay)
    sLatch.off()
```

Diese beiden Routinen benutzt die Funktion **shiftOut**(), um die acht Bits zu übertragen, die wir im Parameter **byte** übergeben müssen. Mit 0x80 = 0b10000000 setzen wir die Maske auf das MSB (Most Significant Bit = Bit 7), das als erstes übertragen wird. Die for-Schleife läuft von i=0 bis i=7.

```
def shiftOut(byte):
    """ MSB first """
    mask=0x80
    for i in range(8):
        bit=(byte & mask) >> (7-i)
        serOut.value(bit)
        pulse()
        mask=mask >> 1
    latch()
```

Wir maskieren das entsprechende Bit von **byte** durch Undieren und schieben es an die Position des LSB, das jetzt 0 oder 1 ist. Den Datenausgang an GPIO **serOut** setzen wir auf diesen Wert und geben den Schiebepuls an **sClk** aus. Danach schieben wir das Maskenbit um eine Position nach rechts für den nächsten Durchgang. Nachdem alle Bits im SN74HC595 angekommen sind, setzen wir den Puls zur Übernahme in die RS-Flip-Flops ab.

Die Funktion **schritt**() führt einen Schritt auf dem in **axis** übergebenen Motor aus. Die globalen Variablen **px** und **py** müssen in der Funktion als **global** erklärt werden, weil ihr Wert geändert wird, und für den nächsten Aufruf wieder zur Verfügung stehen muss. Wären sie nicht als global deklariert, würde MicroPython sie als lokal betrachten und nach dem Beenden der Funktion einstampfen.

```
def schritt(axis):
    global px,py
    if richtungX==0 and richtungY==0:
        return
    if "x" in axis:
        px = (px + richtungX*sm) % 8
    if "y" in axis:
        py = (py + richtungY*sm) % 8
    byte=MotorX[px] | MotorY[py]
    print(richtungY, sm, "{:08b}".format(byte & 0xE1))
    shiftOut(byte)
```

In **richtungX** und **richtungY** wird die Drehrichtung geführt, 1 im Uhrzeigersinn, -1 im Gegenuhrzeigersinn und 0 für Stillstand. Im letzteren Fall gibt es nichts zu tun, daher blasen wir zum sofortigen Rückzug.

Für die entsprechende Achse bestimmen wir ansonsten die neue Schrittposition. **sm** enthält die Information für den Schrittmodus, 1 für Halbschritt und 2 für Vollschritt. Diesen Wert multiplizieren wir mit dem Richtungswert und addieren das Ergebnis zur bisherigen Position. Damit wir im 8-er-Ring bleiben, ermitteln wir schließlich den Teilungsrest modulo 8.

Beispiel:
sm = 2
richtungY=1
py = 4
neuer Wert: py = 6, weil 4 + 1 * 2 = 6 und 6 / 8 ist 0 Rest 6
nächster Wert: py=0, weil 6 + 1 * 2 = 8 und 8 / 8 ist 1 Rest 0

Dann setzen wir durch Oderieren den Bytewert aus den Listen **MotorX** und **MotorY** zusammen, indem wird als Zeiger in die Listen **px** und **py** benutzen. Das Byte lassen wir uns zur Kontrolle in REPL im Binärformat ausgeben und senden es dann an den SN74HC595.

Bevor es in die Hauptschleife geht, deklarieren wir die Variablen und initialisieren sie mit den Startwerten.

```
px=0
py=0
richtungX = 1
richtungY = 1
sm=1
pause=1000
```

```
while 1:
```

Wir starten mit dem Abfragen einer Taste. Ein "q" und Enter brechen das Programm ab, nachdem alle Eingänge der beiden L293D und der Richtungswert für die y-Achse auf 0 gesetzt wurden.

```
    t=input("Taste >")
    if t=="q": # Quit
        shiftOut(0)
        richtungy=0
        exit()
```

Ein "l" oder "r" setzen das Richtungsflag auf Links- oder Rechtslauf.

```
    elif t=="l": # Linkslauf (counter clockwise)
        richtungY=-1
    elif t=="r": # Rechtslauf (clockweise)
        richtungY=1
```

Mit einem "w" leiten wir die Abfrage einer Schrittanzahl ein. Das Ergebnis der input-Anweisung ist ein String, den wir in eine Ganzzahl umwandeln von der wir einen Schritt abziehen. Bei jedem Schleifendurchgang wird nämlich am Ende stets ein Schritt ausgeführt.

```
    elif t=="w": # Schrittzahleingabe
        w=int(input("Schritte > ")) - 1
        for i in range(w):
            schritt("y")
            sleep_us(pause)
```

Ein "v" stellt auf Vollschrittmodus ein. Der Zeiger in die Bitmustertabelle wird um jeweils 2 erhöht, und wir müssen, weil nur jeweils eine Spule eingeschaltet wird mit einer ungeraden Position starten (siehe Tabellen in Abbildung 19 und 24). Wenn der Teilungsrest von **py** durch 2 gleich 0 ist, liegt ein gerader Wert vor, und es ist eine 1 zu addieren, um auf den nächsten ungeraden Wert zu kommen. Damit wir im Achter-Ring bleiben, bilden wir wieder den Teilungsrest modulo 8 wie oben.

```
    elif t=="v": # Vollschritt
        sm=2
        if py % 2 == 0:
            py=(py+1) % 8
```

Für den Halbschrittmodus müssen wir nur **sm** mit 1 belegen.

```
    elif t=="h": # Halbschritt
        sm=1
```

Um die bei "w" eingegebene Schrittzahl zu wiederholen, offerieren wir dem Programm ein "x". Die Schritte werden durch die for-Schleife wiederholt, wie oben.

```
    elif t=="x": # Schrittzahl wiederholen
        for i in range(w):
            schritt("y")
            sleep_us(pause)
```

Die Geschwindigkeit kann mit "p" eingestellt werden. Je länger die Pause zwischen zwei Schritten in Microsekunden dauert, desto langsamer läuft der Motor. So lässt sich auch der Wert herausfinden, bei dem gerade noch Schritte sicher ausgeführt werden können. Darauf hat auch die Belastung der Motorwelle Einfluss.

```
    elif t=="p":
        pause=int(input("Schritte > "))
```

Andere Eingaben werden einfach übergangen, die schicken wir ins Nirwana.

```
    else:
        pass
```

Wird die Eingabe nur mit Enter abgeschickt, dann führt das Programm hier immer einen Einzelschritt aus. Deshalb habe wir bei der Eingabe der Schrittzahl 1 abgezogen.

```
# Einzelschritt
    schritt("y")
```

Ich hoffe, Ihr Motor läuft jetzt wunschgemäß. Für die Demonstration der Funktionsweise ist das Arduino-Motor-Shield gut geeignet, weil sich daran die Vorgehensweise bei der Schrittmotoransteuerung schön darlegen lässt. Ein Handicap bei der Aufbereitung für diesen Beitrag war die schlechte Bildqualität der Dokumentation zum Shield. So musste ich mir die Verbindungen einzeln durch Messungen und Intuition heraussuchen. Zusammen mit den Datenblättern des SN74HC595 und L293D hat es dann schon geklappt.

Die Ansteuerung des Motor-Shields zwickt nun leider recht viel von der Controllerleistung weg, weil jeder Motorschritt alleine für die Übertragung ca. 1,25Millisekunden dauert. Ein Großteil der Rechenzeit fällt daher nur dafür an. Ganz zu schweigen davon, dass Microstepping noch viel mehr Aufwand für den Controller bedeuten würde.

Eine Lösung bietet sich durch zwei andere, im Vergleich zum Motor-Shield, sehr kleine Module an, die das Motortiming und die Schrittverwaltung komplett selbst übernehmen und zudem auch noch Stromstärken bis 2,5 A verkraften können. Wir sagen dann nur noch mit einem Logikpegel in welche Richtung der Motor drehen soll, und geben durch einen Puls den Schritt in Auftrag. Außerdem ist mit den Modulen Microstepping bis 1/16 beziehungsweise 1/32 Schritt möglich. Das ergibt bei einem Motor mit 200 Schritten pro Umdrehung einen Microschrittwinkel von 0,056 Grad. In der nächsten Folge werden wir uns diese Prachtmodule genau anschauen und ein MicroPython-Modul dafür entwickeln.

Bis dann, bleiben Sie dran!