

ChatGPT als Bastelhelfer

- ▶ Unterstützt beim Programmieren
- ▶ Hilft Schaltungen zu entwerfen
- ▶ Erklärt Code-Schnipsel
- ▶ Generiert Code für Arduino und ESP



Projekte

- ▶ Heimcomputer mit Lisp
- ▶ Energieanzeige

Workshops

- ▶ Flirten mit ESP-NOW
- ▶ Kinematik von Laufmaschinen

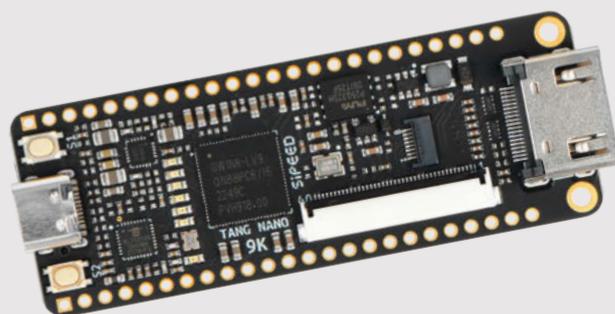
OLED-Tastatur

- ▶ Mini-Displays in Tasten einbauen
- ▶ Eigene Symbole anzeigen lassen
- ▶ GPIOs multiplexen



FPGA-Einstieg

- ▶ Loslegen mit günstigen Boards
- ▶ Aufbau und Funktion verstehen
- ▶ Lauflicht programmieren



7/24

29.11.2024

CH CHF 26,50

AT 14,90

Benelux 15,90

€ 13,50



**DIE KONFERENZ FÜR
SOFTWARE-ARCHITEKTUR**

**BRIDGING
THE GAP**

3.-7. FEBRUAR 2025

ICM MÜNCHEN

**DER WISSENS-HUB
FÜR SOFTWAREENTWICKLUNG**

**Bis 20.12.24
Preisvorteile sichern**



Jedem Anfang wohnt ein Zauber inne!

Sicher kennen Sie das Gefühl, wenn man eine neue Hardware oder Software erstmals ausprobiert – oder ein ganz neues Themengebiet erkundet. Die Neugierde, die freudige Erwartung, die feuchten Hände, das Erfolgserlebnis, wenn die LED zum ersten Mal blinkt, das erste 3D-gedruckte Modell genau passt oder die Fräse das tut, das man glaubte, ihr vorgegeben zu haben.

In diesem Heft haben wir gleich zwei Themen für Sie, in die Sie vergleichsweise schnell und günstig einsteigen und die Sie mit Erfolgserlebnissen absolvieren können: Programmieren mit ChatGPT und FPGAs. Lange Zeit lieferte ChatGPT beim Thema Programmieren eher unbrauchbare Ergebnisse und verlor sich mit seinen Erklärungen zu technischen Themen im Vagen. Deshalb nutzte die Make-Redaktion ChatGPT nur selten und auch unsere Autoren schienen KI zu meiden.

Die aktuellen KI-Modelle sind mittlerweile jedoch so gereift, dass sie nicht nur fehlerfreie Programme ausspucken können und auch noch ziemlich gut deren Funktion erklären, sondern den Maker von der Idee eines Projekts bis zum ersten Prototypen unterstützen, inklusive Vorschläge für Bauelemente. Das Helferlein von Daniel Düsentrieb scheint wahr geworden. Unser Autor Benno Lottenbach und Kollege Johannes Börnsen zeigen auf den Seiten 8 und 28, wie Helferlein Sie unterstützen kann.

FPGAs galten lange als zu teuer und komplex. Die im Vergleich zu Mikrocontrollern früher ziemlich teuren Boards sind dank chinesischer Hersteller nun schon ab 10 Euro zu haben. Das senkt die Schwelle für neugierige Maker ungemein, die ohne finanzielle Wagnisse vielleicht nur mal in die faszinierende Technik reinschnuppern wollen. Unser Primer von Michael Linsenmeier auf Seite 16 führt in die Funktion von FPGAs ein und zeigt am Beispiel eines LED-Lauflichts, mit welcher Software man sie konfiguriert. Fun Fact: Die Boards kommen in der Regel mit vorinstallierter RISC-V-CPU.

Damit komme ich zu meinem derzeitigen Lieblingsthema: RISC-V. Espressif verbaut ja schon länger die freie Architektur in seinen SoCs und dieses Jahr ist auch die Raspberry Pi Foundation mit den zwei RISC-V-Kernen im RP2350 auf dem Pico 2 nachgezogen. Im Hintergrund rollt gerade der chinesische Hersteller WCH mit seinen günstigen CH32-RISC-V-SoCs den Embedded-Markt auf und die Community stellt praktische Tools und Bibliotheken bereit.

Zeit also, sich in 2025 näher mit diesem vielversprechenden Ökosystem zu beschäftigen und in unbe-

kannte Gebiete vorzuwagen. Damit es für Sie auch im neuen Jahr spannend, abwechslungsreich und gefährlich bleibt. Mit Deutschlands gefährlichstem DIY-Magazin.

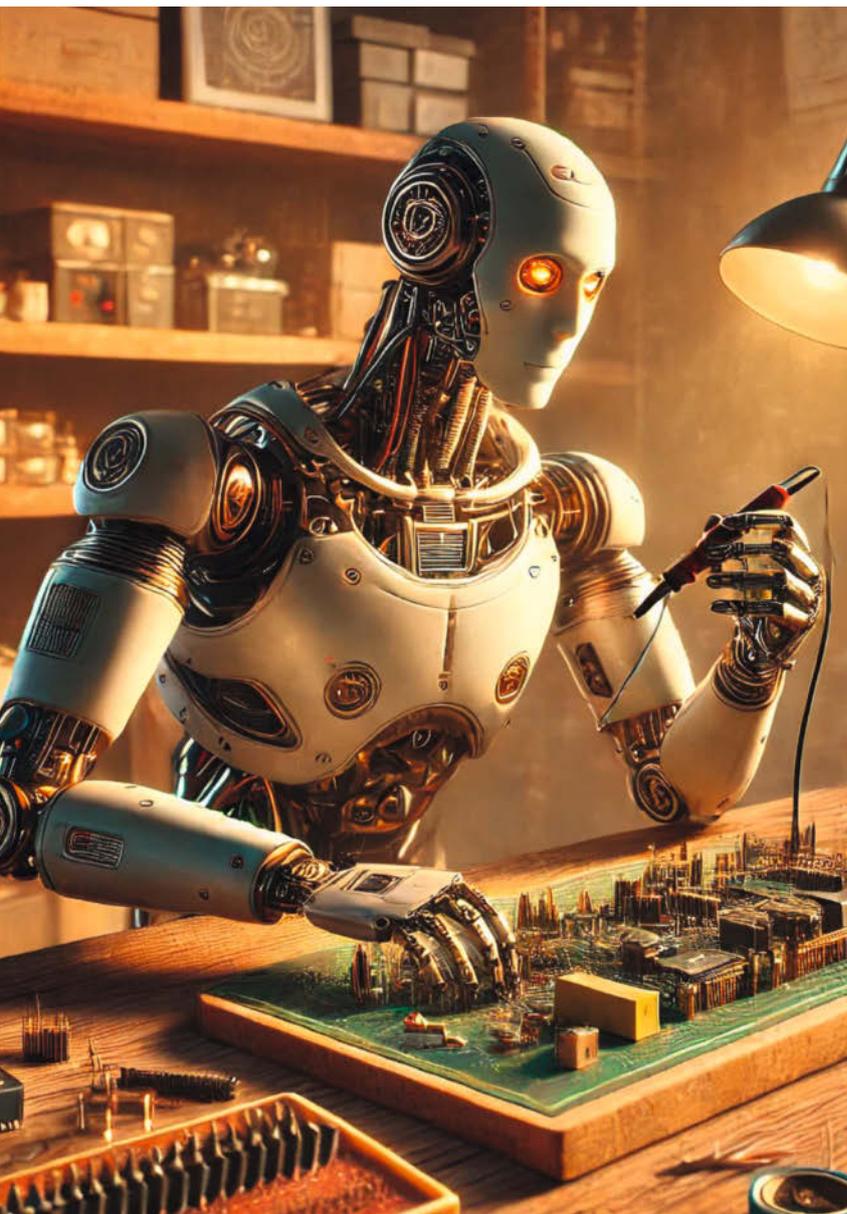
Guten Rutsch & happy Hacking

Daniel Bachfeld

Daniel Bachfeld

► make-magazin.de/x3vq





Wie KI beim Basteln helfen kann

Manchmal mangelt es uns Makern an Inspiration oder Lösungen für knifflige Probleme. Doch jetzt steht kein Maker mehr allein da. Wir zeigen in zwei Artikeln, wie ChatGPT & Co. beim Basteln helfen. Im ersten Artikel schauen wir uns an, wie KI aus einer Idee schnell ein ausgereiftes Konzept mit Schritt-für-Schritt-Anleitungen entwickelt und dabei immer freundlich und konstruktiv bleibt. Der zweite erklärt, wie uns KI beim Programmieren unterstützen kann. Zum Beispiel, um Code zu verstehen und zu modifizieren – besonders einfach durch den neuen Canvas-Modus in ChatGPT-o1.

- 8** ChatGPT als Bastelkollege
- 28** Programmieren lernen mit ChatGPT-o1

Inhalt

Endlos viel Kreativität

Wir bauen ESP-NOW-gesteuerte Flirt-Amulette, die über verschiedene Blinkmuster miteinander kommunizieren können. Dann zeigen wir, wie man einen autarken Lisp-Minicomputer mit Retro-Feeling anfertigt. Um die Energiewende zu unterstützen, geht es in einem weiteren Projekt um ein smartes Display, das anzeigt, wie viel Strom man im Haushalt verbraucht und erzeugt. Außerdem tauchen wir in die Funktionsweise von kinematischen Laufmaschinen ein.

- 34** Flirt-Amulette mit ESP-NOW
- 42** Smarte Energie-Infotafel
- 58** Laufmaschinen konstruieren
- 74** Die Lisp-Box



- 3** Editorial
- 6** Leserforum
- 8** Workshop: ChatGPT als Bastelkollege
- 16** Know-how: FPGA für Maker
- 28** Workshop: Programmieren lernen mit ChatGPT-o1
- 34** Workshop: Flirt-Amulette mit ESP-NOW
- 42** Projekt: Smarte Energie-Infotafel
- 50** Projekt: Weihnachtsstern mit Oxocard
- 56** Unser Team
- 58** Workshop: Laufmaschinen konstruieren
- 65** Make: Online
- 66** Community-Projekt: Poolüberwachung mit Arduino
- 68** Community-Projekt: 80er-Jahre-Spielekonsole im FPGA

Fernsehen über Raspi gucken

Wer noch über einen herkömmlichen Fernsehanschluss verfügt, kann mit einem Raspberry Pi 3 oder höher und der Software Tvheadend kostenlos lineares Fernsehen ins eigene Netzwerk streamen. So kann man problemlos seine Lieblingsserie im Privatfernsehen aufnehmen und auch in Räumen schauen, in denen es keinen Fernsehanschluss gibt. Im Artikel wird auch gezeigt, wie sogar Satellitenempfang möglich ist.

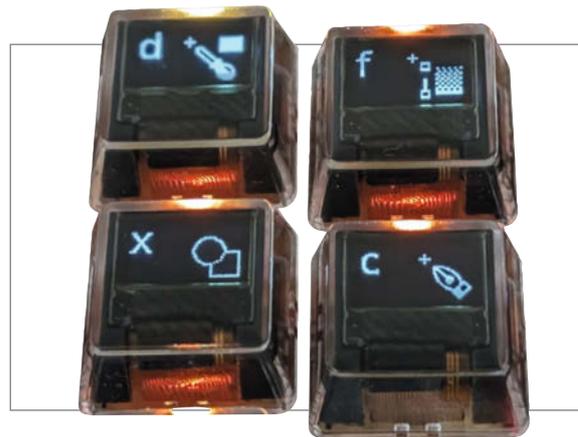
104 Fernsehen mit dem Pi streamen



Tastatur mit Mini-Displays

Die Fülle an Tastenkürzeln für verschiedene Programme kann sich kaum jemand merken. Wäre es daher nicht praktischer, wenn die Tastatur je nach Programm einfach anzeigen könnte, welche Kürzel verfügbar sind? Genau das ermöglicht das Open-Source-Tastatur-Projekt PolyKybd, bei dem jede Taste mit einem eigenen programmierbaren OLED-Display ausgestattet wird. Diese können sogar kleine Piktogramme, wie Emojis, darstellen.

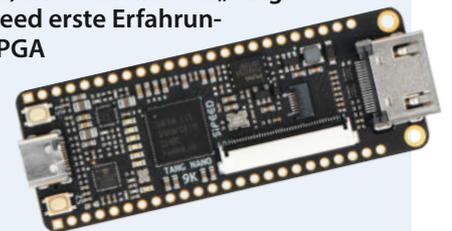
110 Tastatur mit OLED-Displays



- 70 Community-Projekt: Motor aus dem 3D-Drucker
- 72 Reingeschaut: Blinkgeber
- 74 Projekt: Die Lisp-Box
- 82 Know-how: Mehr Hirn für Schimpfolino
- 90 Workshop: Alles wird USB-C
- 94 Projekt: Die String-Art-Maschine
- 100 Projekt: Der Kabelbinder-Aufmacher
- 103 Maker-Faire-Update
- 104 **Projekt: Fernsehen mit dem Pi streamen**
- 110 **Projekt: Tastatur mit OLED-Displays**
- 118 Kurzvorstellungen
- 122 Impressum / Nachgefragt

FPGA meistern

Manche Maker mögen FPGA für zu teuer und zu kompliziert halten. Aber jetzt gibt es die programmierbaren Schaltkreise in neuen Ausführungen, die sowohl preiswert als auch leistungsstark sind. In unserem Einführungsartikel zeigen wir, wie man mit den „Tang Nano“-Boards von Sipeed erste Erfahrungen im Umgang mit FPGA machen kann und welche Vorteile die Technik gegenüber klassischen Mikroprozessoren bietet.



16 FPGA für Maker

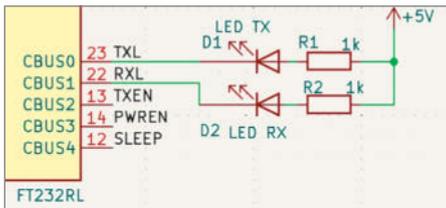
Themen von der Titelseite sind rot gesetzt.

Leserforum

Fehler bei LEDs

Ausgediente Smartphones als Arduino-Ersatz, Make 6/24, S. 94

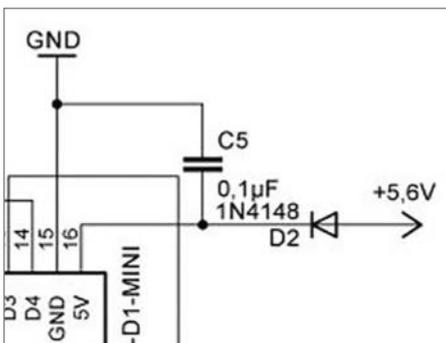
Im Schaltplan auf S. 96 sind die LEDs für RX und TX verkehrt herum eingezeichnet. Wir haben den korrigierten Schaltplan auf GitHub hochgeladen.



Fehlerhafte Laufwerksteuerung

Handy als Steuerung für Bandmaschine, Make 6/24, S. 24

Im Schaltplan auf S. 27 ist die Diode D2 falsch an den Wemos angeschlossen. Der angepasste Schaltplan liegt im Repository des Projekts auf GitHub.



Infocom-Spiele

Apple II – Die Nostalgie-Zeitreise, Make 5/24, S. 98

Auch wenn der Artikel zum Apple II schon im vorletzten Heft war: Das im Artikel erwähnte Hitch-Hikers-Guide-Spiel ist auch heute noch – ohne Apple Emulator – spielbar. Wenn ich unter Debian Linux `apt-cache search infocom` aufrufe, bekomme ich eine ganze Liste von Z-Machine-Game-Interpretern. Das Game-Format von Infocom war nämlich eine Kombination aus Game-Interpreter und

dem eigentlichen Game-File. Letztere kann man auch heute noch (zumindest unter Linux) spielen. Sogar neue Spiele wurden in diesem Format entwickelt und es gibt einen Game-Compiler dafür. Am HHGTTG-Spiel hat Douglas Adams noch persönlich mitgewirkt, Disk-Images mit dem Game-File findet man z. B. noch auf archive.org. Der spezielle Humor des Spieles ist wohl auch heute noch interessant, auch wenn es ein reines Text-Adventure ist.

Zu Eurem Motto, das gefährlichste Magazin zu sein: Wie wäre es, wenn Ihr zur Abwechslung mal auch gefährlich nicht nur für Eure Leser seid? Eure amerikanische Schwesternzeitschrift hatte vor ein paar Monaten mal ein ganzes Heft zum „War on Repair“. Da wäre die europäische Perspektive sehr interessant. Ah und vielen Dank für das Heft zur Weiterverwendung von Mobiltelefonen!

Ralf Schlatterbeck

Alternatives Display

Preisrahmen für Tibber, Make 4/24, S. 8

Ich habe die Anzeige für Strompreise (Tibber) auf das Lilygo T5 4.7"-EPD überführt. Hierzu war eine recht umfangreiche Änderung des Codes notwendig, da das T5 auf einem ESP32 basiert und das EPD andere Befehle erfordert. Diese Lösung ist leichter umzusetzen, da man lediglich das Lilygo T5 kauft, sich ein Case druckt und die Firmware flasht. Ich könnte mir vorstellen, dass dies für Einsteiger (hier zähle ich mich selbst dazu) attraktiver ist. Der Code findet sich hier https://github.com/thebluntblade/Tibberprice_E-Ink-Display.

Andre Krämer

Geht doch!

Funktechniken fürs Smart Home, Make 5/24, S. 117

Im Heft 5/2024 schreiben Sie im Artikel „Funktechniken fürs Smart Home“, dass bei EnOcean keine Selbstbauprojekte möglich wären. Dem muss ich widersprechen, ich habe mehrere selbstgebaute Komponenten in meinem Smart Home (FHEM auf Raspi) im Einsatz. Von EnOcean gibt es zwei Module zum Auflöten: STM300 und TCM300. Am einfachsten funktioniert es mit dem TCM300: Das Modul hat eine RS232-Schnittstelle (3,3V Pegel), an die ein Mi-



krocontroller angeschlossen werden kann. Über die Schnittstelle kann der Mikrocontroller dann EnOcean-Funktelegramme versenden.

Etwas komplizierter ist es mit dem STM300. Darin ist ein 8051-Derivat enthalten, man kann die Standardsoftware ändern (z. B. für eine Sensorabfrage), was ich für den Fenstergriffsensor und den Wassermelder getan habe. Allerdings braucht man dafür den 8051-C-Compiler von Keil (ich habe eine auf 2K Programmcode beschränkte Version für Privatanwender benutzt). Sonst ist der Compiler für privat nicht bezahlbar. Anbei ein Foto eines selbstgebautes Geräts mit EnOcean-Modulen.

Uwe Magnus

Total enttäuscht

Siebdruck mit Laser und Leim, Make 5/24, S. 83

Ich bin total enttäuscht. Heute kam die neue Make. Was lese ich da auf dem Umschlag: „Siebdruck mit Laser“. Super, denke ich mir. Das ist mal was für meine Tochter, die in Potsdam Kommunikationsdesign studiert und auch schon mal was mit Siebdruck gemacht hat. Um so größer ist die Enttäuschung, als ich den Artikel aufschlage. Das Wichtigste habt Ihr weggelassen. Wie Siebdruck geht, lernt manch einer ja schon im Kunstunterricht in der Schule. Ansonsten gibts dazu Anleitungen im Netz wie Sand am Meer. Aber der eigentlich neue interessante Teil, wie das mit dem Laser funktioniert, fehlt weitgehend.

Könnt Ihr Euch nicht vorstellen, dass es Menschen gibt, die so ein Ding noch nie benutzt haben? Und die, weil sie das mit dem

Siebdruck eigentlich toll fänden und gerne machen würden, diese Gelegenheit nutzen würden, so einen Laser mal einzusetzen? Dazu fehlt aber jegliche Info in dem Artikel. Man könnte fast meinen, das hat irgendjemand, der scheinbar die Überschrift nicht gelesen hat, rausgekürzt, weil der Artikel zu lang war. Schade, sehr schade. Ich würde sagen Thema verfehlt, Note 6, setzen!

R. Nase

Hier hat uns leider der sogenannte „Wissens-Fluch“ erwischt: Durch die regelmäßig wiederkehrenden Artikel rund um Lasercutter nahmen wir an, die Angaben zu Leistung und Geschwindigkeit des Lasers wären zum Nachvollziehen für unsere Leser vorhanden. Wir achten künftig darauf, weitere Hinweise in die Artikel einzubauen.

Kein Treffer

Wassertiefe messen und digital abfragen, Make 5/24, S. 53

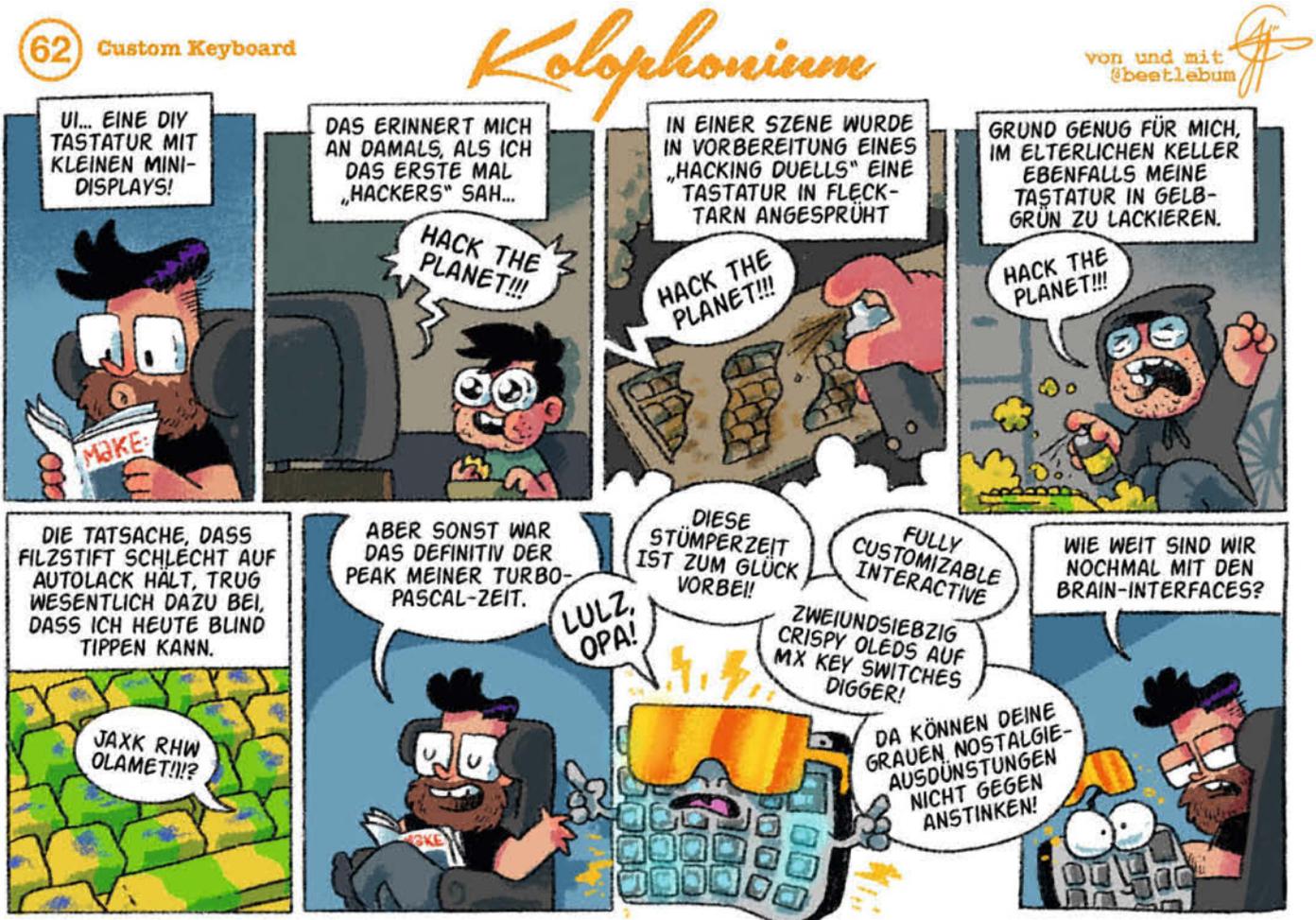
Ich bin jetzt seit einer Weile Abonnent bei euch, und natürlich ist nicht jedes Heft von vorne bis hinten relevant für mich. Trotzdem bin ich immer mal wieder froh, wenn ich auch in Artikeln, die ich eher „der Vollständigkeit halber“ lese, über Informationen und Ideen und Inspiration stolpere. Von daher bin ich mental immer auf eine Mischung aus „super für mich“ und „nicht jeder Schuss ein Treffer“ eingestellt. Bei der Ausgabe 5/2024 bin ich aber an ein paar Sachen hängengeblieben.

Exemplarisch der Artikel „Wassertiefe messen und digital abfragen“. Dafür, dass der Sensor quasi das Einzige ist, was den Artikel merklich von anderen „Messwerte via ESP auffangen, via Funk nach MQTT schieben, in HA

verwenden“-Artikeln unterscheidet, kommt er leider nur sehr „am Rande“ vor. Nicht mal in der dedizierten Sensor-Box wird das Modell des konkreten Beispiel-Sensors genannt. Ähnlich komisch finde ich dann auch die Aufteilung der Bilder: Die Platine bekomme ich als Leser von 4 Richtungen zu sehen, den Sensor nur in seiner Endposition am Boden des Teiches. Der Artikel ist sonst angenehm detailliert.

Um es nicht nur bei der Kritik zu belassen: Ich fand z.B. das Romero-Interview samt Artikel wirklich klasse, und was Hardware-Projekte angeht, war der Artikel über den ESP-Verkaufsautomaten eine Super-Sache. Speziell bei sonst nicht so oft genutzter Hardware wie z.B. dem Münzprüfer waren Detailgrad und Bebilderung im Artikel wirklich hilfreich. Freu mich aufs nächste Heft!

Florian Geldner





Prompt: Benno Lottenbach / KI Leonardo.ai

ChatGPT als Bastelkollege

Den wenigsten Makern dürfte der Umgang mit Elektronik in die Wiege gelegt worden sein. Und so treiben der Aufbau einer Schaltung und das Programmieren eines Mikrocontrollers so manchem Bastler die Schweißperlen auf die Stirn. Wem die Routine fehlt, der kann heutzutage auf die Hilfe von künstlicher Intelligenz zählen und so rasant zu verblüffend guten Ergebnissen gelangen. Unser Autor hat ausprobiert, wie sich ChatGPT in einem klassischen Maker-Projekt nutzen lässt.

von Benno Lottenbach

Kürzlich ist es wieder passiert, beim Schlendern durch den Wald, sogar noch vor der genüsslichen Sandwich-Pause auf der Parkbank. Ein Blitzgedanke fügte sich dem anderen, und plötzlich war die Idee für ein neues Maker-Projekt da: Meine beiden Kinder sollten ihre aktuelle Stimmungslage mithilfe eines „Gemütszustandsbarometers“ visualisieren können. Ein Pfeil, von einem Servomotor angetrieben, würde die aktuelle Stimmungslage in drei Stufen auf einer Skala anzeigen. Der Pfeil ließe sich dabei mithilfe von zwei Buttons auf der Skala verschieben. Um das ganze optisch noch etwas aufzupicken, sollten zusätzliche drei LEDs die Position des Pfeils markieren.

Von plötzlicher Schaffenslust getrieben, begab ich mich eiligen Schrittes nach Hause, wo ich sofort mit der elektronischen Umsetzung des Vorhabens beginnen wollte.

Neue Entwicklungsmethoden

Zugegeben, das Vorhaben stellt kein außergewöhnlich komplexes Projekt dar. Aber ich bin nun mal kein routinierter Software-Entwickler und schon gar kein versierter Elektroniker. Es ist Jahre her, seit ich einen Arduino-Sketch selbst programmiert habe. Und beim Umgang mit LEDs, Widerständen und Motoren muss ich mich bei jedem Projekt wieder von Neuem einlesen. So wie mir dürfte es vielen anderen Makern auch gehen. Die Grundlagen sind zwar da, aber leider werden sie auch immer wieder schnell vergessen. Und so rechnete ich damit, dass der elektronische Aufbau und die Programmierung des Controllers gut und gern einen Tag, vielleicht auch länger beanspruchen würden.

Also habe ich mich diesmal für einen ganz neuen Ansatz entschieden, um dieses Projekt zu realisieren. ChatGPT und KI befinden sich seit Monaten in aller Munde, wohingegen es mein Sandwich bedauerlicherweise bisher nicht in den meinen geschafft hat. Im Rahmen eines Experiments sollte mir doch bitte diesmal künstliche Intelligenz (KI) bei der Umsetzung des Projekts helfen. Ob und wie sich ChatGPT als Bastelkollege für mein Maker-Projekt nützlich zeigte, erfahrt ihr in diesem Artikel.

Um eins vorwegzunehmen: So wie viele Maker-Projekte in gewissem Sinne einmalig sind, so sind auch die Antworten von ChatGPT kaum eins zu eins wiederholbar. Die KI gibt nämlich auf die gleiche Frage nicht immer die identische Antwort, wie ich feststellen musste. Falls ihr also das Gespräch durch Eingabe meiner Prompts rekonstruieren würdet, so würde sich zwar der Inhalt des Gesprächs gleichen, aber die Ausgestaltung der Antworten würde unterschiedlich ausfallen. Deswegen findet ihr auf GitHub (siehe Link in der Kurzinformatik) das Chat-Protokoll, mit welchem ihr

den Chat-Verlauf mit allen exakten Antworten nachlesen könnt.

Direkt los mit ChatGPT

Wie immer startete ich auch dieses Projekt mit einem Blick in die Bastelkiste. Die notwendigen elektronischen Komponenten waren schnell gefunden, und so begeben sich mit allem Kram an meinen Maker-Arbeitsplatz. Kaum auf der Website von ChatGPT eingeloggt, wartet der neue intelligente Kollege bereits auf meine Eingaben (siehe Kasten „Chatprotokoll“). So tippte ich ganz spontan, ohne mir Gedanken über geschliffene Formulierungen oder genau spezifizierte Anforderungen zu machen, den folgenden Prompt:

Prompt: Ich möchte ein kleines Robotikprojekt realisieren. Ich habe zwei Taster, mit welchen ich drei Positionen eines Servos steuern möchte. Die Position des Servos soll durch drei LEDs angezeigt werden. Hierzu habe ich einen Arduino Nano, einen Servo MG90s, verschiedene Widerstände und LED's, zwei Taster sowie Verbindungskabel zur Verfügung. Ausserdem steht mir ein Breadboard, die Arduino IDE und ein Schinken-Sandwich zur Verfügung. Kannst du mir bei diesem Projekt helfen?

Die darauffolgende Reaktion der KI ist beeindruckend. Nach einem kurzen Lob, dass ich ein spannendes Projekt vor mir habe, bietet mir ChatGPT sehr freundlich seine Hilfe an.

Kurzinformatik

- » KI für Maker-Projekte nutzen
- » Elektronische Schaltungen entwerfen lassen
- » Mikrocontroller und PC-Anwendungen programmieren

Checkliste



Zeitaufwand:
ab 1 bis 2 Stunden



Kosten:
keine

Mehr zum Thema

- » Benno Lottenbach, Sandbilder erstellen, Sandmaltisch Teil 2, Make 5/24, S. 86
- » Dirk Wahl, Das KI-Orakel, Make 4/24, S. 48
- » Dirk Wahl, KI-Sprachassistent mit eigenem sprechenden Avatar, Make 7/23, S. 80

Alles zum Artikel
im Web unter
make-magazin.de/xhza



Abbildung 1: Illustration (CAD) der Projektidee: Ein Gemütszustands-Barometer, mit dem sich die aktuelle Stimmungslage anzeigen lässt.



ChatGPT

ChatGPT (Generative Pre-trained Transformer) ist ein KI-gestütztes Sprachmodell, das von OpenAI entwickelt wird. Es wird durch das Training auf riesige Mengen von Text dazu befähigt, auf vielfältige Fragen zu antworten, Konversationen zu führen, Texte zu verfassen und komplexe Konzepte zu erklären. Der Grundservice von ChatGPT mit dem neuesten Sprachmodell GPT-4o ist kostenlos, aber limitiert auf eine bestimmte Anzahl von Anfragen. Wird diese Grenze überschritten, dann beantwortet ChatGPT alle nachfolgenden Anfragen mit einem anderen Sprachmodell, bleibt aber weiterhin kostenlos.

Wer sich für einen kostenpflichtigen Plus-Account bei ChatGPT entscheidet, kann auch das neueste Sprachmodell o1-preview nutzen, das für logische Aufgabenstellungen wie das Erzeugen von Code optimiert wurde. Weiterhin gibt es noch ChatGPT with Canvas, eine Funktion, die es Nutzern ermöglicht, während des Ge-

sprächs Textdokumente oder Code direkt in einem speziellen Inline-Editorbereich zu erstellen und zu bearbeiten. Dies bietet beim Programmieren Vorteile, da man Änderungen im Code übersichtlich nachvollziehen, speichern und iterativ verbessern kann. Die Canvas-Funktion steht im Modell GPT-4o zur Verfügung und ist aktuell von Plus-Nutzern und Enterprise-Kunden verwendbar. Wenn Canvas aus der Beta-Phase raus ist, soll es auch für die kostenlose ChatGPT-Version verfügbar werden.

Die textbasierte Eingabe, die man an ein Sprachmodell wie ChatGPT richtet, um eine Antwort zu erhalten, nennt sich Prompt. Der Prompt kann eine einfache Frage sein, eine detaillierte Beschreibung einer Aufgabe oder eine konkrete Anweisung. Alle Anfragen dieses Artikels wurden mit einem kostenlosen Account durchgeführt. Den Link zu ChatGPT findest du unter dem Link in der Kurzinfor-

Es fasst meine Anfrage in wenigen Sätzen zusammen und führt die für das Projekt notwendigen Komponenten auf. Die fiese falsche Fahrt, die ich mit dem Sandwich bewusst

gelegt habe, vermag ChatGPT gekonnt zu ignorieren. Darauf folgt bereits eine fein säuberlich strukturierte Schritt-für-Schritt-Anleitung: Ich werde unter anderem instruiert, welche Pins

des Arduino-Controllers mit welchen Anschlüssen des Servos, der Taster und der LEDs verbunden werden müssen. Dass die LEDs dabei einen Vorwiderstand zu 220 Ohm benötigen und die Taster einen Pull-down-Widerstand, wird löblicherweise ebenfalls erwähnt.

Aber damit nicht genug. Noch in derselben Antwort liefert ChatGPT einen kompletten, sauber strukturierten und kommentierten Arduino-Sketch. Dieser beinhaltet alle wichtigen Bestandteile, um den Code direkt ausführen zu können: So wird etwa die „Servo.h“-Library zu Beginn eingebunden und eine initiale Servoposition implementiert. Selbst an eine Entprellung des Signals der Taster (Abbildung 2, Zeilen 43/54) wurde gedacht. Der präzentierte Sketch wird dabei am Ende noch kurz und verständlich erläutert und von der KI schließlich mit der Anweisung beendet, dass der Code auf den Arduino hochzuladen und zu testen sei.

Von der Fülle an Informationen und der offenbar fundierten Hilfestellung ermutigt, versuche ich sofort, die Anweisungen in die Realität umzusetzen. Ich stecke die LEDs in das Breadboard und verbinde sie mit den Pins des Controllers. Aber schon bald gerate ich ins Stocken: GPT hat mich mit keinem Wort darauf hingewiesen, dass die Anode der LED mit dem langen Draht am positiven, die Kathode mit dem kurzen Draht am negativen Pol angeschlossen werden muss.

„Okay, dann wollen wir mal sehen!“, denke ich mir, und stecke leise kichernd absichtlich

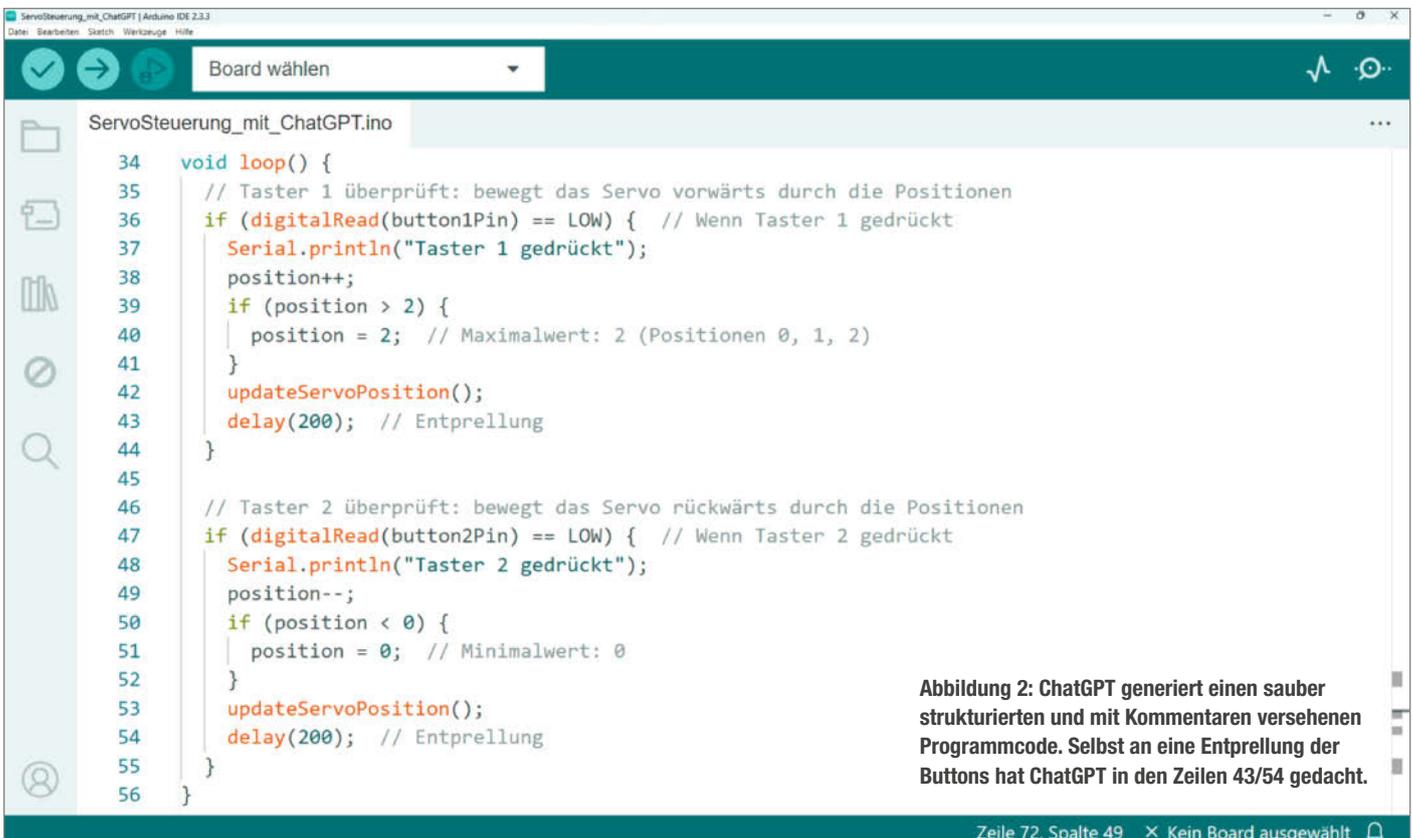


Abbildung 2: ChatGPT generiert einen sauber strukturierten und mit Kommentaren versehenen Programmcode. Selbst an eine Entprellung der Buttons hat ChatGPT in den Zeilen 43/54 gedacht.

Chatprotokoll

Das komplette unredigierte Chatprotokoll findest du online als PDF in unserem GitHub zum Artikel über den Link in der Kurzinfo. Die hier abgedruckten Prompts können Tippfehler und auch schweizerdeutsche Ausdrücke enthalten: Solange die Klarheit erhalten bleibt, haben wir in diesen Fällen nichts geändert.

Die Prompts des Autors sind in fett gedruckt und mit „**Prompt:**“ eingeleitet. Die meisten Antworten von ChatGPT wurden aber der besseren Lesbarkeit wegen im Text zusammengefasst.

eine der drei LEDs falsch herum ins Breadboard. Könnte ja schließlich jedem Maker so passieren. Gleichzeitig kann ich mich daran erinnern, dass ich im Laufe meiner Maker-Karriere diversen Tools zur Berechnung des LED-Vorwiderstands begegnet bin. Ohne irgendeine Nachfrage bezüglich der verwendeten LEDs und deren Farben nimmt ChatGPT also einfach an, Vorwiderstände von 220 Ohm würden genügen. „Na, wird schon passen“, sage ich mir und mache mich auf alles gefasst.

Ganz grundsätzlich muss ich bei der Verdrahtung der Schaltung feststellen, dass ein grafisches Schema der Verbindungen, wie in Abbildung 3 gezeigt, eine große Hilfe wäre. Ein Anfänger wäre sicherlich überfordert bei der Bestückung des Breadboards nur auf Basis der vorhandenen Anweisungen von ChatGPT. Zumal insbesondere das Abgreifen der Versorgungsspannung und das Zusammenführen der Leitungen zu Ground mit keinem Wort erwähnt wurden.

Fragt man ChatGPT nach einem Bild des Aufbaus, ist die KI sich sehr sicher, das darstellen zu können (siehe auch Chat-Aufzeichnung). Das resultierende Bild (Abbildung 4) ist aber nutzlos – man könnte es sogar gruselig nennen.

Als ich die beiden Taster im Breadboard einstecken will, stoße ich auf das erste Problem: Die Taster, die ich benutze, weisen vier Pins auf. Die KI spricht aber nur von zwei Pins pro Taster. Wie denn das? Meine elektronischen Grundkenntnisse würden zwar ausreichen, um dieses Problem mithilfe eines Multimeters selbstständig zu lösen. Aber ich bin gespannt, wie sich ChatGPT dazu äußert. Also frage ich:

Prompt: Wie muss ich die Taster genau anschließen? Die Taster, die ich verwende, weisen vier Pins auf und sie lassen sich in das Breadboard stecken.

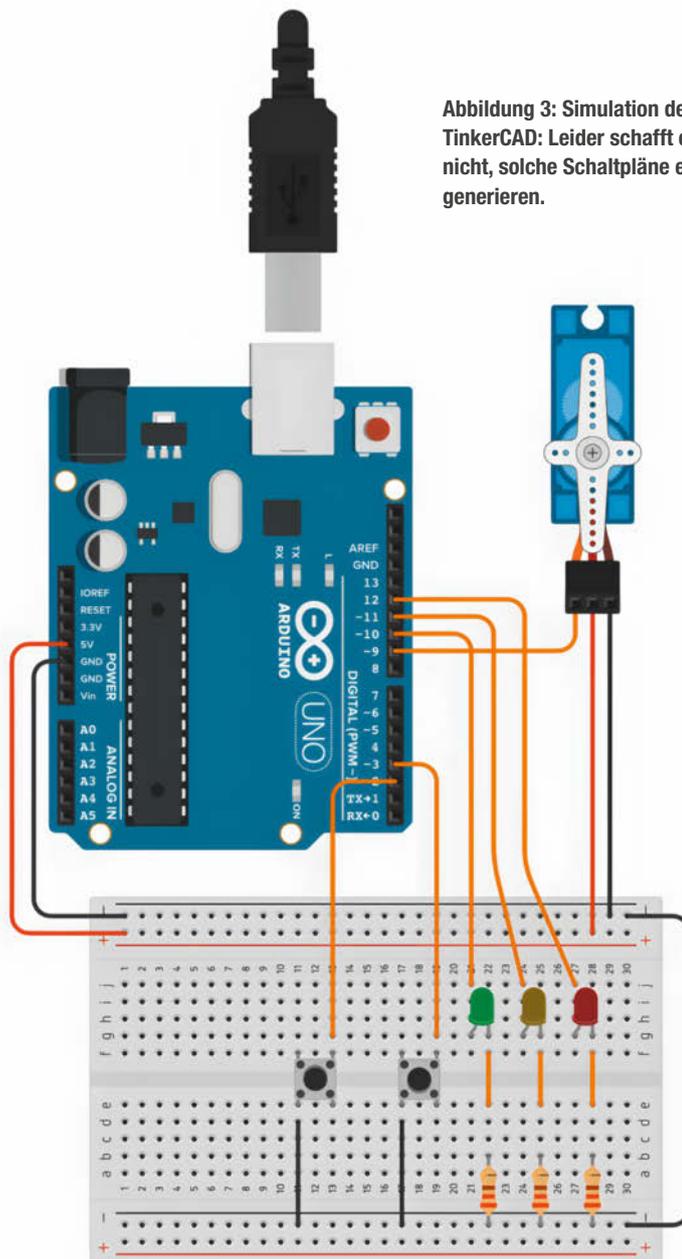


Abbildung 3: Simulation des Projekts in TinkerCAD: Leider schafft es ChatGPT noch nicht, solche Schaltpläne eigenständig zu generieren.

Die Antwort kommt schnell und präzise. Mir wird erklärt, wie diese vierbeinigen Taster intern aufgebaut sind und wie sie sich verhalten, gefolgt von einer detaillierten Anweisung, wie diese spezifische Tastervariante mit dem Arduino zu verbinden ist. Verblüffenderweise scheint sich ChatGPT plötzlich auch ungefragt daran zu erinnern, dass ein Arduino auch interne Pull-up-Widerstände aufweist, die bei Verwendung von Schaltern aktiviert werden können.

Freundlicherweise liefert mir die KI ungefragt die notwendigen Code-Anpassungen für diese einfachere Pull-up-Variante statt der bisher implementierten Pull-down-Variante. Das finde ich großartig und ich melde zurück, dass mir ChatGPT doch bitte den gesamten aktuellen Code ausgeben möge:

Prompt: Ich entscheide mich für deine vorgeschlagene Alternative mit dem internen

Pull-up-Widerstand. Bitte gib mir nun den kompletten Code aus, den ich in die Arduino IDE kopieren kann.

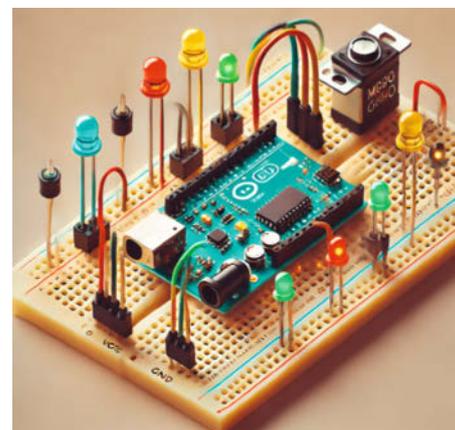


Abbildung 4: So wird das nichts.

Prompt: Benno Lottenbach / ChatGPT

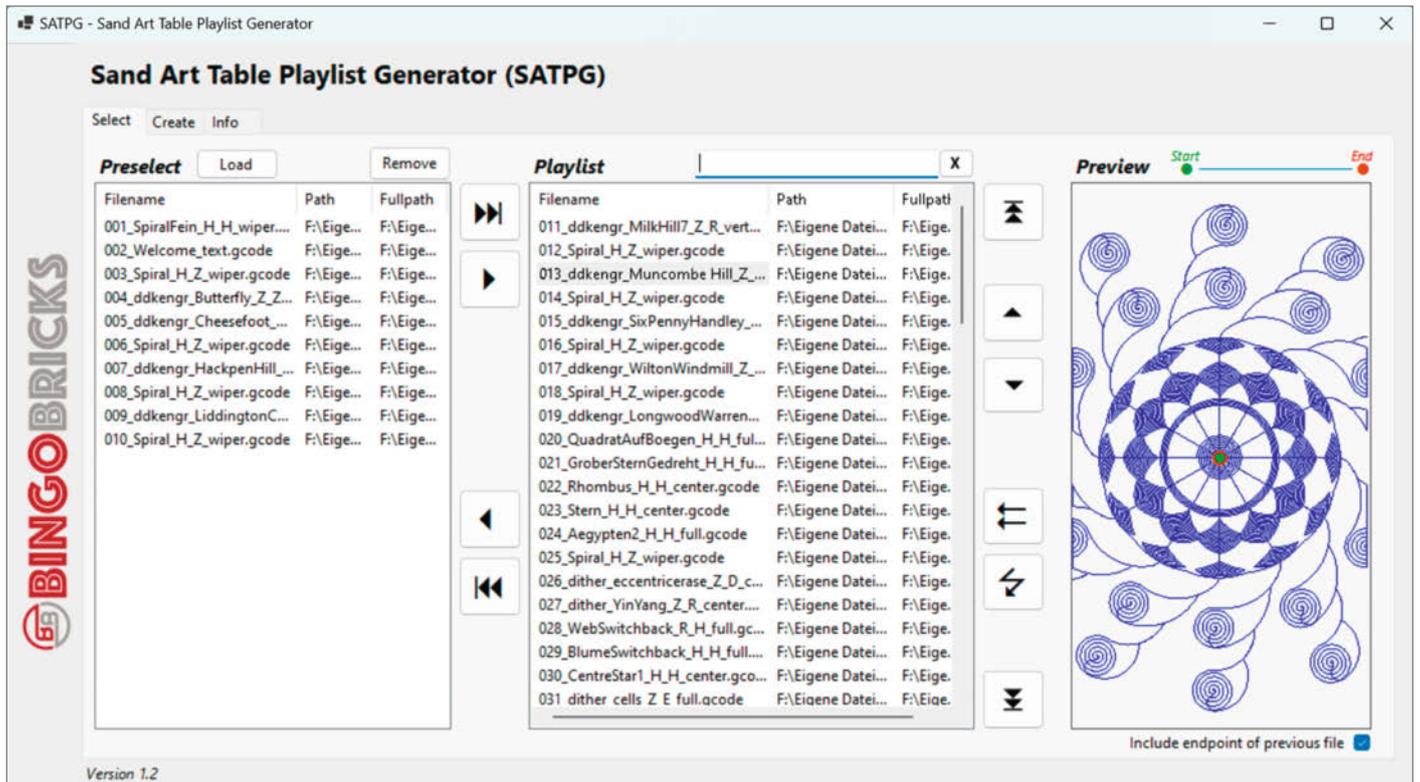


Abbildung 5: Der Programmcode des Tools „SATPG“ wurde größtenteils von ChatGPT generiert. Auch für die Erstellung der Bedienoberfläche ließ ich mich von ChatGPT beraten.

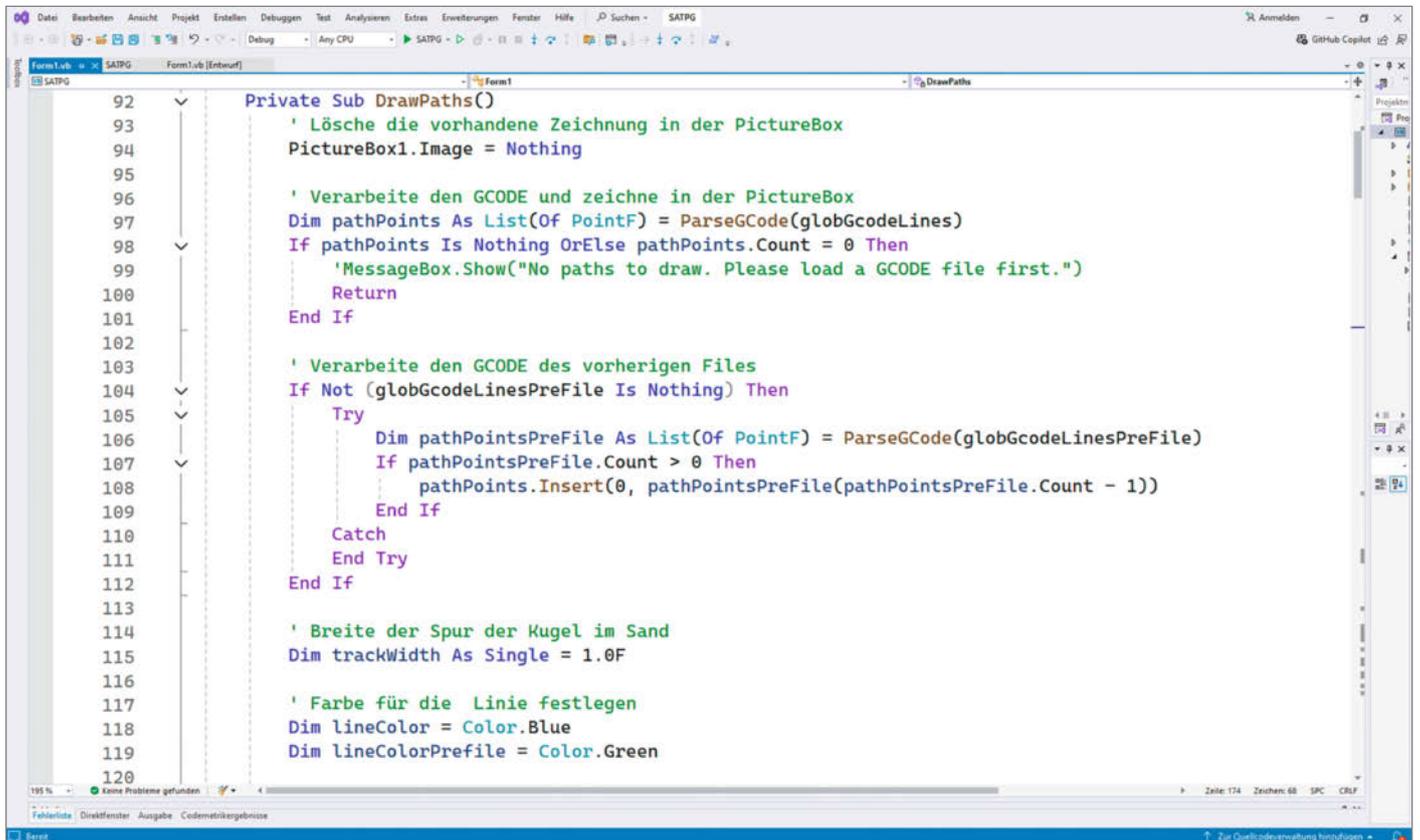


Abbildung 6: Die Übersetzung von Sandspurbreite in Linienbreite in den Zeilen 114/115 machte ChatGPT von sich aus. Die MessageBox in Zeile 99 habe ich auskommentiert, da sie in diesem Kontext unnützlich war.

Mein Wunsch scheint für die KI ein Befehl zu sein. Ich erhalte den kompletten Code inklusive einer erneuten Erläuterung des Code-Inhalts auf dem Bildschirm präsentiert. Diesen kopiere ich eins zu eins in die Arduino IDE und führe einen ersten Testlauf durch, wohl wissend, dass mindestens die eine absichtlich falsch herum angeschlossene LED nicht funktionieren wird. Aber auch abgesehen von dieser einen LED verläuft der anschließende Test leider negativ. Offen gesagt hätte es mich auch verwundert, wenn ChatGPT bereits mit dem ersten Code-Entwurf ein perfektes Ergebnis abliefern. Hämisches Grinsen tippe ich:

Prompt: Leider funktioniert es noch nicht ganz. Wenn ich eine der beiden Taster drücke, bewegt sich der Servo nicht. Der Servo hat sich aber beim Programmstart initialisiert und eine Grundposition eingenommen. Er ist also grundsätzlich ansprechbar.

Die Reaktion verblüfft mich. ChatGPT nennt gleich vier mögliche Ansätze, um dem Problem auf die Spur zu kommen. Nebst einem angepassten Sketch, der auf dem seriellen Monitor Debugging-Informationen ausgibt, weist mich die KI darauf hin, dass ich die Tastenverdrahtung prüfen solle. Und tatsächlich habe ich die beiden Taster im Breadboard unabsichtlich um eine Spalte versetzt eingesetzt. So kann das natürlich nicht funktionieren, ein peinlicher Fehler meinerseits. Etwas verlegen gebe ich den Fehler zu, und versuche dafür, ChatGPT mit der vermuteten LED-Wissenslücke in die Ecke zu treiben:

Prompt: Tatsächlich, ich hatte einen Fehler bei der Verkabelung der Taster gemacht. Jetzt funktioniert die Servo-Steuerung. Allerdings brennt eine der drei LEDs nicht. Was könnte das Problem sein?

Mein Vorpreschen vermag ChatGPT allerdings sofort in die richtigen Bahnen zu lenken, indem es gleich zwei Fliegen mit einer Klappe schlägt: Als erste und wahrscheinlichste Fehlerquelle wird nun nämlich auf die möglicherweise verpolten Anschlüsse (Anode und Kathode) der LED verwiesen. Und gleich an zweiter Stelle macht die KI mich darauf aufmerksam, dass auch die Stärke des Vorwiderstands überprüft werden sollte. Gut gemacht, Kollegin!

Innerhalb von etwa ein bis zwei Stunden hat mir ChatGPT also zu einer funktionierenden elektronischen Schaltung verholfen. Die Schaltung habe ich übrigens nicht nur real aufgebaut, sondern für diesen Artikel auch in Tinkercad nachgezeichnet und das Skript von ChatGPT für die Arduino-Simulation dort übernommen. Ihr könnt mithilfe des Links (siehe Kurzinfo) den Schaltungsaufbau simulieren, anpassen und den kompletten Code einsehen.

Das nächste Level

Beschwingt von dieser enorm temporeichen Unterstützung versuche ich mich anschließend an einem etwas komplexeren Software-Entwicklungsprojekt, das ich ohne ChatGPT niemals gewagt hätte in Angriff zu nehmen: Ich wollte eine Windows-Anwendung programmieren (lassen), mit der man Abspielisten für Sandmaltische zusammenstellen kann (siehe „Mehr zum Thema“). Das Tool soll die in mehreren G-Code-Dateien beschriebenen Pfade in einer grafischen Voransicht anzeigen. Dabei sollen die Dateien vom User in eine bestimmte Reihenfolge gebracht werden können, woraus das Tool schließlich nach gewissen Kriterien eine steuernde Textdatei erzeugen soll. Als Voraussetzung ist das Tool in Visual Basic zu erstellen.

Im Grundsatz ging ich dabei fast identisch vor wie beim Arduino-Projekt. Ganz am Anfang jedoch fragte ich ChatGPT um Rat, mit welcher Entwicklungsumgebung sich ein solches Projekt realisieren ließe und welche Art von Anwendung dafür am besten geeignet wäre. Auch hierzu hat mich ChatGPT bestens und vor allem sehr geduldig in allen Bereichen gut beraten können. Ebenso bei der Erstellung des Programmcodes in Visual Basic war die KI sehr flott und kompetent unterwegs. Gelegentlich gab es ein paar Pausen bei der Ausgabe des Codes, der bei einer Windows-Anwendung schnell recht umfangreich werden kann. Der Trick besteht dann darin, dass man sich in der Entwicklungsphase nur noch einzelne Funktionen von ChatGPT ausgeben lässt. Zwischendurch lässt sich dann der gesamte Programmcode abrufen, den man sich mit Copy-and-paste

in die gewählte Entwicklungsumgebung kopieren kann.

Besonders interessant fand ich, dass ich für ChatGPT auch ganz „technikfern“ meine Anforderungen formulieren konnte und die KI anschließend meine umgangssprachlichen Formulierungen abstrahierte und in Visual Basic übersetzte. So beschrieb ich unter anderem die grafische Voransicht als „grafische Simulation für Sandspuren, die eine Kugel im Sand zeichnet“. ChatGPT merkte dann irgendwie selbstständig, dass ich damit die in der G-Code-Datei angegebenen Pfade meinte. Man darf also ruhig einfach mal drauflos plaudern, genauso, wie man es mit einem echten Bastelkollegen aus Fleisch und Blut tun würde.

Es würde an dieser Stelle zu weit führen, um detailliert auf die Entstehungsgeschichte dieses Softwareprojekts einzugehen. Doch so viel will gesagt sein: Das Resultat vermag durchaus zu überzeugen. Zwar habe ich an einigen Stellen meiner Windows-Anwendung noch selbst Hand angelegt, um die Sache nach meinen Vorstellungen zum Laufen zu bekommen. Aber ChatGPT hat deutlich mehr dazu beigetragen, als nur eine Code-Basis zur Verfügung zu stellen. Ich schätze, dass rund 80 Prozent des Codes von der KI generiert wurden und ich nur die restlichen Prozente beigetragen habe.

Insbesondere die kleinen, einfacheren Funktionen und Änderungen habe ich selbst programmiert, weil eine Beschreibung per Prompt wahrscheinlich länger als eine direkte Programmierung gedauert hätte. Auch die Bedienoberfläche habe ich größtenteils

selbst erstellt, wobei ich mich für die komplexeren Steuerelemente (z. B. den „Datei öffnen“-Dialog oder ListView-Elemente) von ChatGPT führen ließ. Alles in allem muss ich zugeben, dass ich mit meinen eher rudimentären Programmierkenntnissen ohne die Hilfe von ChatGPT wahrscheinlich gescheitert wäre. Jetzt aber steht sie, meine erste ordentliche Windows-Anwendung.

In diesem Sinne möchte ich ChatGPT in meinem Bastelkeller kaum mehr missen. Es mag sein, dass diese KI nicht für jedes Projekt gleichermaßen geeignet ist. Aber es lohnt sich auf alle Fälle, die Hilfeleistung dieses intelligenten Bastelkollegen bei euren Maker-Projekten zu prüfen. Eine Sammlung meiner Erfahrungen und Tipps für das Codieren mit ChatGPT findet ihr im Kasten „Umgang mit ChatGPT“. Und übrigens: Dass auch ChatGPT Humor versteht, zeigt sich insbesondere an einem meiner letzten Prompts für das Arduino-Projekt:

Prompt: Und was machen wir jetzt noch mit dem Schinken-Sandwich, das ich zu Beginn des Projektes bereitgestellt habe?

ChatGPT: Ah, das Schinken-Sandwich! Nachdem dein Projekt jetzt erfolgreich abgeschlossen ist, bleibt wohl nur noch eine würdige Belohnung: Das Sandwich genießen! Du hast es dir definitiv verdient nach der erfolgreichen Arbeit an deinem Robotikprojekt. Guten Appetit, und lass es dir schmecken! Wenn du währenddessen noch über ein weiteres Projekt nachdenken möchtest, bin ich jederzeit bereit! —caw

Make:



DEUTSCHLANDS GEFÄHRLICHSTES ABO-ANGEBOT*

*VON DEUTSCHLANDS GEFÄHRLICHSTEM DIY-MAGAZIN (LAUT LESERN)

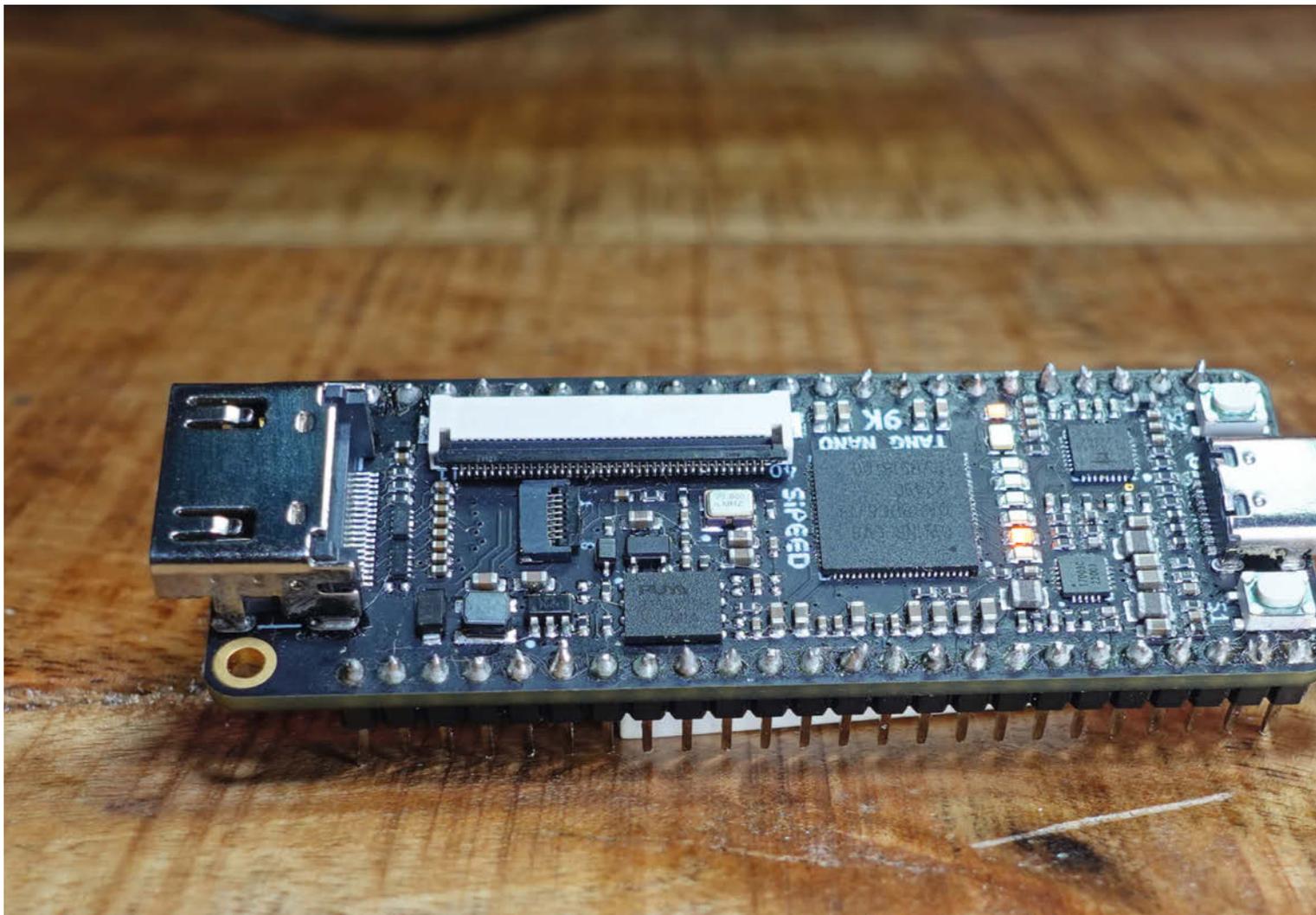
2× Make testen mit über 30 % Rabatt

Jetzt bestellen: make-magazin.de/abo-angebot

Warum eigentlich gefährlich?

Laut Lesern sind wir das "gefährlichste DIY-Magazin" Deutschlands. Das ist aber natürlich nur Spaß! Sie können unser Magazin ganz unbesorgt lesen und 2 Ausgaben als Heft + digital testen, zusätzlich erhalten Sie ein Geschenk Ihrer Wahl – klingt doch eigentlich ganz ungefährlich.





FPGA für Maker

Field Programmable Gate Arrays (FPGA) sind konfigurierbare integrierte Schaltkreise, mit denen sich per Software beliebige digitale Schaltungen realisieren lassen – sogar komplette Mikroprozessoren oder ganze Computer! Unser Artikel hilft beim Einstieg ins Thema.

von Michael Linsenmeier



Kurzinfo

- » Funktionsweise von FPGAs
- » Vor- und Nachteile von FPGAs gegenüber Mikrocontrollern
- » Erstellung eines FPGA-Designs anhand eines Beispiels

Checkliste



Zeitaufwand:
2 Stunden



Kosten:
ca. 30 Euro

Mehr zum Thema

- » Daniel Bachfeld, RISC-V für Maker, Make 4/23, S. 94
- » Carsten Meyer, Digital filtern und oszillieren, Make 6/22, S. 84

Alles zum Artikel im Web unter make-magazin.de/xjra

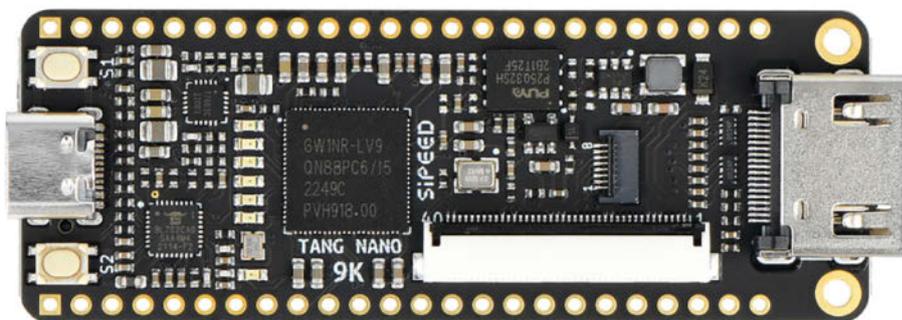


Bild 1: „Tang Nano 9k“-FPGA-Board

Noch fristen FPGAs (Field Programmable Gate Array) bei Elektronikbastlern eher ein Nischendasein. Dies liegt vor allem an ihrer vergleichsweise komplexen Programmierung, dem hohen Preis und dem unpraktischen SMD-Gehäuse. Mit den „Tang Nano Boards“ von Sipeed stehen aber gleich vier FPGA-Boards mit unterschiedlicher Leistung und günstigen Preisen zur Verfügung, die erste Experimente schmackhaft machen. Ich werde im Folgenden die Funktion von FPGA am Beispiel des „Tang Nano 9k“ (Bild 1) erklären.

Das Herz der Platine ist ein „Gowin GW1NR-9“-FPGA (der große Chip in der Mitte). An der Stirnseite befindet sich ein USB-C-Anschluss, über den das Board sowohl mit Strom versorgt als auch konfiguriert und programmiert werden kann. Gegenüber befindet sich ein HDMI-Anschluss. Gowin bietet kostenlose Bibliotheken, die sogenannten IP-Cores (Intellectual Property), an, mit denen das DVI-Protokoll (DVI über HDMI) mit wenig Aufwand implementiert werden kann. Damit eignet sich das „Tang Nano 9k“ bestens für Projekte, die einen Monitor benötigen, zum Beispiel für Retro-

Computer. Außerdem gibt es Anschlüsse für parallele und serielle LC-Displays, einen SD-Karteneinschub, zwei Drucktaster, sechs orange On-Board-LEDs und 45 digitale Ein- / Ausgänge (GPIOs). Doch bevor es ans Praktische geht, ein kleiner Refresher zum Thema.

Ein FPGA ist ein integrierter Schaltkreis, der so konfiguriert werden kann, dass er eine Vielzahl von logischen Funktionen ausführt. Der Begriff „field programmable“ bedeutet, dass das FPGA nach der Herstellung vom Benutzer programmiert bzw. konfiguriert werden kann, um spezifische Funktionen auszuführen. Das macht FPGAs sehr flexibel und anpassbar. Doch wo ergibt der Einsatz von FPGAs Sinn und wann sind Mikrocontroller besser geeignet? Dazu muss man zunächst verstehen, wie ein FPGA funktioniert.

Aufbau eines FPGA

Die Kernelemente jedes FPGA sind die konfigurierbaren Logikblöcke (CLB: Configurable Logic Blocks). Jeder CLB enthält mehrere sogenannte Look-up Tables (LUTs), die einfache

boolesche Funktionen umsetzen, sowie Flip-flops oder Register, um deren Ergebnisse zwischenspeichern. Bei einer LUT handelt es sich um eine Tabelle, die für jede mögliche Kombination von Eingaben das entsprechende Ergebnis einer logischen Funktion liefert. Anstatt logische Funktionen durch Verbindungen zwischen UND-, ODER- und NOT-Gates zu realisieren, geschieht dies in einem FPGA durch das Schreiben von Nullen und Einsen in die LUTs. In Bild 2 ist ein Beispiel einer LUT für eine Exklusiv-ODER-Verknüpfung (XOR) abgebildet.

Die ersten beiden Tabellenspalten A und B bilden die vier möglichen Zustände der beiden Eingänge ab, die dritte Spalte enthält die Ergebnisse der Verknüpfung. Kurz gesagt enthalten die LUTs einfach die Logiktable der zu realisierenden Schaltung. Ein nachfolgender Multiplexer schaltet das Ergebnis-Bit der über die Eingangs-Bits A und B adressierten Zeile auf den Ausgang (Y).

Die LUTs von FPGAs haben im Allgemeinen vier Eingänge und können daher digitale Schaltungen mit bis zu vier Eingängen dar-

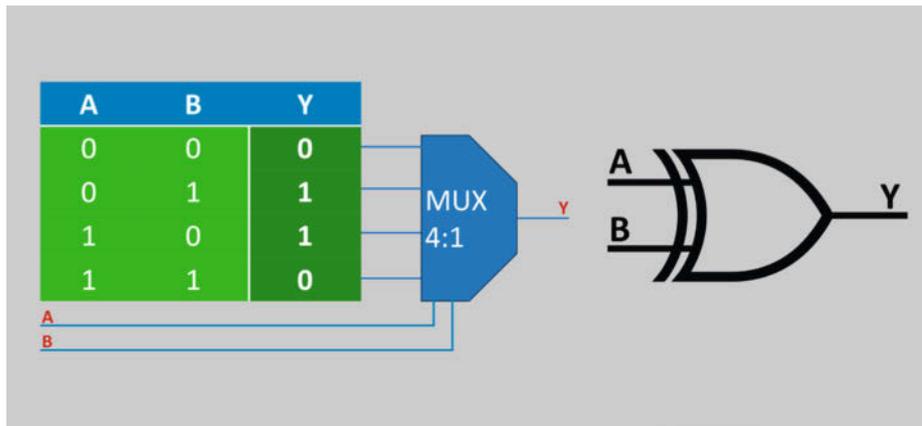


Bild 2: LUT einer XOR-Verknüpfung

stellen. Im FPGA der „Tang Nano“-Boards bilden jeweils zwei LUTs und zwei zusätzliche Flipflops zum Speichern von Zuständen eine Configurable Logic Section (CLS)

LUTs können auch als Speicherzellen mit vier Adresseingängen und einem Bit Zugriffsbreite betrachtet werden. Damit können sie insgesamt 16 Bit speichern.

Die CLS im Bild 3 realisiert einen Volladdierer. Die obere LUT liefert die Summe der Bits A und B unter Berücksichtigung des Übertrags C_{in} . Das Ergebnis wird im Register Sum für spätere Verwendung zwischengespeichert. Die untere LUT bestimmt den Übertrag für den Addierer des nachfolgenden, höherwertigen Bits.

Bei den „Tang Nano“-FPGAs bilden vier dieser CLSs einen Configurable Logic Block (CLB, Bild 4). Mittels der Configurable Routing Unit (CRU) können die Verbindungen zwischen den LUTs und auch weiteren Komponenten des Chips wie Ein-/Ausgabeblocke, Oszillatoren, Speicher usw. per Software konfiguriert werden.

Um beim Beispiel des Addierwerks zu bleiben: Um zwei Bits zu addieren, benötigt man zwei LUTs (eine CLS), für zwei Nibbles (4 Bit) einen CLB und für zwei Byte (8 Bit) schließlich zwei CLBs. Man sieht: Je komplexer die zu realisierende Schaltung ist, desto mehr dieser Blöcke werden benötigt. Die Anzahl der LUTs

ist damit ein wichtiges Kriterium bei der Auswahl von FPGAs und bestimmt maßgeblich deren Preis.

Es gibt sowohl FPGAs mit wenigen hundert LUTs für unter 10 Euro als auch Typen mit mehr als einer Million LUTs für mehrere zehntausend Euro. In Tabelle 1 ist ein Vergleich der verschiedenen „Tang Nano“-FPGAs gelistet. Die Anzahl der Register ist um ein Viertel kleiner als die Anzahl der LUTs, da jeweils eine von vier CLS keine Register enthält.

Damit ein FPGA mit der Außenwelt kommunizieren kann, besitzt er zahlreiche Ein- und Ausgänge, sogenannte IO-Blöcke (IOBs), siehe Bild 5.

Der im „Tang Nano 9k“ verbaute FPGA „Gowin GW1NR-9 (QN88P)“ kann darüber bis zu 70 IO-Pins ansteuern, wovon 45 über den Pin-Header des Boards und weitere über seine Peripherieschnittstellen (HDMI, MicroSD, SPI-Display etc.) nutzbar sind. Außerdem besitzt der Chip zahlreiche integrierte On-Chip-Komponenten wie SRAM, Flash-Speicher, digitale Signalprozessoren (DSP) für schnelle Fließkommaoperationen, einen Oszillator und zwei Phase Locked Loops (PLL) zur Erzeugung beliebiger Taktfrequenzen. Die DSPs in den Gowin-FPGAs sind Recheneinheiten, mit denen Additionen und Multiplikationen mit bis zu 54 Bit Auflösung und sehr hoher Geschwindigkeit ausgeführt werden können: Das Design bestimmt dann, welche Signale verarbeitet werden. Um die On-Chip-Komponen-

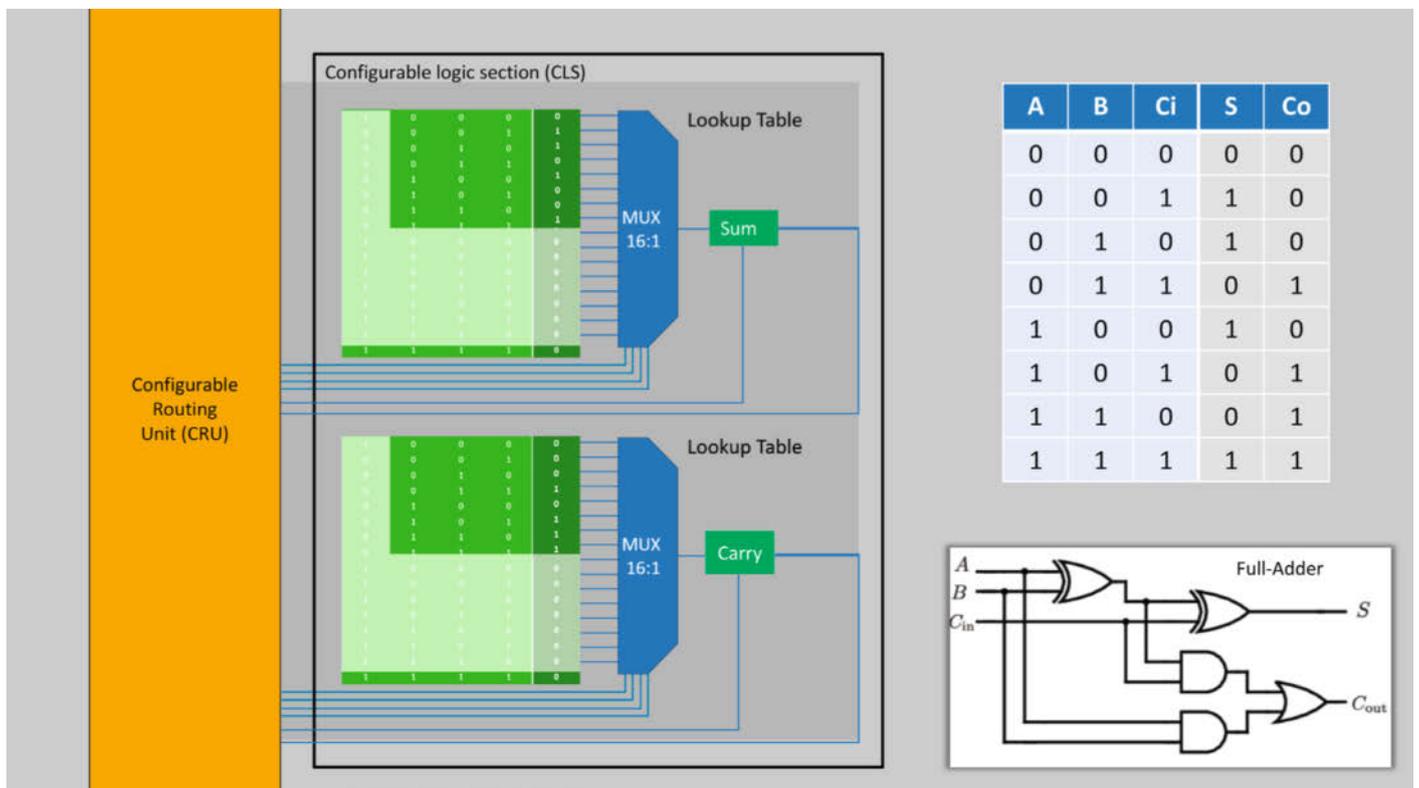


Bild 3: Configurable Logic Section (CLS) für einen Volladdierer

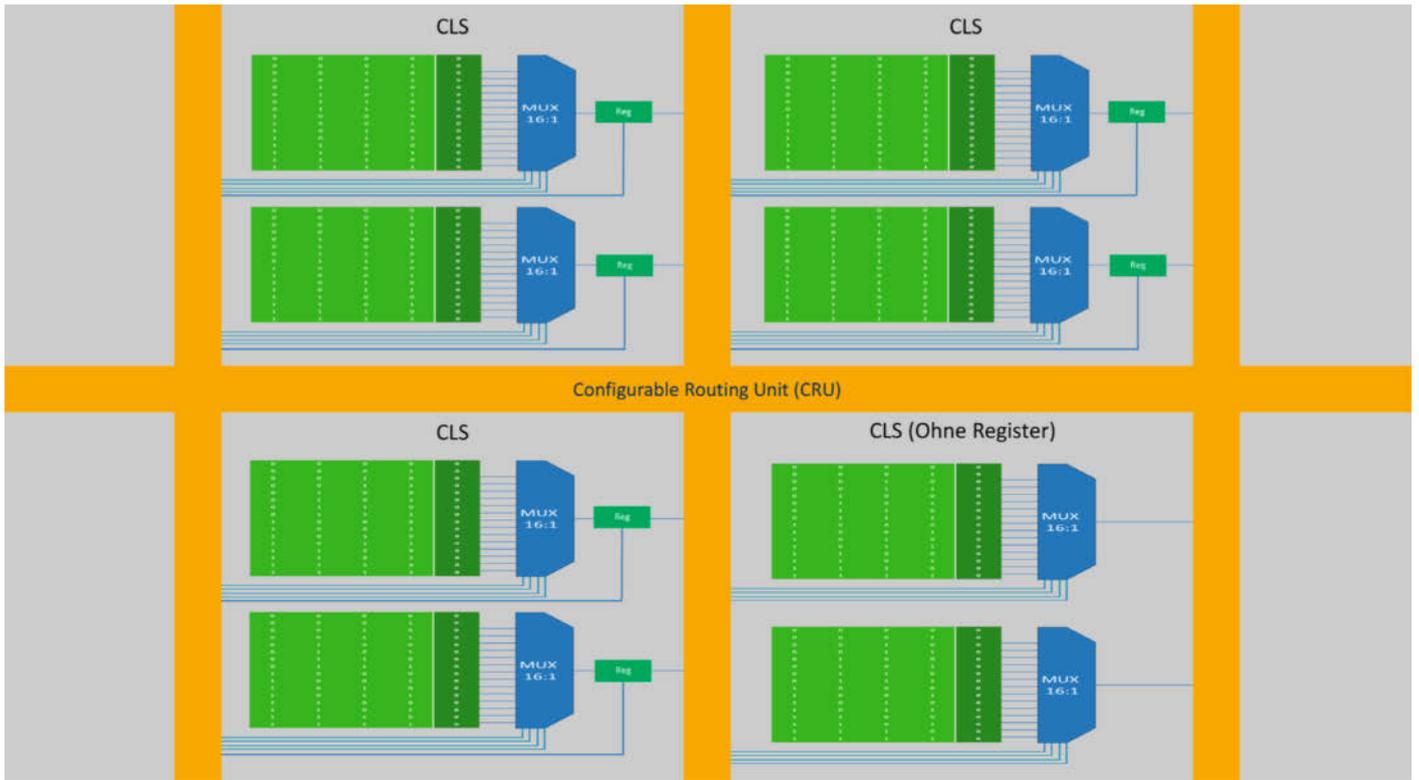


Bild 4: Configurable Logic Block (CLB)

ten nutzen zu können, benötigt man die dazu passenden IP-Cores des Herstellers. Siehe auch Tabelle 1 für die Ausstattung der unterschiedlichen „Tang Nano“-Boards und deren FPGAs.

FPGAs und Mikrocontroller haben jeweils ihre eigenen Vor- und Nachteile. Es ist also eine grundlegende Entscheidung jedes Entwicklers, ob er sein Projekt mit einem FPGA oder einem Mikrocontroller realisieren möchte.

Vorteile von FPGA

FPGAs arbeiten parallel: FPGAs können viele Aufgaben gleichzeitig ausführen, da ihre Logikblöcke parallel arbeiten. Jeder Block kann unabhängig von den anderen konfiguriert werden. Daher sind FPGAs bezogen auf ihre Taktfrequenz deutlich schneller als Mikroprozessoren. Diese arbeiten sequenziell, da sie auf einer CPU mit einem vorgegebenen Takt basieren, die nur eine Aufgabe nach der anderen verarbeitet.

FPGAs sind sehr flexibel: Entwickler können ihre eigene Architektur für bestimmte Aufgaben entwerfen und Hardware-Funktionen implementieren, die auf spezifische Anwendungen optimiert sind (zum Beispiel Signalverarbeitung). Dagegen arbeiten Mikrocontroller mit einer vorgegebenen Architektur und einem festgelegten Befehlssatz. Die Flexibilität ist dadurch stark eingeschränkt.

FPGAs sind schnell: FPGAs bieten hervorragende Echtzeitfähigkeiten, da sie durch die parallele Verarbeitung Hardwareprozesse mit

deterministischem Verhalten implementieren können. Dadurch erhält man hohe Timing-Genauigkeit und geringe Latenzzeiten. Mikrocontroller können zwar auch in Echtzeitanwendungen eingesetzt werden, doch durch die sequenzielle Abarbeitung der Befehle und unterschiedliche Ausführungszeiten ist deren Programmierung in diesem Echtzeitbereich sehr komplex und fehleranfällig.

FPGAs haben viele parallele I/O-Kanäle: Damit können sie eine große Anzahl (und sogar alle) I/O-Pins zeitgleich ansprechen, was für Anwendungen wie Kommunikationschnittstellen oder Steuerungssysteme nützlich

ist. Bei Mikrocontrollern ist nur eine kleine Anzahl von I/O-Pins parallel ansprechbar (z. B. acht bei ATmega)

Vorteile von Mikrocontrollern

Komplexität der Entwicklung: Mikrocontroller sind mit Hochsprachen wie C oder Python einfacher zu programmieren. Entwickler-Tools und -Bibliotheken sind oft benutzerfreundlicher. Rund um Mikrocontroller gibt es große Communities, in denen man bei Problemen schnell Hilfe findet. Das Entwickeln von Anwendungen für ein FPGA ist komplizierter und

	Tang Nano 1k	Tang Nano 4k	Tang Nano 9k	Tang Nano 20k
FPGA	GW1NZ-LV1	GW1NSR-LV4C	GW1NR-9	GW2AR-18
LUTs	1.152	4.608	8.640	20.736
Register	864	3.456	6.480	15.552
Block RAM (18 kBit)	4	10	26	46
PSRAM/SDRAM (MByte)	-	8	8	8
DSP	-	8	10	24
PLL	1	2	2	4
Prozessor	-	ARM Cortex M3	Risc V Soft CPU	-
Flash (kBit)	64	256	608	-
Breadboard I/Os	32	38	45	34
Schnittstellen	RGB 40-pol	HDMI, DVP	HDMI, SPI 8-pol, RGB 40-pol, SD-Karte	HDMI, RGB 40-pol, SD-Karte
Preis (ab ca.)	10 Euro	15 Euro	20 Euro	30 Euro



Bild 5: Vereinfachte Darstellung des „Gowin GW1NR-9“-FPGA

zeitaufwendiger. Es erfordert spezielle Hardware-Beschreibungssprachen (HDLs) wie VHDL oder Verilog. Die Lernkurve ist steiler, besonders für Entwickler, die mit klassischen Programmiersprachen arbeiten.

Kosten: Mikrocontroller sind in der Regel günstiger als FPGAs, sowohl in Bezug auf die Hardware als auch auf die Entwicklungswerk-

zeuge und -umgebung. Vor allem in kleineren Stückzahlen können FPGAs signifikant teurer sein als Mikrocontroller.

Startzeit und Initialisierung: Mikrocontroller starten in der Regel schneller, da das Programm aus dem Speicher geladen und sofort ausgeführt wird. FPGAs brauchen beim Start oft länger, da die Konfiguration aus dem

Flash-Speicher in das FPGA geladen werden muss. Das bedeutet, dass es nicht sofort nach dem Einschalten bereit ist.

Projektabhängig wählen

Damit sind FPGAs ideal für Anwendungen, die extrem schnelle Verarbeitung, parallele Aufgaben und Echtzeitfähigkeiten erfordern, z. B. in der digitalen Signalverarbeitung, Bildverarbeitung und Kommunikationssystemen. Allerdings sind sie teurer und komplizierter zu programmieren. Mikrocontroller hingegen bieten eine kostengünstige und leicht zu implementierende Lösung für viele Anwendungen, insbesondere wenn die Anforderungen an Geschwindigkeit und parallele Verarbeitung moderat sind.

Es soll an dieser Stelle nicht unerwähnt bleiben, dass es auch FPGAs mit integrierter CPU gibt. Zum Beispiel enthält der GW1NSR-LV4C im „Tang Nano 4k“ eine komplette ARM-Cortex-M3-CPU. Wenn man ein FPGA mit genügend Logikblöcken hat, kann man natürlich auch einen Teil davon als CPU „verdrahten“ und die anderen Blöcke für die entsprechende Echtzeit oder High-Speed-Anwendung nutzen. So kommt man mit einem FPGA in den Genuss der Vorteile beider Welten. Es gab in den letzten Jahren einen rapiden Preisverfall bei den FPGAs, daher könnte ich mir gut vorstellen, dass dezidierte Mikrocontroller nach und nach durch „Software-MCUs“ auf FPGAs ersetzt werden.

Für das „Tang Nano 9k“ gibt es ein FPGA-Design für eine „PicoRV32 RiscV“-Soft-CPU. Es gibt dazu auch eine Beispielimplementierung, die den HDMI-Anschluss nutzt. Dies ist ein super Ausgangspunkt für einen eigenen, selbst designten Mikrocontroller, eine Spielekonsole oder einen Computer!

Programmierung von FPGAs

Streng genommen werden FPGAs nicht programmiert, sondern konfiguriert. Ein Programm besteht immer aus einer Serie von Anweisungen, die sequenziell abgearbeitet werden. Das „Programm“ eines FPGA beschreibt hingegen die logischen Funktionen (Inhalt der LUTs) und deren Verbindungen untereinander sowie mit den I/O-Blöcken und anderen On-Chip-Komponenten (Speicher, PLLs etc.). Diese bleiben bis zur „Neuprogrammierung“ unverändert.

Man nennt die „Programme“ für FPGAs daher Design. Diese werden in einer Hardware Description Language (HDL) geschrieben. Die verbreitetsten sind VHDL sowie Verilog und dessen Weiterentwicklung SystemVerilog. Wegen der einfacheren und kompakteren Schreibweise ermöglicht Verilog einen unkomplizierten Einstieg in die Welt der FPGAs.

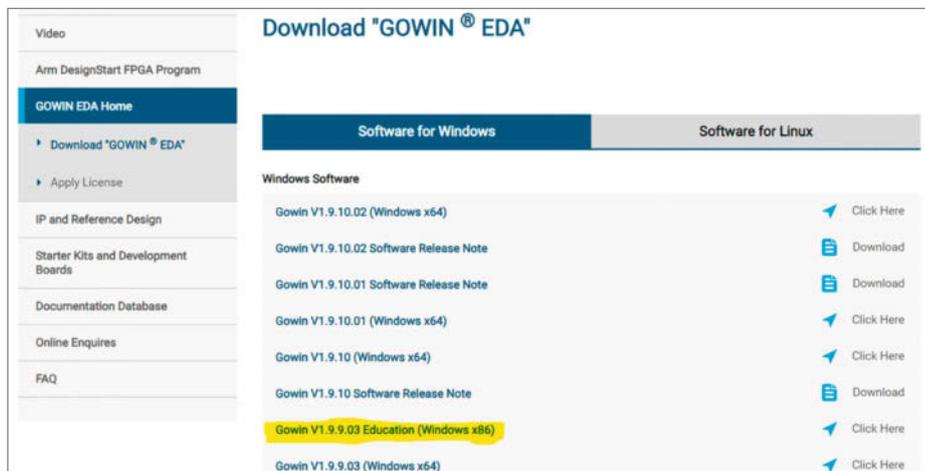


Bild 6: Downloadseite der Gowin EDA

ENTFESSELE DEN GEEK IN DIR

Mehr Möglichkeiten, mehr Inspirationen – Erweitere deine Oxocard Connect um spannende Cartridges!

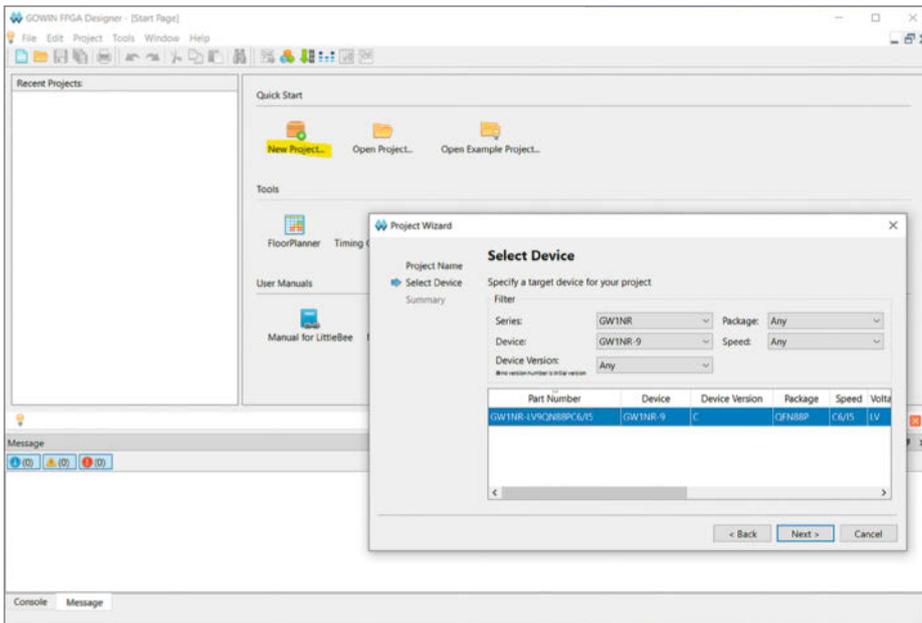


Bild 7: Auswahl des FPGA-Typs im „Project Wizard“

VHDL ist vor allem in Forschung und Wissenschaft, aber auch in älteren Designs weitverbreitet. Verilog und SystemVerilog erfreuen sich aufgrund ihrer C-ähnlichen Struktur und kompakteren, leichter lesbaren Syntax zunehmender Beliebtheit. SystemVerilog fügt dem klassischen Verilog noch weitere Funktionen hinzu, wie Objektorientierung und benutzerdefinierte Datentypen.

Die Entwicklungsumgebung

Jeder Hersteller stellt eine eigene Entwicklungsumgebung (IDE) für seine jeweiligen FPGAs zur Verfügung. In der Basisversion sind diese kostenlos. Möchte man aber deren vollen Funktionsumfang nutzen, muss man auf die kostenpflichtigen Vollversionen zurückgreifen. Für den Einstieg oder für Hobbyzwecke sind die Basisversionen aber völlig

ausreichend. Es gibt inzwischen auch Open-Source-Alternativen. Jedoch werden die zahlreichen FPGA-Typen der unterschiedlichen Hersteller meist entweder gar nicht oder nur unvollständig unterstützt, und auch der Funktionsumfang lässt oft Wünsche offen.

Für die „Tang Nano“-Boards gibt es aber eine recht brauchbare Open-Source-Alternative: Lushay Code von Lushay Labs. Diese Erweiterung für VS Code kann einfach über dessen Erweiterungs-Manager installiert werden. Der Funktionsumfang ist recht ordentlich, allerdings kann man damit nicht die On-Chip-Komponenten wie die PLLs oder den internen Flash nutzen, da die entsprechenden IP-Cores nicht enthalten sind. Mehr Information finden Sie auf der Website über den Link in der Kurzinfor. Dort gibt auch ein Tutorial und zahlreiche interessante Beispieldesigns.

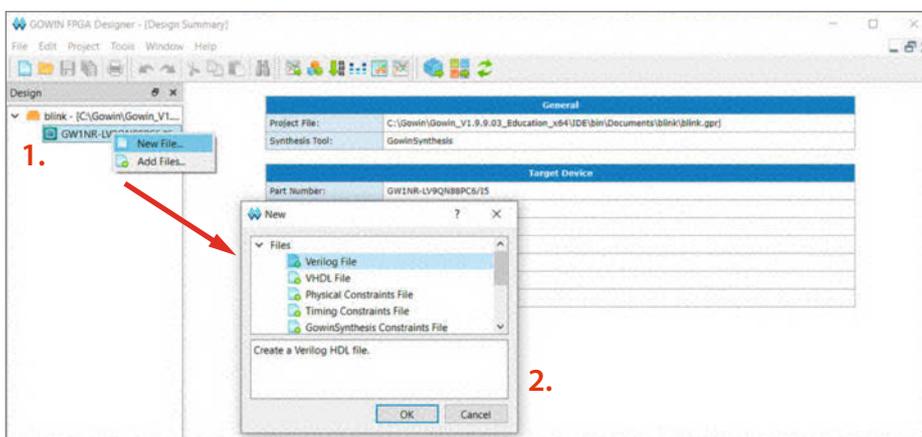
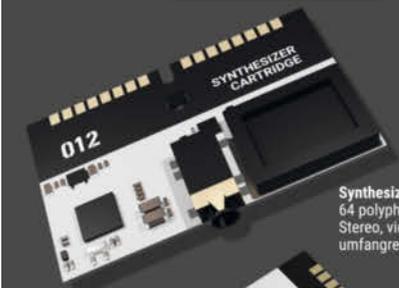


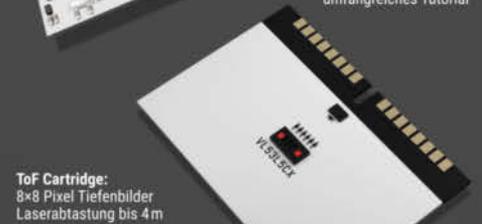
Bild 8: So fügt man eine Verilog-Datei zum Designprojekt hinzu.



Oxocard Connect Modul



Synthesizer Cartridge: 64 polyphone Stimmen, Stereo, viele Klangeffekte, umfangreiches Tutorial



ToF Cartridge: 8x8 Pixel Tiefenbilder Laserabtastung bis 4m



Pixelmatrix Cartridge: 5x5 RGB-Pixelmatrix – sofort einsatzbereit oder zum Herausbrechen und kreativen Basteln

inkl. 25 Diffusoren



AIR Cartridge: CO₂, VOC, NO_x, Temperatur und Feuchtigkeit messen



Jetzt im heise Shop bestellen

www.oxocard.ch

Listing 1: Verilog Lauflicht

```

module lauflicht (
    input sys_clk,           // Takteingang
    output reg [5:0] led    // 6 LEDs
);

reg [23:0] counter;

initial
    led <= 6'b111110;

always @(posedge sys_clk) begin // Bei positiver Flanke des Taksignals
    if (counter < 24'd13_500_000) // So lange counter-Reg. <13.500.000
        counter <= counter + 1'b1; // Erhöhe um 1
    else
        counter <= 24'd0; // Setze "counter" Register zurück
end

always @(posedge sys_clk) begin
    if (counter == 24'd13_500_000) // Passiert alle 0,5 Sekunden
        led[5:0] <= {led[4:0],led[5]}; // Schiebe Bit nach Links
end

endmodule
    
```

Lauflicht für FPGA

Ich möchte nun die sechs LEDs eines „Tang Nano 9k“-FPGA-Boards als Lauflicht ansteuern. Dazu verwenden wir die IDE Gowin EDA des FPGA-Herstellers Gowin. Nach einer kostenlosen Registrierung kann sie von dessen Homepage heruntergeladen werden (Bild 6).

KASTEN: Wires versus Register

Einfache Verschaltungen innerhalb oder zwischen Modulen werden „wire“, also Draht/Signalleitung oder kurz Signal genannt. Mit dem Schlüsselwort `assign` werden die Signale „verdrahtet“. Damit können wir bereits einfache logische Verknüpfungen basteln, hier als Beispiel ein NAND-Gatter (Nicht-UND):

```

module NAND (
    input in1,
    input in2,
    output out
);

    assign out = ~(in1 & in2);

endmodule
    
```

Wenn wir Signale zwischenspeichern wollen, kommen Register, also die Flipflops am Ausgang der LUTs zum Einsatz. Es ist wichtig zu wissen, dass Register nur innerhalb von Blöcken (Initial- oder `always@`-Blöcke) geschrieben werden können, Signale nur außerhalb von Blöcken.

Die kostenlose Education Version ist für unsere Zwecke völlig ausreichend.

Sie unterstützt im Gegensatz zur Vollversion nicht alle FPGA-Typen, und es sind auch nicht alle IP-Cores darin enthalten. Man kann die Version aber für alle „Tang Nano“-Boards verwenden. Ich selbst habe den „Gigatron TTL Computer“ in dem Community-Projekt auf Seite 68 damit in Verilog design. Eine Lizenz für die Vollversion kann über ein Webformular von Gowin angefragt werden. Laut eines Forenbeitrags ist diese seit September 2024 ebenfalls kostenlos.

Nach der Installation und dem Start der Applikation erstellen wir ein neues Projekt, indem wir auf „New Project...“ im Quickstart-Fenster klicken. Nachdem wir einen passenden Namen und Speicherort für unser Projekt vergeben haben, müssen wir den FPGA-Typ (Bild 7) des zu konfigurierenden Boards auswählen. Für den „Tang Nano 9k“ ist das der GW1NR-9. Für die anderen „Tang Nano“-Boards kann der FPGA-Typ aus Tabelle 1 ermittelt werden.

Nun erstellen wir eine neue Verilog-Datei, indem wir mit der rechten Maustaste auf das neue Projekt in der Liste der Designs klicken und im Kontextmenü „New File“ (Bild 8) auswählen.

Und schon können wir loslegen und unser erstes FPGA-Design (Listing 1) im Editor-Bereich schreiben. Die komplette Schritt-für-Schritt-Anleitung mit dem Codebeispiel finden Sie auf der Wiki-Seite der „Tang Nano“-Boards, verlinkt über den Kurzinfo-Link.

Das ursprüngliche Beispieldesign hat einen Rücksetzeingang (Reset), der mit einem der beiden Drucktaster verbunden ist. Per Tastendruck setzt er die LEDs in den Ausgangszustand zurück, in dem die erste LED leuchtet. Für meinen Geschmack verkompliziert der Reset das Design unnötig, ohne irgendeinen Mehrwert zu bringen. Daher habe ich ihn im folgenden Design entfernt, um es leichter verständlich zu machen.

Wer schon in C/C++ programmiert hat, dem wird die C-ähnliche Syntax vertraut vorkommen. Ich möchte nun anhand dieses Beispiels den Aufbau eines Verilog-Designs erklären:

Komplexe elektronische Schaltungen bestehen aus **Modulen**, wobei jedes Modul eine spezifische Aufgabe erfüllt. Ein Design in Verilog besteht aus genau einem Top-Modul, das in komplexeren Designs zahlreiche Submodule miteinander verbinden kann. Bei einem Computer wäre also der Bus oder das Motherboard das Topmodul. RAM, Prozessor, Grafikkarte etc. wären hier durch Submodule repräsentiert. Diese enthalten natürlich ihrerseits ebenfalls Submodule. Einfache Designs wie unser Lauflicht bestehen nur aus dem Top-Modul.

In Verilog setzen sich Module aus dem Schlüsselwort `module` gefolgt von dessen Namen zusammen. Danach folgt in Klammern die Definition der Ein- und Ausgänge, die sogenannten Ports.

```

module LED (
    input sys_clk,
    output reg [5:0] led
);
    ...
endmodule
    
```

Der Eingangsparameter `sys_clk` ist implizit als „wire“ (also Draht, im deutschen Sprachgebrauch für FPGAs eher Signal) definiert. Signale sind einfache Verbindungen zwischen den Komponenten des Designs. Der Aus-

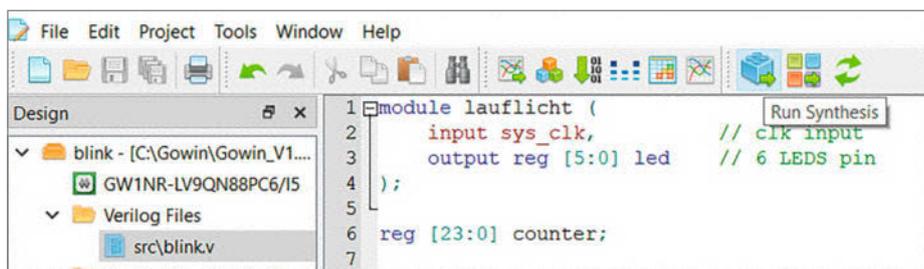


Bild 9: Aufruf der Synthese

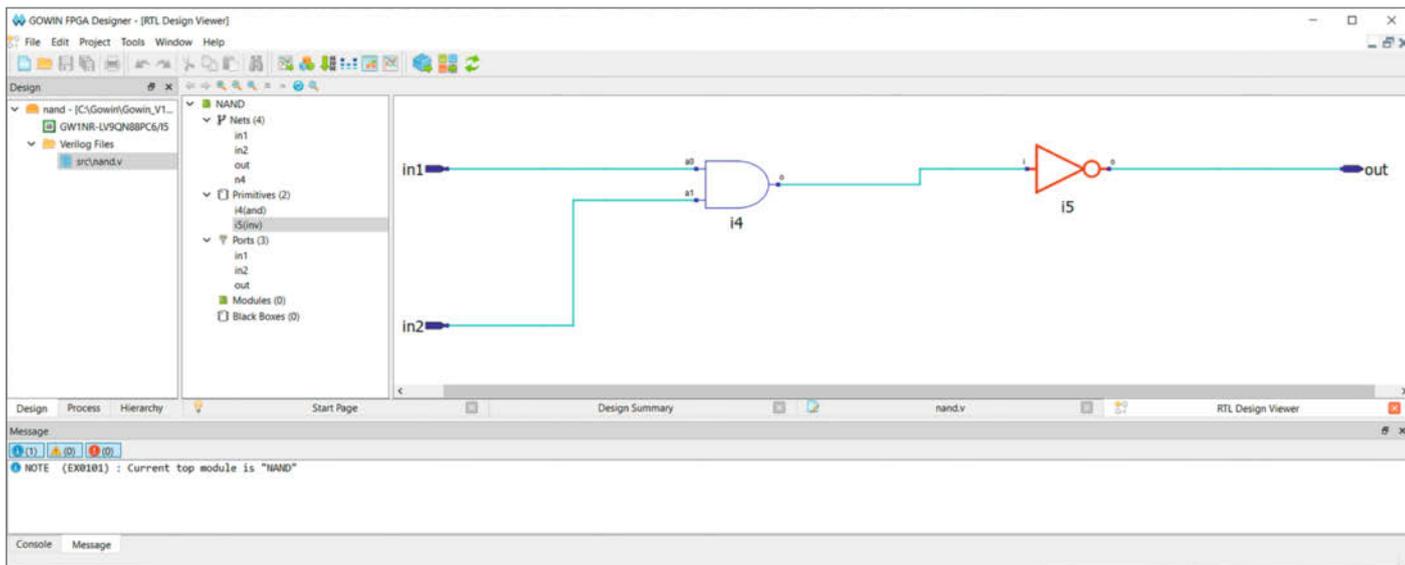


Bild 10: Netzliste eines NAND-Gatters im Design Viewer

gangsparemeter `reg[5:0] led` definiert die Variable `led` als zusammenhängende Gruppe (Vektor) von sechs Flipflops, also ein Register mit sechs Bits. Diese werden durch sechs der Flipflops am Ausgang der LUTs gebildet und später per Verilog mit den On-Board-LEDs verbunden. Danach folgen die **Blöcke**, in denen die eigentliche Funktionalität des Moduls beschrieben wird. Das Schlüsselwort `endmodule` markiert das Ende des Moduls.

Das Lauflicht-Beispiel enthält zwei Arten von Blöcken:

Initial-Blöcke dienen zur Initialisierung von Registern. Der Ausdruck `led <= 6'b111110` bedeutet, dass das Register `led` mit den nachfolgenden sechs Bits beschrieben werden soll. Im Beispiel initialisieren wir also die Register 5 bis 1 mit einer binären Eins und das Register 0 mit null.

Hierbei ist zu beachten, dass die LEDs des „Tang Nano“-Boards invertiert arbeiten: Wenn der Ausgang auf 1 gesetzt wird, ist die LED aus. Bei 0 leuchtet sie. Wenn das Register mit `6'b111110` initialisiert wird, leuchtet also die erste LED.

Der Inhalt von **always-@-Blöcken** wird immer dann ausgeführt, wenn die Bedingung erfüllt ist. Die Zeile `always @(posedge sys_clk)` wird also immer bei einer steigenden Flanke des Taktsignals relevant. Der Begriff `negedge` bezeichnet dementsprechend die fallende Flanke. Wenn ein Block mehr als eine Zeile enthält, wird er mit `begin` und `end` umschlossen.

Im unten dargestellten `always-@-Block` wird der Wert des Registers `counter` bei jeder positiven Flanke des 27-MHz-Taktsignals um eins erhöht, solange der Wert des Zählers kleiner als die Hälfte von 27.000.000 (also 13.500.000) ist. Wenn er diesen Wert nach einer

halben Sekunde erreicht hat, wird der Vektor wieder auf null gesetzt und das Spiel beginnt von vorn. Der Block erfüllt somit die Aufgabe eines Frequenzteilers.

```
always @(posedge sys_clk) begin
    if (counter < 24'd13_500_000)
        counter <= counter + 1'b1;
    else
        counter <= 24'd0;
end
```

Der zweite `always @-`-Block schiebt in der Anweisung `led[5:0] <= {led[4:0], led[5]}` den Zustand der LEDs alle halbe Sekunde (wenn der Zähler den Wert 13.500.000 erreicht hat) um ein Bit nach links. Damit wird der Lauflicht-

effekt erzeugt. Dies implementiert praktisch ein Schieberegister.

```
always @(posedge sys_clk) begin
    if (counter == 24'd13_500_000)
        led[5:0] <= {led[4:0], 1'b1};
end
```

Jetzt müssen wir das Design noch übersetzen und auf das „Tang Nano“ laden.

Synthese, Placing und Routing

Wie ein Computerprogramm muss auch ein FPGA-Design in eine für Hardware verständ-

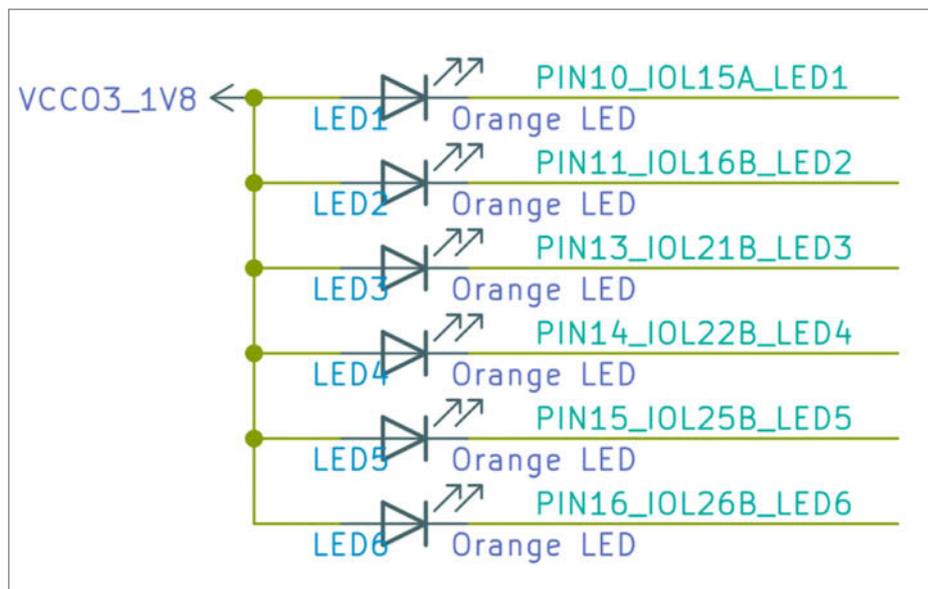


Bild 11: Verschaltung der LEDs des „Tang Nano 9k“

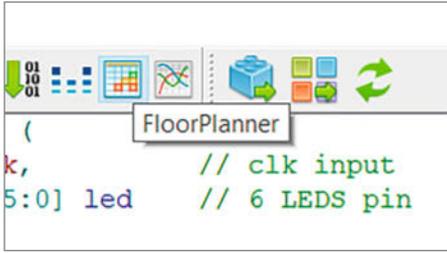


Bild 13: Der „FloorPlanner“

liche Form umgewandelt werden. Die dazu nötigen Schritte heißen Synthese, Placing und Routing.

Bei der **Synthese** wird das Design in eine Netzliste umgewandelt. Diese beschreibt die Schaltung in Form von logischen Grundelementen (wie AND, OR und XOR), Flipflops sowie deren Verbindungen. Die Synthese starten wir mit einem Klick auf das Baustein-Symbol (Bild 9) in der Toolbar der Gowin EDA.

Im Menü können wir dann unter „Tools/ Schematic Viewer“ den „RTL Design Viewer“ (Register Transfer Level, Bild 10) öffnen. Dieser stellt die Netzliste als Schaltplan dar. Im Fenster links davon finden wir die „Nets“ (Verbindungen), die „Primitives“ (logische Grundbausteine, z. B. UND, OR oder Inverter) und die „Ports“ (Ein- und Ausgänge) unserer Schaltung. Doppelklicken auf eines der Elemente hebt es im Schaltplan in Rot hervor.

Das **Placing** ist der nächste Schritt, nachdem die Synthese eine Netzliste erstellt hat. Dabei werden die Primitives, die durch die Netzliste definiert wurden, den physischen Ressourcen des FPGAs (LUTs, Flipflops, IO-Blöcke etc.) zugeordnet und darin implementiert.

Beim **Routing**, dem letzten Schritt, werden die Verbindungen zwischen den platzierten logischen Blöcken hergestellt. Diese Blöcke sind über ein Netzwerk programmierbarer Verbindungen (Configurable Routing Units, CRUs) miteinander verbunden. Der Routing-Prozess konfiguriert also die CRU. Als Resultat

erhalten wir eine Bitstream-Datei, mit der das FPGA dann konfiguriert werden kann.

I/O-Constraints

Bevor wir die Bitstream-Datei für unser Lauflicht-Design erzeugen können, müssen wir aber zuerst die physikalischen Pins des FPGAs mit den entsprechenden Ports (Ein- und Ausgänge) des Designs verbinden. Dies wird in den „I/O Constraints“ festgelegt. Werfen wir dazu einen Blick in den Schaltplan des Boards.

Die LEDs (Bild 11) sind anodenseitig mit +1,8V verbunden und kathodenseitig mit den Pins 10, 11, 13, 14, 15 und 16. Daher sind sie „active low“, d. h., sie leuchten, wenn der Ausgang passiv ist (0 Volt), und erlöschen, wenn er aktiv ist (1,8 Volt). Daher kommt auch ihr inverses Verhalten, das schon angesprochen wurde.

Von Pin 52 (Bild 12) erhalten wir den Systemtakt `sys_clk`, aus dem die halbe Sekunde abgeleitet wird, nach deren Ablauf die aktive LED um eine Stelle nach links verschoben wird (siehe



Bild 12: Der 27-MHz-Systemtakt

Port	Direction	Diff Pair	Location	Bank	Exclusive	IO Type	Drive	Pull Mode	PCI Clamp	Hystere
1 led[0]	output		10	3	False	LVC MOS18	8	UP	N/A	N/A
2 led[1]	output		11	3	False	LVC MOS18	8	UP	N/A	N/A
3 led[2]	output		13	3	False	LVC MOS18	8	UP	N/A	N/A
4 led[3]	output		14	3	False	LVC MOS18	8	UP	N/A	N/A
5 led[4]	output		15	3	False	LVC MOS18	8	UP	N/A	N/A
6 led[5]	output		16	3	False	LVC MOS18	8	UP	N/A	N/A
7 sys_clk	input		52	1	False	LVC MOS18	N/A	UP	ON	NONE

Bild 14: Konfiguration der I/O-Constraints im „FloorPlanner“

Abschnitt zu den always-@-Blöcken). Dieser wird von einem 27-MHz-Quarzoszillator gespeist, wir müssen also 13.500.000 Takte abwarten.

Die Constraints geben wir im „FloorPlanner“ ein. Diesen öffnen wir in der Gowin EDA mit einem Klick auf die Schaltfläche (Bild 13) mit dem Tabellen-Symbol.

Im FloorPlanner (Bild 14) tragen wir dann die Pins in die Spalte Location im Tab „I/O Constraints“ ein.

Nach dem Speichern erhalten wir eine Textdatei mit der Endung .CST, die Constraints-Datei (Bild 15).

Der FloorPlanner kann auf Wunsch und zur Kontrolle bzw. Information die belegten Logikblöcke und I/Os anzeigen. Dazu müssen wir mit der rechten Maustaste auf die schwarze Fläche im Fenster „Chip Array“ klicken und im Kontextmenü „Show Multi-View“ (Bild 16) auswählen. Nun können wir das „Place & Route“ starten, indem wir aus der Oberfläche des GOWIN FPGA Designer auf die Schaltfläche mit den vier Quadraten klicken. Wenn alles fehlerfrei durchgelaufen ist, erhalten wir die Bitstream-Datei, die wir jetzt nur noch auf das FPGA-Board laden müssen.

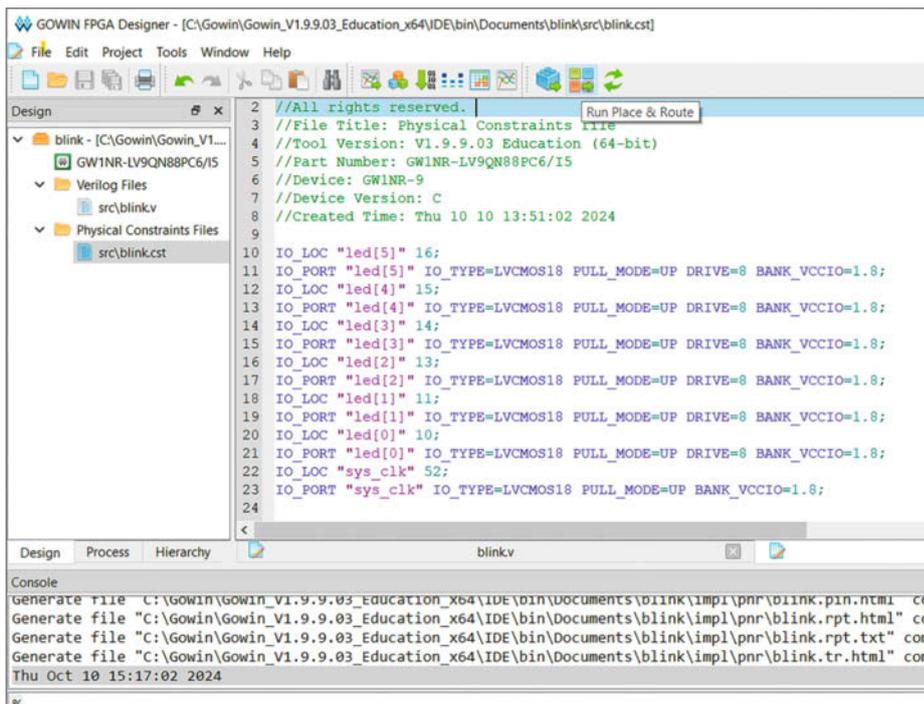


Bild 15: Die Constraints-Datei

betterCode()

GenAI 2025

Die Konferenz zu KI-gestützter Softwareentwicklung

21. Januar 2025 · Online

Mehr als Boilerplate-Code mit Copilot

Das zeigt die betterCode() GenAI am 21. Januar 2025:

- ✔ **Status quo und Zukunft** der KI-gestützten Entwicklung
- ✔ Die **leistungsfähigsten KI-Tools** im Überblick
- ✔ Schnelles Feedback durch **Code Reviews** mit GenAI
- ✔ **Bessere Testergebnisse** durch Einsatz von KI
- ✔ Altsysteme modernisieren: **LLMs unterstützen Migration**
- ✔ **Security:** Risiken und Chancen von KI

Jetzt
Frühbucher-
tickets
sichern!

Workshop am 28. Januar

genai.bettercode.eu

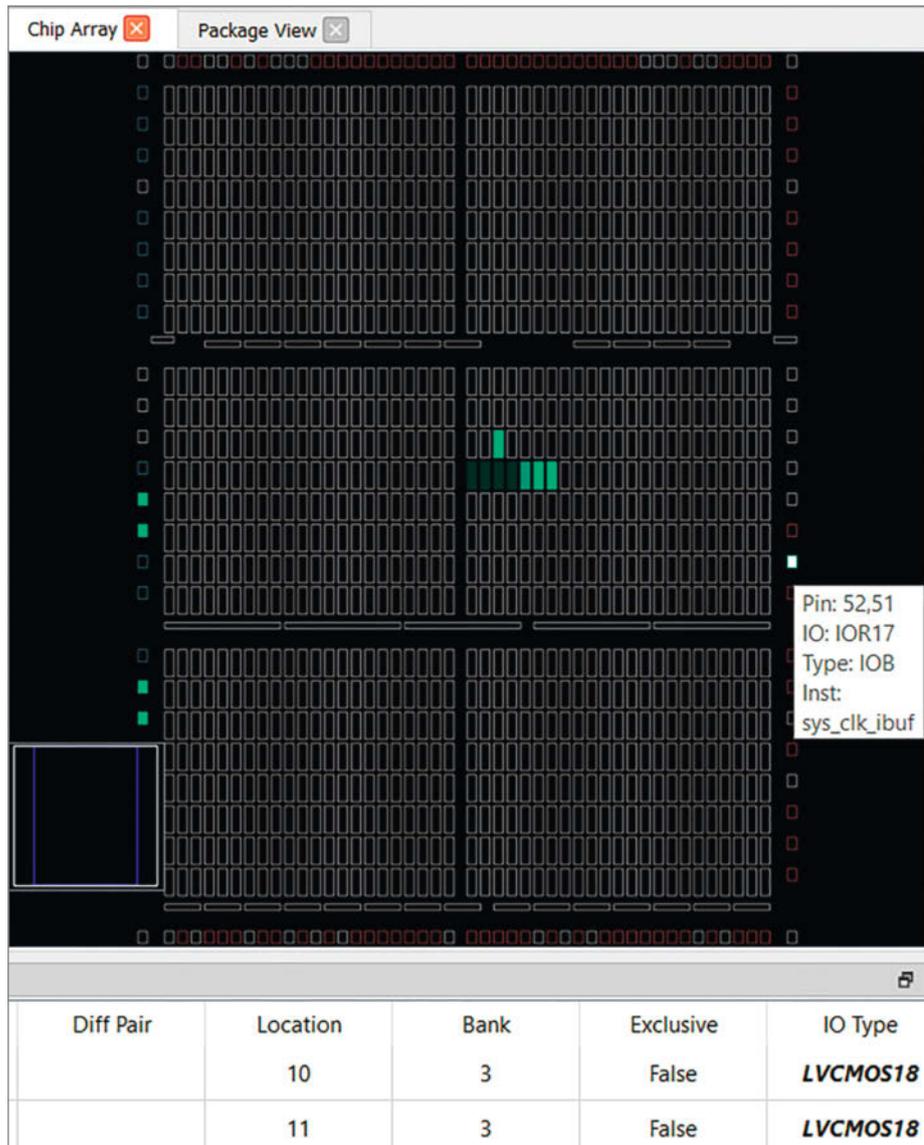


Bild 16: Logikblöcke (CLB) und I/Os des Lauflichts

Laden des Bitstreams

Zum Laden des Bitstreams auf das FPGA über USB nutzen wir die Software „Gowin Programmer“ (Bild 17), die wir mit einem Klick auf das Download-Symbol in der Toolbar der Gowin EDA starten.

Per Voreinstellung wird der Bitstream in das Shadow-SRAM (im Programmer nur „SRAM“ genannt, in der Dokumentation aber mit „SSRAM“ abgekürzt) des FPGA geladen. Das Shadow-SRAM ist ein statisches RAM, das die Konfiguration der LUTs, Register, Flipflops und On-Chip-Komponenten enthält. Da es sich um flüchtigen Speicher handelt, wird die Konfiguration nach dem Ausschalten der Versorgungsspannung wieder „vergessen“ und muss jedes Mal neu geladen werden. Das Laden des Bitstreams in das SSRAM ist damit eine gute Option, wenn wir Designs entwickeln und testen wollen.

Wenn wir sie permanent machen möchten, müssen wir den Bitstream daher in den internen Flash des FPGAs schreiben. Dazu doppelklicken wir auf „SRAM Program“ in der Spalte „Operation“. Im Dialog „Device Configuration“ setzen (Bild 18) wir dann „Access Mode: Embedded Flash Mode“ und „Operation: embFlash Erase, Program“.

Nach dem erneuten Laden befindet sich die Konfiguration im internen Flash des FPGAs und wird bei jedem Neustart von dort in das deutlich schnellere SSRAM kopiert.

Dieser Artikel konnte natürlich nur einen ersten Einstieg in das Thema FPGA geben. Wer aber tiefer in die Materie einsteigen möchte, findet im Internet zahlreiche Foren, Tutorials, Videos und Beispiele, unter dem Kurzlink haben wir eine ganze Reihe gesammelt. —caw

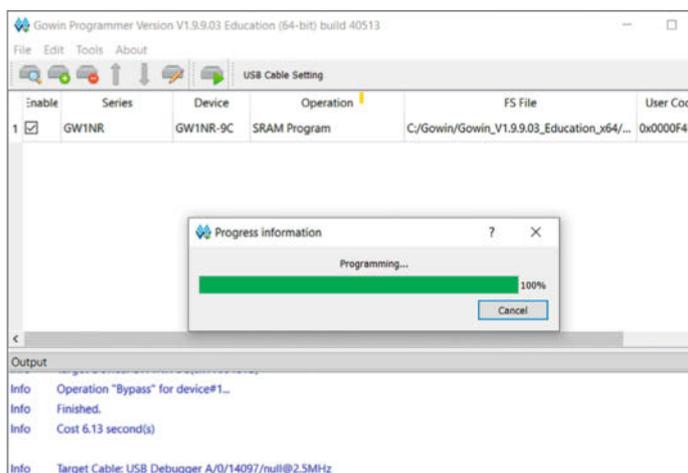


Bild 17: Der Gowin Programmer

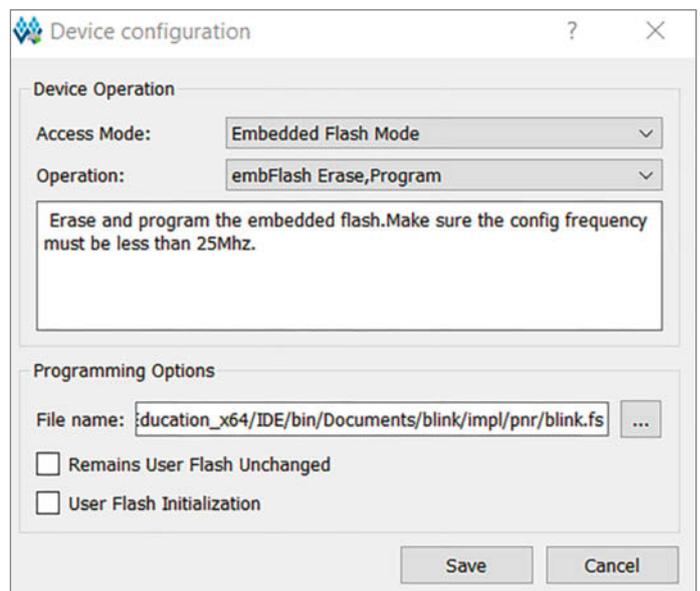


Bild 18: Flash-Konfiguration im Programmer

Wärmepumpentechnik für Einsteiger



Technik verstehen
Machbarkeit prüfen
Angebote beurteilen



Jetzt informieren:

webinare.heise.de/waermepumpen

6. März 2025

Photovoltaik für Einsteiger

Grundlagen verstehen
Angebote beurteilen
Selber bauen



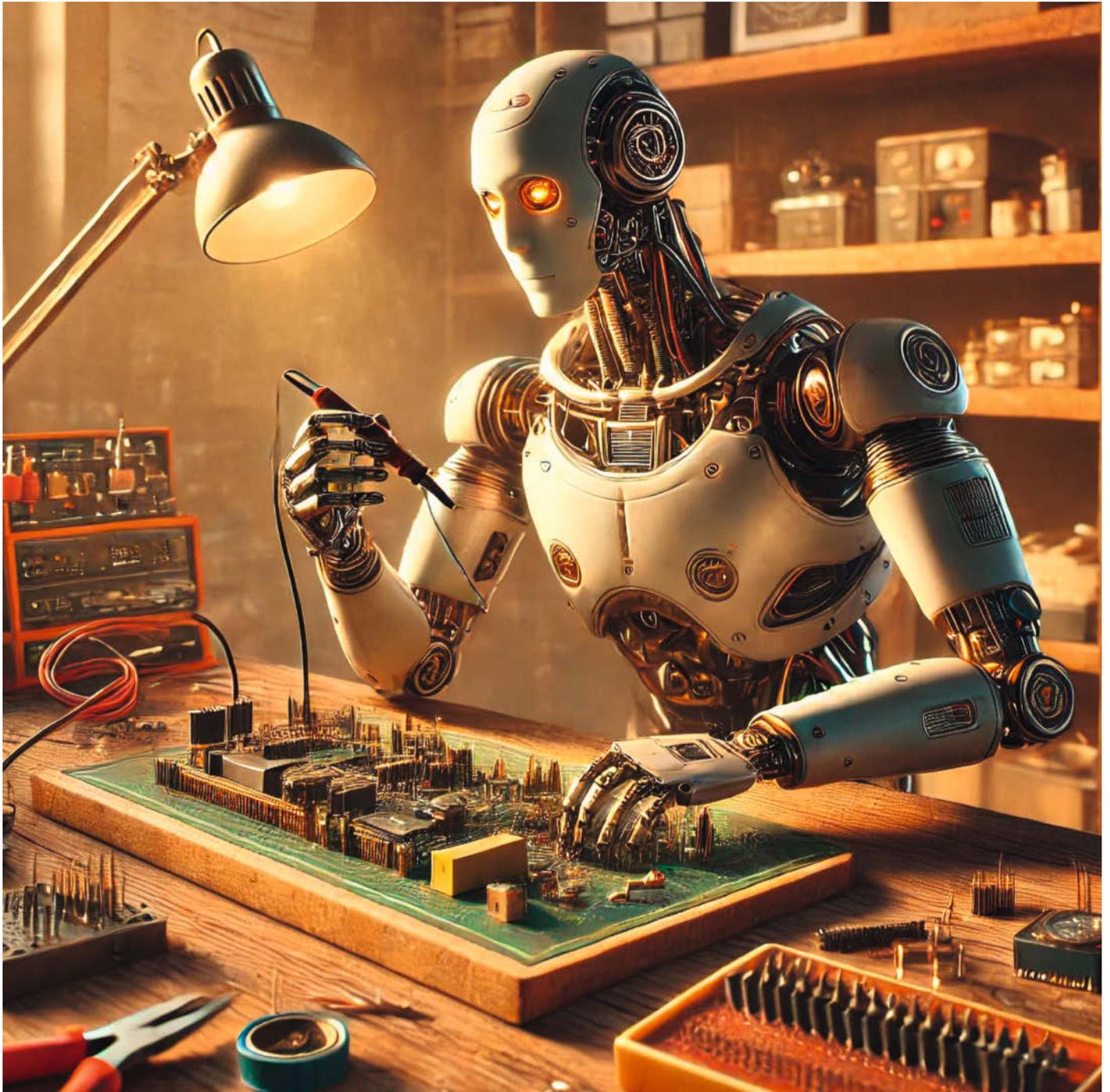
Jetzt Tickets sichern:
webinare.heise.de/photovoltaik



Programmieren lernen mit ChatGPT-o1

Das neue ChatGPT-Sprachmodell o1 hilft beim Programmieren in C++, Python, HTML und Co. und ermöglicht Makern Projekte, die vorher unmöglich erschienen. Wir haben das Modell ausprobiert und zeigen, wie schnell sich mithilfe des KI-Assistenten Programmieren lernen und Code generieren lässt.

von Johannes Börnsen



Im Editorial von Heft 5/24 habe ich bereits von meiner Faszination für das Programmieren mit KI-Hilfe berichtet. Mit der Einführung des neuen Sprachmodells o1-preview haben sich die Möglichkeiten, mittels ChatGPT Programmcode komplett generieren zu lassen, sowie Programmieren zu lernen, noch einmal deutlich erweitert. Mit dem ebenfalls neuen Canvas-Modus enthält ChatGPT nun außerdem einen speziellen Editor zum Programmieren. Perfekt für Maker wie mich, die sich mit den Syntaxen der gängigen Sprachen eher schwertun, bei ihren Projekten auf Arduino, ESP und Co. aber trotzdem nicht verzichten wollen.

Unser Autor Benno Lottenbach beschreibt übrigens im Artikel „ChatGPT als Bastelkollege“ (auf Seite 8) seine Erfahrungen mit dem Programmieren mit ChatGPT, unter anderem beim Bauen des in Heft 4/24 erschienenen Sandmaltisches, und gibt einen detaillierten Einblick in den jeweiligen Gesprächsfaden mit der KI. Dieser Artikel beleuchtet dagegen die Neuerungen beim Sprachmodell o1-preview und den Canvas-Modus.

Denken hilft!

Das Mitte September veröffentlichte Sprachmodell o1-preview wirkt erst einmal vergleichsweise langsam. Stellt man eine komplexe Frage, kann es über eine halbe Minute bis zu einer Antwort dauern. Diese Zeit nutzt das Modell aber zum „Denken“. Es zerlegt die gestellte Frage in einsehbare Einzelteile, erkennt Zusammenhänge und verknüpft Informationen, um zu einer möglichst präzisen und durchdachten Antwort zu kommen.

Natürlich kann eine KI nicht wie ein Mensch denken, der Begriff beschreibt den Prozess aber, wie ich finde, recht treffend. Bereits während der Wartezeit lässt sich der Vorgang mitlesen. Am Anfang steht das Verstehen der Frage, später werden Details bearbeitet und mögliche Lösungswege abgewogen. Dies merkt man vor allem durch bessere Antworten bei komplexen Aufgaben wie dem Schreiben von Programmcode oder dem Lösen von mathematischen Problemen. Das Modell ist seinen Vorgängern überlegen, und der einsehbare Denkprozess hilft, Fehler im Eingabeprompt zu finden (Bild 1).

Aktuell ist das o1-preview-Modell nur für zahlende ChatGPT-Kunden verfügbar, im Vergleich zu den anderen Modellen aber dennoch recht beschränkt: Es kann keine Bilder generieren oder entgegennehmen, nicht aufs Internet zugreifen und mit lediglich 50 Anfragen pro Woche bei o1-preview und 50 Anfragen pro Tag für das abgespeckte, aber schnellere o1-mini ist der Spaß recht schnell vorbei. Dennoch lohnt ein Blick auf das Modell, da der Code weniger Fehler enthält, als der von GPT-4

Kurzinfo

- » Das neue Sprachmodell o1-preview und Canvas im Überblick
- » Code für Mikrocontroller und Webanwendungen generieren
- » Die KI als Programmier-Lern-Coach nutzen

Checkliste



Zeitaufwand:
ab zwei Stunden



Kosten:
20 Euro pro Monat

Mehr zum Thema

- » Ákos Fodor, Mikrocontroller-Projekte simulieren mit Wokwi, Make 3/23, S. 88
- » Dirk Wahl, Das KI-Orakel, Make 4/24, S. 48
- » Video: ChatGPTs neuer Coding-Modus ist das perfekte Maker-Werkzeug!

Material

- » ChatGPT-Pro-Account

Alles zum Artikel im Web unter make-magazin.de/xfpr



Ich möchte ein pneumatisches Blast Gate steuern, indem ich mit einem Arduino Nano einen ACS712-Sensor auslesen. Ich möchte einen Schwellwert definieren können. Ist dieser überschritten, wird die an den ACS712 angeschlossene Maschine als eingeschaltet gewertet. Ist die Maschine eingeschaltet, wird ein ebenfalls an den Arduino Nano angeschlossenes Relais eingeschaltet. Wird der Schwellwert unterschritten, gilt die Maschine als ausgeschaltet. Ist die Maschine ausgeschaltet, bleibt das Relais noch für eine im Code definierbare Zeit eingeschaltet, default sollen 10 Sekunden sein. Danach schaltet das Relais wieder aus.

Gedanken zu 14 Sekunden ^

Steuerung des mechanischen Systems
Die Anleitung beschreibt die Steuerung eines pneumatischen Blast Gats über den Arduino Nano, wobei der ACS712-Sensor zur Erkennung eines definierten Schwellwerts eingesetzt wird. Überschreitet der Schwellwert, schaltet das Relais, bleibt jedoch für 10 Sekunden aktiv, bevor es wieder deaktiviert wird.

Gerätezustände überwachen
Ich arbeite daran, den Zustand eines Geräts zu überwachen, indem ich die ACS712-Stromsensorwerte nutze, Schwellwerte setze und einen Relais-Mechanismus mit Verzögerung implementiere.

Umsetzungsmodalitäten
Ich kläre, wie man einen Arduino Nano programmiert, um einen ACS712 Sensor zu lesen, einen Schwellwertstrom zu definieren und einen Relais zu steuern, abhängig davon, ob die Maschine ein- oder ausgeschaltet ist.

Überprüfen der Richtlinien
Ich schaue mir an, ob Richtlinien das Bereitstellen von Code unterstützen. Da das der Fall ist, kann ich an der Arbeit fortfahren und den Code erklären.

Untersuchung des Codes
Ich schaue mir den Beispielcode für den Arduino Nano an, um besser zu verstehen, wie er funktioniert. Danach gehe ich auf die Erklärung des Codes ein.

Sende eine Nachricht an ChatGPT

Bild 1: Nach dem Absenden einer Aufgabe startet ChatGPT o1-preview den Denkprozess, der sich auch mitlesen lässt.

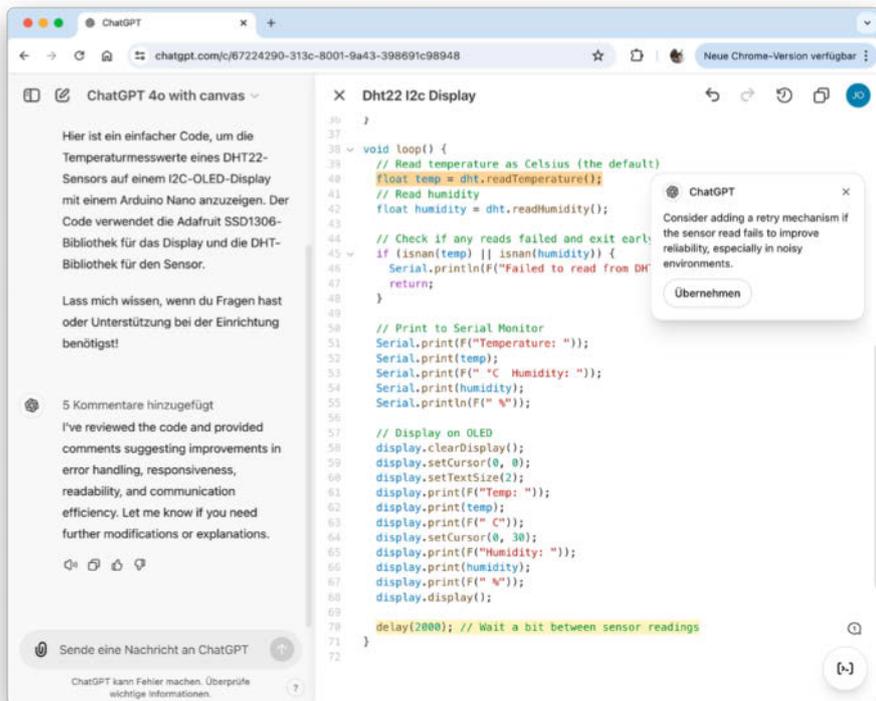


Bild 2: Der neue Canvas-Modus verschafft mehr Überblick beim Hantieren mit längeren Codes. Anmerkungen werden direkt am Code getätigt, einzelne Zeilen lassen sich markieren und getrennt vom restlichen Code bearbeiten.

und GPT-4o, und so deutlich komplexere Programmieraufgaben gelöst werden können.

Canvas statt Chat

Auch der neue Canvas-Modus ist für programmierende Maker und alle, die es werden wollen, spannend. Bestand bisher die Kommunikation mit ChatGPT aus einem klassischen Chat, teilt der Canvas-Modus nun den Bild-



Bild 3: Dieser pneumatische Sperrschieber oben links soll sich öffnen, wenn die Schleifmaschine unten rechts eingeschaltet wird.

schirm in zwei Bereiche. Der gewohnte Chat findet sich in der linken Hälfte, und auf der rechten Seite wird der zu bearbeitende Text oder Code dargestellt (Bild 2). Dies ist deutlich übersichtlicher, weil nicht bei jeder Antwort aufs Neue der komplette Code im Chat wiederholt wird. Stattdessen kann ChatGPT auf der rechten Seite im Code einzelne Zeilen ändern oder korrigieren, während der restliche Code unangetastet bleibt. Dadurch sinkt die Gefahr, dass ChatGPT beim Ändern von Code an anderer Stelle Fehler einbaut.

Außerdem erlaubt die neue Darstellung das Markieren von einzelnen Code- oder Textzeilen. Man kann dann zur markierten Stelle eine Rückfrage stellen oder eine Änderung in Auftrag geben. Besonders praktisch ist dies, wenn man einen Code aus dem Netz verstehen oder anpassen möchte, um ihn für ein eigenes Projekt nutzen zu können. Funktionen wie das Übersetzen in andere Programmiersprachen und das Erstellen von Kommentaren waren auch bisher schon möglich, befinden sich nun aber in einem neuen Menü in der unteren rechten Ecke. Hier gibt es auch einen Button zum Überprüfen von Code. Findet ChatGPT einen Fehler oder einen Verbesserungsvorschlag, macht es an die entsprechende Stelle im Code eine Notiz. Die Fehlerbehebung oder der Vorschlag können dann mit einem Klick innerhalb der Notiz in den Code übernehmen. Auch Canvas ist aktuell nur für zahlende Kunden zugänglich, soll nach Ende der Beta-Phase aber kostenlos verfügbar werden.

Um zu zeigen, wie man mit dem neuen Sprachmodell und Canvas programmieren kann, habe ich zwei Projekte umgesetzt, die schon länger auf meiner Werkbank lagen. Beide Ergebnisse können im Browser ausprobiert werden.

Mikrocontroller-Programmierung via KI

Zunächst habe ich mich an einer, auch schon im Editorial erwähnten, Steuerung von Absperrschiebern für meine Holzbearbeitungsmaschinen versucht. Dabei bestand folgende Aufgabenstellung: Tischkreissäge, Bandsäge und Co. produzieren Holzstaub. Um diesen nicht einzuatmen, möchte ich eine zentrale Absauganlage haben, an die alle stationären Holzbearbeitungsmaschinen permanent mit Rohren angeschlossen sind. Da die Absaugung aber nicht genug Leistung hat, um alle Maschinen gleichzeitig abzusaugen, sollen die Rohre vor den Maschinen mit Sperrschiebern (Bild 3) verschlossen und immer bei der Maschine pneumatisch geöffnet werden, die ich einschalte. Da die Absauganlage nach dem Ausschalten der Maschine noch einige Sekunden nachläuft, soll während dieser Zeit auch der Schieber geöffnet bleiben und anschließend, nach rund zehn Sekunden, wieder schließen.

Umsetzen möchte ich das Projekt mit einem Arduino Nano, der mittels eines ACS712-Stromsensors den Zustand der Maschine erkennen soll. Läuft die Maschine, soll der Arduino ein Relais einschalten, das 24V zu einem Magnetventil schaltet, welches den Pneumatikzylinder am Sperrschieber ein-beziehungsweise ausfahren lässt und so den Schieber öffnet oder schließt.

Sicher gibt es auch Lösungsmöglichkeiten ohne Mikrocontroller, die Arduino-Lösung erschien mir aber leichter anpassbar, und manchmal bastele ich auch einfach um des Bastelns willen (Bild 4).

Da ich dieses Projekt noch vor dem Erscheinen des Canvas-Modus umgesetzt habe, kam der klassische Chat-Modus zum Einsatz. In meinem initialen Arbeitsauftrag für ChatGPT hatte ich alle vorab angestellten Überlegungen zusammengefasst.

Ich möchte ein pneumatisches Blast Gate steuern, indem ich mit einem Arduino Nano einen ACS712-Sensor auslese. Ich möchte einen Schwellenwert definieren können. Ist dieser überschritten, wird die an den ACS712 angeschlossene Maschine als eingeschaltet gewertet. Ist die Maschine eingeschaltet, wird ein ebenfalls an den Arduino Nano angeschlossenes Relais eingeschaltet. Wird der Schwellenwert unterschritten, gilt die Maschine als



Bild 4: Dazu ist die Schleifmaschine an diese Steuerung angeschlossen. Läuft die Maschine, wird das von einem Arduino Nano erkannt und das Magnetventil des Sperrschiebers betätigt, sodass sich der Sperrschieber öffnet.

ausgeschaltet. Ist die Maschine ausgeschaltet, bleibt das Relais noch für eine im Code definierbare Zeit eingeschaltet, default sollen 10 Sekunden sein. Danach schaltet das Relais wieder aus.

Ich nannte der KI den gewünschten Mikrocontroller und Stromsensor und erklärte grob, welcher Logik das Programm folgen soll. ChatGPT antwortete mit einem passenden, in C++ geschriebenen Code (Bild 5 zeigt einen Auszug), der auch bereits Kommentare und Konsolenausgaben enthielt. Außerdem gab ChatGPT mir Hinweise, an welche Pins ich den Sensor und das Relais zu verkabeln habe. Auch zum Code waren einige Hinweise enthalten: Für das Auslesen des Sensors wurde ein Schwellwert definiert, ab dem die Maschine als „eingeschaltet“ gewertet wird. Je nach Maschine muss dieser Wert gegebenenfalls angepasst werden – wo, erklärte ChatGPT ungefragt mit. Auch Hinweise zum Einstellen der Verzögerungszeit waren enthalten, nebst der Information, dass der von mir vorgeschlagene Sensor in unterschiedlichen Versionen (5A, 20A und 30A) erhältlich ist (Bild 6).

Um das Ergebnis schnell zu testen, ohne umständlich bei jeder Änderung den Code über die Arduino IDE kompilieren und übertragen zu müssen, habe ich das Projekt, soweit möglich, im Online-Simulator Wokwi aufgebaut. Ein Arduino Nano, ein Relais und ein Schalter statt des dort nicht enthaltenen Stromsensors sind schnell zusammengekllickt, und viele Fehler fallen auch dort bereits auf.

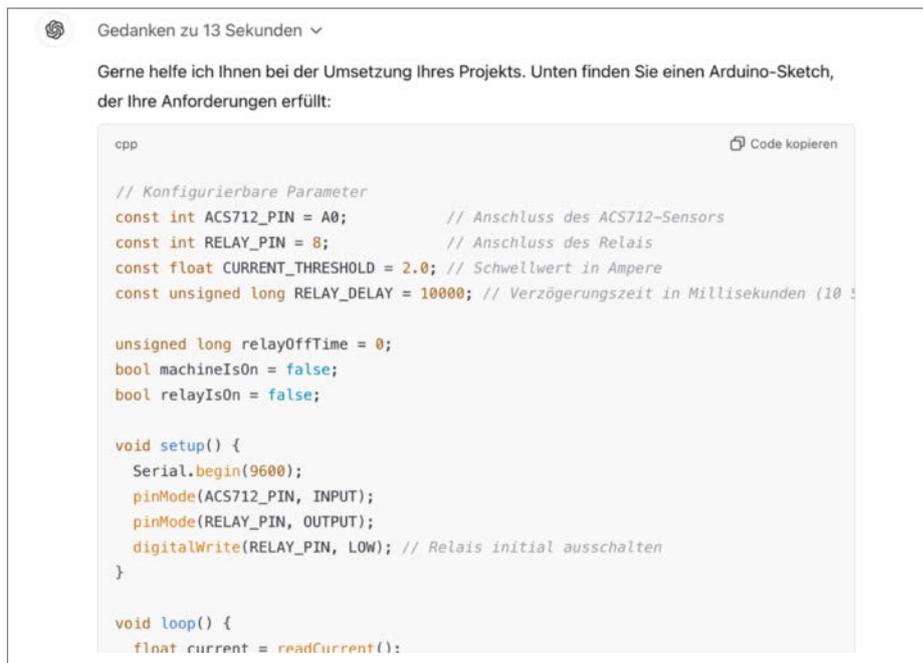


Bild 5: ChatGPT antwortete mit einem kompletten Code ...

Im weiteren Chatverlauf habe ich der KI die in Wokwi auftretenden Fehler nach und nach beschrieben und von ihr korrigieren lassen. Außerdem lassen sich natürlich auch Verständnisfragen zum Code stellen. Es würde wenig Sinn ergeben, hier den kompletten Chatver-

lauf abzdrukken, daher verlinke ich ihn stattdessen hinter der URL in der Kurzinfo.

Voller Euphorie für dieses Projekt, das für mich ohne KI-Hilfe nicht annähernd in der gleichen Zeit umsetzbar gewesen wäre, habe ich noch eine deutlich erweiterte Funktion der

Anmerkungen:

- **Anschluss des ACS712-Sensors:** Schließen Sie den Ausgang des ACS712 an den analogen Eingang A0 des Arduino Nano an. Achten Sie darauf, die Stromversorgung und Erdung korrekt zu verbinden.
- **Anschluss des Relais:** Verbinden Sie das Steuerpin des Relais mit Digitalpin 8 des Arduino Nano. Stellen Sie sicher, dass das Relais mit der richtigen Spannung betrieben wird und dass die Strombelastbarkeit des Arduino-Pins nicht überschritten wird (verwenden Sie ggf. ein Relaismodul mit Optokoppler).
- **Anpassung des Schwellwerts:** Der Schwellwert `CURRENT_THRESHOLD` ist auf 2.0 Ampere eingestellt. Passen Sie diesen Wert an Ihre Anforderungen an.
- **Verzögerungszeit:** Die Verzögerungszeit `RELAY_DELAY` ist standardmäßig auf 10.000 Millisekunden (10 Sekunden) eingestellt. Sie können diesen Wert im Code ändern.
- **Kalibrierung des Sensors:** Die Formel zur Berechnung des Stroms basiert auf dem ACS712-30A-Modul. Wenn Sie ein anderes Modell verwenden (z.B. 5A oder 20A), müssen Sie den Sensitivitätswert in der Formel anpassen (z.B. 0.185 für 5A-Modul, 0.100 für 20A-Modul).

Funktionsweise des Codes:

- Der Arduino liest kontinuierlich den Stromwert vom ACS712-Sensor.
- Wenn der gemessene Strom den Schwellwert überschreitet, wird die Maschine als eingeschaltet betrachtet, und das Relais wird aktiviert.
- Wenn der Strom unter den Schwellwert fällt, wird die Maschine als ausgeschaltet betrachtet, und ein Verzögerungstimer wird gestartet.
- Das Relais bleibt aktiviert, bis die Verzögerungszeit abgelaufen ist.

Wichtig: Achten Sie bei allen elektrischen Arbeiten auf die Sicherheit und stellen Sie sicher, dass alle Komponenten korrekt angeschlossen sind.

Bild 6: ... und ergänzte das Ganze ungefragt um nützliche Anmerkungen und Hinweise.

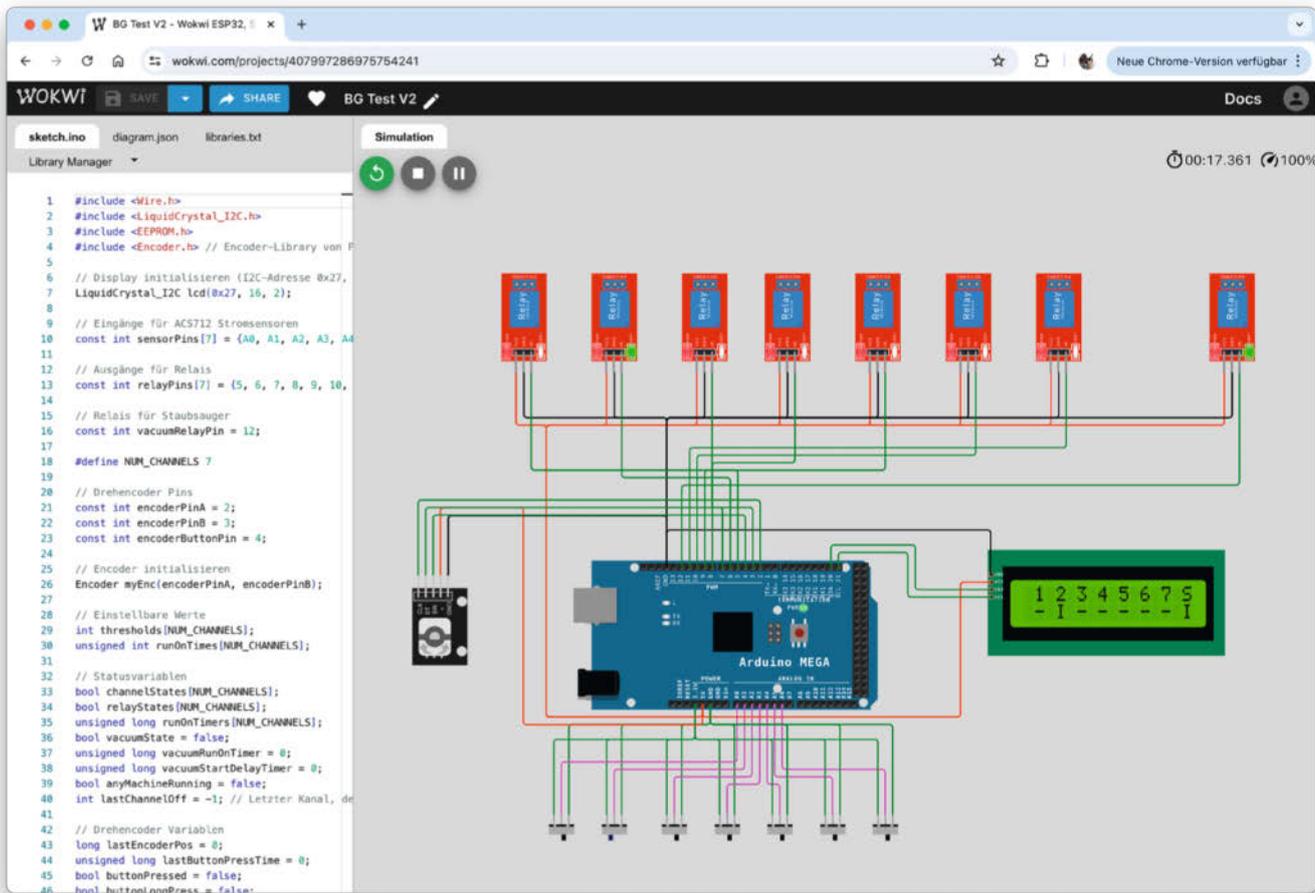


Bild 7: Eine aufgebohrte Absaug-Steuerung für die Holzwerkstatt, hier im Online-Simulator Wokwi. Auf dem Display lässt sich der aktuelle Status ablesen und man kann Menüeinstellungen vornehmen. Der Code in der linken Bildhälfte ist komplett generiert.

Steuerung programmieren lassen. Diese nutzt einen Arduino Mega, ein zweizeiliges I²C-Display, kann bis zu 7 Maschinen gleichzeitig überwachen, einen Staubsauger ein- und ausschalten, und bietet ein Menü, über das mit einem Drehencoder auf dem Display der Schwellwert und die Nachlaufzeit verstellbar und im EEPROM gespeichert werden können. Dieses Projekt habe ich im Online-Simulator Wokwi aufgebaut und in der Kurzinformatik verlinkt (Bild 7).

Webentwicklung via KI

Neben der Entwicklung für Mikrocontroller ist auch das Erstellen von Webprojekten mit ChatGPT möglich. Als Beispiel habe ich einen Stirnholzschneidbrett-Designer programmieren lassen. Um solche Schneidbretter zu bauen, wird eine Holzbohle in Streifen gesägt, umsortiert, wieder verleimt, um 90 Grad gedreht, wieder aufgesägt, um 90 Grad gedreht und verleimt. Wenn ich sie jetzt verwirrt habe: Willkommen im Club! Bis ich mein Hirn dazu gebracht hatte, diesen Prozess beim Betrachten einer Bohle im Kopf durchzugehen und mir das fertige Schneidbrett vorzustellen, hat es einige Fehlversuche gebraucht.

Hier sollte mir eine Designer-Webapp in Zukunft zur Seite stehen. Mit ihrer Hilfe könnte

ich die Maße des fertigen Schneidbretts sowie die einzelnen Elemente, aus denen es verleimt werden muss, im Browser konfigurieren und als Vorschau dreidimensional visualisieren lassen (Bild 8).

Typischerweise bestehen kleine Webanwendungen wie diese aus drei Teilen: Eine HTML-Seite legt Elemente wie Überschriften, Texte, Slider und Checkboxen an. Eine zusätzliche CSS-Datei legt fest, wie diese Elemente aussehen sollen, und eine JS-Datei enthält JavaScript, das die Elemente mit Funktionen versorgt und in diesem Beispiel auch die Visualisierung des Schneidbretts übernimmt.

Bittet man ChatGPT um das Erstellen eines solchen Designers, landen die nötigen CSS- und JS-Befehle gerne mit in der HTML-Seite. Das ist zwar prinzipiell ebenso funktionsfähig, aus Gründen der Übersichtlichkeit ist aber die oben beschriebene Aufteilung in mehrere Dateien üblich. ChatGPT kann jedoch auf Nachfrage den Code entsprechend aufteilen.

Auch dieses Projekt lässt sich im Browser ausprobieren. Der Link findet sich wieder in der Kurzinformatik. Zum Testen während des Entwickelns habe ich die Webseite codepen.io verwendet. Hier lassen sich HTML, CSS und JS in getrennte Bereiche kopieren, als wären es einzelne Dateien. Darunter lässt sich das Er-

gebnis anschauen und ausprobieren. Auch externe Skripte, die via JS aufgerufen werden, lassen sich einbinden.

Genau wie bei der Sperrschieber-Steuerung war auch hier der Ausgangspunkt ein Befehl, der die grundsätzliche Funktionsweise der Webanwendung beschreibt, gefolgt von rund 50 Iterationen, bei denen ich neue Funktionen hinzugefügt und Fehler behoben habe. Auch diesen Chatverlauf finden Sie in der Kurzinformatik-URL.

Mit entsprechender Anleitung hat es ChatGPT geschafft, nicht nur Eingabemöglichkeiten für die gewünschten Schneidbrettparameter anzulegen, sondern auch eine dreidimensionale Visualisierung des Schneidbretts zu erstellen, die sich beim Verschieben der Regler dynamisch anpasst, ohne dass eine neue Berechnung via Klick angestoßen werden müsste. Diese Vorschau lässt sich mit der Maus frei drehen, zoomen und verschieben oder mit einem automatischen Drehen versehen.

Zum Schluss habe ich noch eine Funktion zum Anlegen einer Safrinne hinzugefügt, wie sie bei solchen Schneidbrettern üblich ist. Auch diese lässt sich konfigurieren und beispielsweise näher an den Rand schieben oder in ihrer Breite verändern.

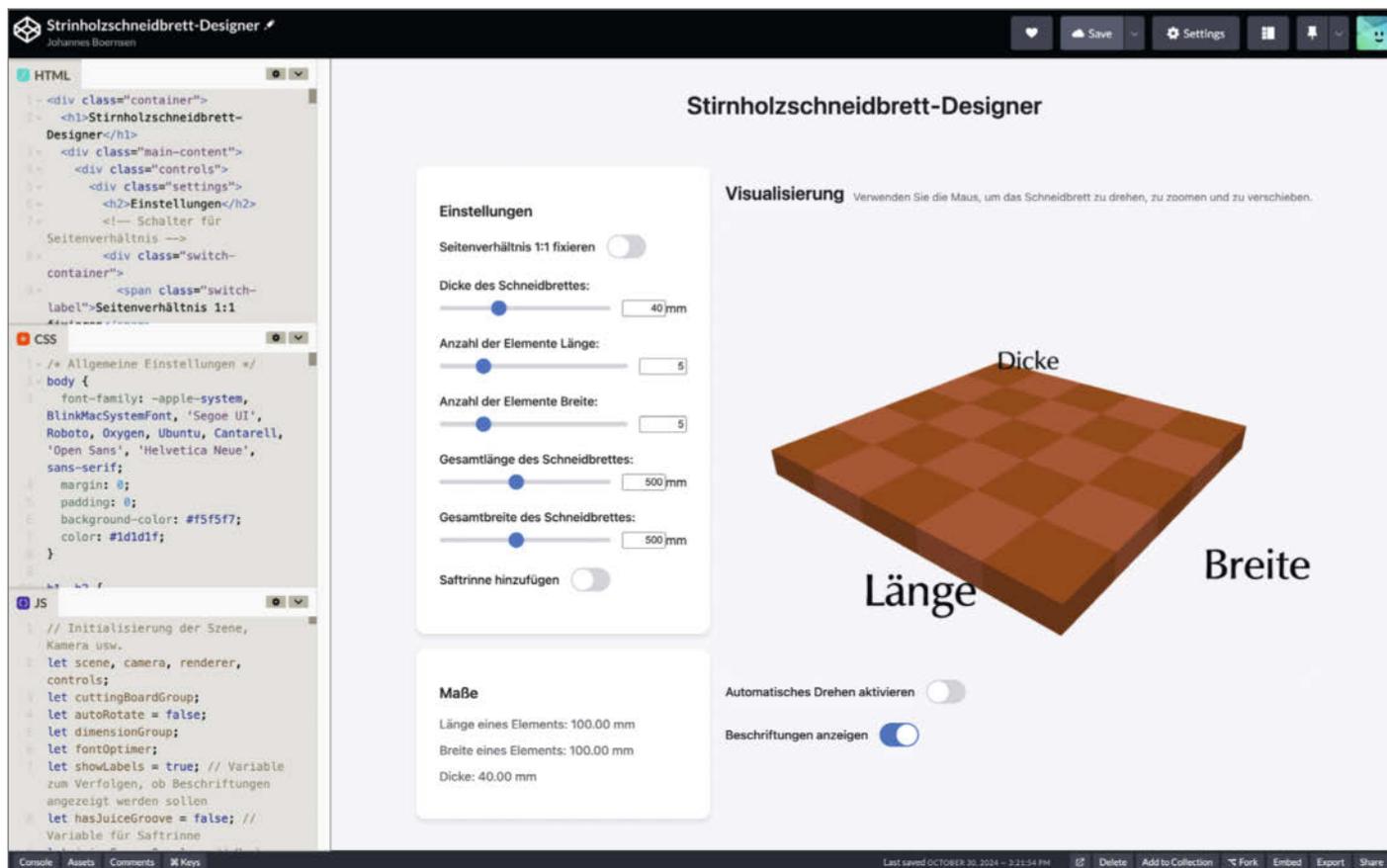


Bild 8: Für diese Webanwendung hat ChatGPT den Code nach HTML, CSS und JavaScript aufgeteilt. Die Webseite codepen.io erlaubt den schnellen Test des Ergebnisses: Ein Stirnholzschnleibrett-Designer mit frei drehbarer 3D-Visualisierung, die sich beim Verändern der Parameter automatisch in Echtzeit aktualisiert.

KI vs. Mensch

Beide Beispiele haben mir gezeigt: Die KI ist erstaunlich gut darin, Code zu schreiben, besser als ich es je war und vermutlich auch je sein werde. Sicher wird aber jemand, der in den genutzten Sprachen fit ist, Verbesserungsvorschläge haben oder in der Lage sein, schöneren oder effizienteren Code zu schreiben. Außerdem sind die ersten Versuche der KI oft doch noch reichlich anpassungsbedürftig. Beim Schneidbrett-Designer hat sich die Visualisierung beispielsweise bei jeder Veränderung der Parameter zurückgesetzt. Erst nach einer Aufforderung hat ChatGPT den Code so geändert, dass ein eingestellter Blickwinkel erhalten bleibt, wenn man beispielsweise die Schneidbrettlänge verändert. Den Programmerteil beherrscht ChatGPT also durchaus, das Erkennen von Problemen oder das Erstellen einer guten Benutzererfahrung jedoch nicht.

Den Zusammenhang zwischen den einzelnen Klötzen der Schneidbretter und deren Drehung versteht ChatGPT nicht von sich aus. Wenn man möchte, dass die KI mit diesem Wissen arbeitet, muss man ihr dies zunächst erklären. Ähnlich war es auch bei den Sperr-

schiebern. Nach meinen Erklärungen hat die KI zwar den Zweck der Steuerung in eine ebenfalls KI-generierte Projektbeschreibung für GitHub korrekt zusammengefasst, dass aber bei der Absaug-Steuerung der Sauger sich natürlich nicht ausschalten soll, wenn bereits während der Nachlaufzeit die nächste Maschine eingeschaltet wird, hat sie nicht von sich aus verstanden.

Stößt man mit dem eigenen Wissen zur Funktionsweise von Sensoren oder einzelnen Details im Code an Grenzen, bietet ChatGPT sich bereitwillig als Helfer an, der jede Frage geduldig beantwortet, sei sie auch noch so einfach oder komplex. Sicher muss man von ChatGPT als Fakt beschriebene Details noch einmal im Datenblatt des Herstellers nachsehen, um wirklich sicher gehen zu können, wie stark man beispielsweise einen Sensor belasten darf. Einen guten Anhaltspunkt für die eigene Recherche bietet die KI aber dennoch, und sei es nur, dass man den passenden Suchbegriff nicht kennt, aber beschreiben kann.

Schöner Lernen?

Meine persönlichen Versuche, endlich besser programmieren zu lernen, waren nicht von

großem Erfolg und Durchhaltevermögen geprägt. Dafür fasziniert es mich einfach nicht genug. Für mich ist Programmieren eher ein Mittel zum Zweck und ich bewundere alle, die die nötige Geduld dafür aufbringen. Die einfache Sperrschieber-Steuerung hätte ich mit viel Gesuche und Probiererei wohl auch selber hinbekommen. Allerdings nicht in der kurzen Zeit und ganz bestimmt wäre ich danach nicht direkt den Schneidbrett-Designer angegangen.

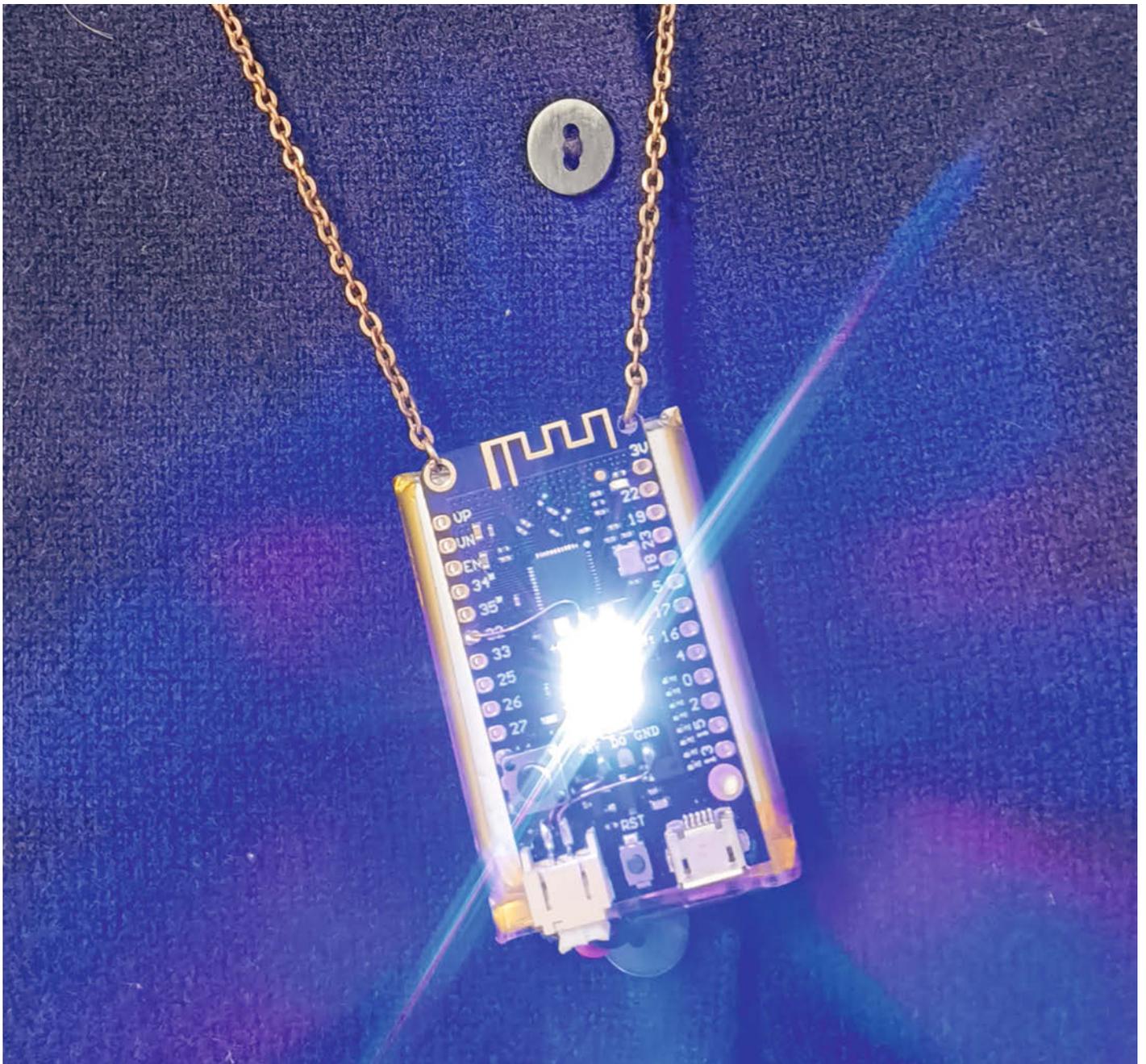
Dass ich mich jedoch auf die Programmlogik konzentrieren konnte, statt mich mit von mir geschaffenen Syntax-Fehlern beschäftigen zu müssen, war ungemein motivierend. Und ganz nebenbei habe ich bei diesen Projekten und den gestellten Rückfragen mehr übers Programmieren gelernt, als bei meinen Versuchen, es mit Tutorials oder Büchern zu lernen.

Letztlich liegt der Spaß am Maken für jeden und jede von uns in anderen Teilaspekten der Projekte. Sich von einer KI bei jenen Teilen eines Hobby-Projektes helfen zu lassen, die man weniger gut kann, für die einem die Zeit fehlt oder die einem keinen Spaß machen, finde ich daher überhaupt nicht verwerflich. —jom

Flirt-Amulette mit ESP-NOW

Auf den ersten Blick wirkt es wie ein hübsches Schmuckstück mit LEDs, die sanft pulsieren. Doch bei Annäherung eines ähnlichen Stücks leuchten beide Amulette plötzlich in bunten Farben auf. In diesem Artikel zeige ich, wie Amulette oder Armbänder, die auf ESP-Modulen basieren, über das ESP-NOW-Protokoll kommunizieren können, um z. B. auf Festivals oder Großveranstaltungen ihren Trägern spielerisch zu helfen, sich gegenseitig zu finden.

von Dr. Stefan Recksiegel



Ich zeige in diesem Artikel zunächst, wie die Kommunikation mithilfe von ESP-NOW und WifiEspNow funktioniert, und bastele dann „Flirt-Amulette“, die mit RGB-LEDs anzeigen, ob (und welche) anderen Amulette sich in der Nähe befinden. Die Amulette sind auch in der Lage, einfache Nachrichten auszutauschen, indem man einen auf die Amulette gelöteten Taster drückt.

ESP-NOW

Die Module der ESP-Serie von Espressif sind bei Makern sehr beliebt, weil sie sich leicht mit der Arduino IDE programmieren lassen. Im Gegensatz zu den Arduinos verfügen die Chips der ESP8266- und ESP32-Familien über WLAN, aber nicht allen Makern ist bekannt, dass es außer den „normalen“ Client- und Access-Point-Modi auch ein eigenes Protokoll für die Ad-hoc-Kommunikation zwischen ESP-Modulen gibt: ESP-NOW (Link in der Kurzinfor).

ESP-NOW kombiniert verschiedene Schichten des OSI-Modells (Open Systems Interconnection), einem Framework zur Strukturierung von Netzwerkkommunikation in mehreren Ebenen, zu einer einzigen Protokollebene. Dies macht es zwar schwierig, ESP-NOW außerhalb des ESP-Ökosystems zu verwenden, aber Kommunikation zwischen ESPs ist einfach aufzusetzen und verbraucht wenig Strom – gut für den mobilen Einsatz mit Batterien oder Akkus.

In der ursprünglichen Version des ESP-NOW, die in der Arduino IDE vorinstalliert ist, muss man die MAC-Adresse (also die Seriennummer des WLAN-Moduls) des Kommunikationspartners im Voraus kennen, was für viele Anwendungen nicht praktikabel ist. Junxiao Shi hat hierfür die auf ESP-NOW aufbauende Bibliothek WifiEspNow geschrieben (Link in der Kurzinfor), mit der ESP-Module in Reichweite sich gegenseitig erkennen können. Die Bibliothek kann wie üblich über den Library Manager in der Arduino IDE installiert werden.

Geeignete ESPs

Mein Code funktioniert auf allen Chips der ESP8266/8285- und der ESP32-Serie, aber die Verdrahtung der internen LEDs und der (falls vorhandenen) eingebauten Taster hängt vom Modul ab. Wenn ihr nicht im Datenblatt oder auf Schaltplänen nachschauen wollt, empfehle ich euch entweder einen ESP D1 mini, bei dem die eingebaute LED zwischen GPIO2 des ESP8266 und 3,3V hängt (leider ohne eingebauten Taster) oder besser noch einen ESP32-C3 Super Mini, der eine LED auf GPIO8 und einen Taster auf GPIO9 hat. Beide Module gibt es für unter 5 Euro pro Stück auf den bekanntesten Verkaufsplattformen zu kaufen. Ihr braucht mindestens zwei Module, es können aber ruhig verschiedene sein.

Kurzinfo

- » Tragbare Schmuckstücke basteln, die miteinander kommunizieren können
- » Kommunikation über ESP-NOW und WifiEspNow
- » Mit Neopixeln und einem Taster farbcodierte Nachrichten verschicken

Checkliste



Zeitaufwand:
3 Stunden



Kosten:
15 bis 20 Euro für zwei Amulette

Material

- » Wemos Lolin D32, ESP D1 mini, ESP32-C3 Super mini oder Ähnliche
- » WS2812(b)-LEDs
- » Akku mit PH-2-Stecker
- » Taster

Werkzeug

- » Lötkolben

Mehr zum Thema

- » Thomas Fischer, ESP Now: Datenübertragung per Funk, Make 3/22, S. 64
- » Carsten Wartmann, Disco is back – Lichtshow mit WLED, Make 2/23, S. 8
- » Marcus Hansson, Nowhere-Festival: Das Maker-Paradies in der Wüste, Make 5/24, S. 64



Alles zum Artikel
im Web unter
make-magazin.de/xyj8

Die Sketches

Ich habe drei verschiedene Sketches geschrieben, die alle über den Link in der Kurzinfor zu finden sind. Im ersten (Make_ESPnow_minimal.ino) werden die MAC-Adressen der ESPs (alle Typen der Serien ESP8266 und ESP32), auf denen der gleiche Sketch läuft, auf das Empfängermodul übertragen und durch unterschiedliche Blinkrhythmen der eingebauten LEDs angezeigt.

Ein etwas erweiterter Sketch (Make_ESPnow_button.ino) ermöglicht das Senden von Nachrichten zwischen den ESPs: Wenn ein eingebauter oder aufgelöteter Taster auf einem der ESPs gedrückt wird, wird auf den anderen ESPs eine Funktion aufgerufen, die eine serielle Meldung ausgibt und den Empfänger per LED anzeigt.

Der dritte Sketch (Make_ESPnow.ino) steuert unter Verwendung der WS2812FX-Bibliothek LEDs an. Wenn andere Module erkannt werden, dann pulsiert die LED langsam in einer Farbe, die von den letzten drei Bytes der MAC-Adresse des Partners abhängt. Wenn man schon einmal in Reichweite dieses Moduls war, kann man also eventuell die Farbe wiedererkennen. Die RGB-Werte werden aus den drei letzten Bytes der MAC-Adresse ermittelt. Im Gegensatz zu den ersten Bytes, die den Hersteller des Moduls angeben und daher oft gleich sind, laufen diese als Seriennummer über den gesamten Bereich von 0 bis 255. Das

führt mit großer Wahrscheinlichkeit dazu, dass die verwendeten Module auch Farben zeigen, die gut voneinander zu unterscheiden sind.

Wenn mehrere andere Module in Reichweite sind, wechselt die pulsierende Farbe auf dem eigenen Modul zwischen den zu diesen Modulen gehörigen. Auch hier kann man durch Drücken eines Tasters die LEDs auf den anderen Modulen für ein paar Sekunden hell aufblitzen lassen.

Die drei Sketches sind untereinander kompatibel – es können also unterschiedliche Sketches auf verschiedene Amulette aufgespielt werden, natürlich mit entsprechender Einschränkung der Funktionalität. Die fest verbauten LEDs auf den verschiedenen Modulen können beispielsweise nur einfarbiges Licht (meist blau) abgeben.

Wie die Module sich finden

Schauen wir uns jetzt den ersten und einfachsten der drei Sketches näher an (Listing 1). Er gibt den Modulen die Anweisung, per LED anzuzeigen, ob ein anderes Modul in Reichweite ist. Wenn das der Fall ist, wird die LED lang angeschaltet und kurz ausgeschaltet. Wenn kein anderes Modul erkannt wird, ist es genau andersherum: kurz ein, lang aus. Außerdem wird eine Information über gefundene Partner über die serielle Schnittstelle ausgegeben, was für Debugging-Zwecke sinnvoll ist. Wenn ihr einen ESP32-C3 nutzt, muss unter

Listing 1

```

#define STATUS_LED 8
#define LEDON LOW
#define LEDOFF HIGH

#if defined(ARDUINO_ARCH_ESP8266)
#include <ESP8266WiFi.h>
#elif defined(ARDUINO_ARCH_ESP32)
#include <WiFi.h>
#endif

#include <WifiEspNowBroadcast.h>
#define WIFI_CHANNEL 1
#define MAX_PEERS 20

void setup() {
  pinMode(STATUS_LED, OUTPUT);

  Serial.begin(115200); Serial.println();
  Serial.printf("Starting ESPnow demo, LED on GPIO%i\n", STATUS_LED);
  WiFi.persistent(false);
  bool wifok = WifiEspNowBroadcast.begin("ESPNOW", 3);
  if (!wifok) { Serial.println("WifiEspNowBroadcast.begin() failed");
    ESP.restart(); }
  Serial.print("MAC address of this node is ");
  Serial.println(WiFi.softAPmacAddress());
}

void loop() {
  WifiEspNowBroadcast.loop();
  WifiEspNowPeerInfo peers[MAX_PEERS];
  int nPeers = WifiEspNow.listPeers(peers, MAX_PEERS);
  Serial.printf("Number of peers: %i ", nPeers);
  for (int i = 0; i < nPeers; ++i) {
    Serial.printf(" %02X:%02X:%02X:%02X:%02X:%02X ", peers[i].mac[0],
      peers[i].mac[1], peers[i].mac[2], peers[i].mac[3],
      peers[i].mac[4], peers[i].mac[5]); }
  Serial.println();

  // if connected: short off, long on - if not: short on, long off
  if( nPeers>0 ) { digitalWrite(STATUS_LED, LEDOFF); delay(50);
    digitalWrite(STATUS_LED, LEDON);}
  else { digitalWrite(STATUS_LED, LEDON); delay(50);
    digitalWrite(STATUS_LED, LEDOFF);}
  delay(950);
}

```

Tools in der Arduino IDE „USB CDC On Boot“ aktiviert sein, damit die serielle Schnittstelle aktiv ist!

Bei der Initialisierung wird `WifiEspNowBroadcast` gestartet. Bei jedem Durchlauf des Loops wird die Kontrolle an `WifiEspNowBroadcast` übergeben, damit andere Module in Reichweite gefunden werden können. Die MAC-Adressen der gefundenen Module wer-

den ausgegeben und der Blinkrhythmus der internen LED wird entsprechend gewählt.

Daten versenden

Wirklich nützlich wird ESP-NOW, wenn wir das Protokoll nicht nur dafür verwenden, andere Module zu finden, sondern auch, um ihnen Daten zu schicken. Im Rahmen der Längen-

beschränkung von 250 Bytes pro Paket sind beliebige Daten möglich. Wir senden nur die kurze Zeichenkette „Button pressed!“. Den kompletten Sketch findet ihr unter dem angegebenen Link. Der entscheidende Befehl zum Senden ist

```

WifiEspNowBroadcast.send(
  reinterpret_cast<const uint8_t*>
    (message), strlen(message)+1);

```

Dieser Befehl wird ausgeführt, wenn der (eingebaute oder selbst angelötete) Taster gedrückt wird. Wer nicht sehr versiert in C ist, tut sich wahrscheinlich mit diesem Befehl etwas schwer: Vor dem Senden wird die Zeichenkette (String) als (konstante) vorzeichenlose 8-Bit-Ganzzahlen (uint8) reinterpretiert. Es wird nichts umgewandelt, die Daten liegen genauso im Speicher wie bisher, es ändert sich lediglich die Bedeutung, denn `WifiEspNowBroadcast.send` erwartet `uint8` und keine Zeichen (char). Warum ist die Länge der zu sendenden Daten um eins länger als der String selbst? Strings werden durch einen 0-Char beendet, der nicht zur Nettolänge zählt. Dieses Endezeichen wollen wir aber mit-senden, damit der Empfänger weiß, dass es sich um einen String handelt. Sonst sind es nur Rohdaten.

Daten empfangen

Die Sache mit dem Empfang ist etwas komplizierter: Wir wollen ja nicht die ganze Zeit darauf warten, ob jemand anderes vielleicht gerade Daten schickt. Daher registriert man mit

```

WifiEspNowBroadcast.onReceive(
  processRx, nullptr);

```

eine sogenannte Callback-Funktion (hat nichts mit Casting-Shows zu tun, auch wenn wir oben einen Pointer casten mussten). Diese Funktion wird aufgerufen, sobald `WifiEspNowBroadcast` ein Paket erhalten hat. Wie die Funktion selbst aussieht, seht ihr in Listing 2.

Die genauen Argumente der Funktion müssen uns hier gar nicht so sehr interessieren. Wichtig ist nur, dass in der Variablen `buffer[]` die empfangenen Daten übergeben werden, und dass wir mit `mac[]` auf die Adresse des Senders zugreifen können.

Listing 2

```

void processRx(const uint8_t mac[WIFIESPNOW_ALEN], const uint8_t* buffer, size_t count, void* arg){
  Serial.printf("Message from %02X:%02X:%02X:%02X:%02X:%02X: ", mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
  for (size_t i = 0; i < count; ++i) Serial.printf(" %i", buffer[i]); Serial.println();
  if(buffer[count-1]==0) Serial.printf("Message is a string: '%s'\n", (char*)buffer);
  if(strcmp(message, (char*)buffer) ==0 ) lastReceivedMillis = millis();
}

```

Wenn das letzte Byte 0 ist, handelt es sich wohl um einen String, den wir dann ausgeben. Wenn der String unserem „Button pressed!“ entspricht, speichern wir die augenblickliche Systemzeit in der Variable `lastReceivedMillis`, um dann z.B. die eingebaute LED aufleuchten zu lassen.

WS2812-LEDs ansteuern

Damit kommen wir dann endlich zum Fun-Part! Ich habe diese Technik bereits 2018 benutzt, um Beleuchtungen auf zwei Fahrrädern auf dem Burning-Man-Festival in Nevada zu steuern (Bild 1). Die synchronen Regenbögen und Blinkeffekte auf den nebeneinander fahrenden Rädern sahen in der dunklen Wüste wirklich spektakulär aus, auch weil sie deutlich heller waren als die Beleuchtung der zahllosen anderen Räder dort. Wir sind oft auf unsere Räder angesprochen und gefragt worden, wie das funktioniert. Ich war sehr stolz darauf, bei meinem ersten Besuch dort gleich etwas mitgebracht zu haben, was so viel Aufmerksamkeit erregt hat.

Wir beschränken uns hier mal auf kleine Anhänger, aber der Zahl der LEDs, die wir ansteuern können, ist im Wesentlichen nur durch den Stromverbrauch eine Grenze gesetzt.

Wir verwenden die WS2812FX-Bibliothek, die sich wiederum auf Adafruit's Neopixel-Bibliothek stützt (siehe Link in der Kurzinfor für die vollständige Dokumentation der Bibliothek). Wenn ihr mal zum Beispiel auf dem Nowhere-Festival, worüber neulich hier in der Make berichtet wurde (siehe Make 5/24, S. 64), LED-Effekte seht, achtet darauf: In sehr vielen Fällen sind diese Effekte aus dem Repertoire von WS2812FX. Anders als der Name suggeriert, kann diese Bibliothek auch viele andere LED-Typen ansteuern. Wir verwenden aber tatsächlich die namensgebenden WS2812(b)-LEDs, die es sehr günstig auf vorgefertigten Streifen zu kaufen gibt.

Die Bibliothek wird verwendet, indem man im `setup()` ein WS2812FX-Objekt definiert, initialisiert und startet (Listing 3). Zudem müssen ein Effekt, eine Geschwindigkeit und die Helligkeit ausgewählt und gegebenenfalls eine Grundfarbe festgelegt werden.

Wichtig ist, dass im Loop die Funktion `ws2812fx.service()` aufgerufen wird. Diese „worker function“ übernimmt die Aktualisierung der Farben und Helligkeiten gemäß dem gewählten Effekt und die Übertragung der Daten an die LEDs. Die WS2812-LEDs hängen in einer Kette: Die erste LED erhält die Daten für alle LEDs, setzt ihre Rot-, Grün- und Blauwerte nach den ersten drei Bytes und reicht den Rest der Daten über ihren Ausgang an die folgenden LEDs durch. Die zweite LED verwendet dann die ersten drei bei ihr ankommenden Bytes und so fort. Die LEDs haben also keine explizite Adresse, vielmehr ist ihre



Foto: Stefan Reckseigel

Bild 1: Hier wird die gleiche Technik benutzt, um auf zwei nebeneinander fahrenden Fahrrädern synchrone Regenbögen zu erzeugen.

Listing 3

```
WS2812FX ws2812fx = WS2812FX(NUM_LEDS, WS2812_PIN, NEO_GRB);
ws2812fx.init();
ws2812fx.setMode(12);
ws2812fx.setSpeed(1000);
ws2812fx.setBrightness(200);
ws2812fx.setColor(0xFF0000);
ws2812fx.start();
```



Bild 2: Der hier verwendete Li-Ion-Akku stammt aus einem Bluetooth-Adapter für ein etwas älteres Paar Bose-Kopfhörer.

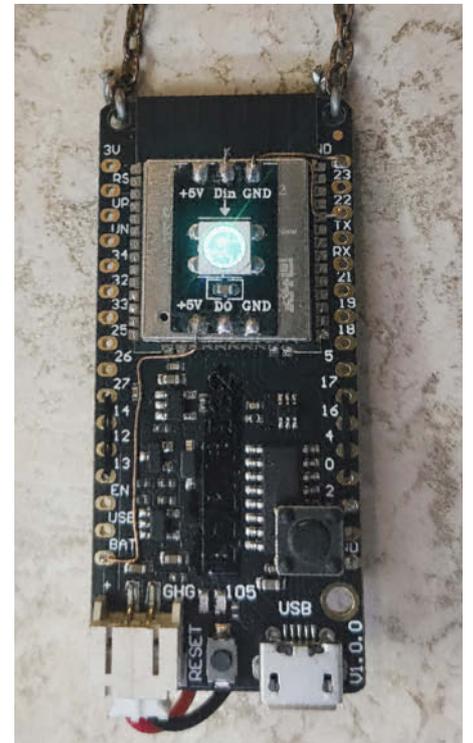


Bild 3: Hier ist ein originaler Wemos D32 mit dem ESP32 in einem kleinen Blechgehäuse zu sehen. Darauf eine WS2812-LED aus einem Streifen.

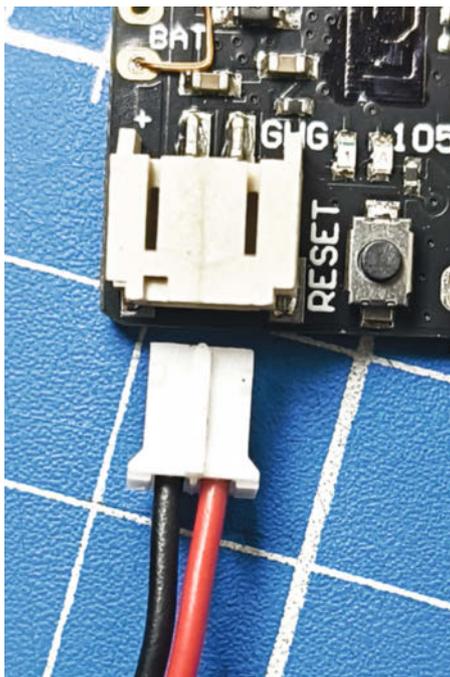


Bild 4: Hier müssen die Kabel des Akkus noch getauscht werden.

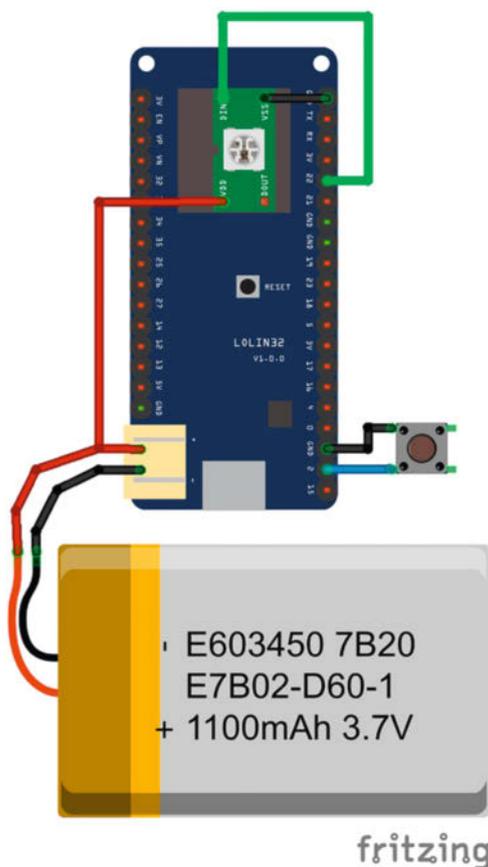


Bild 5: Der Schaltplan ist sehr übersichtlich. Taster und die Datenleitung der LED können an fast alle GPIOs angeschlossen werden.

Adresse durch ihre Position in der Kette vorgegeben und man kann diese LEDs beliebig aneinanderhängen. Sogar Verzweigungen sind möglich – wobei dann natürlich beide Äste die gleichen Effekte zeigen.

Der Pin, an den die LEDs (bzw. die erste LED) angeschlossen werden, ist im Wesentlichen beliebig, aber beim ESP8266 sollte man nicht GPIO0 (wird verwendet, um den ESP beim Booten in den Programmiermodus zu schalten), GPIO1 und GPIO3 (serielle Schnittstelle) oder GPIO2 (interne LED auf vielen Modulen) verwenden. GPIO4 ist eine gute Wahl.

Das Amulett bauen

Für das Basteln eines umhängbaren Amuletts eignet sich ein Wemos Lolin D32 (oder einer der vielen Nachbauten mit ähnlicher Funktionalität) aus zwei Gründen besonders gut: Zum einen, weil zwei passende Löcher vorhanden sind, an denen eine Schnur befestigt werden kann, um das Amulett um den Hals zu tragen, und zum zweiten, weil hier direkt ein Li-Ion-Akku angeschlossen werden kann, der über USB aufgeladen wird (Bild 2 und 3).

Achtung, die Hersteller von Akkus mit PH-2-Stecker und die Boardhersteller scheinen sich nicht über die Polarität der Stecker einigen zu können! Bei allen von mir beschafften Akkus musste ich die Kontakte vorsichtig aus dem Steckergehäuse lösen (mit einer Stecknadel die Widerhaken herunterdrücken) und tauschen (Bild 4), damit die Polarität mit der Buchse auf dem Modul übereinstimmte. Glücklicherweise waren die Steckverbinder überall eindeutig beschriftet und die Kabel der Akkus farblich gekennzeichnet (+: rot, -: schwarz).

Ich hatte einen 103450-Li-Ion-Akku herumliegen, der mit 2000 mAh mehr als genug Kapazität hat (ca. 10 Stunden bei 200 mA Verbrauch, je nach Leuchteffekten und ESP-Modell). Er lässt sich auch noch gut auf der Rückseite der Module befestigen. Ein kleinerer Akku tut es aber auch. Beispielweise sind in vielen Einweg-E-Zigaretten kleine Li-Ion-Akkus mit 3,7V Nennspannung verbaut, die sich sehr gut für Bastelprojekte eignen. Allerdings muss man hier noch Lade-/Schutzschaltungen nachrüsten, die bei aufladbaren Geräten schon vorhanden sind.

An dieser Stelle fragt man sich vielleicht, ob ein 3,7-V-Akku für den ESP32 und die WS2812-LEDs geeignet ist. Meine Antwort ist ja: Man kann sowohl den ESP als auch die WS2812 direkt (mit Akku-Schutzschaltung, falls nicht in den Akku eingebaut, natürlich!) an einen Li-Ion-Akku anschließen. Die 3,3 bis 4V sind dem ESP eigentlich etwas zu viel und dem WS2812 etwas zu wenig, beide funktionierten bei mir aber klaglos.

Was wir noch brauchen, ist mindestens eine WS2812-LED. Eine einzelne ist lichtstark genug, um den gewünschten Effekt zu er-

zeugen. Zu guter Letzt sollte auch ein Taster auf dem ESP-Modul eingebaut werden, falls dieser nicht eh schon vorhanden ist. Der Reset-Taster zählt nicht!

Der Schaltplan (Bild 5) ist denkbar einfach: Der Taster und die Datenleitung der WS2812-LED können an fast jeden GPIO angeschlossen werden. Die entsprechenden GPIOs müssen natürlich dann korrekt im Sketch eingetragen werden. Um die richtige Stelle zu finden, gibt es am Anfang des Sketches Kommentare mit Hinweisen zur Pinbelegung der verschiedenen hier erwähnten Module.

Wer möchte, kann statt des Wemos Lolin D32 auch den weit verbreiteten ESP8266 D1 Mini nehmen (Bild 6). Dieser hat allerdings keine eingebaute Akku-Ladeschaltung. Der Taster muss dann an GPIO0 und eine WS2812-LED an GPIO4 (auf den Modulen bezeichnet als D2) angelötet werden. Bei dem D1 mini habe ich die Leiterbahn des eingebauten Reset-Tasters aufgetrennt und stattdessen eine Verbindung zu GPIO0 (D3) hergestellt, sodass dieser ehemalige Reset-Taster für unsere Zwecke brauchbar wird. Die WS2812-LEDs auf den Bildern sind teilweise einzelne LEDs von einem längeren Streifen und auf dem D1 mini sitzt ein Adafruit-Modul, das die gleiche Funktionalität bietet.

Wer Akku und LED nicht direkt am ESP-Modul befestigen möchte, ist mit dem ESP32-C3 super mini gut bedient: Dieses Modul ist sehr klein, hat aber trotzdem noch Anschlüsse im relativ einfach zu lötenen 2,54-mm-Raster.

Bei dem D32-Klon (rechts im Bild 6) ist der Taster an GPIO14 angeschlossen (weil der gerade einen zum Taster passenden Abstand zum nächsten GND hatte). Die Datenleitung der LED hängt an GPIO32 (ebenfalls gerade an passender Position) und die interne LED an GPIO22 (weil der wohl für den Modulhersteller gerade passend neben dem Platz für die LED lag).

Wichtig ist, dass die WS2812-LEDs, die ja eigentlich eine 5-V-Versorgungsspannung haben möchten, direkt an die Akku-Anschlüsse auf dem Modul gelötet werden. Mit den 3,3 bis 4V der Akkus kommen die Konstantstromquellen in den LEDs gut zurecht, aber sie hinter dem Spannungsregler des ESP anzuschließen, ist bei einer LED unnötig und bei vielen eine sehr schlechte Idee. Der Grund dafür ist, dass der kleine Spannungsregler auf dem Mikrocontroller-Board nicht für die hohen Stromlasten ausgelegt ist, die eine größere Anzahl von LEDs erfordern. Außerdem verschwendet die unnötige Regelung wertvolle Batterieenergie. Bei voller Helligkeit auf allen drei Farbkanälen verbraucht eine einzige WS2812-LED bis zu 60 mA.

Zeit zum Flirten

Und was machen wir jetzt mit der fertigen Hardware? Das ist natürlich völlig euch über-

lassen, aber eine Idee, die ich selber ausprobiert habe, und mit der meine Freunde und ich jede Menge Spaß hatten, war es, beleuchtete Schals mit diesen ESP-Modulen auszustatten (Bild 7). In diesen Schals aus einem Stück Kunstpelz sitzt ein Streifen mit WS2812-LEDs, und wenn man einen Taster drückt, schalten alle gleichzeitig auf einen neuen, vorprogrammierten Modus um. Das sieht sehr beeindruckend aus, wenn mehrere Leute mit solchen Schals auf einer Party sind, insbesondere weil alle Schals synchron leuchten.

Interessant sind unsere Gadgets in Szenarien, in denen die Träger nicht vorher wissen, wer sonst noch so ein Amulett trägt, und somit auf das angezeigte Lichtmuster achten müssen, wenn andere Menschen in ihre Nähe kommen. Dies wird durch das Easter-Egg im letzten Sketch bereichert: Wenn auf einem der Module der Taster gedrückt gehalten wird, dann blitzen die LEDs auf den anderen Modulen in einem sehr schnellen Rhythmus in Rot und Weiß auf. Man kann also andere Träger solcher Amulette auf sich aufmerksam machen. Die Reichweite von der Signalübertragung hängt von der Art der Umgebung ab, aber mit zehn Metern ist auf jeden Fall zu rechnen.

Ob ihr diese Module wirklich zum Flirten verwenden wollt und ob ihr überhaupt mit Menschen, die mit solchen Dingen herumlaufen, flirten wollt, ist euch überlassen, aber hoffentlich habt ihr etwas über Ad-hoc-WLAN-Datenübertragung und coole LED-Gadgets gelernt. Ich bin gespannt auf weitere Anwendungsbeispiele!
—mch



Bild 7: Die gleichen ESP-Module können z. B. auch für beleuchtete Schals verwendet werden.

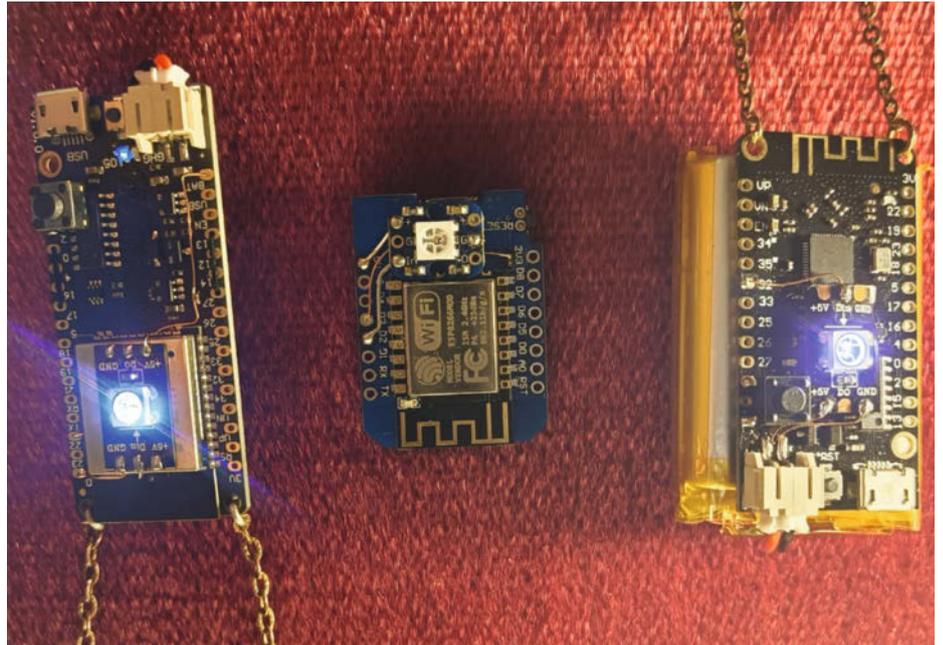


Bild 6: Links ist ein originaler Wemos D32, in der Mitte ist ein ESP8266 D1 Mini mit Adafruit WS2812-Modul und zum GPIO0-Taster umfunktioniernem Reset-Taster. Rechts sehen wir einen D32-Klon ohne Blechgehäuse. Die dünnen Verbindungen auf den Fotos sind Kupferlackdraht, also isoliert.

Wie Konzertarmbänder funktionieren

Die Übertragung von Informationen in tragbare Schmuckstücke kann nicht nur über WLAN erfolgen, wie in unseren Flirt-Amuletten. Drei weitere Möglichkeiten werden in sogenannten Konzertarmbändern genutzt: Infrarotstrahlung, Radiofrequenzen und Bluetooth. Beispielweise erhält jeder Besucher bei den aktuellen „Taylor Swift Eras World Tour“-Konzerten ein LED-bestücktes Armband. In diesen ist ein Infrarotsensor integriert, der in den Stadien von großen Infrarotscheinwerfern Signale empfangen kann, um Lichteffekte und auch Muster auf den Tribünen zu erzeugen. Das funktioniert, indem die beweglichen Scheinwerfer das Infrarotsignal gezielt in bestimmte Bereiche des Publikums senden und so nicht nur alle Armbänder in der gleichen Farbe aufleuchten lassen, sondern auch räumliche Muster erzeugen.

Bereits 2012 hat die britische Band Coldplay mit den sogenannten Xylobands damit begonnen. Die ersten Versionen kommunizierten per Funk, worüber vorprogrammierte Muster aktiviert wurden. Auf der aktuellen Welttour (bis November 2024) verwendet Coldplay Armbänder des Herstellers Pixmob, die zudem kompos-

tierbar sind und teilweise Bewegungssensoren besitzen, um noch weitere Effekte zu erzeugen.

Einen dritten Weg gehen viele K-Pop-Bands aus Korea. Dort werden bei Konzerten sogenannte Light Sticks eingesetzt, für die die Fans viel Geld ausgeben, um das Gefühl haben zu können, dabei zu sein. Die Light Sticks, die während des Konzerts in der Hand gehalten werden, sehen aus wie eine Mischung aus Anime-Zauberstab und Taschenlampe.

Hier wird oft Bluetooth für die Signalübertragung verwendet und die Konzertbesucher werden aufgefordert, eine App zu installieren, damit es funktioniert. Wenn alle gleichzeitig leuchten, entsteht ein sogenannter „Ozean“. Sind die Fans aber mit der einen oder anderen Boyband unzufrieden, werden die Light Sticks alle ausgeschaltet, was als „Black Ocean“ bezeichnet wird – ein Akt der gewollten Respektlosigkeit.

Die Zeiten, in denen das Publikum Feuerzeuge und Handy-Displays zum Leuchten brachte, sind eindeutig vorbei!

secIT by heise

HANNOVER 2025

18. - 20. MÄRZ 2025, HANNOVER

Die Kongress- messe für Security-Profis



A¹ Digital

achelos

Akamai

ALPENSCHILD

Avast

baramundi

Bare.ID

ConSecur
[security and consulting]

CONSIST
Business Information Technology

cybernality
SECURITY
PROPERTIES
EXPRESSION

cybereason

CYBER SAMURAI
WE SERVE AND PROTECT

»CYBERSENSE

CYQUEO
Cyber Security Solutions

e-x-e-o-n
Smart Cyber Security

FB

FB PRO GMBH
System Hardening & Data Protection

FORTINET

Greenbone

heylogin

ITISO

Net at Work
Building IT-Excellence.

NETCOM

netskope

nexttron
digital.com

ninjaOne

norton

semperis

SERUSEC
SERVICE AND SECURITY

SerNet

SITS

SONICWALL

SOPHOS

sosafe

SpaceNet

Offizieller
Eventpartner

itsa EXPO
CONGRESS
HOME OF IT SECURITY

Kooperations-
partner

CCISO
ALLIANCE

NIEDERSACHSEN
DIGITAL

UVN
UNTERNEHMENSBEREICH
NIEDERSACHSEN G.V.

© Copyright by Maker Media GmbH.

Aussteller



meet. learn. protect.



Jetzt anmelden!



secit-heise.de

© Copyright by Maker Media GmbH



Smarte Energie-Infotafel

Energiewende in den eigenen vier Wänden bedeutet zunächst zu verstehen, wann man womit wie viel Strom verbraucht. Diese große LED-Matrix-Uhr mit ESP32 hilft dabei, indem sie den Verbrauch, den mit der PV-Anlage erzeugten Strom und Tibber-Preise anzeigt.

von Joerg Plenert

Energiesparen war bei mir schon immer ein Thema. Mit der Energiewende und dem Ukraine-Krieg waren dann auch meine Frau und meine Kinder von dem „Energiespar-Virus“ infiziert und hatten Fragen: Welches Gerät verbraucht eigentlich wie viel Energie, wie viel erzeugt die PV-Anlage und was kostet der benötigte Strom gerade?

Um meine Familie diese notwendigen Informationen sehen zu lassen, habe ich ein smartes Display mit einem ESP32 und einer LED-Matrix gebaut und damit unsere alte Wohnzimmeruhr ersetzt. Neben der Uhrzeit (übers Internet per SNTP) stellt die Anzeige Informationen zur verbrauchten und produzierten Energie dar und lässt sich mit Werten aus Shelly-Modulen und Tibber speisen.

Das Projekt habe ich Home-Display (kurz HoDi) genannt, weil irgendwie alles bei uns im Haushalt einen Namen braucht. Mittlerweile ist die Anzeige der zentrale Punkt unserer häuslichen Energiewende geworden: „Papa, wir haben gerade Überschuss. Lass uns die Spülmaschine jetzt anmachen.“ Geht doch!

Fliegende Verdrahtung

Am einfachsten und schnellsten geht der Nachbau von HoDi in fliegender Verdrahtung,

Kurzinfo

- » Anzeige für Uhrzeit, Energieverbrauch, -erzeugung und -preise
- » LED-Matrix an ESP32 anschließen
- » Shelly-Geräte einbinden

Checkliste



Zeitaufwand:
2 Stunden



Kosten:
ca. 45 Euro

Werkzeug

- » Lötkolben
- » Seitenschneider
- » Abisolierwerkzeug

Mehr zum Thema

- » Uwe Rohne, Strompreise im Auge behalten, Make 4/24, S. 8
- » Ákos Fodor, LED-Matrizes mit MicroPython steuern, Make 3/24, S. 106
- » Daniel Bachfeld, Pixelart mit Pi, Make 5/21, S. 56



Alles zum Artikel
im Web unter
make-magazin.de/x8bs

HUB75-Panels

Die im Projekt verwendete RGB-LED-Matrix wird auch HUB75 genannt. Es gibt keinen richtigen Standard für diese Panels, aber was alle gemeinsam haben, ist der HUB75-Anschluss mit 2×8 Pins, in den ein Pfostenfeldstecker passt. Darüber hinaus unterscheiden sich die Panels in folgenden Merkmalen:

- » **Pitch:** Er beschreibt den Abstand der RGB-LEDs zueinander. Geläufige Werte sind hier 3 bis 6 mm. Eine Matrix mit 4 mm LED-Abstand wird als P4-Panel verkauft. Bei 64×32 LEDs ist die Darstellungsfläche einer solchen Matrix etwa 25,5×13 cm groß, eine P3-Matrix ist entsprechend kleiner.
- » **LED-Größe:** Auch die Maße der LEDs können variieren. So gibt es etwa Panels mit 2020-LEDs (2,0×2,0 mm) oder 2121-LEDs (2,1×2,1 mm).
- » **Auflösung:** HUB75-Panels haben entweder eine quadratische oder rechteckige Form. Entsprechend gibt es sie mit Auflösungen von 16×16 bis 128×64 LEDs, die man zu größeren Flächen zusammensetzen kann.
- » **Scan:** HUB75-Panels verwenden Multiplexing für die Darstellung. Dadurch leuchten nie alle LEDs gleichzeitig, sondern immer nur einzelne Zeilen nacheinander



Bild 1: Panels mit derselben Auflösung, aber unterschiedlichem LED-Abstand

(siehe Kasten „HUB75-Steuersignale“). Je nach Modell kann das z. B. jede 8., 16. oder 32. sein, was man als Scan bezeichnet. Einen Hinweis darauf, wie viele Zeilen gleichzeitig auf der LED-Matrix leuchten können, findet man in der Typenbezeichnung des Displays. Dort steht dann z. B. 16S für 1/16-Scan bzw. jede 16. Zeile.

Die HoDi-Firmware unterstützt nur Panels mit einer Auflösung von 64×32 LEDs. Folgende Panel-Typen haben bei mir stabil mit HoDi funktioniert:

- » P4-2121-64*32-16S-HL1
- » P4-2020-32x64-16S-S2-RMG
- » P3-2121-6432-16S-D1

HUB75-Steuersignale

Bei einem HUB75-Panel steuern spezialisierte ICs (z. B. ICN2037), die LED-Treiber und Schieberegister vereinen, einzelne LED-Zeilen. Diese werden sequenziell angezeigt. Der Zeilenwechsel passiert dabei aber so schnell, dass das menschliche Auge ein zweidimensionales Bild auf der LED-Matrix wahrnimmt – der Effekt ist ähnlich wie bei einem alten Röhrenfernseher.

Da eine HUB75-Matrix keinen Controller hat, müssen die Inhalte bzw. Daten von außen kommen (hier vom ESP32). Diese nimmt das Panel über den ersten HUB75-Anschluss auf der linken Seite entgegen, von dem aus ein weißer Pfeil wegzeigt (Bild 2). Für jeden Zeilendurchlauf erwartet die von mir verwendete Matrix mit 64x32 LEDs folgende Signale:

- » RGB-Farbwerte an den Pins R0, G0, B0 für die obere Matrix-Hälfte und an R1, G1, B1 für die untere
- » eine Null oder Eins an den Pins A bis D (2⁴ = 16), um die aktuellen LED-Zeilen auszuwählen
- » ein Clock-Signal für jedes RGB-Bit, das ins Register geschoben wird
- » ein Latch-Signal, um die empfangenen seriellen Daten für eine Zeile parallel auszugeben
- » ein Signal an OE (Output Enable), das die LEDs ein- oder ausschaltet. Über ein PWM-

Signal an OE lässt sich die Helligkeit des Panels auch steuern.

Damit man sich über das Timing dieses Ablaufs und die Koordination der Register keine Gedanken machen muss, gibt es zum Glück bereits ein paar Bibliotheken von Adafruit und anderen Entwicklern, die diese Aufgabe übernehmen und vereinfachte Befehle anbieten, um etwas auf einer LED-Matrix darzustellen. Ich habe für HoDi die SmartMatrix-Bibliothek verwendet.

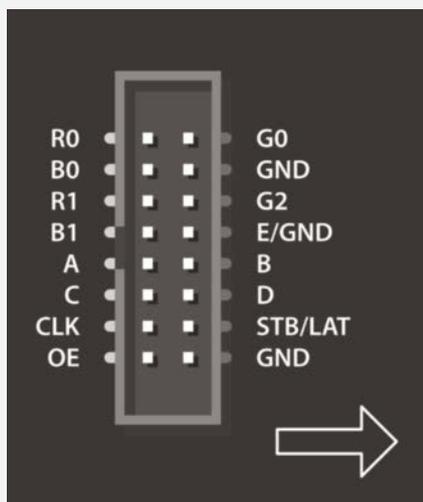


Bild 2: Der HUB75-Anschluss auf der Rückseite einer Matrix

HUB75 mit ESP32-DevKitC verbinden

HUB75-Matrix	ESP32
R0	GPIO0
G0	GPIO15
B0	GPIO4
GND	GND
R1	GPIO16
G1	GPIO27
B1	GPIO17
E/GND	-
A	GPIO5
B	GPIO18
C	GPIO19
D	GPIO21
CLK	GPIO22
STB/LAT	GPIO33
OE	GPIO25
GND	GND
+5V	+5V
GND	GND

Bei manchen HUB75-Panels beginnen die RGB-Pins nicht mit R0, sondern R1. Entsprechend gibt es dann auch R2. Die Anordnung ist aber dieselbe.

da man nur ein ESP32-DevKitC, eine HUB75-LED-Matrix (siehe Kasten „HUB75-Panels“) und die in der Regel mitgelieferten Kabel benötigt.

Das graue Flachbandkabel, mit dem man die LED-Matrix normalerweise mit einem weiteren Panel verbindet, eignet sich auch dazu, einen ESP32 anzuschließen. Dazu müsst ihr den Pfostenfeldstecker zunächst direkt am Kabel abschneiden und die einzelnen Adern ein Stück auftrennen (Bild 3).

Isoliert anschließend die Enden der Adern ca. 5 mm weit ab. Das ist bei den Flachbandkabeln meist etwas knifflig, da die Isolierung sehr weich und die Kabel sehr dünn sind. Da helfen etwas Geduld und ein ruhiges Händchen. Verzinnt die abisolierten Kabel danach und lötet die Datenleitungen an das ESP32-Board an, wie in der Tabelle „HUB75 mit ESP32 verbinden“ und im Schaltplan (Bild 4) angegeben. Was es mit dem Kondensator an C3 auf sich hat, erkläre ich im nächsten Abschnitt.

Verwendet für die Verbindungen zur Spannungsversorgung das beiliegende Stromkabel der LED-Matrix. Der ESP32 leitet die über Micro-USB eingehenden 5V über den 5-V-Pin direkt weiter an die Anzeige – zumindest ist das beim Original-Board von Espressif so, dessen Schaltplan in der Kurzinfor verlinkt ist. Nachbauten aus China können von diesem Referenzdesign abweichen. Achtet auf jeden Fall darauf, dass das Netzteil genug Strom über USB für das Board und die LED-Matrix liefern kann (siehe Kasten „Stromversorgung“).

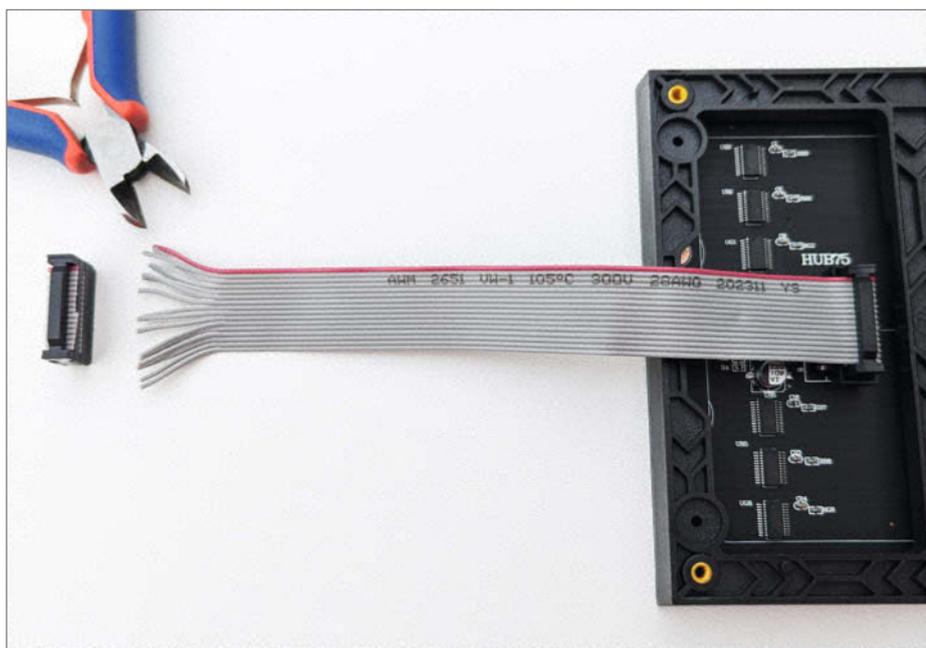


Bild 3: Das graue Verbindungskabel lässt sich zerteilen, um einen ESP32 anzuschließen.

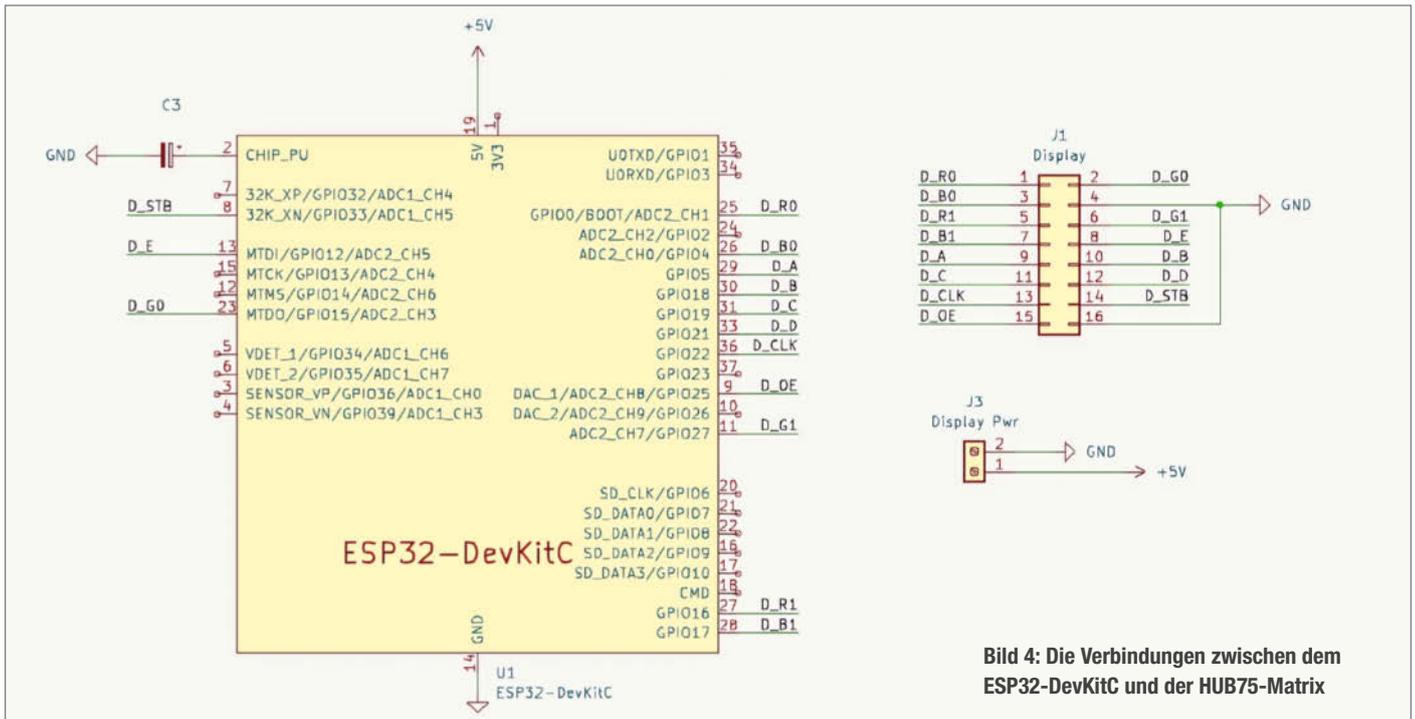


Bild 4: Die Verbindungen zwischen dem ESP32-DevKitC und der HUB75-Matrix

Sobald alle Kabel verbunden sind, müsst ihr nur noch die Firmware auf den Mikrocontroller flashen.

HoDi-Firmware

Die Firmware für HoDi steht kostenfrei als kompilierte Binary und als Source-Code zum Download bereit (siehe Link in der Kurzinfor). Es gibt sie als Voll- und Updateversion, wobei letztere nur über das Web-Interface auf einem bereits programmierten HoDi eingespielt werden kann. Für die Initialprogrammierung benötigt ihr also die Vollversion.

Damit der ESP32 per USB programmiert werden kann, muss er in den Bootloader starten. Das passiert, wenn man beim Einschalten den GPIO0 auf GND legt. Eigentlich kann das Flash-Tool per USB über die Datenleitungen DTR und RTS den GPIO0 und Reset bedienen und so den ESP32 in den benötigten Modus versetzen. Nur leider funktioniert das nicht immer. Bei dem ESP32-DevKitC (und vielen anderen Board-Varianten) gibt es hierfür einen Taster, der mit „Boot“ beschriftet ist. Wenn ihr ihn gedrückt haltet und einmal die Reset-Taste auf dem ESP betätigt, startet der Bootloader. Damit ich das nicht mehr machen muss, habe ich am Chip_PU-Pin einen Kondensator mit 4,7µF angebracht und mit GND verbunden (siehe Bild 4).

Sobald der ESP32 angeschlossen und im Bootmodus ist, habt ihr mehrere Möglichkeiten, den Mikrocontroller zu flashen:

- Die einfachste Methode ist, die Firmware direkt über den Browser (am besten Google Chrome) auf den ESP32 zu laden, indem ihr

auf der HoDi-Website (siehe Link in der Kurzinfor) nach unten scrollt, bis ihr einen blauen „Connect“-Button seht. Klickt diesen an und folgt den weiteren Anweisungen (Bild 7).

- Als Alternative bietet sich auch der ESP-Flasher von Tasmota an. Die benötigte Vollversion als BIN-Datei findet ihr ebenfalls als Download auf der Projekt-Website.

- Als dritte Option könnt ihr zum Übertragen auch das esptool von Espressif verwenden. Falls ihr die Firmware anpassen und selbst kompilieren möchtet, solltet ihr mit Visual Studio Code und der ESP-IDF (Espressif IoT Development Framework) sowie TypeScript vertraut sein. Die notwendigen Repositories findet ihr über den Link in der Kurzinfor.

Stromversorgung

Eine LED-Matrix mit 2048 LEDs kann theoretisch eine ganze Menge Strom verbrauchen. Würden alle LEDs zur gleichen Zeit weiß leuchten (RGB auf voller Helligkeit), kann das schon mal 30 mA je LED verbrauchen. Das wären dann also 61 A für das ganze Display – da wird der LED-Scheinwerfer im Auto neidisch. Durch das Multiplexing leuchten aber nur einzelne Zeilen nacheinander. Bei einem 1/16-Scan ist das jede 16. Zeile, was bei 32 Zeilen den Stromverbrauch auf 2×64 LEDs reduziert. Mit 30 mA multipliziert ergeben sich etwa 3,8 A.

Im Betrieb leuchten aber eigentlich nie alle LEDs weiß, sondern in unterschiedlichen Farben. Und HoDi stellt sogar nur Zeichen auf einem größtenteils dunklen Hintergrund dar. Dadurch bleiben viele LEDs beim Multiplexing ausgeschaltet. Der Stromverbrauch einer P4-Matrix,

die mit HoDi läuft, liegt daher eher bei 300 mA.

Beim Einschalten können aber dann doch schon mal kurzfristig 3 A bei 5V notwendig sein. Hier habe ich gute Erfahrungen mit den 3-Port-3-A-USB-Ladegeräten von Hosow gemacht. Alternativ könnte es auch mit einem schwächeren Netzteil funktionieren, wenn man an der Spannungsversorgung der Matrix einen großen (Stütz-) Kondensator (> 3000 µF) anbringt. Dieser kann kurzzeitig einen Spannungseinbruch bei hoher Stromlast verhindern.

Falls ihr mit der in diesem Artikel gezeigten Verbindungsmethode Projekte plant, die dauerhaft mehr Strom benötigen als HoDi, solltet ihr die Spannungsversorgung der LED-Matrix direkt an das Netzteil anschließen und diese nicht über das ESP32-Board laufen lassen.

Alternativer Aufbau mit HoDi-Platine

Da mir eine fliegende Verdrahtung auf Dauer zu fragil ist und man mittlerweile sehr günstig Platinen fertigen lassen kann, greife ich oft schon während der Prototyping-Phase zur Software KiCad.

Bei dem Design der HoDi-Leiterplatte habe ich insbesondere auf die elektromagnetische Verträglichkeit geachtet. So haben die beiden Layer der Platine große und an vielen Stellen durchkontaktierte Masseflächen. Um Störungen im WLAN-Signal zu vermeiden, empfehle ich zudem, ein ESP32-DevKitC mit externer Antenne zu wählen, also mit einem ESP32-WROOM-32UE auf dem Board (siehe Bild 5). Zudem habe ich mit zwei Abblockkondensatoren (C1 und C2) experimentiert, um Störungen zu verringern.

Die Leiterplatte passt optimal auf eine P4-LED-Matrix (siehe Kasten „HUB75-Panels“). Ihr könnt die Platine auf der Projekt-Website als Gerber-Datei herunterladen, um sie bei den üblichen Dienstleistern (JLCPCB, PCBWay o. Ä.) fertigen zu lassen.

Beim Bestücken der Leiterplatte solltet ihr euch an folgende Reihenfolge halten:

- » der Steckplatz C1 (unter dem ESP32) kann unbestückt bleiben, alternativ mit einem 100-nF-Kondensator bestücken
- » das ESP32-DevKitC auf die Leiterplatte stecken, die Pins verlöten und danach kürzen
- » C2 mit einem 100-nF-Kondensator bestücken
- » C3 mit einem 4,7-µF-Kondensator bestücken (auf die Polarität achten)
- » eine Aderleitung für die Spannungsversorgung der Matrix an der Platine anlöten (siehe VCC-Markierung in Bild 6).

Entfernt anschließend auf der Rückseite der LED-Matrix die Kunststoffführung des HUB75-Anschlusses, sodass die Pins freiliegen. Trennt dazu einfach die Ecken mit einem Seitenschneider auf, dann lässt sich die Führung einfach umknicken und sauber abbrechen. Legt etwas Schaumstoff zwischen die HoDi- und LED-Matrix-Platine, damit sich ihre Pins nicht aus Versehen berühren, und verlötet beide Leiterplatten, wie in Bild 6 zu sehen ist.

Die Aderleitung für die Spannungsversorgung der Matrix könnt ihr unter den Stegen entlangführen und müsst sie danach an VCC der Matrix anlöten. Ein Kabel von der Spannungsversorgung zu GND ist nicht nötig, weil diese Verbindung bereits über den HUB75-Stecker besteht.



Bild 5: Das ESP32-DevKitC gibt es auch mit einem Modul, an das man eine externe Antenne anschließen kann.

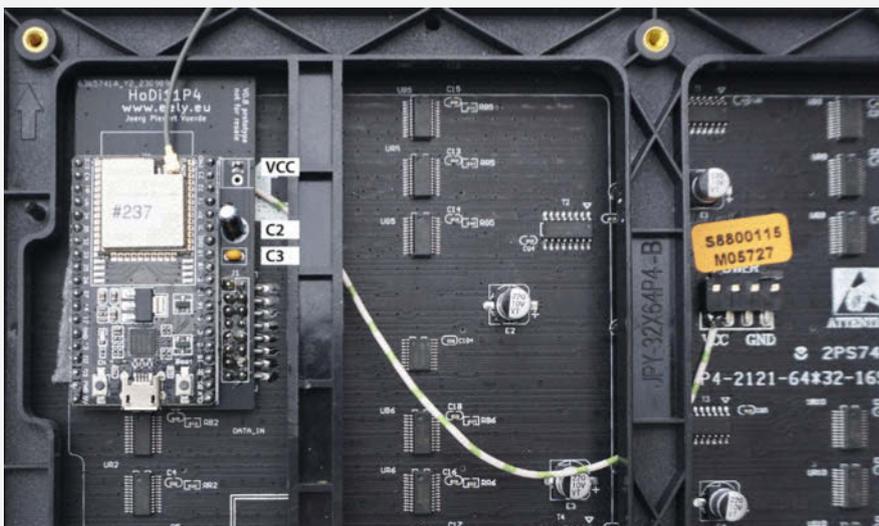


Bild 6: Die HoDi-Leiterplatte mit aufgelötetem ESP32-Board auf der HUB75-Matrix

Installation

Installation of the latest version via [esp-web-tools](#):

- Works best with Chrome
- Connect your HoDi via USB to your PC
- Click  and follow the instructions for installation

Bild 7: Schließt den ESP32 an und klickt auf der Projekt-Website unter „Downloads“ auf „Connect“.

HoDi mit WLAN verbinden

Die schmutzige Hardwarearbeit ist getan, nun geht es an das Einstellen. Da HoDi noch keine Parameter hat, startet der ESP32 erst mal als Access-Point (Bild 8). Sucht auf dem Smartphone (oder Computer) nach einem WLAN mit der SSID „eely_“, gefolgt von sechs Zahlen und Buchstaben. Notiert euch die Bezeichnung, denn ihr braucht sie später noch, um auf das Web-Interface zuzugreifen. Nach dem Anmelden sollte sich auf dem Smartphone direkt die Startseite von HoDi öffnen. Passiert das nicht, könnt ihr im Browser die IP-Adresse des Systems auch manuell eingeben: 192.168.4.1.

Geht danach am oberen Bildschirmrand auf die Einstellungsseite (Zahnrad-Symbol)

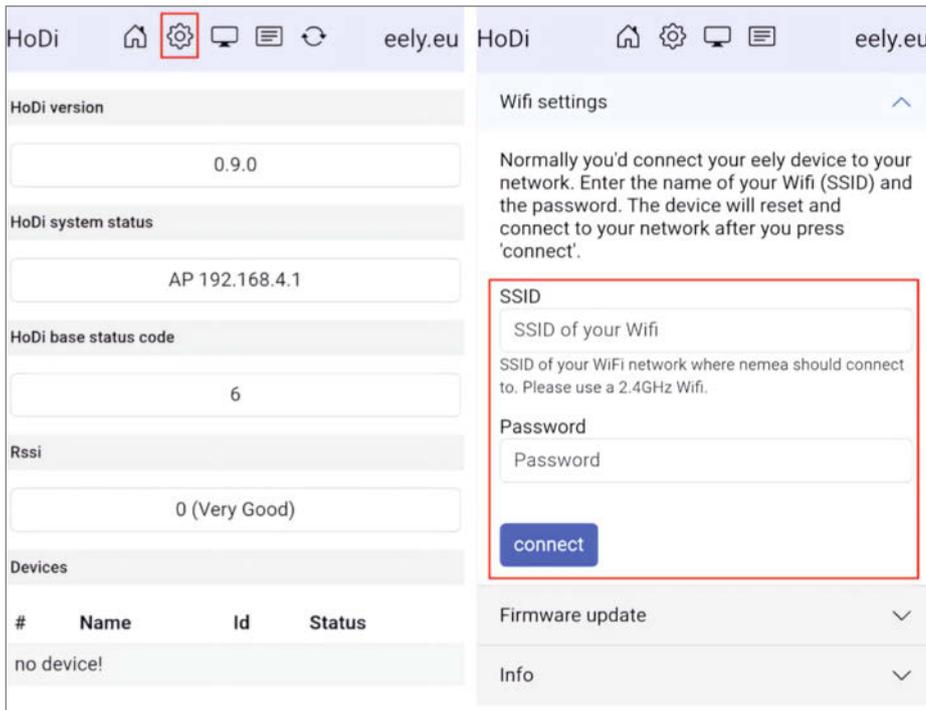


Bild 9: Um HoDi ins lokale Netzwerk einzubinden, gibt man seine WLAN-Zugangsdaten ein.



Bild 8: HoDi zeigt beim ersten Start die IP vom Webinterface an.

und dort in die Kategorie „Wifi settings“. Gebt unter „SSID“ den Namen eures WLANs an und unter „Passwort“ euer Kennwort (Bild 9). Bedenkt dabei, dass der ESP32 und somit auch HoDi nur mit 2,4-GHz-WLANs kommunizieren kann.

Nach einem beherzten Drücken auf „Connect“ startet HoDi neu. Wenn alles funktioniert hat, wird nach einigen Startinformationen die aktuelle Uhrzeit in großen Zahlen auf der LED-Matrix angezeigt (Bild 10). Ansonsten gibt das Panel die Anzahl der fehlerhaften Verbindungsversuche aus. Dann könnt ihr HoDi zurücksetzen (siehe „Reset und Update tut gut“).



Bild 10: Noch ist HoDi eine einfache LED-Uhr.

Jetzt könnt ihr den ESP über euer WLAN-Netzwerk erreichen. Dazu gebt ihr, wenn ihr mit eurem Netzwerk verbunden seid, in einen Browser die zuvor notierte eely-SSID ein oder die IP-Adresse, die ihr über euren Router oder DHCP-Server herausfinden könnt.

Da HoDi weder HTTPS beherrscht noch beim Zugriff auf das Web-Interface eine Authentifizierung erfordert, eignet sich das Projekt nur für die Verwendung in einem geschützten Netzwerk mit WLAN-Verschlüsselung, z. B. zu Hause.

Damit HoDi mehr als nur die Uhrzeit anzeigt, kann man folgende Shelly-Geräte als Sensoren ins System einbinden:

Shelly und Tibber

Damit HoDi mehr als nur die Uhrzeit anzeigt, kann man folgende Shelly-Geräte als Sensoren ins System einbinden:

- Shelly 3EM
- Shelly Pro 3EM
- Shelly H&T
- Shelly Plug S

Die Daten von Shelly-Geräten lassen sich über die Shelly-Cloud oder direkt über das lokale Netzwerk auslesen. Bei netzbasierten Geräten wie dem Shelly 3EM empfehle ich den direkten Zugriff, bei batteriebetriebenen Geräten wie dem Shelly H&T funktioniert der Zugriff in der Regel nur über die Cloud.

Damit HoDi Daten über die Shelly-Cloud abrufen kann, müsst ihr über das Bildschirm-Symbol im Web-Interface unter „Shelly cloud settings“ den „Cloud authorization key“ sowie

Für alles gerüstet!

Tests, Tipps und Tools



Das Sonderheft richtet sich vor allem an Privatnutzer, Freelancer und kleinere Unternehmen und enthält Kaufberatungen, Tests und Praxisanleitungen zu typischen Büroprogrammen, auch abseits von Microsoft Office.



Heft für 14,90 € • PDF für 12,99 €
Heft + PDF 19,90 €

shop.heise.de/ct-homeoffice24

JETZT BESTELLEN!



Generell portofreie Lieferung für Heise Medien- oder Maker Media Zeitschriften-Abonnenten oder ab einem Einkaufswert von 20 € (innerhalb Deutschlands). Nur solange der Vorrat reicht. Preisänderungen vorbehalten. E-Books können einem DRM-Schutz unterliegen.

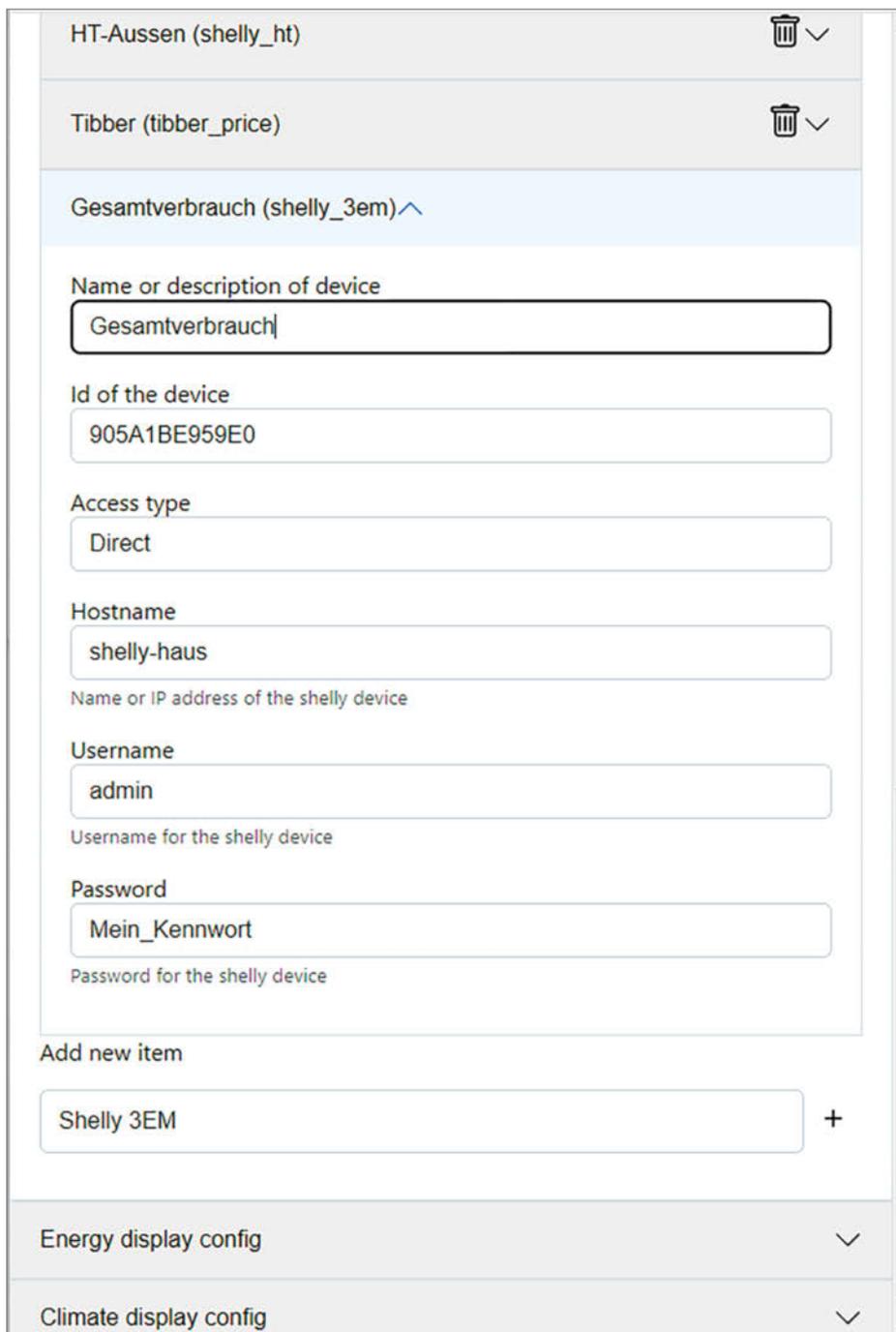


Bild 11: Neue Geräte fügt man über das Plus-Symbol hinzu und konfiguriert sie danach.

den „Cloud host“ eintragen. Beides findet ihr im Shelly-Portal.

Unter „Devices config“ können dann neue Geräte zu HoDi hinzugefügt werden (Bild 11). Dazu wählt man bei „Add new item“ erst aus dem Dropdown-Menü den gewünschten Gerätetyp aus und tippt anschließend auf das Plus-Symbol rechts daneben. In die daraufhin erscheinende Maske trägt man dann alle benötigten Daten für sein Gerät ein. Über das Mülltonne-Symbol lassen sich Geräte wieder löschen.

Auch die Energiepreise von Tibber lassen sich mit HoDi anzeigen. Hierfür muss man unter „Tibber cloud settings“ sein „Tibber cloud token“ angeben, das man auf der Tibber-Entwicklerseite erhält (siehe Link in der Kurzinfo). Danach fügt man Tibber wie die Shellys als Gerät bei „Devices config“ hinzu.

Darstellungsoptionen

Damit sind die Stammdaten in HoDi erfasst. Ob das System mit den Geräten und Diensten

kommunizieren kann, lässt sich auf der Startseite (Haus-Symbol) überprüfen. Im letzten Schritt müsst ihr nun noch konfigurieren, welche Daten HoDi wie anzeigen soll.

Das ist bei Tibber am einfachsten, denn man muss lediglich im Menü „Display settings“ den Haken bei „Show energy prices“ setzen. Dann werden die Energiepreise für die aktuelle Stunde und die beiden darauffolgenden Stunden angezeigt (Bild 12). Dabei erscheinen die einzelnen Preise gemäß der Tibber-Preiskala grün (günstigster Preis), gelb (normaler Preis) oder rot (hoher Preis).

Die Konfiguration der Energieanzeige ist etwas komplizierter. HoDi kann mit mehreren Energiezählern im Haus umgehen und die Werte addieren oder subtrahieren. Wenn ihr nur einen Shelly-Energiezähler direkt am Hausanschluss installiert habt, wählt ihr unter „Energy display type“ den Wert „total“ aus. Im Feld „Total calculation“ gebt ihr danach das Vorzeichen (+/-) und die Nummer des Gerätes ein, wie sie auf der Startseite lautet, z. B. „+1“.

Nach dem Neustart zeigt HoDi nun den Gesamtverbrauch (rot) oder die Einspeisung (grün) an (Bild 13). Sollte das bei euch nicht passen, dann könnt ihr im Feld „Total calculation“ einfach das Vorzeichen wechseln, also auf „-1“.

Mit der Option „Total & production“ lässt sich dann der Gesamtverbrauch und die aktuelle Erzeugung parallel anzeigen. Hierzu braucht man dann aber mindestens zwei Energiezähler (Bild 14). Bei einem Balkonkraftwerk könnte hierzu auch ein Shelly Plug S dienen (Bild 15).

Ganz schön kalt draußen

Eine weitere schöne Option ist die Anzeige von Temperaturen. Hierzu können die „Shelly H&T“-Module genutzt werden, die dann aber über die Cloud kommunizieren. Das liegt daran, dass die batteriebetriebenen Geräte nicht ständig online sind, sondern sich nur kurzfristig mit dem WLAN verbinden, um Updates in die Cloud zu senden. So kann HoDi nicht in Echtzeit auf diese Shellys zugreifen, um die aktuelle Temperatur zu erhalten. Eigentlich sind die H&T-Module auch nicht für den Außeneinsatz gedacht, lassen sich aber doch recht gut draußen betreiben, wenn man sie vor Regen schützt.

Wenn man ein solches Gerät hinzufügt, kann man die Anzeige über den Bereich „Climate display config“ parametrisieren. Die Syntax dafür ist etwas sperrig und jeder UX-Designer würde schreiend davonlaufen, aber sie erfüllt ihren Zweck. Mit dem Wert „I\$4.0**“ wird auf dem Display ein „I“ (für Innentemperatur) gefolgt von dem Temperaturwert des vierten Gerätes („\$4.0“) und ein Grad-Zeichen („**)“ angezeigt. Die Luftfeuchtigkeit des vierten Gerätes würde man mit „\$4.1“ ausgeben.



Bild 12: HoDi kann unter der Uhrzeit Tibber-Preise anzeigen.



Bild 13: Mit HoDi kann man immer sehen, wie viel Strom der Haushalt gerade verbraucht.

Hat man zwei Sensoren (etwa für drinnen und draußen), könnte man mit „I\$4.0* A\$5.0*“ beide Temperaturen darstellen (Bild 16).

Sind mehrere Anzeigen konfiguriert, also z. B. Energieanzeige und Tibber, dann wechselt HoDi alle paar Sekunden die Darstellung.

Reset und Update tut gut

Die HoDi-Firmware ist weit davon entfernt, perfekt zu sein. So kann es, insbesondere bei falschen Parametern, schon mal vorkommen, dass das Display nicht mehr starten will. Für diesen Fall kann man es mit einem Trick vollständig zurücksetzen. Hierzu muss man fünfmal hintereinander den Reset-Knopf drücken, immer wenn beim Startvorgang auf der LED-Matrix „Network init“ steht. Danach löscht HoDi alle Daten und startet wieder im AP-Modus. Von hier aus kann man das System neu konfigurieren.

HoDi bietet auch die Möglichkeit, die Einstellungen als JSON zu sichern und zu setzen. Man muss nur sehr vorsichtig sein, wenn man etwas im JSON manuell ändert, denn der JSON-Parser von HoDi verzeiht keine Fehler, sodass man im Zweifel das System zurücksetzen muss.

Für Fehlerbehebungen und neue Funktionen empfehle ich, ab und zu mal nach Updates Ausschau zu halten. Update-Dateien können über die Funktion „Firmware update“ eingespielt werden. Danach muss man nur noch den ESP neu starten. —akf

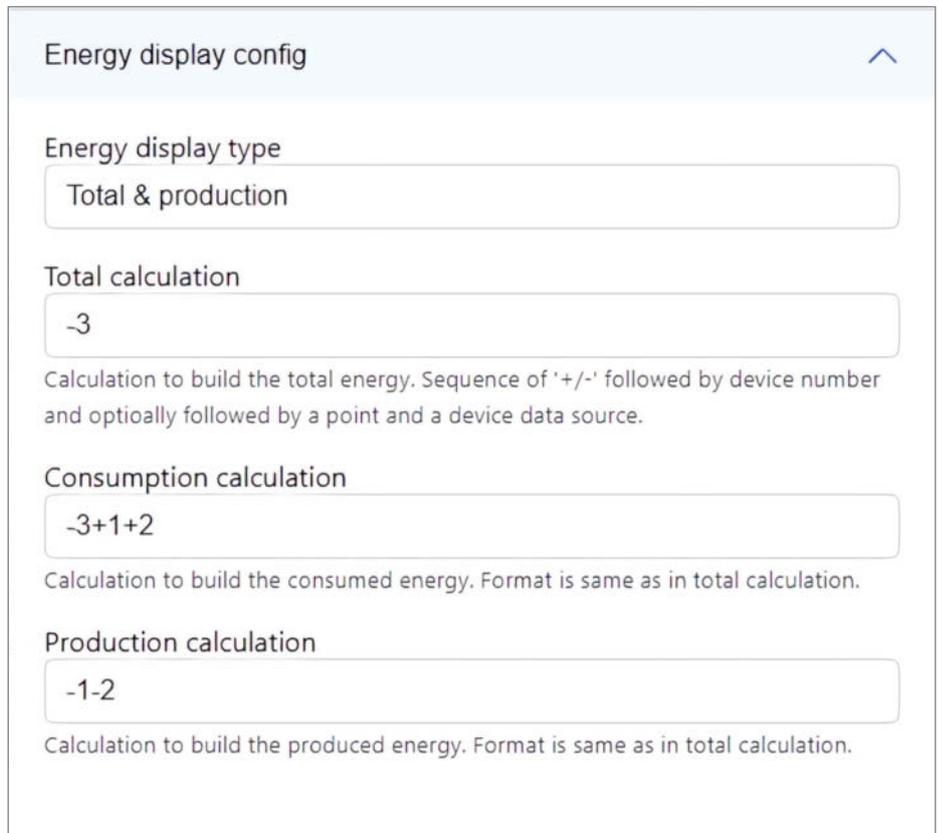


Bild 14: Die Werte einzelner Shellys lassen sich miteinander verrechnen.



Bild 15: HoDi mit Einspeisung von 1,4kW und 4,8kW PV-Erzeugung, 3,4kW werden also gerade im Haus verbraucht.

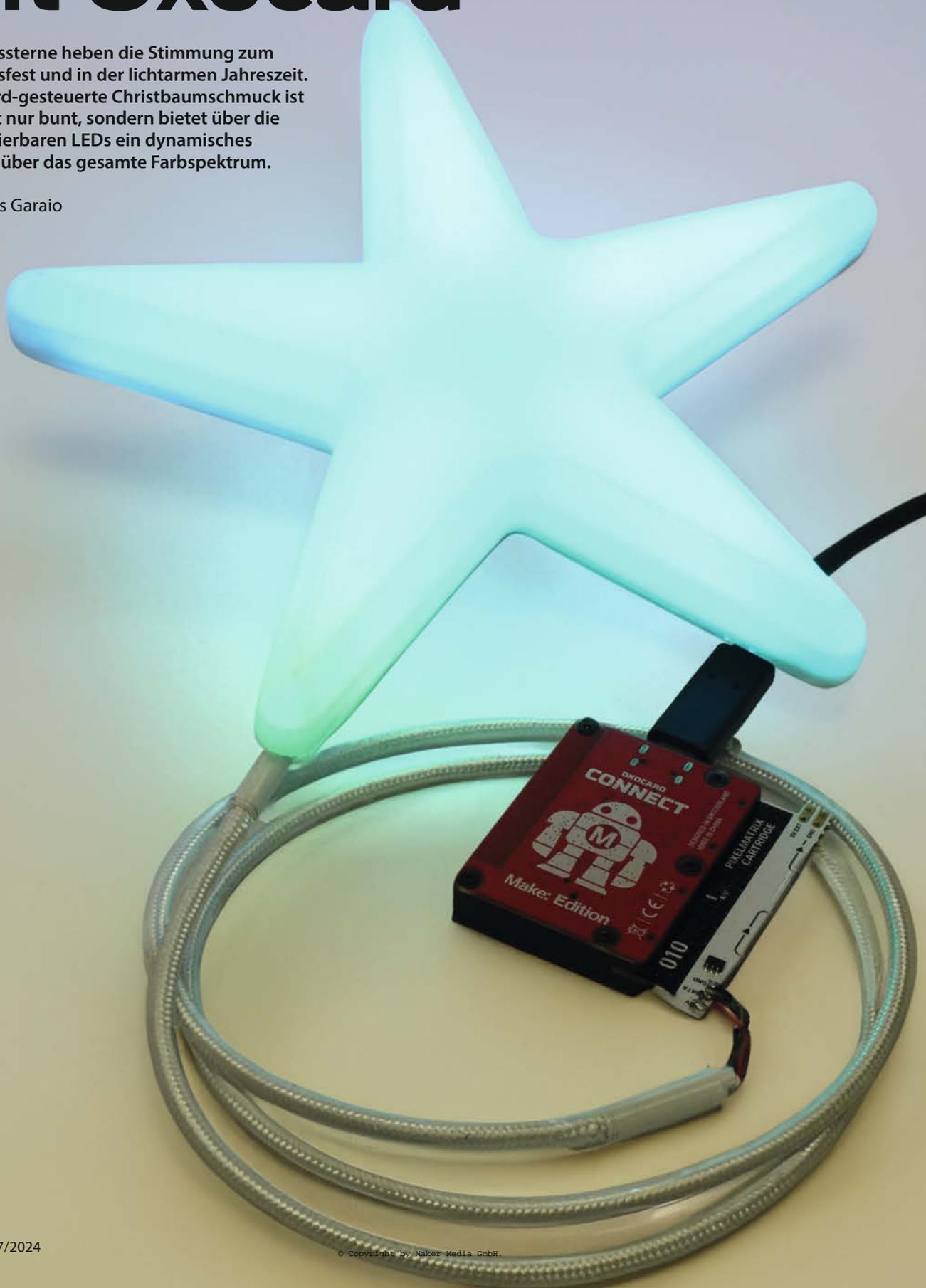


Bild 16: Die Shellys liefern die Innen- und Außentemperatur.

Weihnachtsstern mit Oxocard

Weihnachtssterne heben die Stimmung zum Weihnachtsfest und in der lichtarmen Jahreszeit. Der Oxocard-gesteuerte Christbaumschmuck ist dabei nicht nur bunt, sondern bietet über die programmierbaren LEDs ein dynamisches Feuerwerk über das gesamte Farbspektrum.

von Thomas Garaio



Als Maker steht man häufig unter Rechtfertigungsdruck, wenn man sich am Wochenende in den Bastelkeller zurückzieht und für Familie und Freunde kaum ansprechbar ist. Kann das Ergebnis der tagesfüllenden Bastelei nicht überzeugen, ist der häusliche Frieden ernsthaft in Gefahr. Bei dem Oxocard-LED-Weihnachtsstern besteht diese Gefahr nicht. Innerhalb von wenigen Stunden entsteht ein Schmuckstück, das nicht nur zur Weihnachtszeit Jung und Alt Freude bereitet. Gut möglich, dass die Begeisterung von Familienmitgliedern, Nachbarn und Freunden sogar zu einer Kleinserienproduktion führt, damit jeder so einen persönlichen Schmuckstern sein Eigen nennen kann.

Grundaufbau

Unser Weihnachtsstern enthält 25 einzeln adressierbare RGB-LEDs. Beim Aufbau des Gehäuses haben wir darauf geachtet, dass die einzelnen Lichtpunkte nicht als solche erkennbar sind, sondern ineinander übergehen. Schaltet man alle LEDs ein, wird der Stern homogen ausgeleuchtet. Dadurch ist es möglich, durch geschickte Programmierung interessante Effekte zu erzielen, für die man ansonsten viel mehr Pixel in höherer Auflösung bräuhete. Um den Löt Aufwand möglichst gering zu halten, verwenden wir die Pixelmatrix-Cartridge, aus der wir Streifen herausbrechen, in Sternform neu positionieren und dann zusammenlöten.

Das Gehäuse besteht aus vier Teilen, die sich mit handelsüblichen 3D-Druckern anfertigen lassen. Die vier einzelnen Teile kann man entweder verkleben oder beispielsweise mit vier M3-Schrauben befestigen. Der Deckel dient als Diffusor. Er streut das Licht der einzelnen Pixel derart, dass die Bildpunkte nicht einzeln sichtbar sind.

Innenleben

Im vorgestellten Projekt nutzen wir zur Steuerung die Oxocard Connect, bei der die Ansteuerung über die NanoPy-IDE und das kleine Betriebssystem extrem einfach und flexibel ist. Die Ansteuerung der LEDs und die Lichteffekte sind in NanoPy programmiert, einer für Mikrocontroller optimierten Programmiersprache, die sich stark an Python orientiert.

Die Erweiterung ist problemlos möglich. Über den eingebauten Cartridge-Slot lassen sich verschiedene Erweiterungsplatinen einstecken und sofort nutzen. In unserem Beispiel verwenden wir die aus 25 Pixeln bestehende Pixelmatrix-Cartridge als Basis des Sterns. Steckt man diese ein, startet automatisch eine Animation, ohne dass man zusätzlich etwas installieren oder selbst programmieren muss. Schon damit lassen sich erstaunliche optische Basiseffekte für das kleine Projekt erzielen.

Kurzinfo

- » LED-Schmuck mit der Oxocard steuern
- » LED-Streifen der Pixelmatrix-Cartridge ansteuern
- » Mit NanoPy Farbeffekte programmieren

Checkliste



Zeitaufwand:

2 Stunden (6 Stunden 3D-Druck)



Kosten:

unter 99 Euro (inkl. Oxocard Connect Innovators Kit für 55 Euro)

Werkzeug

- » 3D-Drucker
- » Heißklebepistole
- » Löt Kolben

Mehr zum Thema

- » Daniel Bachfeld, Innovators Kit – Make Edition, Make 5/2024, S 38
- » Daniel Bachfeld, Einführung in Oxocard Connect: Experimentierset für Maker, heise online



Material

- » Oxocard Connect Innovators Kit (Make Edition)
- » USB-Power-Bank
- » weißes Filament
- » Pixelmatrix-Cartridge
- » Isolierband
- » Litze
- » altes USB-Kabel

Oxocard Connect Innovators Kit (Make-Edition)

Das Komplettpaket zum einfachen Einstieg in die faszinierende Welt der Elektronik im heise shop!

- » Enthält eine Oxocard Connect und eine Breadboard-Cartridge
- » 96 Elektronikbauteile mit Temperatur-, Licht- und Bewegungssensor
- » Kompletter Elektronik-Kurs (Online) mit 13 Beispielen

- » 2 MB PSRAM, 8 MB Flash
- » Hochauflösendes Display
- » Vorinstalliertes Betriebssystem mit diversen Spielen und Demoprogrammen
- » Programmierung über WLAN oder USB
- » MQTT, HTTP-/HTTPS-Kommunikation

Preis: 54,90 Euro

Pixelmatrix macht's einfach

Die Pixelmatrix hat den Vorteil, dass auf der Karte bereits fünf fix verdrahtete LED-Streifen existieren, die bereits untereinander verbunden sind. Wir müssen diese nur herausbrechen und neu verdrahten. Dafür sind insgesamt 12 Kabel notwendig, um GND, VDD und den Signalpin der fünf Streifen zu verbinden. Da wir so auch wieder alle 25 Pixel der Original-Matrix verwenden, können wir für erste Tests sogar die bereits existierende Firmware der Cartridge verwenden und müssen nichts neu programmieren.

Bild 1: Die Pixelmatrix-Cartridge enthält fünf fertig verdrahtete LED-Pixel-Streifen.

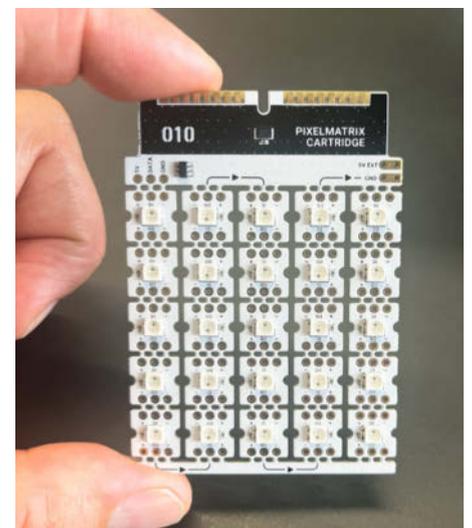




Bild 2: Überblick der 3D-Konstruktion des Sterns mit verschraubtem Gehäuse und optionalem Adapter

Gehäuse drucken

Damit die Leuchtkraft möglichst voll zur Geltung kommt, sollte man die vier Teile des Gehäuses in Weiß drucken. Als Material eignet sich PLA oder vorzugsweise PETG. PETG ist wegen des höheren Schmelzpunkts PLA vorzuziehen. Alle Teile stehen im Download-Bereich in STL-Form zur Verfügung. Zudem steht das Modell auch im STEP-Format bereit. Damit lassen sich im Nachhinein auch geometrische Anpassungen vornehmen. Die Teile `back.stl`, `top.stl` und `pcbholder.stl` werden direkt auf der Druckplatte ohne Raft und Stützstrukturen gedruckt. Lediglich das Teil `adapter.stl` ist mit Stützstrukturen zu drucken. Die auf den Bildern zu sehenden Teile wurden mit einer Schichthöhe von 0.2 mm auf einem Prusa MK4 und einem Prusa XL mit weißem PLA gedruckt.

Löten

Das Teil `pcbholder.stl` wird separat gedruckt und kann zum Löten auf der Basisplatte verbleiben. Schnelles Löten vermeidet dabei eine Überhitzung der empfindlichen LEDs.

Die LED-Streifen haben eine Richtungsangabe auf der Rückseite, die angibt, wie das Signal durch die LEDs geleitet wird. Beim kürzeren Arm setzen wir den Streifen so ein, dass die Pfeile nach innen zeigen, bei den anderen vier zeigen die Pfeile nach außen.

Dann fixieren wir das Ganze mit Klebestreifen und konfektionieren uns ein paar Kabel. In unserem Beispielaufbau verwenden wir Rot für VDD, Schwarz für GND und Grün für das Signal.

Die Bilder 5 bis 8 zeigen, wie gelötet werden muss. Es ist darauf zu achten, dass die LEDs nicht zu heiß werden. Damit die Platine nur möglichst kurz mit der Hitze in Berührung kommt, lohnt es sich, alle Kabel vorher zu verzinnen. Auch auf der Platine setzen wir einen kleinen Lötspitzen, wechseln nach jedem Punkt jedoch die Platine, damit nie zu lange Hitze auf die Elektronik wirkt.

Jetzt ist nur noch die Platine mit der Cartridge zu verbinden. Im Beispiel wurde dazu ein altes USB-Kabel verwendet, bei dem der Stecker defekt war. Diese enthalten in der Regel vier Adern, wobei wir für den Stern nur drei brauchen. Die Punkte GND/VDD/DATA+ sind mit dem Gegenstück auf der Cartridge zu verbinden.

Zusammenbau und Test

Vor dem finalen Zusammenbau empfiehlt sich ein Funktionstest. Hierzu verbinden wir die Oxocard Connect mit einer Powerbank und stecken die Cartridge in den Slot. Wenn alle LEDs leuchten, haben wir alles richtig gemacht. Falls ein oder mehrere Streifen nicht leuchten, kann es nur an einer fehlerhaften Verbindung liegen. Entsprechend sind alle

Lötverbindungen bei gelöstem Stecker auf Kontakt zu prüfen.

Wenn alles einwandfrei funktioniert, kann es an den Zusammenbau gehen. Als Erstes verkleben wir die Rückseite mit Klebeband, damit später nichts verrutscht, wenn wir es in das Gehäuse legen.

Danach können wir das Innenteil vorsichtig von der Druckplatte lösen und in die Unterseite des Gehäuses stecken. Mit etwas Heißkleber fixieren wir das Zuleitungskabel mit dem Gehäuse. Damit schaffen wir eine kleine Zugentlastung, um die Lötstelle zu schützen.

Als letzten Schritt setzen wir das Gehäuse-oberteil auf und verschrauben die beiden Teile. Dabei ist darauf zu achten, dass die beiden Ösen beim Kabel sauber einrasten, bevor alles mit den Schrauben angezogen wird.

Mit dem zusätzlich zu druckenden Adapter kann man den Stern beispielsweise an einer Tannenspitze befestigen. Mit doppelseitigem Klebeband hält der Stern auch an einer Wand oder einer Tür, und natürlich eignet sich der Stern auch hervorragend als Tischdekoration zu Weihnachten.

Cooler Effekte programmieren

Alle LEDs sind einzeln adressierbar und können verschiedene Animationen wiedergeben. Dabei ist häufig weniger mehr. Wir zeigen, wie man coole Regenbogen- und Glitzereffekte umsetzen kann.

Beim Regenbogen verwenden wir den HSV-Farbraum. Der HSV-Farbraum beschreibt Farben durch Helligkeit (Value), Sättigung (Saturation) und Farbton (Hue). In diesem wird die Farbe nicht in Form von Rot-, Grün- und Blautönen gemischt. Stattdessen wählen wir eine Grundfarbe, die uns ein Farbspektrum liefert. Mit zwei weiteren Parametern beeinflussen wir die Sättigung und Stärke der Farbe sowie deren Helligkeit.

Das Programm arbeitet so, dass wir für die erste LED eine Farbe wählen, die durch eine Zahl zwischen 0 und 255 bestimmt ist. Die nächste LED erhält eine Farbe etwas weiter rechts vom Farbspektrum und so weiter. Beim nächsten Durchgang verschieben wir die Farbnnummer der ersten LED um den Wert 1. So entsteht der Effekt eines rollenden Regenbogens. Setzt man hier andere Schrittgrößen ein, ändert sich der Effekt entsprechend.

Durch Klick auf „Code ausführen“ in der Toolbar wird das Programm sofort gestartet. Möchte man, dass das Programm sofort nach dem Einstecken der Oxocard startet, sollte man es über den Eintrag „Auf Cartridge laden“ im Kontextmenü vorher in den Flash-Speicher laden. Die Konstanten für Helligkeit (BRIGHTNESS) und Geschwindigkeit (ANIMATION_DELAY) lassen sich dynamisch zur Laufzeit über die Schieberegler der Entwicklungsumgebung verändern.

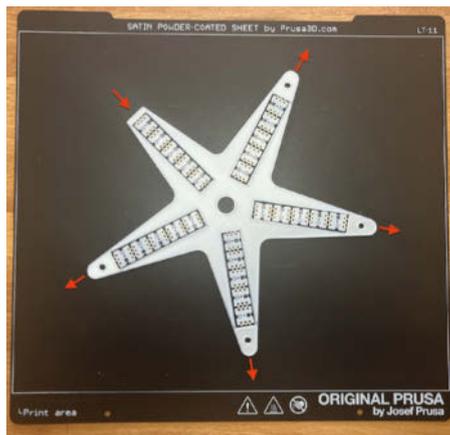


Bild 3: Achtung! Die roten Pfeile geben an, wie die LED-Streifen auszurichten sind.

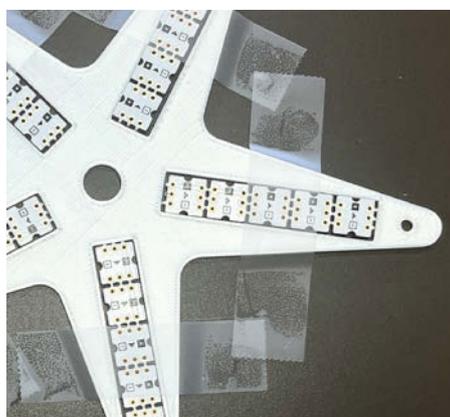


Bild 4: Bevor es ans Löten geht, wird alles mit Klebestreifen fixiert.

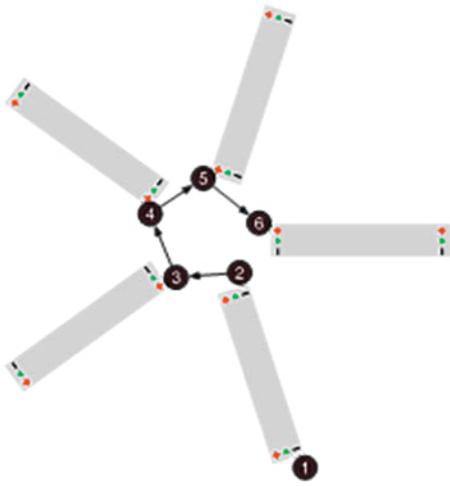


Bild 5: Zuerst löten wir die Ground-Pins.

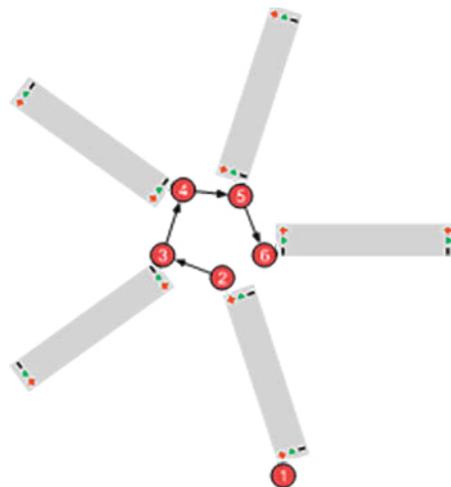


Bild 6: Dann löten wir die VDD-Versorgung.

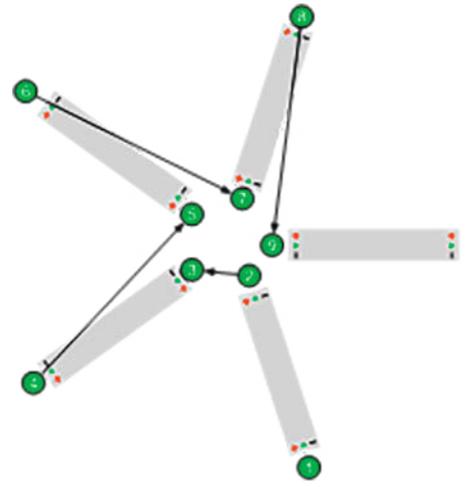


Bild 7: Zuletzt kommen die Datenpins. Mit diesen Kabeln verbinden wir die LEDs zu einer Reihe.



Bild 8: Als Zuleitung dient ein ausgedientes USB-Kabel.



Bild 9: Bevor es ans Zusammenbauen geht, verkleben wir die Rückseite noch mit Isolierband.

TECHNIKUNTERRICHT MACHT ENDLICH SPAß!



Make:Education

Mit **Make Education** erhalten Sie jeden Monat kostenlose Bauberichte und Schritt-für-Schritt-Anleitungen für einen praxisorientierten Unterricht:



Für alle weiterführenden Schulen



Fächerübergreifend



Digital zum Downloaden



Monatlicher Newsletter

© Copyright by Maker Media GmbH.

Jetzt kostenlos downloaden: make-magazin.de/education

```

Rainbow

1 const BRIGHTNESS      = 100      # 0 .. 255
2 const ANIMATION_DELAY = 50       # 0 .. 100
3 const NUMBER_OF_LEDS  = 25
4
5 initDigitalLeds(C_PIN_01, NUMBER_OF_LEDS, C_LED_TYPE_WS2812)
6 setDigitalLedMode(C_LED_MODE_GRB)
7
8 j:byte
9
10 #wifiPause()
11
12 while true:
13   for i in NUMBER_OF_LEDS:
14     c:color
15     c.hsv(j+(i*3),255,BRIGHTNESS)
16     setDigitalLed(i, c.r,c.g,c.b)
17   j++
18   applyDigitalLeds()
19   delay(ANIMATION_DELAY)
20   if getButton() != 0:
21     for i in NUMBER_OF_LEDS:
22       setDigitalLed(i, 0,0,0)
23       applyDigitalLeds()
24       delay(1000)
25     #wifiResume()
26     exit()
    
```

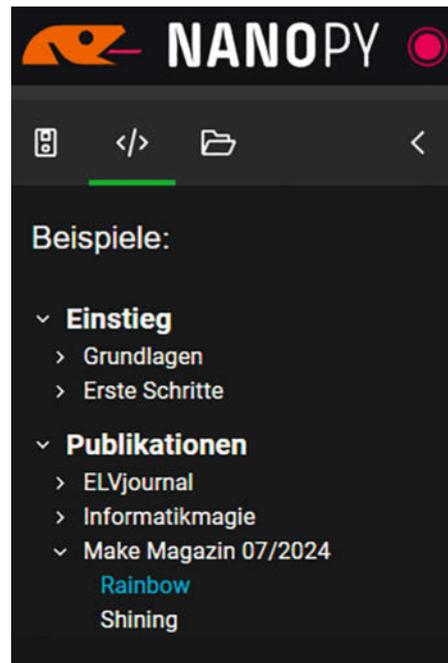


Bild 11: Das Skript kann man direkt unter „Beispiele > Publikationen > Make Magazin 07/2024 > Rainbow“ in der NanoPy-IDE herunterladen.

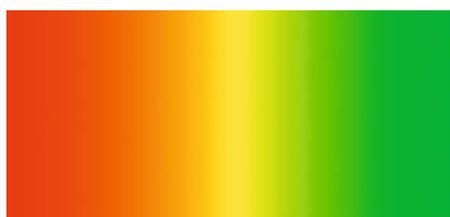


Bild 10: Die Grundfarben des HSV-Farbspektrums liegen im Wertebereich von 0 bis 255.

Hinweise zum Rainbow-Skriptcode

In den ersten beiden Zeilen werden die Helligkeit und die Geschwindigkeit der Animationen definiert. Dies lässt sich direkt im Code oder alternativ über den Konstanteneditor mit den dynamisch arbeitenden Schiebereglern einstellen.

In den Zeilen 3 bis 6 wird die LED-Bibliothek initialisiert. Hier legen wir den Schaltpin, die Anzahl der LEDs und den Controllertyp fest. Die eigentliche Programmlogik befindet sich in den Zeilen 12 bis 19. Innerhalb der Endlosschleife iterieren wir mit einer for-Schleife durch die LEDs. Bei jedem Durchlauf wird eine neue Grundfarbe aus dem HSV-Farbraum nach der Formel $j+(i*3)$ berechnet. i enthält die Nummer der aktuellen LED. Dies bewirkt, dass sich der Farbwert zwischen zwei LEDs jeweils um 3 ändert. Da j bei jedem Durchlauf um eins erhöht wird, verschiebt sich der Wert für den Farbton (Hue) und das Ganze wird als fließendes Farbspektrum wahrgenommen.

`setDigitalLed` gibt die Farbe auf der aktuellen LED aus. In Zeile 18 senden wir die neuen Farbdaten an die LEDs, danach warten wir einige Millisekunden, bis die Schleife wieder startet. Im Codeblock ab Zeile 20 bis 26

wird geprüft, ob der Benutzer eine Taste gedrückt hat. Ist dies der Fall, werden alle LEDs ausgeschaltet und das Programm wird anschließend beendet.

Leuchtender Stern

Im zweiten Beispiel sollen die LEDs unterschiedlich hell leuchten. Dazu verwenden wir für jede LED ein Objekt (Klasse Photon). Ein Photon hat eine bestimmte Helligkeit. Die Animationslogik erhöht in jedem Animationschritt die Helligkeit der LED bis zum hinterlegten Maximalwert und reduziert diese anschließend wieder auf null. Nach einem Durchlauf definieren wir für die LED dann eine neue maximale Helligkeit. Wenn wir dies für alle 25 LEDs unabhängig voneinander machen, ergibt sich ein abwechselnder Effekt, den wir mit den Konstanten `BRIGHTNESS`, `ANIMATION_DELAY` und `DX` beeinflussen können.

Das Skript ist direkt über den Menüpunkt „Beispiele > Publikationen > Make Magazin 07/2024 > Shining“ zu laden.

Erläuterungen zum Shining-Skript

In den Zeilen 7–22 wird die Photon-Klasse deklariert. Mit `init` initialisieren wir Photon. Als Erstes setzen wir hier eine maximale Helligkeit (Variable `maxValue`). In Zeile 27 definieren wir mit `NUMBER_OF_LEDS` insgesamt 25 Photon-Objekte, wobei jeder LED ein Objekt zugewiesen wird. In der Schleife ab Zeile 34 iterieren wir wieder durch jede LED.

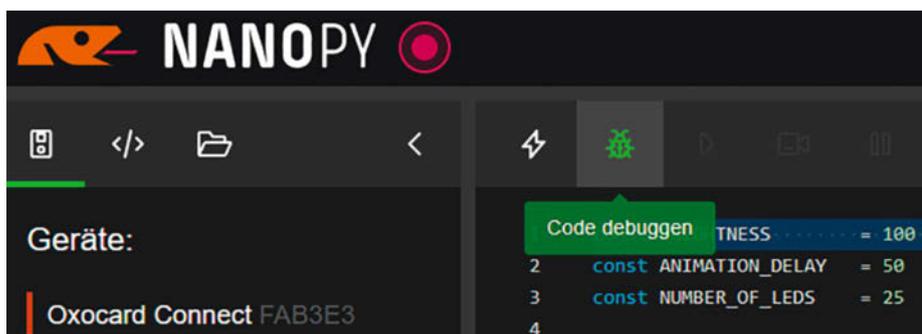


Bild 12: Mit dem integrierten Debugger lassen sich die einzelnen Programmschritte und deren Auswirkung auf den Stern Schritt für Schritt nachvollziehen.

Shining

```

1 const NUMBER_OF_LEDS      = 25
2
3 const ANIMATION_DELAY     = 10 # 0 .. 100
4 const BRIGHTNESS         = 100 # 50 .. 255
5 const DX                   = 3 # 1..20
6
7 class Photon:
8     current:byte
9     maxValue:byte
10    dx:int
11
12    def init():
13        maxValue = random(0,BRIGHTNESS)
14        dx=DX
15
16    def step():
17        if dx > 0:
18            if current < maxValue: current+=dx
19            else: dx=-DX
20        if dx < 0:
21            if current > 0: current+=dx
22            else: init()
23
24    initDigitalLeds(C_PIN_01, NUMBER_OF_LEDS, C_LED_TYPE_WS2812)
25    setDigitalLedMode(C_LED_MODE_GRB)
26
27    photons:Photon[NUMBER_OF_LEDS]
28
29    for i in NUMBER_OF_LEDS:
30        photons[i].init
31
32    #wifiPause()
33
34    while true:
35        for i in NUMBER_OF_LEDS:
36            val = photons[i].current
37            setDigitalLed(i, val,val,val)
38            photons[i].step()
39
40        applyDigitalLeds()
41        delay(ANIMATION_DELAY)
42        if getButton() != 0:
43            for i in NUMBER_OF_LEDS:
44                setDigitalLed(i,0,0,0)
45                applyDigitalLeds()
46                delay(1000)
47            #wifiResume()
48            exit()

```

Bei jedem Durchlauf lesen wir den aktuellen Helligkeitswert (`photons[i].current`) und schreiben den Wert in Zeile 37 in den LED-Buffer. Zeile 38 ruft die `step()`-Routine auf, die in den Zeilen 16 bis 22 bei jedem Aufruf die Variable `current` ändert. Diese enthält die aktuelle Helligkeit. Zuerst erhöhen wir bis zum Maximalwert und reduzieren dann wieder auf 0. Sobald dieser Zustand erreicht ist, initialisieren wir das `Photon`-Objekt mit einem neuen Maximalwert.

Tipps & Tricks

Es lohnt sich, zuerst die fertigen Beispiele auszuprobieren und mit den Konstanten zu experimentieren. In der Entwicklungsumgebung gibt es unter dem Eintrag „Cartridges > Pixelmatrix“ etwas Hintergrundinformationen zur Verwendung der LEDs sowie Inspirationen für eigene Weiterentwicklungen.

Je nach WLAN-Qualität kann es sein, dass die LEDs etwas flackern. Das kann man umgehen, indem die WLAN-Kommunikation während der Animation ausgeschaltet wird. In den Skripten gibt es zwei auskommentierte Zeilen (Rainbow: Zeilen 10 und 25, Shining: Zeilen 32 und 47). Wird dort das Kommentarzeichen (#) entfernt, läuft die Animation flüssiger. Allerdings ist dann das Programm immer erst mit einem Tastendruck auf der Connect zu beenden, um wieder eine Verbindung mit dem Editor zu bekommen.

Fazit

Der Stern ist schnell gebaut und ein toller Blickfang. Nach dem gleichen Prinzip lassen sich über andere Gehäuseformen auch völlig andere Dekorationsobjekte zum Leuchten bringen. Wir sind gespannt auf eure Weiterentwicklungen! —usz

E-Books im heise shop

Jetzt viele Titel als **ePub, mobi und PDF** erhältlich.

Sofort im Zugriff, dauerhaft in Ihrem Account gespeichert.

Raspberry Pi dummies

Python lernen kurz & gut

Twitch dummies

Kryptografie in der Praxis

Security Awareness dummies

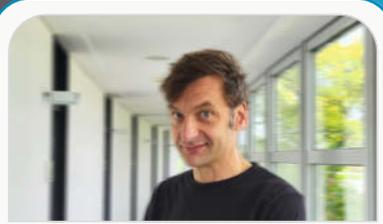
shop.heise.de/e-books

Generell portofreie Lieferung für Heise Medien- oder Maker Media Zeitschriften-Abonnenten oder ab einem Einkaufswert von 20 € (innerhalb Deutschlands). Nur solange der Vorrat reicht. Preisänderungen vorbehalten. E-Books können einem DRM-Schutz unterliegen.



Daniel Bachfeld
 Chefredakteur, dab@make-magazin.de

Mich haben immer der Babelfisch aus „Per Anhalter durch die Galaxis“ und der brabbelnde Taschencomputer Sam aus dem Sci-Fi-Hörspiel „Jonas, der letzte Detektiv“ fasziniert. Mittlerweile ist beides eigentlich keine Science-Fiction mehr, sondern in Smartphones und Earbuds vereint. „Sam, bitte das Rezept für den Pangalaktischen Donnergurgler!“



Marcus Hansson
 Redakteur, mch@make-magazin.de

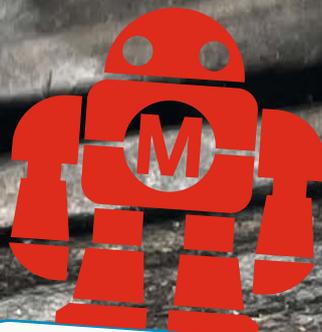
Eine Portal-Gun wie aus „Rick and Morty“, die per Knopfdruck schleimgrüne Portale zu anderen Welten und Realitäten öffnet, würde ich sehr, sehr gerne nachbauen. Optisch erinnert sie an einen Supermarkt-Selbstscanner und ermöglicht die Flucht vor Gefahren, Aliens oder Langeweile. Und genau dafür würde ich sie nutzen.



make:

Welche Sci-Fi-Technologie würdest Du am liebsten nachbauen und wofür würdest Du sie nutzen?

Unser Team




Carsten Wartmann
 Redakteur, caw@make-magazin.de

An sich gar keine. Mit einem Warp-Antrieb würden wir nur den Krieg in das Universum tragen, mit Replikatoren verschmutzen wir unseren Planeten noch schneller und die KI schafft in den meisten Fällen nur die Menschheit ab. Können wir uns vielleicht auf Techniken besinnen, die nicht auf endloses Wachstum ausgerichtet sind?



Akos Fodor
 Redakteur, akf@make-magazin.de

Ich würde einen Dinopark errichten und für den unwahrscheinlichen Fall, dass etwas schiefläuft, vier Bewegungsmelder aus der Alien-Filmreihe bauen, die in alle Himmelsrichtungen scannen. Dann könnten mich die Velociraptoren sicher nie von der Seite überraschen.



Johannes Börnsen
 Redakteur und YouTube-Host, jom@make-magazin.de

Ich hätte gerne eine analoge Halluzinelle in der Werkstatt, wie sie in der Serie „Ijon Tichy: Raumpilot“ an Bord ist. Also einen Assistenten, der die Projekte begleitet und in der Lage ist, zu helfen. Ich fürchte allerdings, dass wir dem in den letzten zwei Jahren schon deutlich näher gekommen sind, als mir tatsächlich lieb ist.





Daniel Schwabe
 Technical Writer,
 das@make-magazin.de

Seth Brundles Teleporter. Schnell mal eben einen Abstecher zur Lieblingseisdiele auf der anderen Seite der Erde machen oder den vermeintlichen Stress bei Urlauben, dass man auch ja nichts vergisst, stelle ich mir sehr angenehm vor. Allerdings würde ich dann versuchen, das Gerät kleiner umzusetzen. Mit einem ESP geht das bestimmt.



Dunia Selman
 Social-Media-Managerin,
 dus@make-magazin.de

Ich würde definitiv eine Zeitverlangsamungs-Technologie nachbauen. Der Tag hat immer zu wenige Stunden. So könnte ich nach der Arbeit noch so viel mehr schaffen. Ob Hobbys, Sport, Einkaufen oder einfach mal in Ruhe ein Buch lesen, während alles um mich herum in Zeitlupe läuft. Das wäre so genial.



Nicole Wesche
 Mediengestalterin, niwe@heise.de

Als völlig unbegabte Bastlerin werde ich bestimmt nichts selbst nachbauen, aber sicherlich wird sich einer meiner netten Make-Kollegen dazu bereit erklären, mir so ein Blitzdingens zu bauen. Als Quatschtante würde mir das sehr zugutekommen, wenn ich, mit einem Klick – aus Versehen – Gesagtes vergessen machen könnte!



Daniel Rohlfing
 Leiter Events und Sales,
 dnr@maker-faire.de

Nachbauen nicht, aber nutzen. Definitiv einen Teleporter. Das würde meine regelmäßigen Ortswechsel zwischen Hannover und Köln deutlich vereinfachen und man könnte sich häufiger sehen. Cool wäre es zudem, sich in der Mittagspause mal eben an einen sonnigen Strand zu beamen, um eine halbe Stunde das Meer und die Wärme zu genießen.



Kristina Fischer
 Projektleitung Maker Faire,
 krfi@maker-faire.de

Mein Held ist Wall-E. Ultra süß und extrem hilfreich. Müll ist ein bekanntes Problem auf der Welt, bisher gibt es keine Lösung, dieses Problem sinnvoll anzugehen. Im besten Fall löst mein selbst gebauter Aufräumberoter dieses direkt. Ansonsten habe ich wenigstens einen süßen Mitbewohner, der mir den Müll nach draußen trägt.



Ulrich Schmitz
 Redakteur, usz@make-magazin.de

Meine Vision wäre der Nachbau eines Nahrungsmittel-Replikators aus dem Star-Trek-Universum. Basis dafür könnte 3D-Druckertechnik in Verbindung mit Nahrungspasten, Schokolade, Süßigkeiten etc. sein. Zweck: Wenig reizvolle Pastennahrung durch essbare Formen „schmackhaft“ machen bzw. Verzierung von Süßspeisen, Erstellung von Fruchtgummis usw.



Nele Krautberger
 Werkstudentin Maker Faire,
 nekr@maker-faire.de

Ich würde gerne die Technologie des Autos aus dem Film „Back to the Future“ nutzen können. Ich würde damit in die verschiedenen Zeitalter reisen und mir die Städte und Handwerkskünste ansehen. Außerdem würde ich die Technologie dazu nutzen, ins alte Ägypten zu reisen, um herauszufinden, wie diese die Pyramiden erbaut haben.



Make:



Rebecca Poweleit
 Werkstudentin Social Media,
 repo@make-magazin.de

Ich würde mich gerne beamen können, weil es super praktisch wäre – dann könnte ich die Welt bereisen und wäre nicht immer chronisch zu spät. Leider glaube ich nicht, dass es diese Technik je geben wird. Sie ist sehr komplex, und die Vorstellung, in Teile zerlegt und in einzelnen Informationen übertragen zu werden, ist gruselig.

Laufmaschinen konstruieren

Ein lebensgroßer mechanischer Strauß, der sich auf metallenen Beinen fortbewegt und dabei eine Kutsche hinter sich herzieht, erregt überall viel Aufmerksamkeit – sei es beim Burning-Man-Festival oder zu Hause auf der Straße. Mithilfe eines relativ einfachen kinetischen Prinzips lässt sich eine naturgetreue Laufbewegung erzeugen. Wir zeigen, wie der Strauß konstruiert ist und wie er sich nachbauen lässt.

von Marcus Hansson



Vor zwei Jahren stieß ich auf YouTube zufällig auf einen Film, der mich sofort in den Bann zog. Ein Mann in den USA hatte ein seltsames Geschöpf gebaut – eine ein Meter hohe Ente aus Metall, die eine kleine, zweirädrige Kutsche mit dem Mann darauf hinter sich her zog (Bild 1). In dem Video war zu sehen, wie das Vogelgeschöpf mit langsamen, unsicheren Schritten vorwärts watschelte.

Das Besondere an dem mechanischen Tier ist die Art und Weise, wie die Kraft vom Motor auf die beiden Metallbeine übertragen wird. Die Bewegungen haben etwas leicht Unheimliches, wie eine Mischung aus Frankenstein und Tier. Bald entdeckte ich mehrere Videos von ähnlichen zweibeinigen Laufmaschinen und entschied, selbst ein solches mechanisches Wesen zu bauen. Um meine Faszination dafür nachvollziehen zu können, lohnt sich ein Blick auf die Videos, die ihr unter dem Link in der Kurzinformatik findet.

Entstanden ist eine Straußendame, die den Namen Betty trägt (Bild 2, 3 und 4). Das Projekt hat sich über zwei Jahre entwickelt und im Jahr 2024 ist sie, wie es sich gehört, mit beweglichen Flügeln, einem beweglichen Schnabel und jeder Menge LEDs ausgestattet, damit sie nachts so richtig strahlen kann. Unter ihrem gefederten Hinterteil verbirgt sich außerdem eine Eierlegemaschine (Bild 5). Auf dem Nowhere-Festival in Spanien war Betty bereits zweimal und hat dort für viel Aufsehen gesorgt. Das Leben in der Wüste ist für eine mechanische Straußendame jedoch sehr hart. Staub, Regen, pralle Sonne und Mangel an passenden Ersatzteilen kosteten viel Kraft und brachten einige bittere Lernerfahrungen mit sich.

Bettys erste Zeit in der Welt war also alles andere als einfach, und das Projekt hat unzählige Arbeitsstunden verschlungen. Sie musste viele Kinderkrankheiten überstehen



Bild 3: Flügel, Schnabel, Eierlegemaschine, LED-Beleuchtung und Lenkung lassen sich über das Steuerpult auf der Kutsche bedienen.



Bild 1: Das Video von YouTube-User Jeep2003 wurde über 41 Millionen Mal angesehen.

Bild 2: Auch auf der Maker Faire Hannover 2024 war Betty unterwegs.

Kurzinformatik

- » Bauweise einer zweibeinigen kinetischen Skulptur, die eine Kutsche zieht
- » Funktionsprinzip der Kinematik der Laufbewegung
- » Antrieb mit E-Bike-Umbausatz



Checkliste



Zeitaufwand:
Mehrere Wochenenden



Kosten:
ab 500 Euro

Material

- » Altes Fahrrad 26" oder 28"
- » Rechteckrohre aus Stahl
20 × 40 mm
- » Stahlrohr 38 mm
- » 10 St. Kugellager, z. B. 6200
- » Eine Kutsche oder einen einfachen Sulky
- » E-Bike-Umbausatz für Hinterrabenantrieb
- » 36- oder 48-Volt-Akku passend zum E-Bike-Motor
- » Linearmotor für die Lenkung

Werkzeug

- » Schweißgerät
- » Flex
- » 38 mm Lochsäge für Stahl
- » Säulenbohrmaschine

Mehr zum Thema

- » Marcus Hansson, Ein Maker-Paradies in der Wüste, Make 5/24, S. 64
- » Joachim Haas, Der Zwölfender, Make 1/16, S. 16
- » Moritz Metz, Einstieg ins Schweißen, Make Sonderheft 2021, S. 104

Alles zum Artikel im Web unter make-magazin.de/xgx1





Bild 4: Die augenförmigen Drahtnetze an den Seiten verhindern, dass sich jemand die Finger einklemmt. Ganz vorne (rechts im Bild) ist die 48V-Batterie positioniert.

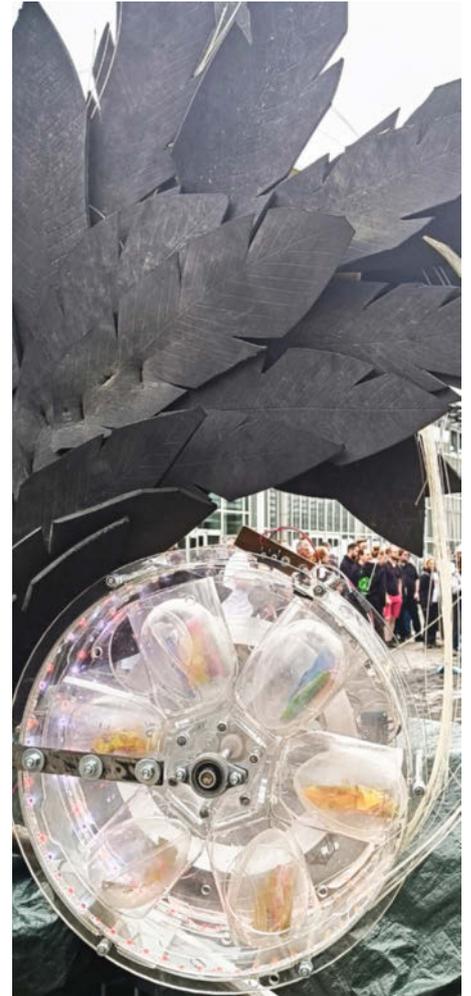


Bild 5: Die Eierlegemaschine bietet Platz für sechs 10 cm große Acryl-Eier.

die Funktion und den Aufbau der Eierlegemaschine eingehen.

Die Laufbewegung

Das Geschöpf im ursprünglichen Film (Bild 1) war zwar schön anzusehen, erschien mir je-

doch sehr komplex und aufwändig in der Konstruktion. Zudem erzeugt der Verbrennungsmotor Abgase und Lärm. Nach weiterer Recherche fand ich auf YouTube ähnliche kinetische Wesen, deren Aufbau viel einfacher war und die mir als Inspiration für meinen eigenen Versuch dienen (Bild 6 und 7).

Was alle diese Projekte gemeinsam haben, ist das Bewegungsprinzip. Es basiert auf dem sogenannten Gelenkvierneck, einer Art „kinematischer Kette“, die aus nur vier Gliedern besteht, die an den Enden miteinander verbunden sind (siehe Bild 8). Wenn eines dieser vier Glieder fest verbaut ist, wird es als Steg



Bild 6: Eine indische Laufmaschine



Bild 7: Eine russische pedalbetriebene Laufmaschine

bezeichnet (A in Bild 8). Die beiden angrenzenden Seiten nennt man Arme (B und C), und die vierte Seite wird Koppel genannt (D).

Nehmen wir an, die Arme sind unterschiedlich lang, und wir greifen den kürzeren der beiden Arme (C) und drehen ihn im Kreis um den festen Punkt, an dem das Steg-Glied befestigt ist, dann müssen sich zwangsläufig die beiden anderen frei beweglichen Teile – der andere Arm und die Koppel – ebenfalls mitdrehen. Von der Koppel geht ein weiteres fest verbundenes Glied nach unten (E in Bild 8). Dieses stellt unser Bein dar, dessen untere Spitze den Fuß bildet. Dabei entsteht ein sehr interessantes Bewegungsmuster (die hellblau gefärbte Spur in Bild 8). Die untere flache Strecke stellt den Abschnitt dar, in dem unser Fuß Bodenkontakt hat und somit den Schritt nach vorne trägt.

Entscheidend für die Funktionsfähigkeit ist, dass der zweite Arm länger als der von uns bewegte Arm ist. Andernfalls kann der kürzere Arm keine volle Umdrehung vollziehen, sondern nur hin- und herschlagen. Hierbei spielt die sogenannte Grashofsche Regel eine wichtige Rolle, die vom deutschen Ingenieur Franz Grashof im 19. Jahrhundert entdeckt wurde. Nach der Grashofschen Regel kann das kürzeste Glied eine volle Umdrehung nur durchführen, wenn seine Länge zusammen mit der des längsten Glieds gleich oder kürzer ist als die Summe der Längen der beiden anderen Glieder. Varianten mit verschiedenen Längenverhältnissen lassen sich beispielsweise in Tinkercad simulieren (siehe Link in der Kurzinfor).

Ich empfehle einen Blick auf die Videos, die unter dem Link in der Kurzinfor zu finden sind. Bewegte Bilder machen es viel einfacher, die Funktionsweise dieses Gelenkvierecks zu verstehen.

Nachbau mit Fahrradteilen

Ich bin von einem 26-Zoll-Mountainbike-Rahmen ausgegangen, den ich für 30 Euro gekauft habe. Dieses Vorgehen bietet viele Vorteile: Eine Kurbel mit Ritzel für die Antriebskette ist bereits vorhanden, und der Rahmen ist stabil. Zudem habe ich ein E-Bike-Umrüstkit als Motor eingebaut. Mehr dazu später.

Hauptsächlich habe ich mit Rechteckrohren in den Abmessungen 20 × 40 mm und 40 × 40 mm sowie mit 38 mm Rundrohren gearbeitet. Mit einer Wandstärke von 1,5 mm passen dann 6200er Kugellager perfekt hinein, da sie eine Höhe von 35 mm und eine Breite von 10 mm haben. Dazu noch eine Achse mit 10 mm Durchmesser und jede Menge M10-Bolzen.

Beginnen wir mit der Konstruktion der oben beschriebenen Laufbewegung (Bild 9), die von der vorhandenen Kurbel ausgeht. Weiter vorne brauchen wir noch eine zweite Achse. Dazu habe ich an der Stelle, an der früher das Lenkrad

saß, auf jeder Seite der Gabelaufnahme ein 4 mm dickes Stahlplättchen angeschweißt. Im rechten Winkel zu den beiden Platten habe ich eine 10 mm Achse, die später die vorderen Arme aufnehmen soll, eingebaut.

Diese Achse befindet sich nun etwa einen halben Meter vor der Kurbelachse, und es ist wichtig, darauf zu achten, dass sie möglichst parallel zur hinteren Kurbelachse steht. Das untere Rohr des Fahrradrahmens bildet also unseren „Steg“ (A in Bild 8 und 9).

Die Arme

Ich nahm zwei gleich lange Rechteckrohre von 40 × 20 mm und schweißte sie mit einem Abstand von 20 mm zusammen (Bild 10). Auf einer Seite wurde ein Stück 38 mm Rohr zur Aufnahme der Kugellager eingeschweißt. Der Abstand zwischen den beiden Rechteckrohren ist notwendig, damit später das eigentliche Bein dazwischen eingehängt werden kann (Bild 11).

Fahrradkurbel

Nun widmen wir uns dem hinteren Arm, also der Fahrradkurbel (C in Bild 8 und 9). Ich habe die Pedale demontiert und auf die nackten Kurbelarme ein Stück 20 × 40 mm Rechteckrohr aufgesteckt und mit durchgehenden Bolzen befestigt (Bild 12). Aber warum wurde das Rechteckrohr dann seitlich nach außen verlängert? Die Tretkurbeln (ohne Pedale) eines Fahrrads haben nur einen seitlichen Abstand von ca. 16-18 cm. Wären die Beine so nah beieinander, wäre unser Strauß sehr wackelig. Bei meiner Konstruktion beträgt der Abstand

zwischen den Beinen 33 cm. Das entspricht auch der Länge der Vorderachse, denn die Beine müssen später in Längsrichtung parallel zueinander stehen.

Die Beine

Mit „Bein“ ist hier das Koppel (D in Bild 8 und 9) im Gelenkviereck gemeint, also das untere Verbindungsglied zwischen den beiden Armen, und das darauf fest verschweißte Rechteckrohr (Ein Bild 8 und 9), das ungefähr in einem Winkel von 85 Grad von der Koppel abgeht. Damit ist unsere Bein konstruktion fertiggestellt und wie in Bild 9 dargestellt zusammengebaut. Bild 13 zeigt den oberen Teil des Beins.

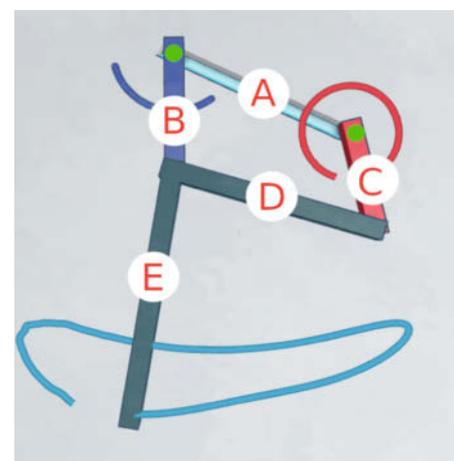


Bild 8: Das Gelenkviereck: Der kürzere Arm C führt volle Umdrehungen aus, während der längere Arm B nur hin- und herschwingt.



Bild 9: Hier ist die Grundkonstruktion mit dem Gelenkviereck klar zu erkennen. Die Achsen sind durch zwei rote Punkte gekennzeichnet. Wenn die Kurbel (C) gedreht wird, setzt sich die Koppel (D) in Bewegung. Zusammen mit E bildet sie das Bein.



Bild 10: Zwischen den beiden Stahlprofilen in dem „Doppelarm“ pendelt später das eigentliche Bein.



Bild 11: Hier ist einer von den beiden „Doppelarmen“ an der vorderen Achse aufgehängt.



Bild 13: Der obere Teil des Beins. Das Kugellager wird in dem „Doppelarm“ aufgehängt.



Bild 12: Die Pedalen sind entfernt und die Kurbel wurde mit Rechteckrohr verlängert.

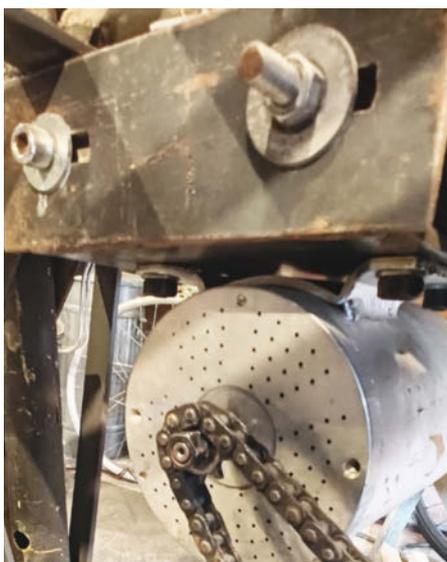


Bild 14: Die Antriebskette kann über Langlöcher gespannt werden, in dem der Go-Kart-Motor nach hinten geschoben wird.



Bild 15: Die obere Kette im Bild wird vom Motor angetrieben. Das große und kleine Ritzel in der Mitte sind auf einer Achse fest verschweißt, damit die Kraft an die Kurbel weitergegeben wird.

Der Antrieb

Wie schon zuvor erwähnt, habe ich mich anfangs für ein E-Bike-Umrüstkit als Antrieb entschieden. Für knapp 250 Euro erhielt ich einen Controller, einen Drehgriffregler, ein kleines LCD-Display und den 1000-Watt-starken Motor. Ein passender 48V-Akku kostete weitere 250 Euro. Da es sich um einen Hinterradantrieb handelt, war die Installation sehr einfach: Den E-Motor in die vorhandene Aufnahme der hinteren Radachse einsetzen und mit Nabenmuttern befestigen, wie in Bild 9 zu sehen ist.

Danach wurde es schwieriger: Die mitgelieferte achtgängige Kassette verfügte über einen Freilauf. Bei einem normalen Fahrrad wird die Kraft vom Motor direkt auf die Straße übertragen, indem sich das Rad um seinen Aufhängepunkt dreht. Dabei sorgt der Freilauf dafür, dass die Pedalen nicht mitdrehen. In meinem Fall wollte ich jedoch die Kraftübertragung auf die Kurbel haben, damit sich diese dreht und somit die Beine in Bewegung kommen. Das Rad hängt frei in der Luft und soll später hinter Vogelfedern verschwinden.

Bis zum Festival in Spanien waren es nur noch wenige Wochen. Schnell musste eine Lösung her. Also baute ich mit ein paar Stahlplatten und Bolzen eine Vorrichtung um den Nabenmotor herum, damit sich die Kasette nicht bewegen konnte. Dann kam das nächste Problem: Bei vorwärts laufendem Motor und fest montierter Kasette zog der Motor die Kette an der Unterseite. Dadurch konnte ich den vorhandenen Kettenspanner nicht mehr benutzen, um die Kette unter Spannung zu halten, da dieser wie bei Fahrrädern üblich, an der Unterseite der Kette montiert ist. Die Lösung war, den Kettenspanner an einem kleinen Arm auf der Oberseite der Kette zu befestigen. Da ich alles in sehr kurzer Zeit erledigen musste, war diese Lösung alles andere als optimal. Der schnell gebastelte „Kettenspanner“ war seitlich nicht ganz stabil, was später, in der Wüste, zu großem Ärger führen sollte.

Kette überstrapaziert

Durch den etwas holprigen Lauf des Vogels wurde die Antriebskette stark beansprucht, und durch zu wenig Spannung und schlechte Ausrichtung sprang die Kette immer wieder ab und verkeilte sich zwischen Fahrradrahmen und Ritzel, wo sie dann durch den starken Motor schnell abgerissen wurde. Im folgenden Jahr (2024) habe ich dann die ursprüngliche Konstruktion mit dem E-Bike-Umbausatz verworfen und auf einen 1800-Watt-48V-Go-Kart-Motor gesetzt (Bild 14). Das Schöne an diesem Motor ist, dass er auch rückwärts laufen kann. Leider war die hohe Drehzahl von bis zu 5000 U/min nicht optimal. Um mehr Kraft auf die Beine zu übertragen, habe ich deshalb ein Getriebe mit einer zweiten Kette eingebaut (Bild 15). Dadurch wurde die Konstruktion aber wesentlich komplizierter, unter anderem, weil jetzt zwei Ketten gespannt werden mussten. Eine klare Empfehlung meinerseits ist, sich einen Motor mit möglichst niedriger Drehzahl zu besorgen, z.B. von einem Golfcart oder Elektrorollstuhl. Insgesamt muss der Antriebsstrang gut durchdacht und stabil sein, denn auch wenn der Strauß langsam unterwegs ist, wird die Konstruktion stark beansprucht.

Das Problem mit der Lenkung

Nachdem im ersten Jahr Motor und Laufbewegung funktionierten, konnte ich eine erste Probefahrt unternehmen (Bild 16). Ich hatte auch eine alte, eingerostete Ponykutsche für 100 Euro gekauft. Die Reifen waren jedoch alle durch und mussten erneuert werden, und die hölzernen Sitzflächen waren teilweise morsch, aber mechanisch war alles in Ordnung. Also habe ich aus 20-mm-Rohr eine



Bild 16: Erste Testfahrt 2023: Die beiden bunten Punkte markieren, wo die Deichsel am Vogelstrauß fest verbaut ist.

Deichsel gebaut, denn ohne Deichsel hat der Strauß keinen Halt und fällt entweder auf die Seite, nach vorne oder nach hinten. Außerdem soll er ja die Kutsche ziehen.

Der ursprüngliche Plan sah vor, in die ehemalige Sattelstütze eine senkrechte, drehbare

Achse zu installieren, um den Strauß mit Riemen von der Sitzbank aus lenken zu können. Dies erwies sich jedoch bald als sehr schwierig, da die Sattelstütze weit vor dem Schwerpunkt des Straußes liegt. Das führte dazu, dass meine erste Deichselkonstruktion



Bild 17: Unterhalb der roten Linie befindet sich eine vertikale Achse, um die sich der Vogel innerhalb der Deichsel (blaue Linien) nach links und rechts drehen kann. Zur Sicherheit gibt es einen zweiten, beweglichen Aufnahmepunkt (roter Kreis).



Bild 18: Mit einem Servo lässt sich der Schnabel auf- und zuklappen.

trotz dicker Schweißnähte ziemlich schnell an der Stelle brach, an der sie an der Sattelstütze angedockt war (lila Punkt in Bild 16). Das Gewicht des hinteren Teils des Fahrrads und des dort verbauten E-Motors erzeugte zu starke Biegekräfte nach unten.

Ich befestigte die Deichsel nun weiter hinten am Fahrradrahmen, um die Biegekräfte aufzufangen (roter Punkt in Bild 16). Das funktionierte gut, hatte aber einen entscheidenden Nachteil: Da der Strauß jetzt an zwei hintereinander liegenden Punkten mit der Deichsel verbunden war, konnte sie sich nicht mehr drehen, was die Steuerfähigkeit stark einschränkte. Der Vogel konnte somit nur geradeaus laufen und war mit dieser Konstruktion fest an der vorderen (drehbaren) Achse der Kutsche befestigt.

Auf dem Festival in Spanien mussten dann freiwillige Helfer bei jeder Kursänderung den Strauß anheben und umsetzen. Das war oft lustig, aber wenig zufriedenstellend. Im darauffolgenden Jahr (2024) habe ich den genauen Gewichtsmittelpunkt des Straußes ermittelt und dort mit kräftigen Verstärkungen eine Gabelaufnahme von einem weiteren ausgeschlachteten Fahrrad installiert, eine drehbare Achse genau in der Mitte unter den Füßen (Bild 17). Mit dieser neuen Konstruktion kann der Vogel sich jetzt um seine eigene Achse drehen, und mit einem 12V-Linearantrieb lässt er sich über die Steuerkonsole der Kutsche lenken (Bild 3). Als Sicherheitsfeature habe ich, wie im ersten Jahr, einen zweiten Aufnahmepunkt eingebaut, diesmal jedoch beweglich (links in Bild 17 sichtbar). Jetzt kann sich Betty links und rechts innerhalb ihrer Deichsel frei bewegen, was notwendig ist, um die Richtung zu ändern.

Mit Kopf und Feder

Im zweiten Jahr habe ich in Zusammenarbeit mit Freunden Betty mit beweglichen und beleuchteten Flügeln ausgestattet. Mit zwei Linearmotoren lassen sich die Flügel ein- und ausfahren, was auf dem Festival für jede Menge Spaß sorgte. Lustig war auch die Eierlegemaschine (Bild 5), oder Eierrevolver, anfangs „Uterus“ genannt, bis Chefredakteur Daniel Bachfeld mich darauf aufmerksam machte, dass Vögel keinen Uterus haben, son-



Bild 19: Gefederte Füße sorgen für einen einigermaßen ruhigen Gang.

dern einen Legedarm und eine Kloake. Für die Eierlegemaschine habe ich sechs etwa zehn Zentimeter lange Plastikrohre mit einem Durchmesser von 100 mm kreisförmig angeordnet und anschließend zwischen zwei runden Holzplatten festgeklebt.

In die Rohren werden dann Acryleier eingelegt. Anschließend wird dieses „Revolvermagazin“ in eine Hülle gesteckt, die unten eine Öffnung aufweist. Beim Drehen des inneren Magazins kann dann ein Ei nach dem anderen gelegt werden. In einem Artikel in der nächsten Make-Ausgabe wird der Aufbau und die dazugehörige Steuerung ausführlich erklärt.

Ohne Kopf wirkte das Wesen nicht wirklich wie ein Strauß. Deshalb haben wir noch einen Kopf gebaut (Bild 18). Mithilfe eines Servos lässt sich der Schnabel öffnen und schließen. Zwei LED-Ringe dienen als Augen. Den ursprünglichen Hals fertigte ich aus einem 40-mm-HT-Rohr, das ich mit einem Heißluftföhn erwärmte und leicht verformte; anschließend wurde es rosa angemalt. Auch hier erwies sich die Konstruktion als nicht robust genug. Nach dem ersten Ausritt über die Playa auf dem Nowhere-Festival brach sich unsere Betty ihr Genick (Bild 4). Obwohl die Füße mit Federungen ausgestattet waren (Bild 19), bereiteten die heftigen Bewegungen dem dünnen Plastikrohr rasch ein Ende. Danach irrte Betty wie ein kopfloses Hühnchen herum.

Fazit

Die Straußendame Betty ist unter Zeitdruck und mit einem sehr begrenzten Budget entstanden, daher waren leider nicht alle Lösungen gut durchdacht. Aber das Gefühl, das mich überkam, als die Menschen zu Hause auf der Straße oder auf einem Festival dieses exotische Wesen mit seinen verrückten Laufbewegungen bestaunten, war die Mühe wert. Aus meiner Sicht lohnt sich ein Nachbau immer. Roboter oder animatronische Figuren aller Art wecken die Fantasie und regen zum Nachdenken an. Und sie machen unglaublich viel Spaß. —mch

Nachbau leicht gemacht

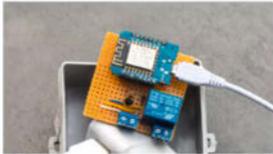
Wer von meiner Geschichte abgeschreckt wurde – keine Sorge, es geht auch einfacher. Hier sind einige Tipps und allgemeine Richtlinien für diejenigen, die gerne eine zweibeinige Laufmaschine als Zugtier bauen möchten.

- » Um eine ausreichend starke Lenkverbindung zwischen Kutsche und Zugtier zu gewährleisten, sollten zwei Kugellager mit einem möglichst großen Abstand zueinander, idealerweise 30 oder 40 Zentimeter, senkrecht in eine drehbare Achse eingebaut werden. Bild 6 (Indische Laufmaschine) veranschaulicht dies deutlich.
- » Ist das Zugtier zu leicht, kann es je nach Straßenbelag passieren, dass es nicht genügend Grip bekommt und durchrutscht. Daher ist es sinnvoll, schwere Teile wie den Akku und Motor möglichst nah an den Beinen zu platzieren. Achte auch darauf, dass das Gewicht in der Längsrichtung einigermaßen gut verteilt ist.

- » Anstelle eines Motors können auch Pedale verwendet werden, wie in Bild 7 (russisches Modell) zu sehen ist.
- » Wer mehrere alte Fahrräder zur Verfügung hat, kann sich viel Konstruktions- und Schweißarbeit ersparen. Eine zweite Kurbel kann für die Aufhängung der beiden Arme im Gelenkviereck (siehe oben) verwendet werden, eine dritte Kurbel als Pedalantrieb.
- » Der Motor sollte ein möglichst hohes Drehmoment und eine niedrige Drehzahl haben.
- » Wenn die Beine sehr lang sind, verlagert sich das Gewicht nach oben, was zu einer seitlichen Instabilität führt, besonders wenn Motor und Batterie im Körper eingebaut sind. Die Beine, wie sie in Bild 16 zu sehen sind, wurden später deutlich gekürzt.
- » Ich empfehle gefederte Füße (Bild 19), damit die Stöße des Körpers beim Aufsetzen des Fußes gedämpft werden.

Make: Online

Beliebt auf **heise +**



Automatisiert Strom sparen mit selbstgebautes Tibber-Relais

Als Tibber-Nutzer sparen Sie mit diesem selbstgebautes Tibber-Relais Zeit und Energie. Es steuert Geräte automatisch und findet den günstigsten Zeitpunkt dafür.

01.10.2024 252 Make Magazin



Festnetzanschluss mit Smartphone nutzen

Man kann ein Smartphone über den Festnetzanschluss nutzen. Das bringt Kostenvorteile für die anrufende Verwandtschaft und spart besonders im Ausland Gebühren.

15.10.2024 127 Make Magazin



Smarte Überwachung: Füllstand von Wassertanks messen und digital abfragen

Um den Füllstand von Tanks & Teichen zu kontrollieren, nutzt man Drucksensoren. Den Stand ruft man im Browser ab oder bindet ihn mit MQTT in Home Assistant ein.

17.09.2024 132 Make Magazin

So kommen Sie als Make-Abonnent an Artikel hinter der Paywall von heise+: <https://heise.de/-7363373>

ChatGPTs neuer Coding-Modus ist das perfekte Maker-Werkzeug!

Programmieren mit ChatGPT war bisher kompliziert und fehleranfällig. In diesem Video zeigt Johannes den neuen Programmiermodus von ChatGPT: „ChatGPT with Canvas“. Der Unterschied liegt vor allem in der Benutzeroberfläche: Statt der üblichen Chat-Umgebung ist die Oberfläche zweigeteilt – links der Chat, rechts der Code. Das erleichtert das Arbeiten am Code, ohne ständig durch den Chatverlauf scrollen zu müssen. Außerdem kann ChatGPT nun Anmerkungen direkt als Notiz im Code machen und die vorgeschlagenen Änderungen lassen sich mit einem Klick über-



nehmen. Damit spart man nicht nur Zeit, sondern man lernt durch die detaillierten Erklärungen von ChatGPT. —dus

► www.youtube.com/@MakeMagazinDE

Weitere aktuelle Videos:



Bleib informiert:

www.make-magazin.de

@makemagazinde



Instagram: @makemagazinde
@makerfaireddeutschland



Facebook: @makemagazin.de



X (Twitter): @MakeMagazinDE



TikTok: @makemagazinde



Bluesky: @makemagazin.de



WhatsApp Channel:
Make Magazin Deutschland



GitHub: MakeMagazinDE



Threads: @makemagazinde



LinkedIn: linkedin.com/company/maker-media-gmbh/



Kontakt zur Redaktion

Leserbriefe und Meinungen an:
heise.de/make/kontakt/



Oder diskutiere in unseren Foren
online über Themen und Artikel:
www.make-magazin.de/forum



Nichts mehr verpassen:
Abonniere unseren Newsletter!
Weitere Infos unter:
www.maker-faire.de

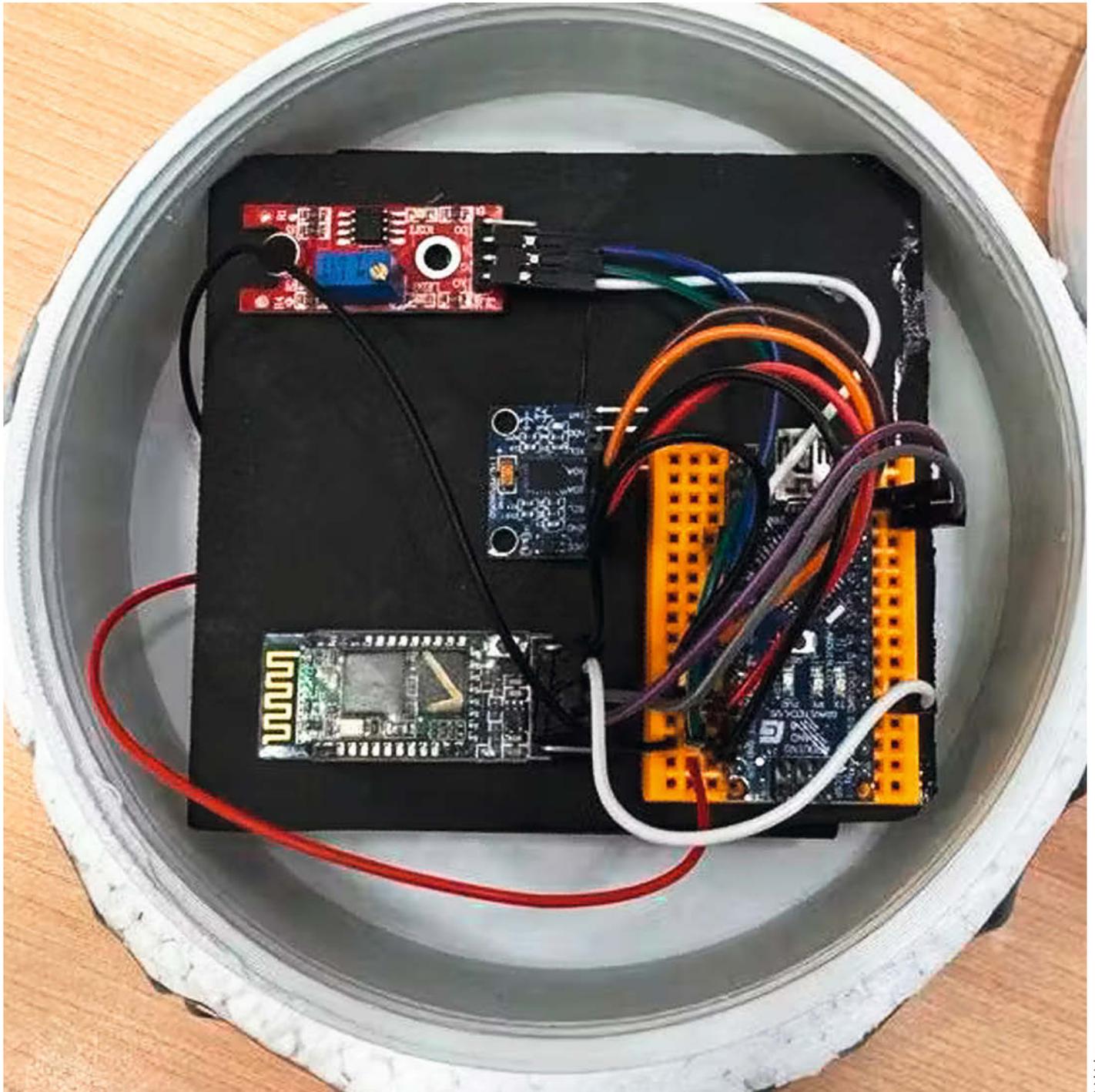


Wo finde ich die Make am Kiosk?
www.mykiosk.com

Poolüberwachung mit Arduino

Über Wasserbewegungen und den Geräuschpegel erkennt dieses Arduino-Projekt, ob jemand in den heimischen Pool oder Teich gefallen ist.

von Daniel Schwabe



Zaid Jabber

Als Eltern oder Tierbesitzer trägt man eine besondere Verantwortung, um sicherzustellen, dass Kinder oder Tiere niemals unbeaufsichtigt in der Nähe eines Pools oder Teichs spielen. Um der eigenen Wachsamkeit noch eine zusätzliche Sicherheitskomponente hinzuzufügen, kann man mit einem Arduino Uno, einem Bewegungssensor und einem Mikrofon ein Überwachungssystem für den privaten Pool oder Teich bauen.

Bei dem Projekt wird davon ausgegangen, dass es zu heftigen Wellenbewegungen führt und laute Geräusche verursacht, wenn jemand in einen Pool fällt.

Die Umsetzung besteht aus zwei Teilen: einem Schwimmer und einer Basisstation. Im Schwimmer befindet sich ein Arduino UNO, der Beschleunigungs- und Drehratensensor MPU-6050, der Mikrofonsensor VMA309 und das Bluetooth-Modul HC-05.

Die Technik ist verpackt in einem 3D-gedruckten Gehäuse, das auf einer Scheibe aus Styropor befestigt ist, um schwimmen zu können. Im Inneren des Schwimmers wird die Technik dann über ein Batteriepack mit Strom versorgt. Hier ist zu beachten, dass es keine Überwachung des Batteriestands gibt. Man muss selbst darauf achten, dass die Batterien nicht leer sind, eine Überwachung wäre dann nicht mehr gegeben. Zusätzlich ist noch ein hoher Wimpel befestigt, damit man den Schwimmer auch als physischen Indikator für Wasserbewegungen benutzen kann. Erkennt der Schwimmer Wellenbewegungen und laute Geräusche, sendet er über das Bluetooth-Modul ein Signal zu einem zweiten Arduino, der als Basisstation an Land dient.

Diese Basisstation ist ebenfalls mit einem Bluetooth-Modul ausgestattet, über welches das Signal der Poolstation empfangen wird. Sendet die schwimmende Station ein Alarmsignal, dient dieses als Trigger und an der Basisstation wird über einen angeschlossenen Buzzer und LEDs ein Alarm ausgegeben. Programmiert wird das System über die Arduino IDE.

So eine DIY-Sicherheitsvorkehrung ist natürlich nicht die eierlegende Wollmilchsau im Bereich Poolsicherheit. Die Batterien des Systems müssen immer geladen sein und eine physische Sicherheitsbarriere ist durch nichts ersetzbar. Generell sollte sich jeder Besitzer an die DIN EN 16582-1 halten. In dieser Norm sind die geltenden Richtlinien für private Pools festgehalten. Darunter auch Sicherheitsbestimmungen.

Den Code, eine Stückliste und einen Schaltplan findet man auf der Projektseite unter folgendem Link: —das

► make-magazin.de/xf51



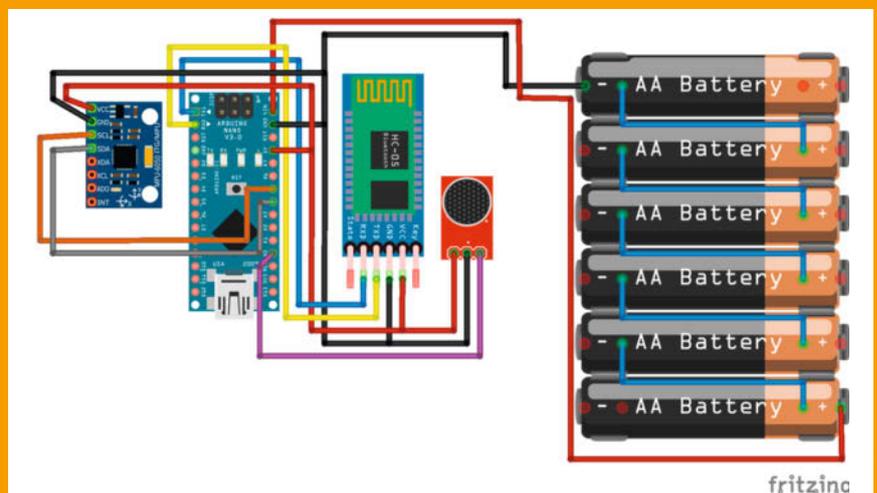
Zaid Jaber

Der Schwimmer ist 3D-gedruckt und verfügt über einen Wimpel.



Zaid Jaber

Zusammengebaut ist der Schwimmer einsatzbereit.



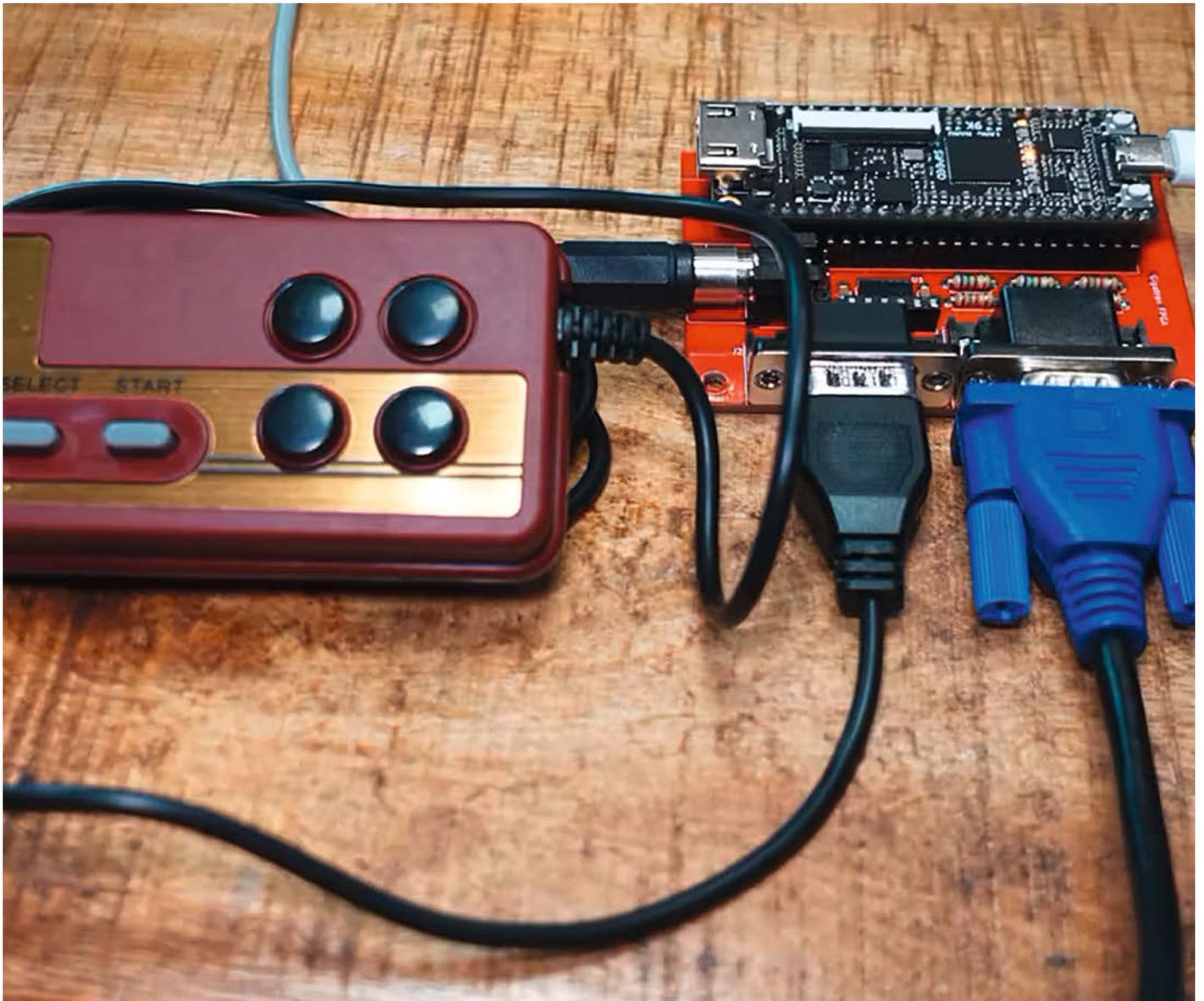
Zaid Jaber

Für die Technik im Schwimmer braucht man viele Batterien.

80er-Jahre- Spielekonsole im FPGA

Auch für Retro-Fans bieten FPGAs unzählige Möglichkeiten. So lassen sich leicht komplette Homecomputer und Spielekonsolen im FPGA aufbauen. Hier habe ich einen Computer, der aus zahlreichen 74xx-TTL-ICs aufgebaut ist, mit einem „Tang Nano 9k“-FPGA-Board realisiert.

von Michael Linsenmeier



TTL-Computer wurden noch bis Anfang der 80er-Jahre in Arcade-Maschinen verwendet. Die hier gezeigte FPGA-Implementierung des Retro-Computers „TTL-Gigatron“ hat 76 Kilobyte ROM, das durch den eingebauten Flash-Speicher des FPGA gebildet wird.

Darin befindet sich unter anderem ein rudimentäres Betriebssystem, das per Bit-Banging eine VGA-Grafik mit 160 Spalten mal 120 Zeilen und 64 Farben darstellen kann. Um das Spielerlebnis zu perfektionieren, kann auch Sound mit vier getrennten Stimmen erzeugt werden. Eine kleine Auswahl an Spielen ist ebenfalls in das Flash-ROM „eingebrennt“. Diese kann man über ein Menü auswählen. Wer selbst programmieren will, kann das mit BASIC, Coder GCL (Gigatron Control Language), einer eigens für diesen Computer entwickelten Programmiersprache, tun. Mit einem Arduino UNO oder Nano können die selbst geschriebenen Programme dann vom PC auf den Gigatron geladen werden.

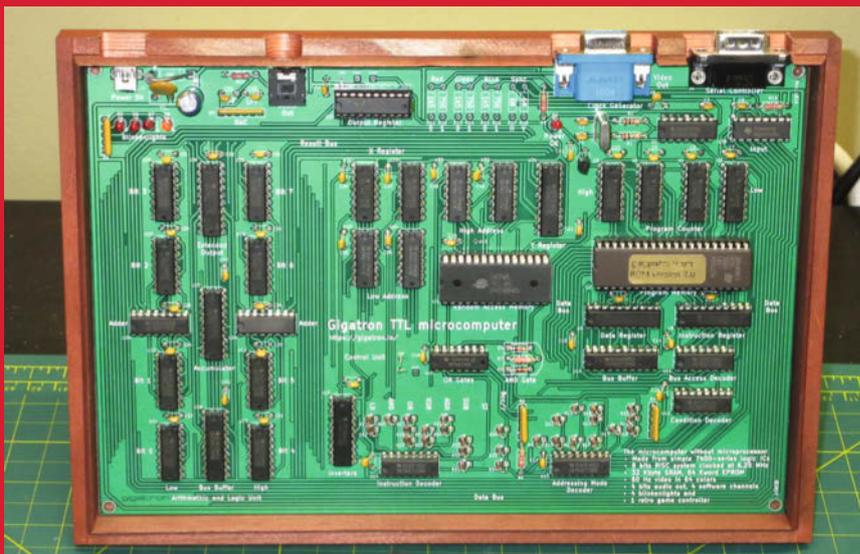
Zusätzlich zum FPGA-Design habe ich auch eine Adapterplatine entwickelt, in die das FPGA-Board eingesteckt wird. So erhält man die Anschlüsse für einen VGA-Monitor, einen NES-kompatiblen Controller sowie eine Cinch-Buchse für den Sound. Für Monitore mit HDMI-Eingang habe ich zusätzlich eine HDMI-Ausgabe implementiert: Durch die hohe Geschwindigkeit des FPGA ist dies im Gegensatz zu HDMI-Emulationen auf Mikroprozessoren leicht möglich.

Da das Tang Nano selbst keine D/A-Wandler (digital zu analog) für Video oder Audio hat, geschieht die D/A-Wandlung der digitalen Signale durch Widerstände auf dem Adapterboard, die als R2R-Netzwerk verschaltet sind.

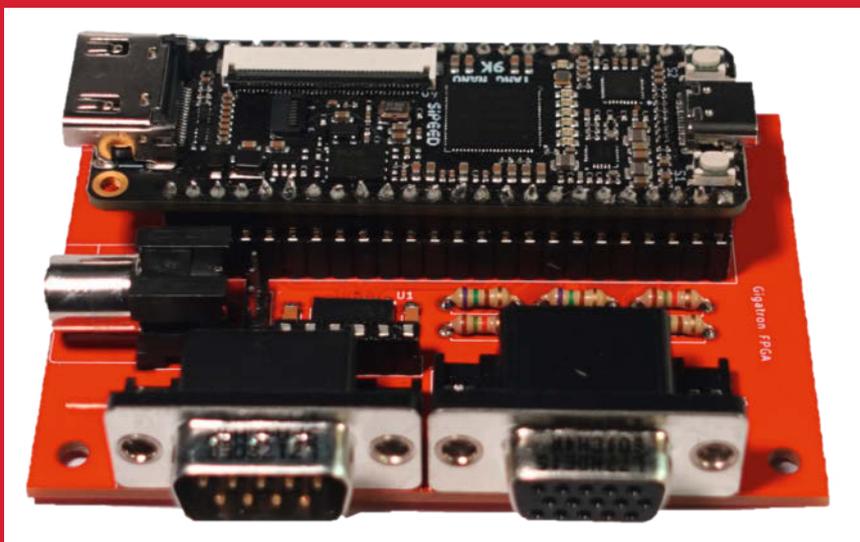
Die Vorlage für den FPGA-Computer besteht aus 34 einzelnen 74xx-TTL-ICs, 128 Kilobyte ROM und 32 Kilobyte RAM. Diese Komponenten sind nach wie vor problemlos erhältlich. Wer den Aufwand scheut, all diese Teile zusammenzubauen, für den ist die FPGA-Lösung ideal. Der günstige Tang Nano 9k hat allerdings den Nachteil, dass er nur 76 KByte ROM stellen kann. Daher musste ich die Anzahl der über das Menü verfügbaren Programme von zwölf auf acht reduzieren.

Der TTL-Gigatron wurde 2018 von Marcel van Kervinck und Walter Belgers vorgestellt und wird seither von einer kleinen Community weiterentwickelt und gepflegt. So kamen im Laufe der Zeit zahlreiche Erweiterungen hinzu, wie das „Pluggy McPlugface“, eine kleine Platine mit einem ATtiny85, die in den Gameport eingesteckt wird und dann den Anschluss einer PS2-Tastatur oder das Laden zusätzlicher Programme ermöglicht. Außerdem wurden zahlreiche Spieleklassiker (Snake, Racer, Tetris, Bricks ...) auf den Gigatron portiert. Es gibt sogar einen Emulator für den Apple-1 (6502).
—caw

► make-magazin.de/x5xm



Der „TTL-Gigatron Retro“-Computer hat keine CPU und kann dennoch eine 6502-CPU emulieren.



Die Adapterplatine bietet VGA-, NES-Controller- und Audioanschlüsse. Die Stromversorgung erfolgt per USB-C und HDMI direkt am Tang Nano 9k.

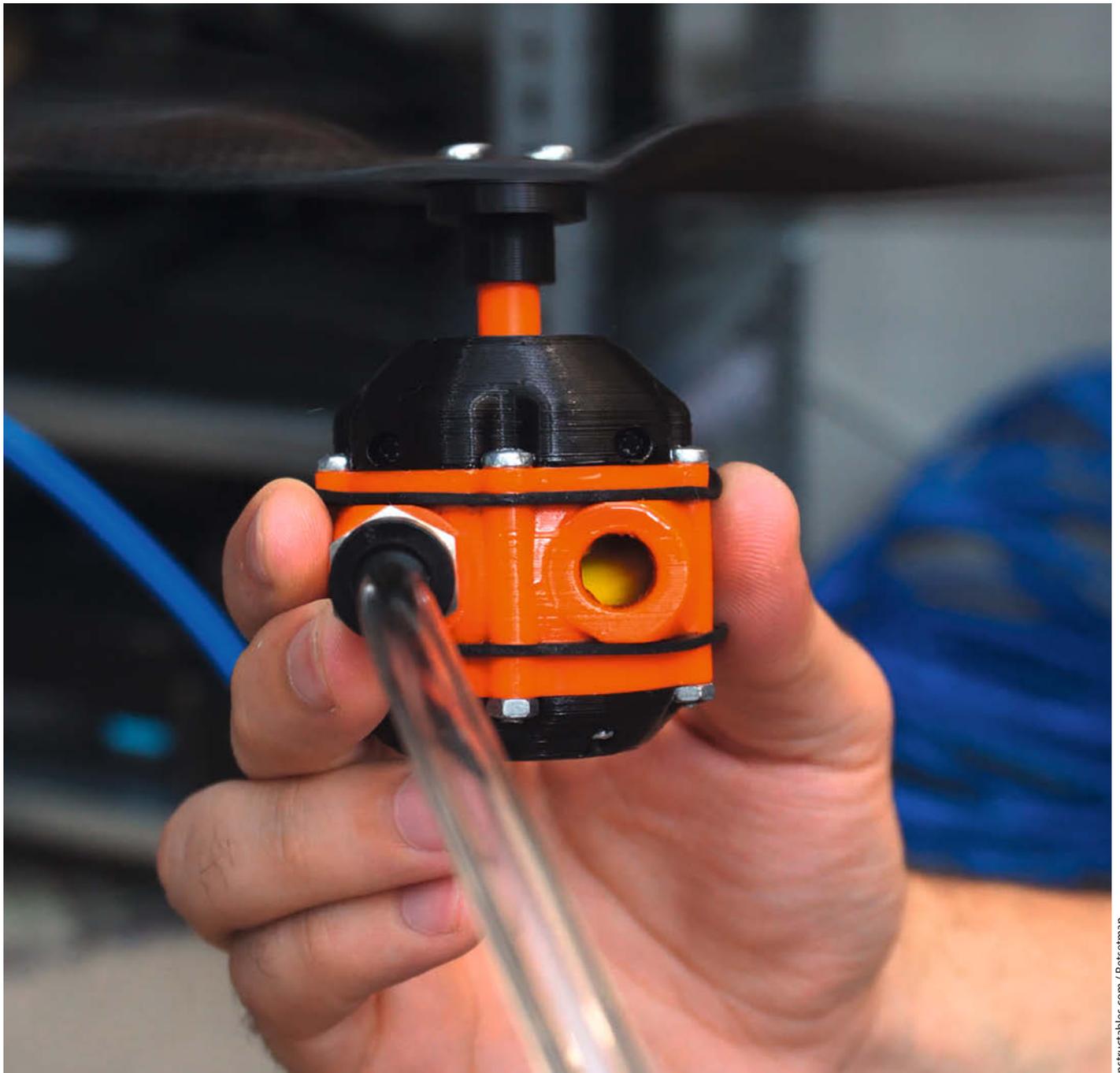


Ohne CPU und spezielle Grafikchips ist die Auflösung begrenzt. In jeder zweiten Zeile bleibt Zeit, das laufende Programm zu berechnen.

Motor aus dem 3D-Drucker

Technische Publikationen können eine tolle Inspiration für Maker-Projekte sein – selbst dann, wenn sie schon hunderte von Jahren alt sind. Denn viele Ideen lassen sich mit einem 3D-Drucker leicht wieder zum Leben erwecken und modifizieren, wie der hier vorgestellte Motor zeigt.

von Ákos Fodor



instructables.com/Retsetman

Der Maker Retsetman hat einen 3D-gedruckten Motor gebaut, der sich mithilfe von Druckluft bewegt. Das mechanische Prinzip ist von einem NASA-Prototyp aus dem Jahr 2006 abgeschaut, doch die Idee einer „Nutating Disc Engine“, eines taumelnden Scheibenantriebs, lässt sich sogar bis ins frühe 19. Jahrhundert zurückverfolgen.

Bei dem Design sitzt eine Kombination aus Scheibe und Kugel, die optisch an den Planeten Saturn erinnert, im Inneren eines geschlossenen Gehäuses (Bild 1). Dieses bewegliche Element ist schräg eingefasst und in der Lage, im Kreis zu taumeln, ohne sich dabei zu drehen. Das geschieht, wenn Druckluft sich ihren Weg vom Eingang zum Ausgang bahnt und sich abwechselnd an der Oberseite und der Unterseite der Scheibe entlang drückt (Bild 2). Durch die Schrägstellung der Welle ergibt sich an ihrer Spitze dann eine kreisförmige Bewegung, die man als Antrieb nutzen kann – z. B. für einen Propeller, wie in Retsetmans Projekt.

Bereits im Jahr 1830 erhielten die Brüder Edward und James Dakeyne ein Patent für einen Vorläufer dieses Antriebs, der Wasser nutzte, um Flachsmühlen (Leinenherstellung) im britischen Two Dales (Derbyshire) anzutreiben. Im Londoner *Mechanics' Magazine* aus 1833 (Bild 3) vermutete ein Leser, dass diese Maschine sogar dort einsetzbar wäre, wo die Strömung zu schwach für Wasserräder sei.

Dass das Prinzip auch heute noch eine Relevanz hat, zeigen Versuche des 20. Jahrhunderts, die Technik für Verbrennungsmotoren zu verwenden (siehe das verlinkte abgelaufene US-Patent 5.251.594 von 1993). Der taumelnde Antrieb hat nämlich die positive Eigenschaft, dass die Scheibe und die Innenwände des Gehäuses sich nicht berühren. Das verringert die mit der Reibung verknüpfte Hitzeentwicklung und den Verschleiß.

Der NASA-Prototyp, den Retsetman als Vorlage genutzt hat, verwendet zwei Scheiben und startet zunächst auch mit Druckluft. Sobald sich die Welle stabil dreht, wird Benzin eingespritzt und entzündet. Damit erreichte der Motor in Laborversuchen etwa 3000 Umdrehungen pro Minute und ein Drehmoment von 254 Nm. Ein optimiertes Design, das aber nur in der Konzeptphase blieb, sollte laut NASA einen Wirkungsgrad erreichen, den kein anderer Verbrennungsmotor liefern könne.

Ganz so stark ist der 3D-gedruckte Motor von Retsetman zwar nicht, aber das Projekt zeigt sehr anschaulich, wie 3D-Druck die Technologie demokratisiert, indem er es jedem ermöglicht, mit diesen interessanten Design zu experimentieren. Weitere Details und eine Aufbauanleitung gibt es unter dem folgenden Link.

—akf

► make-magazin.de/x2mq



Bild 1: Das Innere des taumelnden Motors. Von rechts dringt Druckluft ein und bringt das bewegliche Element (gelb) dazu, zu taumeln. Dadurch entsteht am oberen Ende der Welle eine Kreisbewegung, die sich auf den Propeller überträgt.

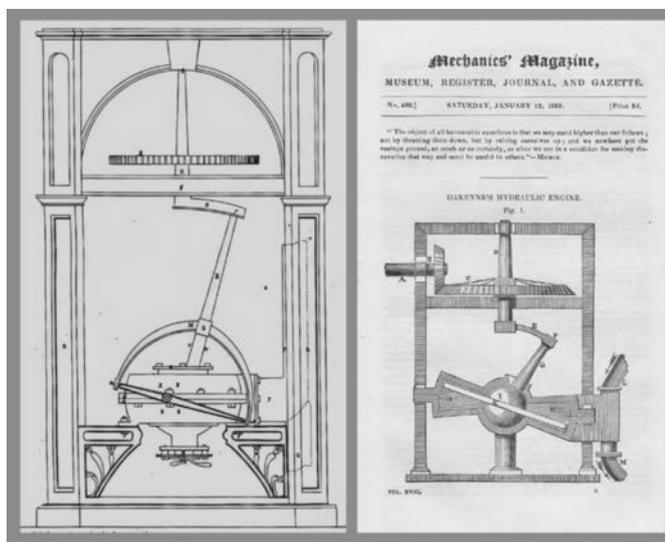


Bild 2: Links der Entwurf der Dakeynes von 1830, rechts eine Zeichnung des Prinzips aus der Erinnerung eines Lesers, wie 1833 im *Mechanics' Magazine* veröffentlicht. Letztere gibt die Funktion nicht ganz richtig wieder, da sich der Arm E nicht frei drehen kann.

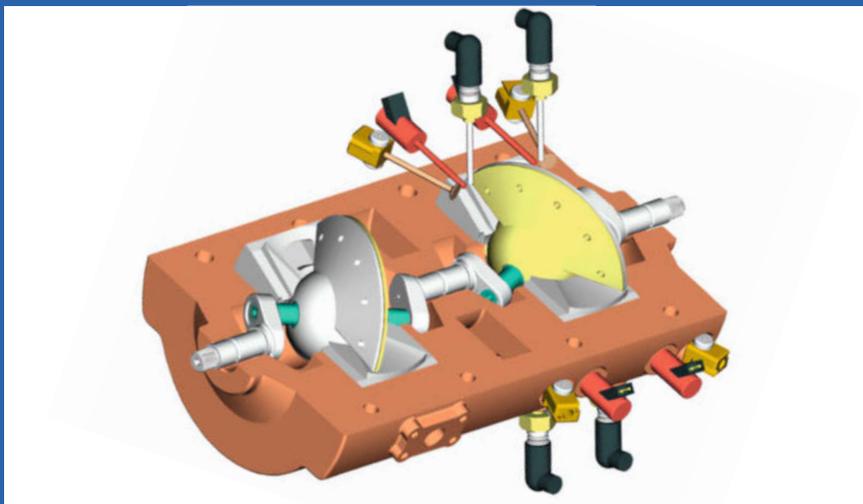
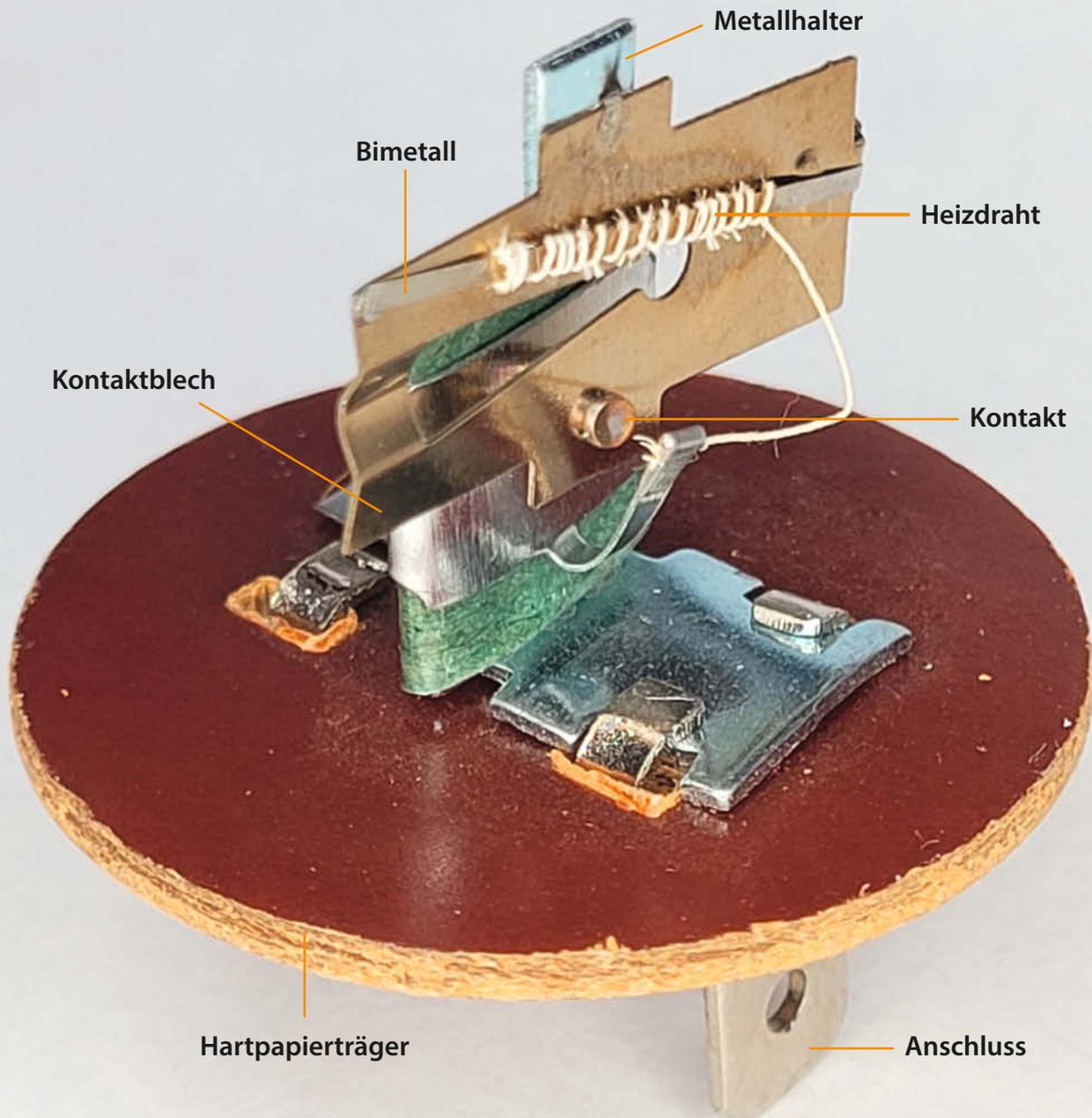


Bild 3: Für unbemannte Flugzeuge entwickelte die NASA im Auftrag der US-Army diesen Antrieb. Er nutzt zwei Scheiben und Benzin.

Blinkgeber

Ohne Elektronik und Mikrocontroller: Früher brachte die Kombi aus Strom, Bimetallen und Heizdraht die Lampen in Autos zum Blinken. In Oldtimern und älteren Mokicks findet man sie immer noch. Wir haben uns die Funktionsweise mal angeschaut.

von Daniel Bachfeld



Als neulich die Blinker an unserem Klein-kraftrad „Simson S51“ (Sie wissen ja, Bachfelds reiten auf der Vintage-Welle) nicht mehr funktionierten, stellte sich nach kurzer Fehlersuche ein defekter Blinkgeber als Ursache heraus. Das Teil hatte die Größe eines Bechereikos, also wie ein sehr großer Kondensator mit zwei Anschlüssen.

Natürlich packte mich die Neugier, was da eigentlich drinsteckt. Der Becher war schnell aufgehebelt. Ich habe mich zuvor noch nicht mit Blinkgebern beschäftigt, die landläufig (und fälschlicherweise) auch mal gerne als Blinker-Relais bezeichnet werden. Um so mehr war ich über den genial simplen Aufbau überrascht.

Es handelte sich um einen sogenannten Bimetall-Blinkgeber. Er besteht aus zwei Steckschuhanschlüssen, einem von einem Bimetallstreifen gesteuerten Kontaktblech und einem Heizdraht. Im großen Bild sieht man, wie der Heizdraht um den Bimetallstreifen gewickelt ist. Wir erinnern uns: Zwei Metalle mit unterschiedlichen Ausdehnungskoeffizienten verlängern sich bei Erwärmung in unterschiedlichem Maß. Die unterschiedliche Verlängerung führt zu einer Verbiegung des Bimetallstreifens in eine Richtung.

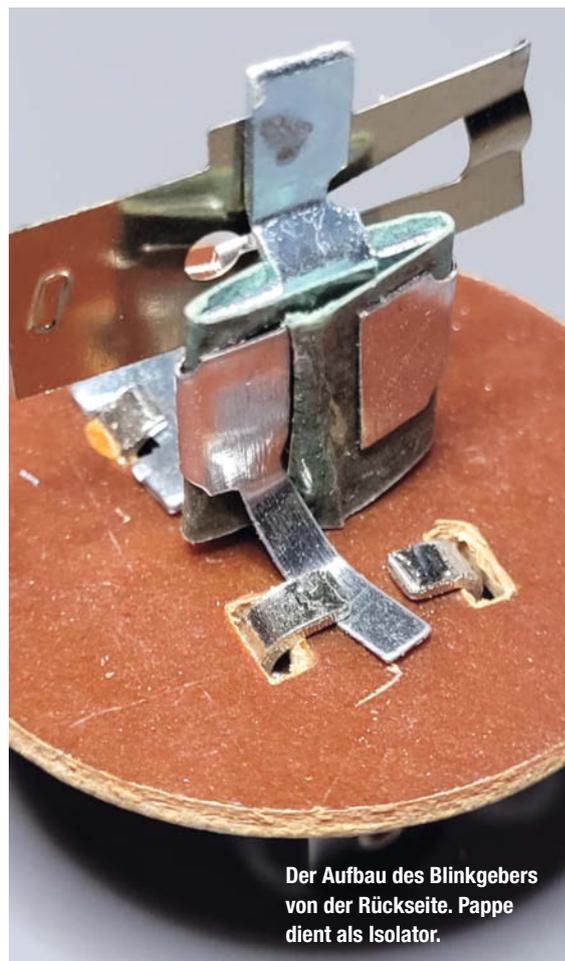
Gibt man eine Spannung auf beide Anschlüsse, so beginnt ein Strom durch den Heizdraht zu fließen. Mit dem Ohmmeter habe ich einen Widerstand zwischen 60 und 120 Ohm gemessen, an 12V ergibt sich also ein Strom zwischen 100 und 200 mA. In einem Blinkgeberstromkreis würden noch die Widerstände der Glühlampen dazukommen. Auf jeden Fall ist der resultierende Strom zunächst zu klein, um die Lampen zum Leuchten zu bringen.

Allerdings reicht der Strom, um den Heizdraht so stark zu erhitzen, dass der Bimetallstreifen sich schlagartig verbiegt und damit

den Kontakt auf dem Kontaktblech gegen einen weiteren Kontakt schlägt. Dadurch entsteht ein Kurzschluss des Stromkreises und der Widerstand des Blinkgebers geht gegen null. Die Lampen leuchten.

Gleichzeitig wird durch den Kurzschluss auch der Heizdraht überbrückt, sodass durch ihn kein Strom mehr fließt und er abkühlt. In der Folge schnappt der Bimetallstreifen wieder zurück, öffnet den Kurzschlusskontakt, die Lampen erlöschen und das Spiel beginnt von vorne, bis man den Stromkreis mit dem Fahrtrichtungsschalter am Lenker unterbricht. Das deutlich hörbare Klicken des Blinkgebers ist auf das sich schlagartig verbiegende Bimetall und das Geräusch der aufeinanderprallenden Kontakte zurückzuführen.

Die Blinkfrequenz bzw. das Verhältnis von Leuchtdauer zu -pause hängt unter anderem von der Spannung, den Widerständen im Stromkreis und der Außentemperatur ab. Je nachdem, ob man eine Lichtmaschine für 6V oder 12V hat, müssen auch die Lampen, Blinkgeber und Spannung zueinander passen. Je niedriger die Spannung, desto niedriger die Blinkfrequenz. Das vergleichsweise große Gehäuse des Blinkgebers dient vermutlich dazu, ihn toleranter gegenüber hohen und niedrigen Außentemperaturen zu machen. Sonst hätte



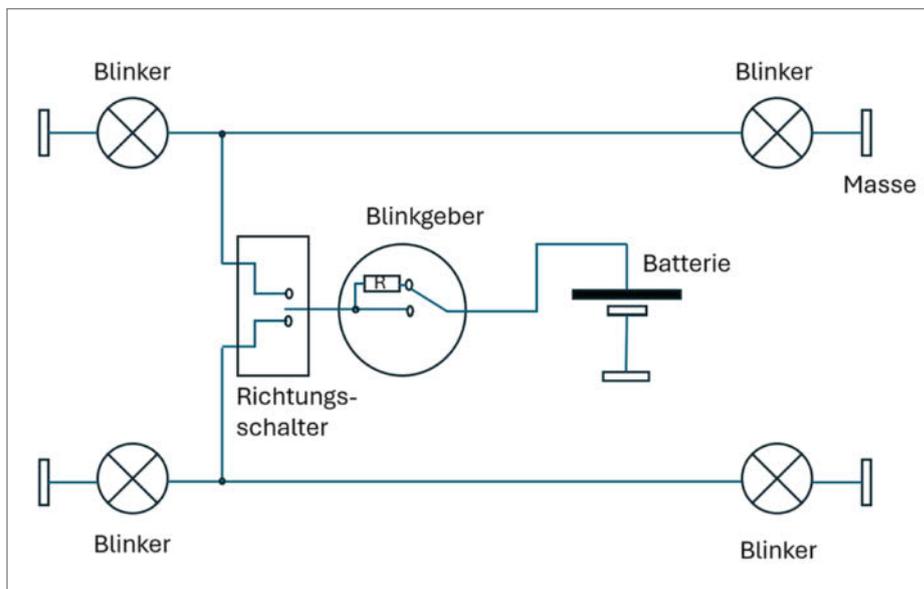
Der Aufbau des Blinkgebers von der Rückseite. Pappe dient als Isolator.

man im Winter einen sehr langsamen Blinker, während er im Sommer hektisch flackert? Wissen Sie es? —dab

► make-magazin.de/xqww



Der Blinkgeber im ungeöffneten Zustand

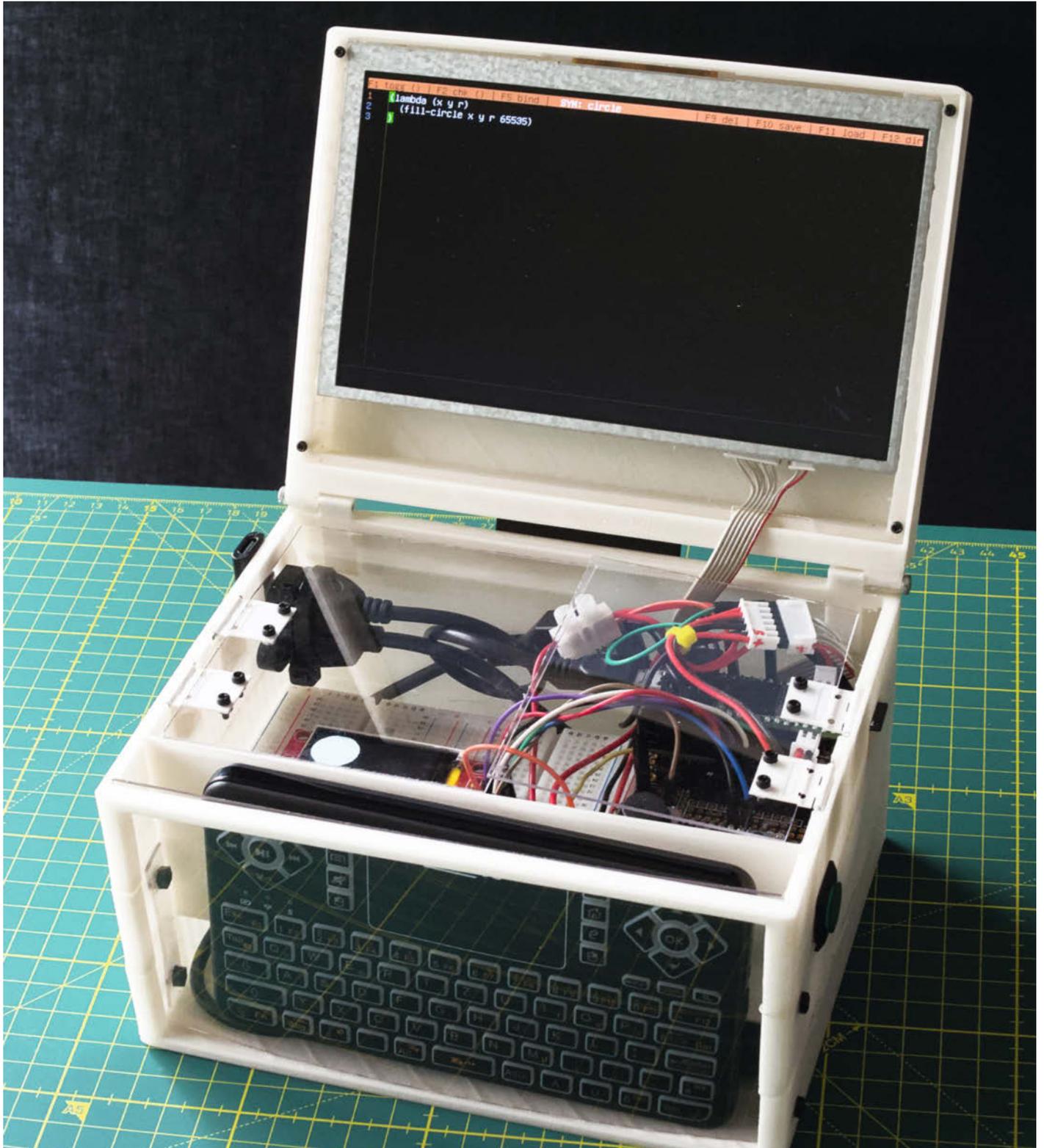


Der Blinker-Stromkreis mit Blinkgeber und Lampen

Die Lisp-Box

Retro-Computing-Feeling, aber mit Zukunftssicherheit und hohem Nutzwert – das bietet die Lisp-Box, ein schneller Bastel-Heimcomputer ohne Betriebssystem.

von Hartmut Grawe



Einen Rechner, den man bis hinab zu den letzten Bits und Registern programmieren, erforschen und verstehen kann. So stellen sich viele, die wie ich im Goldenen Zeitalter der Heimcomputer aufgewachsen sind, noch heute den idealen PC vor – kein Betriebssystem installieren, keine Blackbox unter der Haube, direkt nach dem Einschalten loslegen wie damals mit dem eingebauten BASIC.

Wie man sich so einen Computer mit der Programmiersprache Lisp bzw. der Mikrocontroller-Implementierung namens uLisp (gesprochen Micro-Lisp) selbst bauen kann, zeige ich in diesem Artikel mit meiner Lisp-Box. Die Dateien für den 3D-Druck finden sich wie auch die nachfolgend beschriebene Firmware in meinem GitHub-Repository (siehe Link in der Kurzinfo).

Wieso uLisp?

Eigentlich wurde ich schon vor einigen Jahren auf die Programmiersprache uLisp aufmerksam, die wie MicroPython, CircuitPython oder TinyBasic die Programmierung und Steuerung von Mikrocontrollern ohne ständiges Rekompilieren verspricht. Zwei Besonderheiten von uLisp machten mir dabei den Mund wässrig: Der Interpreterkern als solcher ist vergleichsweise winzig und läuft nicht nur auf Chips wie dem RP2350 oder ESP32, sondern auch noch auf älteren 8-Bit-Prozessoren wie dem ATmega328 des Arduino UNO (bis Rev3) und Nano – ein zentraler Unterschied zu MicroPython.

Außerdem ermöglicht uLisp mit geeigneter Hardware die interaktive Programmierung auf dem Mikrocontroller selbst: Die „REPL“ (Read-Eval-Print-Loop), also die Befehlsein-gabeschleife des Interpreters, ist nicht nur extern über USB an einem PC zugänglich, sondern auch über eine Tastatur und einen Bildschirm, die man direkt am Mikrocontroller anschließt.

Da Lisp wie BASIC und Python einen Interpreter verwendet, läuft Code unter Umständen rund 400-mal langsamer als ein vergleichbares C-Programm. In der Bastelpraxis fällt das jedoch selten ins Gewicht. Falls aber doch, bietet uLisp die Möglichkeit, Assemblerprogramme inline einzubetten – und neuerdings einen (noch eingeschränkten) Compiler für rechenintensive Abschnitte zu nutzen.

Die Website des Projekts verspricht eine extrem einfache Installation: Dazu muss man nur die uLisp-Firmware für den gewählten Mikrocontroller herunterladen, mit der Arduino IDE kompilieren, flashen und den seriellen Monitor starten.

Komplizierter wurde ein erster Test auch tatsächlich nicht – und ich war unmittelbar angefixt. Mit uLisp müsste ich zukünftig nicht mehr Board-abhängig zwischen einem Python- und C-Workflow wechseln und könnte

Kurzinfo

- » Mikrocontroller mit Lisp steuern
- » Autarker Minicomputer im Gehäuse mit Experimentierfeld und 7-Zoll-Bildschirm
- » Mit integriertem Fullscreen-Editor leichter programmieren

Checkliste



Zeitaufwand:

ein Wochenende inkl. 3D-Druck



Kosten:

120 bis 190 Euro

Material

- » Teensy 4.1
- » 7-Zoll-TFT-Display für RA8875-Treiber
- » Display-Controllerboard RA8875
- » USB-Host-Adapter (Buchsenleiste 2,54 mm auf USB-A-Buchse)
- » Kunststoffplatten 1 mm dick, transparent
- » Kavan Ruderscharniere (34×16 mm)
- » 5-V-Akku oder Powerbank
- » MicroSD-Karte

Optional

- » Adafruit Teensy 3.x Feather Adapter
- » Adafruit Feather Tripler
- » Breadboards halbe Baugröße
- » Mini-Tastatur von Joy-IT

Werkzeug

- » Seitenschneider
- » Cutter
- » 3D-Drucker (für das Gehäuse)

Mehr zum Thema

- » Maik Schmidt, Agon Light – der schnellste 8-Bit-Computer der Welt, Make 3/23, S. 28
- » Thomas Euler, Raspberry Pi Pico: Back to BASIC, Make 1/23, S. 72



Alles zum Artikel
im Web unter
make-magazin.de/xzvg

Die Lisp-Story

Lisp (**List Processing**) ist neben Fortran eine der ältesten und einflussreichsten Hochsprachen. In ihr findet man bereits viele Konzepte, die in modernen Programmiersprachen wie Python oder JavaScript weit verbreitet sind (z. B. dynamische Typisierung, Rekursion und First-Class-Funktionen). Auch ein noch relativ junges, aber mächtiges Python-Feature, die „List Comprehension“ (siehe Link in der Kurzinfo), ist in Lisp bereits seit Jahrzehnten integriert.

John McCarthy entwickelte Lisp ab 1958 am MIT (Massachusetts Institute of Technology) in Boston. Die Sprache lief zuerst auf einem IBM 704, einem Mainframe aus der Zeit der Lochkartenleser. Wegen ihrer speziellen Struktur empfahl sie sich für die frühen Gehversuche mit künstlicher Intelligenz und verlor daher in diesem Bereich jahrzehntelang nicht an Bedeu-

tung. Sogenannte Expertensysteme, die einfach strukturierte Fragen aus einem eng begrenzten Wissensschatz beantworten konnten, waren oft in Lisp implementiert.

Über die Jahre entwickelten sich viele Lisp-Dialekte. Breitere Bedeutung haben heute noch die Varianten Common Lisp, Scheme und Emacs Lisp sowie der moderne Abkömmling Clojure, dessen Entwicklung erst 2007 begann.

Die Mikrocontroller-Variante uLisp implementiert eine sorgfältig ausgewählte Untermenge von Common Lisp, die ausreicht, um komplexe Ideen sehr kompakt auszudrücken. Daher kann man auch mit extrem wenigen Zeilen Code eine Umgebung für das objektorientierte Programmieren implementieren und Projekte wie die Lisp-Box umsetzen.



Bild 1: Deckel zu – alles aufgeräumt. Rechts ist der Ein/Aus-Schalter und ein MicroSD-Kartenslot.

eigene Routinen und Erweiterungen auf praktisch all meinen Boards verwenden. Auf diese Art einen eigenständigen Lisp-Computer zu bauen, schien mir daher sehr sinnvoll.

Konzeption

Als Mikrocontroller wählte ich einen Teensy 4.1, um genügend Leistungsreserven für kommende Projekte zu haben. Zudem wollte ich einen Fullscreen-Editor in uLisp schreiben und nutzen sowie weitere Projektmonitore über SPI anschließen können. Auf einer MicroSD-Karte sollten sich Programme dauerhaft speichern lassen. Wer ein anderes Board für das

Projekt verwenden möchte, muss die Firmware der Lisp-Box entsprechend an seine Hardware anpassen.

Um einen genügend großen Bildschirm anzusteuern, der über die üblichen winzigen TFT- oder OLED-Screen-Gadgets hinausgeht (z. B. M5Stack Cardputer) und wirklich arbeits-tauglich ist, entschied ich mich für den schon lange verfügbaren RA8875-Controller. Er verfügt über eigenen RAM und ein Character-ROM, um Text und Grafiken darzustellen und den Speicher des Mikrocontrollers zu entlasten – bei einer maximalen Auflösung von 800×480 Pixeln. Man erhält passende TFT-Bildschirme bis zu einer Diagonale von 7 Zoll

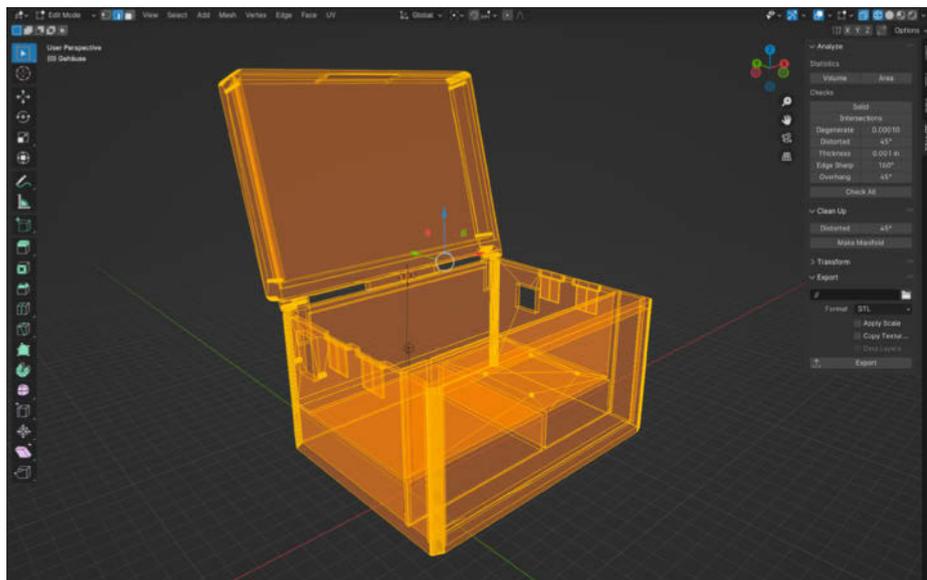


Bild 2: Das Gehäuse für den 3D-Druck habe ich in Blender entworfen.

(ca. 18 cm), bzw. auch Varianten, auf deren Rückseite der RA8875 bereits aufgelötet ist. Ich habe mich vorerst gegen einen Touchscreen entschieden, aber auch das ist eine Option.

Gehäuse und Innenleben

Das Gehäuse sollte nicht nur TFT-Bildschirm, Mikrocontroller und Akku beherbergen, sondern auch direkt Platz zum Experimentieren bieten. Deshalb habe ich es so breit wie den Bildschirm angelegt, gleichzeitig aber relativ hoch, um Breadboards und eingesteckte Jumper-Kabel unterzubringen. Das Ergebnis ist schachtelförmig: Die Lisp-Box, in der ein ganzes Projekt aufgeräumt verbleiben kann. Programm auf MicroSD-Karte abspeichern, abschalten, Deckel zu, Box wegstellen (Bild 1).

3D-gedruckte Teile entwerfe ich gerne mit Blender (Bild 2). Die Maße der Lisp-Box ergaben sich aus der notwendigen Breite für den Bildschirm und dem maximalen Druckraum meines Prusa mini (18 cm³). Ich hatte Glück, denn die 18 cm in jeder Richtung reichten gerade so aus, um das Gehäuse wenigstens horizontal nicht halbieren zu müssen. Den Teensy platzierte ich auf einem „Adafruit Teensy 3.x Feather Adapter“ und diesen wiederum auf einem „Adafruit Feather Tripler“ (Bild 3). Für den Teensy 4.0/4.1 muss man am Teensy Feather eine Leiterbahn (Program Pin) auf der Board-Unterseite durchtrennen (siehe Link in der Kurzinfo). Für den Fall, dass ihr den Program Pin später doch benötigt, lötet ihr am besten ein kleines Kabel so an, dass die Verbindung bei Bedarf wieder geschlossen werden kann.

Beide Adapter-Boards sind aber keineswegs zwingend, wie das einfache Schaltbild zeigt (Bild 4). Um den Computer zum Laufen zu bringen, benötigt ihr nur die beschriebene Board-Konstellation mit USB-Host und MicroSD-Karte sowie die geeignete Bildschirm-Controller-Kombination. Links neben dem Tripler befinden sich bei mir zwei Breadboards halber Baugröße (Bild 5). So kann ich Feather-Module und beliebige Schaltungsprototypen gleichzeitig nutzen.

Durch erneutes pures Glück passten die Breadboards und der Tripler nebeneinander genau auf die Breite des 7-Zoll-Bildschirms. Der Experimentierraum ist mit zwei durchsichtigen Klappen geschützt, befestigt an gängigen Modellbauscharnieren, die ins Gehäuse eingelassen sind. Eine „Durchreiche“ hinten erleichtert die Kabelverbindung zum Akku im untersten Stockwerk – neben dem wiederum Platz für eine kleine Schachtel mit wichtigen Bauteilen bleibt. Der USB-Host-Anschluss sowie der USB-Programmierschluss des Teensy sind links nach außen geführt, rechts befindet sich ein Ein/Aus-Schalter.

Im vorderen Bereich habe ich noch ein Tasaturfach vorgesehen; hier passt eine komfor-

table Minitastatur hinein, die die Lisp-Box vollends autark und transportabel macht. Als Akku findet bei mir derzeit eine alte NiMH-basierte Powerbank Verwendung, die ich gerade zur Hand hatte.

Den 3D-Drucker für den Druck des unteren Teils der Box über zehn Stunden (und daher teils unbeaufsichtigt) laufen zu lassen, war für mich nicht alltagstauglich. Daher habe ich ihn im Slicer wie eine Torte in drei Schichten geteilt, diese später verleimt und die Verbindung vorne nachträglich mit verschraubten Laschen verstärkt. Ich empfehle, solche Gehäuse-Teilstücke nicht mit Zapfen für die Passung zu drucken, wie sie Slicr anbietet. Das habe ich ausprobiert und musste die kleinen Kegel aber letztlich abzwicken, weil die Druckgenauigkeit nicht ausreichte, um sie lückenlos in die entsprechenden Löcher zu drücken.

Für die transparenten Innendeckel, die Abdeckung des TFT-Screens und die „Laderaum-Tore“ vorne habe ich Platten aus PETG in einer Stärke von 1mm zugeschnitten. Man bekommt sie für wenig Geld z.B. im Handel für Architekturmodelle.

Wenn ihr die 3D-Druckdaten aus dem GitHub unverändert übernehmt, benötigt ihr für die Modellbauscharniere (Bild 6) sowie die Deckel-Scharniere M2-Schrauben mit Muttern und zur Befestigung der TFT-Schutzscheibe selbstschneidende Gehäuseschrauben, die etwa M1,5 entsprechen. Ich hatte selbige zufällig aus unbekannter Quelle herumliegen. Alternativ ist z.B. noch Platz, die Löcher im Deckel vorsichtig auf 2mm aufzubohren und 20 bis 22mm lange M2-Schrauben mit Muttern zu verwenden. Die vorderen transparenten Platten werden einfach in die eingearbeiteten Gehäuseschlitze gesteckt und herausgezogen, wenn man z.B. an den Akku heran will.

Verkabelung

Um das RA8875-Treiberboard anzusteuern, nutze ich die SPI-Pins 26 und 27 sowie 38 und 39 – das entspricht der SPI-Schnittstelle SPI1 des Teensy (Bild 7). Als Reset-Pin für den Bildschirmcontroller dient Pin 10. Den Strom holt sich das Display über eine Abzweigung direkt vom Akku, d. h., es wird mit 5 Volt angesteuert – eine Konsequenz des RA8875-Boards von Adafruit, das praktischerweise bereits mit Level-Shiftern ausgestattet ist. So benötigen weder der Teensy noch der TFT-Bildschirm eine externe Reduktion der Powerbank-Spannung auf 3,3 Volt, da auch das Mikrocontroller-Board mit 5V gefüttert werden kann. Es ist aber auch möglich, die Versorgungsspannung des TFT-Screens (oder die eines zweiten Bildschirms) auf den 3,3-Volt-Ausgang des Teensy zu legen. Die 250 mA, die er dort zur Verfügung stellen kann, reichen dafür aus.

Um ein „Back Powering“ der USB-Verbindung zum PC zu vermeiden, habe ich die

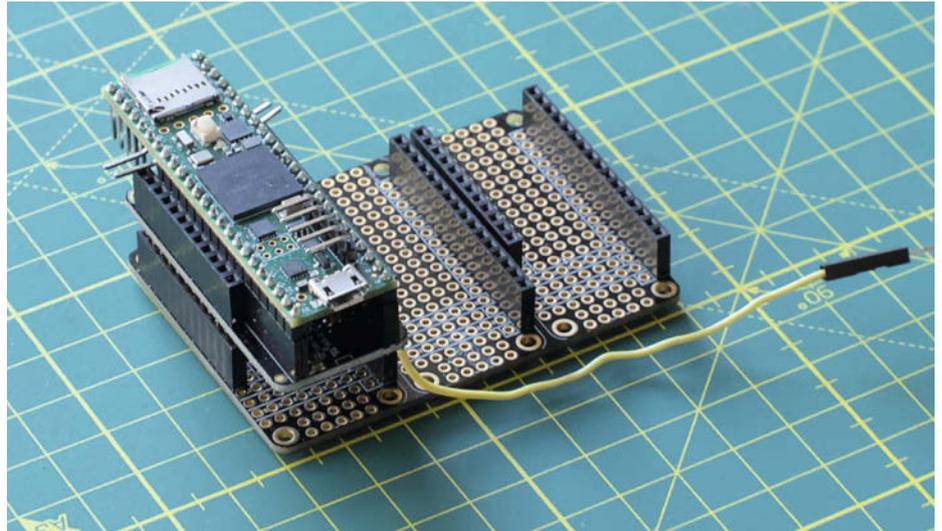


Bild 3: Der Teensy im Feather-Adapter, der wiederum im Tripler steckt. Mit dem gelben Kabel kann man den Program Pin bei Bedarf nutzen.

dafür vorgesehene „VIN/USB“-Verbindung des Teensy-Boards durchtrennt. Dadurch wird der Teensy nicht mehr über den USB-Programmieranschluss mit Strom versorgt, sondern muss immer mit der externen Spannungsquelle betrieben werden.

Firmware flashen

Um die Lisp-Box in Betrieb zu nehmen, ladet ihr die Firmware aus dem GitHub-Repository

und extrahiert sie in ein eigenes Verzeichnis, das so heißt wie die zentrale INO-Datei „ulisp-lispbox“. Zusätzlich müssen in der Arduino IDE die Bibliotheken „Adafruit BusIO“, „Adafruit GFX Library“, „Adafruit ST7735 and ST7789“, „Adafruit NeoPixel“ und „Adafruit LED Backpack Library“ installiert sein (die meisten dienen der Unterstützung von Erweiterungen, die in der Lisp-Box-Firmware enthalten sind). Die übrigen externen Bibliotheken wie „USBHost_t36“, die man im Source-Code fin-

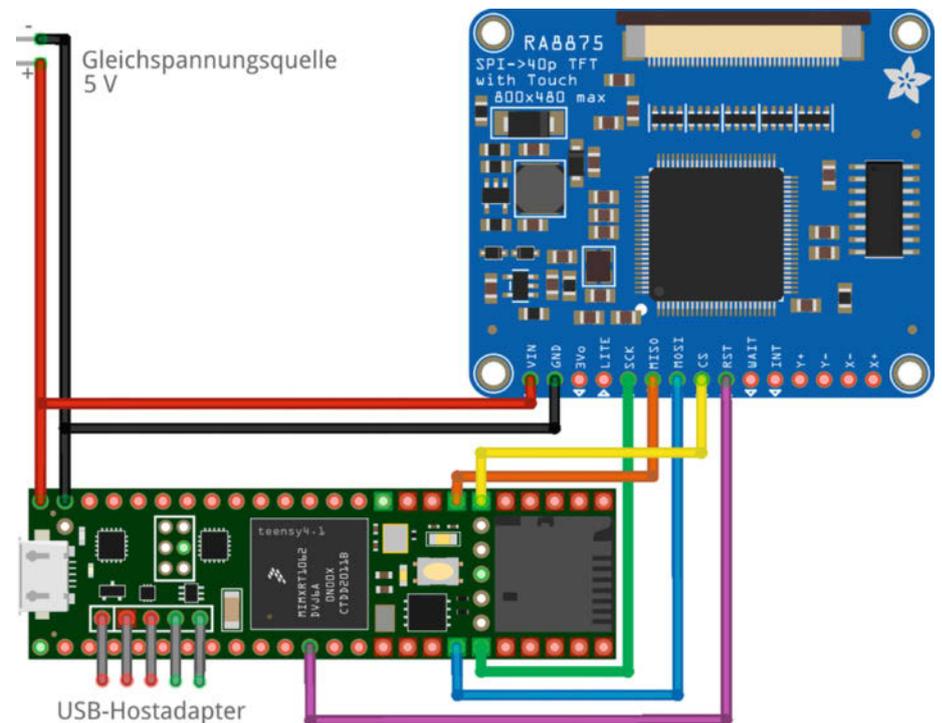


Bild 4: So muss man die Elektronik der Lisp-Box verkabeln.

fritzing

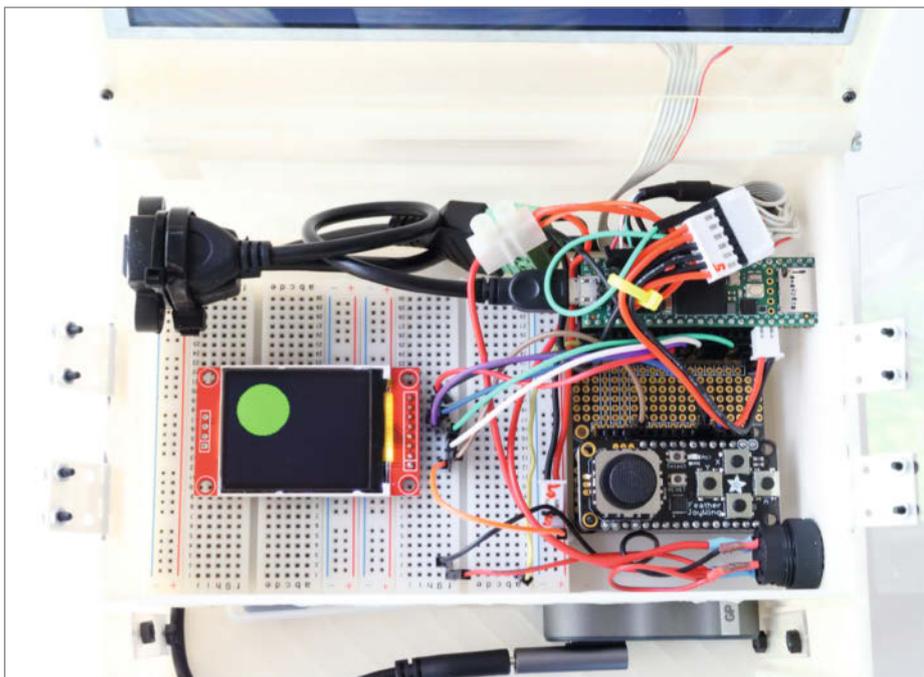


Bild 5: Das Innenleben der Lisp-Box mit einem Testprojekt.

det, werden bei der Installation der Teensy-Boardtreiber (auch „Teensyduino“ genannt) automatisch mitinstalliert (siehe Link in der Kurzinfor). Da das Display um 180 Grad gedreht montiert ist, um das Folienkabel zwischen TFT und Controller zu schonen, enthält das GitHub-Repository der Lisp-Box auch eine modifizierte Version der Bibliothek „Adafruit_RA8875“ – die aber nicht installiert werden muss, weil der Sketch lokal auf sie zugreift.

Verbindet den Teensy per USB mit dem Computer und öffnet in der Arduino IDE den Sketch `ulisp-lispbox.ino`. Wählt danach im Menü „Werkzeuge/Board“ euer Teensy-Board aus der Liste aus und stellt den Port ein, an dem er angeschlossen ist. Klickt schließlich auf

den „Hochladen“-Button links oben in der IDE und überträgt die Firmware auf den Mikrocontroller.

Erste Schritte

Ist die Firmware auf den Teensy geschrieben, meldet sich der Mikrocontroller nach einem Reset ca. sechs Sekunden später mit dem Startprompt von uLisp:

```
uLisp 4.6
47206>
```

Ähnlich wie in Python – bzw. eigentlich chronologisch umgekehrt, weil Lisp älter ist – sind Variablen in Lisp nicht typisiert. Der Interpreter

entscheidet aus dem Kontext, wie der aktuelle Inhalt einer Variable zu behandeln ist. Der RAM wird also dynamisch reserviert und mit einer automatischen „Garbage Collection“ von nicht mehr benötigten Objekten gereinigt. Daher meldet sich der uLisp-Prompt in der REPL immer mit einer vorangestellten Zahl. Sie gibt die Anzahl freier Lisp-Objekte an und ist ein Maß für den noch verfügbaren Speicher.

Hinter dem `>`-Zeichen kann man nun unmittelbar mit der Programmierung beginnen – entweder über das USB-Keyboards der Lisp-Box oder, sofern angeschlossen, über das serielle Terminal der Arduino IDE auf dem PC.

Probiert Lisp aus, indem ihr `(* 9 6)` und die Eingabetaste eintippt – es wird unmittelbar das Ergebnis `54` ausgegeben (mehr dazu im Kasten „Besonderheiten von Lisp“).

Der einfache Arduino-Test, das „Blink“-Programm, sieht in uLisp so aus:

```
(pinmode 13 t)
(loop
  (digitalwrite 13 t)
  (delay 1000)
  (digitalwrite 13 nil)
  (delay 1000)
)
```

Um das Programm abzubrechen, tippt man `~` (Tilde) und bestätigt mit der Eingabetaste.

Die Argumente der `loop`-Schleife erkennt man sofort als typische Arduino-Anweisungen, wobei `t` für „True“ steht und `nil` für „False“. Beim `pinmode`-Befehl setzt `t` den Pin 13 als Ausgang fest (mit `nil` wäre er ein Eingang). Funktionen von uLisp wie `pinmode`, `digitalwrite` oder `delay` existieren in einem „normalen“ Lisp wie Common Lisp nicht; sie wurden dem Sprachkern eigens für die Mikrocontroller-Umgebung hinzugefügt.

Das ist typisch Lisp: Man kann und soll die Sprache für eigene Zwecke modifizieren und erweitern. Im Fall von uLisp ist das besonders

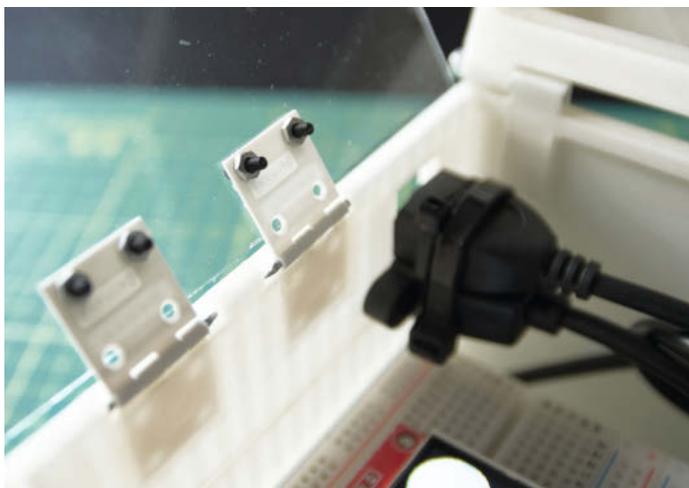


Bild 6: Die Schutzklappen für das Experimentierfeld sind mit Modellbau-scharnieren (Ruderscharnieren) befestigt.

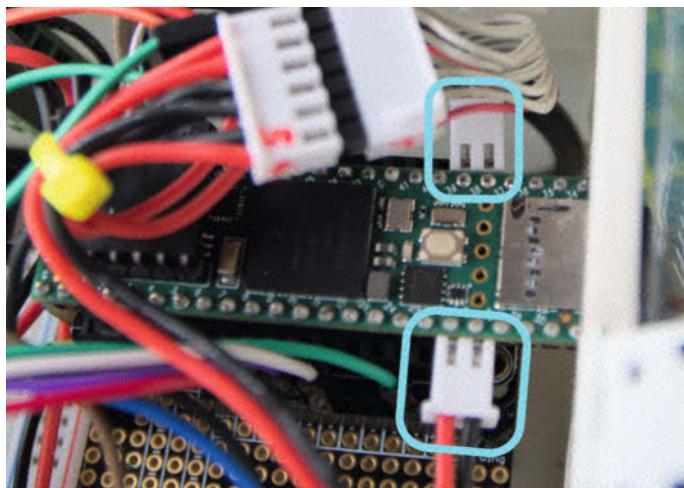


Bild 7: Die Pins für SPI1 des Teensy sind mit gewinkelten Headern bestückt und erlauben so den leichten Anschluss von JST-Verbindungskabeln.

Besonderheiten von Lisp

Die grundlegende Datenstruktur in Lisp besteht aus Listen, die beliebig viele Elemente (auch weitere Listen) enthalten können. Diese definiert man mithilfe von runden Klammern, z. B.:

```
(1 2 3 4 5)
((a b) (c (d e)))
```

Einrückungen, Zeilenumbrüche und zusätzliche Leerzeichen spielen bei der Strukturierung eines Lisp-Quelltextes überhaupt keine Rolle. Man muss einzig die runden Klammern um Daten und Befehle beachten sowie die durch mindestens ein Leerzeichen abgetrennten Symbole darin.

Wenn man in Lisp eine Funktion verwendet, wird diese grundsätzlich vorangestellt. Ein Befehl hat also immer die Form

```
(funktion argument1 argument2 ... )
```

und wird stur von links nach rechts und, was geschachtelte Klammern angeht, von innen nach außen ausgewertet. Eigentlich sehr einfach und doch ungewohnt, wie man an einer einfachen Textausgabe sehen kann. Diese schreibt man in Lisp wie folgt:

```
(princ "Hallo Welt")
```

Als Ergebnis erhält man `Hallo Welt` ohne Anführungszeichen, wenn man `princ` statt `print` verwendet. Wie so vieles in Lisp hat das historische Gründe. Der Befehl `princ` bedeutet schlicht `print characters` – reine Gewöhnungssache.

Bei einer Berechnung sieht die Syntax genauso aus. Auch hier steht der Operator an erster Stelle und die Operanden befinden sich dahinter:

```
(* 4 (+ 5 6))
```

einfach. Wie es geht, erklärt die Website zu `uLisp` ausführlich. Dieser Mechanismus erlaubt, z. B. sämtliche Arduino-C-Bibliotheken nach eigenem Bedarf in meist minimalistische `uLisp`-Wrapper zu kleiden, die dann automatisch gemeinsam mit dem `uLisp`-Kern kompiliert werden. Anschließend stehen die Funktionen innerhalb der Hochsprache zur Verfügung.

Zusätzlich ist es in Lisp üblich, eine „Lisp Library“ aufzubauen: Man schreibt in Lisp nützliche Hilfsfunktionen und sammelt sie in einer Datei, die jeweils beim Start des Systems

Diese Schreibweise hat den Vorteil, dass es durch die Verschachtelung keine Mehrdeutigkeit geben kann, z. B. in welcher Reihenfolge gerechnet werden muss.

Flexible Funktionen

Lisp unterscheidet grundsätzlich nicht zwischen Befehlen und Daten. Dadurch können Funktionen sowohl Werte (Zahlen oder Strings) als auch andere Funktionen als Wert zurückgeben. Damit kann man Funktionen schreiben, die während der Laufzeit neue Funktionen (und Variablen) erstellen, die sofort verwendbar sind. Lisp gilt deshalb auch als Programmiersprache, mit der man letztlich eine Programmiersprache nach seinen eigenen Bedürfnissen entwickeln kann. Ein vereinfachtes Beispiel:

```
(defun multiply (x)
  (lambda (y) (* x y)))
```

Mit `defun` definieren wir die Funktion `multiply`, die den Wert `x` erwartet. Sie gibt die anonyme Funktion `lambda` ohne Namen zurück, die ihrerseits den Wert `y` annimmt, um `x` und `y` zu multiplizieren.

Als Nächstes definieren wir mit `defvar` eine Variable namens `times10` und speichern in ihr die `multiply`-Funktion mit `10` als fest übergebenen `x`-Wert:

```
(defvar times10 (multiply 10))
```

Ab jetzt können wir `times10` nutzen, um etwas mit `10` zu multiplizieren, z. B. so:

```
(times10 5)
```

Als Ergebnis erhalten wir `50`. Nach diesem Muster lassen sich selbst komplexeste Funktionen und Zusammenhänge programmieren – und für eine bessere Lesbarkeit vereinfachen.

eingelassen wird. Ein kleiner Nachteil dieser Methode ist, dass jede so zur Verfügung gestellte Lisp-Funktion etwas RAM verbraucht, während in C verfasste Erweiterungen im Flash-Memory lagern.

Um bereits ausgeführte Befehle in der REPL erneut aufzurufen, stellt `uLisp` eine rudimentäre History zur Verfügung: Durch einen Druck auf `TAB` kann man die vorherige Eingabe am Prompt wiederholen. Ein Druck auf `ESC` löscht in der Lisp-Box den Bildschirm und setzt den Cursor wieder in die linke obere Ecke. Die Cursorstasten links/rechts sind in

Ihr Partner für IT-Weiterbildung Videokurse für IT-Professionals



Objektorientierte Programmierung in PHP

Vertiefen Sie Ihr Wissen in PHP und lernen Sie die Konzepte und Techniken der objektorientierten Programmierung (OOP) kennen, um besser skalierbare und wartbare Webanwendungen zu entwickeln.



GitHub Actions und Azure Bicep – Infrastructure as Code

In diesem Kurs entdecken Sie, wie Sie mit GitHub Actions und Bicep Infrastructure-as-Code-Strategien umsetzen können, um Ihre Infrastrukturmanagementprozesse zu modernisieren und automatisieren.



Synchronisation von Identitäten: Entra Cloud Sync

Erfahren Sie, wie und in welchen Anwendungsszenarien Sie das neue Identitätssynchronisationstool Entra Cloud Sync in Ihrem hybriden Setup einsetzen können.

Jetzt alle
Videokurse entdecken:
heise-academy.de

Ein Tipp für Umsteiger

Die Konventionen von Common Lisp verlangen eigentlich, geschachtelte Klammernkonstruktionen nicht wie in dem im Artikel gezeigten „Blink“-Beispiel zu notieren. Demnach müsste die schließende Klammer der loop-Schleife direkt hinter der schließenden Klammer vom letzten Argument in der Schleife stehen

```
...
(delay 1000))
```

und nicht wie oben in der nachfolgenden Zeile ganz links. Mir hilft es beim Lesen meines Lisp-Codes jedoch enorm, die Konventionen zu ignorieren und die Klammern mit wenigen Ausnahmen so zu setzen wie die geschweiften Block-Klam-

mern in C, d. h., aus der C-Endlosschleife

```
while(1) {
    ...
}
```

wird dann eben

```
(loop
    ...
)
```

Innere Blöcke rücke ich entsprechend ein und behandle deren schließende Klammern ebenfalls wie geschildert. Das verhindert die berüchtigten langen Lisp-Klammerreihen))))))) am äußersten Ende von geschachtelten Blöcken, die mich persönlich nach wie vor nerven würden.

der REPL für mehr Eingabekomfort mit runden Klammern belegt.

Der Editor

Abschließend stelle ich noch die Bedienung meines Fullscreen-Programmeditors vor, der in der Lisp-Box-Firmware als Erweiterung erhalten ist. Um ihn zu benutzen, startet ihr ihn ganz schlicht mit

```
(se:sedit)
```

Daraufhin erscheint ein weitgehend leerer Bildschirm mit einer Statusleiste ganz oben, der Zeilennummer „1“ am linken Rand und dem Cursor. Nun können beliebige Texte frei eingetippt werden (Bild 8). Wer programmieren will, gibt uLisp-Quellcode ein, aber der Editortaugt auch dazu, beliebige unformatierte Texte zu schreiben und abzuspeichern. Der

Editor folgt rudimentären Konventionen von Emacs, aber nicht ausschließlich. Wie ihr in der Tabelle „Tastenkombinationen des Lisp-Box-Editors“ sehen könnt, sind einige Funktionen über mehrere alternative Tastenkombinationen erreichbar.

Möchtet ihr mit dem Editor eine existierende Funktion bearbeiten, gebt ihr in der REPL folgende Zeile ein:

```
(se:sedit 'MeineFunktion)
```

Den Platzhalter `MeineFunktion` ersetzt ihr durch den Namen, dem ihr zuvor eine Funktion zugewiesen habt, die nun bearbeitet wird. Durch das einfache Anführungszeichen davor wird die Zeichenfolge des Symbolnamens als solche übergeben und nicht zuvor ausgewertet (was Lisp ohne das Hochkomma versuchen würde).

Wenn ihr mit „F5“ den Editor wieder verlasst, ist der aktuelle Symbolname vorausgewählt und muss nur mit Enter bestätigt werden. Es ist aber natürlich auch möglich, den Symbolnamen zu ändern und somit den Quelltext einem neuen Symbolnamen zuzuweisen, etwa um eine andere Variante einer existierenden Funktion zu erstellen.

Um eine Funktion im Editor korrekt zu notieren, müsst ihr dort allerdings `lambda` statt `defun` verwenden. Das liegt daran, dass der Fullscreen-Editor selbst in Lisp geschrieben ist und den `defun`-Befehl nicht verwenden kann. Daher nutze ich das Äquivalent

```
(defvar Funktionsname
 '(lambda (.Argument) (.Code)))
```

Fertig, los!

Während ich Lisp im Studium nicht ausstehen konnte, weil mir die Syntax als pure Zumutung erschien, stören mich die leidigen Klammern heute nicht mehr. Dass mein Editor mit „bracket matching“ zusammengehörige Klammern hervorhebt, macht das Programmieren auch viel angenehmer – denn

Wieso se:edit statt edit?

Das Kürzel `se:` (meine Abkürzung für Screen Editor) ist allen internen Funktionen des Editors vorangestellt (auch der, mit der ihr ihn aufruft), um Konflikte mit euren eigenen Funktionen im uLisp-Namensraum zu vermeiden. Andernfalls

wäre es möglich, aus dem Editor heraus versehentlich eine Editor-Funktion zu überschreiben, wenn man zufällig denselben Namen vergibt. Mit Doppelpunkt abgetrennte Präfixe sind zur Unterteilung von Namensräumen in Lisp üblich.

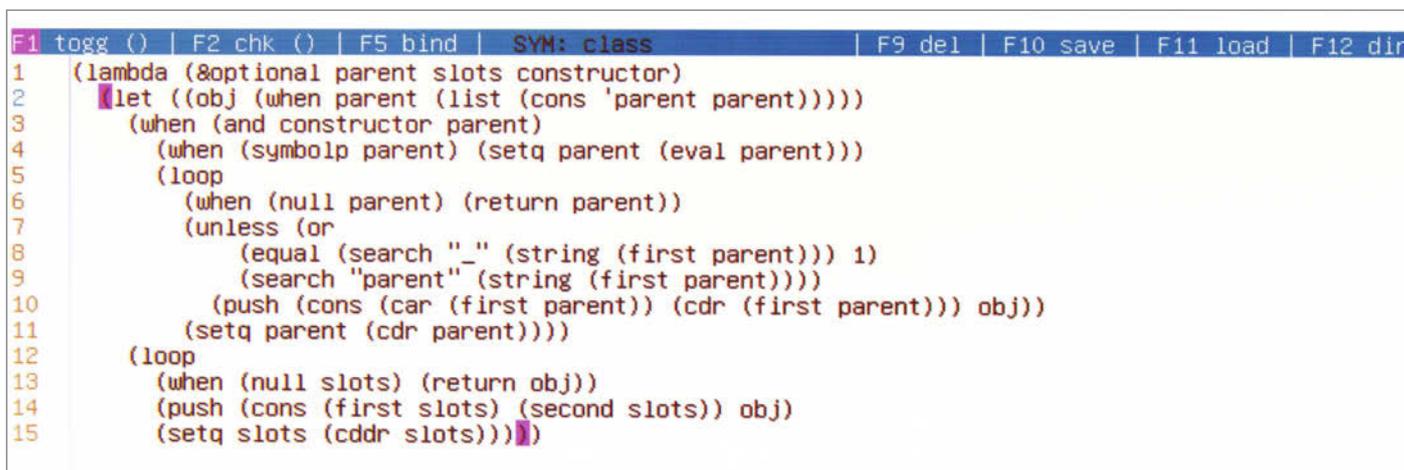


Bild 8: Der Editor (hier mit invertierten Farben) mit aktiviertem „bracket matching“. Bearbeitet man wie hier eine bereits existierende Funktion, gibt sie der eingebaute „pretty printer“ von uLisp gemäß der Common-Lisp-Konvention aus.



30% Rabatt!

*Endlich Wochenende! Endlich genug Zeit, um in der c't zu stöbern. Entdecken Sie bei uns die neuesten Technik-Innovationen, finden Sie passende Hard- und Software und erweitern Sie Ihr nerdiges Fachwissen. **Testen Sie doch mal unser Angebot: Lesen Sie 5 Ausgaben c't mit 30 % Rabatt – als Heft, digital in der App, im Browser oder als PDF. On top gibt's noch ein Geschenk Ihrer Wahl.**

Jetzt 5 × c't lesen
für 24,00 € statt 31,75 €**

** im Vergleich zum Standard-Abo

Jetzt bestellen:
ct.de/meintag



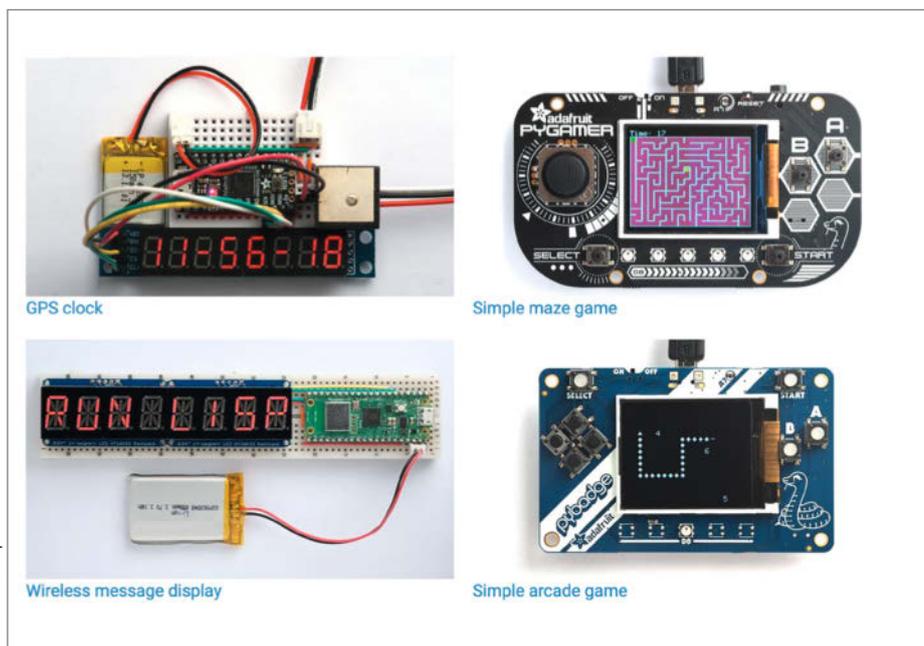
niemand will ständig verschachtelte Klammern zählen. Jetzt muss ich mich nur noch tiefer in die Möglichkeiten von uLisp einarbeiten. Zum Glück gibt es auf der Projekt-Website eine sehr umfassende und leicht

verständliche Dokumentation mit vielen Tutorials und beeindruckenden Beispielen (siehe Link in der Kurzinfor) bis hin zum Ray-tracing auf Mikrocontrollern mit nur sehr wenigen Zeilen Code (Bild 9). —akf

Tastenkombinationen des Lisp-Box-Editors

Strg+Q/Strg+C	Editor beenden und zur REPL zurückkehren
Strg+X/Strg+B/ Strg+N	Aktuellen Textspeicher verwerfen (= „new“, neue Datei)
Strg-K/Strg-L	Zeile ab Cursorposition löschen
Strg-A/Pos1	Cursor zum Zeilenanfang
Strg-E/Ende	Cursor zum Zeilenende
^	Cursor zum Anfang des Textspeichers
Bild rauf/ runter	Eine Bildschirmseite hoch oder runter
F1	„bracket matching“ an/ aus, also die Hervorhebung zusammengehöriger Klammern
F2	Prüfen, ob die Klammer unter dem Cursor im Textspeicher eine zugehörige Klammer hat. Wenn ja, werden beide temporär hervorgehoben (sinnvoll, wenn kontinuierliches „bracket matching“ abgeschaltet ist – gedacht zur Performance-Verbesserung auf schwächeren Systemen oder in sehr großen Quelltexten).
F5	Den Inhalt des Textspeichers einem Namen (einem „Symbol“, sagt man in Lisp) eurer Wahl zuweisen und den Editor verlassen
F9	Eine Datei auf der SD-Karte löschen
F10	Den Text auf SD-Karte speichern
F11	Text von der SD-Karte in den Speicher laden (und somit den aktuellen Textbuffer überschreiben)
F12	Inhaltsverzeichnis der SD-Karte anzeigen (weiterer Tastendruck kehrt zum Textspeicher zurück)

Für die Vergabe der Dateinamen folgt der Editor dem FAT- und damit dem sogenannten „8.3“-Standard: Der Dateiname darf maximal acht Zeichen haben, gefolgt von einem Punkt und einem Suffix von maximal drei Zeichen Länge. Sowohl Name als auch Suffix müssen in Großbuchstaben vergeben werden.



Screenshot: ulisp.com

Bild 9: Auf der uLisp-Projekt-Website gibt es einige interessante Projekte für Mikrocontroller.



Mehr Hirn für Schimpfolino

Schimpfolino bekommt ein EEPROM, um noch mehr Schimpfwörter zusammensetzen zu können. Sein Hunger nach Worten soll aber nicht mehr Hunger nach Strom bedeuten. Im Gegenteil, Schimpfolino wird in diesem Artikel, was den Stromverbrauch angeht, Genügsamkeit beigebracht.

von Nikolai Radke

Im letzten Artikel wurde Schimpfolino auf einem Steckbrett zum Leben erweckt. Dieser Artikel beschäftigt sich mit den Fragen, wie größere Datenmengen in einem externen EEPROM gespeichert werden können, wie sie ausgelesen werden und vor allem, wie sie da schnell reinkommen. Anschließend werden, wenn auch ziemlich trocken, Techniken zum Stromsparen und nachhaltigen Beleidigen vorgestellt.

Internes EEPROM

Spätestens seit Make 1/24 (siehe Link in der Kurzinfor) kennen wir die Vorteile eines EEPROMs: Es ist wiederbeschreibbar und behält auch ohne Stromversorgung seine Daten. Somit eignet es sich z. B. hervorragend für das Loggen von Sensordaten. Praktischerweise bringen die ATtiny25/45/85 ein internes EEPROM in den jeweiligen Größen 128/256/512 Bytes mit.

Der Zugriff über die Arduino IDE ist denkbar einfach: Mit der Library EEPROM.h stehen die wichtigsten Befehle zur Verfügung. Der Befehl `EEPROM.write(adresse,wert)`; schreibt ein Byte (`wert`) an eine Adresse im EEPROM. Die Adresse ist vom Typ Integer und startet ab 0. Analog dazu wird mit `wert = EEPROM.read(adresse)`; ein Byte aus einer Adresse ausgelesen. Da eine Adresse immer auf ein Byte zeigt, muss bedacht werden, dass bei größeren Datentypen die Adresse für den nächsten Eintrag entsprechend erhöht werden muss, um nicht versehentlich einen Teil des anderen Datentyps zu überschreiben, sofern es aus mehreren Bytes besteht.

Das interne EEPROM hat allerdings seine Grenzen: Es ist mit etwa 3 ms pro Schreibzugriff ziemlich langsam. Zudem ist nach etwa 100.000 Schreibzyklen Schluss, dann ist der entsprechende Speicherbereich, zumindest statistisch, kaputt oder stark vergesslich.

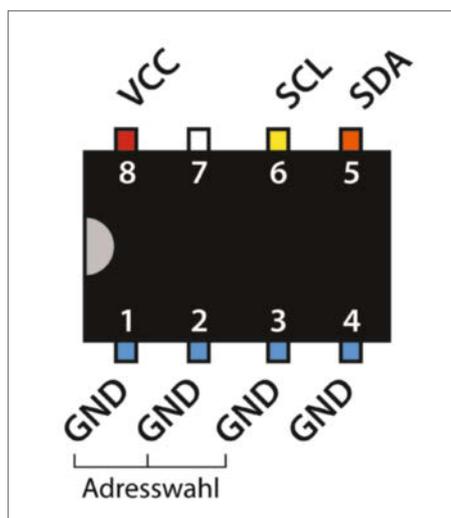


Bild 1: Die Pinbelegung des 24AA64-EEPROMs

Kurzinfor

- » EEPROMs extern mit einem seriellen Terminal beschreiben
- » Schimpfolino um ein externes EEPROM ergänzen
- » Stromspartechniken mit dem ATtiny

Checkliste

- Zeitaufwand:** 2 bis 4 Stunden
- Kosten:** ca 1 Euro

Mehr zum Thema

- » Nikolai Radke, Der Schimpfroboter, Make 5/24, S. 8
- » Florian Schäffer, ATtiny statt Arduino, Make 2/24, S. 72
- » Daniel Bachfeld, Überblick Speicherarten, Make 1/24, S. 104

Material

- » 24AA64 EEPROM oder größer
- » 2x2-kΩ-Widerstände (optional)
- » Arduino UNO

Software

- » Arduino IDE 2.x
- » Serielles Terminal, z. B. CoolTerm

Alles zum Artikel im Web unter make-magazin.de/xzfq

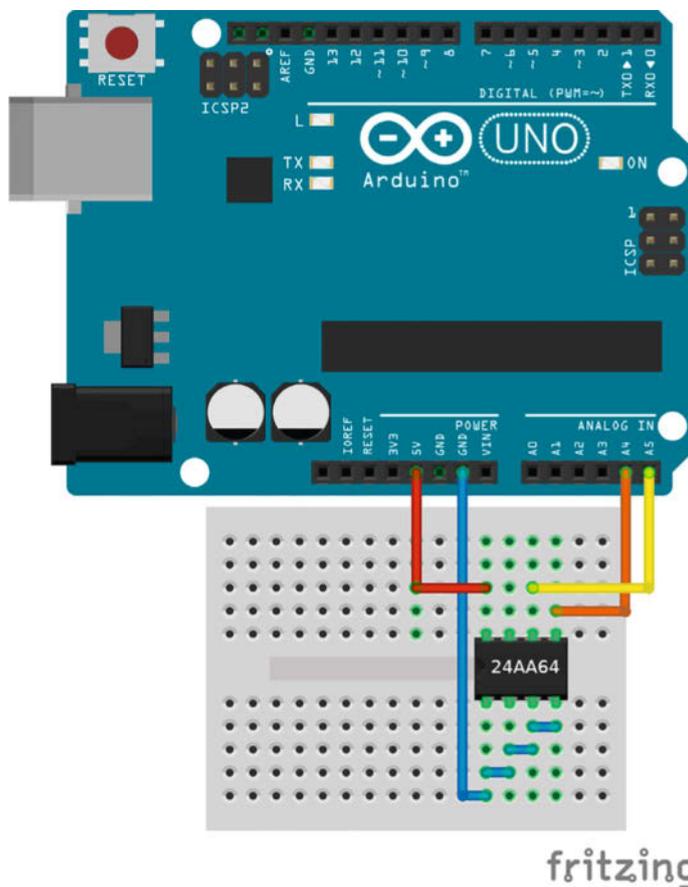


Bild 2: Beschreiben des EEPROMs mithilfe eines Arduino UNO über den I²C-Bus

Der I²C-Bus

Der „I-Square-C“-Bus oder umgangssprachlich „I-Two-C“ wurde 1982 von Phillips zur seriellen Kommunikation zwischen verschiedenen Schaltungen untereinander entwickelt. Ein Mikrocontroller kann als Master viele verschiedene Geräte kontrollieren. Der Bus benötigt neben Spannung und Masse die Leitungen SDA (Serial Data, die eigentlichen Inhalte) und SCL (Serial Clock, Taktimpulse). Die Adressierung der Geräte ist 7-bittig, manchmal auch 10-bittig möglich.

Daher sollte nur geschrieben werden, wenn es notwendig ist. Eine weitere Grenze ist die Kapazität: Mit 512 Bytes beim ATtiny85 ist es als Speichererweiterung zusätzlich zum Flashspeicher nur bedingt zu gebrauchen, das wären lediglich 51 Worte. Zumindest zu dieser Grenze bietet ein externes EEPROM eine Alternative mit geringem Strom- und Platzbedarf.

Externes EEPROM

Im letzten Artikel (Make 5/24) haben wir Schimpfolino ohne EEPROM gebaut. Sein Wortschatz von 450 Wörtern mit insgesamt 4500 Zeichen wurde im Flash-Speicher des ATtiny, dem PROGMEM, abgelegt. Um nun sein gesamtes entsetzliches Potenzial zu entfesseln,

speichern wir den kompletten Wortschatz von 800 Wörtern mit 8000 Zeichen in ein 24AA64-EEPROM und lesen ihn mit dem ATtiny aus. Dieser Chip aus selbigem Hause Microchip ist wie der ATtiny im handlichen DIP-8-Gehäuse erhältlich (Bild 1). Die 24 ist die Bezeichnung für die Teileserie (serielle EEPROMs), AA steht für die Betriebsspannung (1,7 bis 5,5V) und 64 für die Kapazität in KBit. Auch Varianten mit deutlich mehr Speicher bis maximal 512 KBit wie beim 24AA512 werden von Schimpfolino unterstützt. Das wären schon über 6550 Worte, was viel Kreativität und einen intensiven Hang zur Polemik erfordert.

EEPROM beschreiben und lesen

Zunächst wird das EEPROM mit einem Arduino UNO verbunden (Bild 2). Die Kommunikation erfolgt über den I²C-Bus (siehe Kasten „Der I²C-Bus“). Pin 5 wird mit SDA verbunden, Pin 6 mit SCL. Beim Arduino liegen die Bus-Pins bei A4 (SDA) und A5 (SCL). Über die Pins 1 bis 3 lässt sich die I²C-Adresse des EEPROMs festlegen. Für die Adresse 0x50 wurden daher die Pins mit GND verbunden. Pin 7 bleibt leer. Wird er jedoch mit VCC verbunden, wird das EEPROM schreibgeschützt.

Moment mal, braucht I²C nicht Pull-up-Widerstände? Ja, das Datenblatt des 24AA64 verlangt danach, ideal wären Widerstände mit 2kΩ zwischen SDA und VCC sowie SCL und VCC. Der Arduino hat jedoch einen 20-kΩ-Pull-up-Widerstand an jedem I/O-Pin. Diese werden bei I²C-Verbindungen hochgezogen, sofern die Bibliothek Wire.h verwendet wird. Das ist zwar nicht der korrekte Wert, aber arbeitet dennoch zuverlässig bei langsamen Verbindungen. Und das hier ist eine sehr langsame Verbindung. Wer unsicher ist, kann natürlich gerne externe Pull-ups hinzufügen. Aber wie sieht das später bei Schimpfolino aus? Das I²C-Display bringt bereits passende Widerstände auf dem Board mit!

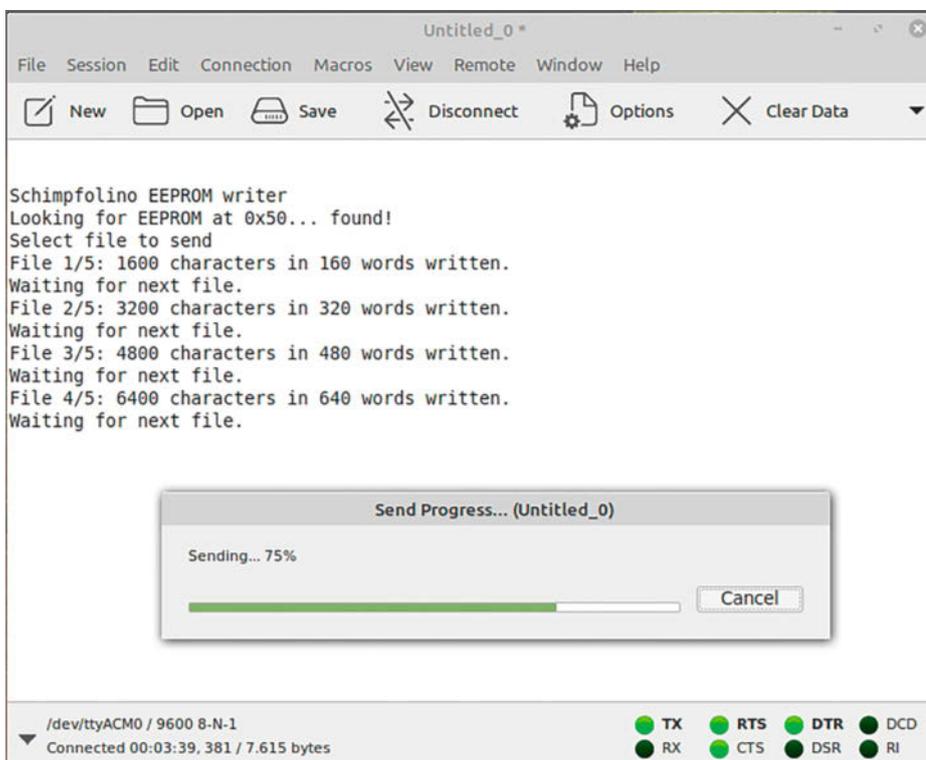


Bild 3: Mit CoolTerm werden über den Arduino Daten zum EEPROM übertragen.

Verbinden mit dem seriellen Terminal

Um nun Schimpfolinos EEPROM-Hirn zu füllen, braucht er eine passende Software. Im Schimpfolino-Ordner unter src/eprom liegen im Unterordner files die Wortlisten und in eprom_writer der gleichnamige Sketch, der nun auf den Arduino gespielt werden muss (siehe Link in der Kurzinfor).

Als serielles Terminal soll hier exemplarisch CoolTerm verwendet werden, da es für die gängigen Plattformen Windows, Linux und Mac erhältlich ist (Bild 3). Die Bedienung ist denkbar einfach (siehe Kasten „Serielle Verbindung mit CoolTerm“).

Der Arduino wartet geduldig auf das Einstecken des EEPROMs und öffnet dann eine

Serielle Verbindung mit CoolTerm

1. Auf „Options“ klicken, Port und Einstellungen prüfen: Baudrate: 9600, Data Bits: 8, Parity: None, Stop Bits: 1. Alles sollte voreingestellt sein.
2. Links auf „Transmit“ klicken und den Haken bei „Use transmit character delay – Delay (ms): 3“ setzen, dann „OK“ klicken.
3. Auf „Connect“ klicken und die Verbindung herstellen.
4. Unter „Connection - Send File(s)...“ auswählen, dann „Select File...“ klicken.
5. Im Ordner „src/eprom/files“ die Datei „eprom1.txt“ auswählen und „Öffnen“ drücken.
6. Den Fortschrittsbalken bewundern und Schritt 5 mit den anderen Dateien bis eprom5.txt wiederholen.

serielle Verbindung. Danach wartet der Mikrocontroller auf Daten, die jeweils mit der Verzögerung von 3 ms – denn EEPROMs sind langsam – byteweise geschrieben werden. Dabei gibt der Sketch dienstfertig Auskunft über das, was er als Bindeglied zwischen Computer und EEPROM tut.

Für so eine Verbindung sind im Wesentlichen nur ein paar Zeilen nötig. Das Listing „Beispielsketch zum Beschreiben des EEPROMs“ stellt eine verkürzte und modifizierte Version des Codes zur Veranschaulichung dar: Es wird eine serielle Verbindung zum Computer mit 9600 Baud eröffnet, analog zu den Einstellungen von CoolTerm mit den Grundeinstellungen SERIAL_8N1. Wer jemals andere Einstellungen benötigt, kann sie als zweiten Parameter in Serial.begin() übergeben.

Mit Wire.begin() wird die Verbindung zum I²C-Bus hergestellt. Als Mastergerät muss keine Adresse als Parameter übergeben werden. Die folgende while-Schleife wartet auf den Empfang eines Bytes vom Terminal, das dann auf Ausrufezeichen (0x21) als Markierung für das Ende der Datei und auf Line Feed (0x0A) geprüft wird, das Fehler erzeugen würde. Das Format ist dabei UTF-8. Die Adresse wird um eins erhöht und die Hauptschleife wiederholt sich. Die Funktion write_byte() schreibt entsprechend ein einzelnes Byte an die angegebene Adresse ins EEPROM mit der I²C-Adresse 0x50. Diese ist vom Typ uint16_t und muss daher in zwei Bytes aufgeteilt nacheinander gesendet werden, zunächst das Most Significant Byte (MSB), dann das Least Significant Byte (LSB). Grob gesagt: Das LSB ist der Rest der Division durch 256, also das untere Byte.

Der Sketch eeprom_writer.ino macht noch ein wenig mehr als dieses Beispiel. Neben Textausgaben am Terminal werden auch die Wörter gezählt und ihre Anzahl als Startadressen der einzelnen Listen in den ersten Adressen 0 bis 9 abgelegt. Die Startadressen werden dann von Schimpfolinos Gehirn ausgelesen, damit er weiß, wo welche Liste beginnt.

Ähnlich wie die Schreibfunktion ist die Lesefunktion aufgebaut (siehe Listing „Beispielfunktion zum Lesen eines Bytes“). Es wird eine Verbindung zur Adresse 0x50 hergestellt, dann das MSB und das LSB der zu lesenden Adresse übergeben. Schließlich wird das gewünschte Byte empfangen und von der Funktion übergeben.

Eigene Wortlisten erstellen

Wem 800 Wörter nicht reichen, der kann diese gerne ergänzen oder gar ganz andere Varianten kreieren, wie einen Komplimentino oder ein Ekellino – wer es braucht. In dem ersten Artikel wurden die fünf Wortlisten, hier eeprom1.txt bis eeprom5.txt, vorgestellt. Jedes

Beispielfunktion zum Lesen eines Bytes

```
uint8_t read_eeprom(uint16_t address) {
    Wire.beginTransmission(0x50);
    Wire.write(address >> 8);
    Wire.write(address & 0xFF);
    Wire.endTransmission();
    Wire.requestFrom(0x50, 1);
    return Wire.read();
}
```

Beispielsketch zum Beschreiben des EEPROMs

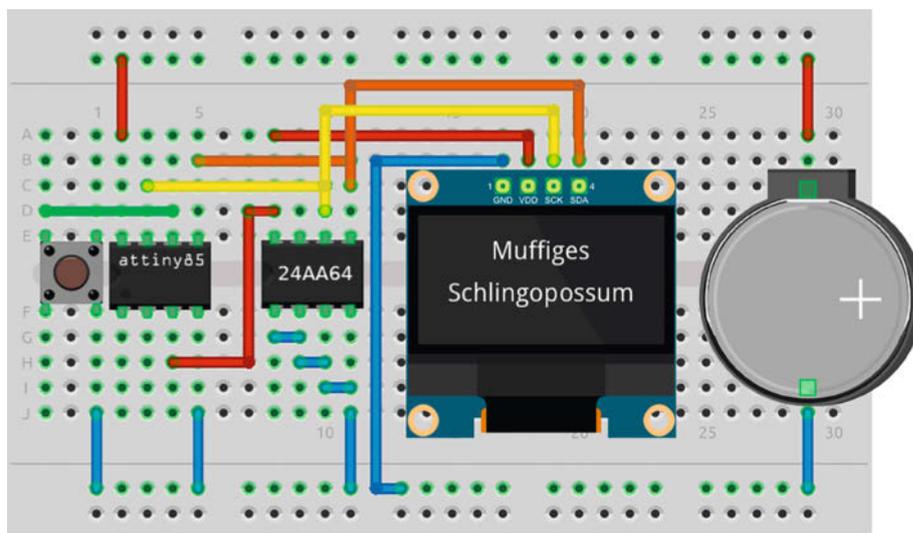
```
#include <Wire.h>

uint16_t address = 0;
uint8_t c;

void setup() {
    Serial.begin(9600);
    Wire.begin();
}

void loop() {
    while(Serial.available() == 0);
    c = Serial.read();
    if ((c != 0x0A) && (c != 0x21)) {
        write_byte(address, c);
        address ++;
    }
}

void write_byte(uint16_t address, uint8_t data) {
    Wire.beginTransmission(0x50);
    Wire.write((uint16_t)(address >> 8));
    Wire.write((uint16_t)(address & 0xFF));
    Wire.write(data);
    Wire.endTransmission();
    delay(1);
}
```



fritzing

Bild 4: Schimpfolino komplett mit EEPROM

Wort hat genau zehn Zeichen. Bei kürzeren Wörtern ist das Auffüllen mit Leerzeichen Pflicht! Das letzte Zeichen einer Datei muss ein ! sein. Die Bearbeitung muss mit einem Texteditor erfolgen. Deutsche Umlaute müssen umgewandelt werden: ä zu #, ö zu \$, ü zu %, ß zu *, Ä zu &, Ö zu ' und Ü zu (, denn in UTF-8 bestehen Umlaute aus zwei Bytes, was die Verarbeitung deutlich erschweren würde. Daher werden übrige, ein Byte lange Zeichen

verwendet, die Schimpfolino wieder in Umlaute konvertiert.

Wem das zu viele Regeln sind, dem steht im Verzeichnis src/EEPROM/files mit EEPROM.html ein Browser-basierter Editor zur Verfügung, der Leerzeichen automatisch auffüllt und Umlaute konvertiert. Aufgrund der begrenzten JavaScript-Fähigkeiten des Erstellers ist er jedoch noch reichlich Verbesserungsbedürftig.

EEPROM hinzufügen

Neben der Spannungsversorgung wird das EEPROM wie beim Arduino in Bild 2 mit SDA und SCL (zusammen mit dem Display) mit dem ATtiny verbunden, die Adresspins mit GND (Bild 4). Der im letzten Artikel verwendete Sketch erkennt das EEPROM und ignoriert seinen internen Wortschatz. Fertig ist die risige Kleckerwuppe.

Strom sparen mit dem ATtiny

Im Folgenden sollen nun die wichtigsten Techniken zur Einsparung von Strom erklärt werden: Wahl der Spannung, Sleep Mode mit Interrupts, das Power Reduction Register und das Schalten externer Geräte.

Welche Spannung reicht aus

Zunächst sollte die benötigte Spannung ermittelt werden. Je geringer die Spannung, desto geringer der Verbrauch. Dazu genügt ein Blick in die (gerne unübersichtlichen) Datenblätter. Ein ATtiny benötigt bei 8 MHz 2,7 V (Bild 5). Das 24AA64-EEPROM braucht 1,7V. Aber was ist mit dem Display? Hier variieren die Datenblätter der verschiedenen Anbieter in den Angaben für ein und denselben Displaytyp. Einen Hinweis gibt ein Eintrag eines der vielen Datenblätter: „VDD = 1.65V to 3.3V for IC logic“. Messungen ergaben, dass das Display bei ca. 1,7V problemlos, aber sehr funzelig läuft. Mit höherer Spannung steigt die Displayhelligkeit, bei 2V leuchtet sie quasi voll. Alle drei Geräte benötigen demnach Spannungen unter 3 V.

Ein Betrieb mit höherer Spannung, z. B. 4,5V, würde den Stromverbrauch des ATtiny um 25 Prozent erhöhen. Als Batterie ist daher der Typ CR2032 mit 3V ideal. Sie ist platzsparend, langlebig, günstig und überall erhältlich. Ein ATtiny ist von Hause aus sparsam. Der ATtiny85 im Schimpfolino läuft bei 8 MHz, weniger würde die Ausgabe am Display unerträglich langsam machen – aus der Beleidigung soll ja kein Treppenwitz werden.

Gemäß des ATtiny-Datenblatts liegt bei 3V dessen Stromverbrauch bei etwa 3 mA. Das klingt erst einmal nach wenig, auf längere Zeit ist es jedoch eine ganze Menge. An einem Tag beträgt der Stromverbrauch somit bereits $3\text{ mA} \times 24\text{ h} = 72\text{ mAh}$. Schimpfolino benötigt insgesamt mit dem sehr sparsamen OLED-Display und dem EEPROM ohne jegliche Optimierungen etwa 5,1 mA, mit abgeschaltetem Display und EEPROM sind es 3,1 mA.

Die Kapazität der Batterie beträgt 230 mAh, allerdings nimmt sie mit höherer Belastung ab. Laut Datenblatt der verwendeten Batterie ist sie für einen Dauerstrom von 0,2 mA

ausgelegt. Bei 3,1 mA sinkt die Kapazität auf weniger als 100 mAh (Bild 6), zudem sinkt die Batteriespannung im letzten Drittel der Kapazität auf unter 2,7V. Hier beginnen Display und ATtiny nach den Angaben in den Daten-

blättern, den Dienst zu quittieren (Safe Operating Area), wobei eigene Messungen mit einem Labornetzteil gezeigt haben, dass dies erst unterhalb von 1,7V passiert. Eine weitere Begrenzung besteht darin, dass eine CR2032

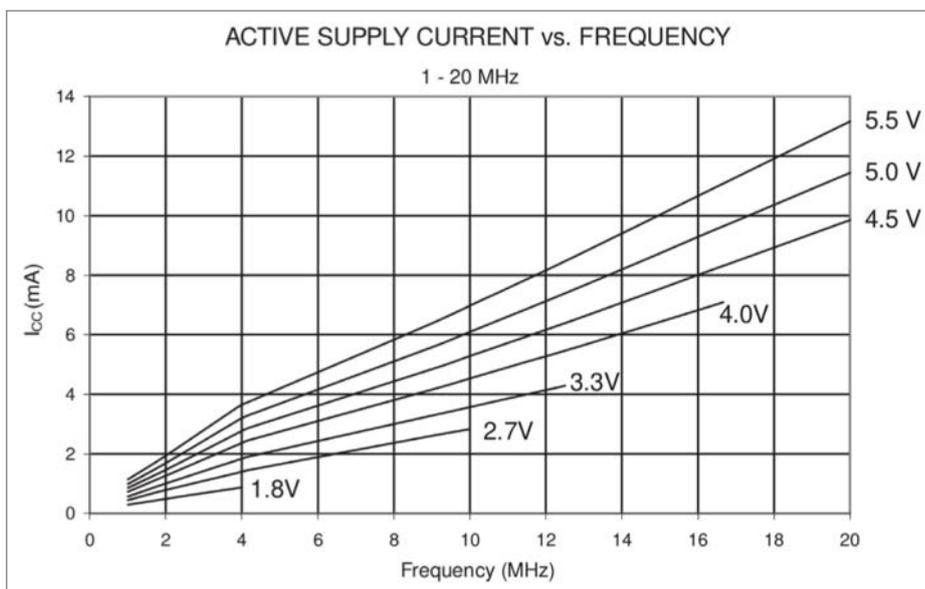


Bild 5: Stromverbrauch und Frequenz

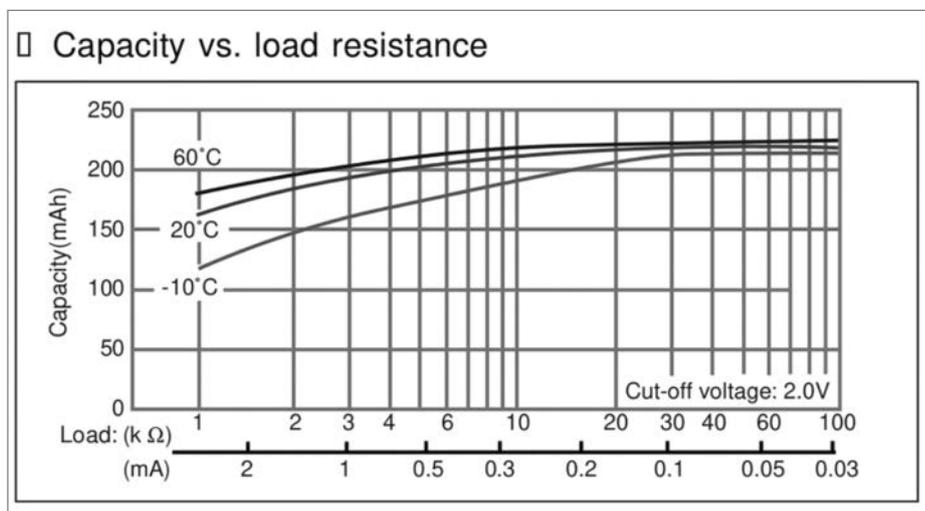


Bild 6: Kapazität und Last

bereits bei einer Spannung unter 2V keinen Strom mehr liefert. Dennoch ist die Batterie nach nicht mal 1 1/2 Tagen leer. Und das, ohne Schimpfolino überhaupt schimpfen zu lassen. Das würde ihn zu einem höchst unpraktischen Gerät machen, wenn er ständig, wenn es brenzlich wird, feige schweigt.

Die CR2032 ist also für kleine Ströme ausgelegt. Ein entschiedener Grund, Stromspartechniken zu verwenden, um den Stromverbrauch, sooft es geht, unter 0,2 mA zu halten. Schimpfolino wurde mit einigen solcher Techniken ausgestattet, die seinen Verbrauch auf 200 nA = 0,0002 mA im Schlafmodus reduzieren, was die Lebensdauer der Batterie auf theoretische 131 Jahre erhöht. Das ist doch eine ganze Menge. Für wenige Taktzyklen werden 5,1 mA benötigt. Wenn eine Beschimpfung acht Sekunden gezeigt wird, liegt der Verbrauch bei 1,7 mA. Ein Blick auf die Kurve (Bild 6) zeigt, dass die Kapazität der Batterie dabei noch etwa 140 mAh beträgt. Schimpfolino kann also fast dreieinhalb Tage dauerschimpfen.

Der Sleep Mode

Das wichtigste Werkzeug ist der Schlafmodus (Sleep Mode) des ATtiny. Darin befindet sich Schimpfolino, wenn er auf den nächsten Knopfdruck wartet. Wird der Knopf gedrückt, wird er über einen Interrupt (eine Unterbrechungsfunktion) geweckt. Die Stromversorgung von Display und EEPROM kommt von einem der Pins des ATtiny, der dann auf HIGH gestellt wird (Spannung liegt an). Der Schimpfwortgenerator, der im ersten Artikel erklärt wurde, generiert die Beleidigung, die am Display ausgegeben wird, danach legt sich der ATtiny sofort wieder für acht Sekunden schlafen, während das Display eingeschaltet bleibt.

Sollte währenddessen wieder der Knopf gedrückt werden, beginnt der Ablauf erneut. Nach den acht Sekunden wird der ATtiny von einem anderen Interrupt geweckt und schaltet Display und EEPROM wieder ab, indem der Ausgangspin auf LOW (Spannung liegt nicht an) gesetzt wird. Dann legt sich der Mikrocontroller wieder schlafen und wartet von Neuem auf den Knopfdruck. Die meiste Zeit macht Schimpfolino also nichts als Schlafen. Und das ist das ganze Geheimnis: schlafen, wenn es nichts zu tun gibt, und aus, was nicht an sein muss.

Um den ATtiny in den Schlaf zu versetzen, braucht es nur drei Zeilen (siehe Listing „ATTiny Schlafen schicken“). Die verwendete Library sleep.h ist sehr schlank und macht den Code gegenüber Registerbefehlen (dazu später mehr) übersichtlicher. Mit `set_sleep_mode()` wird der Schlafmodus `SLEEP_MODE_PWR_DOWN` festgelegt und mit `sleep_mode()` der ATtiny schließlich schlafen gelegt. In diesem

ATtiny schlafen schicken

```
#include <avr/sleep.h>
[...]
set_sleep_mode(SLEEP_MODE_PWR_DOWN);
sleep_mode();
```

Table 7-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes

Sleep Mode	Active Clock Domains					Oscillators	Wake-up Sources					
	clk _{CPU}	clk _{FLASH}	clk _{I/O}	clk _{ADC}	clk _{PCK}	Main Clock Source Enabled	INT0 and Pin Change	SPM/EEPROM Ready	USI Start Condition	ADC	Other I/O	Watchdog Interrupt
Idle			X	X	X	X	X	X	X	X	X	X
ADC Noise Reduction				X		X	X ⁽¹⁾	X	X	X		X
Power-down							X ⁽¹⁾		X			X

Note: 1. For INT0, only level interrupt.

Bild 7: Schlafmodi des ATtiny85

Zustand werden je nach Bedarf möglichst viele interne Funktionen wie Timer oder Schnittstellen deaktiviert und die CPU angehalten, um keine Instruktionen auszuführen.

Je nach Mikrocontroller stehen verschiedene Schlafmodi zur Verfügung. Der ATmega328 des Arduino UNO bietet sechs, der ATtiny25/45/85 drei Sleep Modes (Bild 7):

- **Idle:** Hier bleibt die CPU stehen, während alle anderen Geräte und Verbindungen aktiv bleiben. Es stehen zudem viele Aufweckmöglichkeiten zur Verfügung, auch `millis()` zählen weiter. Der Stromverbrauch sinkt bei 3V auf 1,75 mA. Der Modus wird mit `set_sleep_mode(SLEEP_MODE_IDLE)`; ausgewählt.
- **ADC Noise Reduction:** Mit 0,75 mA ist dieser Modus schon deutlich sparsamer. Alles, was die Messung des ADC (Analog-to-Digital-Converter) stören könnte, wird abgeschaltet, um genauere Messungen zu erzielen. Ausgewählt wird er mit `set_sleep_mode(SLEEP_MODE_ADC)`;
- **Power-down:** Hier werden alle Uhren bzw. Oszillatoren außer dem Watchdog (siehe weiter unten) angehalten und damit so ziemlich alles, was Strom braucht, deaktiviert. Der Verbrauch sinkt auf 0,235 mA. Sein Auswahlbefehl lautet `set_sleep_mode(SLEEP_MODE_PWR_DOWN)`; . Das ist der bevorzugte Schlaf für Schimpfolino. Gemäß Datenblatt wird empfohlen, ungenutzte Pins auf Eingang zu stellen und den Pull-up-Widerstand zu verwenden, also `pinMode(PB?,`

`INPUT_PULLUP)`; , damit mögliche Schwingungen keinen Strom verbrauchen können. Wie der Verbrauch auf winzige 200 nA gesenkt werden kann, wird allerdings erst später verraten.

Aufwecken mit Interrupts

Wenn ein Programm aufgrund eines Ereignisses unterbrochen werden soll, so werden dafür Interrupts verwendet. Wird z. B. am Computer eine Taste auf der Tastatur gedrückt, so unterbricht ein Interrupt des Tastaturtreibers die CPU. Das gilt auch für den ATtiny, z. B. durch Timer oder externe Geräte (etwa Sensoren). Schlafen ist schön – aber nicht für immer. Interrupts werden auch verwendet, um den Schlafmodus zu beenden. Bild 7 zeigt solche Wake-up-Sources. Schimpfolino verwendet zwei von ihnen: Watchdog und Pin Change.

Watchdog (Wachhund) ermöglicht ein zeitgesteuertes Aufwachen, während Pin Change die Veränderung an einem Pin überwacht, wenn also das Signal von HIGH zu LOW oder umgekehrt wechselt. Wird der Knopf gedrückt, wird der Pin Change Interrupt ausgelöst. Der Watchdog Interrupt weckt Schimpfolino, wenn das Schimpfwort acht Sekunden angezeigt wurde. Er hat einen eigenen Taktgeber, der, wenn er eingerichtet wurde, im Tiefschlaf weiter läuft. Dabei wird etwas mehr Strom verbraucht, was aber im Vergleich zur Einsparung, die der Tiefschlaf mitbringt, nur minimal ist.

Microchip Datenblatt ATtiny85, 08/2013, S. 34

WDTCR – Watchdog Timer Control Register

Bit	7	6	5	4	3	2	1	0
0x21	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDPO
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	X	0	0	0

Bild 8: Watchdog Timer Control Register

Watchdog Interrupt

Eingerichtet wird der Watchdog über das Watchdog Timer Control Register (WDTCR). Register sind festgelegte Bereiche im Speicher, die zur Steuerung des Mikrocontrollers benötigt werden. Beim ATtiny haben interne Register 8 Bit. Im Data Direction Register (DDRB) beispielsweise können mit einzelnen Bits die Pins auf INPUT oder OUTPUT gestellt werden. Im PORTB können diese dann auf HIGH oder LOW gezogen werden. Alle Register werden im Datenblatt erklärt (siehe dazu auch Make 2/24).

Es gibt eine Bibliothek `wdt.h`, die alles sehr vereinfacht, allerdings hat sie einen entscheidenden Nachteil: Nach jedem Aufwachen wird ein Systemreset durchgeführt, was bei vielen

Projekten wenig sinnvoll ist. Es ist daher notwendig, die Bits des Registers (Bild 8) einzeln zu setzen (siehe Kasten „Watchdog für 8 Sekunden einstellen“): `cli()`; sperrt alle Interrupts. Das hat einen guten Grund: Die Programmierung des Watchdogs erfolgt in einer Timed Sequence. Die Kommandos müssen also in der richtigen Reihenfolge und innerhalb bestimmter Taktzyklen erfolgen, sonst ist sie ungültig. Ein in dieser Zeit ausgelöster Interrupt könnte zusätzliche Zyklen erzeugen.

Um den Reset zu verhindern, wird im Microcontroller Unit Status Register (MCUSR) das Bit Watchdog Reset Flag (WDRF) gelöscht. Das geschieht mit dem Kommando `MCUSR &= ~(1<<WDRF)`; Bitweise Operationen sind für Einsteiger etwas unübersichtlich, aber im We-

sentlichen einheitlich. Zum Setzen eines Bits wird `REGISTER |= (1<<BIT)`; ausgeführt, zum Löschen `REGISTER &= ~(1<<BIT)`; . Um ein Bit zu setzen und die anderen auf null zu setzen, wird `=` verwendet.

Nun beginnt mit dem Setzen der Bits Watchdog Change Enable (WDCE) und Watchdog Enable (WDE) die Timed Sequence, in der die Zeit zum Aufwachen anhand der vier Bits Watchdog Timer Prescaler (WDPO bis WDP3) zwischen 16 Millisekunden und acht Sekunden definiert wird (Bild 9). Für acht Sekunden müssen WDPO und WDP3 gesetzt werden, alle anderen Bits im Register werden auf null gesetzt. Zum Abschluss werden mit `sei()`; die Interrupts wieder entsperrt.

Vor dem Schlafmodus muss der Watchdog mit `WDTCR |= (1<<WDIE)`; gestartet werden, dabei wird das Bit Watchdog Timeout Interrupt Enable (WDIE) gesetzt. Nun wird geschlafen und nach acht Sekunden Nickerchen weckt Watchdog pflichtbewusst wieder auf. Bevor jedoch im Programmablauf fortgefahren wird, wird die Interrupt Service Routine (ISR) aufgerufen. Sie hat den Parameter `WDT_vect`, den Interrupt-Vektor bzw. die Adresse, zu der bei einem Interrupt gesprochen wird. Jede Interrupt-Methode hat einen eigenen Vektor.

Diese Funktion ist zwingend notwendig, kann aber leer sein, wie bei Schimpfolino. Dort könnten beispielsweise boolsche Variablen gesetzt werden, um im weiteren Ablauf Ereignisse auszulösen. Zu beachten ist hierbei, dass diese Variablen „volatile“ sein müssen, z.B. `volatile bool awake = false;`. Der Begriff `volatile` bedeutet, dass die Variable direkt aus dem SRAM und nicht aus dem Speicherregister geladen wird. Bei einem Interrupt könnte sonst ein falscher Wert ausgelesen oder übermittelt werden.

Zu guter Letzt wird das WDIE-Bit wieder mit `WDTCR &= ~(1<<WDIE)`; gelöscht. Watchdog ist aus und das Programm kann fortfahren. Der Watchdog kann wunderbar anstelle von Delays verwendet werden, wenn sie z.B. regelmäßig innerhalb der Hauptschleife auftreten. Auch als Absicherung gegen Programmabstürze ist er, ggf. mit Reset, denkbar. Wenn mehr als acht Sekunden benötigt werden, kann die Schlafzeit in einer `for`-Schleife oder Rekursion leicht verlängert werden.

Pin Change Interrupt an PB1

```
cli();
GIMSK |= (1 << PCIE);
PCMSK |= (1 << PCINT1);
sei();
[...];
set_sleep_mode(SLEEP_MODE_PWR_DOWN);
sleep_mode();
[...];
ISR(PCINT1_vect) {
[...];
}
```

Watchdog für 8 Sekunden einstellen

```
#include <avr/sleep.h>
[...];
cli();
MCUSR &= ~(1 << WDRF);
WDTCR |= (1 << WDCE) | (1 << WDE);
WDTCR = (1 << WDP0) | (1 << WDP3);
sei();
[...];
WDTCR |= (1 << WDIE);
set_sleep_mode(SLEEP_MODE_PWR_DOWN);
sleep_mode();
WDTCR &= ~(1 << WDIE);
[...];
ISR(WDT_vect) {
[...];
}
```

Table 8-3. Watchdog Timer Prescale Select

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V _{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16 ms
0	0	0	1	4K (4096) cycles	32 ms
0	0	1	0	8K (8192) cycles	64 ms
0	0	1	1	16K (16384) cycles	0.125 s
0	1	0	0	32K (32764) cycles	0.25 s
0	1	0	1	64K (65536) cycles	0.5 s
0	1	1	0	128K (131072) cycles	1.0 s
0	1	1	1	256K (262144) cycles	2.0 s
1	0	0	0	512K (524288) cycles	4.0 s
1	0	0	1	1024K (1048576) cycles	8.0 s

Microchip Datenblatt ATtiny85, 08/2013, S.46

Bild 9: Watchdog-Prescaler-Konfiguration

Pin Change Interrupt

Dieser Interrupt ist wesentlich leichter einzurichten. Im General Interrupt Mask Register (GIMSK) muss dazu das Bit Pin Change Interrupt Enable (PCIE) gesetzt werden: `GIMSK |= (1<<PCIE)`; (siehe Kasten „Pin Change Interrupt an PB1“). Nun wird im Pin Change Mask Register (PCMSK) dem Interrupt ein Pin zugewiesen. Dabei gilt PCINT0 für PB0 bzw. D0 bis PCINT5 für PB5 bzw. D5. Der Befehl dazu lautet bei Schimpfolino `PCMSK |= (1<<PCINT1)`;, da der Button, der den Interrupt auslösen soll, an PB1 hängt. Natürlich wird auch hier eine ISR benötigt. Der Parameter lautet jedoch `PCINT0_vect`. Die verwirrende Bezeichnung hat nichts mit dem Interrupt-Pin zu tun und sagt lediglich, dass der erste Interrupt-Vektor mit der Nummer 0 genutzt werden soll. Der ATtiny85 hat nur einen Vektor für Pin Change. Im Beispiel in dem Kasten „Pin Change Interrupt an PB1“ bleibt der Interrupt die ganze Zeit aktiv. Soll er deaktiviert werden, wird das Bit im PCMSK mit dem Befehl `PCMSK &= ~(1<<PCINT1)`; zurückgesetzt.

Interne Geräte schalten

Einige Teile des ATtiny lassen sich nicht nur schlafen legen, sondern auch jederzeit ein- und ausschalten, sofern sie nicht benötigt werden. Der Brown Out Detector (BOD), dessen Aufgabe es ist, einen Reset durchzuführen, sobald die Spannung zu niedrig wird, ist bereits im Arduino IDE deaktiviert.

Andere lassen sich über das Power Reduction Register (PRR) ausschalten: Timer0 mit Setzen des Bits PRTIM0, sofern u.a. `millis()`, `delay()` und PWM an PB0 nicht benötigt werden. Timer1 mit Bit PRTIM1, wenn u.a. PWM an PB1 und PB4 nicht gebraucht wird. Ebenso

kann das Universal Serial Interface (USI) mit dem Bit PRUSI deaktiviert werden, was jedoch die emulierten Schnittstellen SPI, UART und I²C und somit die Kommunikation mit EEPROM und Display deaktiviert. Das Abschalten der Geräte ist für den aktiven Modus und Idle-Schlafmodus gedacht. Im Tiefschlaf sind sie ohnehin deaktiviert. Timer0 und Timer1 werden nach dem Aufwachen automatisch wieder aktiviert, USI bleibt aus.

In etlichen Sketches taucht die Bibliothek `avr/power.h` auf. Sie enthält Makros, die, wie beschrieben, im PRR Bits setzen, z. B. `power_timer0_disable()`; für `PRR |= (1<<PRTIM0)`; bzw. `power_timer0_enable()`; für `PRR &= ~(1<<PRTIM0)`;

Neben den drei genannten Geräten ist noch ein weiteres Bit im PRR reserviert: der Analog-Digital-Converter (ADC). Der ADC wird für `analogRead()` benötigt und soll laut Datenblatt abgeschaltet werden, wenn keine analogen Messungen benötigt werden. In allen Schlafmodi bleibt er aktiv, insbesondere in `SLEEP_MODE_ADC`, denn dafür ist der ja da. Leider funktioniert das nicht mit dem PRR und `power_adc_disable()`;, sondern mit dem Löschen des Bits ADC Enable (ADEN) im ADC Control and Status Register A (ADCSRA) mit `ADCSRA &= ~(1<<ADEN)`;. Je nach angelegter Spannung wird der Stromverbrauch um ca. 200 – 350 µA reduziert, sodass im Tiefschlaf die im Datenblatt angegebenen 200 nA tatsächlich erreicht werden können.

Externe Geräte schalten

Nicht nur der ATtiny verbraucht Strom. Alleine ist er doch ein ziemlich uninteressantes Ding und wird daher mit weiteren Modulen oder Bauteilen verwendet. Bei Schimpfolino sind

es das Display und das EEPROM. Solche Bauteile brauchen mal mehr, mal weniger Strom, manche haben eigene Schlafmodi, aber auch dann zehren sie an der Batterie. Daher liegt es nahe, sie einfach abzuschalten, wenn sie nicht gebraucht werden.

Schimpfolino schaltet beide Geräte über PB4 des ATtiny mit einem einfachen `digitalWrite(4, LOW)`; ab. Dabei muss beachtet werden, dass pro Pin maximal 40 mA gezogen werden dürfen. Mit rund 2 mA sind Display und EEPROM locker im sicheren Bereich.

Jetzt ist es so weit. Schimpfolino verbringt die meiste Zeit im Tiefschlaf und schaltet alle nicht benötigten inneren und äußeren Verbraucher ab. Es bleibt nur der dämmernde ATtiny, der laut Datenblatt 200 nA zieht. Das hauseigene Multimeter zeigt dasselbe Ergebnis. Der Verbrauch im Schlaf wurde auf ein 25.500stel reduziert. Schimpfolino ist bereit für ein langes Leben voller gesundem Schlaf und ungesunder Pöbeleien.

Im nächsten Heft

Schön, so ein Steckbrett-Schimpfer. Aber mit der Verdrahtung wenig alltagstauglich, daher wird an Schimpfolinos Street Credibility gearbeitet. Wie wird aus dem Steckbrett mit KiCad eine gedruckte Schaltung? Und wie kommt die in ein furchteinflößendes Gehäuse? Die Make 1/25 klärt es auf. —mch



Alles wird USB-C

USB-C hat sich glücklicherweise als Standard bei vielen Endgeräten durchgesetzt. Den Laptop kann man damit laden, das Handy ohnehin (sogar iPhones) und viele Tastaturen und Mäuse verwenden es ebenfalls. Aber oft hat man noch dieses eine Gerät, das trotzdem Micro-USB oder Mini-USB verwendet. Wie man den Port eines solchen Gerätes durch USB-C-ersetzt, schauen wir uns in diesem Artikel an.

von Daniel Schwabe



Für den Umbau ist es wichtig, die Unterschiede zwischen USB-C und seinen Vorgängern zu erkennen. Micro- und Mini-USB verfügen jeweils über fünf Pins. Zwei für die Datenübertragung, zwei für Stromzufuhr und einen, der als „ID-Pin“ betitelt wird. Dieser wird für USB OTG (USB On-The-Go) verwendet. Das ist für die hier gezeigte Mod aber nicht relevant. Deshalb bleiben vier Pins übrig.

Ein voll bestückter USB-C-Port hat 24 Pins. Acht für Strom, zwölf für Datenübertragung, zwei zum Erkennen der Steckerausrichtung und zwei für einen Überspannungsschutz. Jetzt stellt sich natürlich die Frage: Wie komme ich von 24 Pins auf 4? Das schauen wir uns jetzt genau an.

Nicht alle USB-C-Ports sind gleich

USB-C-Buchsen gibt es bei einschlägigen Händlern in diversen Ausführungen, die verschieden viele Pins bieten. Das reicht von 16-Pins über 6-Pins bis hin zu Mini-Platinchen mit USB-C-Buchsen, auf denen nur die für uns wichtigen Pins V+, GND, D+ und D- angeschlossen sind.

Doch was genau unterscheidet diese 4-Pin-Boards vom Rest? USB-C verfügt normalerweise über zwei dedizierte Pins, die für das Erkennen der Orientierung und der genutzten Modi verantwortlich sind. Diese Pins nennt man CC-Pins. Sie erkennen, ob das angeschlossene Gerät eine Stromquelle oder ein Verbraucher ist, wie viel Strom das Gerät braucht oder ob es sich um eine spezielle Peripherie handelt.

Dafür gibt es aber schon vorgefertigte Boards, die diese Widerstände vorinstalliert

Kurzinfo

- » USB-Buchsen ablöten
- » Mini- und Micro-USB ersetzen
- » Unterschiede zwischen USB-C und USB-B

Checkliste



Zeitaufwand:
30 Minuten



Kosten:
8 Euro

Material

- » USB-C-Breakoutboards vierpolig
- » Klingeldraht

Werkzeug

- » Lötkolben mit Lötzinn
- » Heißluftpistole

Mehr zum Thema

- » Hans Borngräber, Handy als Steuerung für Bandmaschine, Make 6/24, S. 24
- » Justin Atkins, Löten mit einer Hand, Make 7/23, S. 98



Alles zum Artikel
im Web unter
make-magazin.de/xze6

haben. Außerdem geht die hier gezeigte Modifikation davon aus, dass die Geräte mit einem USB-A-2.0-auf-USB-C-Kabel genutzt werden.

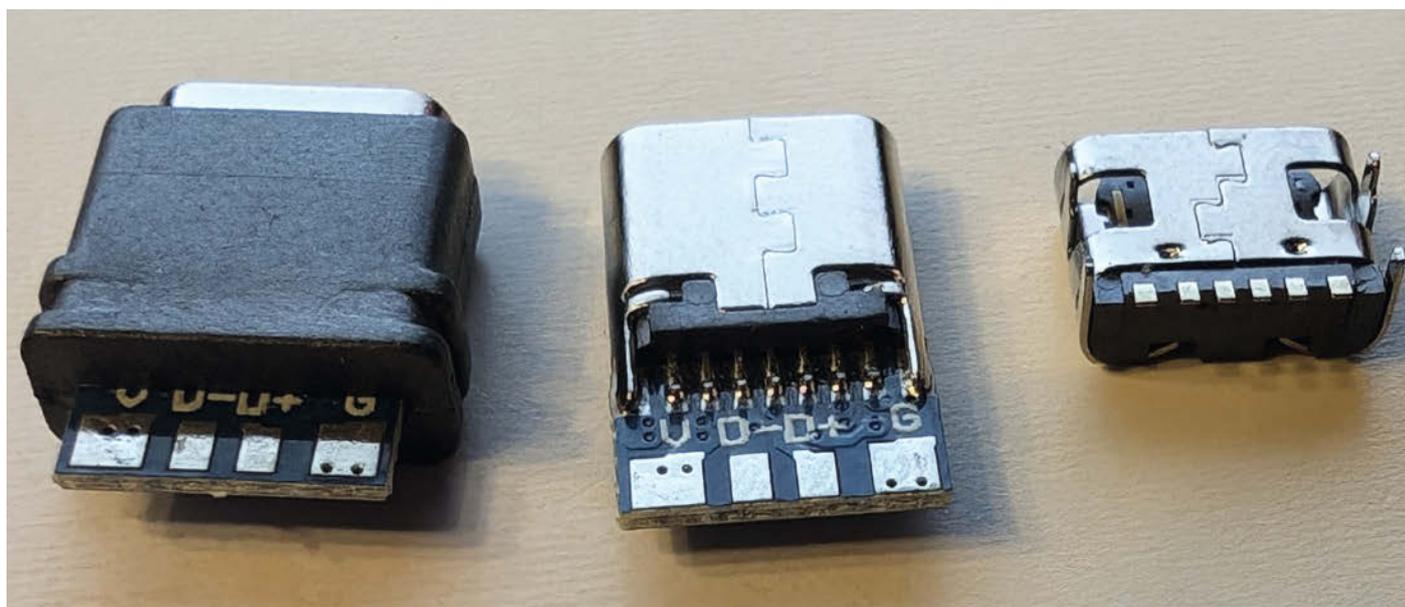
Löten

Hat man ein Gerät mit einer Mini- oder Micro-USB-Buchse, löst man diese am besten vom PCB ab, indem man eine Entlötstation benutzt. Mit der dazugehörigen Heißluftpistole (im

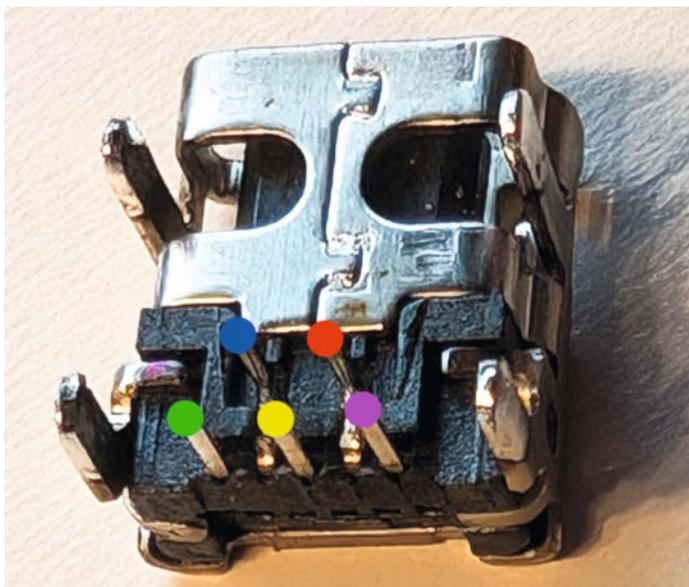
Endeffekt ein Föhn) kann man vorsichtig, mit einem engen Aufsatz, der den Luftstrom auf einen kleinen Punkt konzentriert, die Buchse erhitzen und dann vorsichtig lösen.

Hier darf keine Gewalt angewendet werden, sonst reißt man die Löt pads ab. Wenn das Lötzinn, das die alte Buchse am Board festhält, flüssig ist, fällt das Bauteil fast von selbst ab.

Ist die alte Buchse entfernt, kann man sich mal die Pads genauer anschauen. Die winzigen Kontakte erfordern einiges an Geschick



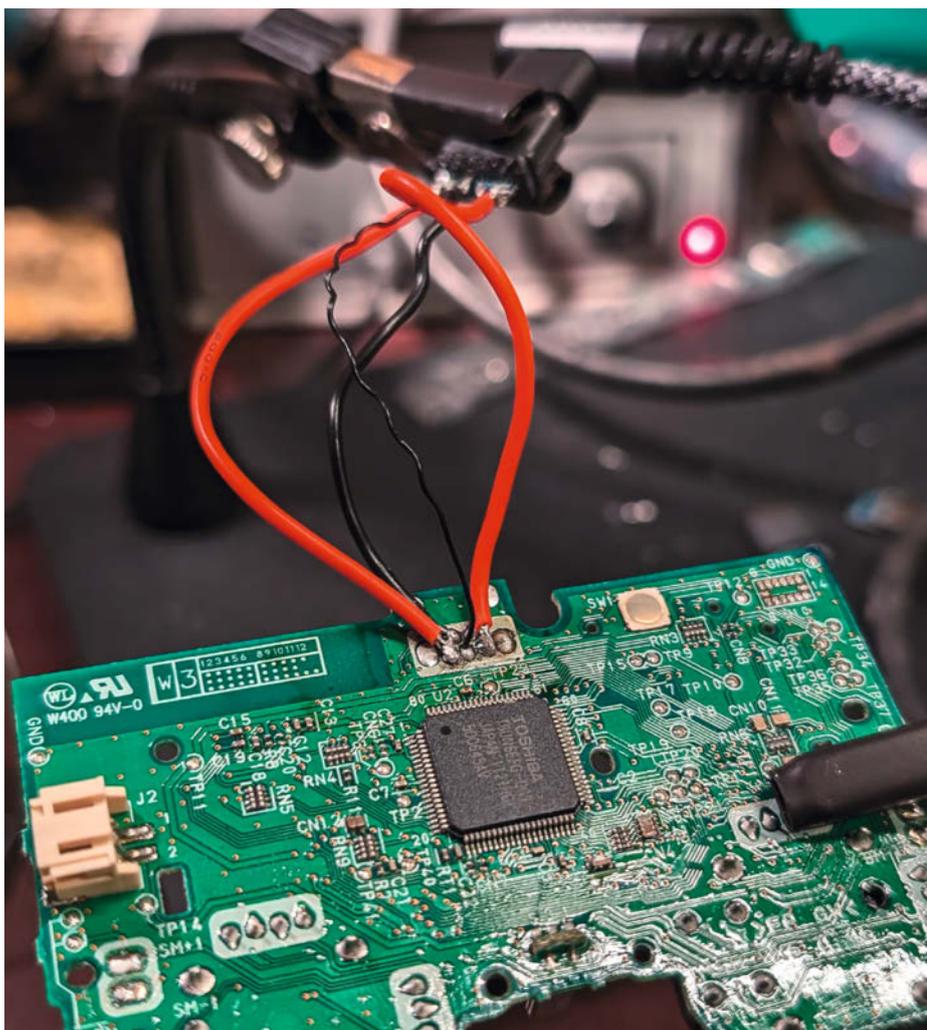
Die beiden Boards links erleichtern einem den Hack.



Das ist eine Mini-USB-Buchse. Grün ist GND, Gelb ist D+, Pink ist V+, Rot ist D- und Blau ist der ID-Pin.



Das ist eine Micro-USB-Buchse. Grün ist GND, Gelb ist D+, Pink ist V+, Rot ist D- und Blau ist der ID-Pin.



Wegen der Breakout-Boards braucht man kurze Kabel zum Anlöten.

beim Lötten. Dankenswerterweise haben immerhin die vorbereiteten Boards größere Löt pads, sodass man nur auf der Peripherie-Platine friemeln muss. Wie man unschwer erkennen kann, würde eine reine USB-C-Buchse weder mechanisch noch von der Pin-Belegung her passen.

Nun muss man die Kontakte verbinden. Doch welcher ist welcher auf der alten Peripherie? Dafür reicht ein Blick in die Pin-Belegungsdiagramme. Achtung hierbei: Das sind die Pin-Belegungen der Buchse. Nicht die der PCBs, auf denen sie aufgetragen wurden. Das bedeutet: Man muss nicht zwischen + und - verbinden, sondern zwischen + und + und - und -.

Wenn man ein Gerät nutzt, das nur mit Strom versorgt werden muss und keine Daten überträgt, muss man natürlich nur GND und V+ verlöten. D+ und D- können frei bleiben. Viele Geräte haben trotzdem die Pads für diese beiden Pins auf dem PCB. Diese sind dann zwar verlötet, allerdings nicht wirklich verschaltet.

Bonus: USB-A

Die Idee zu diesem Artikel kam mir, weil ich ein sehr bizarres Gerät mein Eigen nenne: eine VGA-Capture-Card. Damit kann ich ein VGA-Signal über den Computer wie bei einer Webcam abgreifen und aufzeichnen. Dieses Gerät hat die Eigenart, dass es nicht über Micro- oder Mini-USB angeschlossen wird, sondern über ein USB-A-auf-A-Kabel. USB-A ist der große Desktop-USB-Anschluss.

Selbstverständlich habe ich das mitgelieferte USB-A-auf-USB-A-Kabel direkt verbum-



Auch so ein exotisches Gerät mit einem USB-A-Port kann dieser Hack zähmen.

melt und besitze natürlich auch kein weiteres. Das Anbringen einer USB-C-Buchse ist bei so einem Gerät aber sehr einfach. Vier große Kontakte mit viel Platz dazwischen. Ich musste nicht einmal die alte Buchse auslöten.

Die Gehäuse

Nachdem man sich nun der alten USB-Ports entledigt und sichergestellt hat, dass alles noch richtig funktioniert, muss man sich noch um das Gehäuse kümmern. Die Aussparung des alten Ports ist zu klein. Hier hilft leider nur Aufschneiden. Wenn man zu einem in Plastik gegossenen USB-C-Port gegriffen hat wie im Vergleichsbild ganz links, kann man gut mit einer Presspassung ein sauberes Äußeres des Gerätes bewahren.

Fazit

Durch vorgefertigte Boards, die einem direkt die wichtigen Pins für USB 2.0 auf angemessene große Pads herausführen, ist das ein verhältnismäßig einfacher Hack.

USB-C-Kabel besitzt man meist mehr, als man braucht, und dass man die Stecker egal wie rum einstecken kann, war geradezu revolutionär.

Aber werde ich mir für alle meine USB-2.0-Geräte die Arbeit machen und andere Buchsen einlöten? Ich denke eher nicht. Dafür sind die Kontakte auf den Boards wirklich viel zu klein und für die Breakout-Boards muss auch genug Platz im Gehäuse sein. Trotzdem finde ich es gut zu wissen, dass es an sich einfach geht und die Möglichkeit besteht. —das



Die USB-C-Buchse ist größer als die originale Mini-USB-Buchse. Das Gehäuse muss angepasst werden.

NEU: c't Desinfec't 2024/25

Das Notfall-System für den Ernstfall

GRATIS:
Signatur-Updates
bis Oktober 2025



Komplett auf
32 GByte USB-Stick.
Desinfec't startet
direkt vom Stick.



shop.heise.de/desinfec24

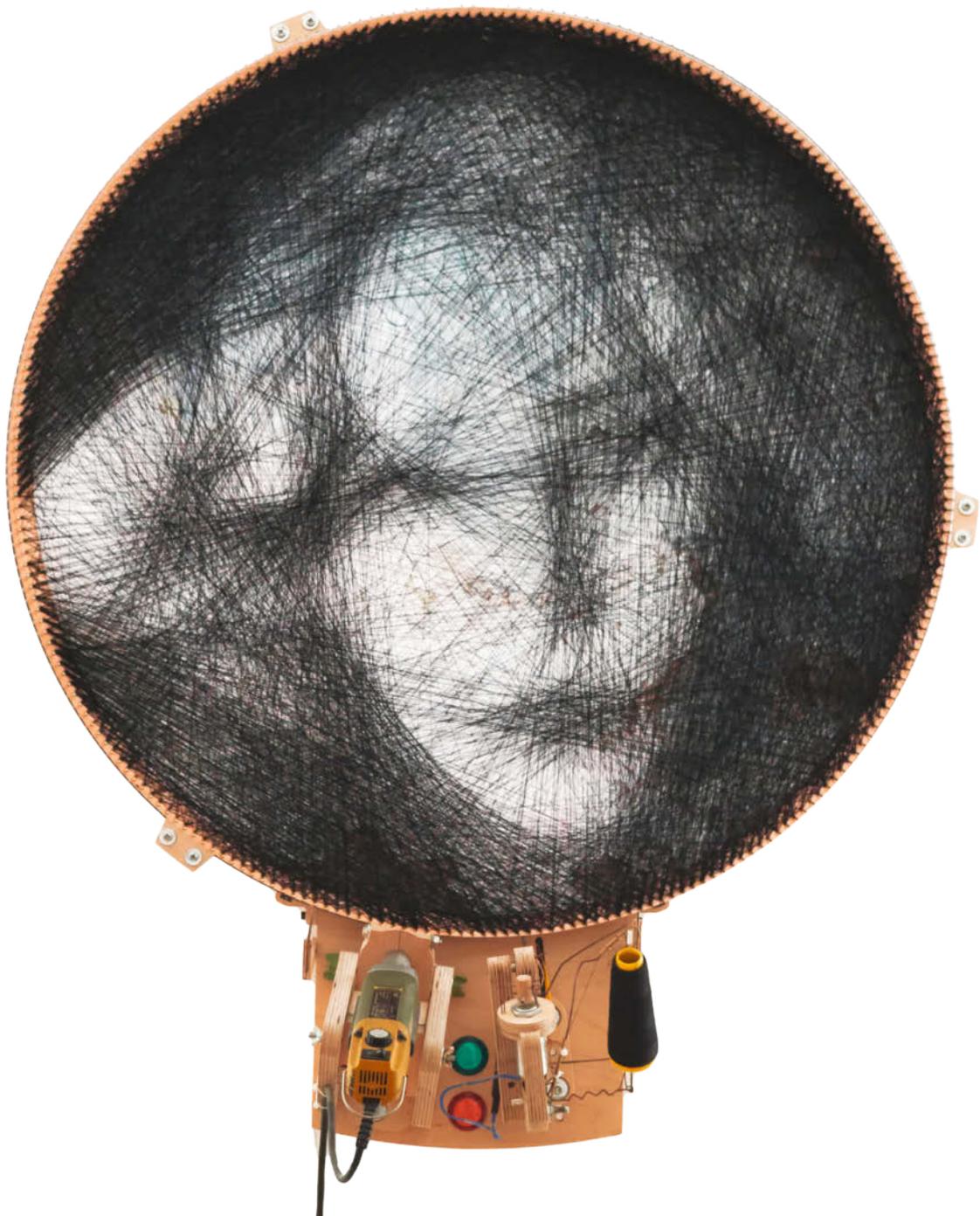
Generell portofreie Lieferung für Heise Medien- oder Maker Media Zeitschriften-Abonnenten oder ab einem Einkaufswert von 20 € (innerhalb Deutschlands). Nur solange der Vorrat reicht. Preisänderungen vorbehalten.



Die String-Art-Maschine

Dieser selbst gebaute String-Art-Automat spannt automatisch einen kilometerlangen Faden zu einem Bild und verbindet dabei Technik mit Ästhetik. Als Vorlage braucht er lediglich ein Foto oder eine Grafik.

von Philip Oettershagen



Mit String-Art lassen sich beeindruckende Bilder erzeugen, die aus einem einzelnen Faden gesponnen werden. Dabei besteht die Leinwand in der Regel aus einem runden Rahmen (z. B. einem Holzreifen), der an den Kanten mit Nägeln versehen ist. Verbindet man diese Nägel nach einem bestimmten Muster mit dem Faden, ergibt sich durch die tausendfachen Überlagerungen und Kreuzungen das Motiv. Aus ein paar Metern Entfernung interpoliert das Gehirn diese abstrakten Strukturen zu einem konkreten Bild.

Solch eine String-Art lässt sich mit viel Geschick zwar auch von Hand fertigen, auf Dauer ist eine Maschine, die diese Aufgabe übernimmt, aber zumindest schneller. Das Umwandeln einer Pixelgrafik in die Pfade (als G-Code), mit denen sich das Gerät bewegen soll, kann man dabei bequem einem Algorithmus überlassen.

In diesem Artikel stelle ich meine Version einer String-Art-Maschine vor und wie sie funktioniert. Online gibt es ergänzend dazu eine Bauanleitung (siehe Link in der Kurzinfor).

Der Anfang des Projekts

Auf die Idee, mir eine String-Art-Maschine zu bauen, kam ich ursprünglich durch ein Video von Barton Dring aus dem Jahr 2019 (verlinkt in der Kurzinfor). Er stellt darin den Prozess vor, mit dem er so ein Gerät gebaut hat, und zeigt, wie die Maschine einen Teller dreht und die Fäden zwischen den Nägeln spinnt. Die Kombination aus Technik und Ästhetik hat mich sofort in ihren Bann gezogen.

Allerdings ist der Proof-of-Concept-Charakter von Drings Maschine nicht auf hohe Geschwindigkeit optimiert; für mein Design hatte das aber eine höhere Priorität, da ich die Kunstwerke verkaufe und damit mein Geld verdiene.

Die Geschwindigkeitsoptimierung ließ sich durch verschiedene Anpassungen erzielen. Zum einen habe ich den massiven Holzteller, der in der Inspiration genutzt wird, durch einen schmalen Reifen ersetzt. Das spart sehr viel Materialgewicht ein, wodurch die Motoren nicht so viel Masse bewegen müssen. Einen weiteren Geschwindigkeitsgewinn konnte ich durch das Ersetzen eines Motors erzielen, der eine Klappbewegung des Fädelmechanismus mit einem Elektromagneten durchführt.

Zusätzlich entschied ich mich recht früh in der Planung für die Wandmontage, um die Maschine nicht ständig abbauen zu müssen. Als Wandmöbel geplant, fügt sich der Automat zudem sehr gut in eine Wohnung ein.

Eine prinzipiell lauffähige Version war nach einem halben Jahr fertig. Bis alles zuverlässig funktionierte, vergingen aber noch ein paar Monate. Denn obwohl das technische Funktionsprinzip eigentlich recht einfach ist, taten sich beim Nachbau an vielen Stellen Fragen

Kurzinfor

- » Digitale Bilder in String-Art umwandeln
- » Maschine mit G-Code steuern
- » Automaten aus Holz herstellen

Mehr zum Thema

- » Bernd Heitkamp, StringArt: Wie Sie Kunst aus dünnen Fäden selber wickeln, heise+
- » Johannes Börnsen, Siebdruck mit Leim und Laser, Make 5/24, S. 82



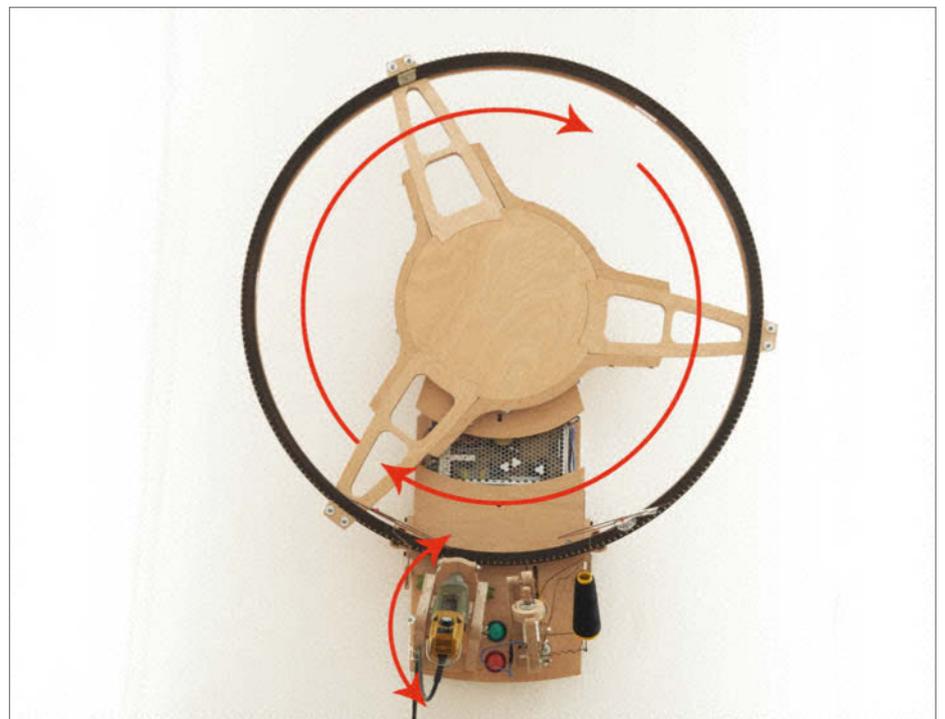
auf, die knifflig waren und die ich unabhängig voneinander lösen musste.

Automatisch spinnen

Auch das Prinzip meiner Maschine besteht darin, dass sich der Rahmen dreht, in dem das Bild gespannt wird. Dadurch kann die Technik, die den Rahmen vorbereitet und den Faden spannt, stationär bleiben, sodass sich alle Komponenten gut erreichen lassen. Bevor ich auf diese einzeln eingehe, erkläre ich kurz, wie meine Maschine arbeitet.

1. Als Erstes läuft ein eingespannter, runder Rahmen, in den die String-Art gesponnen werden soll, einmal rundherum in der Maschine. Während dieses Durchgangs werden Löcher in den Rahmen gebohrt.

2. Der Rahmen wird ausgespannt, die Löcher mit Nägeln versehen und der Rahmen wieder eingespannt.
3. Jetzt wird der Faden gespannt. Um einen Start- bzw. Endpunkt einer Linie zu erstellen, bleibt das Rad so stehen, dass sich die Vorrichtung zum Spannen des Fadens zwischen zwei Nägeln befindet.
4. Der Fädler, bisher in Ruheposition auf der Innenseite des Nagelkranzes verharrend, bewegt sich im zweiten Schritt zwischen den Nägeln hindurch auf die Außenseite.
5. Das Rad dreht sich eine Einheit weiter.
6. Jetzt kann der Fädler wieder in seine Ruheposition zurückkehren und der Faden verläuft nun außen um den Nagel und bildet damit einen Fixpunkt. Jetzt wird der nächste Punkt angefahren.



Bei dieser Konstruktion kommt das reine Spinnen des Bildes mit zwei Achsen aus: der Drehung des Rades und einer kurzen linearen oder drehenden Bewegung des Fädlers.

Material für ein String-Art-Kunstwerk

Der genutzte Faden muss einerseits dünn sein, denn er baut sich an den Nägeln schnell auf. Andererseits ist Reißfestigkeit sehr wichtig, damit alles der hohen Belastung durch die vielen Umlenkungen bei hoher Laufgeschwindigkeit standhält. Overlock-Garn bietet hier einen guten Kompromiss. Eine Spule fasst ca. 2,7 km, was für ein Bild meistens reicht.

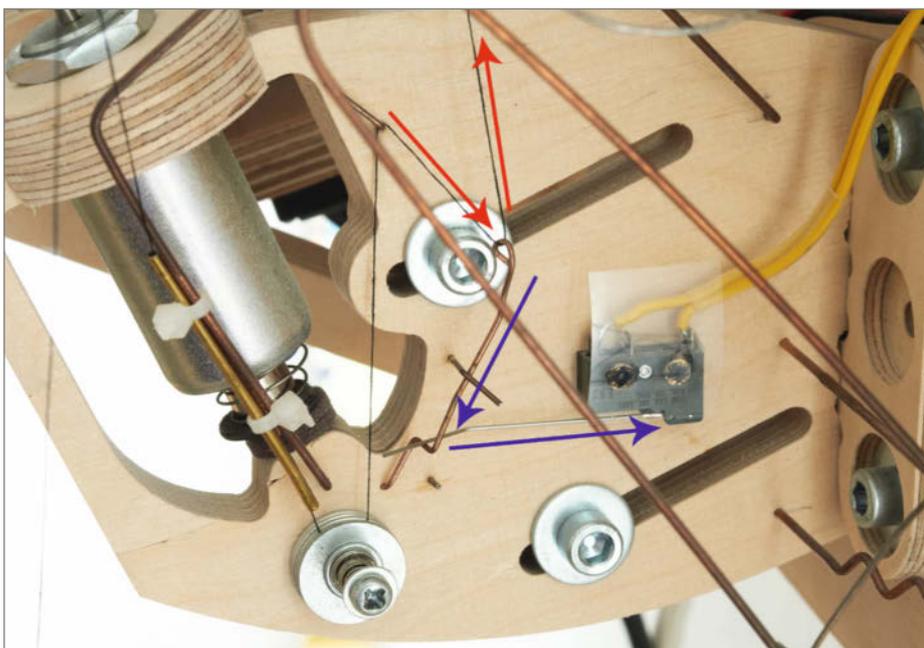
Bei den Nägeln habe ich mich für vermessingte Stahlnägel entschieden, diese sind etwas günstiger als reine Messingnägel. Nach Versuchen mit Baumarktware

schwankender Qualität bin ich zu Markenware aus dem Eisenhandel gewechselt. Der Durchmesser von 1,6 mm muss exakt zum Bohrerdurchmesser passen, 20 mm Länge reichen.

Der Reifen, der als Rahmen für die String-Art fungiert, ist ein einfacher Hula-Hoop-Reifen aus mehrschichtig verleimtem Buchenholz. Diese Reifen kann man unbehandelt mit gefassten Kanten im Sportfachhandel kaufen. Vor dem Einsatz sollte man ihn allerdings noch abschleifen.



In Reih und Glied: Die Nägel müssen möglichst perfekt ausgerichtet sein.



Sollte der Faden (rote Pfeile) reißen, wird ein Schalter (an den gelben Kabeln) gelöst (über den Mechanismus mit den blauen Pfeilen), der die Maschine stoppt.

256 Nägel gleichmäßig verteilt

Der Rahmen meiner String-Art-Bilder hat einen Außendurchmesser von 800 mm. Je mehr Nägel darauf verteilt sind, desto höher ist theoretisch der spätere Detailgrad des Bildes. Allerdings werden die Abstände zwischen den Nägeln auch immer geringer. Man kann versucht sein, hier alles herauszuholen und so viele Nägel wie möglich unterzubringen, aber ab ca. 250 Nägeln ist der Detailzuwachs eher gering, dafür kommen Toleranzprobleme auf. Denn damit der Fädler später zuverlässig mehrere tausend Mal durch die Zwischenräume huschen kann, müssen diese breit genug und die Verteilung der Nägel so gleichmäßig wie möglich sein.

Die Nägel haben einen Kopfdurchmesser von 3 mm, bei 256 Nägeln ergibt sich ein Abstand von 6,7 mm zwischen zwei Nagelköpfen. Der Fädler hat an der Spitze eine Breite von 2,3 mm. Es bleiben also 2,2 mm Toleranz zu beiden Seiten des Fädlers, bis er an den nächstgelegenen Nagelkopf stößt. Das reicht für einen zuverlässigen Betrieb.

Der Schrittmotor, der den Rahmen dreht, wird mit Achtschritten betrieben, daraus ergeben sich 37,5 Schritte pro Einheit. Eine Einheit ist der Abstand zwischen zwei Nägeln.

Damit man die Nägel in den Rahmen stecken kann, bohrt man die benötigten Löcher direkt mit der String-Art-Maschine. Dazu spannt man den Rahmen ein und ein kleiner Servo bewegt einen Bohrer ins Holz und wieder hinaus, das Rad dreht sich eine Einheit weiter und das nächste Loch wird gebohrt. Sind alle Löcher fertig, wird der Rahmen entnommen und alle Nägel mit der Hand eingesteckt. Dieses Vorbohren stellt nicht nur die perfekte Ausrichtung der Nägel sicher (Abstand, Tiefe und Neigung), sondern schützt auch das Holz des Rahmens davor, beim Einschlagen eines Nagels zu splintern. Anschließend wird der Rahmen wieder eingesetzt. Die gebohrten Löcher sind nicht ganz im 90-Grad-Winkel zum Rahmen, sondern um 20 Grad schräg nach außen gebohrt. Das hilft dann beim Besspinsen des Rahmens. Mehr dazu später.

Auch wenn die Nägel augenscheinlich absolut gleichmäßig verteilt scheinen, sind sie es nicht. Der Antrieb durch gefräste Holzzahnräder hat immer leichte Abweichungen im Rundlauf. Das bedeutet, dass die Nägel mal etwas größere, mal kleinere Abstände zueinander haben, was sich zu mehreren Millimetern Abweichung aufbauen kann. Es sollte natürlich präzise gearbeitet werden, aber perfekt wird es nie.

Wenn man den Rahmen beim Wiedereinsetzen zum Spinnen genau gleich ausrichtet wie zum Bohren der Löcher, gleichen sich Ungenauigkeiten aber wieder aus, weil der Rah-

men genau so am Fädler vorbeifährt wie vorher am Bohrer.

Bei mehreren tausend Linien, die sich bei einem Rahmen mit 800 mm Durchmesser zwischen den 256 Nägeln spannen, stapeln sich teilweise sehr viele Wicklungen um einzelne Nägel. Deshalb sind die Löcher der Nägel auch etwas schräg gebohrt, um die Wicklungen aus dem Bewegungsbereich des Fädlers abzutransportieren. Die Fadenspannung zieht die Wicklung beim Weiterdrehen nach unten zum Holz. Zusätzlich helfen aus Schweißdraht gebogene Niederhalter zu beiden Seiten des Fädlers nach, falls der Faden hängenbleibt. Manche Nägel haben kleine Unebenheiten, die ein Herunterschieben verhindern können.

Die Fadenlänge pro Bild beträgt bei den hier genannten Umständen ca. 2,5–3 km. Die Nägel können zwar mit der Hand wieder herausgezogen werden, halten aber durch die sich aufbauende Fadenspannung sicher im Holz.

Reißt der Faden, öffnet sich ein Taster, der bei Betrieb durch die Fadenspannung geschlossen bleibt. Er ist mit der Door-Funktion des CNC-Controllers verbunden, die bei CNC-Maschinen gewöhnlich mit einem Kontakt an der Tür gekoppelt ist und dafür sorgt, dass die Maschine bei geöffneter Tür stehen bleibt.

Reißt der Faden, bleibt die Maschine stehen und man kann neu einfädeln, die losen Fadenenden provisorisch mit Klebeband fixieren und mit der grünen Resume-Taste fortfahren.

Die Fadenführung ist bis auf zwei Messingröhrchen komplett mit Schweißdrähten verschiedener Durchmesser aufgebaut. Man kann beim Bau so einfach ausprobieren, wo man die Führungen platzieren möchte, was z. B. bei der Ermittlung einer guten Hebeluntersetzung für den Taster hilft.

Ist das Bild fertig gesponnen, werden die Fadenenden mit Sekundenkleber fixiert und knapp abgeschnitten.

Der Antrieb und Steuerung

Den grundsätzlichen Antrieb habe ich vom Vorbild übernommen. Das Rad wird von einem Schrittmotor mit 1,8 Nm Haltemoment angetrieben. An der Motorwelle hängt ein Zahnrad mit 12 Zähnen. Es greift direkt ins Rad, in dessen Rückseite ein Zahnkranz mit 120 Zähnen eingräst. Das ergibt eine Untersetzung von 1:6. Das Zahnrad hat ein Modul von 2 mm. Die Zahnradgeometrie wurde mit dem Plug-in „Spur Gear“ für Fusion 360 erzeugt.

Bis die Verbindung zwischen Motorwelle und Zahnrad spielfrei und zuverlässig war, hat es etwas gedauert. Das letztlich genutzte Bindeglied ist ein kleiner, billig gegossener Wellenflansch, der sehr unpräzise ist und nur mit einer Rändelschraube über die Welle kratzt und Spiel erzeugt. Letztlich habe ich eine Nut in die Welle gefeilt, auf deren planer

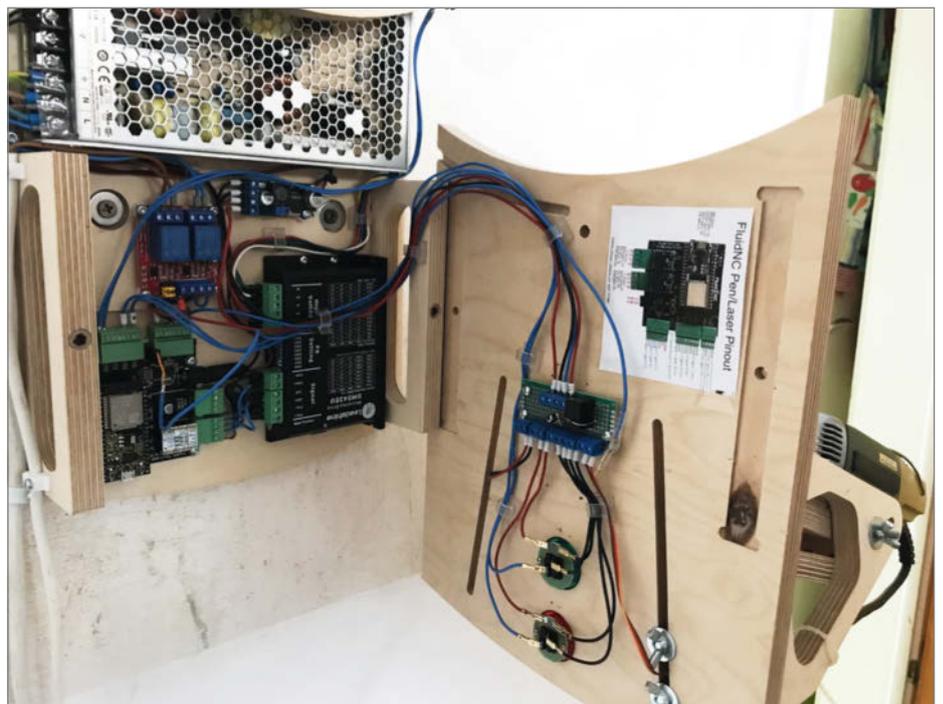


Aus Holz gefräste Zahnräder bewegen den Rahmen.

Fläche die Schraube besseren Halt findet als auf der runden Welle. Gewindekleber hilft zusätzlich. Beim nächsten Mal würde ich direkt einen Schrittmotor mit Welle im D-Profil wählen.

Das Rad etwas außerhalb der Drehachse anzutreiben, ist günstig für die Kraftübertragung, weil es weniger Stress für die Kompo-

ponenten des Antriebs bedeutet. Der Zahnradantrieb ist erstaunlich robust und kann die Kraft gut übertragen. Die Befürchtung, dass sich Multiplex abnutzen könnte und gehärtet werden müsste, hat sich nicht bestätigt. Wie sich die Druckteile des Vorbilds aus Kunststoff dagegen verhalten, kann ich leider nicht beurteilen.



In dem ganzen Holz arbeitet ordentlich Technik.

Der String-Art-Algorithmus

Wer sich für die Funktionsweise eines String-Art-Algorithmus interessiert, findet in einem Paper von Malcom Drougard eine gut verständliche Beschreibung. Der Link dazu befindet sich in der Kurzinfo.

Es gibt viele quelloffene Implementierungen von String-Art-Algorithmen. Ich habe mich für Knitter von Christian Siegel entschieden, das in Processing geschrieben ist. Es ist gut anpassbar, robust und schnell. Die Motivgenerierung dauert ca. 1–2 Sekunden, so kann man recht schnell verschiedene Einstellungen ausprobieren. Die Ausgabe des Sketches kann man unkompliziert erweitern, um z. B. das fertige Motiv als PNG-Grafik auszugeben oder die verwendeten Einstellungen als JSON zu speichern.

Bei der Vorbereitung des Motivs durch Bildbearbeitung hilft es, die Kontraste zu

erhöhen und speziell bei Porträts den Hintergrund abzudunkeln und Details zu entfernen. Es kann nicht jeder Bildbereich denselben Detailgrad aufweisen, deshalb ist meist ein wenig Art-Direktion nötig. Bei Gesichtern bewirkt aktive Retusche mit dem Pinselwerkzeug im Bereich von Augen und Mund oft, dass sich der typische Ausdruck der Person auf das String-Art-Bild überträgt.

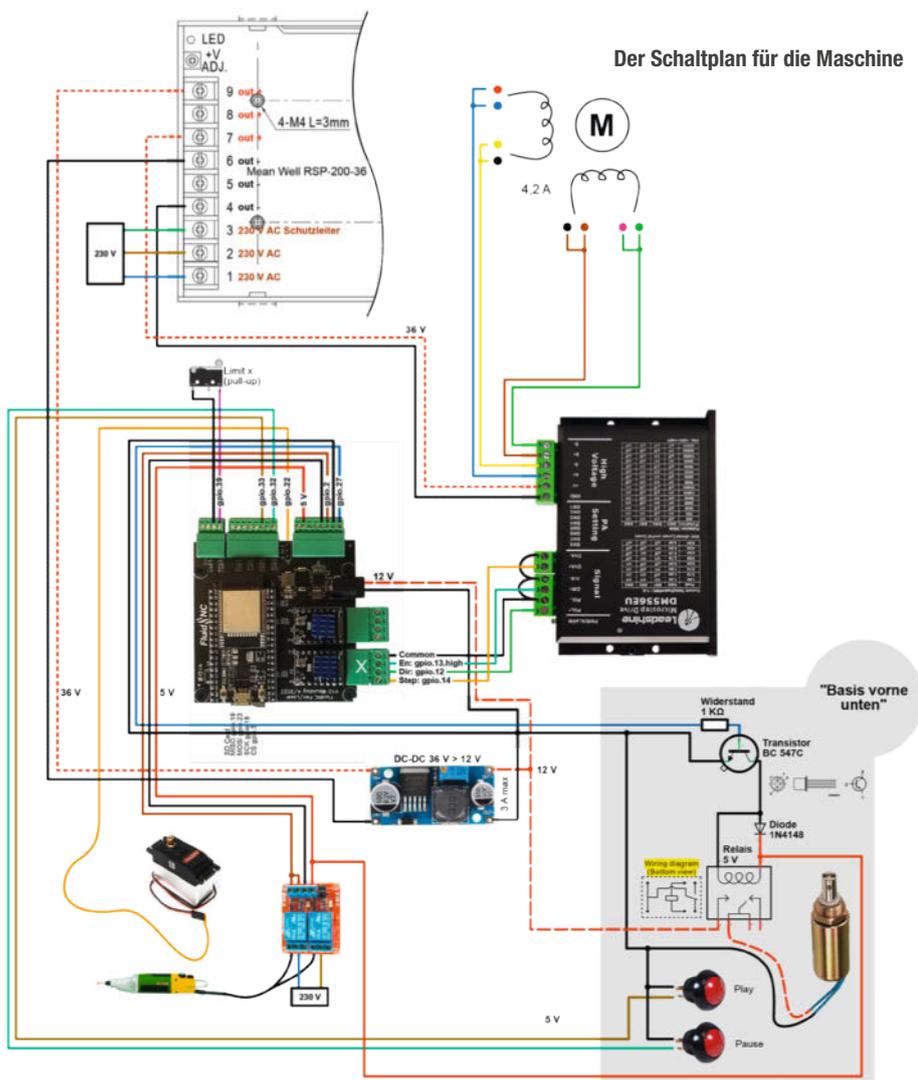
Neben der Anzahl der Fäden können noch weitere Parameter im Sketch eingestellt werden, die die Bildgenerierung beeinflussen. Durch Ausprobieren bekommt man ein Gefühl für gute Werte; sie hängen vor allem vom Motiv und der späteren Größe und Nagelanzahl des Bildes ab. Das Quellmaterial kann in erstaunlich geringer Auflösung vorliegen. Hier können, abhängig vom Motiv, auch um die 200×200 Pixel

ausreichen. Hat man die richtigen Einstellungen gefunden, kann man sich diese als neuen Standard im Sketch speichern und hat gute Referenzwerte für spätere Durchgänge.

Der Algorithmus selbst gibt zunächst eine lange Liste mit der jeweils nächsten Nagelnummer aus. Für die Bildschirmausgabe werden diese mit Linien verbunden. Um reale Fäden zu spannen, muss aus dieser Nummernliste G-Code generiert werden.

Für den G-Code-Export habe ich das Python-Skript von Bart Dring in Processing übersetzt. Es schaut für jede neue Linie zunächst nach dem kürzesten Weg von der aktuellen Position zum nächsten Nagel (Rechts- oder Linksdrehung) und gibt den entsprechenden G-Code aus.

Der Schaltplan für die Maschine



Elektronik

Das Herz der Steuerung ist der Pen/Plotter-Controller (SPI-Version) von Bart Dring. Raddrehung, Bohren und Fädlerbewegung werden durch x-, y- und z-Achse im G-Code abgebildet.

Die x-Achse ist die Drehbewegung des Rahmens. Der Controller ist eigentlich für den Betrieb von bis zu zwei TMC2130-Schrittmotortreibern im StepStick-Format vorgesehen. Da sie nur für 2 A ausgelegt sind und nicht für den Betrieb des NEMA-23-Schrittmotors (4 A) ausreichen, habe ich einen der zwei Steckplätze mit einem kleinen Dummy gebrückt, sodass die Step/Direction-Signale an einen externen Treiber weitergereicht werden. Ich habe mich für einen DM532E von Leadshine entschieden. Dieser bietet mit 50 V bei 4,2 A genug Power.

Die G-Code-Werte für die y-Achse steuern den Fädler. Der Elektromagnet, der den Fädler bewegt, wird über ein Reed-Relais geschaltet. Dieses wird mittels eines Transistors über den Controller gesteuert.

Ein 5-V-Output des Controllers schaltet daher einen Transistor, der wiederum ein kleines Reed-Relais schaltet. Eine Freilaufdiode schützt den Schaltkreis vor induktiver Last, die der Elektromagnet erzeugen kann. Wichtig ist zu beachten, dass die Kraft eines Elektromagneten nicht über den gesamten Hub konstant ist, sondern gerade am Anfang der Bewegung deutlich geringer ist als gegen Ende. Dafür liegen bei Elektromagneten Kraft-Weg-Diagramme bei. Die helfen einem bei der Nut-

zung. Da der Elektromagnet nicht nur den Fädler bewegen muss, sondern auch den Faden nachzieht, muss er recht kräftig sein und wird auch heiß. Man muss also auf eine gute Luftzirkulation achten.

Die z-Achse bildet den Bohrer ab. Den 5-V-Servo zum Fahren des Bohrers kann man direkt an den Controller anschließen. Da Servos mit PWM angesteuert werden, müssen hier nur die minimale und maximale Pulsweite in der Konfiguration angegeben werden. Der Bohrer selber ist ein Feinbohrschleifer von Proxxon, der mit ca. 8.500 U/Min läuft. Da er mit Netzspannung betrieben wird, schaltet ihn eine fertig zu kaufende Relaischaltung.

FluidNC

Der Controller wird mit der Firmware FluidNC betrieben, die auf einem ESP32 läuft. FluidNC ist Open Source und hat eine aktive Community. Die Firmware kann sehr flexibel mit unterschiedlichster Hardware umgehen; die Konfiguration wird einfach als YAML-Datei auf den Controller hochgeladen.

In der Ansteuerung mit G-Code ist sie vollständig kompatibel zu GRBL, bietet aber deutlich mehr Möglichkeiten. Für dieses Projekt

sind die vielfältigen Ansteuerungsmöglichkeiten und möglichen Outputs (Schrittmotor, Servo und Elektromagnet) wichtig. Eine Verbindung via USB ist eigentlich nur für das erste Flashen des ESP32 nötig, danach kann man komplett auf eine Kabelverbindung verzichten und via WLAN das Webinterface nutzen. Darüber kann man dann homen und G-Code-Dateien direkt auf den Controller hochladen und ausführen. Auch „Over-the-Air“-Updates (OTA) sind so möglich.

Wer das Webinterface nicht mag, kann FluidNC auch via Telnet oder WebSockets steuern und sich so ein eigenes, spezialisiertes Interface bauen.

Es gibt einen Discord-Server, auf dem man Hilfe findet. Die allermeisten Fragen beantwortet aber bereits das gut gepflegte und umfangreiche Wiki. Links dazu findet man in der Kurzinfor.

Recap und Fazit

Das Projekt sah von außen einfacher aus, als es sich praktisch darstellte. Speziell das Faden-Handling und die Präzision beim Bohren und Fädeln machten viel Ausprobieren und Testen erforderlich.

Eine liegende Bauweise wie im Vorbild vereinfacht die Balance des Rades, sodass eine Achse mit 8 mm Durchmesser ausreicht. Meiner Bauweise hätten eine steifere Achse und eine stärkere Verbindung zum Drehteller gut getan. Zudem ist mir ein Denkfehler erst so spät aufgefallen, dass ich das Rad nochmal neu hätte anfertigen müssen: Die Achse ist auf beiden Seiten kugelgelagert, das ist nicht direkt nötig.

Die zweite größere Sache, die in der nächsten Version verändert werden wird, ist der Antrieb. Ich bin zwar froh, auf möglichst viele fertige Komponenten verzichten zu haben, und die Zahnräder aus dem ohnehin verwendeten Holzwerkstoff mit auszufräsen war sehr elegant und hat Spaß gemacht. Zahnräder sind leichter zu erzeugen, als gedacht. Aber gerade die Präzision hat mich einiges an Nerven gekostet.

Dieses Projekt hat in der Umsetzung sehr verschiedene Herausforderungen und bietet viele Möglichkeiten zur Verbesserung. Die nächste Version ist schon in Planung, mit den oben beschriebenen Veränderungen und vielen kleinen Verbesserungen. Ein großes Ziel ist die weitere Automatisierung mit Bohren, Nageln und Spinnen in einem Arbeitsgang. —das



DDUX

Konferenz für Digital Design und UX Professionals
Dortmund • 25./26. Juni 2025

Digital Design. Verbindet.

Ganzheitlich gut gestaltete digitale Produkte und Lösungen sind der Schlüssel für erfolgreiches Business. Das erfordert **spezifisches Know-how** und **neue Konzepte** der Zusammenarbeit von **Produktentwicklern, Technologieexpertinnen und UX Designern.**

Jetzt
Vortrag
einreichen!
Bis 7.12.2024

Workshops am 27. Juni 2025

www.dd-ux.de

Veranstalter



MAIBORNWOLFF



Kooperationspartner



VERBAND
DEUTSCHER
INDUSTRIE
DESIGNER



Mit freundlicher Unterstützung der
Wirtschaftsförderung der Stadt Dortmund

Copyright by Makez Magazine

Der Kabelbinder- Aufmacher

Kabelbinder können mit einem kleinen, einfach herzustellenden
Werkzeug geöffnet und wiederverwendet werden.

von Kenneth Weiss



Möchtest du kostenlose große Kabelbinder, die du immer wieder verwenden kannst? Dann wirst du dieses kleine, praktische Werkzeug lieben, das ich entworfen habe. Die Idee dazu kam mir nach einem Wahlkampf hier in Dänemark, wo ich lebe.

In den Wochen vor der Wahl wurden jeder einzelne Laternenpfahl, jeder Zaun, jedes Geländer und andere Gegenstände, an denen man irgendwas mit Kabelbindern befestigen kann, vollständig mit Plakaten der verschiedenen politischen Parteien zugehängt. Oft wurden dafür kräftige Kabelbinder benutzt.

Es ist illegal, während der Wahlperiode irgendwelche Plakate zu entfernen oder zu beschädigen, so verlockend es auch sein mag. Aber nach der Wahl müssen alle Plakate innerhalb weniger Tage wieder abgenommen werden. Politiker, die dies nicht rechtzeitig tun, müssen mit Bußgeldern rechnen. Daher sind die Freiwilligen, die die Plakate abnehmen, nur daran interessiert, diese Arbeit so schnell wie möglich zu erledigen. Das hinterlässt oft jede Menge Kabelbinder, die noch an Laternenpfählen und anderen Orten befestigt sind und dort Wochen bleiben, bis die Gemeinde Zeit hat, sie zu entfernen.

Kabelbinder im Überfluss

Warum also nicht helfen und im Gegenzug eine Ladung kostenloser Kabelbinder erhalten, indem man ein Werkzeug herstellt, das die Kabelbinder entfernen kann, ohne sie zu beschädigen? Dadurch verwandeln wir im Grunde Einwegplastik in etwas Wiederverwendbares.

Dafür habe ich den UnZipper entworfen (Bild 1) – ein kleines und praktisches Werkzeug, das man immer bei sich haben kann, zum Beispiel am Schlüsselbund.

Der Trick besteht darin, eine dünne Metallklinge zu nehmen und einen kleinen Haken daran zu formen (Bild 2), den man verwenden kann, um die kleine Verriegelungslasche, die alle Kabelbinder haben, nach oben zu drücken. Es ist dieser Verriegelungsmechanismus, der den Kabelbindern ihr charakteristisches Klickgeräusch gibt, wenn man sie enger zieht.

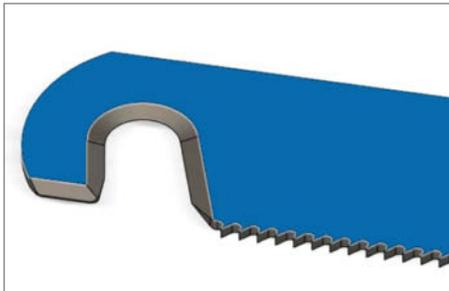


Bild 2: Der Haken sollte etwa 4 mm breit sein und die Öffnung ungefähr 5 mm betragen. So können die meisten Größen von Kabelbindern geöffnet werden.

Kurzinfo

- » Erstellung eines Werkzeugs zum Öffnen von Kabelbindern
- » Einmal benutzte Kabelbinder wiederverwenden
- » Die Entstehungsgeschichte von Kabelbindern

Checkliste



Zeitaufwand:
30 Minuten



Kosten:
3 Euro

Material

» Metallsägeblatt

Werkzeug

» Dremel oder Proxxon

Alles zum Artikel im Web unter make-magazin.de/x6rd

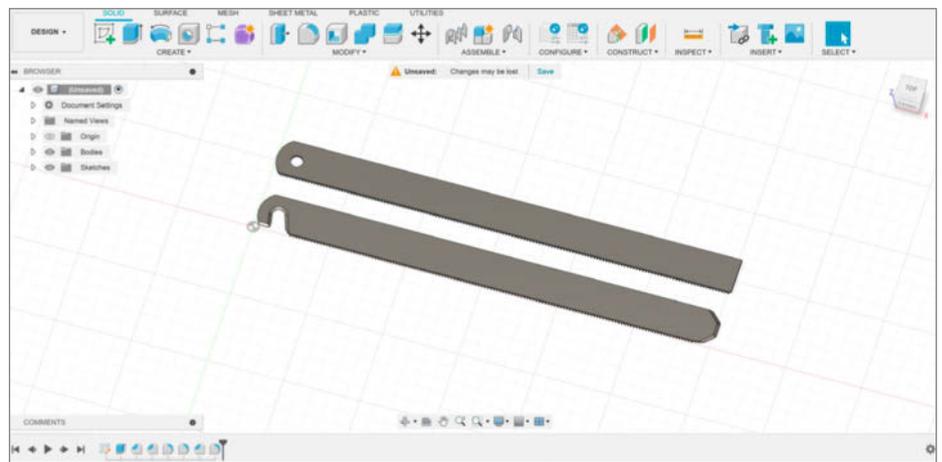


Bild 1: Das Design ist denkbar einfach.



Bild 3: Der Haken wird von innen eingeführt und mit sanftem Druck verschiebt man dann den Verriegelungsmechanismus rückwärts entlang des Kabelbinders.

Um die Verriegelungslasche zu erreichen und den Kabelbinder schonend zu öffnen, ohne ihn zu beschädigen, wird der UnZipper an der „Innenseite“ des Kabelbinders eingeführt, also genau dort, wo früher die Spitze des Plastikstreifens eingeführt wurde, um den Kabelbinder zusammenzuziehen (Bild 3).

Den UnZipper bauen

Um dieses Werkzeug herzustellen, benötigst du nur einen Proxxon oder Dremel mit einer Schneidescheibe und einem Schleifbit. Wenn du dich erfahren genug fühlst, kannst du auch einen Winkelschleifer verwenden.

Die ersten UnZipper, die ich hergestellt habe, waren aus 0,5×12,7 mm Fühlerlehrenband. Davon habe ich ein etwa 150 mm langes Stück abgeschnitten und an jedem Ende einen Haken eingeschliften (einen für kleinere Kabelbinder und einen für größere) und die Kanten abrundet. Das funktionierte großartig, aber da nicht jeder Zugang zu diesem Material hat, suchte ich nach einer etwas einfacheren Lösung und fand heraus, dass Sägeblätter von Metallsägen dieselben Abmessungen und die nötige Steifigkeit haben. Jedes alte Metallsägeblatt erledigt den Job. Schneide es einfach in zwei Hälften und du hast, was du brauchst, um zwei UnZipper herzustellen.

Kabelbinder gibt es in vielen verschiedenen Marken und Größen, und je nach Größe des Kabelbinders, den du aufmachen möchtest, kannst du den Haken am UnZipper größer oder kleiner machen.

Da das Werkzeug aus einem Metallsägeblatt gemacht ist, kann es auch als kleine Taschenhandsäge verwendet werden. Mit ein wenig Experimentieren mit verschiedenen Formen und den Bereichen, die du schleifst/schärfst, kannst du den Haken auch zusätzlich als Schnurschneider benutzen. Das andere Ende lässt sich auch dafür verwenden, um etwa kleine Geräte aufzuhebeln (Bild 4).

Bei der Herstellung musst du nicht so genau arbeiten. Wichtig ist aber, dass der Haken schmal genug ist, um durch die Öffnung zu passen und die Verriegelungslasche zu erreichen. Ich habe die Erfahrung gemacht, dass ein Haken mit einer Breite von etwa 4 mm und einer Öffnung von etwa 5 mm gut funktioniert, um die meisten mittelgroßen bis großen Kabelbindern zu öffnen. Vergiss nicht, die Spitze des Hakens abzuflachen, damit er leichter unter die Verriegelungslasche gleitet.

Viel Erfolg beim Helfen, die Welt von ungenutzten Kabelbindern zu befreien. Hoffentlich wirst du nie wieder in deinem Leben einen Kabelbinder kaufen müssen! —mch

Von Kabel binden zu Kabelbindern

Der Kabelbinder aus Kunststoff, wie wir ihn heute kennen, wurde in den 1950er Jahren erfunden, und wir können größtenteils dem in Schottland geborenen Ingenieur Maurus C. Logan dafür danken. Er arbeitete damals für die Elektroindustrie-Firma Thomas & Betts in den USA. Während er ein Flugzeugfertigungswerk von Boeing besuchte, hatte er beobachtet, wie die Arbeiter hunderte Meter von Kabelbündeln mit gewachster Nylonschnur abbinden mussten. Es handelte sich um eine zeitaufwändige und anstrengende Arbeit, bei der die Kabelbündel mit verschiedenen komplexen Knotentechniken, vergleichbar mit Seemannsknoten, straff zusammengezogen wurden. Dabei schnitten die glatten Schnüre den Arbeitern oft in die Hände und verursachten dadurch Geschwüre. Auf diese Weise verwundete und angeschwollene Hände nannte man „Hamburger Hände“.

So machte sich Maurus C. Logan daran, eine bessere Alternative zu entwickeln. Nach zwei Jahren Entwicklungsarbeit reichte er 1958 sein Patent ein – einen langlebigen Kabelbinder aus Nylon mit einer kleinen Metallzunge als Schließmechanismus. Als der „Ty-Rap“ auf den Markt kam, wurde er sehr schnell populär und bekam unzählige Nachfolger. Das ursprüngliche

Modell ist auch heute noch erhältlich und wird von der Firma ABB als Ty-Rap-Stahlnasenkabelbinder vermarktet.

Als Erfinder des Kabelbinders wird manchmal auch der US-Amerikaner George M. Rapata genannt. Er beantragte bereits 1954 ein Patent auf einen Kabelbinder aus Polyethylen. Seine Konstruktion war jedoch weniger erfolgreich als die von Logan, unter anderem, weil das Material nicht haltbar genug war und zur Befestigung zusätzlich ein Spezialwerkzeug erforderlich war.

Heute werden Schätzungen zufolge jährlich 100 Milliarden Kabelbinder produziert. Trotzdem ist die vorherige Technik, Kabelbündel mit Schnüren zu binden, weit davon entfernt, ausgestorben zu sein. Im Internet findet sich eine Vielzahl von Anleitungen dazu, wie man diverse Knotentechniken korrekt verwendet, weil die Technik immer noch Anwendung in Bereichen mit besonders hohen Sicherheitsstandards findet, beispielsweise in der Weltraumtechnik und auch heute noch in der Flugzeugelektronik, wo alles damals begann.

Wenn ihr mehr darüber lesen möchtet, findet ihr mehr Informationen über den Link in der Kurzinfo.

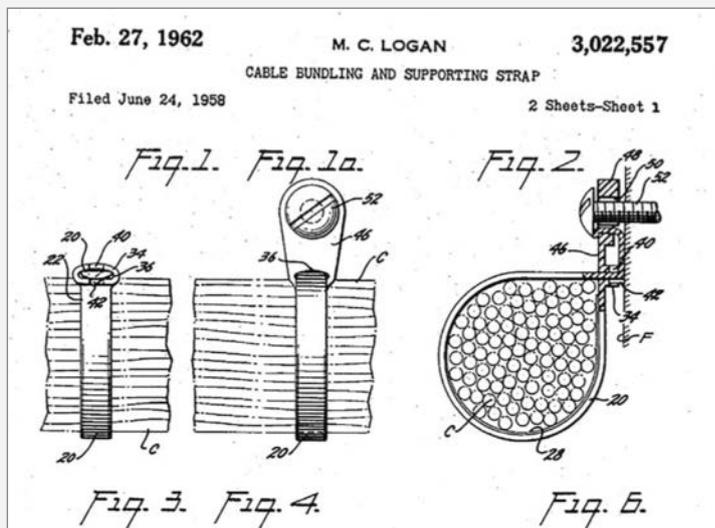


Bild 4: Nicht nur Kabelbinder, sondern auch Kunststoffgehäuse aller Art lassen sich mit dem UnZipper öffnen



Maker Faire 2025

Das Maker-Faire-Hannover-Jubiläumsjahr 2024 geht zu Ende. Aber für 2025 stehen schon die nächsten Messen in den Startlöchern!

von Daniel Schwabe



Für Deutschland stehen schon drei Maker-Highlights fest: Eröffnet wird das Maker-Jahr von der Maker Faire **Heilbronn am 8. Februar 2025** im Science Center experimenta.

Weiter geht es mit der Maker Faire Ruhr in **Dortmund vom 29. bis 30. März 2025** in der DASA Arbeitswelt Ausstellung. Bei dieser Messe ist auch dieses Jahr wieder die Make-Redaktion inklusive heise-Shop vertreten. Wer

Heftprojekte hautnah erleben will oder coole Maker-Gadgets braucht, ist hier richtig.

Und natürlich ist auch das Jahres-Highlight mit der Maker Faire **Hannover vom 23. bis 24. August 2025** wieder fest gesetzt. Natürlich wird die Redaktion wieder alle Register ziehen und die beeindruckendsten Projekte aus den Heften dabei haben. Auch die hauseigene Makey:Lab-Hardware wird ausgestellt. Gleiches gilt für die Oxocard Connect in der

Make-Edition, die es seit diesem Jahr im heise shop gibt. Für die Messe Hannover ist auch schon der Call for Makers offen.

Die Redaktion freut sich schon auf ein aufregendes Jahr 2025 mit vielen inspirierenden Ideen und tollen Makern auf den verschiedenen Messen. —das

WILLKOMMEN IM NEUEN IOT-ÖKOSYSTEM

Mit LoRaWAN und C-Programmierung über lange Distanzen messen und steuern



Heft + LoRaWAN-Set

Make Special LoRaWAN inkl. Experimentierset für 64,90 €

JETZT BESTELLEN!



shop.heise.de/make-lorawan24

Fernsehen mit dem Pi streamen

Lineares Fernsehen findet mittlerweile auch im Internet statt. Öffentlich-rechtliche Sender kann man kostenlos über die Mediathek streamen. Bei den Privatsendern braucht man dann aber wieder ein kostenpflichtiges Abo. Hat man noch einen traditionellen Fernsehanschluss, kann man mit einem Raspi und der Software Tvheadend lineares Fernsehen kostenlos im eigenen Netzwerk streamen.

von Daniel Schwabe



Trotz der geradezu überwältigenden Angebote von Serien und Filmen verschiedener Streamingsservices gibt es immer mal wieder den einen Fernsehbeitrag im Privatfernsehen, den man nirgendwo sonst streamen kann. Der Receiver ist allerdings schon lange abgebaut oder der Anschluss für die Satellitenschüssel liegt leider nicht im richtigen Zimmer.

Mit einem Raspberry Pi 3 oder höher und einem USB-Empfänger kann man sich das TV-Programm ins Netzwerk streamen. Dafür verwendet man die Software Tvheadend, die einem über eine grafische Web-Oberfläche die Möglichkeit bietet, TV-Empfänger zu verwalten, Sender zu suchen und freizugeben.

Für die folgende Anleitung wird ein Raspberry Pi mit einer aktuellen Raspberry-OS-Version vorausgesetzt. Wie man die aufsetzt, ist im Artikel „Verbindungen zum Raspberry Pi Server“ aus Make 5/23 S. 72 nachzulesen. Die verwendete Software wird mithilfe von Docker und dem Docker-Frontend Portainer installiert. Die Anleitung dafür findet man im Video in der Kurzinfor.

Hardware

Neben dem Pi braucht man auch einen für Linux und den verwendeten Empfangsstandard kompatiblen Empfänger. In Deutschland stehen Satellitenfernsehen (DVB-S), Kabelfernsehen (DVB-C) und terrestrisches Fernsehen (DVB-T) zur Verfügung.

Mittlerweile wurde DVB-T durch DVB-T2 und DVB-S durch DVB-S2 abgelöst. Deshalb sind viele alte Adapter heute nur noch Elektroschrott. Wenn man einen modernen Adapter hat, kann man über eine Tabelle (in der Kurzinfor verlinkt) die Kompatibilität mit Linux checken.

Hier im Artikel wird ein über USB angeschlossenes TechnoTrend TT-connect S2-3650 CI für DVB-S2-Empfang und das extra für den Raspberry Pi entwickelte Raspberry Pi TV HAT für DVB-T2 verwendet.

Software installieren

Bevor man die Software installiert, muss man die Hardware zum Fernsehempfang an den Pi anschließen. Dann verbindet man ein LAN-Kabel (oder nutzt die Möglichkeit, beim Flashen der Software die Daten für eine WLAN-Verbindung mit einzugeben), die Stromversorgung mit dem Raspberry Pi und wartet einen Moment, bis dieser hochgefahren ist.

Kurzinfor

- » Fernsehen im Netzwerk streamen
- » Fernsehprogramm aufnehmen
- » Fernsehen über Kodi

Checkliste



Zeitaufwand:
1 Stunde



Kosten:
90 Euro

Material

- » Raspberry Pi Zero oder volle Größe
- » Raspberry Pi TV HAT
- » DVB-T-Antenne

Alles zum Artikel im Web unter make-magazin.de/xgyf



Mehr zum Thema

- » Daniel Schwabe, Reverse-Proxy auf dem Raspberry Pi, Make 2/24, S. 116
- » Daniel Schwabe, Private Maker-Tools auf dem Pi, Make 2/24, S. 110
- » Video: Docker auf dem Raspberry Pi installieren (Einsteiger-Tutorial)



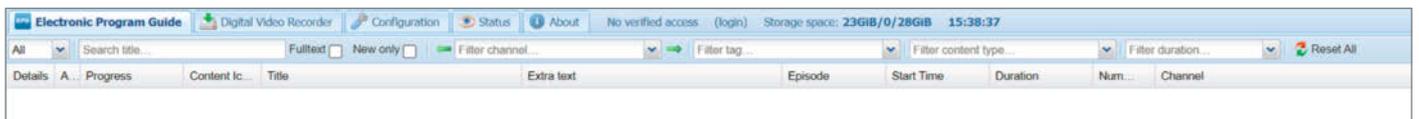
Jetzt kann man sich mit den Login-Daten, die man beim Aufspielen des Betriebssystems festgelegt hat, per SSH auf dem Pi einloggen. Dafür öffnet man auf einem beliebigen Windows-Computer im Netzwerk die Anwendung „Terminal“ und gibt dort den Befehl `ssh <Pi-Benutzername>@<IP-Adresse-des-Pi>` ein. Nachdem man sich mit dem Passwort

authentifiziert hat, kann man mit den Befehlen `sudo mkdir /TV`, `sudo mkdir /TV/Daten` und `sudo mkdir /TV/Aufnahmen` drei Ordner anlegen, die für die Installation von Tvheadend benötigt werden. Jetzt beginnt man mit der Installation über Docker.

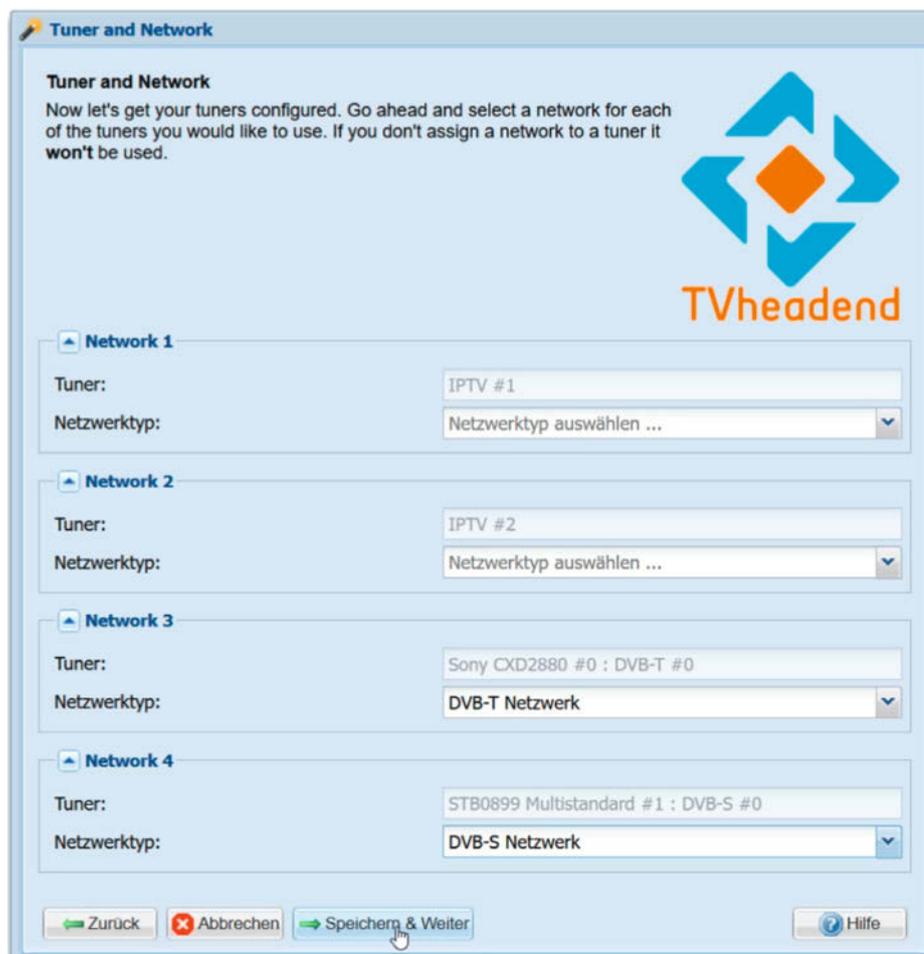
Über `<IP-Adresse-des-Pi>:9433` im Browser kann man nun auf die Weboberfläche von

Tvheadend – Docker-Compose

```
services:
  tvheadend:
    image: lscr.io/linuxserver/tvheadend:latest
    container_name: tvheadend
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Etc/DE
    volumes:
      - /TV/Daten:/config
      - /TV/Aufnahmen:/recordings
    ports:
      - 9981:9981
      - 9982:9982
    devices:
      - /dev/dvb:/dev/dvb
    restart: unless-stopped
```



So sieht die Bedienoberfläche im Browser aus.



Jedem Empfänger muss man ein Netzwerk zuweisen.

Portainer zugreifen. Dort dann auf der linken Seite die Option „Stacks“ anklicken. Danach oben rechts den Button „+ Add Stack“ klicken und den Inhalt aus dem Kasten „Tvheadend – Docker-Compose“ in das große Feld in der Mitte einfügen. Dann noch im Feld „Name“ einen Namen eingeben und zu guter Letzt unten auf „Deploy Stack“ klicken.

Wenn die Installation erfolgreich war, findet man unter <IP-Adresse-des-Pi>:9981 die Weboberfläche von Tvheadend.



Für DVB-T richtet Tvheadend die Muxe selbst ein.

Empfänger einrichten

Tvheadend verfügt über einen Einrichtungsassistenten, der Schritt für Schritt durch die Einrichtung der Software – egal für welchen Empfänger – führt. Der Assistent erlaubt auch, mehrere angeschlossene Empfänger parallel einzurichten. Deshalb werden im Folgenden die DVB-T2-Karte und die DVB-S2-Karte gleichzeitig behandelt. Man muss natürlich nicht beide haben, sondern kann auch nur eine von beiden einrichten.

Um diesen Assistenten zu starten, klickt man nacheinander auf „Configuration/General/Base/Start wizard“. Im sich öffnenden Fenster stellt man als erstes die Sprache ein. Wechselt man vom voreingestellten Englisch zu einer anderen Sprache und klickt dann auf den grünen „Save & Next“-Pfeil, lädt die Website einmal neu. Ab jetzt werden in diesem Artikel alle Teile der Bedienoberfläche mit ihren deutschen Bezeichnungen angeführt.

Im nächsten Assistenzenster legt man Zugangsdaten zu Tvheadend fest. Der Administratorzugang erlaubt den Zugriff auf die Weboberfläche, die gerade bedient wird. Sobald dort Daten eingetragen wurden, muss man bei jedem neuen Aufruf der Seite diese

Zugangsdaten eingeben. Das Benutzerlogin ist dafür gedacht, später vom Server das Fernsehprogramm streamen zu können.

Für diesen Artikel verwende ich den Namen „zuschauer“ für das Benutzerlogin und den Namen „admin“ für den Administratorzugang. Das Feld „Netzwerkzugang“ lässt man frei und klickt unten wieder auf den grünen Pfeil. Im Deutschen steht dort „Speichern & Weiter“.

Das nächste Fenster ist für die Einrichtung der Empfänger (Tuner). In diesem Fall werden vier „Network“-Einträge angezeigt. Zwei IPTV-Tuner, von denen keine im Netzwerk gefunden werden, und darunter die beiden an den Raspberry Pi angeschlossenen Empfänger.

Der Sony-Tuner ist der DVB-T2 Empfänger des TV HAT und der STB0899 ist der beispielhaft verwendete USB-DVB-S2-Empfänger. Die Tuner-Felder kann man selber nicht ändern. Nur das Feld Netzwerktyp darunter. Dort muss passend zum Empfänger DVB-T, DVB-S oder DVB-C eingetragen werden. Auch, wenn es DVB-T2 und DVB-S2 Empfänger sind, gibt es die ausgewählten Netzwerke nur ohne die 2 im Namen. Bestätigen erfolgt mit „Speichern & Weiter“.

Jetzt muss man sogenannte Muxe für die TV-Karten auswählen. Ein Mux ist eine (virtuelle) Senderquelle, auf die mehrere Signale nacheinander geschaltet (gemultiplext) wurden. Ein Mux kann also mehrere Sender abdecken.

Für die DVB-T2-Karte ist hier aus dem Dropdown-Menü „Germany: de-All“ wählbar. Das Feld für den DVB-S2-Empfänger bleibt leer. Hat man eine DVB-C-Karte, sollte hier der richtige Kabelanschluss eingetragen werden. Die Bezeichnungen dafür beginnen mit „Germany: de-“ (z.B. „Germany: de-Muenchen“).

Mit „Speichern & Weiter“ startet auf der DVB-T2-Karte die Sendersuche.

Steht der Fortschrittsbalken auf 100%, geht es mit „Speichern & Weiter“ zum letzten Einrichtungsfenster. In dem kann man festlegen, ob die gefundenen Sender von einem Client gestreamt werden können. Deshalb klickt man hier auf „Alle Services zuordnen“ und danach ein letztes mal auf „Speichern & Weiter“.

Die Webseite lädt neu und man muss sich als Admin einloggen.

Ist nur eine DVB-T2- oder DVB-C-Karte vorhanden, ist man jetzt fertig. Und kann direkt zu dem Teil des Artikels springen, in dem es um das Streamen der Sender geht. DVB-S2 muss noch weiter von Hand konfiguriert werden.

Extraschritte für DVB-S2

Für Satellitenempfang funktioniert der Sendersuchlauf leider nicht so automatisch.

Als Erstes muss für DVB-S2 der richtige Satellit ausgewählt werden. Dafür wählt man unter „Konfiguration/DVB-Inputs/Netzwerke“

den Eintrag „DVB-S Netzwerk“ aus und klickt auf Bearbeiten. Im sich öffnenden Fenster wählt man im Bereich „Orbitale Position“ den Eintrag „19.2E : Astra 1KR/1L/1M/1N“ aus.

Als Nächstes muss manuell ein Mux angelegt werden. Das geht unter „Konfiguration/DVB-Inputs/Muxes/Hinzufügen“. Nach dem Klicken auf „Hinzufügen“ weist man den Mux einem Netzwerk zu. In diesem Fall DVB-S. Im darauffolgenden Fenster alle Daten wie im Bild eintragen. Das Ganze mit „Anlegen“ abschließen.

Jetzt sollten automatisch neue Muxe generiert und Sender gefunden werden. Sollte das nicht geschehen und neben dem Mux in der Liste nur „FEHLER“ auftauchen, muss man das DVB-S-Netzwerk noch einmal dem Adapter zuordnen. Dafür unter „Konfiguration/DVB-Inputs/TV-Adapter“ den richtigen Empfänger auswählen, das Netzwerk einstellen und auf „Speichern“ klicken. Dann den Mux noch einmal anlegen.

In der Netzwerk-Liste sieht man, wie weit der Suchvorgang ist.

Ist der Sendersuchlauf beendet, befindet sich idealerweise im Tab „Konfiguration/DVB-Inputs/Services“ eine lange Liste mit Fern-

Die Einstellungen für den initialen DVB-S-Mux.

sehsendern. Diese müssen jetzt zugeordnet werden. Dafür in der Bedienleiste auf das Drop-down-Menü „Gewählte Services zuord-

nen“ klicken und dann auf „Alle Services zuordnen“. Das Ganze mit „Services zuordnen“ bestätigen.



KANBAN DAYS

Den Wertstrom teamübergreifend optimieren

26./27. Februar • Mannheim

Kanban ist eine Methode zum **Management von Arbeitsabläufen**, die hilft, Engpässe zu identifizieren und den **Workflow** zu verbessern – auf allen Ebenen der Organisation.

Erfahrene Experten und **innovative Vordenkerinnen** bieten **praxisnahe Einblicke**, wie sie mit Kanban eine nahtlose und **effiziente Wertschöpfungskette** gestalten.

Die Konferenz richtet sich an **Führungskräfte** und **Verantwortliche** für Services, Produkte und Projekte sowie **Kanban** und **Agile Coaches** und **Scrum Master**, die sich für das Thema interessieren oder es vertiefen wollen. Wer teilnimmt, sollte ein Grundverständnis von Kanban mitbringen.

Jetzt
Frühbucher-
ticket
sichern!

Workshops am 28. Februar 2025

kanban-days.de

Veranstalter



dpunkt.verlag

Kooperationspartner
© Copyright by Maker Media GmbH.



colenet
messbar schneller





Hat alles geklappt, werden DVB-S-Sender gesucht.



Mit einer geänderten Einstellung funktioniert das Playlist-Streamen.

Streamen

Öffnet man über „Konfiguration/DVB-Inputs/Services“ die Senderliste, kann man bei jedem aufgeführten Sender links im Fenster auf ein kleines, blaues Play-Zeichen klicken.

Das startet den Download einer m3u8-Datei mit einem langen Namen, der aus Buchstaben und Zahlen besteht. Diese Datei kann mit einem Videoplayer wie z. B. VLC-Media-player geöffnet werden.

Um nicht für alle Kanäle eine einzelne Datei zu haben, kann man auch eine Playlist mit allen zugeordneten Kanälen herunterladen.

Dafür muss man die Zugangsdaten für den Zuschauer aus der Installation anpassen. Das geschieht unter „Konfiguration/Benutzer/Passwörter“. Dort den passenden Eintrag (in diesem Beispiel heißt der Eintrag „zuschauer“) auswählen und auf Bearbeiten klicken. Dort dann den Eintrag „Persistent authentication“ aktivieren und speichern.

Jetzt kann man mit dem Link <Zuschauername>:<Zuschauerpasswort>@<Rasperry-PiIp>:9981/playlist/auth/channels eine komplette Playlist in einer Datei herunterladen.

Über diese Liste kann man jetzt die Channel auswählen.

Wer ein eleganteres Verfahren nutzen möchte als das Hantieren mit einzelnen Dateien oder Playlists in VLC, kann Tvheadend über Kodi streamen.

Dafür unter „Addons/Herunterladen/PVR-Clients“ den „Tvheadend HTSP Client“ auswählen und installieren. Jetzt befindet sich unter „Addons/Benutzer-Addons“ der Eintrag „PVR-Clients“. Dort auch wieder den „Tvheadend HTSP Client“ auswählen und über die Schaltfläche „Konfigurieren“ die Einstellungen aufrufen.

Dort den Eintrag „127.0.0.1“ [aktiviert] bearbeiten“ aufrufen.

Im neuen Fenster dann den Namen in „Rasperry Pi“ ändern und den Eintrag „Hostname oder IP-Adresse des Tvheadend-Servers“ auf die IP des Rasperry Pi setzen. Dann weiter unten die Felder „Benutzername“ und „Passwort“ mit den Daten des Zuschauer-Accounts befüllen. Zum Schluss „OK“ klicken.

Jetzt Kodi neu starten. Im Kodi-Hauptmenü gibt es unter dem Eintrag TV nun eine Liste mit allen zugeordneten Sendern des Tvheadend-Servers.

Für FireTV-Nutzer eignet sich die App „Dream Player TV for Tvheadend“ als Client. Diese App ist direkt im Amazon-Store erhältlich und führt Schritt für Schritt durch die Einrichtung. Der Link zu dem Programm steht in der Kurzinfor.

Die gleiche App gibt es auch im Google Playstore für Android-Geräte. Für Apple-Handys heißt die App TvhttpClient.

M3U8

M3U-Dateien sind einfache Textdateien. Sie werden genutzt, um auf Mediendateien zu verweisen. Beispielsweise bei einer Playlist. Die von Tvheadend erzeugten Dateien sehen wie folgt aus:

```
#EXTM3U
#EXTINF:-1,zdf_neo HD / ZDFvision
http://tv.local:9981/stream/service/3a15796c9c82246f54d6e89f668c0d47?ticket=a47a-840a59f7d7168a34441a5c05abb2851aa479
```

Die erste Zeile legt mit #EXTM3U fest, dass es sich tatsächlich um eine M3U-Datei handelt. Die zweite Zeile #EXTINF enthält Informationen über den verlinkten Medieninhalt. Länge, Name, Album usw. In diesem Fall steht -1 dafür, dass die Länge nicht bekannt ist, und „zdf_neo HD / ZDFvision“ ist der Name.

Darunter folgt dann der Link zum eigentlichen Medieninhalt. In diesem Fall ist es ein Link zu einem Server. Allerdings kann es auch ein lokaler Link auf einem Computer sein.

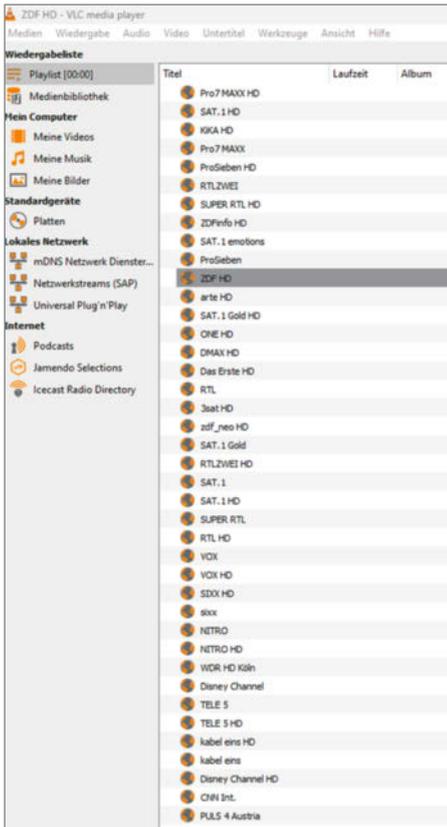
Die 8 am Ende der Dateieindung gibt an, dass die Datei in UTF-8 formatiert ist.

Aufnahmen

Wenn es schon etwas Exklusives im Fernsehen gibt, will man das manchmal auch gerne aufnehmen. Auch das geht mit Tvheadend.

Dafür in der Weboberfläche im Tab „Electronic Programm Guide“ das Programm auswählen, das gespeichert werden soll. Dazu kann man nach dem Titel der Sendung suchen oder nach einem Kanal filtern. Die Einstellungen dafür sind oben in der Bedieneleiste. Wenn man das richtige Programm gefunden hat, klickt man einfach auf den Namen und auf der sich öffnenden Übersicht auf „Record“. Die Sendung wird dann automatisch aufgenommen. Handelt es sich um eine Serie, steht „Auto-Aufnahme“ zur Verfügung: Damit werden alle kommenden Folgen automatisch aufgezeichnet.

Die Videodateien liegen im Ordner /TV/ Aufnahmen auf dem Rasperry Pi.

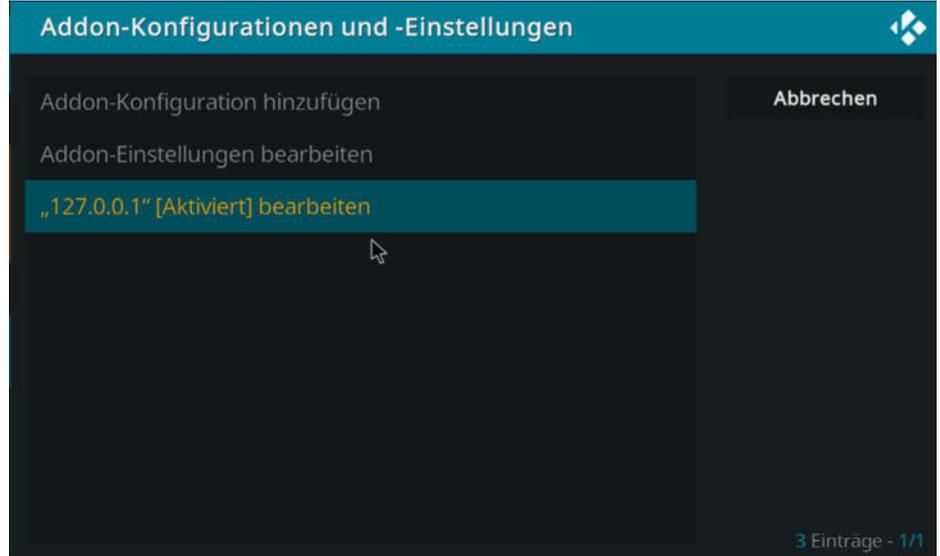


So hat man dann alle Sender in einer Liste.

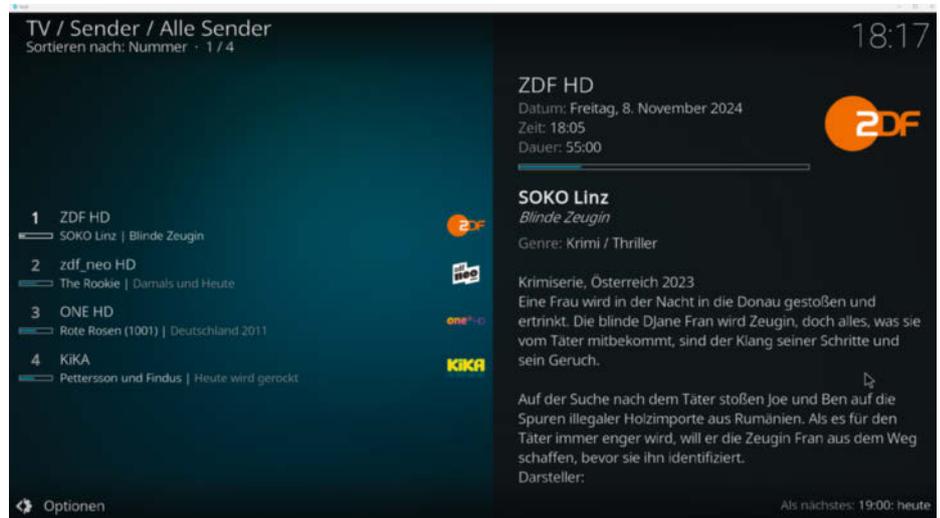
Fazit

Es ist super, die Option zu haben, spontan auch mal ins lineare Fernsehen reinzuschauen, ohne eine App für jeden öffentlich-rechtlichen Sender zu installieren oder ein Abo für den Stream eines Privatsenders zu bezahlen.

Mit Tvheadend werden auch Antennen wieder „recycelt“.



Hier muss die Einstellung angepasst werden.



Jetzt hat man eine gute Übersicht über alle Sender.

Ihre IT-Feuerwehr im Ernstfall

Mit iX gut gerüstet bei Cyberangriffen

Heft + PDF mit 28% Rabatt

Der iX-Notfallguide
Ransomware, Datenlecks und Co. erfolgreich überleben

Angriffe erkennen
Die wichtigsten Angriffsmuster, Erkennen von Cyberangriffen, Erkennen von Logg-, DDoS- und SQL-Injection-Angriffen

Incident Response
Notfallpläne und -maßnahmen, Kommunikation mit dem Management, Berichterstattung an die Behörden, Die Rolle der IT-Abteilung

Folgen bewältigen
Notfallpläne und -maßnahmen, Wiederherstellung von Daten, Kommunikation mit dem Management, Berichterstattung an die Behörden, Die Rolle der IT-Abteilung

Vorbereitet sein für den Cyberangriff
Notfallpläne und -maßnahmen, Kommunikation mit dem Management, Berichterstattung an die Behörden, Die Rolle der IT-Abteilung

Heft für 19,90 € • PDF für 17,49 €

• Heft + PDF 26,50 €

shop.heise.de/ix-notfallguide24

Jetzt bestellen!



Tastatur mit OLED-Displays

Mit Displays in den Keyboard-Tasten hat man Shortcuts immer im Blick. Kommerzielle Lösungen bieten aber oft kein ideales Schreibgefühl. Das Open-Source-Projekt PolyKybd löst das, indem es die Displays so in den Tastenkappen verstaut, dass man sie auf herkömmliche Switches setzen kann.

von Thomas Pollak



Kurzinfo

- » Split-Keyboard mit OLED-Displays bauen
- » Tastenkürzel sichtbar machen
- » Keyboard-Layouts nach Bedarf anpassen

Checkliste



Zeitaufwand:
ca. 8 Stunden



Kosten:
ca. 400 Euro

Mehr zum Thema

- » Carsten Wartmann, Extra-Tastatur für Videokonferenzen, Make 3/21, S. 8
- » Allen Wong, Intelligentes Makro-Pad, Make 2/22, S. 32
- » Andreas Känner, Ergonomische Split-Tastatur im Eigenbau, Make 2/22, S. 70

Alles zum Artikel
im Web unter
make-magazin.de/xjn5



Rückblick: Tastaturen mit Displaytasten

Im Jahr 1984 unternahm der deutsche Erfinder Reinhard Engstler mit dem „TheBoard“ („LC Board“ in den USA) einen frühen Versuch, Displays in Tasten zu integrieren. Dafür stattete er alle F-Tasten mit Punktmatrix-LCDs aus, wobei die Auflösung bei 20×8 Pixeln lag, was ausreichte, um ein paar Textzeichen oder Symbole anzuzeigen, die man mit einem Konfigurator auch selbst erstellen konnte (siehe Link in der Kurzinfo).

Die Firma „Art. Lebedev Studio“ wagte 2007 einen Anlauf mit dem Optimus Maximus. Es handelte sich dabei um ein „Full Size“-Keyboard mit 113 Tasten, wobei sich in jeder Taste ein OLED-Farbdisplay befand. Diese unterstützten eine Auflösung von 48×48 Pixeln mit 65.536 Farben. Ein Quantensprung, allerdings mit einem Preisschild von etwa 1600 US-

Dollar und einem externen Netzteil, um die Stromversorgung sicherzustellen. Wenn man den Berichten der wenigen Leute traut, die jemals eine solche Tastatur benutzt haben, waren die Tasten sehr schwergängig und das Tippen häufig ermüdend.

Aktuelle Konzepte wie Flux oder Nemeio setzen auf einen großen Farb- oder eInk-Bildschirm, der sich hinter transparenten Tasten befindet, die seitlich mechanisch oder magnetisch gelagert werden. Der Vorteil dieses Designs liegt auf der Hand: Es ist wesentlich einfacher, ein einziges Display anzusteuern als rund 100 separate Displays. Im Bereich von Makro-Pads und Stream-Decks hat sich diese Technik bereits etabliert. Allerdings ist das Schreibgefühl nicht mit einer richtigen Tastatur vergleichbar.

Jedes Programm verfügt über eine Vielzahl von Tastenkürzeln, die das Arbeiten erleichtern sollen. Während sich Befehle, die man oft verwendet, noch gut einprägen, muss man für selten verwendete Shortcuts allerdings Menüs studieren – was sicher nicht jedermanns Hobby ist. Wäre es da nicht praktischer, wenn die Tastatur je nach Programm einfach anzeigen könnte, welche Kürzel zur Verfügung stehen?

Genau das war meine Motivation, mit dem Keyboard-Projekt PolyKybd zu starten, bei dem jede Taste ein Display enthält. In diesem Artikel erkläre ich, wie das Projekt entstanden ist, und aus welchen Komponenten der Entwurf besteht. Den Source-Code, Schaltpläne und PCB-Layouts, 3D-Druckdaten sowie weitere Hinweise für den Nachbau findet ihr im GitHub-Repository des Projekts und als Video.

Erste Versuche

Mein Ziel war es von Anfang an, in erster Linie eine ordentliche mechanische Tastatur zu bauen, deren Tasten (Switches) auch Displays besitzen. Diese sollten aber die Grundfunktion nicht beeinflussen.

Als ich Ende 2020 damit anfang, ein kleines monochromes 0,49-Zoll-OLED-Display auf einen handelsüblichen Keyboard-Switch zu setzen, hatte ich allerdings vorher keine Recherche zum historischen Hintergrund

durchgeführt. Eher zufällig, da ich ein kleines Display am Schreibtisch neben der Tastatur herumliegen hatte, begann ich damit zu experimentieren. Daraus entstand ein CAD-Modell für einen ersten Versuch mit dem 3D-Drucker (Bild 1).

Die Teile aus transparentem PETG hielten das Display mit Platine an Ort und Stelle. An die Platinen lötete ich handelsübliche Litzen-drähte. Diese waren leider etwas starr, aber für einen einfachen Test gut genug. Ein Arduino

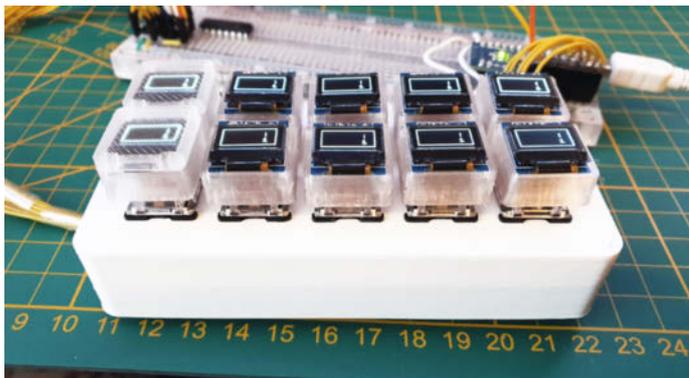


Bild 1: Ein erster funktionierender Prototyp. Jetzt musste alles nur noch etwas kompakter werden.



Bild 2: Das Display sitzt mit einer 3D-gedruckten Halterung auf dem Switch. Zum Schluss noch die Kappe drauf und fertig.

Nano übernahm die notwendige Kommunikation mit dem Displaychip.

Obwohl nur 0,49 Zoll klein, waren die Displays dennoch zu groß für die Tastenkappen (Keycaps). Damit es passte, musste ich die Seitenwände der Keycaps rechtwinklig drücken. Im Normalfall laufen sie jedoch nach oben etwas schräg zusammen.

Nach einer kurzen Suche stellte sich heraus, dass es noch kleinere Displays mit 0,42 Zoll gibt, die sogar ohne Trägerplatine und damit kompakter geliefert werden. Diese Displays mit dem SSD1306-Controller lassen sich per SPI ansteuern und haben eine Auflösung von 72×40 Pixeln.

Auf engstem Raum

Es hat einige Versuche gebraucht, bis ich zu einem (für mich) zufriedenstellenden Ergebnis kam. Eine Display-Taste sollte nach meinen Maßstäben keinen spürbaren Widerstand beim Auslösen aufweisen und der Zusammenbau sollte so einfach sein, dass mit Ausnahme 3D-gedruckter Elemente keine Spezialteile benötigt werden.

Die 0,42-Zoll-Displays passten exakt in sogenannte „Relegendable Keycaps“, wie sie in Kassensystemen häufig zu finden sind – also

transparente Tastenkappen, unter denen sich eigentlich kleine beschriftete Kärtchen befinden. Mit kleineren Modifikationen des inneren Plastikstamms konnte ich die Displays anstelle der Kärtchen einlegen.

Anfangs habe ich noch jeden Plastikstamm einzeln mit Feile, Messer und zum Teil mit dem Lötkolben bearbeitet. Eine 3D-gedruckte Halterung mit der richtigen Passform stellte sich aber als viel praktikabler heraus (Bild 2).

Viele Switches haben zufälligerweise auch einen Schlitz, der eigentlich für eine Farb-LED gedacht ist, sich aber zum Glück für die Flachbandleitung des Displays eignet. Bei meinem ersten Versuch war der Schlitz der Switches etwas zu schmal, daher habe ich ihn mit einer Feile bearbeitet, bis die Breite passte.

Bei der handelsüblichen Variante des Displays waren die Bandleitungen leider zu kurz. Um die mechanische Beanspruchung zu minimieren, führte ich das Flachbandkabel vom Switch durch die Platine mit einer 270-Grad-Schleife bis zum Sockel (Bild 3). Dafür musste ich die Bandleitungen verlängern, indem ich die Pins des Flachbandkabels mit denen des Displays flach zusammengelötet habe (Bild 4). Allerdings war das ziemlich aufwendig, und weil danach auch nicht jedes Display funktionierte, suchte ich nach Alternativen.

Die Lösung fand ich bei einem chinesischen Display-Hersteller, der entsprechende Anpassungen vornehmen konnte. Nachdem ich die Anpassungsgebühren gezahlt und versprochen hatte, mehrere tausend Stück abzunehmen, bekam ich eine Vorabversion des Displays, das schmalere und vor allem längere Bandleitungen aufwies.

Wer einen vollständigen Eigenbau plant, muss natürlich nach wie vor selbst eine Verlängerung des Displaykabels vornehmen (genaue Anweisungen findet ihr im GitHub-Repository des Projekts). Ich versuche derzeit, ein PolyKybd-Kit auf die Beine zu stellen, um diesen Schritt etwas zu vereinfachen. Interessenten können sich über den CrowdSupply-Link in der Kurzinformatik registrieren und bekommen Bescheid, sobald ich es geschafft habe.

Physisches Layout

Auf der Suche nach einem ergonomischen Design entschied ich mich für ein Split-Key-board, sodass eine Tastaturhälfte genau für die jeweilige Hand-Orientierung zurechtgerückt werden kann – der Form der Finger folgend und mit einem Column-Stagger anstatt des üblichen Row-Staggers. Tastenreihen sind also vertikal verschoben statt horizontal (Bild 5).



Bild 3: Das Displaykabel ist von unten an der Platine befestigt und bildet dabei eine Schlaufe.

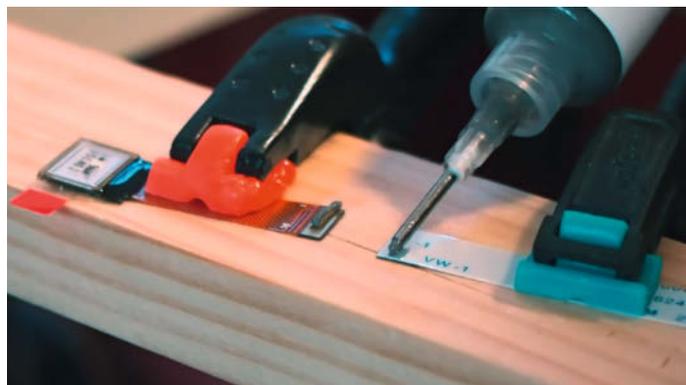


Bild 4: Da die Flachbandkabel so filigran sind, erfordert es viel Fingerspitzengefühl, sie zu verlängern.



Bild 5: Das Tastaturlayout des PolyKybd nutzt vertikal versetzten Reihen (Column-Stagger).

Zum Glück weisen die meisten Split-Keyboards auch keine größeren Tasten wie eine 6-U-Leertaste oder eine 2,5-U-Umschalttaste auf (siehe Kasten „Keycap-Größen“). Beim PolyKybd nutze ich daher lediglich 1-U- und 1,25-U-Keycaps, wodurch ich keine Stabilisierer für breitere Tasten benötige. Das vereinfacht die mechanische Konstruktion. Allerdings bedeutet die Trennung auch, dass auf der Software-Seite eine Synchronisation zwischen den Tastaturhälften nötig ist.

72 Displays adressieren

Da die Chip-Knappheit im Jahr 2021 viele Mikrocontroller für mein Projekt zu teuer

machte, ich aber ein leistungsstärkeres Modell als den Nano brauchte, um meine Tastatur umzusetzen, entschied ich mich für einen Raspberry RP2040 und plante für jede Tastaturhälfte einen eigenen Chip ein.

Dadurch standen mir zweimal 30 GPIO-Pins zur Verfügung. Ich musste mit ihnen aber nicht nur die eigentlichen Tasten, sondern auch noch jeweils 36 Displays über SPI ansteuern. Während sich SCLK (serielle Taktleitung), SDIN (serielle Datenleitung) und D/C (unterscheidet zwischen Daten- und Steuerkommandos) bündeln ließen, brauchte ich für jedes Display eine eigene Chip-Select-Leitung, um es individuell anzusprechen – also mehr Anschlüsse, als mir zur Verfügung standen.

Keycap-Größen

Die Tastenbreite wird bei Keyboards in „U“ angegeben. 1U entspricht einem horizontalen Mitte-zu-Mitte-Abstand von 19,05 mm, wobei die typische Breite aufgrund des Abstands der Tasten zueinander etwa 18 mm beträgt. Die Fingerauflage hat je nach Form zwischen 12 und 14 mm Seitenlänge. Abhängig vom Tastentyp variiert die Breite in der Regel zwischen 1U (u. a. für Buchstaben, Zahlen oder F-Tasten) bis 6,25 U für die Leertaste.

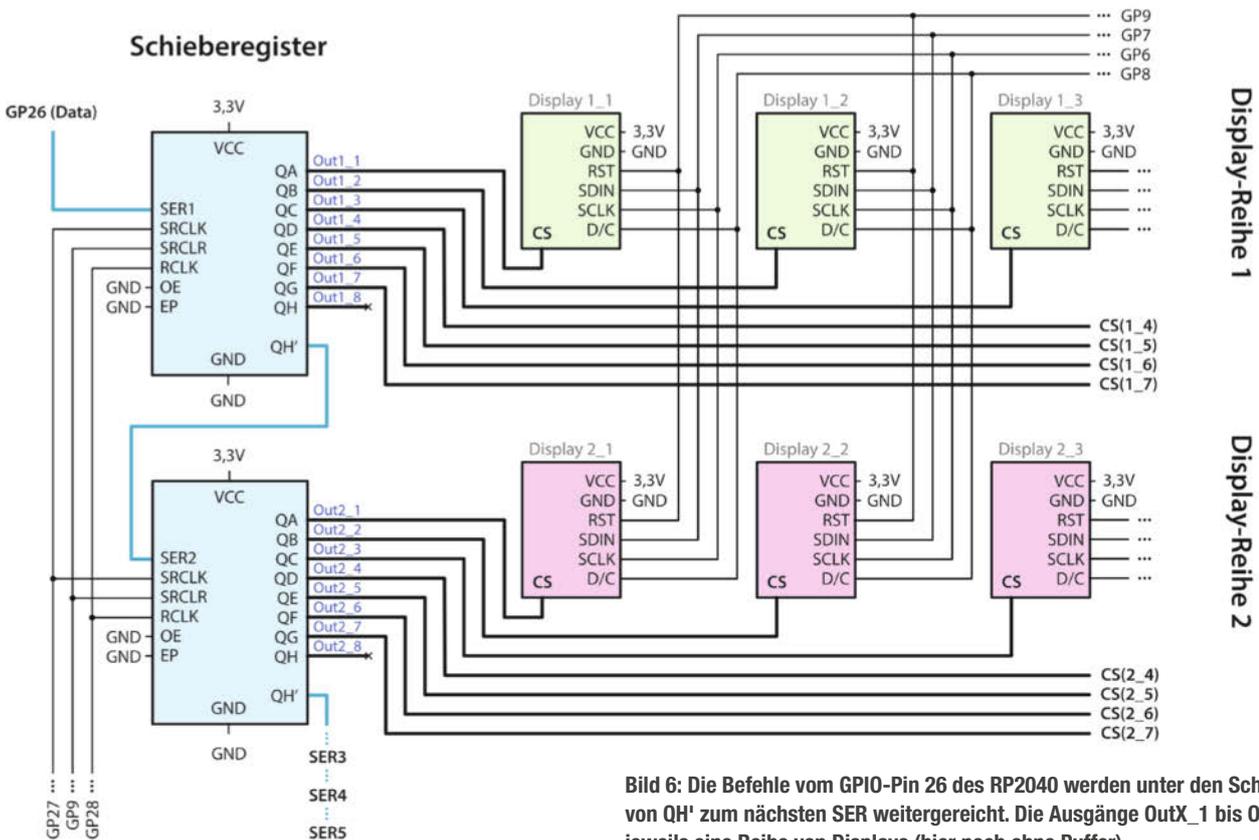


Bild 6: Die Befehle vom GPIO-Pin 26 des RP2040 werden unter den Schieberegistern von QH' zum nächsten SER weitergereicht. Die Ausgänge OutX_1 bis OutX_8 steuern jeweils eine Reihe von Displays (hier noch ohne Puffer).

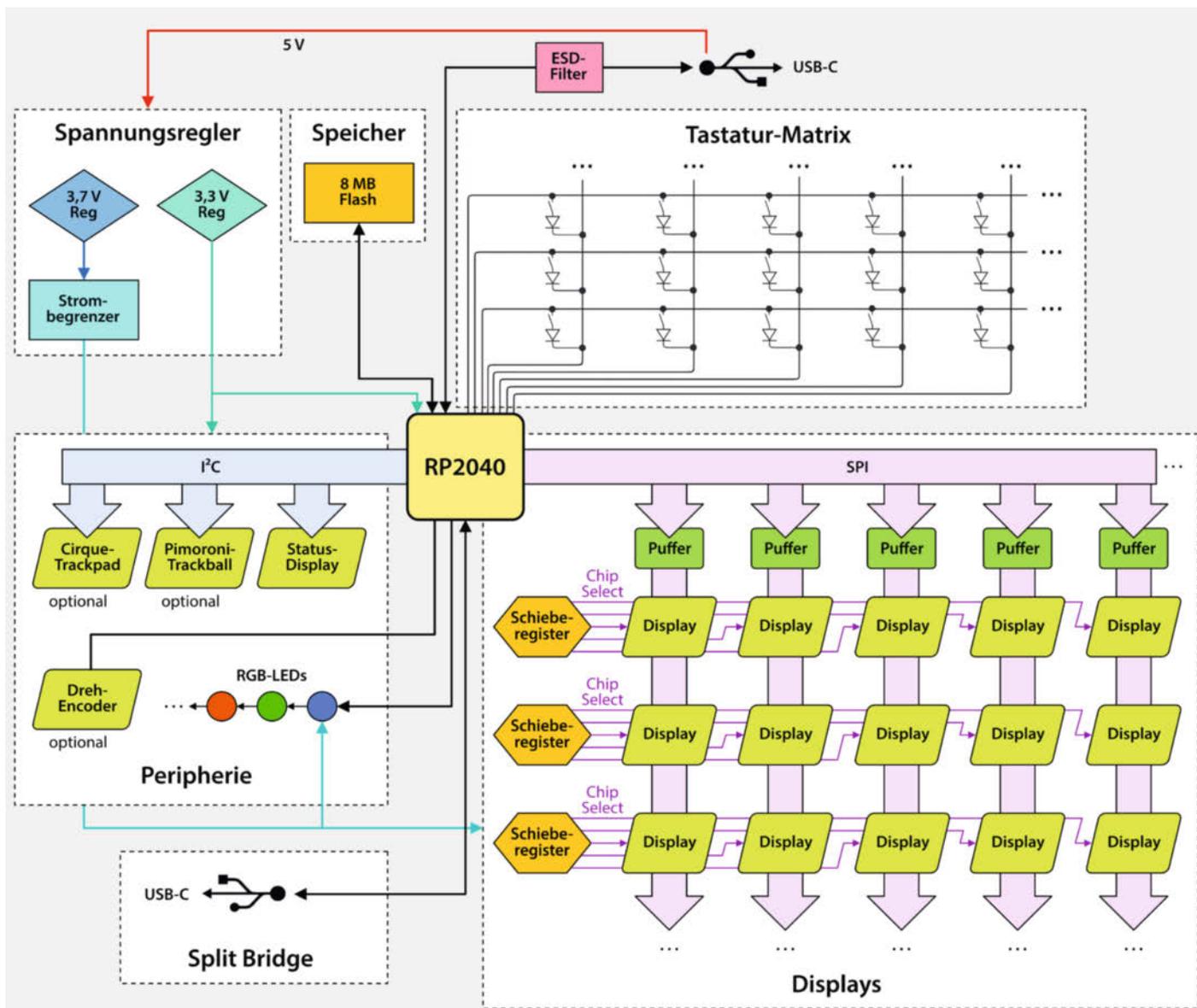


Bild 7: Schematischer Aufbau einer Tastaturhälfte. Die vollständigen Schaltungen findet ihr als KiCad-Dateien im GitHub-Repository des Projekts.

Nach einer kurzen Recherche zum Thema IO-Expander entschloss ich mich, 8-Bit-Schieberegister (Typ 74HC595BQ) zu verwenden, denn die lassen sich einfach hintereinander schalten und können ein serielles Signal entgegennehmen. Ein kurzer Testaufbau auf dem Steckbrett funktionierte auch ohne weitere Probleme, woraus sich die Verschaltung in Bild 6 ergab. Jedes Schieberegister steuert eine Reihe von bis zu acht Displays an – für 36 Displays benötigt man also fünf Register (für 4x7 und 1x8 Displays).

Um einen MCU-Pin zu sparen, habe ich den Output-Enable-Pin (OE) der Register auf GND gezogen. Damit die Displays Befehle entgegennehmen, müssen ohnehin Daten über den SPI-Bus gesendet werden (also Clock und Daten im Einklang übertragen werden). Somit konnte ich es mir erlauben,

immer das jeweilige Ausgangssignal der Schieberegister direkt auf die Chip-Select-Leitungen der Displays zu legen.

Schaltungsentwurf

Mit dieser Lösung entwarf ich zwei Platinen für meine Tastatur. Die für den RP2040 nötige Peripherie entnahm ich dem Datenblatt des Pi Pico, wählte einen Flash-Speicher mit 8 MB und spendierte der Schaltung eine USB-C Buchse. Mit den 30 verfügbaren Pins je Seite steuerte ich die Peripherie nun wie folgt an:

- 13 Pins für die 5x8 Tastenmatrix (mehr Hinweise zur Keyboardmatrix über den Link in der Kurzinfo)
- 4 Pins für den SPI-Bus zur Datenübertragung und die Displays

- 2 Pins für Reset und Command/Data-Selection, verbunden mit allen Displays
 - 3 Pins, um mit den Schieberegistern zu sprechen, die entscheiden, welches Display das Chip-Select-Signal erhält
 - 2 Pins für die Kommunikation mit der anderen Tastaturhälfte
- Damit blieben noch sechs Pins übrig, die ich nicht für die Displaytasten des PolyKybd brauchte. Diese wollte ich natürlich nicht verschwenden und fügte eine Beleuchtung, alternative Eingabemöglichkeiten und ein Statusdisplay hinzu (Bild 7):
- 3 Pins (I2C) für die Kommunikation mit dem Statusdisplay und dem Trackpad (Pimoroni Trackball PIM447)
 - 2 Pins für einen Dreh-Encoder
 - 1 Pin für die Datenleitung der RGB-Beleuchtung (à la WS2812)



Bild 8: Die Verstrebungen stabilisieren die Schale, damit sie sich nicht verzieht.



Bild 9: Die Auflösung der Displays reicht sogar für kleine Piktogramme wie Emojis.

RP2040 am Limit

Wie sich beim ersten Prototypen zeigte, stießen die Ausgänge des RP2040 mit 36 Displays auf den SPI-Takt- und Datenleitungen an ihre Fan-Out-Grenze (maximale Belastbarkeit für ausgehenden Strom). Dies machte sich durch scheinbar zufällig gesetzte Pixel auf einigen wenigen Displays bemerkbar. Anfangs tappte ich bei diesem Problem im Dunklen. Den Fehler durch das Anhängen eines Logikanalysators aufzudecken, gelang mir nicht, da das Problem verschwand, sobald der Logikanalysator an den Busleitungen hing.

Erst durch ein eher zufälliges Gespräch mit einem Arbeitskollegen kam das Fan-Out-Problem zutage. Die Lösung brachte dann ein 1-Bit-Hardwarepuffer (Typ SN74LVC1G126) pro Display-Spalte (also für jeweils fünf Displays). Damit musste der RP2040-Ausgang nur noch die 1-Bit-Puffer treiben.

Platinenlayout

Für das Platinenlayout habe ich KiCad verwendet, wobei ich zum Positionieren der Switches das Raster auf 1/8 U, also 19,05/8 mm setzte. Da ich bei jedem Switch auch diverse Kondensatoren und Widerstände für den Betrieb der Displays benötigte, konstruierte ich das komplette Layout zunächst für einen einzelnen Switch und achtete darauf, dass sich Busleitungen horizontal bzw. vertikal weiter verdrahten ließen.

Mit dem „Replicate Layout“-Plug-in konnte ich das Layout danach vervielfältigen, indem ich einen weiteren Switch positionierte und alle Komponenten und Traces in dessen „Courtyard-Bereich“ (sicherer Abstand zu anderen Bauteilen) wiederherstellen ließ. So musste ich nur mehr die Busleitungen zwischen den Switches verbinden. Die einzige Voraussetzung dafür war, dass alle Bauteile, die wiederhergestellt werden sollten, sich in einem „Hierarchical Sheet“ befinden mussten, also in der Schematik im gleichen Schaltblock. Aus diesem Grund packte ich den Switch, alle relevanten Komponenten für das Display mit dem Flachbandkabel-Sockel sowie die RGB-LED in ein solches „Hierarchical Sheet“.

Den Rest der Schaltung (Spannungsregler für 3,3V, RP2040 samt Flash-Speicher, USB-C-Sockets usw.) musste ich händisch anlegen. Die erste Version der Platine entstand zweilagig. Da ich im Nachhinein ein paar Anpassungen vornehmen musste, sich aber manche Leitungen dabei nicht zum Ziel führen ließen, schwenkte ich schließlich auf vier Lagen um.

Gehäuse

Um ein passendes Gehäuse zu erstellen, exportierte ich in KiCad den Platinenumriss sowie Montagelöcher, Status-LED und Reset-Taster-Aussparungen als SVG-Dateien. Diese las ich in OpenSCAD ein und extrudierte sie, sodass ich (mit Rücksicht auf Toleranz) ein perfektes Gehäuse erhielt. Lediglich ein paar op-

tionale seitliche Bohrlöcher für das Tenting-Kit, mit dem man den Winkel der Tastatur anpassen kann, positionierte ich manuell.

Das Gehäuse habe ich bei JLC-PCB mit Resin-3D-Druck herstellen lassen, wobei bei der Größe meiner Tastaturhälften bereits gewisse Verformungen auftraten. Um diese Verformungen so weit wie möglich zu verhindern, fügte ich später vertikale Verbinder und ein Gitter mit Verstrebungen am Boden des Gehäuses ein (Bild 8).

Zusammenbau

Für den letzten Schritt bereitete ich alle Tasten mitsamt der Switches und Displays vor. Da ich mich für Hotswap-Sockel entschieden hatte, konnte ich die Switches, ohne zu löten, einfach aufstecken. Damit man die Platine nachher nicht sieht, habe ich eine Abdeckplatte (im Keyboard-Jargon einfach „Plate“ genannt) als Aluminium-PCB fertigen lassen.

Firmware

Um nicht von Grund auf eine neue Keyboard-Firmware zu programmieren, habe ich „Quantum Mechanical Keyboard“ genutzt, kurz QMK. Die Firmware ist in C geschrieben und nutzt für ARM-basierende Mikrocontroller ChibiOS als Basis. Damit bekommt man frei Haus zahlreiche Grundfunktionen: verschiedene Keyboard-Layer, Unicode-Input, die Möglichkeit, Layer mithilfe der Software VIA

Fontkonverter Beispiele

```
fontconvert -f~/NotoSans-Regular.ttf -s14 -v _Base_ 0x20 0x7e > ~/gen/0NotoSans-Regular_Base_14pt.h
fontconvert -f~/NotoSans-Regular.ttf -s16 -v _Greek_ 0x384 0x385 0x391 0x3c9 > ~/gen/NotoSans-Regular_Greek_16pt.h
fontconvert -f~/NotoSans-Regular.ttf -s16 -v _Cyrillic_ 0x401 0x46b 0x490 0x4bb 0x4d8 0x4e9 > ~/gen/NotoSans-Regular_Cyrillic_16pt.h
fontconvert -f~/NotoEmoji-Medium.ttf -s20 -r50 -v _Emojis_ -n0x10000 0x1f600 0x1f64f > ~/gen/NotoEmoji-Medium_Emojis_20pt.h
```

lang_lut.c - gekürzt

```

/*[[[cog
import cog
import os
import string

from openpyxl import load_workbook

wb = load_workbook(filename = os.path.join(os.path.abspath(os.path.
dirname(cog.inFile)), "lang_lut.xlsx"), data_only=True)

sheet = wb['key_lut']

#build a list of known glyphs
named_glyphs = []
sheet_named_glyphs = wb['named_glyphs']
for r in range(2,sheet_named_glyphs.max_row+1):
    named_glyphs.append(sheet_named_glyphs.cell(row = r, column = 1).value)

...

]]]*

```

über den Browser zu konfigurieren, und noch vieles mehr.

Um die Displays zu initialisieren, sende ich 36 gesetzte Bits seriell an die Schieberegister. Somit sind alle 36 Chip-Select-Leitungen aktiv

run_cog.sh

```

#!/bin/bash
cog -r lang/lang_lut.c
cog -r lang/lang_lut.h
cog -r lang/named_glyphs.h
cog -r keymap.c
cog -r keycodes.h
cog -r uni.h

```

und ich muss den Initialisierungsbefehlssatz für den SSD1306-Controller lediglich einmal für alle senden. Danach kann jeder Display-Puffer individuell geschrieben werden, ich setze lediglich einmal eine „1“ und schiebe dieses Bit für jedes Display einfach weiter.

Wie bereits erwähnt, wird im Normalfall bei Split-Keyboards nur die Keyboard-Matrix der einen Hälfte zur anderen übertragen. Damit weiß die sendende Hälfte nicht, ob auf der anderen Seite eine Umschalttaste gedrückt wurde oder ein anderer Layer oder eine andere Sprache aktiv ist. Da dies alles Situationen sind, bei denen ich die Displays aktualisieren müsste, wurde mir schnell klar, dass eine beidseitige Kommunikation für meine Zwecke

nötig war. Für Rückmeldungen bietet QMK zum Glück passende RPC-Callbacks, die man anpassen kann.

Font-Rendering und Sprachen

Die Schriftzeichen auf den 72×40 Pixeln der kleinen Displays stelle ich mit der „Adafruit GFX“-Bibliothek dar, die es mir mit dem mitgelieferten Tool fontconvert erlaubt, TTF/OTF-Schriften auf eine gegebene Größe zu rastern und nachher in einen Puffer zu schreiben.

Da die Bibliothek und der Fontkonverter nur 1-Byte-Chars unterstützen, habe ich das GitHub-Repository geforkt (siehe Link in der Kurzinfo) und das Interface erweitert, um 2-Byte-Chars zu verwenden. Ein paar zusätzliche Anpassungen des Fontconverters erlauben mir außerdem, Unicode-Glyphen (also solche, die eigentlich mehr als 2 Bytes benötigen) über 0xffff in sogenannte „Private Use“-Bereiche zu verschieben und auch gleich mehrere Unicode-Intervalle auf einmal zu konvertieren (siehe Listing „Fontkonverter-Beispiele“).

Neben verschiedenen Sprachen verwende ich auch Emojis, die über den Unicode-Support von QMK an die Displays gesendet werden (Bild 9).

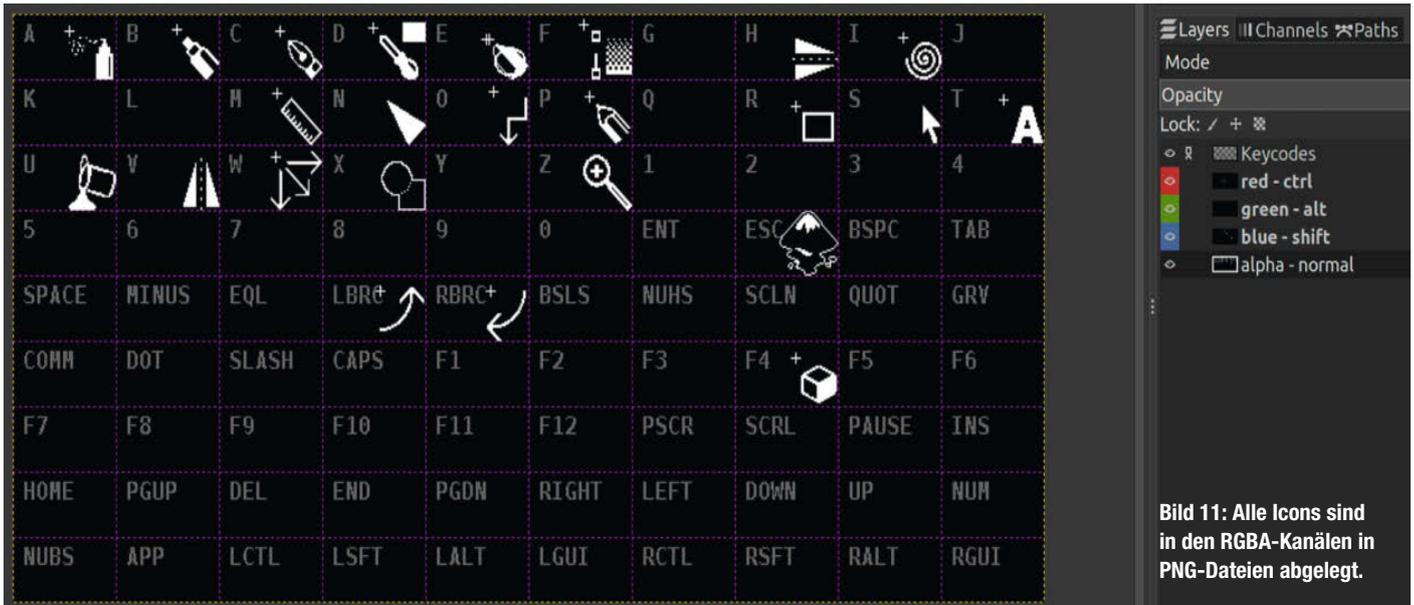
Im Moment unterstützt mein Projekt etwa 20 verschiedene Sprachen. Die Tastenbelegung für die jeweilige Sprache organisiere ich in einer Tabelle, denn alle im Source-Code zu warten, wäre unübersichtlich. Vor dem Kompilieren der Firmware muss ich die Daten nur noch in den C-Code integrieren. Dafür verwende ich die Python-Anwendung cog (siehe Link in der Kurzinfo). Sie erkennt speziell formatierten Python-Code in den Kommentaren meines C-Codes, führt ihn aus und fügt das Ergebnis als ausführbaren C-Code ein (siehe Listing „lang_lut.c – gekürzt“). Da ich cog in mehreren Dateien verwende, nutze ich das Skript run_cog.sh, um alle Source-Codes auf einmal zu aktualisieren, wenn ich in der Tabelle etwas ergänzt habe.

Mit der Unicode-Unterstützung lässt sich aber noch mehr machen: Als hauptberuflicher Programmierer habe ich das US-ANSI-Layout sehr gerne, da alle Klammern und sonstige Sonderzeichen sehr leicht erreichbar sind. Damit bleibt kaum Platz für ä, ö, ü, und ß. Dafür nutze ich auf der Tastatur eine INTL-Taste (für „International“), mit der ich meine Umlaute über INTL+A, +O, +U und +S erreiche. Diese Sonderzeichen mache ich danach zur Laufzeit wählbar (Bild 10).

Da es nun keine richtige Keyboard-Sprache mehr gibt, müssen alle Zeichen per Unicode versendet werden. Das Praktische daran: Ich kann auf dem Computer bei einem Layout bleiben, z. B. US. Dies funktioniert allerdings nur für Latein-basierende Sprachen. Bei Grie-



Bild 10: Für manche Zeichen reicht es, sie nur bei Bedarf einzublenden.



chisch, Koreanisch, Japanisch oder dergleichen bleibt der Keyboard-Sprachwechsel am Computer nicht aus.

Software für den Computer

Um nun das volle Potenzial meiner Tastatur entfalten zu können, benötigte ich schließlich noch ein Programm auf dem Computer, das dem Keyboard mitteilt, welche Sprache aktiv ist, und auch den Sprachwechsel vom Keyboard aus erlaubt. Auch die Shortcut-Symbole mussten abhängig vom gerade aktiven Programm an das Keyboard gesendet werden. Dafür programmierte ich den PolyKybdHost (siehe Link in der Kurzinfo). Damit das Programm plattformübergreifend funktioniert, verwendete ich Python.

Die Kommunikation erfolgt für Eingabegeräte über das HID-Protokoll (Human Interface Device). Auf der Host-Seite gibt es eine

fertige Python-Bibliothek HidHelper, auf der Firmware-Seite kann man die QMK-Funktion `raw_hid_receive(uint8_t *data, uint8_t length)` in der Datei `keymap.c` anpassen. Leider lassen sich maximal nur 32 Bytes in einem HID-Report senden. Daher teile ich im Host-Programm die 2880 Bits bzw. 360 Bytes, die jeweils an ein Tasten-Display gesendet werden, in 15 Reports je 24 Bytes auf.

Die Grafiken für die Tastenkürzel organisiere ich derzeit in PNG-Dateien, wobei ich alle Kanäle (RGBA) einzeln nutze: den A-Kanal (Alpha) für den Normalzustand, den R-Kanal für Zeichen in Kombination mit der STRG-Taste sowie G und B für Zeichenkürzel mit ALT und Umschalt (Bild 11). Für Modifier-Kombinationen mit STRG+ALT wird ein zweites PNG benötigt.

Welche Tastenkürzel gerade angezeigt werden sollen, erkennt das Host-Programm (`pk-host.py`) am PC mit der `PyWinCtl`-Bibliothek (Bild 12) am gerade aktiven Fenster und der

darin laufenden Anwendung. Für das Mapping zwischen den Symbolen in der PNG-Datei und dem aktuell verwendeten Programm nutze ich derzeit eine YAML-Datei (siehe Listing „Mapping mit YAML“), die das Host-Programm einliest. Dann erscheinen die Grafiken für die Tastenkürzel an der richtigen Stelle (siehe Titelbild des Artikels).

Ausblick

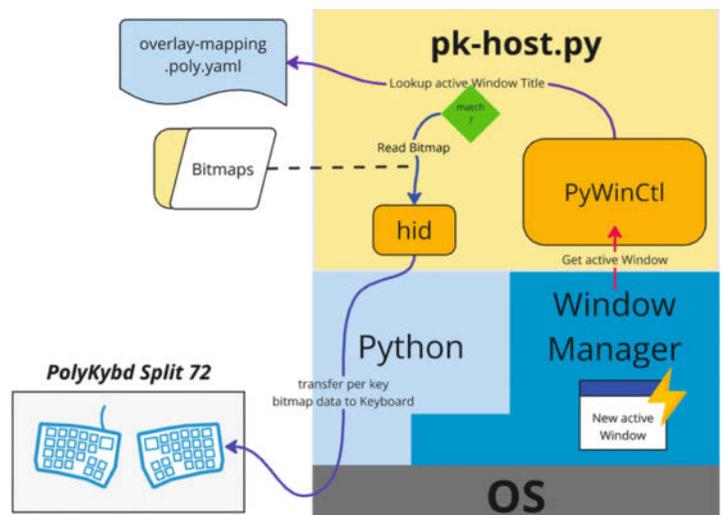
Mit der zusätzlichen Shortcut- und Sprachverwaltung auf der Keyboard- und Computerseite ist der Source-Code des Projekts bereits ordentlich angewachsen. Sicherlich gibt es hier noch das eine oder andere nachzubessern. Im Moment nehme ich Änderungen meist im Bedarfsfall vor. Mit der Technik lässt sich sicher auch ein Makro-Pad bauen und ich werde in Zukunft bestimmt noch mal mit farbigen OLEDs experimentieren. —*akf*

```

Mapping mit YAML

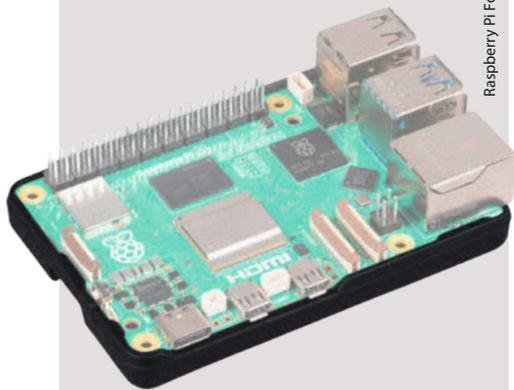
Gimp:
  overlay: overlays/gimp_template.mods.png
  app: gimp.*
Discord:
  overlay: overlays/discord_template.mods.png
  app: Discord.*
Inkscape:
  overlay: overlays/inkscape_template.mods.png
  app: inkscape.*
  title: .*Inkscape
    
```

Bild 12: Das Host-Programm erkennt, in welchem Fenster man arbeitet, und blendet die passenden Kürzel ein.



Pi Bumper

Verhüterli für Raspberry Pi 5



Ein Raspberry Pi ist toll, in der Experimentierphase bei einem Projekt liegt er gerne mal offen auf dem Schreibtisch, aus allen Seiten kommen Kabel, die GPIOs sind mit Sensoren und Co. verbunden. Allzu leicht bleibt man an einem Kabel hängen und der Pi rutscht über den Tisch und vielleicht über einen Widerstand, der dann mit seinen Beinchen einen Kurzschluss verursacht. Im besten Fall bootet der Pi neu, im schlimmsten Fall geht etwas kaputt.

Der Pi Bumper für den Raspberry Pi kann das verhindern, er schützt die Unterseite und sorgt für Rutschsicherheit. Das Produkt wurde gerade von der Raspberry Pi Ltd. angekündigt und soll 3 US-Dollar (Listenpreis) kosten. In den einschlägigen Shops wie BerryBase oder Reichelt ist er mit 2,60 Euro gelistet.

Der Pi Bumper besteht aus Silikon und wird einfach auf die Unterseite aufgezo-gen. Die Anschlüsse, der Power-Button und die SD-Karte bleiben gut erreichbar. Auch die Bohrungen in der Platine bleiben offen, sodass man ihn auch mit dem Bumper fixieren und auch halb offen in ein größeres Gehäuse, z. B. aus Metall, einbauen kann.

Alles in allem ein praktisches Zubehörteil, das man immer parat haben sollte. In unserem Forum wurden natürlich auch viele gute Vorschläge gemacht, wie man Ähnliches als Maker selbst bauen oder 3D-drucken kann. —caw

Hersteller	Raspberry Pi Foundation
URL	make-magazin.de/xxwh
Preis	2,60 €

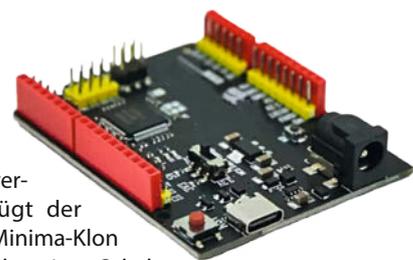
UNO-R4-Minima-Klon

Renesas RA4M1 Minima

Der chinesische Hersteller Maker Go hat einen Klon des Arduino UNO R4 Minima mit einigen Zusatzfeatures auf den Markt gebracht. Wie das Original basiert das Board auf dem RA4M1-Chip von Renesas, verfügt über einen USB-C-Port zum Bespielen mit Software und eine Hohlbuchse für die Stromversorgung.

Das Pinout ist identisch zum Vorbild, aber alle Pins sind doppelt auf dem Board herausgeführt. Dadurch lassen sich sowohl Jumperkabel mit dem Male- als auch Female-Ende anschließen. Die Header muss man selbst auflöten. Eine weitere Änderung ist der Umschalter für die Logikspannung. Der R4-Minima nutzt 5 Volt für die Kommunikation mit anderen Bauteilen und Modulen. Deshalb müssen auch diese 5 Volt unterstützen. Andere Mikrocontroller wie der ESP32 verwenden 3,3 Volt als Logikspannung. Daher gibt es auch Bauteile, die für 3,3 Volt ausgelegt sind und nicht unter 5 Volt funktionieren bzw. kaputtgehen können.

Damit man mit dem Board von Maker Go auch die vielen 3,3-V-Module nutzen kann,



verfügt der Minima-Klon über einen Schalter, der zwischen 5 Volt und 3,3 Volt umschaltet. Außerdem lässt sich das Board selbst mit einer Spannung von 6 bis 50 Volt betreiben. Das Original unterstützt Spannungen von 6 bis 24 Volt.

Über eine Tastenkombination mit der Reset-Taste und einer zusätzlichen MD-Taste kann auch in den Bootloader-Modus gewechselt werden, um neue Firmware aufzuspielen. Ansonsten sind die Spezifikationen mit dem Original-Board identisch: 256 KByte Flashspeicher und 32 KByte RAM, I2C-, UART-, CAN- und SPI-Support. —das

Hersteller	Maker Go Store
URL	make-magazin.de/xxwh
Preis	ca. 15 €

RP2350 Stamp (XL)

Raspberry Pico in Briefmarkenformat

Der Entwickler Artur Pacholec (Gründer von Solder Party) aus Schweden hat kleine Boards mit RP2350-Chip herausgebracht. Dabei sind die Stamps vollständige Picos, es reichen also 5 V und/oder ein LiPo-Akku, um sie zu benutzen. Sogar eine Ladeschaltung befindet sich mit auf dem Board. Damit eignen sie sich ideal für kleine Projekte und die Integration in eigene PCB-Layouts, denn man kann sie mit ihren „Castelated Edges“ leicht auf PCBs löten.

Für die Experimentier- und Prototypenphase steht das „Stamp Carrier XL“ zur Verfügung, auf das die Stamps aufgebracht werden. Dabei kann man sich entscheiden, entweder zu löten, 2-mm-Pitch-Header zu benutzen oder die FlexyPins vom gleichen Entwickler: Damit lassen sich die Stamps schnell austauschen.

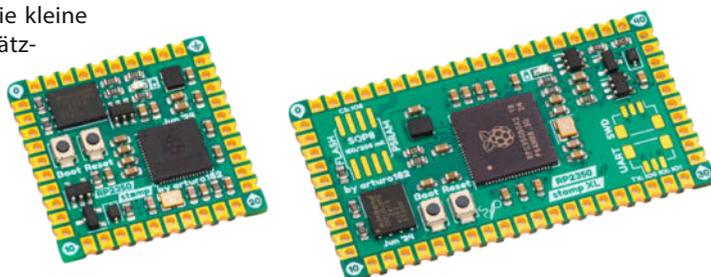
Das XL-Stamp bietet mit 48 GPIOs ganze 18 GPIOs mehr als die kleine Variante, Platz für ein zusätzliches QSPI-FLASH/PSRAM sowie die Möglichkeit, SWD und UART per Standard-Header zu verbinden. Beide Boards werden mit einem UF2-Bootloader geliefert und können per Reset-Tas-

ter oder über den BOOTSEL-Pin neu programmiert werden.

Der Carrier bietet 2,45 mm Pitchabstand, Stromversorgung (7 bis 12V) über einen Hohlstecker, USB-C, USB-A-Host, einen microSD-Slot, DVI-D, Qwiic-, UART- und SWD-Buchsen.

Die Footprints, also die Lage der Pins und Castellations, liegen in verschiedenen Formaten für den Einsatz in KiCad, Eagle, Altium und EasyEDA vor. Wie immer ist die komplette Hardware Open Source und liegt auf GitHub, ebenso die Dokumentation, Designfiles für die E-CADs sowie Zertifikate. —caw

Hersteller	Solder Party AB (Schweden)
URL	make-magazin.de/xxwh
Preis	RP2350 Stamp (XL) 12,29 €, Stamp Carrier XL 8,38 €



Ausprobiert
— von Make: —

iFixit FixHub

Tragbare Lötstation

Die tragbare Lötstation kombiniert eine USB-C-Powerbank-Station mit einem smarten LötKolben. Dieser ist auch einzeln erhältlich und funktioniert ebenso solo an Powerbanks und Netzteilen mit Power Delivery. Dann ist die Temperatureinstellung aber nur per Browser auf dem PC möglich.

Die Verpackung ist sicher und schick. Der stabile Innenkarton ist auf jeden Fall gut und noch lange nutzbar, um Sachen zu verstauen. Ich würde mir dennoch eine einfachere, umweltgerechtere Version ohne Farbe und Schickimicki wünschen.

Alles ist fein mit Beipack-Guides und auf Online-Seiten dokumentiert, aber jeder, der auch nur etwas Erfahrung hat, lötet bereits wenige Minuten nach dem Auspacken. Mit einem längeren Druck auf den linken Knopf schaltet man die Station ein, die Anzeige zeigt die aktuelle Temperatur und den Status des Akkus. Auch der LötKolben gibt Lebenszeichen mit seinem LED-Ring ab. Die Heizung ist allerdings über den Schiebeschalter am Kolben gesichert. Dieser Schalter ist nicht zu leichtgängig und wird sich so wohl auch nicht aus Versehen anschalten lassen.

Der Kolben ist klein und leicht (unter 40 g) und liegt mit seinem gummierten Griff gut in der Hand. Das mitgelieferte Kabel ist sehr flexibel und so kann man problemlos in allen möglichen Lagen löten, zumal man die Station ja auch leicht passend positionieren kann.

Die Betriebszustände wie Aufheizen, Stand-by-Temperatur, Fehler etc. werden durch den farbigen LED-Ring am Kolben signalisiert, der beim Löten immer gut zu sehen ist.

Der Dreh-Drück-Encoder auf der rechten Seite der Powerbank-Station dient dazu, die aktuelle Temperatur des Kolbens einzustellen. Mit einem Klick gelangt man in ein Menü, um weitere Parameter einzustellen und Informationen abzurufen. Ein Fehler-Log ist auch vorhanden, denn das System bietet einige Features, die einen sicheren Betrieb ermöglichen. So gibt es etwa ein Akzelerometer im Kolben – der sorgt nicht nur dafür, dass der Kolben nach längerer Ruhe in den Standby-Mode geht, sondern auch für eine Sturzerkennung, die den Kolben abschaltet, sowie einen Kabelcheck, der zu dünne USB-Kabel erkennt.

Der Kolben hat eine Schutzkappe aus Silikon und Metall, die angeklipst an die Station auch als LötKolbenständer dienen kann. In der Anleitung wird ausdrücklich gesagt,

dass man diesen nicht benutzen sollte, wenn die Standby-Funktion deaktiviert ist, etwa weil eine zu schwache Powerbank benutzt wird.

Die Basis besitzt einen stabilen Aufsteller. Irgendwie wäre es cool, wenn man die Basis an den Gürtel anklippen könnte. Das ruft nach einer 3D-gedruckten Erweiterung, aber die Station passt auch in normale Hosentaschen.

Die Lötspitzen werden einfach durch Ab- und Anstecken gewechselt, mitgeliefert wird eine 1,5 mm abgeschrägte Lötspitze. Diese bietet einen Kompromiss aus einer Fläche für gute Wärmeübertragung und der Kante für feinere Einsätze, etwa an SMD. Mag man diese Art der Spitzen nicht, so sind aktuell noch eine 0,8-mm-Spitze in Bleistiftform und eine 1,5-mal-0,5-mm-Meißelform-Spitze erhältlich. Mehr sollen folgen.

Die Station kann auch als normale USB-C-Powerbank für zwei Geräte dienen, ein USB-C-Port zum Laden befindet sich zusätzlich an der Rückseite.

Der LötKolben und der Hub sind bestens bei iFixit dokumentiert: Anleitungen und Guides bis zu der hoffentlich nicht nötigen Reparatur und verfügbare Ersatzteile sind vorhanden und es gibt auch Schaltpläne online.

Ein schneller Versuch zeigte, dass man mit einem Terminal-Programm (etwa Tera Term) in eine Kommandozeile auf dem LötKolben kommt. Das System ist ein Zephyr Realtime Operating System (RTOS) mit einem Linux-Kernel auf einem stm32g491xx-Prozessor.

Ich löte sehr gerne mit dem FixHub. Auch in der Werkstatt, obwohl dort gute (wenn auch nicht so smarte) Lötstationen stehen. Das ist vor allem der Ergonomie geschuldet und dem niedrigen Gewicht. Dass man bei der Arbeit an größeren Geräten die Station schnell mal umstellen kann, ist viel wert. Auf der nächsten Maker Faire wird er auf jeden Fall mit dabei sein.

Ganz günstig ist das Gerät allerdings nicht. Wer aber regelmäßiger lötet, sollte den Preis vielleicht auch nicht als Argument nutzen. Ein weiterer Wermutstropfen: Der FixHub verkauft sich so gut, dass aktuell nur eine Vorbestellung mit Lieferdatum im Januar 2025 möglich ist. —caw

Der FixHub wurde uns von iFixit zur Verfügung gestellt.



```
COM5 - Tera Term VT
File Edit Setup Control Window Help
ifixitiron:~$ help
Available commands:
adc
bootloader
comms
gpio
heater
help
i2c
log
logging
mcu_sn
mxc4005
nvs
ob
otp
pubsub
reset
settings
thermistor
thermocouple
uptime
usbpd
version
ifixitiron:~$ thermistor get
COLD_JUNC_TEMP: 32.23 C
ifixitiron:~$ uptime
247654
ifixitiron:~$ thermocouple get
THERMOCOUPLE: 599.00 C
ifixitiron:~$ thermistor get
COLD_JUNC_TEMP: 34.55 C
ifixitiron:~$ thermocouple get
THERMOCOUPLE: 47.13 C
ifixitiron:~$
```

Hersteller	iFixit GmbH
URL	de.ifixit.com/fixhub-de
Preis	269,95 €

Ausprobiert
— von Make: —

Widerstands- sortiment

in Buchform schnell
parat



Standard-Transistoren, Dioden und Kondensatoren sind in ihrer Anzahl noch überschaubar in Fächer zu sortieren. Bei Widerständen wird es schon unübersichtlich und man braucht viele Fächer. Vor allem, wenn mehrere Werte lose in den Fächern liegen, dauert das Suchen lange. Muss es schnell gehen, so wird ein Wert aus dem Gurt gerupft und später wieder einfach in das Fach geworfen ... Also muss man das nächste Mal die Farbcodes entschlüsseln oder den Widerstand messen.

Für SMD-Bauteile gibt es Ringordner oder Bücher, die die Bauteile in Taschen halten. Mit etwas Suchen findet man auch für bedrahtete Bauteile noch solche Bücher. Oft als Muster-Kataloge gemeint, erscheinen sie auf Auktionsplattformen, wo man aber Glück haben muss, ein vollständiges Exemplar zu finden.

Im Funduino-Shop gibt es aber noch ungenutzte Bücher dieser Art! Die Qualität der Widerstände ist innerhalb der angegebenen 1-Prozent-Toleranz, die Anschlussbeinchen sind allerdings etwas dünn, aber das macht nichts: Wenn die zehn Stück pro Wert verbraucht sind, kann man aus gegurteten Sortimenten nachfüllen. Da auch nur ein Wert in jeder Tasche steckt, fällt es auch leichter, Ordnung zu halten und Widerstände, die man etwa auf einem Breadboard nicht mehr benötigt, wieder zurückzustecken.

In den Büchern stecken 1400 Widerstände, jeweils 10 Stück im Bereich von 1Ω bis 4,7MΩ. Mir helfen diese Bücher sehr, besonders in der Experimentierphase an einer Schaltung. —caw

Vertrieb	Funduino GmbH
URL	make-magazin.de/xxwh
Preis	15,43 €

Shaper Origin: Larkspur-Update

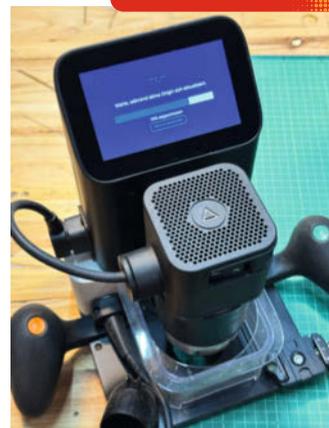
Für alle Versionen der
smarten Oberfräse

Der zu Festool gehörende Werkzeughersteller Shaper hat für die handgeführte CNC-Oberfräse Origin eine neue Firmware namens Larkspur veröffentlicht. Die Origin kann nun den Durchmesser von Rundstäben und Kreisabschnitten ermitteln, indem drei Punkte des Kreises mit der Fräse abgetastet werden. Auch Rechtecke lassen sich ausmessen, indem an jeder Seite des Rechtecks abgetastet wird. Das erleichtert etwa das Erstellen von Zapfen für klassische Holzverbindungen.

Auf der Fräse erstellte Workspaces lassen sich jetzt exportieren und wahlweise in „Shaper Studio“ im Browser bearbeiten oder die Inhalte eines Workspace können in einen anderen übernommen werden.

Auch bei „Shaper Studio“, dem Online-Tool zum Erstellen von Fräsdaten für die Origin, gibt es Änderungen: Alle Funktionen sind nun kostenlos verfügbar. Lediglich die Anzahl der Exporte auf die Fräse ist bei kostenloser Nutzung auf drei pro Monat begrenzt. Das Abo mit unlimitierten Exporten kostet 70 Euro pro Jahr (statt bisher 119 Euro für die Vollversion). Alternativ können unlimitierte Exporte für einmalige 350 Euro erworben werden, zukünftige Updates sind enthalten.

Auch der aufpreispflichtige Autopass-Modus bekommt ein Update: Bisher war die



Ausprobiert
— von Make: —

Funktion nur beim Fräsen von Innen- oder Außenkonturen nutzbar, jetzt lassen sich auch Taschen mit automatischer Tiefenänderung pro Durchgang fräsen. Das Update zum Taschenfräsen ist für bisherige Autopass-Käufer automatisch enthalten.

Alle neuen Funktionen sind auch für die erste Generation der Fräse verfügbar. Das Update lässt sich über das Menü der Fräse einspielen, wenn die Maschine im WLAN ist. —jom

Hersteller	Shaper Tools GmbH
URL	shapertools.com
Preis	kostenlos

Raspberry Pi AI HAT+

Turbo für KI-Aufgaben

Das Raspberry Pi AI HAT+ ist im Gegensatz zum Vorgänger direkt auf das HAT aufgelötet und wird nicht im M.2-Formfaktor ausgeliefert. Außerdem gibt es ein Leistungs-Upgrade: Neben einer Version mit dem bekannten Hailo-8L-Chip wie im Vorgänger gibt es auch eine Version, die auf dem Hailo-8-Chip (ohne den L-Zusatz) basiert. Dieser bietet mit 26 TOPS die doppelte KI-Power. Zum Vergleich: Für eine Copilot+-Zertifizierung von Microsoft gilt ein Minimum von 40 TOPS.

Die Unterstützung für die KI-Beschleuniger ist mittlerweile standardmäßig in Raspberry Pi OS integriert. Durch den HAT+-Standard wird die angeschlossene Erweiterung beim Booten erkannt und automatisch eingerichtet. Dadurch wird direkt die Geschwindigkeit der verwendeten PCIe-Schnittstelle vom standardmäßig eingestellten Gen 2 (5 Gbit/s) auf Gen 3 (8 Gbit/s) umgestellt.

Laut Hersteller erlaubt die schnellere Version des Chips, mehrere KI-Modelle gleichzei-



Raspberry Pi Foundation

tig zu betreiben. Als Beispiel werden in einem YouTube-Video Objekterkennung, Ausrichtung eines Objektes und Objektsegmentierung vorgestellt.

Auf der Dokumentationsseite sind die Installationsanleitungen für diese Demos erhältlich. Weitere stehen auf der Hailo-GitHub-Seite zur Verfügung. Außerdem ist dort der Hailo-Dataflow-Compiler verlinkt. Mit diesem Tool können neuronale Netze für die im HAT verbaute Hailo-Hardware kompiliert werden. In Hailos „Model Zoo“ auf GitHub werden weitere, vortrainierte Modelle bereitgestellt. Laut Raspberry Pi Limited sind Modelle für den Hailo-8L (der 13-TOPS-Version) mit dem Hailo-8 (der 26-TOPS-Version) kompatibel. —das

Hersteller	Raspberry Pi Foundation
URL	make-magazin.de/xxwh
Preis	79,90 €

CrowPanel ESP32 HMI 2.8-inch Display

Touchdisplay mit ESP32

Kleine Bedien- und Infopanels fürs Smart Home und Projekte lassen sich mit dem CrowPanel des Herstellers Elecrow spielend leicht realisieren. Dazu trägt nicht nur die flexible Hardware bei, sondern auch die gute Dokumentation mit ihren vielen Beispielen und Anleitungen.

Das CrowPanel besteht aus einem ESP32 (WROOM) in Kombination mit einem resistiven 2,8-Inch-Touchdisplay (320x240 Pixel). Das Board verfügt über diverse Anschlussbuchsen für I²C, GPIO und UART, die der Hersteller Crowtails nennt. Sie sind kompatibel zum Grove-System von Seeed. Zum Lieferumfang gehört ein dazu passender Stecker mit vier Kabeln, sodass man erste Experimente auf einem Breadboard durchführen kann. Alternativ bietet Elecrow viele fertige Sensor-Module mit passendem Stecker an.

Das Board lässt sich wahlweise über USB-C oder einen Akku mit Strom versorgen. Dank einer Ladeschaltung kann der Akku über USB-C mitgeladen werden. Ein MicroSD-Slot nimmt Karten auf, auf denen man Daten speichern kann.

Zur Programmierung des Boards und des Touchdisplays bietet der Hersteller Tutorials

und Code für die Arduino IDE, Platformio, MicroPython und die ESP IDF an. Daneben stellt er Anleitungen für die Integration in ESPHome und Home Assistant bereit.

Zur Erstellung der grafischen Oberfläche für Projekte nutzt Elecrow in seinen Anleitungen die Software SquareLine Studio. Diese ist für nicht-kommerzielle Zwecke mit leichten Einschränkungen kostenlos. In der Umgebung designt man seine GUI und platziert Buttons, Schieberegler und andere Bedienelemente und ordnet ihnen Texturen und Events zu.

Daraus erzeugt das Tool einen Satz an Dateien, die die populäre Grafikbibliothek LVGL für Embedded Systems nutzt. Für Arduino-Projekte erhält man beispielsweise einen Satz an C- und H-Dateien inklusive einer INO-Datei, die als Grundgerüst fungiert und erweitert werden kann. Für MicroPython exportiert das Studio PY-Dateien, die man auf den Controller überträgt. In den recht detaillierten Anleitungen sind allerdings Schritte dokumentiert, die in der aktuellen Fassung der zum Download angebotenen Beispiele nicht mehr notwendig sind.

SquareLine ist etwas gewöhnungsbedürftig, man bekommt damit aber leicht und ohne



tiefer gehende Kenntnisse Bedienoberflächen gestaltet. Das Panel ist ab 20 Euro erhältlich. Eine ideale und günstige Ergänzung für Projekte. —dab

Hersteller	Elecrow
URL	elecrow.com
Preis	ab 20 €

FOR BASIC TO ONE-LINERS

Knobeln mit BASIC

Der Verlag LOOK BEHIND YOU gibt Bücher zu Videospiele, Retrocomputing und Ähnlichem heraus. Das Buch „FOR BASIC TO ONE-LINERS“ ist im Juni 2024 erschienen und beschreibt und erklärt eine besondere Art des Programmierens auf Basic-Rechnern: in einer Zeile Programmcode ein besonders schönes oder schnelles Programm oder gar Spiel zu schreiben. Dies mündet regelmäßig in freundschaftlichen Wettbewerben in Foren von Retro-Fans.

Dabei wird mit allen Tricks gearbeitet, um Zeichen zu sparen und die Geschwindigkeit zu erhöhen. Diese Art des Wettkampfes ist auch heute noch sehr beliebt, erlaubt es doch, an einem kleinen Problem zu tüfteln und Neues zu lernen.

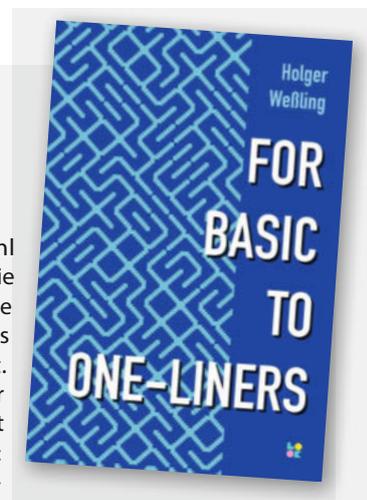
Ein Muss für alle, die in der Zeit vor dem Internet mit dem Programmieren begonnen haben. Wer keinen BASIC-Rechner mehr besitzt, kann sich an den Emulatoren für die Klassiker austoben.

Nach einer kurzen Einführung, was Einzeiler sind, wie Basic funktioniert und wie man sie eintippt, geht es ans Eingemachte: Viele Tipps zeigen, wie man Bytes spart und die Geschwindigkeit optimiert. Allerdings ist das irgendwie der 2. Schritt, denn die Einzeiler entstehen ja aus einer Programmidee und zuerst mehreren Zeilen. Daher sollten Einsteiger vielleicht mit Kapitel 14 weitermachen, in dem einige One-Liner und die Optimierungen vorgestellt werden. Hier kann man dann üben und sich für eigene Programme inspirieren lassen. In diesem Prozess werden dann die Kapitel 5 bis 13 zum sinnvollen Nachschlagewerk.

Das Buch ist gespickt mit Links zu den originalen Quellen, Wettbewerben und YouTube-Videos, die noch einmal einige Stunden Material beinhalten und zum Weiter surfen verführen. Zum Glück hat der Verlag die Links auf einer Website gesammelt, sodass man sie nicht aus dem Buch abtippen

muss. Wohl dem, der die elektronische Variante als PDF besitzt. Wer immer noch nicht genug hat: Es gibt natürlich auch 2-Liner, 10-Liner, usw.

Hinten im Buch finden sich noch sehr hilfreiche Tabellen von Befehlen und den Grafiken des Commodore 64. —caw



Titel	FOR BASIC TO ONE-LINERS
Autor	Holger Weßling, Martin Roscher
Verlag	Look behind you
ISBN	9783982602226
Seiten	160
URL	make-magazin.de/xxwh
Preis	24,90 € (Buch), 14,90 € (PDF), 29,90 € (Buch+PDF)

IMPRESSUM

Make: Nächste Ausgabe erscheint am 07. Februar 2025

Redaktion

Make: Magazin
 Postfach 61 04 07, 30604 Hannover
 Karl-Wiechert-Allee 10, 30625 Hannover
 Telefon: 05 11/53 52-300
 Telefax: 05 11/53 52-417
 Internet: www.make-magazin.de

Leserbriefe und Fragen zum Heft: info@make-magazin.de

Die E-Mail-Adressen der Redakteure haben die Form xx@make-magazin.de oder xxx@make-magazin.de. Setzen Sie statt „xx“ oder „xxx“ bitte das Redakteurs-Kürzel ein. Die Kürzel finden Sie am Ende der Artikel und hier im Impressum.

Chefredakteur: Daniel Bachfeld (dab)
 (verantwortlich für den Textteil)

Redaktion: Johannes Börnsen (jom), Ákos Fodor (akf), Marcus Hansson (mch), Daniel Schwabe (das), Ulrich Schmitz (usz), Duniá Selman (dus, Social Media), Carsten Wartmann (caw)

Mitarbeiter dieser Ausgabe: Beetlebum, Thomas Garario, Hartmut Grawe, Michael Linsenmeier, Benno Lottenbach, Philip Oettershagen, Joerg Plenert, Thomas Pollak, Nikolai Radke, Dr. Stefan Recksiegel, Kenneth Weiss

Assistenz: Susanne Cölle (suc), Martin Triadan (mat)

Layout und Satz: Steffi Martens, Lisa Reich, Nicole Wesche, Heise Medienwerk GmbH & Co.KG

Korrektur: Dörte Bluhm, Lara Bögner, Marei Stade, Christiane Tümmler, Heise Medienwerk GmbH & Co. KG

Titel: Nicole Wesche

Fotografie und Titelbild: Andreas Wodrich

Digitale Produktion: Melanie Becker, Thomas Kaltschmidt, Pascal Wissner

Hergestellt und produziert mit Xpublisher: www.xpublisher.com

Verlag

Maker Media GmbH
 Postfach 61 04 07, 30604 Hannover
 Karl-Wiechert-Allee 10, 30625 Hannover
 Telefon: 05 11/53 52-0
 Telefax: 05 11/53 52-129
 Internet: www.make-magazin.de

Herausgeber: Christian Heise, Ansgar Heise

Geschäftsführung: Ansgar Heise, Beate Gerold

Anzeigenleitung: Daniel Rohlfing (-844)
 (verantwortlich für den Anzeigenteil),
 mediadaten.heise.de/produkte/print/
 das-magazin-fuer-innovation

Leiter Vertrieb und Marketing: André Lux (-299)

Service Sonderdrucke: Julia Conrades (-156)

Druck: Dierichs Druck + Media GmbH & Co.KG,
 Frankfurter Str. 168, 34121 Kassel

Vertrieb Einzelverkauf:
 DMV DER MEDIENVERTRIEB GmbH & Co. KG
 Meßberg 1
 20086 Hamburg
 Telefon: +49 (0)40 3019 1800
 Telefax: +49 (0)40 3019 1815
 E-Mail: info@dermedienvertrieb.de
 Internet: dermedienvertrieb.de

Einzelpreis: 13,50 €; Österreich 14,90 €; Schweiz 26.50 CHF;
 Benelux 15,90 €

Abonnement-Preise: Das Jahresabo (7 Ausgaben) kostet inkl. Versandkosten: Inland 80,50 €; Österreich 88,90 €; Schweiz 123.90 CHF; Europa 95,20 €; restl. Ausland 100,80 €

Das Make-Plus-Abonnement (inkl. Zugriff auf die App, Heise Magazine sowie das Make-Artikel-Archiv) kostet pro Jahr 6,30 € Aufpreis.

Abo-Service:

Bestellungen, Adressänderungen, Lieferprobleme usw.:
Maker Media GmbH
Leserservice
 Postfach 24 69
 49014 Osnabrück
 E-Mail: leserservice@make-magazin.de
 Telefon: 0541/80009-125
 Telefax: 0541/80009-122

Eine Haftung für die Richtigkeit der Veröffentlichungen kann trotz sorgfältiger Prüfung durch die Redaktion vom Herausgeber nicht übernommen werden. Kein Teil dieser Publikation darf ohne ausdrückliche schriftliche Genehmigung des Verlags in irgendeiner Form reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Alle beschriebenen Projekte sind ausschließlich für den privaten, nicht kommerziellen Gebrauch. Maker Media GmbH behält sich alle Nutzungsrechte vor, sofern keine andere Lizenz für Software und Hardware explizit genannt ist.

Für unverlangt eingesandte Manuskripte kann keine Haftung übernommen werden. Mit Übergabe der Manuskripte und Bilder an die Redaktion erteilt der Verfasser dem Verlag das Exklusivrecht zur Veröffentlichung. Honorierte Arbeiten gehen in das Verfügungsrecht des Verlages über. Sämtliche Veröffentlichungen in Make erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes.

Warennamen werden ohne Gewährleistung einer freien Verwendung benutzt.

Published and distributed by Maker Media GmbH under license from Make Community LLC, United States of America. The 'Make:' trademark is owned by Make Community LLC. Content originally partly published in Make: Magazine and/or on www.makezine.com, ©Make Community LLC 2024 and published under license from Make Community LLC. All rights reserved.

Printed in Germany. Alle Rechte vorbehalten.
 Gedruckt auf Recyclingpapier.

© Copyright 2024 by Maker Media GmbH

ISSN 2364-2548

Nachgefragt

Welche nervige Aufgabe würdest Du am liebsten einer KI oder einem Roboter übergeben?



Thomas Garaio
 Liebefeld, hat mit der Oxocard den Weihnachtsstern auf S. 50 gebaut.

Die Kompost-, Glas-, Papier-, Plastik- und Mülltrennung bereitet mir am wenigsten Freude. Am liebsten alles wieder in die Tüte und dann ab zu einem Musk-Roboter, der den Müll in stoischer Ruhe und ohne die Nase zu rümpfen in Verwertbares aufteilt.



Stefan Recksiegel
 Garching, bringt mit seinen Flirt-Amuletten auf S. 34 Menschen zum Leuchten.

Ich würde gerne das Badezimmer putzen lassen, mit dem Staubsaugen funktioniert das ja schon ziemlich gut. Alternativ – und vielleicht etwas realistischer – würde ich gerne meine Steuererklärung von einer KI machen lassen.



Hartmut Grawe
 Bad Sachsa, zeigt auf S. 74, wie man sich mit Lisp einen Heimcomputer baut.

Eindeutig: die Reinigung der Dusche vom Kalk!



Philip Oettershagen
 Köln, erstellt auf S. 94 Porträts aus Fäden.

Ich würde gerne sämtliche Steuerangelegenheiten in eine KI kippen. Die scheint mir wie gemacht für solche Aufgaben und hätte vielleicht sogar Spaß daran.

Inserentenverzeichnis

dpunkt.verlag GmbH, Heidelberg.....	13	TUXEDO Computers GmbH, Augsburg.....	124
OXON AG, CH-Liebefeld.....	21		
SIGS-DATACOM GmbH, Troisdorf.....	2		

Ein Teil dieser Ausgabe enthält Beilagen der DIMABAY GmbH, München.

Maker Faire®

Das Format für
Innovation & Macherkultur

Die nächsten Events



... weitere folgen.

maker-faire.de



Notebooks für jeden Bedarf Von ultramobil bis Desktop-Ersatz



Sehr leichtes
3-in-1 Linux-Convertible



Hoch effizienter Desktop-Ersatz
mit leistungsstarker
AMD Ryzen 9 7945HX

Egal ob ultramobil für unterwegs oder als leistungsstarker Desktop-Ersatz – bei TUXEDO finden Sie das passende Notebook für jeden Bedarf. Unsere Geräte sind speziell für den reibungslosen Einsatz mit Linux optimiert und können individualisiert bestellt werden.

Ob für berufliche Anforderungen, private Nutzung oder Gaming – mit TUXEDO erhalten Sie erstklassige Hardware, maßgeschneidert für Ihre Einsatzzwecke. Entdecken Sie jetzt unser vielfältiges Angebot in unserem Online-Shop und erleben Sie Linux auf höchstem Niveau!

