G3078E (D) € 14,90 | CHF 27,00 | (A, B, L) € 16,00

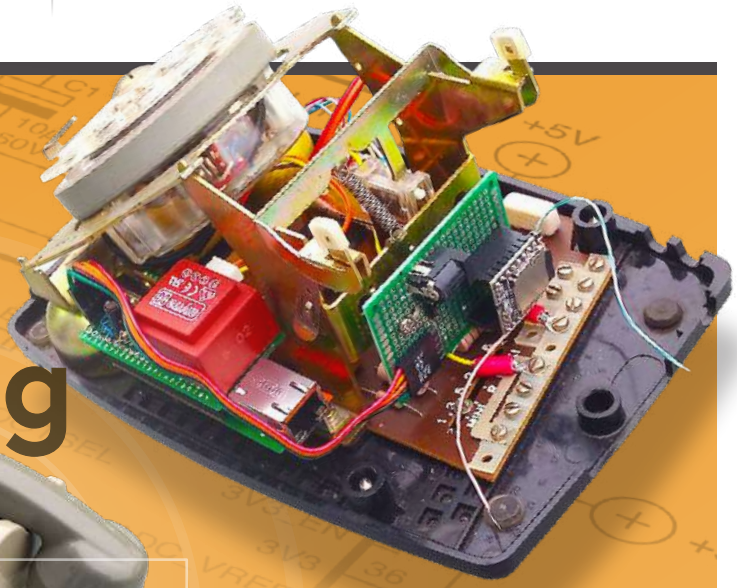
elektorMAG

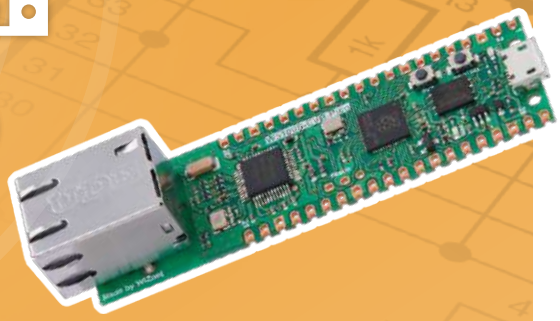
design > share > earn

594
JULI/AUGUST 2023
ELEKTORMAGAZINE.DE
* SEIT 1961 *

RP2040-basierte

Telefon-Fernbedienung

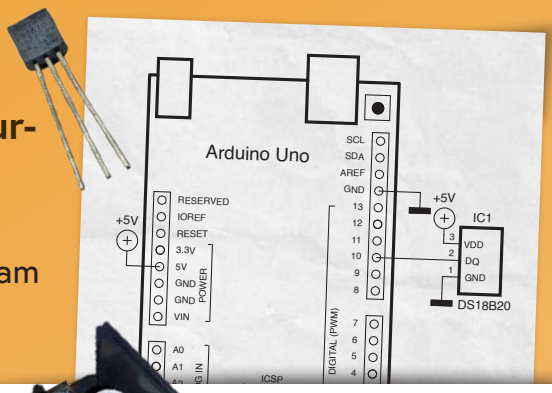




IM FOKUS

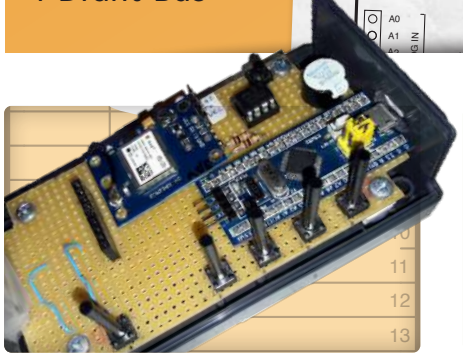
IoT und Sensoren

Temperatur-Sensor DS18B20
Messungen am 1-Draht-Bus

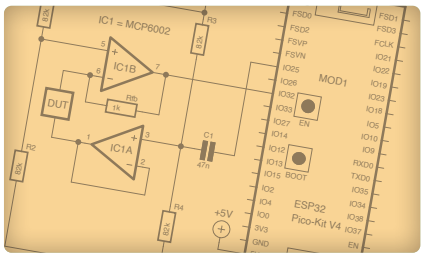


matter

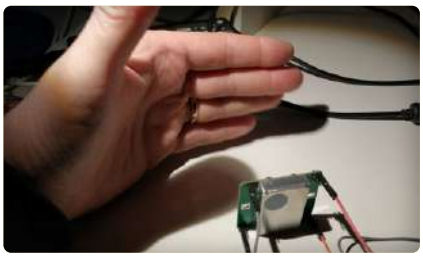
Der Matter-Standard
Rettet Matter das Smart Home?



GPS-gestützter Tempowächter
Nie mehr rasen, nie mehr Knöllchen!
S. 11



ESP32-basierter Impedanz-Analysator
Einfach, klein, preiswert!
S. 30



Doppler-Bewegungssensor HB100
Ein ungewöhnlicher Sensor
S. 90

4 198630 314904 30008

MACH DICH BEREIT FÜR

einen epischen Projektsommer!

Aufruf an alle Elektronik-Enthusiasten und Elektronik-Bastler! Das Elektor Circuit Special 2023 Magazin erscheint im August und ist vollgepackt mit vielen bemerkenswerten Projekten. Macht euch bereit, tief in die spannende Welt der Elektronik einzutauchen und **EURE KREATIVITÄT ZU ENTFESSELN!**

Natürlich erhalten alle unsere Abonnenten diese Sonderausgabe automatisch zugeschickt, zusätzlich wird sie am Kiosk erhältlich sein.

www.elektormagazine.de/circuit-special

elektor
design › share › earn

IMPRESSUM

54. Jahrgang, Nr. 594
Juli/August 2023
ISSN 0932-5468

Das Elektor Magazin wird 8 mal im Jahr herausgegeben von
Elektor Verlag GmbH
Lukasstraße 1, 52070 Aachen (Deutschland)
Tel. +49 (0)241 95509190

www.elektor.de | www.elektormagazine.de

Für alle Ihre Fragen
service@elektor.de

Mitglied werden
www.elektormagazine.de/abo

Anzeigen
Büsra Kas
Tel. +49 (0)241 95509178
busra.kas@elektor.com
www.elektormagazine.de/mediadaten

Druck
Senefelder Misset, Mercuriusstraat 35
7006 RK Doetinchem (Niederlande)

Distribution
IPS Pressevertrieb GmbH, Carl-Zeiss-Straße 5
53340 Meckenheim (Deutschland)
Tel. +49 (0)2225 88010

EDITORIAL

Jens Nickel

Chefredakteur ElektorMag

KI-Horror, KI-Spaß

Was ist das Gegenteil von Marlon Brando? Wissen Sie nicht? Dann schauen Sie sich einmal das (gruselige) YouTube-Video an, das unsere Autorin Ilse Joostens in ihrem Beitrag auf Seite 46 verlinkt. In der Kolumne geht es diesmal um die Künstliche Intelligenz sowie die zahllosen Tools, welche Bilder, Texte und Programmcode generieren. Übrigens erschafft AI inzwischen ganze Videos, und auch hier ist das Ergebnis erstaunlich, manchmal lustig, aber oft auch erschreckend. Man mag sich schon fragen, warum Künstliche Intelligenz sehr oft zur Erzeugung von Horror und apokalyptischen Bildern neigt - auch in unschuldiger Pizza-Werbung im Stile der 90er Jahre (https://youtu.be/MpvEXrhnoyw) ;-).

Noch sind wir natürliche Intelligenzen allerdings Herr der Lage. Und so lange das so ist, sollten wir das Beste aus der KI herausholen. Gerade Programmierern und Elektronikern bieten sich hier etliche Möglichkeiten, die auch wir bei Elektor gerade erst ausloten. Ich bin mir ziemlich sicher, dass Sie bald KI-gestützt einen passenden Artikel aus unserem riesigen Zeitschriftenarchiv heraussuchen können. Manuell hätte es jedenfalls unzählige Jahre gedauert, den zahllosen Querverbindungen zwischen den Projekten und Grundlagen-Artikeln nachzugehen und das Ganze in einen gesamtheitlichen Wissenschatz zu überführen (wenn uns dafür überhaupt ein praktikables System eingefallen wäre).

Auch mit der automatischen Generierung von Programmcode werden wir uns in den kommenden Ausgaben beschäftigen. Ich habe Freunde, die inzwischen wie selbstverständlich ChatGPT anwerfen, um sich Batchdateien für die Bearbeitung von Audiofiles schreiben zu lassen. Ich bin gespannt, was hier speziell für Elektroniker noch zu erwarten ist. Vielleicht haben Sie schon eigene Experimente und gute oder schlechte Erfahrungen gemacht?
Schreiben Sie mir unter redaktion@elektor.de!

Veröffentlichen Sie bei Elektor!

Ihr Fachwissen über Elektronik ist willkommen: Schicken Sie uns Ihr Video, Ihren Artikelvorschlag oder Ihre Idee für ein Buch! Wir haben unseren Leitfaden für Autoren und Ersteller von Inhalten aktualisiert. Alle Einzelheiten finden Sie unter:

www.elektormagazine.com/submissions

Elektor-Labs: Ideen und Projekte

Die Plattform Elektor Labs ist offen für jeden. Hier können Sie Ideen und Projekte zum Thema Elektronik veröffentlichen, technische Probleme diskutieren und mit anderen zusammenarbeiten.

www.elektormagazine.de/labs

Drehscheibentelefon als Fernbedienung

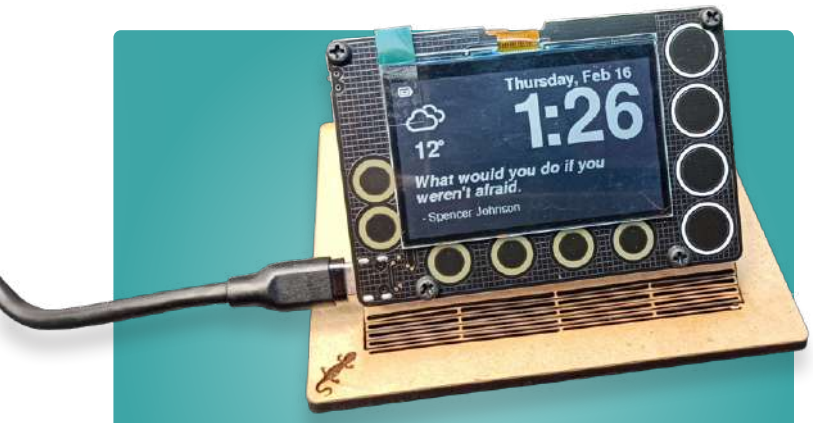
Licht einschalten, wähle die 1, Kaffeemaschine einschalten, wähle die 2

6

Rubriken

Hintergrund

Industry



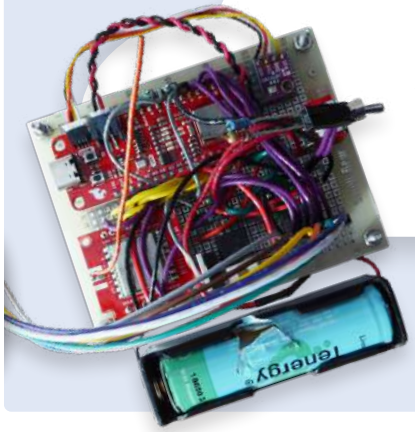
Bauen Sie ein cooles IoT-Display

Mit dem Phambili Newt

88

LoRa-Wetterstation mit niedriger Stromaufnahme

Bauen Sie Ihre entfernte Wetterstation selbst!



116

Projekte

Vorschau

Elektor Circuit Special 2023
Ganz in Tradition der „Halbleiterhefte“ ist die nächste Ausgabe besonders dick und prall gefüllt mit mehr als 50 Selbstbauprojekten, Retronik-Schaltungen, Tipps und Tricks für Elektroniker und vielem mehr.

Aus dem Inhalt:

- Aktiver Gleichrichter
- Billiges Frequenznormal
- Einfacher Dynamikkompressor
- Winzige Solarversorgung
- THD-Generator
- Programmierbarer Video-DAC
- Große RGB-Ziffer
- ChatGPT und Arduino
- Solarbetriebene Weihnachts-UKW-Radiokugel
- Winziger DCF-Simulator

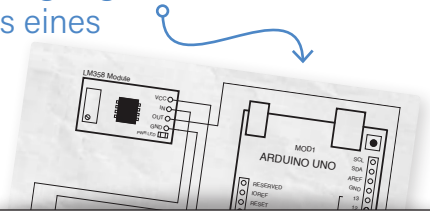
Und vieles mehr!

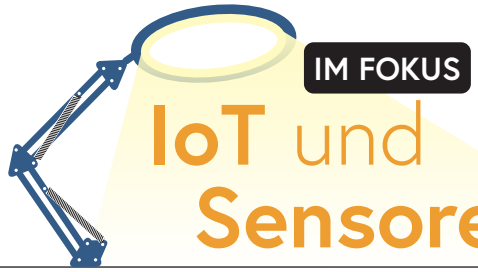
Elektor Circuit Special 2023 erscheint am 9. August 2023.
Änderungen vorbehalten!

Der Doppler-Bewegungssensor HB100

Theorie und Praxis eines ungewöhnlichen Sensors

90





Drehscheibentelefon **als Fernbedienung**

Von Clemens Valens (Elektor)

Auch wenn Licht und Geräte durch ein Hausautomationssystem gesteuert werden, möchte man doch eine übergeordnete Steuerung haben, um eine Lampe, einen Ventilator oder ein anderes Gerät ein- oder auszuschalten. Das in diesem Artikel vorgestellte modifizierte und dennoch dekorative Drehscheibentelefon ermöglicht dies, indem es die Nummer des gewünschten Geräts wählt.

In diesem Projekt verwandeln wir ein altes analoges Telefon mit Wählscheibe, dem so genannten Nummernschalter, in eine Fernbedienung und Alarmanlage in einem Hausautomationssystem. Außer als dekorative Fernbedienung kann das modifizierte Telefon auch als Requisite zum Beispiel in einem Escape-Game verwendet werden. Ich bin sicher, dass Ihnen noch viele andere Anwendungen einfallen werden.

Entwurf mit RP2040

Das Gehirn für dieses Projekt ist ein RP2040-Mikrocontroller auf einem W5100S-EVB-Pico-Board von WIZnet (**Bild 1**). Dabei handelt es sich im Grunde um ein Raspberry-Pi-Pico-Board, dem ein Ethernet-Anschluss und ein „Internet-Chip" W5100S hinzugefügt wurde. Das Board ist pinkompatibel mit dem beliebten Pico-Board, der einzige Unterschied liegt darin, dass einige der Pins (GPIO16...GPIO21) für die Kommunikation mit dem W5100S reserviert sind.

Verdrahtet vs. drahtlos

Auch wenn drahtlose Netzwerke heutzutage Standard sind, gibt es immer noch viele Anwendungen für kabelgebundenes Ethernet. Ein Vorteil ist die Möglichkeit, die angeschlossenen Knoten mit Strom zu versorgen. Diese Möglichkeit wird in diesem Projekt genutzt. Da Drehscheibentelefone prinzipiell mit einem Kabel zu einer Steckdose ausgestattet waren, wäre es auch seltsam, eines ohne zu haben. Daher ist das Board W5100S-EVB-Pico eine vertretbare Wahl für diese Anwendung.

Fernsteuerung über MQTT

Der Home Automation Controller (HAC), der über das Drehscheibentelefon gesteuert wird, ist *Home Assistant* (HA), eine beliebte Automatisierungsplattform, die täglich an Bedeutung gewinnt. Da die Fernbedienung jedoch über MQTT kommuniziert, kann sie problemlos auch in andere Hausautomatisierungssysteme integriert werden.

Anschließen der MCU an das Telefon

Das von mir verwendete Telefon, ein klassisches französisches Modell (S63) mit Impulswahl, erzeugt Impulse mit einer Frequenz von 10 Hz. Der Wählmechanismus des alten Telefons (**Bild 2**) kann als zwei Schalter betrachtet werden: Einer zeigt an, dass der Wählvorgang läuft (besetzt/leer), während der andere je nach gewählter Ziffer eine bestimmte Anzahl von Impulsen auslöst. Eine „1" erzeugt einen Impuls, eine „9" neun Impulse und eine „0" zehn Impulse.

Diesen Mechanismus an die W5100S-EVB-Pico-Platine anzuschließen ist einfach; ihn ohne Modifikation des Telefons anzuschließen erfordert allerdings etwas mehr Überlegung. Ich wollte das Telefon so nah wie möglich an seinem ursprünglichen Zustand belassen. Durch die Wahl der richtigen Anschlusspunkte im Telefon und die Anpassung der erforderlichen Pull-up-Widerstände kann dies jedoch erreicht werden (**Bild 3**).

Der Handapparat liegt auf einem normalerweise offenen Schalter (NO), der durch Abheben des Handapparats geschlossen wird. Dadurch ändert sich der Stromkreis des Telefons und damit auch die Impedanz einiger Anschlusspunkte des Wählmechanismus. Glücklicherweise lässt sich dies durch die Wahl relativ niedriger Werte für einige der Pull-up-Widerstände beheben.

Hochspannungs-Inverter für die Glocke

Die Glocke des Telefons (**Bild 4**) benötigt zum Läuten eine relativ hohe Wechselspannung, mindestens 35 V, wie ich festgestellt habe. Deshalb habe ich einen einfachen Spannungsinverter mit geringem Nennstrom gebaut, der von der MCU gesteuert wird. Die MCU erzeugt zwei 50-Hz-Rechteckwellen mit einem Phasenunterschied von 180°, die die Sekundärseite eines kleinen 230-V-Netztrafos (2 × 9 V, 1 VA) ansteuern. Die Glocke ist an die Primärseite des Transformators angeschlossen.

Bild 2. Der Nummernschalter des S63-Telefons erzeugt Impulse mit einer Frequenz von 10 Hz.

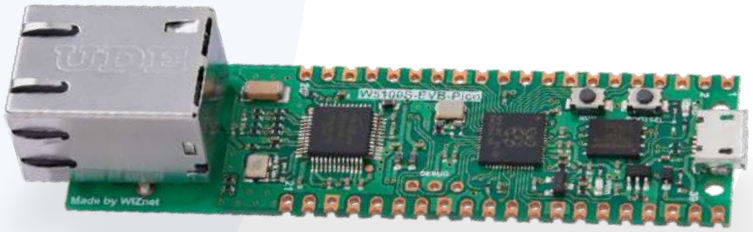


Bild 1. Das Board W5100S-EVB-Pico von WIZnet verfügt über einen RP2040-Mikrocontroller, der mit einem Internet-over-Ethernet-IC W5100S verbunden ist. Das Board hat die gleiche Pinbelegung wie ein Raspberry-Pi-Pico-Board, jedoch sind GPIO16...GPIO21 mit dem W5100S-Chip verbunden. (Quelle: WIZnet)

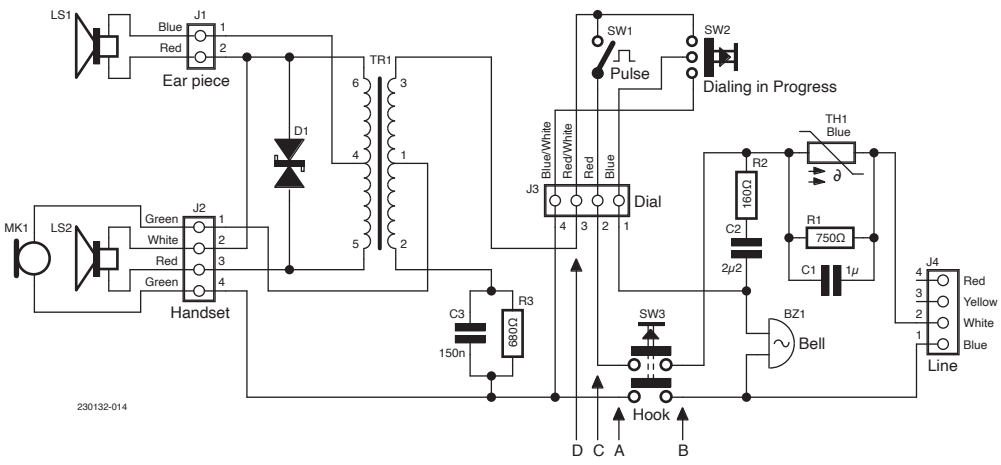


Bild 3. Ohne das Telefon verändern zu müssen, wird die Mikrocontroller-Platine an den Punkten A (GND), B (abgehobener Handapparat), C (Wählimpulse) und D (laufender Wählvorgang) angeschlossen.

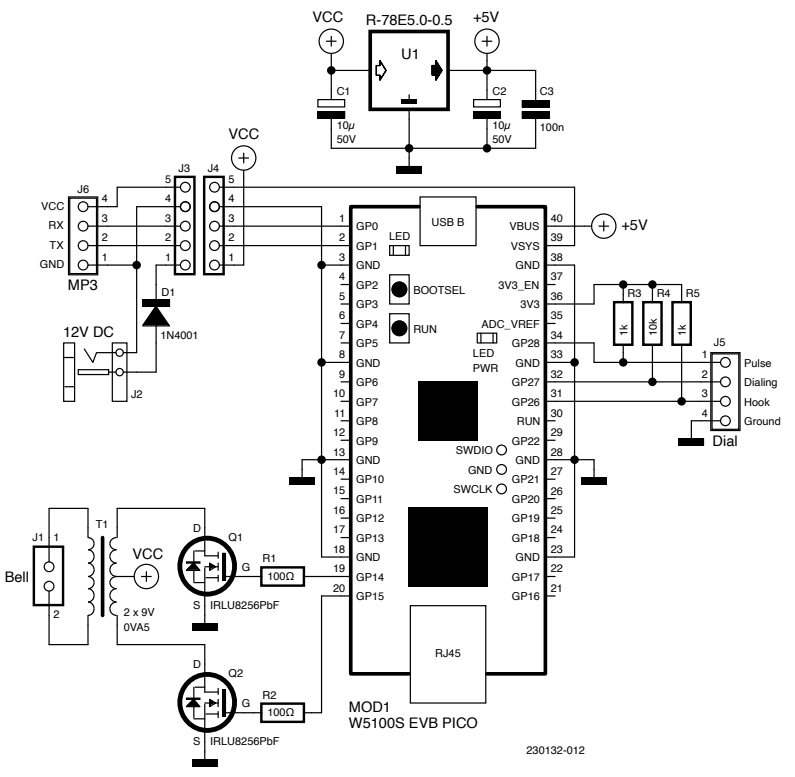
Bild 4. Der Glockenmechanismus des alten Telefons.

Der kleine Trafo reicht aus, um eine anständige Lautstärke der Glocke zu erzielen. Der vollständige Schaltplan der Telefon-zu-Mikrocontroller-Schnittstelle ist in **Bild 5** dargestellt.

MP3-Spieler

Ein MP3-Player-Modul wurde hinzugefügt, um Audiodateien über die Hörkapsel des Handapparats abzuspielen. Das MP3-Modul kommuniziert mit der MCU über eine serielle Schnittstelle mit 9.600 Baud. Der HAC kann MQTT-Nachrichten an das Telefon senden, um die abzuspielende MP3-Datei auszuwählen, oder die MCU kann sie selbst steuern. Auf diese Weise lassen sich Warteschleifenmusiken oder aufgezeichnete Nachrichten abspielen, analoge Telefonleitungstöne wiedergeben oder eine sprechende Uhr realisieren. Die Audio- und Musikdateien werden auf einer microSD-Karte gespeichert. Aus Platzgründen ist der MP3-Player zusammen mit dem Stromanschluss auf einer zweiten (Lochraster-) Platine untergebracht. Die beiden Platinen sind über J3 und J4 miteinander verbunden (**Bild 6**). Die Hörkapsel im Handapparat ist direkt mit dem Ausgang des MP3-Players verbunden.

Bild 5. Für den Anschluss des Mikrocontrollers an das Telefon werden nur wenige Teile benötigt. Beachten Sie, dass die Pull-up-Widerstände für den Nummernschalter (R3, R4 und R5) verschieden sind.



Stromversorgung

Die Platine W5100S-EVB-Pico benötigt eine 5-V-Gleichspannungsversorgung, der Hochspannungsinverter für die Glocke mindestens 10 V_{DC}. Für die Stromversorgung des Telefons werden die beiden freien Adernpaare eines normalen Ethernet-Kabels verwendet (**Bild 7**). Mein fünf Meter langes Kabel führte zu einem spürbaren Spannungsabfall, selbst als für jeden Stromversorgungsanschluss zwei Adern parallel geschaltet waren. Daher habe ich das Telefon über ein 12-V-AC/DC-Steckernetzteil versorgt. Im Inneren des Telefons erzeugt ein kleiner DC/DC-Wandler dann die 5 V für die Controllerplatine.

Unterstützende Software

Das W5100S-EVB-Pico-Board wird von Bibliotheken unterstützt, die für die offizielle MCU-Entwicklungsumgebung für den Raspberry Pi RP2040 vorgesehen sind. Ich mag diese Umgebung nicht besonders, da ich sie mit all dem Cmake-Zeugs für zu komplex und unübersichtlich halte. Glücklicherweise gibt es auch ein Arduino-Boards-Paket für den RP2040, und so habe ich die WIZnet-Bibliotheken auf diese IDE portiert. Das macht die Sache viel einfacher und, wie ich finde, auch zugänglicher und leichter zu teilen. Das komplette Anwendungsprogramm besteht nun aus einem Arduino-Sketch und einer einfach zu installierenden Bibliothek [1].

MQTT

Ich habe das Programm für die Fernsteuerung auf der Grundlage des von WIZnet bereitgestellten MQTT-Beispiels entwickelt. So konnte ich schnell loslegen. Das einzige Problem, auf das ich stieß, war ein fehlender und undokumentierter Aufruf des `MilliTimer_Handler` (siehe in der Datei *mqtt_interface.h*). Nachdem ich ihn zu meinem Millisekunden-Timer-Callback hinzugefügt hatte, funktionierte MQTT einwandfrei.

mDNS

Da meine Home-Assistant-Installation auf DHCP angewiesen ist, um sich mit dem Netzwerk zu verbinden, kann ihr jederzeit eine neue IP-Adresse zugewiesen werden. Um seine Peripheriegeräte zu finden und mit ihnen zu kommunizieren, verwendet Home-Assistant Multicast-DNS, auch bekannt als mDNS. Da ich nicht jedes Mal, wenn sich die Adresse des HAC ändert, eine temporäre IP-Adresse in meinem Programm fest codieren und neu kompilieren wollte, habe ich die mDNS-Unterstützung hinzugefügt. Dazu habe ich die WIZnet-DNS-Bibliothek angepassen können, da mDNS dem normalen DNS recht ähnlich ist. Jetzt stellt das Programm eine mDNS-Anfrage, um die IP-Adresse des HAC zu erhalten, bevor es versucht, eine Verbindung zu ihm herzustellen. Dies macht das System viel flexibler und zuverlässiger.

Erkennung und Anwahl von Handgeräten

Das Auslesen der Schalter des Wählmechanismus erfordert eine Entprellung in der Software. Danach wird das Zählen der Impulse zu einer trivialen Aufgabe. Das Wählen mit aufgelegtem Handapparat erzeugt einstellige Nachrichten; das Wählen mit abgehobenem Handapparat erzeugt eine mehrstellige Nummer, die nach Auflegen des Handapparats gesendet wird. Die Ziffer oder Nummer wird als Nutzlast eines MQTT-Pakets gesendet. Beachten Sie, dass Ziffern „minus eins“ gesendet werden, damit die zehn Werte in ein Zeichen passen. Die 1 wird also zur 0, die 9 als 8 und die 0 als 9 gesendet.

Textnachrichten

Wenn der Handapparat abgehoben wird, werden beim Wählen neben einer mehrstelligen Zahl auch Buchstaben zum Verfassen von Textnachrichten gesendet. Die Buchstabenverteilung ist auf dem verwendeten französischen Telefon S63 aufgedruckt (**Bild 8**) und entspricht in etwa der auf Mobiltelefonen aus den „Nullerjahren“ zum Verfassen von Textnachrichten, mit dem Unterschied, dass es keine Anzeige gibt.

Bei deutschen IWV-Fernsprechapparaten vom W48 bis zum FeTAp 89 gab es einen solchen Aufdruck nicht, hier waren auf der Papiereinlage im Zentrum der Wählscheibe nochmals die Ziffern aufgedruckt, damit man stets überprüfen konnte, ob alle Ziffern an ihrem behördlich zugewiesenen Platz waren, oder die Nummern von Polizei und Feuerwehr, falls man nicht in der Lage sein sollte, eine dreistellige Ziffernfolge im Kopf zu behalten. Falls Sie also Textnachrichten mit dem Telefon verschicken wollen, empfiehlt es sich, ein neues Inlay mit den korrespondierenden Buchstaben zu entwerfen.

Die 26 Buchstaben des Alphabets sind auf die Ziffern 2 bis 9 verteilt, drei Zeichen pro Ziffer, außer bei der 6, die nur zwei Buchstaben hat (m und n). Die Ziffer 0 hat ebenfalls zwei Buchstaben (o und q), die Ziffer 1 hat keine. Aus irgendeinem Grund gibt es kein z, und so habe ich es zur Ziffer 0 hinzugefügt.

Um einen Buchstaben zu wählen, muss die entsprechende Ziffer bis zu dreimal gewählt werden. Als Beispiel: a ist 2, b ist 22 und c ist 222. Um aaa zu schreiben, ist die Wählsequenz 2-2-2, für abc ist sie 2-22-222. Der Bindestrich steht hier für eine Pause von mindestens einer Sekunde, wenn bei einer kürzeren Pause zwischen zwei identischen Ziffern wird der nächste Buchstabe gewählt, der der Ziffer zugeordnet ist.

Wenn der Handapparat wieder aufgelegt wird, wird die verfasste Nachricht zusammen mit der mehrstelligen Nummer als Nutzlast eines MQTT-Pakets an den HAC gesendet. Der HAC kann die Nachricht dann beispielsweise an einen SMS-Dienst weiterleiten oder sie auf seinem Dashboard anzeigen.

Bild 8. Dank der Zeichentabelle rund um den Nummernschalter ist das Schreiben von Texten mit diesem alten französischen Telefon einfach. Für Beispiel: „elektor“ wird als 33-555-33-55-8-0-77 gewählt, wobei der Bindestrich für eine Pause von einer Sekunde steht.

◄

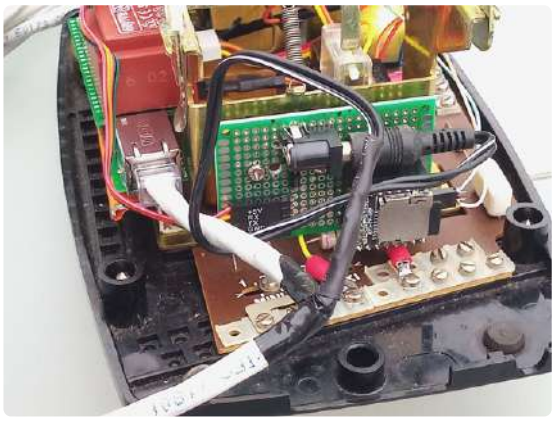


Bild 7. „Power over Ethernet“ auf die ineffiziente Art. Es funktioniert, solange die Eingangsspannung hoch genug ist, um den durch das Kabel verursachten Spannungsabfall zu überwinden.

◄

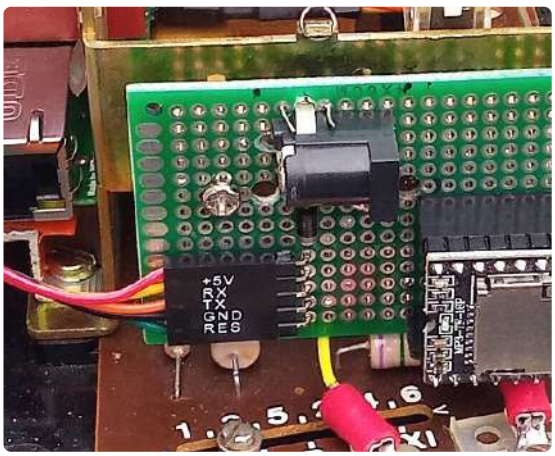


Bild 6. Das MP3-Player-Modul und der Stromversorgungsanschluss teilen sich eine eigene kleine Platine.

◄

Was Sie vielleicht ändern möchten

Die in diesem Artikel vorgestellte Elektronik wurde für das französisches Telefon S63 entwickelt und so gebaut, dass sie in dieses Telefon passt (**Bild 9**). Möglicherweise müssen Sie einige Teile für einen anderen Fernsprechapparat mechanisch ändern.

Anstelle von Ethernet können Sie auch WLAN oder eine andere drahtlose Kommunikationsmethode bevorzugen. Die Software lässt sich leicht an andere physikalische Layer anpassen, da das Anwendungsprogramm eine Standard-Netzwerktreiber-API verwendet. Der TCP/IP-Stack wird vollständig vom W5100S-IC übernommen

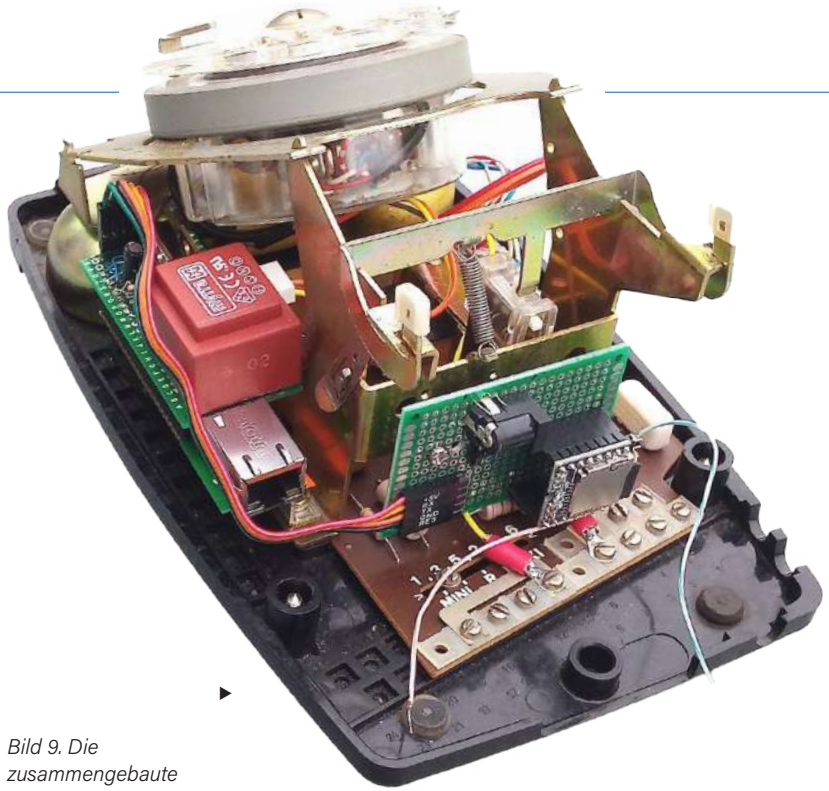


Bild 9. Die zusammengebaute Telefon-Fernbedienung. Links ist der Transformator des Hochspannungs-Inverters für die Glocke zu sehen, der auf die Platine W5100S-EVB-Pico aufgesteckt ist. Ein Kabel verbindet diese Baugruppe mit dem MP3-Player-Modul und dem Stromanschluss auf der rechten Seite. Die Anschlüsse für den Nummernschalter sind nicht sichtbar, da sie sich auf der anderen Seite des Telefons befinden.

und kann daher durch ein anderes Kommunikationsmodul ersetzt werden.

Das Hinzufügen eines guten Akkus würde eine kabellose Fernsteuerung ermöglichen (die von Zeit zu Zeit aufgeladen werden müsste). Wenn die ursprüngliche Schaltung des Telefons entfernt wird, dürfte genug Platz für einen solchen Akku sein.

Der Hochspannungs-Wechselrichter für die Glocke kann auf verschiedene Weise gebaut werden. Ich habe einen Transformator verwendet, weil ich gerade einen zur Hand hatte, aber auch ein geeignetes, kostengünstiges Konvertermodul aus dem Internet kann die Aufgabe erfüllen.

VoIP

Derzeit wird das Mikrofon des Telefons nicht benutzt, aber man könnte es zum Beispiel für einen Voice-over-Internet- oder VoIP-Dienst verwenden. Der RP2040 unterstützt I^2S, so dass Standard-Mikrofon-Interface-ICs dafür verwendet werden könnten.

Die Fernsteuerungsanwendung besteht aus einem Arduino-Sketch und einer einfach zu installierenden Arduino-Bibliothek, die auf dem offiziellen WIZnet-Repository [1] zu finden ist.

RG – 230132-02

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Schicken Sie eine E-Mail an den Autor unter clemens.valens@elektor.com oder kontaktieren Sie Elektor unter redaktion@elektor.de.

WEBLINK

[1] Dieses Projekt auf Github: https://github.com/polyvalens/rotary_dial_remote

Fernbedienungsfunktionen

Das modifizierte Drehscheibentelefon hat die folgenden Funktionen:

- Wenn der Handapparat aufgelegt ist und eine Nummer gewählt wird, wird das entsprechende Gerät ein- oder ausgeschaltet, je nachdem, in welchem Zustand es sich gerade befindet.
- Wird der Handapparat abgehoben, kann mit der Wählscheibe (mühsam) eine Textnachricht verfasst werden, ähnlich wie bei Mobiltelefonen von vor etwa 20 Jahren. Legt man den Handapparat wieder auf, wird die Nachricht verschickt.
- Die Glocke des Telefons steht dem Home Automation Controller (HAC) als Alarm zur Verfügung und kann zum Beispiel auf die Türklingel geroutet werden oder als Küchentimer oder (unangenehmer) Weckruf fungieren.
- Die Hörkapsel des Handapparats wird mit einem MP3-Player verbunden, um vorgespeicherte Nachrichten oder Musik abzuspielen. Auf diese Weise lässt sich zum Beispiel eine sprechende Uhr realisieren, wie sie im vorigen Jahrhundert üblich war.

Die Kommunikation zwischen dem Telefon und dem HAC basiert auf MQTT, während mDNS automatisch eine Verbindung zwischen den beiden herstellt.

Passende Produkte

- **RP2040-basiertes Evaluation-Board W5100S-EVB-Pico von WIZnet** SKU 19971: www.elektor.de/19971
- **Ethernet-HAT für Raspberry Pi Pico von WIZnet** SKU19970: www.elektor.de/19970
- **C. Valens, *Mastering Microcontrollers helped by Arduino*** (3. Auflage) Buch, Paperback, SKU 17967: www.elektor.de/17967

GPS-gestützte Geschwindigkeitsüberwachung

Nie mehr rasen, nie mehr Knöllchen!

Von Olivier Croiset (Frankreich)

Es gibt viele Projekte im Internet, die ein GPS-Modul mit einem Mikrocontroller verwenden, um GPS-Koordinaten auf einer Karte darstellen. Aber was kann man sonst mit den Koordinaten anfangen? Ich wollte etwas Nützlicheres entwickeln und kam auf die Idee einer GPS-basierten Tempoüberwachung für ein Auto oder ein Boot. So kann man fahren, ohne den Tacho im Auge zu haben und böse Überraschungen in Form von Knöllchen vermeiden.

Bei dem hier vorgestellten Gerät handelt es sich nicht um einen Geschwindigkeitsbegrenzer, der den Umbau des Fahrzeugs (und eine Genehmigung) erfordern würde, sondern um einen Alarm, der signalisiert, wenn eine vorgewählte Geschwindigkeit erreicht und überschritten wird. Damit Sie Ihre Augen nicht von der Straße abwenden müssen, gibt der Summer zwei lange Töne von sich, wenn die gewählte Geschwindigkeit überschritten wird, während zwei kurze Töne anzeigen, dass das Fahrzeug auf eine Geschwindigkeit unterhalb des Limits abgebremst hat.

Vier Voreinstellungen

Der Geschwindigkeitsmonitor verfügt über vier programmierbare Voreinstellungen für üblichen Geschwindigkeitsbegrenzungen, was für die meisten Fahrten und Tempolimits ausreichend sein sollten. Bei nur vier angezeigten Grenzwerten auf dem Display sollte der Fahrer während der Fahrt in der Lage sein, schnell eine Voreinstellung zu wählen, ohne auf das Gerät zu schauen. Ich wollte keinen Touchscreen verwenden; der Fahrer soll die Tasten „ertasten“ können. Aus ergonomischen Gründen befinden sich diese Tasten oberhalb der vier auf dem Display angezeigten Geschwindigkeitsbegrenzungen.

Versuche mit 32 Bit

Die ersten Versuche mit einem 8-Bit-Mikrocontroller ATmega328 in Verbindung mit einem GPS-Modul haben mir schnell gezeigt, dass trotz der vielen Qualitäten des Controllers der 32 KB große Flash-Speicher für eine anspruchsvollere Anwendung nicht ausreicht, insbesondere wenn die Daten grafisch auf einem TFT-Bildschirm dargestellt werden sollen.
Also beschloss ich, die nächste Stufe zu erklimmen und einen 32-Bit-Mikrocontroller STM32 auszuprobieren. Die für Hobbyisten wie mich am besten geeignete Form ist das BluePill-Board mit dem Controller STM32F103C8. Diese kleine Platine erleichtert die Verwendung dieses

Mikrocontrollers erheblich. Es kann einfach über den USB-Anschluss mit der Arduino-IDE programmiert werden (vorausgesetzt, das Board wurde zuvor mit einem geeigneten Bootloader programmiert), es ist kompakt und mit einem Preis von nur wenigen Euro ziemlich billig.

Größe des Flash-Speichers

Dieses Projekt ist mein erster Versuch, den STM32F103C8 zu verwenden. Beachten Sie, dass der C**8** offiziell mit 64 KB Flash-Speicher ausgestattet ist, während die C**B**-Version 128 KB hat. Aber manchmal wird auch der C8 mit 128 KB geliefert (gefälschte Teile?). Dies ermöglicht es denjenigen, die es reizt, weiter zu lernen, indem sie andere Peripheriebausteine wie eine SD-Karte, ein SIM-Modul, Datenübertragung über eine Funkverbindung und so weiter hinzufügen.

Das GPS-Modul

Das in diesem Projekt verwendete GPS ist ein NEO-6-Modul von u-blox. Seine Verwendung ist einfach, da die Bibliothek TinyGPS+ [2] die gesamte Arbeit der Dekodierung des vom GPS-Modul ausgegebenen Datenstroms im NMEA-0183-Format übernimmt. Die wichtigsten Parameter, die wir auf diese Weise erhalten, sind:

- UTC-Datum und -Zeit
- Längen- und Breitengrad, in Dezimalgraden
- Geschwindigkeit
- Ortsmissweisung
- Höhe
- Die Anzahl der sichtbaren Satelliten
- Horizontalgenauigkeit (HDOP)

Die Horizontalgenauigkeit HDOP [2] hängt von der Position der Satelliten in Reichweite des GPS-Empfängers ab. Je niedriger dieser Wert ist (nahe bei 1), desto besser ist die Genauigkeit der Koordinaten. Ein HDOP-Wert von 10 bedeutet, dass die Koordinaten nicht sehr genau oder sogar ungültig sind. Der Speed-Monitor verwendet diesen Parameter nicht.

Das TFT-Display

Als Display habe ich ein handelsübliches 2,2″-TFT-Modul mit ILI9341-Treiberchip und SPI-Schnittstelle verwendet. Es besitzt eine Auflösung von 320 × 240 Pixel, was für die Daten, die wir anzeigen wollen, ausreichend ist (vier Geschwindigkeitsbegrenzungen mit einer kurzen Meldung, ob die Geschwindigkeitsgrenze erreicht wurde oder nicht). Der Geschwindigkeitsmonitor verfügt im normalen Gebrauch über zwei Hauptanzeigemodi (Displays). Der aktuelle Anzeigemodus wird im EEPROM gespeichert und beim erneuten Einschalten des Systems abgerufen. Die Voreinstellung für die Geschwindigkeitsgrenzen wird ebenfalls im EEPROM gespeichert. Nur sechs Bytes

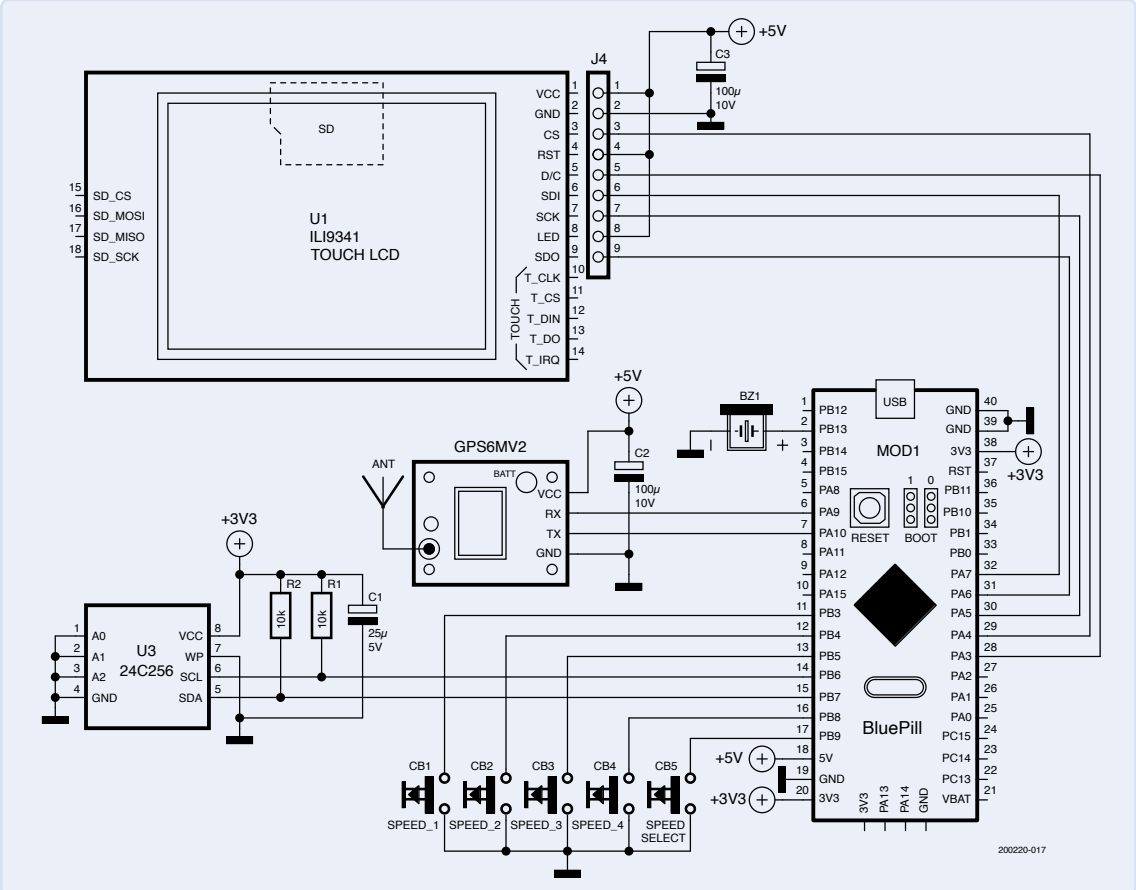


Bild 1. Der Geschwindigkeitsmonitor vereint leicht erhältliche Module mit wenigen elektronischen Bauteilen.

dieses EEPROMs werden verwendet: vier Bytes für die Geschwindigkeitsgrenzen, ein Byte für das zuletzt gewählte Display und ein Byte für die zuletzt gewählte Geschwindigkeitsbegrenzung.

Die Schaltung

Der Schaltplan des Geschwindigkeitsmonitors ist in **Bild 1** dargestellt. Das BluePill-Modul unten rechts ist das Herzstück der Schaltung.

Die Stromversorgung der Elektronik erfolgt über den USB-Anschluss der BluePill. Neuere Fahrzeuge sind häufig mit einem USB-Anschluss ausgestattet; andernfalls ist ein kleiner 12-V-USB-Adapter erforderlich (oder man verwendet eine Powerbank). Das BluePill-Modul wird mit 5 V versorgt und wandelt diese Spannung in 3,3 V für internen Gebrauch sowie extern zur Versorgung des EEPROMs um. TFT-Display und GPS-Modul werden mit 5 V versorgt. Die Stromaufnahme der Schaltung beträgt etwa 200 mA. Das Display wird über einen SPI-Bus angesteuert, das GPS-Modul über eine serielle Verbindung, und das EEPROM ist mit dem I²C-Bus verbunden (mit zwei Pull-up-Widerständen). Für meinen Prototyp habe ich das 32 KB große EEPROM 24C256 verwendet, aber auch ein 128 Byte kleiner 24C01 wäre völlig ausreichend. Zwölf GPIO-Ports bleiben frei. Es ist also möglich, weitere Peripheriebausteine, Eigenschaften und Funktionen hinzuzufügen, solange das erweiterte Anwendungsprogramm in den Flash-Speicher passt. Die kompilierte Firmware dieses Projekts belegt 64.020 Bytes, also fast den gesamten Flash-Speicher.

Bau des Geschwindigkeitsmonitors

Das wichtigste Kriterium beim Aufbau dieses Projekts ist die Ergonomie. Der Fahrer muss in der Lage sein, die Tasten ohne Probleme und ohne Sichtkontakt zu finden. Die verwendeten Drucktasten haben einen 25 mm langen Hebel - den längsten, den ich finden konnte.

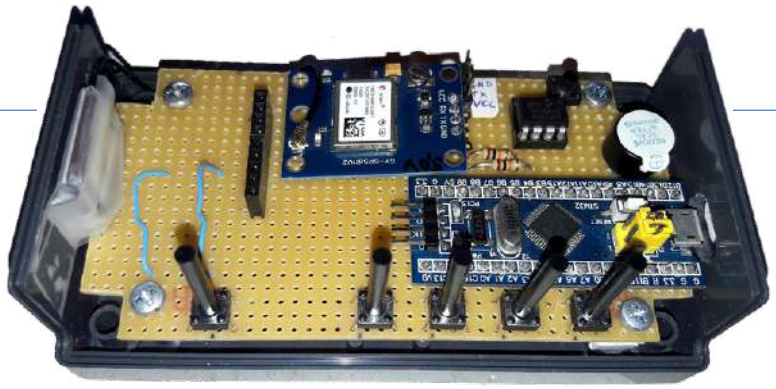
Ich habe diesen Prototyp auf einer Lochrasterplatine aufgebaut (**Bild 2**); eine echte gedruckte Platine würde einen noch kompakteren Aufbau in einem dünneren Gehäuse ermöglichen. Der USB-Anschluss muss zugänglich bleiben, da er nicht nur für die Aktualisierung der Software benötigt, sondern im normalen Gebrauch für die Stromversorgung der Schaltung verwendet wird.

Die GPS-Antenne sollte so weit wie möglich vom TFT-Display-Anschluss entfernt sein (mindestens einige Zentimeter).

Hinweise zum GPS-Modul

Das GPS-Modul verfügt über eine winzige Backup-Zelle, die einen schnellen „hot start" ermöglich. Wenn das GPS einige Tage hintereinander nicht benutzt wird, ist diese Zelle entladen und die Daten im internen RAM sind verloren. Wenn Sie das GPS dann wieder einschalten, müssen Sie möglicherweise einige Minuten warten, um die Daten wiederherzustellen (und die Zelle wieder aufzuladen). Die LED des GPS-Moduls blinkt, wenn es die Satellitendaten erfolgreich entschlüsselt hat. Wenige Augenblicke später wird die Anzahl der sichtbaren Satelliten angezeigt.

Die GPS-Antenne muss die Satelliten „sehen". Der Empfang der Signale darf daher so wenig wie möglich behindert werden. In städtischen Gebieten, zwischen zwei Gebäuden, kann der Empfang schwierig sein. Dennoch funktioniert der Geschwindigkeitsmesser, der in meinem Auto zu Füßen des Beifahrers angebracht ist, einwandfrei.



▲ *Bild 2. Der Prototyp wurde auf einer Lochrasterplatine aufgebaut. Die Abmessungen des Gehäuses betragen 12 × 6,5 × 4 cm.*

Geschwindigkeitsbegrenzungen einstellen

Dies sollte nur geschehen, wenn das Fahrzeug steht! Beim ersten Einschalten des Geschwindigkeitsmonitors sind die Geschwindigkeitsbegrenzungen alle auf den Wert 255 eingestellt. Dies entspricht FF in Hexadezimal, also das, was ein leeres EEPROM enthält.

Drücken Sie im Display 3 (**Bild 6**), das die Satellitendaten zeigt, die dritte Taste, um die Einstellung der Geschwindigkeitsgrenzen zu aktivieren. Verwenden Sie dann die erste und zweite Taste, um den gewünschten Grenzwert einzustellen. Drücken Sie erneut die dritte Taste, um die Geschwindigkeitsbegrenzung zu bestätigen und zur nächsten Taste zu wechseln, und so weiter. Achten Sie

◀ *Bild 3. Display 1 zeigt die Geschwindigkeitsbegrenzungen als runde Verkehrsschilder zusammen mit einer kurzen Meldung.*

◀ *Bild 4. In Display 2 wird die aktuelle Geschwindigkeit als simulierte 7-Segment-Anzeige dargestellt.*

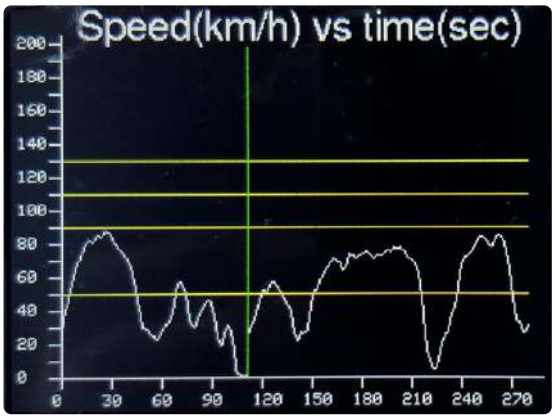


Bild 5. Der Geschwindigkeitsverlauf wird in Display 5 dargestellt. Die vier gelben horizontalen Linien zeigen die vier programmierten Grenzwerte, und die grüne vertikale Cursorlinie zeigt die aktuelle Geschwindigkeit an.

Bild 6. Die GPS-Daten sind in Display 3 verfügbar, von dem aus die Definitionen der Geschwindigkeitsbegrenzungen von Display 4 aufgerufen werden können.

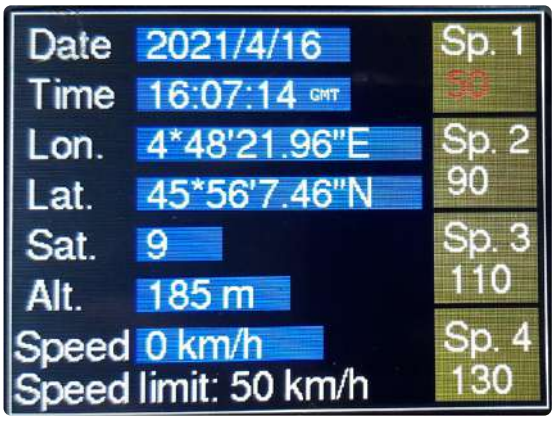


Bild 7. In Display 4 können Sie die Geschwindigkeitsbegrenzungen festlegen. Hier wird gerade der erste Grenzwert „Sp. 1" eingestellt (in rot, wenn Sie genau hinsehen).

darauf, dass die Grenzwerte in einer logischen Reihenfolge stehen, da die Software sie nicht sortiert.

Verwendung des Geschwindigkeitsmonitors

Der Geschwindigkeitsmonitor ist einfach zu bedienen. Im fahrenden Fahrzeug ist darauf zu achten, dass das Gerät und insbesondere das Versorgungskabel nicht irgendwo im Sichtfeld herumbaumeln. Sobald das Gerät an einer geeigneten Stelle platziert ist, wählen Sie den gewünschten Anzeigemodus. Display 1 (**Bild 3**) zeigt runde Verkehrsschilder, Display 2 zeigt die aktuelle Geschwindigkeit als 7-Segment-Ziffern an (**Bild 4**). Wählen Sie dann das gewünschte Tempolimit aus. Der Summer ertönt, wenn Sie zu schnell unterwegs sind.
Ihr Beifahrer kann sich die Geschwindigkeitskurve über einen Zeitraum von 4,5 min (270 s) ansehen, indem er Display 5 auswählt (**Bild 5**). In diesem Display ertönt der Summer nicht. Die horizontalen Linien zeigen die vier Geschwindigkeitsbegrenzungen an, die vertikale Linie die aktuelle Geschwindigkeit. Ein scrollendes Diagramm wäre vielleicht schöner gewesen, erfordert aber für jeden neuen Datenpunkt eine vollständige Aktualisierung, und dies würde zu viel Zeit erfordern. Das Bewegen des Cursors ist viel einfacher und schneller.

Das Display GPS-Daten

Display 3 (**Bild 6**) zeigt die GPS-Daten und ermöglicht es Ihnen, sich auf einer Karte mit Koordinatenmarkierungen zu positionieren. Die Genauigkeit beträgt etwa 30 Meter, was für unser kleines und preisgünstiges Gerät recht gut ist. In diesem Anzeigemodus ertönt der Summer ebenfalls nicht. Display 4 (**Bild 7**) ist Display 3 sehr ähnlich, mit dem Unterschied, dass Sie hier die Geschwindigkeitsbegrenzungen einstellen können.
Beachten Sie, dass das GPS-Datum und die Uhrzeit UTC-Werte sind, also auf dem Meridian von Greenwich bezogen. Wenn Sie sie verwenden möchten, müssen Sie diese Werte in Ihre lokale Zeitzone und die Sommer-/

Tabelle 1: Die verschiedenen Displays und wie man durch sie navigiert.

Display	Taste 1	Taste 2	Taste 3	Taste 4	Taste 5
Display 1 (Runde Verkehrsschilder)	Geschwindigkeitsgrenze 1	Geschwindigkeitsgrenze 2	Geschwindigkeitsgrenze 3	Geschwindigkeitsgrenze 4	zu Display 2 gehen
Display 2 (7-Segment-Anzeige)	Geschwindigkeitsgrenze 1	Geschwindigkeitsgrenze 2	Geschwindigkeitsgrenze 3	Geschwindigkeitsgrenze 4	zu Display 3 gehen
Display 3 (GPS-Daten)	Geschwindigkeitsgrenze 1	Geschwindigkeitsgrenze 2	zu Display 4 gehen	ohne Funktion	zu Display 1 gehen
Display 4 (Grenzen einstellen)	Grenze verringern (–5 km/h)	Grenze erhöhen (+5 km/h)	Bestätigung und weiter zum nächsten Grenzwert, zurück zu Display 3 am Ende	zu Display 5 gehen	ohne Funktion
Display 5 (Geschwindigkeitsverlauf)	ohne Funktion	ohne Funktion	ohne Funktion	zu Display 2 gehen	ohne Funktion

Winterzeit umrechnen. Da der Speed-Monitor keine Uhr ist, gibt es dafür auch kein praktisches Menü.

Die Software

Der Quellcode für das Programm kann von [3] heruntergeladen werden und ist leicht zu modifizieren. Für die Kompilierung des Arduino-Sketches muss die Arduino-IDE mit dem *STM32 Boards Package for Arduino* ausgestattet sein (wir haben v2.3.0 verwendet) [4]. Der größte Teil des Codes ist der grafischen Benutzeroberfläche gewidmet, und der Rest kümmert sich um die einfache Aufgabe, GPS-Daten zu empfangen, die Geschwindigkeit zu extrahieren und sie mit den Grenzwerten zu vergleichen. Wie in einem Arduino-Sketch üblich, kümmert sich die `setup()`-Funktion um die Initialisierung aller Peripheriebausteine. Danach zeigt sie fünf Sekunden lang einen Begrüßungsbildschirm an.

Die Funktion `loop()` beginnt mit dem Lesen der Drucktasten, bevor sie das GPS auf neue Daten überprüft (in der Funktion `dataDecode()`). Je nachdem, welches Display aktiv ist, werden dann die entsprechenden Daten auf das TFT-Display geschrieben. Ist die aktuelle Geschwindigkeit höher als die gewählte Höchstgeschwindigkeit, wird ein Alarm ausgelöst.

Beachten Sie, dass die Debugging- und Statusinformationen über die serielle Schnittstelle (115200,N,8,1) gesendet werden.

Variationen und Add-ons

Hier sind Ideen für ein paar Dinge, die Sie vielleicht hinzufügen oder ändern möchten:

- Verwenden Sie andere Geschwindigkeitseinheiten. Die GPS-Bibliothek ermöglicht andere Geschwindigkeitseinheiten wie Knoten oder Meilen/Stunde (mph). Dazu müssen Sie auch die Displays ändern, also beispielsweise km/h durch kn (Knoten) und die Schrittweite der Grenzwertinkremente (in der Funktion `SpeedSettings`) ersetzen. Dies kann für ein Boot nützlich sein.

Wenn Ihr BluePill-Modul über 128 KB Flash-Speicher verfügt, können Sie zusätzliche Funktionen hinzufügen, zum Beispiel:

- Aufzeichnung auf SD-Karte. Auf der Rückseite des TFT-Display-Moduls befindet sich ein SD-Kartensteckplatz. Hier können beispielsweise die GPS-Daten für eine spätere Auswertung auf einem PC aufgezeichnet werden.
- Hinzufügen eines Touchscreens für neue Funktionen. Achten Sie in diesem Fall darauf, dass die Fahrsicherheit nicht leidet!
- Einstellen der Ortszeit.

RG — 200220-02

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare? Dann wenden Sie sich bitte per E-Mail an das Elektor-Redaktionsteam unter redaktion@elektor.de.

Passende Produkte

- **OPEN-SMART GPS - Serielles GPS-Modul für Arduino**
  SKU 18733: https://elektor.de/18733
- **2,2"-TFT-Display-Modul mit SPI ILI9341 (240×320)**
  SKU 18419: https://elektor.de/18419
- **Majid Pakdel, Advanced Programming with STM32 Microcontrollers**
  Buch, Paperback (SKU 19520): https://elektor.de/19520
  E-Buch, PDF (SKU 19527): https://elektor.de/19527

WEBLINKS

[1] Bibliothek TinyGPS++: http://arduiniana.org/libraries/tinygpsplus/
[2] Was genau ist HDOP?: https://de.wikipedia.org/wiki/Dilution_of_Precision
[3] Projektdateien bei Elektor Labs: https://elektormagazine.de/labs/save-money-with-this-speed-monitoring-by-gps
[4] STM32-Boards-Paket für Arduino: https://github.com/stm32duino/BoardManagerFiles/raw/main/package_stmicroelectronics_index.json

RGB-Stroboskop mit Arduino

Farbenfrohe Anwendung eines nützlichen Werkzeugs

Von Roel Arits (Niederlande)

Vor Urzeiten, bevor die moderne Mikrocontroller-Elektronik unter der Motorhaube Einzug hielt, wurde die Zündung des Motors mit einem Stroboskop eingestellt. Jetzt wird ein solches Gerät eher selten benötigt – denn in einem nicht allzu antiken Auto wird die Zündung von der ECU (Electronic Control Unit) mit Hilfe zahlreicher Sensoren variabel gesteuert.

Früher war alles nicht unbedingt besser, aber doch anders. In einer nicht einmal so grauen Vergangenheit gab es genug Platz unter der Motorhaube eines Autos, um selbst zu basteln, zu reparieren und einzustellen. Dank moderner computergestützter Konstruktionstechniken ist es möglich, alle Komponenten unter der Motorhaube so kompakt und dicht aneinander zu platzieren, dass selbst der Austausch einer gewöhnlichen Lampe für Laien fast unmöglich wird. Und natürlich fand auch der Computer selbst (in Form einer großen Anzahl miteinander verbundener Mikrocontroller) seinen Platz unter der gleichen Haube. Aber zurück zu - sagen wir - den 70er oder 80er Jahren des letzten Jahrhunderts. Für das einwandfreie Funktionieren eines benzinbetriebenen Verbrennungsmotors müssen die Zündkerzen jedes Zylinders genau im richtigen Moment einen Funken erzeugen. Dafür sorgte der Verteiler (ein mechanisches Gerät, das bei bestimmten Automarken furchtbar empfindlich auf Feuchtigkeit reagierte), der genau mit dem Motor synchronisiert werden musste. Und hier kommt unser Stroboskop ins Spiel.

In einem solchen Verteiler gab es genauso viele Unterbrecherkontakte, wie der Motor Zylinder besaß. Jedes Mal, wenn ein solcher Unterbrecherkontakt geöffnet wurde (daher der Name), entlud sich die in der Zündspule gespeicherte elektrische Energie über die Zündkerze in den Brennraum in Form eines Funkens, der das Benzin-Luft-Gemisch entzündete. Und dies musste genau zu einem Zeitpunkt geschehen, in dem der Kolben im Zylinder seinen Oberen Totpunkt gerade verlassen hatte (also das Gemisch die maximale Kompression erreicht hatte). Die Zylinder des Motors werden von der Kurbelwelle bewegt, an deren Ende die Keilriemenscheibe fest verbunden ist. Deshalb ist die Position der Riemenscheibe stets synchron zur Stellung zur Kolbenstellung. Auf der drehenden Riemenscheibe war eine weiße Markierung angebracht und daneben auf einem Metallwinkel eine zweite, fixe Markierung. Das für die Einstellung des Zündzeitpunkts verwendete Stroboskop bestand aus einer Xenon- oder Neon-Blitzröhre und einem Auslösekabel. Das Kabel wurde mit einem induktiven Sensor an der Referenzzündkerze angeschlossen, so dass jedes Mal, wenn die Referenzzündkerze zündete, auch das Stroboskop einen sehr kurzen Lichtblitz erzeugte. Beim Auslösen des Stroboskops durch die Referenzzündkerze mussten beide Marken aufgrund des Stroboskopeffekts genau gegenüberliegend erscheinen, ansonsten musste der Verteiler eingestellt werden. Wir halten fest, dass bei dieser Anwendung das Stroboskop mit der Drehbewegung synchronisiert wurde. Es geht aber auch ohne ein Triggersignal!

Triggern? Völlig überflüssig!

Wenn wir eine Drehung mit einem nicht synchronisierten Stroboskop messen wollen, müssen wir die Frequenz der Lichtblitze so einstellen, dass die Markierung auf der Scheibe (fast) still zu stehen scheint und nur eine Markierung sichtbar ist. Wenn sich das Rad im Uhrzeigersinn dreht und die Markierung sich langsam im Uhrzeigersinn zu

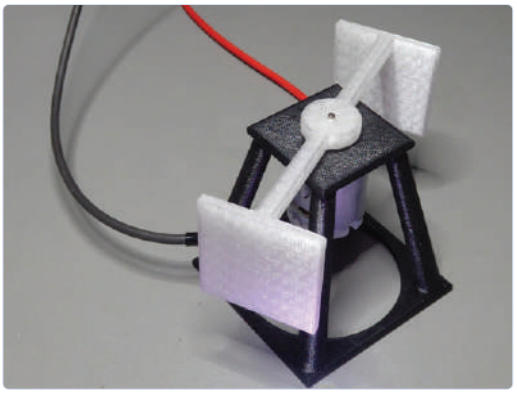


Bild 1. Der Rahmen mit den am Propeller angebrachten Reflektoren.

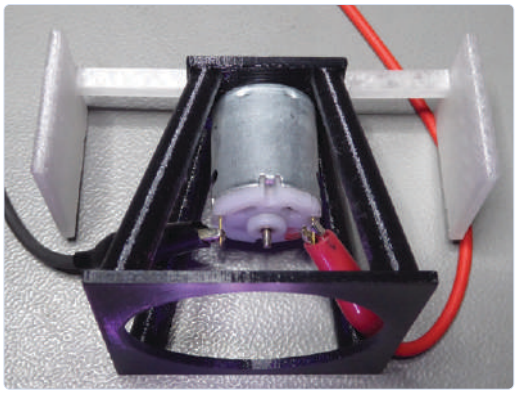


Bild 2. Nahaufnahme des Motors.

bewegen scheint, dann ist die Blinkfrequenz etwas zu niedrig. Der Blitz kommt zu spät, so dass sich die Markierung in Drehrichtung zu bewegen scheint. Und wenn sich das Rad mit dem Uhrzeigersinn dreht und die Markierung sich langsam gegen den Uhrzeigersinn zu bewegen scheint, dann ist die Blinkfrequenz etwas zu hoch: Jeder Blitz kommt etwas zu früh. Es kann auch vorkommen, dass mehrere (fast) stationäre Markierungen sichtbar sind. Dann ist die Blitzfrequenz ein Vielfaches der Drehzahl oder die Drehzahl ein Vielfaches der Blitzfrequenz. Die Drehzahl kann gemessen werden, indem die Blitzfrequenz des Stroboskops so eingestellt wird, dass die Markierung zu „stehen" scheint. Die Dauer jedes Blitzes muss kurz genug sein, um eine klare Reflexion (also eine deutlich sichtbare Markierung) zu erhalten. Ist die Blitzdauer im Verhältnis zur Blitzfrequenz zu lang, „verschmiert" die Reflexion.

Nun wird es spaßig...

Was passiert, wenn wir mehrere Blitze mit der gleichen Frequenz, aber mit unterschiedlichen Farben und mit einer Phasenverschiebung zwischen den einzelnen Blitzen erzeugen? Und wenn wir auch ein rotierendes weißes Objekt verwenden und die Frequenz, Phasenverschiebung und Breite der verschiedenfarbigen Blitze variieren?
So etwas ist mit Hilfe eines Mikrocontrollers nicht schwer zu realisieren, also hindert uns nichts! Für die verschiedenfarbigen Blitze können wir eine RGB-LED verwenden – oder besser drei, um hellere Blitze zu erhalten. Zur Steuerung von Frequenz, Phasenverschiebung und Blitzdauer verwenden wir einen Arduino Pro Mini, weil das Board über genügend I/O-Ports zum Spielen verfügt und für unseren Zweck mehr als schnell genug ist. Diese LEDs werden – wie könnte es anders sein – mit pulsbreitenmodulierten Signalen (PWM) angesteuert.
Natürlich hätten wir auch drei spezielle PWM-Module des Arduino Pro Mini verwenden können, aber diese Module verwenden drei verschiedene Timer. Das macht es schwieriger, sie zu synchronisieren oder Phasenverschiebungen zu programmieren. Aber wir benötigen nur relativ niederfrequente PWM-Signale und eine Auflösung von 16 Bit wäre maßlos übertrieben. Für unseren Zweck sind softwaregenerierte PWM-Signale (softPWM) also mehr als gut genug. Bei softPWM werden die Signale auf normalen digitalen Ausgängen ausgegeben. Die Ausgänge werden mit einem Zähler gesetzt und zurückgesetzt. Beim Erreichen eines bestimmten Zählerstandes wird ein Interrupt erzeugt. Auf diese Weise wird ein Intervall realisiert, das kurz genug ist, um die Pulsbreite oder Phasenverschiebung des PWM-Signals mit ausreichender Genauigkeit einzustellen.

Die Praxis

Um dies zu realisieren, verwendete der Autor keine Kurbelwelle, sondern einen 12-V-Gleichstrommotor (MOT3N) und entwarf dafür eine kleine Halterung. Der Motor setzt einen „Propeller" mit zwei vertikal angebrachte Reflektoren in Bewegung. All dies wurde mit einem 3D-Drucker hergestellt; die Druckdateien sind Teil des Downloads [3]. **Bild 1** und **Bild 2** zeigen, wie diese Mechanik aussieht.
Hinweis: Der vom Autor verwendete 12-V-Motor dreht sich (unbelastet) mit etwa 11500 U/min, was sich als viel zu viel für den 3D-gedruckten Propeller erwies. Beim ersten Probelauf flogen dem Autor die Bruchstücke buchstäblich um die Ohren und – viel gefährlicher – um die Augen. Also, es ist keine verrückte Idee, beim Experimentieren eine Schutzbrille zu tragen!
Der Motor wurde daher mit einer Gleichspannung von nur etwa 3 V angesteuert. Die Spannung ist einstellbar, um die Drehzahl zu synchronisieren. Bei 3 V beträgt die Drehzahl etwa 2100 U/min, was 35 Hz entspricht. Da der Propeller zwei Reflektoren besitzt (was das Auswuchten etwas erleichtert), ist die Blitzfrequenz doppelt so hoch, also 70 Hz (Periodendauer ungefähr 14 ms).
Die Schaltung in **Bild 3** ist die Einfachheit selbst, so dass sie leicht auf einem Steckbrett realisiert werden kann, wie der Aufbau des Autors in **Bild 4** beweist. Die drei RGB-LEDs werden über bescheidene BC547-Treiber auf Befehl des Arduinos ein- und ausgeschaltet. Die Versorgungsspannung für die Schaltung beträgt 5 VDC, die Stromaufnahme liegt bei etwa 500 mA.
Das Timing des Stroboskops wird durch einen Timer-Interrupt gesteuert, der alle 100 µs aufgerufen wird. Dieses Intervall von 0,1 ms ist die Auflösung, mit der die Blitzfrequenz variiert werden kann. Dies bietet genügend Möglichkeiten, um das „stillstehende" Bild des Propellers zu verschieben.

```
...
// Setup 16bit timer1 in normal operation with
   interrupt at 100us
// 16MHz/1 = 6.25ns. So to get 100us we need to let
   the timer count 1600 ticks.
TCCR1A = 0;
TCCR1B = _BV(CS10); //prescaler divide by 1
TCNT1 = 0xFFFF - 1600; // overflow is at 65535 =
   0xFFFF
TIMSK1 = _BV(TOIE1); // overflow interrupt
TCNT1 = 0;
sei();
}
ISR (TIMER1_OVF_vect)
{
TCNT1 = 0xFFFF - 1600; //100us interrupt
...
```

Die Blitzperiode ist in der Software auf 144 hartkodiert - das ist die Anzahl der Timer-Interrupts zwischen zwei aufeinanderfolgenden Blitzen. Die Periode beträgt also 144 * 100 µs = 14,4 ms. Die Blitzdauer für jede LED ist auf 8 festgelegt, was 8 * 100 µs = 800 µs entspricht. Das Tastverhältnis (Puls/Pausen-Verhältnis) beträgt somit 800 µs / 14,4 ms = 5,5%.

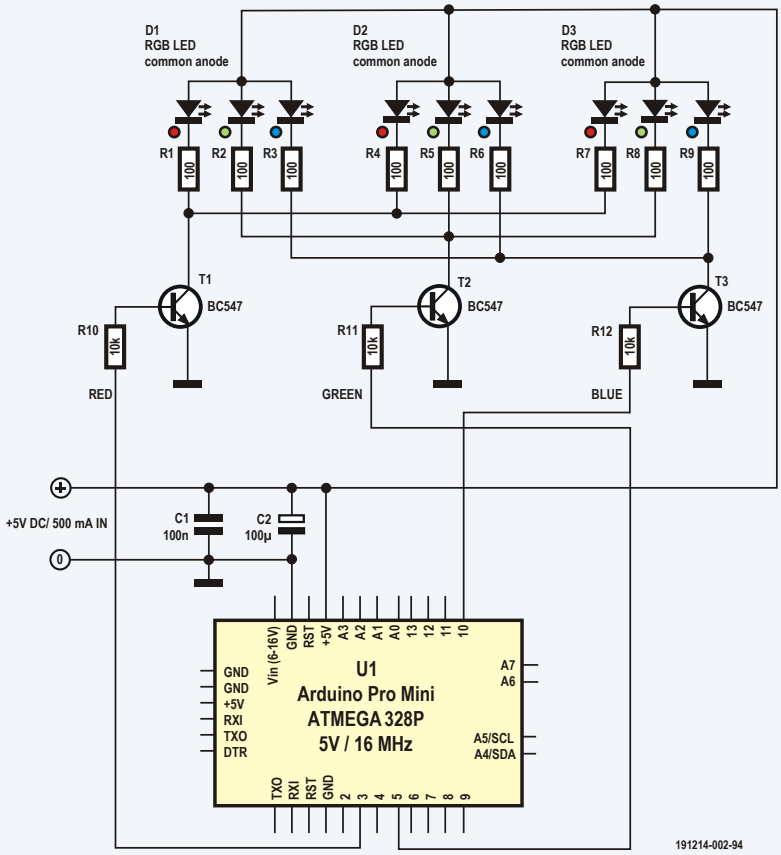


Bild 3. Die Schaltung ist nicht besonders kompliziert.

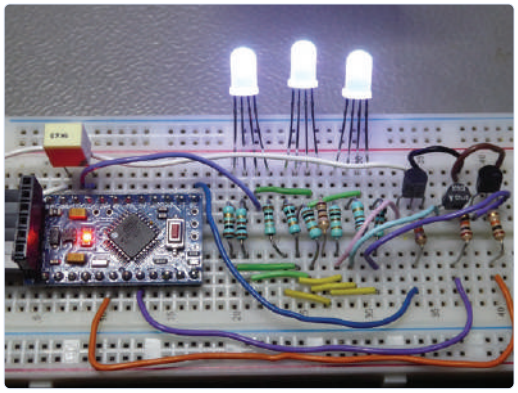


Bild 4. Der Aufbau ist unkritisch. Für Experimente eignet sich ein Steckbrett.

```
#define STROBE_PERIOD 144

TSoftPwm Pwm[] = {TSoftPwm(ID_RED, 0, 8,
   STROBE_PERIOD),
TSoftPwm(ID_GREEN, 0, 8, STROBE_PERIOD),
TSoftPwm(ID_BLUE, 0, 8, STROBE_PERIOD)};
```

Mit diesem Arbeitszyklus erhalten wir (sobald die Motordrehzahl optimal eingestellt ist) ein schön „fest stehendes" Bild.
Die Schaltung samt Software, die kostenlos von der Projektseite dieses Artikels [4] heruntergeladen werden kann, ist kein abgeschlossenes Projekt. Der Autor versteht es eher als „proof of concept" und hat es deshalb nicht mit einer schönen Benutzeroberfläche und ähnlichem versehen. Die Software demonstriert aber die Möglichkeiten, indem sie mit den Parametern Blitzfrequenz, Blitzdauer und Phasendifferenz spielt. Da die Software mit vielen und deutlichen Kommentaren versehen ist, brauchen wir hier nicht weiter ins Detail zu gehen.
Natürlich ist ein nicht synchronisiertes Stroboskop nicht wirklich optimal. Im Prinzip dürfte es gar nicht so schwierig sein, am Propeller eine Lichtschranke zu montieren, deren Signal als Auslöser für das Stroboskop dienen kann. Wir überlassen dies jedoch dem interessierten Leser, also Ihnen!

Der Autor hat zwei Videos auf Youtube gestellt, in denen man die Funktionsweise des Stroboskops [1] und das Verhältnis der LED-Steuersignale auf dem Bildschirm eines Oszilloskops [2] bewundern kann. Das Projekt ist auch auf Elektor Labs [3] zu finden.

RG — 191214-02

WEBLINKS

[1] Demo-Video 1: https://youtu.be/WHv6DCWM82k
[2] Demo-Video 2: https://youtu.be/3tYqUin4-vQ
[3] Projektseite auf Elektor Labs: www.elektormagazine.de/labs/arduino-rgb-color-stroboscope
[4] Projektseite zu diesem Artikel: www.elektormagazine.de/191214-02

Erfolg beginnt mit den richtigen **elektronischen Bauelementen**

Als autorisierter Distributor von Anbietern wie Molex, Omron oder Phoenix Contact bieten wir ein breites Sortiment an elektronischen Bauelementen – zu fairen Staffelpreisen.

conrad.de/elektronische-bauelemente

Alle Teile des Erfolgs

CONRAD

Drahtloser Notruf-Taster

Erhöhte Sicherheit mit LoRa

Von Somnath Bera (Indien)

Der in diesem Artikel beschriebene drahtlose Notrufknopf ist ein Beispiel dafür, wie ein Problem in einer komplizierten industriellen Umgebung mit ein paar preiswerten elektronischen Modulen und etwas Arduino-Programmierung gelöst werden kann. Das Ergebnis ist ein einfaches drahtloses System, das auch in vielen anderen Situationen nützlich sein kann.

Ursprünglich wurde der hier vorgestellte drahtlose Notrufknopfes (auf gut englisch: Emergency Push Button, EPB) speziell für eine industrielle Anwendung entwickelt, der Probeentnahme von Kohle, die in Zügen an ein Elektrizitätskraftwerk geliefert wurden. Eine Kohleprobe muss von der Oberseite eines Kohlewaggons entnommen werden, um den Heizwert zu bestimmen, bevor die Kohle verwendet werden kann. Dies ist ein wichtiger Parameter für ein Kraftwerk.

Vor der Probenentnahme drückte der Probenehmer auf die vorhandene Rufanlage, um dem Zugführer seine Anwesenheit zu signalisieren, damit dieser den Zug stoppen konnte. Leider kann der Probenehmer wegen der Länge des Zuges, der Krümmung des Gleises und seiner eigenen Position manchmal das Signal nicht sehen. Gelegentlich wurde beobachtet, dass der Zug bei der Probenahme versehentlich rollte, was zu Unfällen führte und die Person verletzte. Das Rufsystem war also verbesserungsbedürftig.

Dinge, die zu beachten sind

Kohlezüge können sehr lang sein, bis zu 500 m, und es können bis zu vier nebeneinander stehen, die gleichzeitig beprobt und entladen werden. Hinzu kommen noch die gebogenen Gleise, und Sie werden verstehen, dass es kompliziert ist, zu überblicken, was alles vor sich geht. Außerdem ist die Umgebung durch das Entladen der Züge sehr laut. Eine akustische Rückmeldung ist daher ebenfalls schwierig.

Das bestehende EPB-System ist kabelgebunden, was in einer solchen Umgebung recht riskant ist. Lange Kabel können unbemerkt brechen, und sie zu ersetzen oder mehr Kabel zu verlegen, um eine sichere Rückmeldung zu erhalten, ist kompliziert und teuer. Auch die Erhöhung des Signalmastes, damit er von weitem sichtbar ist, ist keine praktische Lösung, da es ein gutes Sehvermögen erfordert, um zu erkennen, für welches Gleis das Signal bestimmt ist.

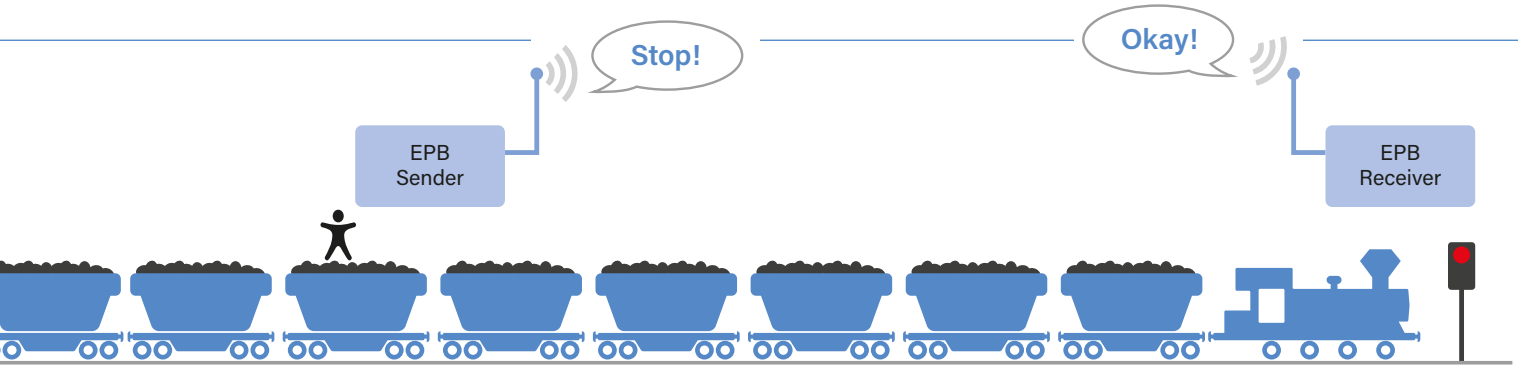


Bild 1. Überblick über das drahtlose Notfalltastensystem (EPB).

Drahtlos mit LoRa

Die Lösung, die wir uns ausgedacht haben, ist ein drahtloser Notrufknopf (**Bild 1**). Er besteht aus zwei Teilen, einem Sender und einem Empfänger. Eigentlich sind die beiden Geräte fast identisch und können sowohl senden als auch empfangen. Wenn Sie möchten, können Sie sich die Lösung auch als Master-Slave- oder Client-Server-System vorstellen.

Bevor eine Kohleprobe entnommen wird, drückt der Probenehmer eine Taste am EPB-Sender, um seine Anforderung für eine Probenahme zu signalisieren. Wenn der EPB-Empfänger in der Kontrollkabine diese Anforderung empfängt, aktiviert er ein Relais, um die Taste am bestehenden EPB-System zu „drücken", und sendet eine Bestätigung zurück an den EPB-Sender. Jetzt weiß der Probenehmer sicher, dass sein Signal angekommen ist, und er kann gefahrlos eine Probe nehmen. Wenn er fertig ist, drückt er einen zweiten Knopf am EPB-Sender, um das System freizugeben. Diesmal gibt der EPB-Empfänger den vorhandenen EPB frei, indem er ein zweites Relais aktiviert. Auch hier wird dies dem EPB-Sender bestätigt. Das System ist nun bereit für eine neue Probe. Falls das System nicht verfügbar ist oder der signalgebende EPB aus betrieblichen Gründen überbrückt wird, kehrt das Rückmeldesignal nie zum Sender zurück, wodurch der Sender gewarnt und eine Fehlkommunikation vermieden wird.

Schaltungen für Sender und Empfänger

Der EPB-Sender ist eine exakte Nachbildung der bestehenden EPB-Einheit, die jedoch zusätzlich mit einer kleinen Antenne und einem kleinen OLED-Display ausgestattet ist. Der Sender wird von einem einzelligen LiPo-Akku versorgt. Er besitzt zwei Drucktasten (1×an). Im Inneren (**Bild 2**) befindet sich ein ESP32-Modul, das das OLED-Display und ein LoRa-Transceiver-Modul steuert. Das ESP32-Modul wurde nur der Bequemlichkeit halber verwendet, nicht wegen seiner drahtlosen Fähigkeiten. Jedes andere billige, stromsparende Mikrocontroller-Modul mit den richtigen Schnittstellen (I²C, UART, 2×GPIO) kann verwendet werden.

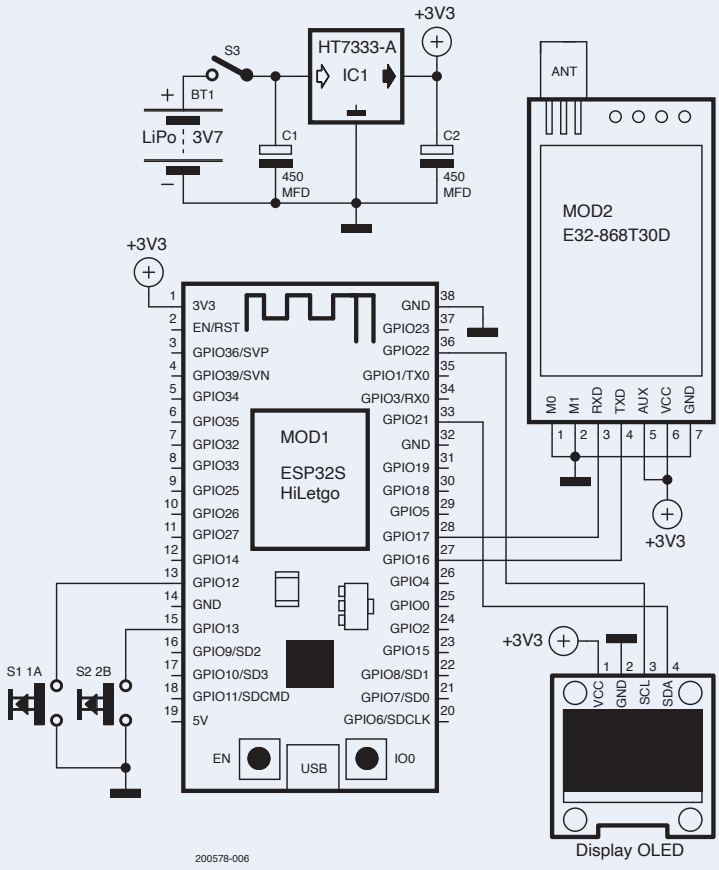
Der EPB-Empfänger ähnelt der Sendeeinheit, nur dass er statt der Drucktasten zwei Relais und kein Display besitzt (**Bild 3**). Die Software sorgt dafür, dass nur eines der beiden Relais aktiv sein kann.

Sicherheit

Um die Sicherheit zu erhöhen, damit das System nicht durch Störsignale gestört wird, haben wir stromsparende LoRa-Transceivermodule von EByte ausgewählt. Sie verwenden nicht nur das störungsresistente Verfahren der Frequenzspreizung, sondern stellen auch drei Parameter zur Verfügung: Kanalfrequenz, Übertragungsraten und eine 4-Byte-ID. Eine Kommunikation kann nur stattfinden, wenn diese drei Parameter an beiden Enden der Verbindung gleich sind.

Das Modul verfügt über eine serielle Schnittstelle und liefert bis zu 500 mW (21...30 dbm) im ISM-Band von

Bild 2. Der EPB-Sender besteht aus einem OLED-Display und zwei Drucktasten (neben dem LoRa-Transceiver).



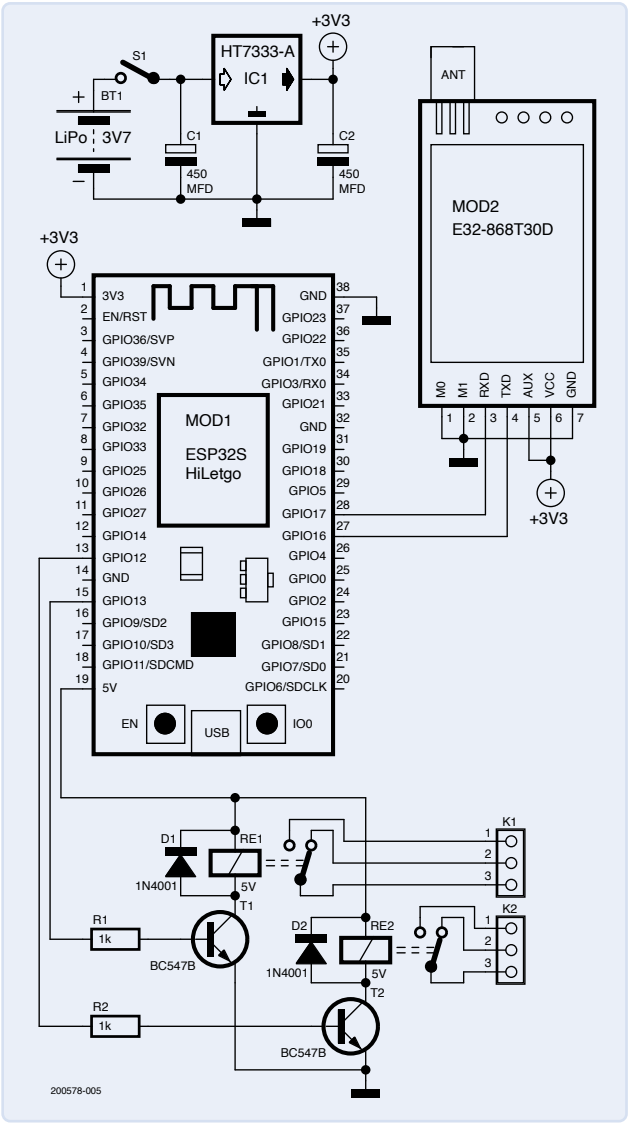


Bild 3. Die beiden Relais des EPB-Empfängers „drücken“ die Tasten des bestehenden EPB-Systems.

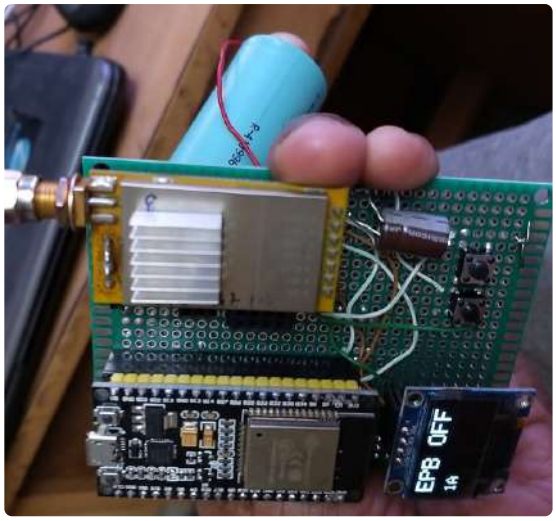


Bild 4. Prototyp des EPB-Senders, auf einem Lochrasterabschnitt aufgebaut.

868 MHz (**Bild 4**). Mit einer kleinen Handheld-Antenne und der Antenne der Empfangseinheit auf dem Dach der Kabine / des Kontrollraums kann das Signal sehr bequem und sicher 500 m entlang der Bahnstrecke zurücklegen.

Software

Wie die Hardware, so ist auch die Software für beide Geräte, zwei kurze Arduino-Sketches, ähnlich aufgebaut. Beide Sketches konfigurieren in der `setup()`-Funktion die serielle Schnittstelle, die für die Kommunikation mit dem LoRa-Transceiver verwendet wird, als 9600N81. Der EPB-Sender konfiguriert zwei GPIO-Pins als Tastereingänge, während der EPB-Empfänger sie als Ausgänge zur Ansteuerung der Relais konfiguriert. Der EPB-Sender bereitet auch das OLED-Display vor.

In der Funktion `loop()` prüfen sowohl der Sender als auch der Empfänger zunächst, ob eine Nachricht von der anderen Einheit empfangen wurde. Wenn ja, aktualisieren sie ihren Zustand entsprechend. Der Empfänger sendet eine Bestätigung an den Sender zurück und beginnt, auf eine neue Nachricht zu warten.

Der Sender fährt fort, indem er den Zustand seiner beiden Drucktasten überprüft. Je nachdem, welche Taste gedrückt wird, sendet er einen kurzen ASCII-String über die serielle Schnittstelle an den LoRa-Transceiver. Es wird immer nur ein Tastendruck akzeptiert.

Die Software kann von der Projektseite bei Elektor Labs [1] heruntergeladen werden. Sie können sie nach eigenem Ermessen verändern. Die Zeichenketten, die zwischen den beiden Geräten ausgetauscht werden, sind mehr oder weniger willkürlich gewählt und können durch alles ersetzt werden, was Sie Ihnen in den Sinn kommt..

RG – 200578-02

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Kontaktieren Sie bitte Elektor unter redaktion@elektor.de.

Passende Produkte

- **ESP32-DevKitC-32D**
  SKU 18701: https://elektor.de/18701
- **0,96“-OLED-Display (blau, I²C, 4-Pin)**
  SKU 18747: https://elektor.de/18747
- **Claus Kühnel, LoRaWAN-Knoten im IoT**
  Buch, kartoniert, SKU 19950: https://elektor.de/19950
  E-Buch, PDF, SKU 19951: https://elektor.de/19951

WEBLINK

[1] Projektdateien bei Elektor Labs: https://elektormagazine.de/labs/wagon-top-coal-sampling-remote-epb-module

Aller Anfang...

...muss nicht schwer sein

Folgen Sie dem Emitter!

Von Eric Bogers (Elektor)

Letztes Mal haben wir uns schon ganz kurz mit dem Transistor als Verstärker - im Gegensatz zum Schalttransistor - beschäftigt. Als erstes schauen wir uns jetzt deshalb den so genannten Emitterfolger an - so ziemlich die einfachste Transistorschaltung, die man sich vorstellen kann.

Die Kollektorschaltung

Die erste Verstärkerschaltung, die wir uns ansehen werden, ist die sogenannte (gemeinsame) Kollektorschaltung. Diese Schaltung wird so genannt, weil der Kollektor des Transistors der gemeinsame Bezugspunkt für Eingangs- und Ausgangssignal ist. Dieser Name ist eigentlich nicht besonders anschaulich; die Schaltung ist deshalb vor allem unter dem Namen *Emitterfolger* bekannt - und das beschreibt eigentlich ganz gut, was die Schaltung macht: Die Spannung am Emitter des Transistors „folgt" der Spannung an seiner Basis. Wozu das gut ist, werden wir in dieser Folge sehen.
Stellen Sie sich vor, Sie möchten ein Mischpult für eine Beleuchtungsanlage bauen - nichts Professionelles, nur etwas „schnell Dahingelötetes" und mit niedriger Spannung. Stellen Sie sich eine Reihe von Lampen (Kanäle) vor, die jeweils einzeln in ihrer Helligkeit eingestellt werden können (in diesem Beispiel von 0 V bis 10 V), die aber auch alle auf einmal mit einem Master-Potentiomenter gedimmt werden können. Bar jeglichem Elektronikwissen könnte man ein Master-Poti nehmen und für jede Lampe noch ein Kanal-Poti einsetzen. Wir haben dies oben in **Bild 1** skizziert.
Dies ist jedoch der falsche Weg. Alle diese Kanalpotis zusammen bilden eine große Last, die die Kennlinie des Hauptpotis ziemlich aus dem Gleichgewicht bringen würde; außerdem würden sich alle Potis gegenseitig beeinflussen, so dass es keine sinnvolle Einstellung geben kann.
Wir müssen deshalb das Hauptpoti so weit wie möglich entlasten und auch die einzelnen Kanalpotis voneinander entkoppeln. Das erreichen wir mit Emitterfolgern, wie sie unten in Bild 1 zu sehen sind. Dies funktioniert wie folgt:

Solange ein Transistor leitet, ist die Spannung an seinem Emitter etwa 0,7 V niedriger als die Spannung an seiner Basis. Das bedeutet, dass der Einstellbereich von 0 V bis 11,3 V reicht - das ist zwar etwas mehr als die nominellen 10 V dieses speziellen Beispiels, aber es bedeutet, dass, wenn die Potis ganz aufgedreht sind, die Lampen auch mit voller Helligkeit leuchten. Fällt die Spannung an der Basis unter 0,7 V, schaltet der Transistor ab und die Ausgangsspannung beträgt 0 V.
Zwischen diesen beiden „extremen" Zuständen folgt die Spannung am Emitter aber gehorsam der Spannung an der Basis, allerdings mit dem *Offset* der Basis-Emitter-Spannung von 0,7 V. Der große

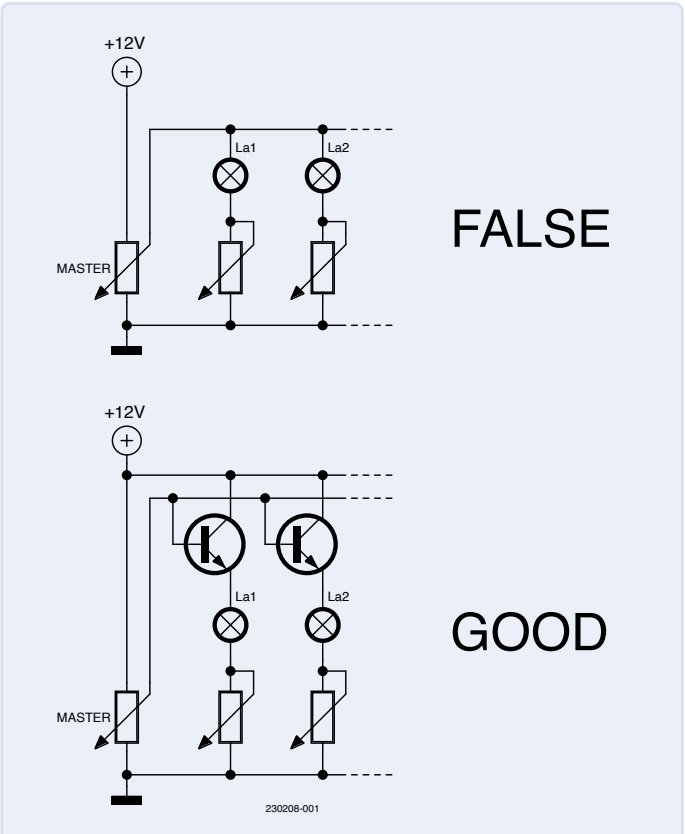


Bild 1. Ein (zugegebenermaßen sehr einfaches) Lichtmischpult - oben falsch, unten richtig.

Vorteil des Emitterfolgers besteht darin, dass die Basis nur eine sehr geringe Belastung für den Schleifer des Hauptpotis darstellt, da der Basisstrom viel kleiner ist als der Emitterstrom. Das Verhältnis von Emitter- zu Basisstrom h_{FE} (oder B) ist der Gleichstromverstärkungsfaktor des Transistors. In der Praxis bedeutet dies, dass das Hauptpoti einen Lastwiderstand „sieht“, der um den Faktor h_{FE} kleiner ist als der tatsächliche Wert des Kanalpotis.
Die Kollektorschaltung oder der Emitterfolger bewirkt also keine Spannungsverstärkung, aber eine erhebliche Stromverstärkung, einen hohen Eingangswiderstand und einen niedrigen Ausgangswiderstand. Der Emitterfolger reagiert äußerst stabil auf Laständerungen: Wenn der Lastwiderstand sinkt, würde man eigentlich erwarten, dass auch die Ausgangsspannung sinkt. Infolgedessen erhöht sich jedoch U_{BE} und auch I_B nimmt zu. Dies wiederum führt zu einem Anstieg von I_C und damit auch von I_E, was den Rückgang der Ausgangsspannung sofort ausgleicht.
Für die Eingangs- und Ausgangsimpedanz des Emitterfolgers gilt:

$$Z_{in} = (h_{FE} + 1) \cdot Z_{load}$$

$$Z_{out} = \frac{Z_{source}}{h_{FE} + 1}$$

Die stabilisierte Stromversorgung

In einer früheren Folge haben wir nach den notwendigen Berechnungen festgestellt, dass die Stabilisierung einer Spannung mit nur einer Z-Diode und einem Vorwiderstand erhebliche Nachteile hat. Eine solche Stabilisierungsschaltung funktioniert viel, viel besser, wenn wir dort auch einen Emitterfolger einbauen, wie in **Bild 2** skizziert.
Berechnen wir nun diese Schaltung mit den Werten aus dem früheren Beispiel (eine Z-Spannung von 48 V und ein maximaler Laststrom von 14,1 mA - das ist eine Phantomspeisung für ein Mikrofon). Für den Strom in die Basis des Transistors können wir dann schreiben:

$$I_B \approx \frac{I_E}{\beta} = \frac{14.1\text{mA}}{100} = 0.14\text{mA}$$

Die Stromverstärkung gilt streng genommen für den Kollektorstrom, weil der Emitterstrom gleich I_C plus I_B ist- aber im Kleinsignalbetrieb, bei einer Stromverstärkung über 100, können wir I_C und I_E in guter Näherung gleichsetzen. Den genauen Wert der Stromverstärkung des Transistors kennen wir nicht, aber mit der Annahme eines Faktors von 100 für Kleinsignaltransistoren bleiben wir vorerst auf der sicheren Seite.
Während einer Periode der Eingangswechselspannung beträgt die Spannung über den Pufferelkos etwa 60 V; über den Serienwiderstand der Zenerdiode fallen also 12 V ab:

$$R_{series} = \frac{U}{I_B} = \frac{12\text{V}}{0.14\text{mA}} = 85\text{k}\Omega$$

Für den Reihenwiderstand R_S nehmen wir den Standardwert

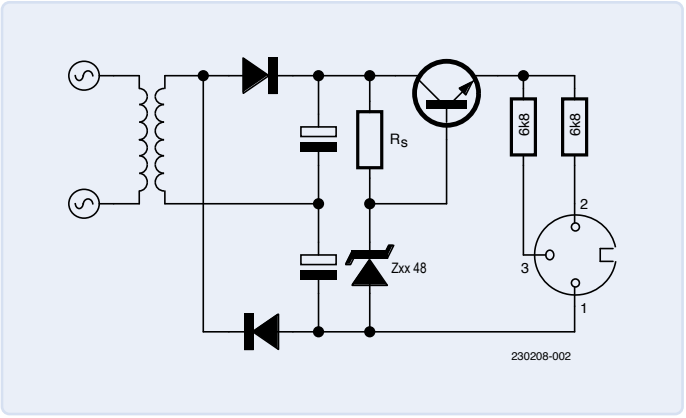


Bild 2. Stabilisierte Stromversorgung (Phantomspeisung für ein Mikrofon).

von 82 kΩ. Die Verlustleistung in der Z-Diode ist maximal, wenn keine Last angeschlossen ist und somit kein Strom in der Basis des Transistors fließt. Es gilt dann:

$$P_{zener} = U \cdot I = 48\text{V} \cdot \frac{63\text{V} - 48\text{V}}{82\text{k}\Omega} = 8.78\text{mW}$$

Natürlich wird auch im Transistor Leistung abgeführt:

$$P_{transistor} = U \cdot I = (63\text{V} - 48\text{V}) \cdot 14.1\text{mA} = 211.5\text{mW}$$

Die Summe der Verlustleistung der Z-Diode und des Transistors ist bereits kleiner als die der Z-Diode ohne Transistor (die tatsächlich etwas über 900 mA betrug). Die Schaltung hat aber noch einen zweiten Vorteil: In der Schaltung ohne Transistor tritt die maximale Verlustleistung auf, wenn kein Verbraucher angeschlossen ist, und die minimale Verlustleistung tritt bei einer Last von 0 Ω auf (was in der Praxis natürlich nie vorkommt). Im Gegensatz dazu wird in der Schaltung mit Emitterfolger die maximale Leistung bei einem Lastwiderstand von 0 Ω abgeführt, während die Verlustleistung unter „normalen“ Betriebsbedingungen geringer ist.

Der Emitterfolger als Wechselstromverstärker

Bisher haben wir den Emitterfolger ausschließlich als Gleichstromverstärker verwendet; nun wollen wir Wechselströme verstärken - siehe **Bild 3**.
Wenn über den Kondensator C1 eine Wechselspannung angelegt wird, ändert sich die Spannung an der Basis (relativ zur Masse), und der Emitterstrom ändert sich dann so, dass die Spannung am Emitter immer etwa 0,7 V niedriger ist als an der Basis. Auch hier handelt es sich also nicht um eine Spannungsverstärkung, sondern um eine Stromverstärkung.
Um den Ausgangsbereich der Schaltung zu maximieren, sollte die Ruhespannung des Emitters etwa gleich der halben Versorgungsspannung sein, in diesem Beispiel also 6 V. Zur Erinnerung: Ruhespannung und Ruhestrom sind Größen, die gemessen werden, wenn keine Signalspannung (Wechselspannung) anliegt.
Wenn wir für den Ruhestrom einen Wert von 1 mA annehmen, erhalten wir den Wert des Emitterwiderstands RE:

$$R_E = \frac{U_R}{I_R} = \frac{6\text{V}}{1\text{mA}} = 6\text{k}\Omega$$

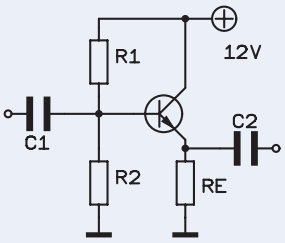


Bild 3. Wechselspannungsverstärker.

Dieser Wert von 6 kΩ liegt zwischen zwei E12-Werten. Wir wählen 5,6 kΩ. Die Spannung an der Basis des Transistors ist 0,7 V höher als die Emitterspannung, also 6,7 V gegen Masse. Wir stellen diese Spannung mit einem Spannungsteiler ein. Für R2 wählen wir willkürlich einen Wert von 100 kΩ, was uns erlaubt, R1 zu berechnen. Diese Berechnung überlassen wir getrost Ihnen; wir kommen auf das Ergebnis von 79,1 kΩ und wählen einen Wert von 82 kΩ. Damit fallen die Basis- und Emitterspannungen um etwa 0,1 V niedriger aus als eigentlich vorgesehen, aber das sollte uns keine grauen Haare wachsen lassen.

Nun müssen wir nur noch die beiden Kondensatoren dimensionieren. Da die Schaltung über einen Frequenzbereich von 20 Hz...20 kHz eine möglichst lineare Übertragung aufweisen soll, wird die untere Grenzfrequenz in beiden Fällen auf 10 Hz festgelegt. Der Kondensator C1 bildet zusammen mit der Parallelschaltung von R1 und R2 und der Impedanz des Transistors einen Hochpass. Wie wir nun wissen, hängt die Impedanz des Transistors vom Lastwiderstand ab. Wenn wir davon ausgehen, dass dieser Lastwiderstand nicht kleiner als RE ist, dann ergibt die Parallelschaltung von diesem Lastwiderstand und RE 3 kΩ. Bei einer angenommenen Stromverstärkung von 100 beträgt die Eingangsimpedanz des Transistors (R_{in}) dann etwa 300 kΩ. Die Parallelschaltung von R1, R2 und R_{in} ergibt dann 39,2 kΩ (rechnen Sie selbst!). Wir dimensionieren dann C1 wie folgt:

$$C_1 = \frac{1}{2 \cdot \pi \cdot f \cdot Z} = \frac{1}{2 \cdot 3.14 \cdot 10\text{Hz} \cdot 39.2\text{k}\Omega} = 0.4\,\mu\text{F}$$

Mit einem Wert von 0,47 µF sind wir zumindest auf der sicheren Seite. Für C2 rechnen wir mit einer angenommenen Mindestimpedanz der Last von 6 kΩ:

$$C_2 = \frac{1}{2 \cdot \pi \cdot f \cdot Z} = \frac{1}{2 \cdot 3.14 \cdot 10\text{Hz} \cdot 6\text{k}\Omega} = 2.6\,\mu\text{F}$$

Hier würden wir dann einen Wert von 3,3 µF wählen (oder 4,7 µF, wenn leichter verfügbar).

Noch ein Hinweis für die Pete-Precise-Leser: Bei der Berechnung von R1 haben wir nicht berücksichtigt, dass die DC-Impedanz des Transistors (die gleich dem Produkt aus Stromverstärkung und RE ist) parallel zu R2 liegt.

Wenn wir das sauber in die Berechnung einbeziehen wollen, müssen wir die Stromverstärkung des Transistors einigermaßen genau kennen - wenn wir dort den „sicherheitshalben" Wert von 100 annehmen, würde sich der Arbeitspunkt ziemlich verschieben. Bei Kleinsignal-Transistoren ist ein Wert von 300...500 aber praxisnäher, so dass wir seinen Einfluss auf die Berechnung von R1 ruhigen Gewissens vernachlässigen können.

So weit zu dieser Folge; beim nächsten Mal wird es richtig spannend, denn dann werden wir mit der Emitterschaltung Spannungen verstärken (und das ist etwas ganz anderes als der Emitterfolger!).

RG — 230208-02

Die Artikelreihe „Aller Anfang ..." gründet auf dem Buch „Basiskurs Elektronik" von Michael Ebner, erschienen im Elektor-Verlag.

Haben Sie Fragen oder Anmerkungen?

Haben Sie technische Fragen oder Anmerkungen zu diesem Artikel? Dann senden Sie bitte ein E-Mail an die Elektor-Redaktion unter: redaktion@elektor.de.

Passendes Produkt

› **Michael Ebner, Basiskurs Elektronik**
E-Buch, PDF, SKU 19155: www.elektor.de/19155

Beliebige und unabhängige Hysteresestufen für Komparatoren

mit Simulationen, Tabellenkalkulationen und etwas Algebra

Von Stephen Bernhoeft (Großbritannien)

Oft benötigt man Hysteresepegel, die sowohl genau als auch unabhängig voneinander sind. Auch mit den üblichen Komparator-Konfigurationen ist so etwas möglich! So geht's.

Abgesehen von dem Fall, dass die Schaltschwellen gleich weit von den Versorgungsschienen entfernt sind, wird die Auswahl von Widerstandswerten zur genauen Einhaltung von Hysteresepegeln bei Komparatorschaltungen in der verfügbaren Literatur nicht befriedigend behandelt. Deshalb sollen hier mit Hilfe einiger grundlegender algebraischer Verfahren Formeln sowohl für gebräuchliche Schaltungen mit invertierenden wie nicht-invertierenden Eingängen hergeleitet werden. Entsprechende Tabellen, Ableitungen und *LTSpice*-Schaltpläne stehen zum Download zur Verfügung [1]. Die vorgestellten Ergebnisse gelten für:

- Bauteile mit Rail-to-Rail-Ausgängen
- Open-Collector-Bauteile mit $R_{PULLUP} << R_{FEEDBACK}$

Für einen Open-Collector-Entwurf wählen Sie zunächst R_{PULLUP} und machen dann $R_{FEEDBACK}$ (im weiteren Verlauf dieses Artikels mit „*F*" bezeichnet) mindestens zehnmal so groß wie R_{PULLUP}. Die anderen Widerstände werden auf der Grundlage von *F* berechnet.
Um die Schaltung zu entwerfen, wählen Sie die gewünschten Schwellwerte und notieren Sie die Ausgangspegel. Die Werte sind ratiometrisch und können skaliert werden, aber bei Open-Collector-Bauteilen muss der Pull-up-Wert berücksichtigt werden.

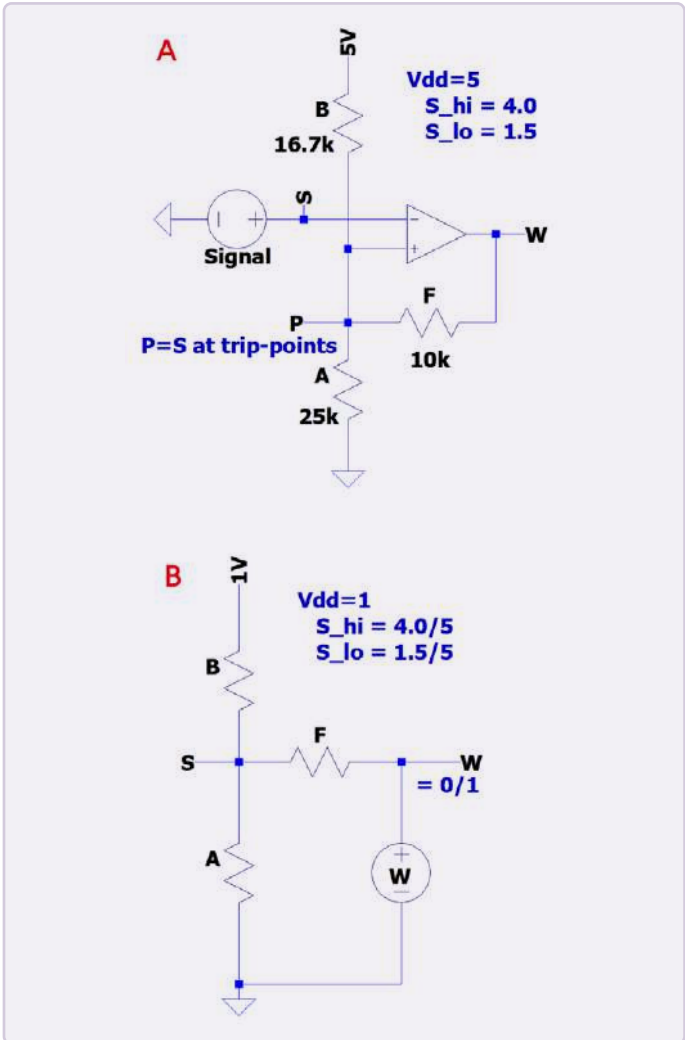


Bild 1. Oben die reale Schaltung (A), unten eine vereinfachte Version für die Analyse (B).

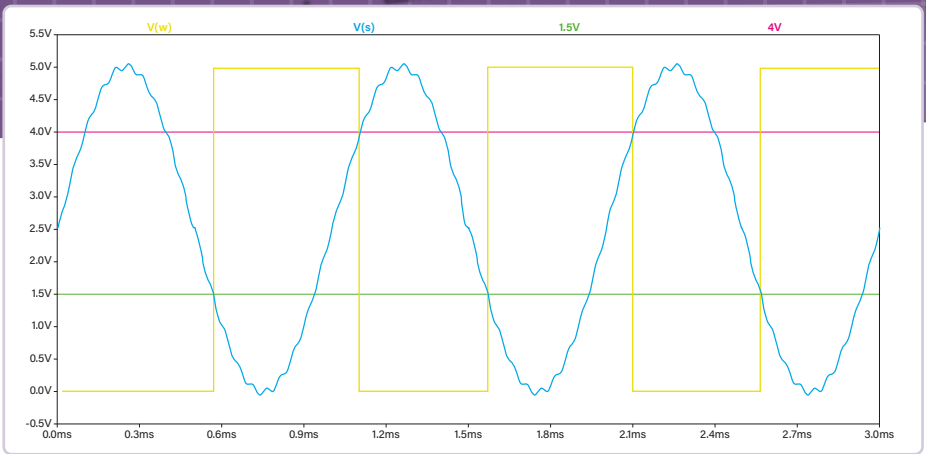


Bild 2. Die simulierten Signale der Schaltung (A) aus Bild 1.

Schaltung mit invertierendem Eingang

Der erste Fall ist in **Bild 1** dargestellt. Die Vorteile dieser Konfiguration bestehen darin, dass keine separate Referenz benötigt wird und dass sie eine hohe Eingangsimpedanz und eine geringe Anzahl von Bauteilen aufweist. Nachteilig ist die invertierende Wirkung, die unerwünscht sein kann, insbesondere wenn man die Anzahl der Bauteile insgesamt gering halten will.

In der abstrahierten Schaltung in Bild 1B ist *F* parallel zu *B* (wenn *W* = 1) oder *A* (wenn *W* = 0) geschaltet. Die Versorgungsspannung beträgt hier 1 V, um die Berechnungen zu vereinfachen, aber kann selbstredend auch skaliert werden. Eine Versorgung von 5 V zum Beispiel mit einer Auslösespannung von 4 V entspricht bei einer Versorgung von 1 V einem Auslösepunkt von 4/5 V oder 0,8 V.

W ändert seinen Zustand, wenn *P* = *S*, so dass wir davon ausgehen können, dass bei jedem Auslösepunkt *P* = *S* ist. In Bild 1A bewegt sich der Knoten *P* entsprechend dem Ausgang des Komparators. Im Schaltplan von Bild 1B können wir *P* in *S* umbenennen, da er denselben Wert wie *S* hat.

Wenn wir nun definieren:

S_{LO} = Unterer Auslösepunkt
S_{HI} = Oberer Auslösepunkt

und weiterhin annehmen, dass

$$k_{BF} = \frac{S_{HI} - S_{LO}}{S_{LO}} = \frac{B}{F}; \qquad k_{AB} = \frac{S_{LO}}{1 - S_{HI}} = \frac{A}{B}$$

dann ist

Fall 1: A gegeben

$$B = \frac{A}{k_{AB}}; \qquad F = \frac{B}{k_{BF}}$$

Fall 2: B gegeben

$$A = B \cdot k_{AB}; \qquad F = \frac{B}{k_{BF}}$$

Fall 3: F gegeben

$$B = F \cdot k_{BF}; \qquad A = B \cdot k_{AB}$$

Wir haben damit die Verhältnisse *A* : *B* und *B* : *F* in Bezug auf die Auslösepunkte festgelegt. Durch die Wahl eines Widerstandes (*A*, *B* oder *F*) können die beiden anderen Werte berechnet werden. **Bild 2** zeigt die simulierten Signale der Schaltung (A).

Überlegungen zum Entwurf

Bei der Wahl der Auslösepunkte sind verschiedene Überlegungen anzustellen:

- Für die Zeitmessung von RC-Schaltungen: Vermeiden Sie den extrem langsamen „Schwanz" der RC-Ladekurve, da winzige Spannungsänderungen große Zeitänderungen bedeuten.
- Die Auslösepegel sollten weitestgehend unabhängig von den Komparator-Offsetspannungen sein. Daher muss der niedrigste Auslösepunkt viel größer sein als der (absolute) Wert der Offset-Spannung.

Anhand einer Kalkulationstabelle [1] sehen wir, dass die Wahl eines sehr niedrigen Wertes für S_{LO} zu einer großen Streuung der erforderlichen Widerstandswerte für *A*, *B* und *F* führt, was im Hinblick auf die relative Genauigkeit der Werte nicht ideal ist. Die beste relative Genauigkeit ist gegeben, wenn *A*, *B* und *F* in etwa im gleichen Bereich liegen.

Anwendungen

Der Autor war an der Entwicklung eines Rauchwarnsystems beteiligt und kann dieses für ein praktisches Beispiel heranziehen. Ursprünglich wurde ein analoger Entwurf auf der Grundlage eines Metalloxidsensors in Erwägung gezogen (ehrlich gesagt, wir trauten unseren Fähigkeiten nicht, einen zuverlässigen Code für eine so wichtige Anwendung zu schreiben).

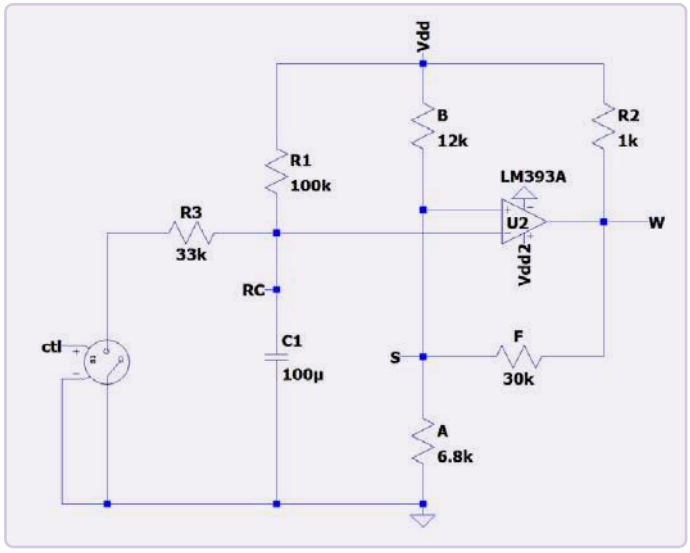


Bild 3. Die Eingangsstufe der Schaltung eines Rauchmelders. Wenn der Schalter geschlossen ist, ist R_{EQ} = 25 kΩ, V_{EQ} = 1,25 V. Wenn der Schalter geöffnet ist, ist R_{EQ} = 100 kΩ, V_{EQ} = 5 V. ΔV = (5-1,25) = 3,75 V.

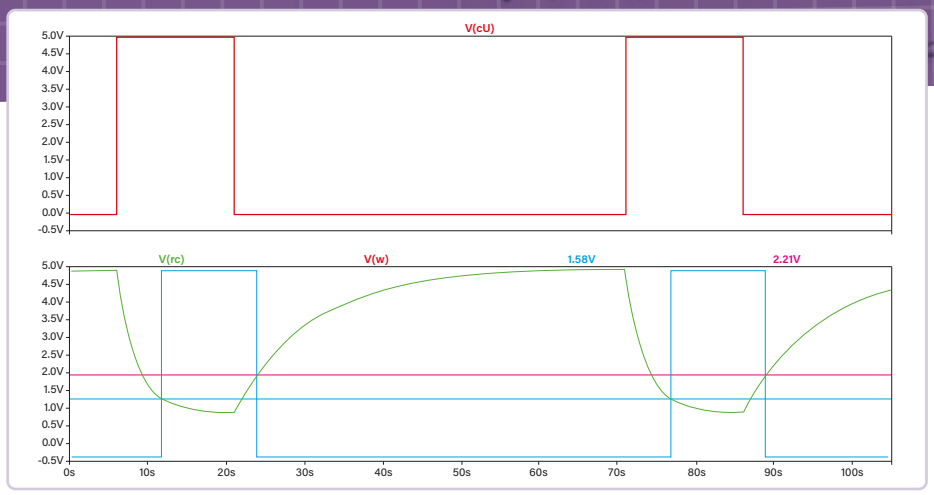


Bild 4. Der simulierte Ausgang der Schaltung aus Bild 3.

Angenommen, wir benötigen drei Sekunden Raucherkennung, bevor der Alarmausgang aktiviert wird, aber sechs Sekunden ohne Rauch, bevor der Alarm stummgeschaltet wird. **Bild 3** zeigt eine mögliche Schaltung, **Bild 4** zeigt den simulierten Ausgang.
Wenn kein Rauch erkannt wird, ist der Schalter eingeschaltet und der Kondensator lädt sich über einen effektiven Widerstand von R_{EQ} = R1|R3 auf eine „Basis" von 5/4 V auf. Wenn Rauch erkannt wird, öffnet das Signal *ctl* den Schalter S1, wodurch sich der RC-Kreis von 1,25 V in Richtung Vdd = 5 V auflädt.
Eine Kalkulationstabelle gibt die erforderlichen Auslösepunkte an, und eine weitere zeigt die Umrechnung dieser Spannungen in Widerstandswerte für *A*, *B* und *F*.

Nicht-invertierender Eingang

Betrachten wir nun die positive oder nicht invertierende Konfiguration, wie in **Bild 5** dargestellt. Eine solche Schaltung weist eine Eingangsimpedanz von *A*+*F* auf und ist etwas schwieriger zu analysieren.

Es sei:

$$\Delta S = S_{HI} - S_{LO}\ ; \qquad \Delta W = W_{HI} - W_{LO}$$
$$\Sigma S = S_{HI} + S_{LO}\ ; \qquad \Sigma W = W_{HI} + W_{LO}$$

dann:

$$\frac{A}{F} = \frac{\Delta S}{\Delta W}$$

$$M = \frac{A \cdot \Sigma W + F \cdot \Sigma S}{2 \cdot (A + F)}$$

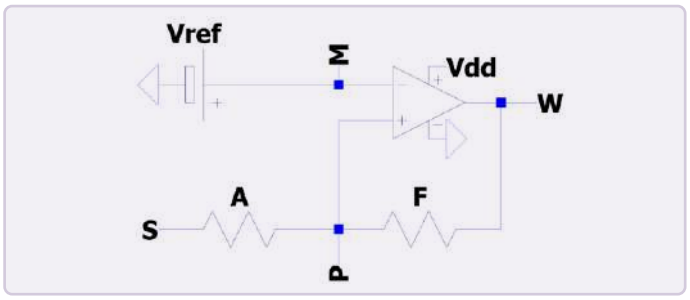


Bild 5. Die positive oder nicht-invertierende Konfiguration.

M kann auch unabhängig von *A* oder *F* ausgedrückt werden:

$$M = \left(\frac{1}{2}\right) \cdot \left(\frac{1}{\Delta W + \Delta S}\right) \cdot \left(\Delta S \cdot \Sigma W + \Delta W \cdot \Sigma S\right)$$

Ein Beispiel

Angenommen, der untere Auslösewert sei S_{LO} = 0,5, der obere Auslösewert S_{HI} = 3,0, während die Ausgangspegel W_{LO} = 0 und W_{HI} = 5 sind. Dann ist *A*/*F* = (3-0,5)/(5-0) = 0,5 und
M = 0,5 × 1/(5+2,5) × (2,5 × 5 + 5 × 3,5) = 2 V.

Bild 6 zeigt die Ergebnisse.

Stromeingangs-Hysterese

Durch Anlegen des Widerstands *A* an Masse wird die Schaltung aus Bild 3 in eine Schaltung umgewandelt, die für einen Stromsignaleingang geeignet ist, wie in **Bild 7** und **Bild 8** dargestellt. Der Wert des Widerstands *F* ergibt sich direkt aus Δ*W* und Δ*I*. Dann kann das Teilerverhältnis ß ermittelt werden. Schließlich wird der Wert von *A* berechnet. Eine Kalkulationstabelle ist hier hilfreich, da sowohl ß als auch *A* so genau wie möglich sein sollten, wenn man Standardwiderstandswerte verwendet.

$$\beta = \frac{A}{A+F}\ ; \qquad R_P = \frac{A \cdot F}{A+F}$$

$$P = \beta \cdot W_H + I_L \cdot R_P = \beta \cdot W_L + I_H \cdot R_P$$

$$\beta \cdot \Delta W = R_P \cdot \Delta I\ ; \qquad \frac{R_P}{\beta} = F = \frac{\Delta W}{\Delta I}$$

$$W_L = \frac{P - I_H \cdot R_P}{\beta} = \frac{P - I_H \cdot F \cdot \beta}{\beta} = \frac{P}{\beta} - I_H \cdot F$$

$$\frac{P}{\beta} = W_L + I_H \cdot F \quad \rightarrow \quad \beta = \frac{P}{W_L + I_H \cdot F}$$

$$A = \frac{F \cdot \beta}{1 - \beta}$$

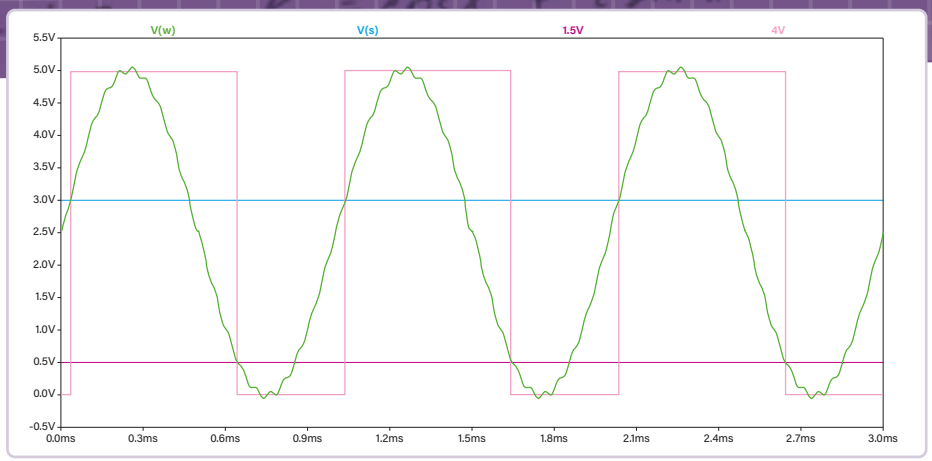


Bild 6. Die simulierten Signale der Schaltung aus Bild 5.

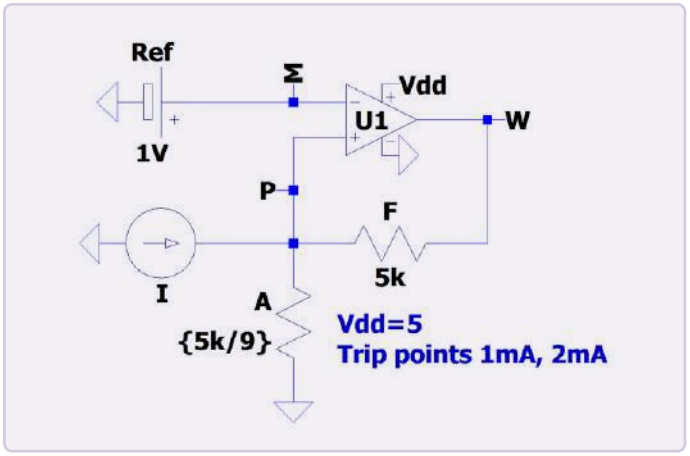


Bild 7. Die Schaltung aus Bild 3 angepasst für einen Stromeingang.

Simulationen, Tabellen und Algebra

Simulationen, Tabellen und vollständige Ableitungen für die in diesem Artikel angegebenen Formeln stehen auf der Webseite dieses Artikels [1] zum Download bereit.

RG – 200559-02

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Dann nehmen Sie Kontakt zur Elektor-Redaktion auf unter redaktion@elektor.de.

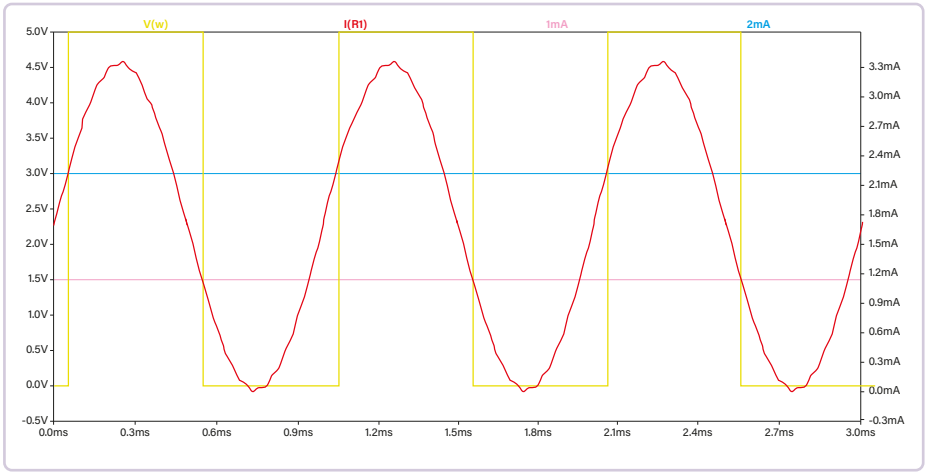


Bild 8. Ausgabe des Simulators für die Schaltung aus Bild 7.

Passende Produkte

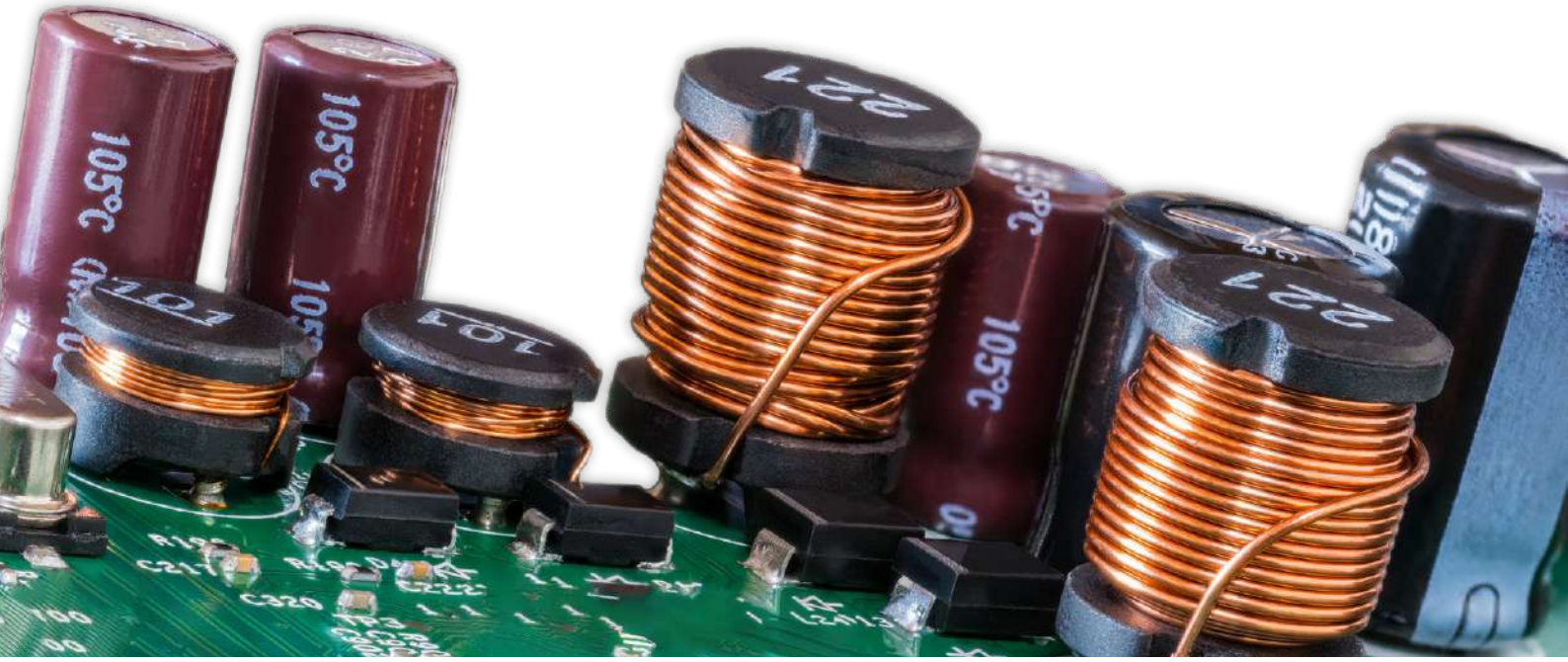
- **Paul Horowitz, Winfield Hill, The Art of Electronics** (3. Auflage)
  Buch, Hardcover, SKU 17167:
  https://elektor.de/17167
- **Gilles Brocard, The LTspice XVII Simulator**
  Buch, Hardcover, SKU 19741:
  https://elektor.de/19741
  E-Buch, PDF, SKU 20076:
  https://www.elektor.de/the-ltspice-xvii-simulator-e-book

WEBLINK

[1] Downloads für diesen Artikel: https://elektormagazine.de/200559-02

ESP32-basierter Impedanz-Analysator

Einfach, wenige Bauteile und preiswert!



Von Volker Ziemann (Schweden)

Ein ESP32 als Impedanzanalysator, der sogar die Resonanz eines LC-Netzwerks überprüfen kann? Ja, dieses preiswerte Controller-Board kann das, mit nur einer Handvoll zusätzlicher Bauteile, und bietet dennoch eine webbasierte Benutzeroberfläche.

Beginnen wir mit etwas Theorie. Die Impedanz eines elektronischen Bauteils $Z = U/I$ ist das Verhältnis zwischen der Spannung U, die über dem Bauteil abfällt, und dem Strom I, der durch es fließt. Das Ohmsche Gesetz, geschrieben in der Form $R = U/I$, ist ein Spezialfall, bei dem der Widerstand R die einfachste Form einer Impedanz ist, weil sich Strom und Spannung „in Phase" befinden. Für einen Kondensator mit der Kapazität C hängt die Impedanz von der Frequenz $\omega = 2\pi f$ ab und ist gegeben durch $-i/\omega C$. Die imaginäre Einheit i ist eine kompakte Beschreibung dafür, dass die Phase der Spannung der des Stroms um 90° nacheilt. Ebenso ist die Impedanz einer Spule mit der Induktivität L durch $i\omega L$ gegeben. Auch hier zeigt die imaginäre Einheit i an, dass die Spannung dem Strom voraus ist. Wir können also die Impedanz eines Bauteils ermitteln, indem wir die Frequenzabhängigkeit seiner Impedanz und die Phasenbeziehung zwischen Spannung und Strom untersuchen.

Besonders interessant werden diese Messungen bei einer Schaltung mehrerer Bauteile. Sind eine Spule und ein Kondensator parallel geschaltet, hat der Stromkreis die höchste Impedanz bei der Resonanzfrequenz, bei der die Wechselstromwiderstände beider Bauteile gleich sind. Bei einer Reihenschaltung ist die Impedanz an dieser Frequenz am niedrigsten. Mit dem hier beschriebenen Impedanzanalysator, der

auf dem ESP32 von Espressif basiert, kann man die frequenzabhängige Impedanzkurve von RLC-Netzwerken aufzeichnen und so deren Eigenschaften bestimmen.

Alles, was wir dazu brauchen, ist eine Möglichkeit, ein Sinussignal mit konstanter Amplitude zu erzeugen, dessen Frequenz in einem möglichst weiten Bereich zu wobbeln und schnell den Strom zu messen, der durch das/die Bauteil(e) fließt, die man als Prüfling (Device under Test, DUT) bezeichnet. Bemerkenswerterweise kann der spottbillige ESP32-Mikrocontroller, den ich von Elektor zur Teilnahme an einem Entwicklungswettbewerb im Jahr 2018 erhalten habe, die meisten dieser Dinge. Er hat einen eingebauten Frequenzgenerator und einen DAC, der Spannungen mit Frequenzen bis zu einigen 100 kHz ausgibt. Außerdem verfügt er über einen ADC, der Spannungen mit einer Rate von bis zu einer Million Mal pro Sekunde auslesen kann, wenn auch mit ein wenig Schummelei, aber dazu kommen wir noch. Da der ESP32 nur einen ADC-Kanal mit hoher Geschwindigkeit auslesen kann und der ADC und der DAC unabhängig voneinander laufen, können wir mit dem ESP32 allein nur den Betrag der Impedanz, nicht aber die Phase bestimmen. Die Ermittlung der Phasenverschiebung, die durch das Messobjekt verursacht wird, erfordert spezielle Hardware und könnte/wird daher das Thema eines separaten Artikels sein.

Analoges Frontend

Wir müssen sicherstellen, dass die Amplitude der DAC-Spannung konstant ist und nicht von der Impedanz des Prüflings abhängt. Das ist die Aufgabe des unteren Operationsverstärkers in **Bild 1**. Er dient als 1x-Puffer für das Signal von Pin 25 des ESP32. Sein niederohmiger Ausgang treibt den Prüfling. Der andere Anschluss des Prüflings ist mit dem nicht-invertierenden Eingang eines zweiten Operationsverstärkers verbunden, der als Transimpedanzverstärker beschaltet ist. Ein in seinen nicht-invertierenden Eingangsanschluss fließender Strom wird durch den Rückkopplungswiderstand R_{fb} in eine korrespondierende Spannung an seinem Ausgang umgewandelt, die zum ADC-Pin 32 des ESP32 geführt wird. Der ADC kann nur positive Spannungen verarbeiten. Daher wird der Gleichspannungspegel des Signals mit R1 und R2 auf die Hälfte der Versorgungsspannung verschoben. Mit dieser Schaltung wird der Prüfling mit einer Wechselspannung mit konstanter Amplitude beaufschlagt, während wir den Strom mit dem ADC messen.

Bild 2 zeigt im Vordergrund das analoge Frontend auf einem kleinen Stück Lochraster, wobei der Kondensator als Messobjekt (DUT) links unten zu sehen ist. Der hochkant neben dem Operationsverstärker montierte Widerstand ist der leicht zu ändernde R_{fb}. Die vier Drähte, die diese kleine Platine mit dem ESP32 verbinden, sind GND (schwarz), 3,3 V (rot), DAC-Pin 25 (grün) und ADC-Pin 32 (blau).

Hochgeschwindigkeits-ADC

Der Hochgeschwindigkeitsmodus des ADC auf dem ESP32 ist Teil des I²S-Subsystems, das normalerweise für die Bearbeitung von digitalem Sound verwendet wird. Darüber hinaus unterstützt der ESP32 einen Modus, bei dem die Ausgangsworte des ADC ohne Zutun der CPU mittels direktem Speicherzugriff (DMA) in einen Puffer kopiert werden. Der ADC läuft frei, und DMA greift die Daten mit einer bestimmten Rate ab. Solange diese Rate niedriger ist als die maximale Wandlungsrate

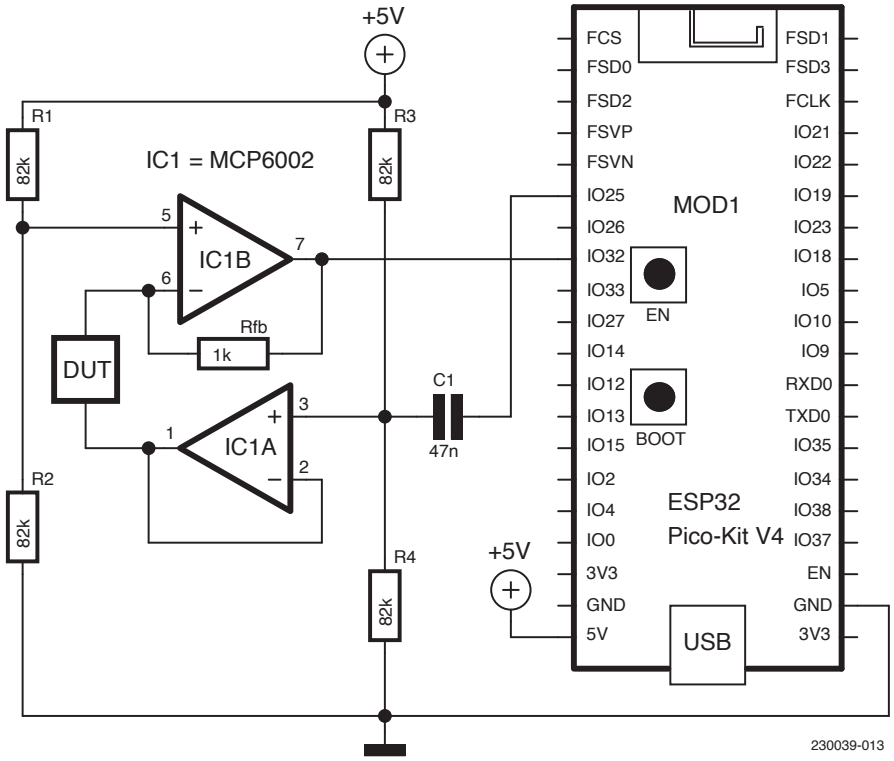


Bild 1. Das analoge Frontend.

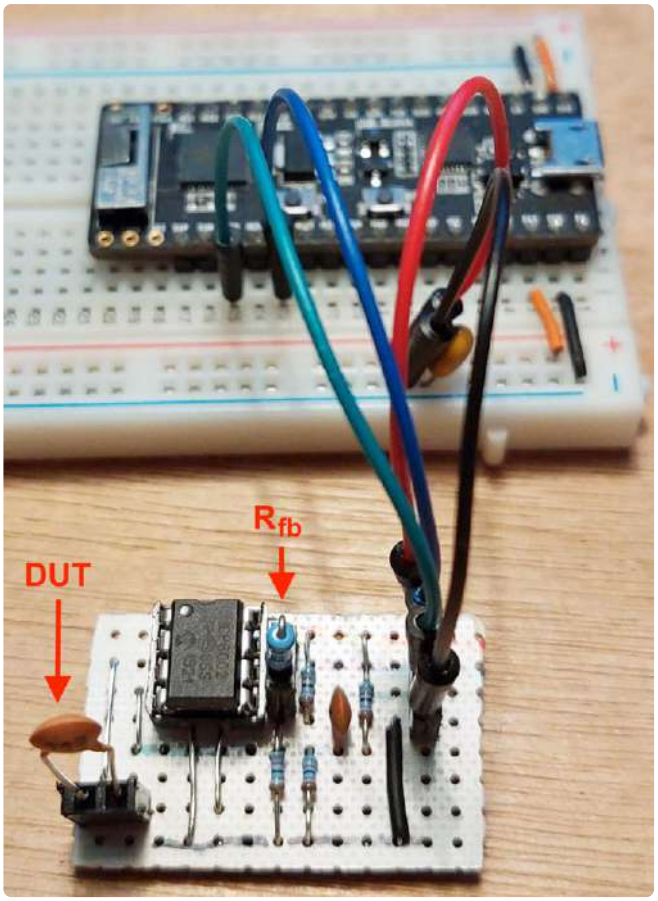


Bild 2. Damit ein ESP32 die Impedanz eines Prüflings messen kann, sind nur wenige externe Bauteile erforderlich.

des ADC, die bei etwa 250 kS/s liegt, sind alle Abtastwerte „frisch". Ist die Rate hingegen zu hoch, werden dieselben Abtastwerte in den Puffer kopiert, was sich als Treppenstufen in den Daten der erfassten Wellenform bemerkbar macht.

Die ADC-Abtastung wird mit der Funktion `i2sInit()`, die dem Sketch *HiFreq_ADC.ino* aus dem ESP32-Support-Paket nachempfunden ist, eingestellt. Der größte Teil der Konfiguration besteht darin, die Struktur `i2s_config` mit dem gewünschten Betriebsmodus, der Abtastrate und ein paar Flags zu versehen, die wir nach längerer Suche in [1] gefunden haben. Bevor wir die Funktion verlassen, stellen wir die Abschwächung so ein, dass der Skalenendwert des ADC der Versorgungsspannung von 3,3 V entspricht, wählen den ADC-Kanal und schalten den Ausgang frei.

Frequenzgenerierung

Der ESP32 hat einen eingebauten Funktionsblock, der sukzessive Werte einer Cosinusfunktion mit einer einstellbaren Rate ausspuckt. Das Ausgangssignal dieses Blocks kann durch geeignetes Setzen mehrerer Konfigurationsbits zum Eingangsregister des DACs geführt werden, der daraus eine cosinusähnliche Ausgangsspannung erzeugt. Dieser Frequenzgenerator wird durch direktes Beschreiben der Register `SENS_SAR_DAC_CTRL1_REG` und `SENS_SAR_DAC_CTRL2_REG` gesteuert, was in [1] kurz und ausführlicher in [2] beschrieben ist. Nach den Erläuterungen in [2] wird die Funktion `cwDACinit()` vorbereitet, um diese Register mit Werten zur Konfiguration der Ausgangsfrequenz und -amplitude zu füllen. Nach der Einbindung von Header-Dateien, die die Verwendung von mnemonischen Namen ermöglichen, verwenden wir `SET_PERI_REG_MASK(reg,bits)`, um `bits` im Register `reg` zu setzen. So aktiviert der erste der folgenden Befehle

```
SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL1_REG,
   SENS_SW_TONE_EN);
SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL1_REG, SENS_SW_FSTEP,
  frequency_step, SENS_SW_FSTEP_S);
```

den internen Hochgeschwindigkeits-Frequenzgenerator, der zweite Befehl bestimmt die Ausgangsfrequenz durch Angabe der Ganzzahl `frequency_step`, deren Wert von einem internen Takt abgeleitet wird. Die gewünschte Ausgangsfrequenz folgt einer leicht angepassten Gleichung aus [1].

ESP32-Sketch seriell gesteuert

Ich habe den ESP32 mit der Legacy-Version 1.8.19 der Arduino-IDE [3] programmiert. Als ich den Artikel schrieb, unterstützte die neue IDE 2 noch nicht ein später benötigtes Add-on. Ich musste auch die ESP32-Unterstützungsfunktionen aus [4] installieren und die Installationsanweisungen befolgen.

Die folgenden Sketche basieren auf [5], wo eine ausführliche Diskussion vieler Aspekte zu finden ist. Die Absicht war und ist es, die Frequenzerzeugung und -erfassung über die serielle Leitung zu steuern, basierend auf einem einfachen Protokoll, bei dem nur Strings zwischen dem ESP32 und einem seriellen Kommunikationsprogramm (Python, Octave oder LabView) auf einem PC hin- und hergeschickt werden. Das Protokoll ähnelt dem SCPI-Protokoll, das von Oszilloskopen und anderen Messgeräten verwendet wird, wobei ein angehängtes Fragezeichen einen Befehl anzeigt, der eine Antwort vom ESP32 erwartet. Das Senden von `STATE?` löst zum Beispiel die Antwort `STATE fmin fmax fstep` aus, wobei die drei numerischen Werte den Bereich und die Schrittweite des Frequenzsweeps angeben.

Nun zu den einzelnen Teilen des Sketches. Zunächst werden zwei Header-Dateien eingebunden - eine für die selbstentwickelten Support-Funktionen für ADC und DAC, die andere für den Zugriff auf ein Flash-basiertes SPIFFS-Dateisystem auf dem ESP32 zur Speicherung von Kalibrierungsdaten. Standardwerte für den Bereich des Frequenzsweeps und andere relevante Variablen werden ebenfalls angegeben. Die `setup()`-Funktion des Sketches initialisiert die serielle Kommunikation, stellt die Ausgangsfrequenz und die Amplitude des DAC ein und liest Kalibrierungsdaten aus dem SPIFFS-Dateisystem, falls diese verfügbar sind.

Die Funktion `loop()` prüft mit dem Aufruf von `Serial.available()`, ob eine Anfrage vom Host-Computer eingetroffen ist, liest eine Zeile (abgeschlossen durch ein Zeilenvorschubzeichen `\n`) und konvertiert das, was eingetroffen ist, in das Zeichen-Array `line`. Nun werden einige Tests durchgeführt, ob die Zeile eine bestimmte Zeichenkette enthält. Wenn `line` beispielsweise `FREQ 20000` enthält, wird der numerische Wert mit `atoi(&line[5])` extrahiert. Die Zeichenkette, die an der fünften Position in `line` beginnt, wird mit der eingebauten Funktion `atoi()` in eine Ganzzahl umgewandelt. Anschließend wird dieser Wert in die Variable `dac25freq` kopiert und der Frequenzgenerator mit `cwDACinit()` initialisiert. In ähnlicher Weise sind alle relevanten Variablen vom Host-Computer aus zugänglich.

Der interessanteste Befehl ist `SWEEP?` Nach Erhalt dieses Befehls und der Deklaration der in diesem Abschnitt verwendeten Variablen iteriert eine for-Schleife die Variable `f` von `fmin` bis `fmax` mit einer Schrittweite von `fstep`. Innerhalb der Schleife wird zunächst die Frequenz des DAC auf diese Frequenz gesetzt, und nach einer kurzen Verzögerung werden die Messwerte des ADC mit `is2_read()` abgerufen.

```
for (float f=freqmin; f<freqmax;f+=freqstep) {
  dac25freq=f;
  cwDACinit(dac25freq,dac25scale,0);
  delay(50);
  i2s_read(I2S_NUM_0, &buffer,
     sizeof(buffer), &bytes_read, 15);
...
}
```

Die Größe des Eingangsarguments `buffer`, die oben im Header definiert ist, bestimmt die Anzahl der Samples, hier 1024. Die Funktion gibt auch die Anzahl der `bytes_read` zurück. Da jede Abtastung zwei Bytes zurückgibt, müssen wir durch zwei teilen, wenn wir in der Schleife über die Samples laufen, um die minimalen und maximalen Werte zu bestimmen. Da die vom ADC gemessene Spannung proportional zum Strom ist, der durch den Prüfling fließt, verwenden wir diese Spitze-Spitze-Variation als Maß für die Stromamplitude.

Kalibrierung

Während des Schleifenablaufs über die Samples werden auch die Summen `q1` und `q2` akkumuliert. Diese passen zusammen mit den Variablen `S0`, `S1` und `S2` eine Gerade an die Datenpunkte an, deren Parameter für die Kalibrierung benötigt werden. Die Einzelheiten des Algorithmus sind in Anhang B von [5] erläutert. Der Befehl `SAVECALIB` speichert diese Parameter im persistenten SPIFFS-Speicher; der Befehl `GETCALIB?` ruft sie vom PC ab.

Die Kalibrierungskonstanten `calib_slope` und `calib_offset` werden benötigt, um verschiedene unbekannte Dämpfungsfaktoren im System zu berücksichtigen, die zum Beispiel aufgrund des AC-Kopplungskondensators am Eingang des unteren Operationsverstärkers entstehen und durch die Kalibrierung des Systems mit einem bekannten Widerstand als Prüfling behoben werden. Alle anderen Impedanzen werden dann in Bezug auf diesen Kalibrierwiderstand bestimmt. Im Idealfall sollte dieser Widerstand unabhängig von der Frequenz sein, aber kleine systembedingte Abweichungen werden berücksichtigt, indem zusätzlich zum Offset die Steigung als Funktion der Frequenz verwendet wird. Der Wert des Kalibrierungswiderstands sollte ungefähr den gleichen Wert aufweisen wie der Rückkopplungswiderstand des Transimpedanzverstärkers.

Vor der Verwendung des Systems sollte das SPIFFS-Dateisystem auf dem ESP32 initialisiert werden, indem ein leeres Verzeichnis mit dem Namen `data` in dem Verzeichnis angelegt wird, in dem der Sketch gespeichert ist. Dann muss das Arduino-ESP32-Filesystem-Uploader-Plugin aus [6] installiert werden. Danach erscheint ein neuer Menüpunkt *ESP32 Sketch Data Upload* im *Werkzeuge*-Menü der Arduino-IDE. Bei geschlossenem seriellen Monitor klickt man auf diesen Eintrag, um das SPIFFS-Dateisystem auf dem ESP32 zu erstellen.

Steuerung mit Octave oder Python

Um die Datenerfassung vom PC aus zu steuern, muss nur die serielle Schnittstelle geöffnet werden, deren Name sich von Betriebssystem zu Betriebssystem unterscheidet. Unter Windows heißt sie typischerweise `COMn`, auf einem MAC `/dev/tty.usbserial-n` und unter Linux `/dev/ttyUSBn`. Zum Senden eines Befehls, zum Beispiel zum Einstellen der Startfrequenz des Sweeps, muss nur `FMIN 10000` zur seriellen Schnittstelle geschrieben werden. Um die aktuellen Einstellungen zu lesen, schreibt man `STATE?`, wartet eine Weile und liest die zurückgelieferten Zeichen bis zum Zeilenvorschubzeichen `\n`. Die Antwort lautet `STATE fmin fmax fstep`. Die Durchführung eines Frequenzsweeps wird durch das Schreiben von `SWEEP?` eingeleitet und die Werte werden empfangen, jeweils ein Wert pro Zeile. Sobald alle Daten verfügbar sind, können wir die serielle Schnittstelle schließen und eine Plotdarstellung vorbereiten.

Skripte sowohl für Octave mit installierter Instrumenten-Toolbox als auch für Python mit dem Paket *python-serial* sind im Software-Archiv zu diesem Artikel enthalten. Für erste Tests kann auch einfach ein Terminalprogramm wie Putty verwendet werden.

Verwendung des Systems

Um das System in Betrieb zu nehmen, wird ein genauer 1-kΩ-Kalibrierungswiderstand als Prüfling verwendet und ein Frequenzsweep durchgeführt. Anschließend wird der Befehl `SAVECALIB`, der im zugehörigen

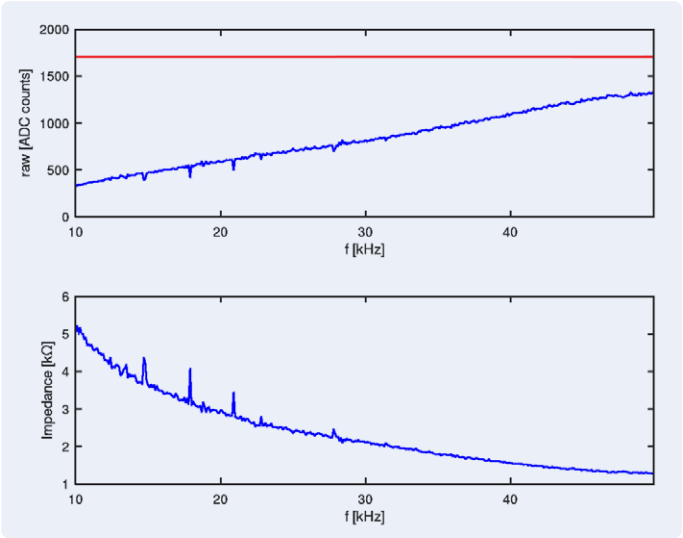


Bild 3. Rohe Messwerte und Impedanz für einen Kondensator als DUT.

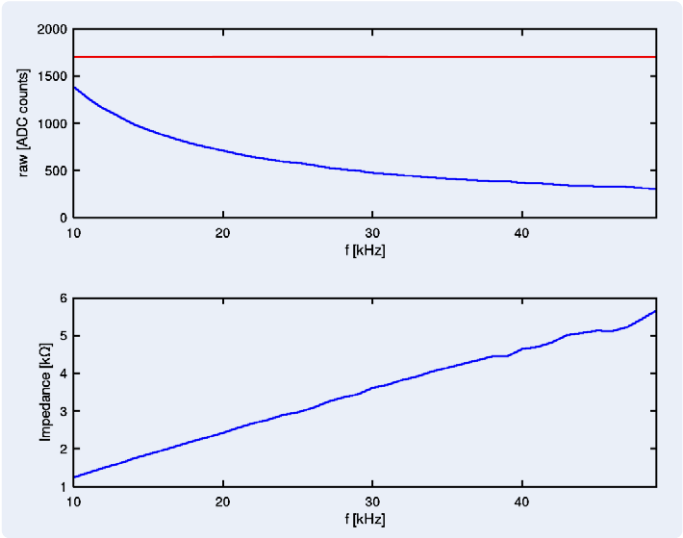


Bild 4. Rohe Messwerte und Impedanz für eine Induktivität als Prüfling.

Octave-Skript `nwa.m` standardmäßig auskommentiert ist, ausgeführt, um die Kalibrierungsdaten auf dem ESP32 zu speichern. Wenn man nun den Befehl `SAVECALIB` wieder auskommentiert, den Widerstand durch einen 2,2-nF-Kondensator ersetzt und `nwa.m` ausführt, erscheint das in **Bild 3** gezeigte, korrekt kalibrierte Diagramm. Es zeigt die erwartete inverse Abhängigkeit von der Frequenz, aus der wir mit dem Octave-Befehl die Kapazität bestimmen:

```
capacitance_nF=
    mean(1./(2*pi*Zabs*1e3.*xx*1e3))*1e9
```

mit der Formel:

$C = 1/(2\pi f Z_{abs})$

Die Mittelwertbildung über alle verfügbaren Datenpunkte erfolgt mit der eingebauten Funktion `mean()`. Die Zehnerpotenzen berücksichtigen die Kilos in kΩ und kHz sowie die Nanos in nF. Zum Beispiel wandelt `1e9` am Ende die Kapazität von Farad in Nanofarad um.
Wiederholt man den Sweep statt mit dem Kondensator mit einer 22-mH-Induktivität, erhält man die in **Bild 4** gezeigten Diagramme. Wie erwartet, steigt im unteren Bild die Impedanz einer Spule linear mit der Frequenz, so dass die Induktivität nach folgender Formel bestimmt werden kann:

$L = Z_{abs}/2\pi f$

In Oktave geschrieben sieht das so aus:

```
inductance_mH=
    mean(1e3*Zabs./(2*pi*xx*1e3))*1e3
```

wobei auch hier mit `mean()` über alle Datenpunkte gemittelt wird. Auch hier werden die Zehnerpotenzen benötigt, um die Kilos in kΩ und kHz sowie die Millis in mH zu berücksichtigen.

Web-basierte Benutzeroberfläche

Bisher wurde die Impedanzmessung von Octave oder Python aus gesteuert, aber jetzt soll ein Webbrowser zur Steuerung und zur Anzeige der Messungen verwendet werden. Dazu wird der ESP32 so konfiguriert, dass ein Webserver läuft, der die in **Bild 5** gezeigte Webseite ausgibt. Sobald der Browser diese Webseite empfängt, öffnet der eingebettete JavaScript-Code einen zweiten Kommunikationskanal, der auf *websockets* basiert, zum ESP32. Hier übernimmt ein Websocket die Rolle der seriellen Leitung und wird verwendet, um textbasierte Nachrichten hin und her zu senden. Das Senden dieser Nachrichten wird durch Klicken auf die Schaltflächen am oberen Rand der Webseite ausgelöst. Sie lösen einen Frequenzsweep aus, löschen die angezeigten Kurven und speichern die Kalibrierungsdaten. In der Zeile darunter können Parameter zur Steuerung des ESP32, einschließlich des Bereichs des Sweeps und der Abtastrate des ADC, über Auswahlmenüs eingestellt werden. Die Schaltflächen in der dritten Zeile berechnen die Kapazität, den ohmschen Widerstand und die Induktivität und zeigen das Ergebnis in den Statuszeilen unterhalb des Diagramms an. In der unteren Zeile werden die vom ESP32 empfangenen Informationen angezeigt. Im Diagramm zeigt die horizontale Achse die Frequenzachse in dem Bereich an, der in den Menüs am oberen Rand der Seite angegeben ist. Die vertikale Achse zeigt Z_{abs} relativ zum Kalibrierungswiderstand, der als horizontale blaue Linie angezeigt wird. Wenn Sie auf das Diagramm klicken, werden die Frequenz und Z_{abs} in der Statuszeile angezeigt.

Der Sketch auf dem ESP32

Nach der Anpassung des WLAN-Namens (SSID) und des Passworts müssen die Dateien zur Unterstützung von WLAN, Websocket und Webserver hinzugefügt werden. Dann konfigurieren wir den Webserver `server2` so, dass er auf Netzwerk-Port 80 hört, und den `websocket` so, dass er über Port 81 kommuniziert. Die zwischen Browsern gesendeten Textnachrichten sind JSON-formatiert und haben die Form `{“INFO”:”Yada yada”}`. Das Parsen dieser Nachrichten und das Erstellen komplexerer Nachrichten wird von der `ArduinoJson`-Bibliothek übernommen, während die Unterstützung für den ADC und DAC wie bisher von *ESP32_I2Sconfig.h* bereitgestellt wird.

```
const char* ssid = “YOUR_SSID”;
const char* password = “YOUR_PASSWORD”;;
#include <WiFi.h>
#include <WebSocketsServer.h>
WebSocketsServer webSocket = WebSocketsServer(81);
#include <WebServer.h>
#include <SPIFFS.h>
WebServer server2(80);
#include <ArduinoJson.h>
#include “ESP32_I2Sconfig.h”
```

Nach dem Festlegen mehrerer Variablen werden Hilfsfunktionen definiert, von denen `webSocketEvent()` die wichtigste ist. Sie wird immer dann aufgerufen, wenn eine Nachricht vom Browser eintrifft. Wenn es sich um eine JSON-formatierte Nachricht handelt, extrahiert das folgende Codeschnipsel den Befehl `cmd` und den Wert `val` mit Hilfe von Funktionen aus der `ArduinoJson`-Bibliothek. Je nach Befehl löst er dann die entsprechenden Aktionen aus. Empfängt er zum Beispiel den Befehl `SWEEP`, meldet er `sendMSG()` an den Browser zurück und setzt die Variable `mmode` auf eins, was in der Hauptschleife interpretiert wird. Da `webSocketEvent()` asynchron aufgerufen wird,

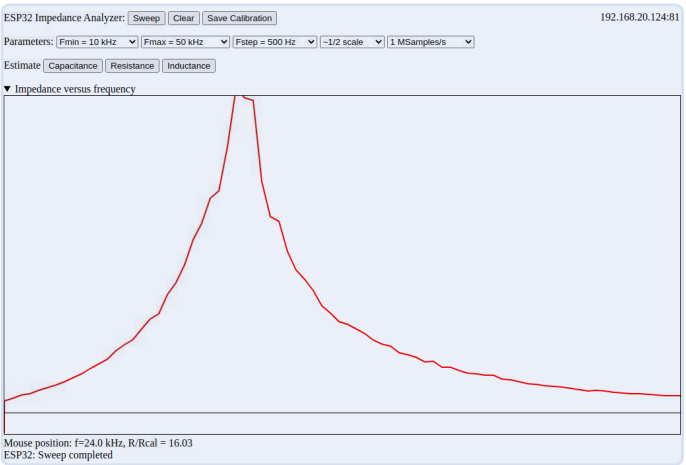


Bild 5. Die von ESP32 erstellte Webseite zeigt im Diagramm die Resonanz einer Parallelschaltung eines 2,2-nF-Kondensators, einer 22-mH-Induktivität und eines 33-kΩ-Widerstands.

unterbricht es (möglichst kurz) anderen Aktivitäten. Daher wird der Sweep im Hauptprogramm durchgeführt. Anderseits verbraucht das Setzen der Startfrequenz des Sweeps `freqmin` nicht viele CPU-Zyklen und wird innerhalb von `webSocketEvent()` behandelt.

```
DynamicJsonDocument root(300);
deserializeJson(root,payload);
const char *cmd = root[“cmd”];
const long val = root[“val”];
if (strstr(cmd, “SWEEP”)) {
  sendMSG(“INFO”, “ESP32: Received Sweep command”);
  mmode=1;
} else if (strstr(cmd, “FMIN”)) {
  freqmin=val;
  Serial.printf(“FreqMin = %g\n”,freqmin);
} sonst

...
```

Eine Anzahl weiterer Befehle wird ähnlich gehandhabt. Der folgende Codeschnipsel aus der Funktion `setup()` stellt zunächst eine Verbindung zum WLAN mit den zuvor angegebenen Anmeldeinformationen her. Er gibt Punkte auf der seriellen Leitung aus, bis er erfolgreich Verbindung aufgenommen hat, und gibt dann die IP-Nummer aus, startet die Websocket-Kommunikation und registriert `webSocketEvent`, um die über den Websocket eingehenden Nachrichten zu verarbeiten. Beachten Sie, dass die serielle Leitung in diesem Teil nicht unbedingt benötigt wird, aber es ist insbesondere während der Entwicklung des Systems doch angenehm zu beobachten, was auf dem ESP32 passiert.

```
WiFi.begin(ssid, password);
while(WiFi.status() != WL_CONNECTED)

Serial.print(“\nConnected to “);
Serial.print(ssid);
Serial.print(“ with IP-Address: “);
Serial.println(WiFi.localIP());
webSocket.begin();
webSocket.onEvent(webSocketEvent);
```

Noch in `setup()` prüfen wir, ob das SPIFFS-Dateisystem existiert, genau wie im ersten Teil, und laden die Kalibrierungsdaten. Jetzt enthält das SPIFFS-Dateisystem aber auch die Datei *esp32_impedance_analyzer.html*, die die Webseite beschreibt. Nach dem Start von `server2` wird diese HTML-Datei so registriert, dass sie standardmäßig an verbundene Browser ausgeliefert wird, wie das erste Argument in `server2.serveStatic()` angibt:

```
server2.begin();
server2.serveStatic(“/”,SPIFFS,
     “/esp32_impedance_analyzer.html”);
```

Der Rest der Funktion `setup()` ähnelt weitgehend dem früheren Sketch. In der Funktion `loop()` wird zunächst alles behandelt, was mit den http- und Websocket-Servern zu tun hat, bevor die Variable `info_available` überprüft wird. Diese wird von `sendMSG()` gesetzt, um anzuzeigen, dass der `info_buffer` eine JSON-formatierte Nachricht enthält, die umgehend mit einem Aufruf von `webSocket.sendTXT()` an den Browser gesendet wird.

```
server2.handleClient(); // handle http server
webSocket.loop(); // handle websocket server
if (info_available==1) {
  info_available=0;
  webSocket.sendTXT(websock_num,
     info_buffer,strlen(info_buffer));
}
```

Dann wird der Wert von `mmode` geprüft und die entsprechende Aktion gestartet. Bei `mmode==1` wird zum Beispiel der Frequenzsweep mit einem Code durchgeführt, der dem zuvor verwendeten sehr ähnlich ist. Hier wird nur eine JSON-formatierte Nachricht mit dem Namen `WF0` (für waveform) erstellt und in der Variablen `doc` gespeichert:

```
doc[“WF0”][nn-1]=floor((512/16)*Z);
```

wobei die Variable `nn` eine Schleife über die Frequenzpunkte erzeugt und `Z` der absolute Wert der Impedanz bei dieser Frequenz ist. Beachten Sie, dass die Variable für 512 Pixel skaliert wird, die den Bereich von Null bis 16 kΩ abdecken, und die Werte mit `floor()` in eine ganze Zahl umgewandelt werden. Nach Beendigung der Schleife wird die Wellenform zum Browser gesendet:

```
serializeJson(doc,out);
webSocket.sendTXT(websock_num,out,strlen(out));
sendMSG(“INFO”, “ESP32: Sweep completed”);
```

Der Abschluss des Sweeps wird mit *sendMSG()* gemeldet. Die anderen Werte von `mmode` speichern die Kalibrierung und melden die geschätzte Kapazität, den Widerstand oder die Induktivität an den Browser.
Die Datei *esp32_impedance_analyzer.html* beschreibt die Webseite und enthält das JavaScript, das sie zum Leben erweckt.

Die Webseite

Die meisten interaktiven Webseiten verwenden `<style>`-Tags, um allgemeine Aspekte des Aussehens der Dinge zu beschreiben, die auf der Seite angezeigt werden, gefolgt von HTML-Befehlen, die die Webseite beschreiben, und JavaScript-Anweisungen, um sie interaktiv zu machen.
Die folgenden Style-Befehle oben in *esp32_impedance_analyzer.html* bewirken, dass ein Rahmen um den Anzeigebereich gezogen wird und dass zwei angezeigte Messkurven in Rot und Schwarz dargestellt werden. Die letzte Anweisung bewirkt, dass die IP auf der rechten Seite der Webseite angezeigt wird:

```
<style>
#displayarea { border: 1px solid black; }
#trace0 { fill: none; stroke: red; stroke-width: 2px;}
#trace1 { fill: none; stroke: black; stroke-width: 1px;}
#ip {float: right;}
</style>
```

Der Hauptteil des Webseiten-Skripts steht zwischen den Tags `<BODY>` und `</BODY>`. Ganz oben befindet sich die Beschreibung

der Schaltflächen. Die Definition der ersten Schaltfläche ist zwischen `button`-Tags eingeschlossen und lautet wie folgt:

```
<button id=”sweep” type=”button”
     onclick=”sweep();”>Sweep</button>
```

Sie bewirkt, dass *Sweep* auf der Schaltfläche angezeigt wird und ein Klick auf den Button die JavaScript-Funktion `sweep()` ausführt. Solche an ein Ereignis gebundenen Aktionen werden gemeinhin als *Callback-Funktionen* bezeichnet. Wenn wir der Schaltfläche eine ID zuweisen, nämlich `sweep`, können wir später ihre Eigenschaften ändern, zum Beispiel den angezeigten Text oder die auszulösende Aktion. Die Definition der anderen Schaltflächen erfolgt nach dem gleichen Schema. Das Auswahlmenü in der zweiten Zeile folgt einer etwas anderen Syntax. Die Definition dieses Menüs wird zwischen `SELECT`-Tags eingeschlossen. Wenn einer der Einträge ausgewählt wird, wird `setDacFreqMin(this.value)` aufgerufen, wobei `this.value` der in den verschiedenen `OPTION`-Tags angegebene Wert ist. Die `OPTGROUP`-Tags sind nur schmückendes Beiwerk.

```
<SELECT onchange=”setDacFreqMin(this.value);”>
 <OPTGROUP label=”Sweep start frequency”>
  <OPTION value=”1000”>Fmin = 1 kHz</OPTION>
  <OPTION value=”2000”>Fmin = 2 kHz</OPTION>
  <OPTION value=”5000”>Fmin = 5 kHz</OPTION>
  <OPTION selected=”selected” value=”10000”>Fmin = 10
kHz</OPTION>
  <OPTION value=”20000”>Fmin = 20 kHz</OPTION>
  <OPTION value=”50000”>Fmin = 50 kHz</OPTION>
 </OPTGROUP>
</SELECT>
```

Der Bereich zur Anzeige der Impedanz schließlich ist eine *Skalierbare Vektorgrafik* (SVG) mit der angegebenen Größe von 1024 × 512 Pixeln und zeigt zwei `path` mit den IDs `trace0` und `trace1`. Die `id` ermöglicht es uns später, sie individuell zu ändern. Die Eigenschaft `d` beschreibt die Wellenform. Die Initialisierung erfolgt, indem wegen `M0 200` der Cursor auf das Pixel (0,200) bewegt wird. Später wird `d` mit der Wellenform `WF0`, die vom ESP32 kommt, neu definiert.

```
<svg id=”displayarea” width=”1024px” height=”512px”>
<path id=”trace0” d=”M0 200” />
<path id=”trace1” d=”M0 200” />
</svg>
```

Schließlich werden die beiden Statuszeilen definiert mit:

```
<div id=”status”>Status window</div>
<div id=”reply”>Reply from ESP32</div>
```

Sie werden über `id` ausgewählt, was es später ermöglicht, den angezeigten Text zu ändern. Ebenso wird der Bereich zur Anzeige der IP-Nummer oben rechts `ip` genannt.

JavaScript

Alle JavaScript-Befehle werden von `SCRIPT`-Tags umschlossen. Nach dem öffnenden Tag werden mehrere Variablen definiert, und die IP-Nummer des ESP32 wird ermittelt:

```
var ipaddr=location.hostname + “:81”;
document.getElementById(’ip’).innerHTML=ipaddr;
```

Hier wird die Portnummer des Websockets des ESP32 angehängt und sofort der Text des Tags mit der Bezeichnung `ip` aktualisiert. Der Platz für die Anzeige des Textes wird durch die etwas langatmige Konstruktion in der zweiten Zeile erreicht, in der sich `document` auf die Webseite selbst bezieht. Der Teil nach dem Punkt ermöglicht den Zugriff auf das benannte Element `ip`, und `innerHTML` bezieht sich auf den angezeigten Text, der geändert wird, um `ipaddr` anzuzeigen. Diese Konstruktion wird ausgiebig verwendet, um auf benannte Elemente zuzugreifen und deren Eigenschaften zu ändern. Die Funktionen `toStatus()` und `toReply()` folgen dieser Vorlage, um Text in die Statuszeilen am unteren Rand der Webseite zu kopieren.
Wenn die Adresse des Websockets auf dem ESP32 bekannt ist, wird er mit `new WebSocket()` geöffnet und Callback-Funktionen an die Ereignisse `onopen, onclose` und einige andere angehängt. Oft wird auch nur eine kurze Nachricht mit `console.log()` an die JavaScript-Konsole gesendet, die über die Entwicklertools des Browsers oder mit `Strg-Umschalt-I` in Chromium oder `Strg-Umschalt-K` in Firefox zugänglich ist.

```
var websock = new WebSocket(’ws://’ + ipaddr);
websock.onopen = function(evt)
    ;
```

Die interessanteste Callback-Funktion reagiert auf ankommende Nachrichten vom ESP32. Die leicht verkürzte Funktion `websock.onmessge()` interpretiert zunächst die ankommende JSON-formatierte Nachricht, die in `event.data` gespeichert ist, und legt die Befehl-Wert-Paare in der Struktur `stuff` ab. Wenn `stuff` den Befehl `INFO` enthält, wird der zugehörige Wert in `val` gespeichert und mit `toReply()` auf der Webseite angezeigt. Wenn der Befehl `WF0` lautet, enthält er die Wellenform mit den Impedanzdaten. Die Anzahl der ankommenden Datenpunkte wird mit `val.length` bestimmt und zur Anpassung der horizontalen Achse verwendet, um die verfügbaren 1024 Pixel zu füllen. Dann wird die Eigenschaft `d` des Pfades initialisiert, und es werden Punkte angehängt, einer für jeden Eintrag in `val`. Da Pixel (0,0) oben links liegt, muss die vertikale Position durch Setzen von Pixel `512-val[i]` invertiert werden. Schließlich wird die schwarze Referenzlinie mit dem Kalibrierungswiderstand angezeigt.

```
websock.onmessage=function(event) {
var stuff=JSON.parse(event.data);
var val=stuff[“INFO”]; // info
if (val != undefined)
var val=stuff[“WF0”]; // waveform0
if (val != undefined) {
nstep=Math.floor(0.5+1024/val.length)
pixmax=nstep*val.length;
var d=”M0 511”;
for (i=0; i<val.length; i++)

document.getElementById(’trace0’).setAttribute(’d’,d);
d=”M0 480 L1024 480”;
```

```
document.getElementById('trace1').setAttribute('d',d);
}
}
```

Der Rest des JavaScripts besteht hauptsächlich aus Callback-Funktionen für Schaltflächen und Menüs. Die Funktion `sweep()`, die durch die entsprechende Schaltfläche ausgelöst wird, lautet:

```
function sweep() {
websock.send(JSON.stringify
     ({ "cmd" : "SWEEP", "val" : -1 }));
}
```

Die Funktion `JSON.stringify()` packt ihr Argument in eine richtig formatierte Nachricht und `websocket.send()` sendet sie an den ESP32. Alle anderen Callback-Funktionen folgen dem gleichen Muster. Kurz vor dem Ende des JavaScript-Abschnitts wird `showCoordinates()` definiert, um die Frequenz und Impedanz anzuzeigen, die dem angeklickten Pixel im Diagrammbereich entspricht. Diese Funktion erhält ein Callback für ein `Mousedown`-Ereignis, indem ein `addEventListener()` an `displayarea` angehängt wird:

```
document.getElementById("displayarea")
 .addEventListener('mousedown', showCoordinates, false);
```

Mit dieser Funktion lassen sich Frequenzen und die entsprechenden Impedanzen direkt aus dem angezeigten Diagramm bestimmen. In Bild 5 ist zu sehen, dass das Resonanzverhalten eines parallelen RLC-Schaltkreises, den wir als Prüfling verwendet haben, eine Resonanzspitze bei 24 kHz aufweist, bei der die Impedanz 16 kΩ übersteigt.
Die Firmware und andere Dateien können kostenlos von [7] heruntergeladen werden. Jetzt ist der Impedanzanalysator autark in dem Sinne, dass kein Steuerprogramm erforderlich ist. Die gesamte Kommunikation findet zwischen dem ESP32 und einem Browser statt, auch wenn dieser auf einem Smartphone läuft.

RG — 230039-02

WEBLINKS

[1] ESP32 Technical Reference Manual (Version 4.7): https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
[2] Cosinuswellen-Generator mit ESP32: https://github.com/krzychb/dac-cosine
[3] Arduino-Website: https://www.arduino.cc
[4] Arduino-Support ESP32: https://github.com/espressif/arduino-esp32
[5] V. Ziemann, „A Hands-on Course in Sensors Using the Arduino and Raspberry Pi", 2. Auflage, CRC Press, Boca Raton, 2023:
[6] ESP32fs-Plugin: https://github.com/me-no-dev/arduino-esp32fs-plugin
[7] Quellcode für dieses Projekt auf GitHub: https://tinyurl.com/4awvvemc

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Senden Sie uns eine E-Mail unter redaktion@elektor.de.

Über den Autor

Volker Ziemanns' Interesse an Elektronik begann mit dem 40-W-Edwin-Verstärker (Elektor, Mitte der 70er Jahre), aber er wurde abgelenkt und studierte Physik und arbeitete seitdem mit Teilchenbeschleunigern - am SLAC in den USA, am CERN in Genf und jetzt in Uppsala, Schweden. Da die Elektronik eine wichtige Rolle bei der Steuerung und Datenerfassung in Beschleunigern spielt, war sein frühes Interesse während seiner gesamten Laufbahn von Nutzen. Er lehrt jetzt an der Universität Uppsala. Eines seiner Bücher über Datenerfassung mit Arduino und Raspberry Pi berührt das Thema dieses Artikels.

Passende Produkte

- **ESP32-DevKitC-32D** SKU 18701: www.elektor.de/18701
- **1-Kanal Oszilloskop mit Multimeter OWON HDS1021M-N** SKU 18778: www.elektor.de/18778

Das besondere Projekt

Ermutigung zum Heimwerken

Bild 1. Ruuds Hobbylabor im aufgeräumten Zustand.

Von Ruud van der Meer (Niederlande) und Eric Bogers (Elektor)

Nach Erreichen des Rentenalters am Ende einer langen und erfolgreichen Karriere in der Elektronikbranche widmen sich viele Menschen etwas völlig anderem - wie dem Anbau von Tomaten oder der Zucht von Meerschweinchen - solange es nur nichts mit Elektronik zu tun hat. Ruud van der Meer aus Roelofarendsveen in den Niederlanden ist eine Ausnahme. Nach mehr als 40 Jahren bei Siemens ist sein Enthusiasmus immer noch ungebremst.

Es wird zu wenig DIY-Elektronik gebaut.

Nach der Berufsschule und einem Studium der Mess- und Regeltechnik sowie einer Weiterbildung zum Elektroniker begann Ruud van der Meer im Jahr 1970 bei Siemens in der Abteilung für Mess- und Regeltechnik, Messgeräte und Kalibrierung. Ruuds eigene Worte:

„Es gab viele Möglichkeiten. In kurzer Zeit entwickelte ich eine Reihe von Projekten, darunter LED-Anzeigen, PI-Regler und Messübertrager, und durch die Entwicklung von Testgeräten und Prüfsystemen konnte ich viele Arbeitsabläufe verbessern. Nach einigen Jahren wurde ich Teamleiter und entwickelte (zusammen mit dem Siemens Hobby Computer Club) unser eigenes PC-System namens SUMO80."

Ruud merkte bald, dass einige Leute ein wenig Hilfe brauchten, um sich in den neuen Technologien zurechtzufinden, und so begann er, im Unternehmen und im Regionalen Schulungszentrum Kurse in den Bereichen Elektrotechnik und Elektronik, digitale Methoden, Kommunikationstechnik und Mechatronik zu geben.

„Irgendwann suchte ich eine neue Herausforderung, und die ergab sich, als ich gebeten wurde, die Leitung der betrieblichen Ausbildung bei Siemens zu übernehmen. Damals gab es etwa 60 Studenten im Trainingsprogramm Elektrotechnik. Eine meiner Aufgaben (neben dem theoretischen Teil des Programms) war es, den Zustrom neuer Mitarbeiter in das Unternehmen zu verbessern, der über die Jahre hinweg zurückgegangen war. Das habe ich unter anderem dadurch erreicht, dass ich in Zusammenarbeit mit Dozenten der TU Delft das Ausbildungsprogramm im letzten Ausbildungsjahr um die Bereiche Elektronik, SPS- und Kommunikationstechnik sowie Mechatronik erweitert habe."

Die betriebliche Ausbildung bei Siemens wurde 2005 eingestellt, so dass Ruud sich nach einer Stelle in einem anderen Bereich (aber immer noch bei Siemens) umsehen

Bild 3. Der Elektronikbereich mit den Test- und Messgeräten.

musste. Schließlich fand er eine neue Stelle in der Abteilung, die für Gebäudeautomatisierung und Brandschutz zuständig ist.

„Um es kurz zu machen: Was die Arbeitsabläufe angeht, sah es dort aus wie in der Steinzeit, aber nach einer Weile gelang es mir, die Abteilung effizienter zu gestalten. Und dann bin ich im Jahr 2016 in den Ruhestand gegangen."

Es war also wieder Zeit für etwas Neues. Ruud wurde der erste Energiecoach der Gemeinde Kaag en Braassem und hat in der Zwischenzeit viele Empfehlungen zur Nachhaltigkeit gegeben. Unter anderem deshalb hat er viele Hausautomationssysteme auf der Basis von Arduino entwickelt.

„Nach einer Reihe von Präsentationen über Arduino beim Hobby Computer Club wurde ich gebeten, dasselbe für die Volkshochschule Leiderdorp (LVU) zu tun. Das begann vor sechs Jahren. Danach habe ich auch an den Volkshochschulen von Alphen aan de Rijn, Lisse und Hillegom Arduino-Kurse abgehalten. Inzwischen haben wir eine große Menge begeisterter Menschen um uns versammelt, und wir organisieren ein monatliches Arduino-Café und natürlich den jährlichen Welt-Arduino-Tag. Auch der Elektronik-Lötkurs stößt auf großes Interesse. Es wird zu wenig DIY-Elektronik gebaut, obwohl es mit Arduino so einfach ist."

Heimlabor

Ruud hat eine Menge Lernmaterial (Schaltpläne, Platinen und Software) für all die Kurse und Schulungen entwickelt. Das meiste davon wurde in seinem Heimlabor erstellt (und anfangs auch verschiedene Dinge für Siemens). **Bild 1** vermittelt einen Eindruck von seinem Heimlabor.

„Ich habe seit 1965 mein eigenes Heimlabor (oder zumindest einen Hobbyraum). Was Sie auf dem Foto sehen, ist der Raum, den ich seit 2005 benutze. Die PC-Ecke ist links, die Elektronik- und Messtechnik-Ecke ist in der Mitte und der Teil für mechanische Arbeiten ist rechts. Dort habe ich zum Beispiel meinen mechanischen Putzroboter entwickelt, als praktische Übung in Mechatronik (**Bild 2**).

Ich habe eine Menge Test- und Messgeräte entwickelt. Mein selbstgebautes Röhrenoszilloskop habe ich leider nicht mehr, aber es sind noch viele Exemplare erhalten, die von Kollegen (nach-)gebaut wurden. **Bild 3** gibt einen Eindruck davon."

Eines meiner jüngsten Projekte basiert auf dem bestehenden DIY-Taschenrechner von KKmoon. Meiner Meinung nach hatte er in seiner ursprünglichen Form nicht genügend Funktionen, und es war schwierig, ihn neu zu programmieren. Da es sich jedoch um ein schönes Hardware-Design handelt, habe ich ein neues Hirn dafür entwickelt - natürlich auf der Basis eines Arduino-Boards. Wir bauen jetzt mit einer Gruppe von Arduino-Schülern eine Reihe dieser DIY-Taschenrechner. Die Funktionen (neben den Grundfunktionen eines Taschenrechners) richten sich an Menschen, die oft mit Programmieren oder anderen technischen Aktivitäten zu tun haben. **Bild 4** zeigt das Äußere und einen Blick ins Innere."

Dieser Arduino-Taschenrechner wird in einer der nächsten Ausgaben des Elektor-Magazins ausführlicher besprochen werden.

SG — 230035-02

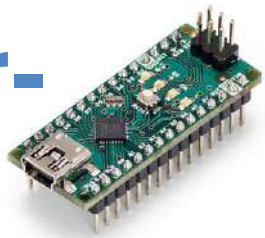
Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Schicken Sie eine E-Mail an die Elektor-Redaktion unter redaktion@elektor.de.

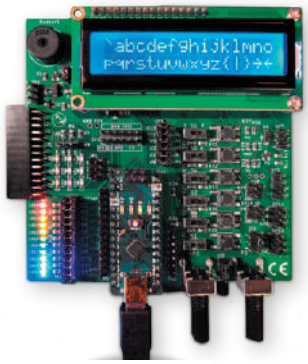
Bild 2. Der Reinigungsroboter.

Bild 4. Der Multitaschenrechner: außen und innen.

Mikrocontroller-Praxiskurs für Arduino-Einsteiger mit Trainingsboard



All-in-One-Trainingshardware für den Mikrocontroller-Kurs



Von Wolfgang Trampert (Deutschland) und Jan Buiting (Elektor)

Der Zusammenbau von kleinen Arduino-Erweiterungsschaltungen besteht normalerweise darin, Teile auf ein Breadboard zu stecken und Verbindungen mit einem Haufen bunter Jumperkabel herzustellen. Dieses ganze Anschließen und Ausprobieren nimmt jedoch oft mehr Zeit in Anspruch als das Schreiben des Programms. Auch das Anordnen der Bauteile, zum Beispiel für ein Lauflicht mit elf LEDs und ebenso vielen Vorwiderständen, ist fehleranfällig und kostet wertvolle Stunden bei der Fehlersuche in der Hardware! In solchen Fällen ist ein spezielles Trainingsboard wie das neue „MCCAB“ von Elektor vorteilhafter, zumal es mit einem tollen Anleitungsbuch geliefert wird.

Anmerkung der Redaktion. *Dieser Artikel ist ein Auszug aus dem 455-seitigen Elektor-Handbuch* Mikrocontroller-Praxiskurs für Arduino-Einsteiger *(Elektor, 2023). Der Auszug wurde so formatiert und leicht bearbeitet, dass er den redaktionellen Standards und dem Seitenlayout der Zeitschrift Elektor entspricht. Der Autor und der Redakteur haben bei der Bearbeitung versucht, „Abhängigkeiten“ zum Buchtext zu vermeiden und sind bei Rückfragen gerne behilflich. Kontaktinformationen finden Sie im* **Kasten Fragen oder Kommentare?**

Das „MCCAB“, das nach dem englischen Titel des Kurses „Microcontroller Crash Course for Arduino Beginners“ benannt wurde, ist ein Trainingsboard für den Arduino Nano, das speziell vom Autor entwickelt wurde und exklusiv für Elektor hergestellt wird, um den dicken, grünen „Mikrocontroller-Praxiskurs für Arduino-Einsteiger“ zu unterstützen. Das MCCAB-Board, ein Arduino-Nano-Board und der Leitfaden (in englischer oder deutscher Sprache) und sind als Produktpaket im Elektor Store [1] erhältlich.

Auf dem Trainingsboard - im Folgenden kurzerhand MCCAB genannt - ist ein umfangreiches Sortiment an Zubehörkomponenten bereits mit dem aufgesteckten Mikrocontrollermodul verbunden, weitere Bauteile können über einfache Jumperkabel angeschlossen werden. Dazu gehören Schalter/Taster, LEDs, Potentiometer, Summer, ein LCD, Schnittstellen und Pegelwandler für serielle Verbindungen sowie Treiberstufen für externe Bauteile und Bausteine.

Auch ganze externe Module können über eine Buchse in das MCCAB gesteckt oder über die seriellen Schnittstellen mit dem Mikrocontroller auf dem Board verbunden werden. Damit entfällt der mühsame Aufbau von Experimentierschaltungen, und Sie können sich auf das Wesentliche konzentrieren - nämlich Ihre Software, das „Programm“. Die in den Übungen erstellten Programme werden einfach in den ATmega328P-Mikrocontroller auf dem MCCAB geladen, wo sie dann ausgeführt werden.

Einige der 32 Pins des ATmega328P-Mikrocontrollers werden für seine Stromversorgung, für den Anschluss

eines Quarzes an den eingebauten Oszillator zur Takterzeugung oder als RESET-Eingang für den Drucktaster des an das MCCAB angeschlossenen Mikrocontrollermoduls verwendet. Durch dessen Betätigung wird der Mikrocontroller auf einen definierten Ausgangszustand zurückgesetzt und das Programm neu gestartet. Der Großteil der Anschlüsse steht jedoch als Allzweck-Ein- und Ausgänge zur Verfügung, die der Mikrocontroller zur Verbindung mit der Außenwelt nutzen kann. Auf dem MCCAB sind diese Anschlüsse - bis auf wenige Ausnahmen, die für interne Zwecke reserviert sind - auf Steckerleisten herausgeführt.

Bild 1 zeigt eine Ansicht des MCCAB mit seinen farblich markierten Funktionsblöcken. Die Bedienungsanleitung für das MCCAB mit einer ausführlichen Beschreibung aller Komponenten kann kostenlos von der Elektor-Webseite [2] heruntergeladen werden.

Die Funktionsblöcke auf dem MCCAB

Auf dem MCCAB lassen sich folgende Funktionsblöcke und Module unterscheiden (siehe Bild 1).

(1) **Mikrocontroller-Modul Arduino Nano** mit *RESET*-Taster (Pfeil 1a), Leuchtdiode „L“ (Pfeil 1b) und Mini-USB-Buchse für den Anschluss an den PC des Anwenders.

(2) Anschlussleisten SV5 und SV6 **für die Mikrocontroller-Ein- / Ausgänge**. Über diese beiden Stiftleisten können mit Hilfe von Dupont-Kabeln (Jumperkabel) die GPIOs (General Purpose Inputs / Outputs) des Mikrocontrollers mit den internen Komponenten des Trainingsboards oder mit externen Modulen verbunden werden.

(3) **11 LEDs** LD10...LD20 (Zustandsanzeigen für die Ein-/Ausgänge D2...D12 des Mikrocontrollers). Jede LED kann über eine Steckbrücke (Jumper) auf der Pfostenleiste JP6 mit dem zugeordneten GPIO D2...D12 verbunden werden.

(4) **3×3-LED-Matrix** LD1...LD9 (neun rote Leuchtdioden). Die Spaltenleitungen sind fest mit den Ausgängen D6...D8 des Mikrocontrollers verbunden, die Zeilenleitungen können durch Jumper auf der Pfostenleiste JP1 mit den GPIOs D3...D5 verbunden werden.

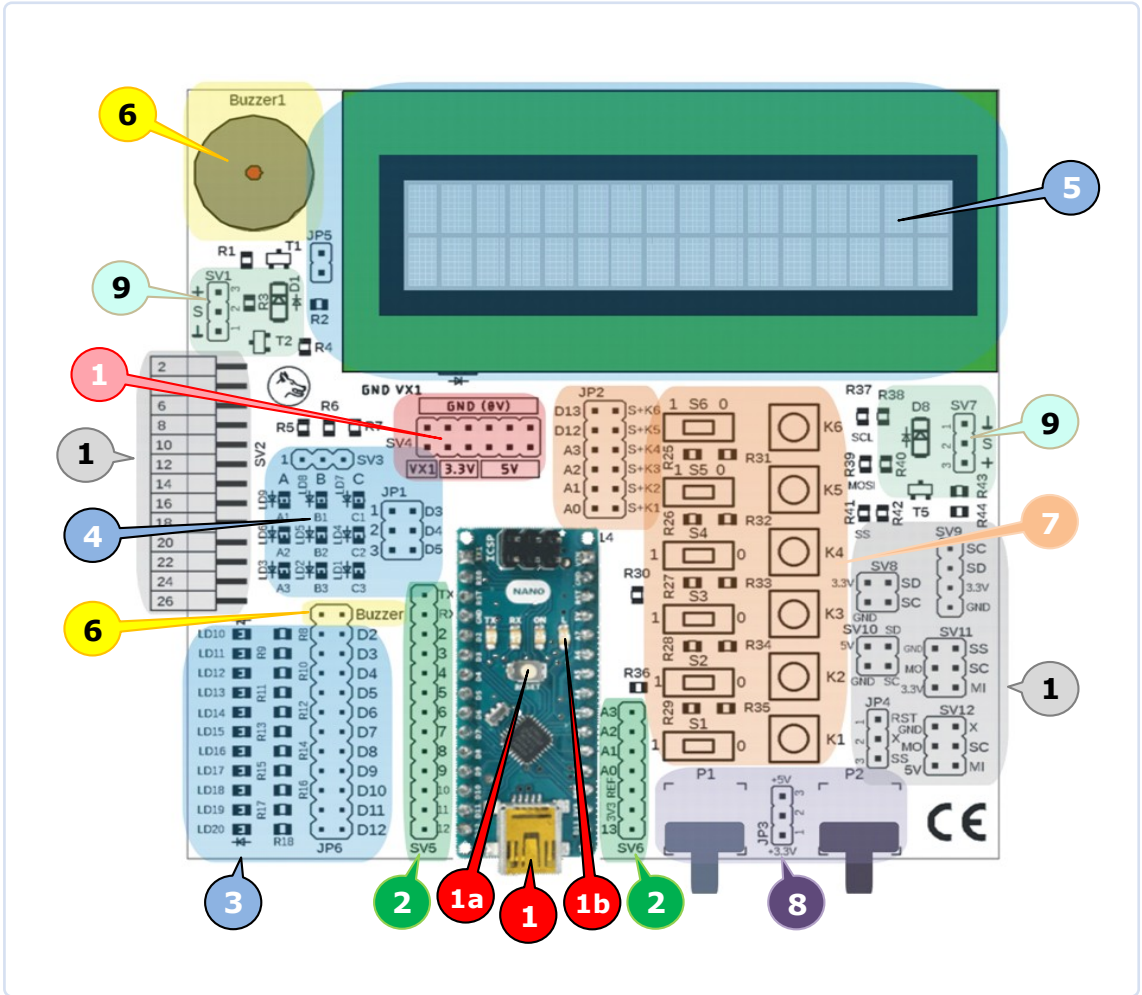


Bild 1. Funktionsschema des Elektor Arduino Nano Training Board MCCAB.

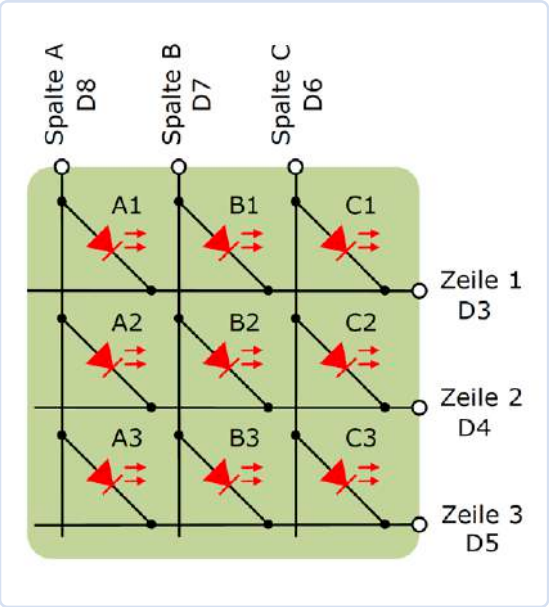


Bild 2. 3×3-LED-Matrix.

Achtung! Wenn die Zeilenleitungen der 3×3-LED-Matrix entweder über Jumper auf der Pfostenleiste JP1 mit den GPIOs D3…D5 oder über Dupont-Kabel mit anderen GPIOs des Mikrocontrollers verbunden sind, dürfen diese Zeilenleitungen sowie die Spaltenleitungen D6…D8 in einem Sketch keinesfalls für andere Aufgaben verwendet werden. Eine Doppelbelegung der Matrix-GPIOs würde zu Fehlfunktionen oder sogar zur Beschädigung des MCCAB-Trainingsboards führen!

(5) **LC-Display** mit 2×16 Zeichen, über den I²C-Bus verbunden mit den Pins A4 und A5 des Mikrocontrollers. Die Betriebsspannung des LCDs kann in Übungen, in denen die Anzeige nicht verwendet wird, durch Ziehen der Steckbrücke JP5 abgeschaltet werden.
(6) **Piezo-Summer** Buzzer1 kann über einen Jumper auf der Position „Buzzer" der Stiftleiste JP6 mit GPIO D9 verbunden werden.
(7) **6 Schiebeschalter S1…S6**, parallelgeschaltet zu **6 Tastern K1…K6**. Sie können über Steckbrücken auf der Pfostenleiste JP2 mit den Eingängen A0…A3, D12 und D13 verbunden werden.
(8) **Potentiometer P1 und P2.** Ihre Schleifer sind verbunden mit den Analog-Eingangspins A6 und A7 des Mikrocontrollers. Über die Stiftleiste JP3 kann wahlweise die Spannung 3,3 V oder 5 V an die Potentiometer gelegt werden.

Achtung! Die Pins A6 und A7 des ATmega328P sind aufgrund der internen Chip-Architektur als analoge Eingänge festgelegt. Ihre Konfiguration mit der Funktion `pinMode()` ist nicht gestattet und kann zu einem Fehlverhalten des Programms führen.

(9) Die Stiftleisten SV1 und SV7, **Schaltausgänge für externe Geräte**.
(10) Stiftleisten zum seriellen **Anschluss externer SPI- und I2C-Module**.
(11) Steckerleiste SV2 mit 2×13 Stiften zum **Anschluss externer SPI- und I²C-Module**.
(12) Stiftleiste SV4, **Verteiler für die Betriebsspannungen** des Boards. Die Spannungen können über Dupont-Kabel mit internen Komponenten des Trainingsboards oder externen Modulen verbunden werden.

Auf die beiden etwas komplexeren Funktionseinheiten in Bild 1, die 3×3-LED-Matrix und das LC-Display wollen wir hier noch etwas ausführlicher eingehen, für alle anderen sei auf die oben erwähnte ausführliche Beschreibung in der Bedienungsanleitung des MCCAB [2] verwiesen.

Info: Zur Steuerung der 3×3-LED-Matrix, der LEDs LD10…LD20, der Taster K1…K6 und Schalter S1…S6 sowie des Summers Buzzer1 steht die Bibliothek *MCCAB_Lib* zur Verfügung. Die Bibliothek kann von MCCAB-Besitzern kostenlos heruntergeladen und in die Arduino-IDE integriert werden.

Zur Ansteuerung des LCDs verwenden wir die Bibliothek *LiquidCrystal_I2C*, die ebenfalls kostenlos aus dem Internet heruntergeladen und in die Arduino-IDE eingefügt werden kann.

Die 3×3-LED-Matrix

Auf dem *MCCAB* befinden sich neun Leuchtdioden, die in Form einer Matrix angeordnet sind (Bild 1). **Bild 2** zeigt ihre prinzipielle Beschaltung.
Die Matrix besteht aus drei Spalten und drei Zeilen. Die Spalten sind mit A, B und C bezeichnet, die Zeilen sind von 1 bis 3 durchnummeriert. An jedem der neun Kreuzungspunkte einer Zeile mit einer Spalte ist eine Leuchtdiode platziert, deren Anode mit der Spaltenleitung und deren Kathode mit der Zeilenleitung verbunden ist. Die neun LEDs sind nach den an sie angeschlossenen Spalten und Zeilen bezeichnet. So liegt LED B2 an Spalte B und Zeile 2. Damit eine LED leuchtet, muss ihre Spaltenleitung auf logisch 1 (+5 V) und Ihre Zeilenleitung auf logisch 0 (0 V) liegen.
Der Bedienungsanleitung für das *MCCAB* ist zu entnehmen, dass die Spaltenleitungen fest mit den GPIOs D6…D8 des Mikrocontrollers verbunden sind. Die Zeilenleitungen können über Jumper auf der Pfostenleiste JP1 an die GPIOs D3…D5 des Mikrocontrollers angeschlossen werden.
Wenn die Zeilenleitungen der 3×3-LED-Matrix entweder über Jumper auf der Stiftleiste JP1 mit den GPIOs D3…D5 oder über Dupont-Kabel mit anderen GPIOs des Mikrocontrollers verbunden sind, dürfen diese Zeilenleitungen sowie die Spaltenleitungen D6…D8 in einem Sketch nicht für andere Aufgaben verwendet werden. Eine derartige Beschaltung der Matrix-

Bild 3. Periodische schrittweise Aktivierung der drei Zeilen, aus denen die Matrix besteht.

GPIOs würde zu Fehlfunktionen oder schlimmstenfalls sogar zur Beschädigung des *MCCAB* führen! Wenn die Matrix in einem Sketch nicht verwendet wird, sollten die Jumper auf dem *MCCAB-Header* entfernt werden.

Vorteile der Matrix-Anordnung

LED-Matrizen kennen wir zum Beispiel aus der Bandenwerbung in Fußballstadien, wo farbige Hochleistungs-LEDs in Matrix-Anordnung zur Erzeugung bewegter Bilder zum Einsatz kommen.
Würden wir neun Leuchtdioden einzeln steuern, so wären dazu neun GPIOs des Mikrocontrollers erforderlich. Durch die Anordnung der LEDs in Form einer Matrix benötigen wir dagegen nur sechs GPIOs. Je mehr LEDs angesteuert werden müssen, umso mehr tritt der Vorteil der Matrixsteuerung zutage: Mit einer aus acht Spalten und acht Zeilen bestehenden Matrix kann man 64 LEDs steuern. Gegenüber der Einzelansteuerung können so 48 GPIOs eingespart werden!

Ansteuerung der Matrix im Multiplex-Modus

Bild 2 kann man entnehmen, dass jede Spaltenleitung und jede Zeilenleitung der Matrix auf dem *MCCAB* mit jeweils drei LEDs verbunden ist. Somit würde die gleichzeitige Ansteuerung aller Zeilen und Spalten nicht funktionieren, da ungewollt LEDs, die eigentlich ausgeschaltet sein sollten, ebenfalls leuchten würden. Stattdessen darf – wie in **Bild 3** gezeigt – immer nur eine Zeilenleitung aktiv sein und die Spaltenleitungen müssen das Bitmuster für die drei LEDs der gerade aktivierten Zeile anlegen. Die beiden anderen Zeilenleitungen müssen während dieser Zeit offen oder durch einen angelegten HIGH-Pegel abgeschaltet sein, sodass über sie kein Strom fließen kann.
Wenn die drei Zeilen in genügend rascher Folge wie in Bild 3 zyklisch aktiviert werden, sieht der Betrachter aufgrund der Trägheit des menschlichen Auges ein stehendes Bild aller neun LEDs. Die Ansteuerung der LED-Matrix durch das Anwenderprogramm erfolgt in einer Endlosschleife, in welcher zyklisch eine der drei Zeilen 1, 2 oder 3 auf LOW-Potential gelegt wird, während die beiden anderen Zeilen auf HIGH-Pegel liegen. Die Spaltenanschlüsse aller Leuchtdioden, die in der gerade aktiven Zeile leuchten sollen, werden auf HIGH-Pegel gelegt. Die Spaltenanschlüsse der LEDs in der aktiven Zeile, die nicht leuchten sollen, liegen dabei auf LOW-Potenzial.
Um beispielsweise die beiden LEDs A3 und C3 leuchten zu lassen, müssen Zeile 3 auf Low-Pegel und die Spalten A und C auf High-Pegel liegen, während die beiden Zeilenleitungen 1 und 2 auf High-Pegel liegen und Spalte B Low-Pegel führt.

Die Flüssigkristall-Anzeige (LCD)

Das Elektor-MCCAB ist mit einem LC-Display (LCD) ausgerüstet, das die Ausgabe von Texten, Zahlenwerten und sogar selbstdefinierten Sonderzeichen ermöglicht. Das eingesetzte Display verfügt über zwei Zeilen mit jeweils 16 Spalten. In jeder Spalte kann ein Zeichen dargestellt werden. Jedes Zeichen wird aus den Punkten einer 5×8-Matrix gebildet, wie in **Bild 4** zu sehen ist. Die erforderlichen Bitmuster der 5×8-Punktmatrix für jedes einzelne Zeichen sind im LCD-Controller gemäß der ASCII-Tabelle gespeichert. Nun ist der ASCII-Code ein 7-Bit-Code, während unser Mikrocontroller ATmega328P aber eine „8-Bit-Maschine“ ist, was bedeutet, dass er ein Byte in einem Arbeitsschritt verarbeitet und auch seine Daten im Byte-Format speichert. Die Entwickler des Display-Controllers HD44780 (der in dieser Kategorie von Anzeigen Standard ist und sich auf fast allen LCD-Modulen befindet) wollten das achte Bit des Bytes aber nicht ungenutzt lassen und haben den ASCII-Zeichensatz um weitere 128 Zeichencodes erweitert (wobei das achte Bit auf logisch 1 liegt). Daher sind bei unserem LCD auch die Zeichencodes 128…255 (mehr oder weniger durchgehend) mit Zeichen belegt.

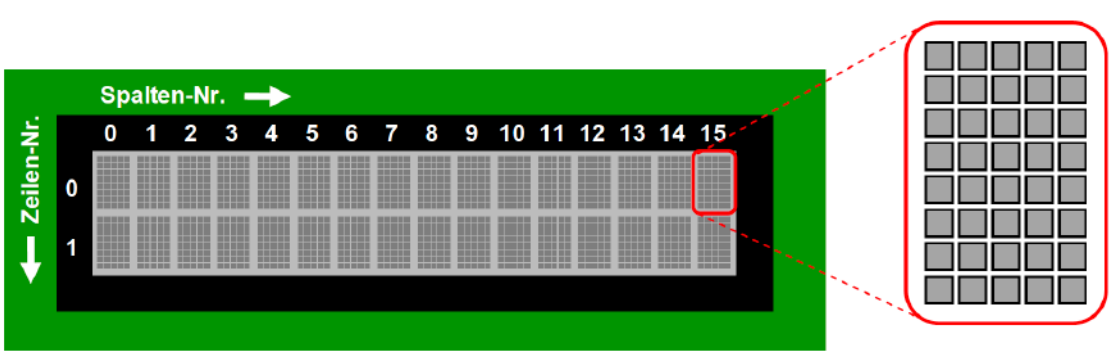


Bild 4. Illustration des LC-Displays auf dem MCCAB mit seinen 16×2 anzeigbaren Zeichen.

Tabelle 1:
Speicherbelegung für die Zeichen bei beiden HD44780-Varianten.

Zeichen-Code	ROM-Code A00	ROM-Code A02
0–7	benutzerdefinierte Zeichen	benutzerdefinierte Zeichen
8–15	benutzerdefinierte Zeichen	benutzerdefinierte Zeichen
16–31	(keine Anzeige)	sichtbare Zeichen
32–127	sichtbare Zeichen	sichtbare Zeichen
128–159	(keine Anzeige)	sichtbare Zeichen
160–255	sichtbare Zeichen	sichtbare Zeichen

Der ASCII-Code weist neben den 95 druckbaren Zeichen mit den Codes 32...126 auch 33 Steuerzeichen mit den ASCII-Codes 0...31 und 127 auf, die nicht druckbar sind. Das Display wertet diese Steuerzeichen nicht zu Steuerungszwecken aus. Der Display-Controller HD44780 ist in zwei Versionen lieferbar: Mit dem ROM-Code A00 oder alternativ mit dem ROM-Code A02. Die Version A00 zeigt bei den Codes 16...31 sowie 128...159 leere Punktmatrizen an, während die Version A02 auch diese Adressräume zur (Sonder-)Zeichenausgabe nutzt (siehe **Tabelle 1**). Welche der beiden ROM-Varianten in dem LC-Display des MCCAB verbaut ist, kann nicht vorhergesagt werden, da der Hersteller des LCD-Moduls diese Auswahl trifft.
Für den Anwender besteht die Möglichkeit, bis zu acht eigene Sonderzeichen selbst zu definieren und auf dem LCD anzuzeigen. Diesen acht Sonderzeichen sind die ASCII-Codes 0...7 zugewiesen. Wahlweise können die gleichen acht Sonderzeichen auch über die ASCII-Codes 8...15 angesprochen werden.

Bild 4 zeigt die Nummerierung der Zeilen und Spalten des Displays, die in beiden Fällen mit 0 beginnt. Durch die Angabe dieser Koordinaten kann ein Zeichen gezielt an eine bestimmte Stelle des Displays geschrieben werden. Zu diesem Zweck verfügt das LCD über einen Cursor, der die Position des zu schreibenden Zeichens festlegt. Die in diesem Buch beschriebene LCD-Bibliothek enthält Methoden zur Positionierung dieses Cursors.

Das Display kann zwar in jeder Zeile nur 16 Zeichen anzeigen, der Speicher, in dem die Zeichen im Display-Controller abgelegt werden, verfügt aber für jede Zeile über 40 Speicherplätze. Wie Sie in **Bild 5** sehen, besteht zwischen der letzten Display-Adresse 39 der ersten Zeile und der Anfangsadresse 64 der zweiten Zeile eine Lücke von 24 Speicherplätzen. Durch Schiebebefehle, die in der LCD-Bibliothek enthalten sind, kann der im Display sichtbare Speicherbereich wie ein Fenster über den gesamten Speicherbereich geschoben werden.
In Bild 5 sind drei Ebenen zu sehen:

- oben: das sichtbare Fenster des Displays in der Grundeinstellung ohne vorherige Schiebeoperationen.
- in der Mitte: der Displayinhalt der Grundeinstellung nach einer Schiebeoperation nach links.
- unten: der Displayinhalt der Grundeinstellung nach einem Schieberegler nach rechts.

Einstellen des LCD-Kontrasts

Im Auslieferungszustand des *MCCAB* ist kein bestimmter Kontrast für das Display eingestellt. Aus diesem Grund und weil sich der Kontrast des Displays durch die Umgebungsbedingungen (Temperatur) oder durch Alterung verändern kann, besitzt das LCD zur Einstellung des Kontrastes ein Trimmpotentiometer, das von der Unterseite des Trainingsboards zugänglich und mit einem Pfeil markiert ist. So kann der Kontrast bei Bedarf mit einem kleinen Schraubendreher jederzeit nachjustiert werden.

Hinweis: Wenn bei der ersten Benutzung des LCDs nach der Textausgabe keine Zeichen auf dem Display zu sehen sind, liegt das höchstwahrscheinlich an einer fehlenden/falschen Kontrasteinstellung des LCDs!

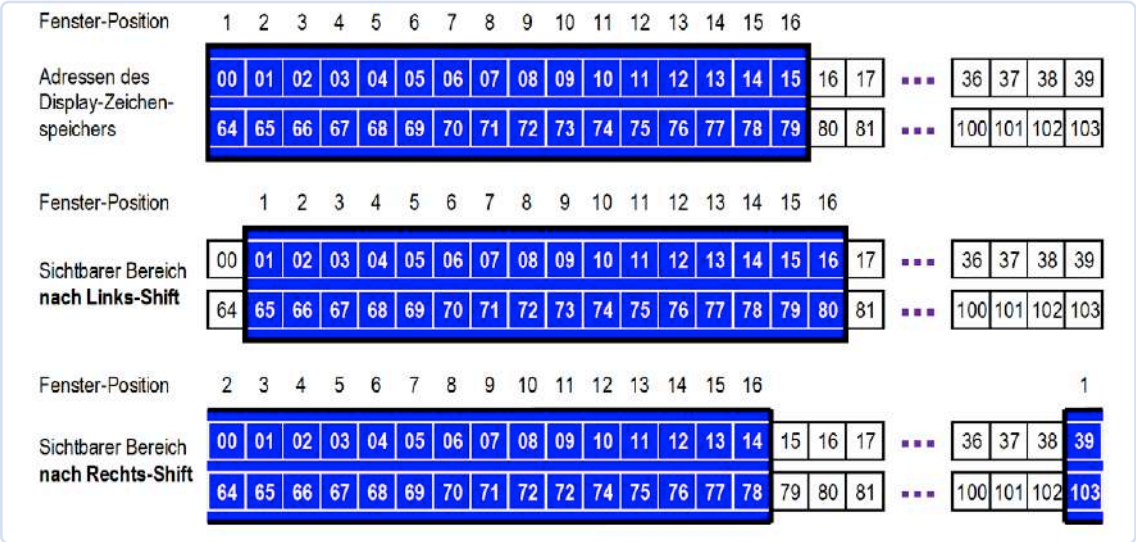


Bild 5. Der sichtbare Inhalt des Displays vor und nach der Verschiebung.

Datenübertragung vom Mikrocontroller zum LCD

Auf jedem LCD-Modul befindet sich ein Display-Controller HD44780, der die vom Mikrocontroller gesendeten (ASCII-)Codes der Zeichen über ein Interface entgegennimmt, aus den Codes die entsprechenden Zeichen der oben beschriebenen 5×8-Punktmatrix generiert und diese anzeigt.

Der LCD-Controller HD44780 verfügt nur über eine parallele Schnittstelle zum Empfang der darzustellenden Daten, das heißt, der Mikrocontroller schreibt ein aus den acht Bits D0…D7 bestehendes Datenbyte mit Hilfe der drei Steuersignale RS, RW, E in das Eingangsregister des Display-Controllers, wie in **Bild 6** gezeigt.

Dieses Verfahren der parallelen Datenübertragung würde die Ressourcen auf dem MCCAB erheblich einschränken, denn von den insgesamt 16 frei verfügbaren Pins des Mikrocontrollers ATmega328P wären ja bereits elf alleine durch das LCD belegt!

Auch die zweite Möglichkeit der Datenübertragung zum LCD-Controller HD44780, bei der die acht Datenbits in zwei Paketen zu je vier Bits nacheinander übertragen werden, bringt noch keine wirkliche Abhilfe, denn auch in diesem Fall wären immerhin noch sieben Pins des Mikrocontrollers durch das LCD gebunden, da die drei Steuersignale RS, RW, E auch in diesem Fall benötigt werden.

Aus diesem Grund wird das LC-Display auf dem MCCAB über den I²C-Bus, einen synchronen seriellen Bus gesteuert, der nur aus einer Datenleitung SDA und einer Taktleitung SCL besteht und die Daten nacheinander Bit für Bit überträgt. Der Datenverkehr über die I²C-Schnittstelle erfolgt dabei über die beiden Leitungen A4 (SDA) und A5 (SCL) des Mikrocontrollers ATmega328P (siehe **Bild 7**).

Ein zusätzlicher Adapter an der Unterseite des LCD-Moduls setzt die I²C -Signale in die parallelen Signale um. Dabei nutzt der Adapter das oben erwähnte Verfahren, nacheinander zweimal vier Bits über die dieselben Datenleitungen D4…D7 zu übertragen.

Da die beiden Leitungen A4 und A5 des Mikrocontrollers auf dem MCCAB ohnehin für die I²C-Schnittstelle reserviert sind, gehen bei dieser Art der Datenübertragung zum LCD keinerlei Ressourcen verloren, denn an den I²C-Bus können im Prinzip mehrere Teilnehmer gleichzeitig angeschlossen werden. Da jeder an den Bus angeschlossene Teilnehmer seine eigene I²C-Adresse hat, fühlt dieser sich nur dann angesprochen, wenn ein Datenpaket mit seiner Adresse eintrifft.

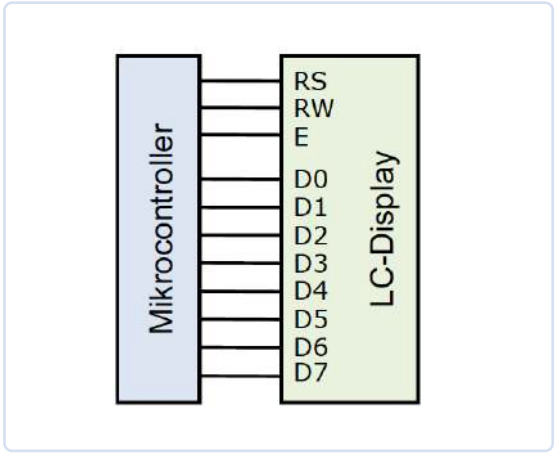


Bild 6. LCD-Steuerung mit acht Datenbits und drei Steuerleitungen.

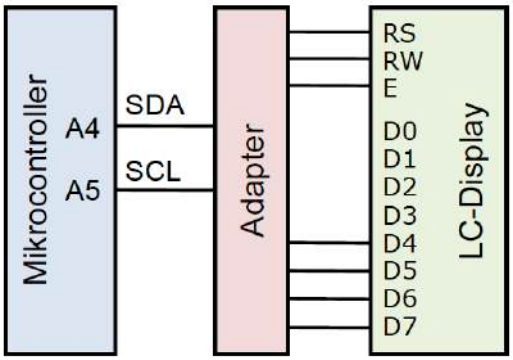


Bild 7. LCD-Ansteuerung über den I²C-Bus statt parallel (vergleiche Bild 6).

Das LC-Display auf dem MCCAB hat im Normalfall die I²C-Adresse 0x27. Falls die Adresse herstellerbedingt einmal abweichen sollte, ist dies auf dem Display angegeben.

RG – 230215-02

Haben Sie Fragen oder Kommentare?

Wenn Sie technische Fragen zu diesem Artikel haben, wenden Sie sich bitte an Elektor unter der E-Mail-Adresse redaktion@elektor.de.

Passendes Produkt

- **Mikrocontroller-Praxiskurs für Arduino-Einsteiger (Bundle)**
  1 × Buch, kartoniert: Mikrocontroller-Praxiskurs für Arduino-Einsteiger
  1 × MCCAB-Trainingsboard für den Arduino Nano
  1 × Arduino Nano
  In Deutsch: https://elektor.de/20412
  In Englisch: https://elektor.com/20440

WEBLINKS

[1] Elektor Arduino Nano Trainingsboard MCCAB und Begleitbuch: https://elektor.de/20412
[2] Bedienungsanleitung für das Elektor Arduino Nano Trainingsboard MCCAB: https://elektor.de/20412

Quelle: Shutterstock / Metamorworks

Aus dem Leben gegriffen

Moderner Luddismus

Von Ilse Joostens (Belgien)

Es stimmt, dass die Leistung der Hardware ständig zunimmt, was auch die Entwicklung immer besserer Softwareanwendungen ermöglicht. Als ich vor etwa 20 Jahren als Systemadministratorin arbeitete, war ich verrückt nach neuen Technologien und meine persönliche Mission war es, normale Computerbenutzer für neue Technologien zu begeistern. Das war, gelinde gesagt, nicht immer so einfach. Jetzt, wo ich etwas älter und vor allem weiser bin, hat sich mein Enthusiasmus für neue Technologien etwas abgekühlt, und ich habe bei den jüngsten Entwicklungen in der KI ein ungutes Gefühl.

Künstliche Dummheit

Wenn Sie in den letzten Monaten nicht in einer Höhle ohne Internetanschluss gelebt haben, werden Sie sicherlich von der neuen technologischen (R)Evolution im Bereich der künstlichen Intelligenz gehört haben, insbesondere vom Chatbot *ChatGPT* von OpenAI[1]. Das Interesse ist riesig, und Microsoft engagiert sich mit einer 10-Millionen-Dollar-Investition in dieses Startup-Unternehmen voll und ganz für generative KI. In guter Tradition will auch der Konkurrent Google nicht zurückbleiben und arbeitet derzeit mit Hochdruck an einem ähnlichen Bot mit dem wohlklingenden Namen *Bard*. Sogar Baidu ist mit von der Partie und hat mit der Entwicklung einer chinesischen Version von ChatGPT namens *Ernie* begonnen.

Als Hobbyfotografin habe ich mich schon eher für *DALL-E* [2], *StabilityAI* [3] und *Fotor* interessiert. Diese Anwendungen können Text in Bilder umwandeln und bestehende Bilder verändern oder sogar kombinieren. Es wurde zu einer Art Sucht, und ich verbrachte Stunden damit, mit ihnen zu experimentieren, bis Rauch aus meiner Tastatur aufstieg (**Bild 1**). Die Ergebnisse reichten von enttäuschend bis verblüffend und manchmal sogar erschreckend. Letzteres lässt sich vielleicht durch meine Vorliebe für das Horrorgenre erklären. Diese Technologie steht auch im Mittelpunkt der Gruselgeschichten um „Loab" [4][5], eine dämonische Frau, die sich tief in den dunkeln Tiefen der KI versteckt haben soll.

Aus Neugier habe ich natürlich auch ChatGPT ausprobiert, und auf den ersten Blick erinnerte mich der Bot an „Evarist der Computer", eine Figur aus der flämischen Jugendserie Merlina [6], die man alles fragen konnte. Die Antworten auf meine Fragen waren sprachlich in Ordnung, aber im Gegensatz zu Evarist verfehlte ChatGPT oft das Ziel und einige der Antworten waren unglaublich komisch - im Stil von Xavier De Baere (einem bekannten flämischen Fernsehkomiker – Anmerkung der Übersetzerin). Obwohl nicht die ursprüngliche Absicht, sieht es so aus, als ob der Bot auch Quellcode generieren kann. Ein einfaches Arduino-Projekt war kein Problem (**Bild 2**), aber eine komplexere Anfrage, ein C-Programm für einen AVR zu schreiben, um das Ethernet-IC W5500 von WIZnet mit einer festen IP-Adresse zu initialisieren, ohne irgendwelche Bibliotheken zu

Bild 1. KI-Kreation von Ilse Joostens.

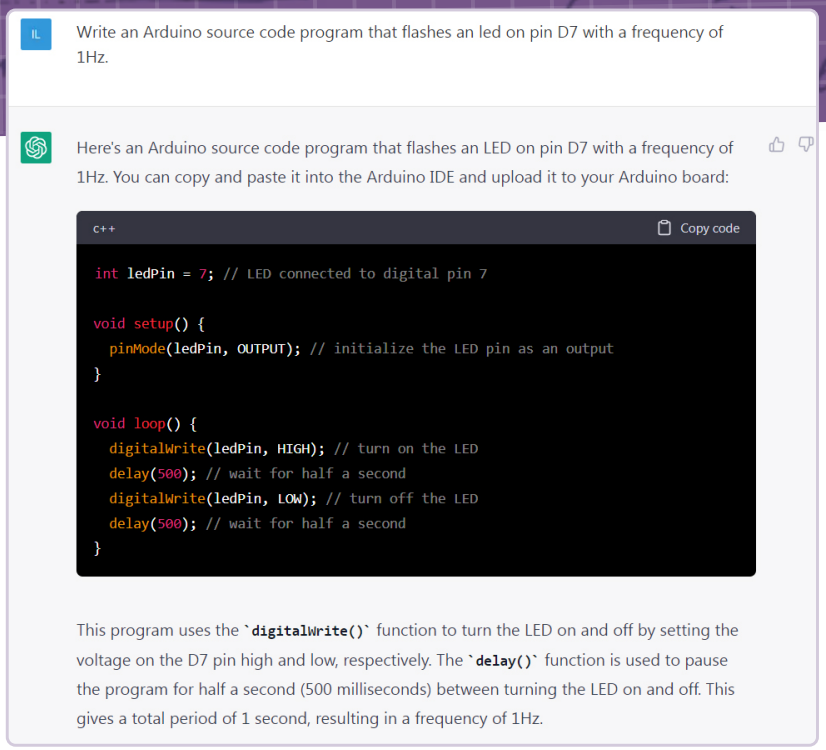


Bild 2. ChatGPT schreibt Software…

verwenden, schlug jedes Mal fehl. Vielleicht wäre die kostenpflichtige Version mit dem neuen ChatGPT-4 besser, aber das ist keine Option für eine Anwendung, für die ich (noch) keine konkrete Verwendung habe. Ich kann den Chip selbst zum Laufen bringen, und die banaleren Fragen des Alltags, wie man ein vegetarisches Drei-Gänge-Menü mit einem Schokoladendessert zaubert oder was man alles mit Kleinkindern basteln kann, interessieren mich nicht besonders.

So wie „Cloud" ein Modewort für die Verarbeitung und Speicherung von Daten über das Internet ist und das Schlagwort „IoT" Geräte wie „intelligente" Lampen bezeichnet, die mit dem Internet verbunden sind, ist „künstliche Intelligenz" ein Sammelbegriff für leistungsstarke Algorithmen und Modelle in Verbindung mit Big Data. Im Fall von ChatGPT-3 handelt es sich um 570 Gigabyte an Input, während DALL-E 2 mit mehr als 650 Millionen Bildern auskommt. Der Begriff „Intelligenz" ist tatsächlich nicht angebracht, und Forscher, die neuronale Netze so trainieren, dass sie nach Verbindungen suchen, die nicht wirklich existieren, schaffen sozusagen „künstliche Dummheit" [7].

Die dunkle Seite der KI

Ich gebe zu, dass generative KI inspiriert und neue Erkenntnisse liefert, aber ich bin kein großer Fan davon, wenn man bedenkt, dass die generierten Texte und Bilder im Grunde genommen Plagiate sind, weil sie auf der Arbeit von Millionen von Menschen beruhen. Dazu kommt, dass jede KI, die auf der Grundlage historischer Daten trainiert wurde, die unweigerlich menschliche Vorurteile enthalten, diese übernimmt - manchmal in verstärkter Form. Ich glaube nicht, dass ChatGPT diese Kolumne in der Zukunft schreiben wird, aber für den Online-Verkauf von Fotos und Kunstwerken ist es schon jetzt ziemlich schwierig. Zusammen mit den sich ständig verbessernden Kameras in Smartphones ist dies ein neuer Rückschlag für Amateurfotografen und Künstler, die sich etwas dazuverdienen wollen [8].

Es gibt auch viele Möglichkeiten des Missbrauchs, die von sprachlich einwandfreien Phishing-E-Mails bis hin zu Hassreden und tiefgreifenden Falschmeldungen reichen. Dank ChatGPT ist KI jetzt in aller Munde, aber es gibt sie schon viel länger und sie kann auch für andere Zwecke eingesetzt - oder missbraucht - werden, zum Beispiel von Regierungsbehörden in China mit ihrem Sozialkreditsystem oder in den USA, wo Gesichtserkennungstechnologie zur Identifizierung von Demonstranten eingesetzt wird. In Flandern will man KI auf Luftaufnahmen anwenden, um festzustellen, ob jemand in seinem Garten den einen oder anderen Baum zu viel gefällt hat. Ich bin in der Vergangenheit mehrfach Opfer von Verbrechen geworden, aber man hat sich nie die Mühe gemacht, die Täter zu finden. Dagegen sind alle Mittel recht, wenn es darum geht, kleinere Vergehen unbescholtener Bürger aufzuklären. Trauriger Höhepunkt ist der Fall eines Mannes aus Heusden-Zolder, der auf dem Heimweg von einem Restaurant mit seiner Familie kurz anhielt, um eine volle Windel seiner kranken Tochter in einen öffentlichen Mülleimer zu werfen. Zu seiner Überraschung erhielt er einige Zeit später einen zwanzigseitigen (!) Polizeibericht mit Fotos wegen illegaler Müllentsorgung per Post [9]. Offenbar bringt das Einsperren echter Verbrecher nicht genug Geld in die Kasse.

Es gibt genügend Anlass, um zu einem modernen Ludditen [10] zu werden.

SG – 230196-02

WEBLINKS

[1] ChatGPT: https://chat.openai.com
[2] DALL·E 2: https://labs.openai.com
[3] StabilityAI's Stable Diffisuion: https://huggingface.co/spaces/stabilityai/stable-diffusion
[4] Loab, eine KI-generierte Entität: https://loab.ai
[5] YouTube: Die verstörende Kunst der KI (gruselig!): https://youtu.be/i9InAbpM7mU
[6] Merlina: Erinnerungen an die erfolgreiche BRT-Jugendserie aus den 1980er Jahren (niederländisch): https://merlina.info/evarist/uitvindingen
[7] EOS Wetenschap: Kunstmatige Domheid (Künstliche Dummheit, niederländisch): https://eoswetenschap.eu/technologie/kunstmatige-domheid
[8] YouTube: AI Art Apocalypse: https://youtu.be/HDbu2rvhCk4
[9] Nieuwsblad: Heusdens gezin riskeert GAS-boete voor weggooien van pamper (niederländisch): https://nieuwsblad.be/cnt/dmf20220901_96655325
[10] Luddismus: https://de.wikipedia.org/wiki/Luddismus

Sensor 1x1: Temperatur-Sensor DS18B20

Anschluss am 1-Draht-Bus

Von Mathias Claußen (Deutschland)

Der Sensor DS18B20 von Maxim Integrated liegt vielen Einsteiger-Kits für den Einstieg in die Programmierung bei. Für den grundlegenden Umgang mit diesem Sensor ist etwas Wissen um den 1-Wire-Bus und den Anschluss des Sensors hilfreich. Ein kleiner Ausflug in die Temperaturerfassung mit dem DS18B20!

Wer Temperaturen mit einem Mikrocontroller erfassen möchte, hat die Wahl aus unterschiedlichen Sensoren und Bussystemen. Der DS18B20 von Maxim Integrated ist ein Sensor, der Temperaturen zwischen -55 °C und 125 °C (-67 °F bis +275 °F) mit einer Genauigkeit von ±0,5 °C erfassen kann. Er eignet sich dadurch nicht nur als Sensor für Umgebungstemperaturen, auch die Überwachung von Kühltruhen oder -räumen ist mit diesem Sensor keine Hürde. Daher soll der DS18B20 hier vorgestellt werden; Beispiele mit Quelltext und Schaltplan erleichtern den Einsatz in eigenen Projekten.

DS18B20

Der Sensor verwendet das durch Dallas Semiconductor 1989 patentierte 1-Wire-Bussystem, an das sich mehrere Sensoren anschließen lassen. Der DS18B20 wird mit 3,0 V bis 5,5 V versorgt, sodass er direkt an die I/O-Controllerpins von Arduino Uno bis Raspberry Pi Pico angeschlossen werden kann. 1-Wire erlaubt es Sensoren, auch einen parasitären Versorgungsmodus zu nutzen, bei dem Energie für den Betrieb des Sensors aus der Datenleitung bezogen wird. Der 1-Wire-Bus benötigt nur einen Pin an einem Mikrocontroller (MCU), um eine größere Menge an Sensoren anzubinden und wird deshalb vor allem bei MCUs mit wenigen Pins gerne eingesetzt. Neben dem DS18B20 gibt es noch weitere 1-Wire-Sensoren, die hier aber nicht weiter betrachtet werden sollen.

Der Sensor ist in unterschiedlichen Formen erhältlich, wie in **Bild 1** und **Bild 2** zu sehen. Durch den 1-Wire-Bus lässt sich nun

Bild 1. DS18B20 in TO-92 Form.

Bild 2. DS18B20 mit wasserdichtem Schutzrohr und Anschlussleitung.

Bild 3. Pinout des DS18B20.

sehr einfach eine größere Anzahl an Sensoren anschließen. Das 1-Wire-Protokoll selbst sieht kein Limit der Sensoren auf einem 1-Wire-Bus vor, jedoch geben sich Limits durch die elektrischen Eigenschaften des Busses.

1-Wire-Bus

Beim 1-Wire-Bus werden drei Leitungen zu den Sensoren benötigt: Masse (GND), Daten (DQ) und Versorgungsspannung (VCC). Der 1-Wire-Bus ist bidirektional aufgebaut und arbeitet mit einem Controller/Target-Konzept (Master und Slave). Controller und Target tauschen Daten über die eine Datenleitung aus. Das Pinout für den DS18B20 ist in **Bild 3** zu sehen.

Die Datenpins von Controller und Target sind hierbei als Open-Drain ausgelegt, was bedeutet, dass sie die Datenleitung nur aktiv nach Masse ziehen können. Das verhindert, dass Busteilnehmer einen Kurzschluss produzieren, falls einer einen High-Pegel und ein anderer zugleich einen Low-Pegel treiben würde. Da kein Teilnehmer aktiv den Pegel auf dem 1-Wire-Bus nach VCC ziehen kann, ist ein Pull-Up-Widerstand nötig. **Bild 4** zeigt den Anschluss eines 1-Wire-Sensors an einem Arduino Uno.

Neben der elektrischen Anbindung ist für den Betrieb von 1-Wire-Bauteilen aber auch noch ein Protokoll nötig. Mikrocontroller brauchen hierfür keine spezielle Hardware, das Protokoll wurde so gestaltet, dass ein UART ausreicht, der in den meisten Mikrocontrollern zu finden ist. Eine passende Beschaltung ist in **Bild 5** zu sehen.

Für die Ansteuerung per UART stellt Analog Devices einen passenden Artikel [1] bereit. Der UART ermöglicht es hier, die CPU zu entlasten, da Teile der Kommunikation und des Timings durch den UART übernommen werden. Jedoch ist dieses bei heutigen MCUs nicht zwingend nötig., denn selbst ein kleiner ATtiny hat ausreichend Ressourcen, um per 1-Wire-Bus angebundene Sensoren mit einer reinen Softwarelösung anzusprechen; ein I/O-Pin ist ausreichend. Die meisten Bibliotheken, die einen DS18B20 ansteuern können, sind für den Betrieb mit einem einzelnen I/O-Pin ausgelegt, die Verwendung eines UARTs ist eher eine Seltenheit.

Sensoridentifikation

Wenn alle Sensoren an einem Bus angeschlossen sind, müssen diese individuell ansprechbar sein, damit ihre Daten gelesen werden können. Dazu besitzt jeder Teilnehmer auf dem 1-Wire-Bus einen einmaligen, 64 Bit breiten Identifikationscode (UUID). Dieser setzt

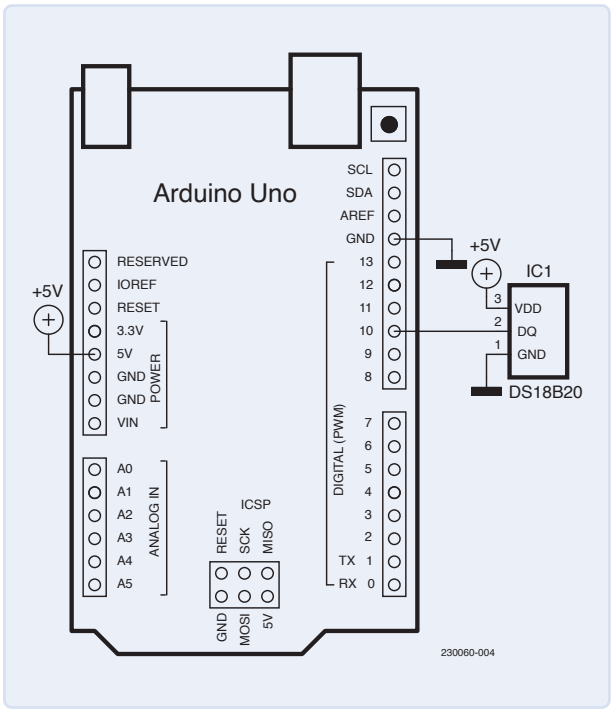


Bild 4. Schaltplan des Arduino Uno mit DS18B20.

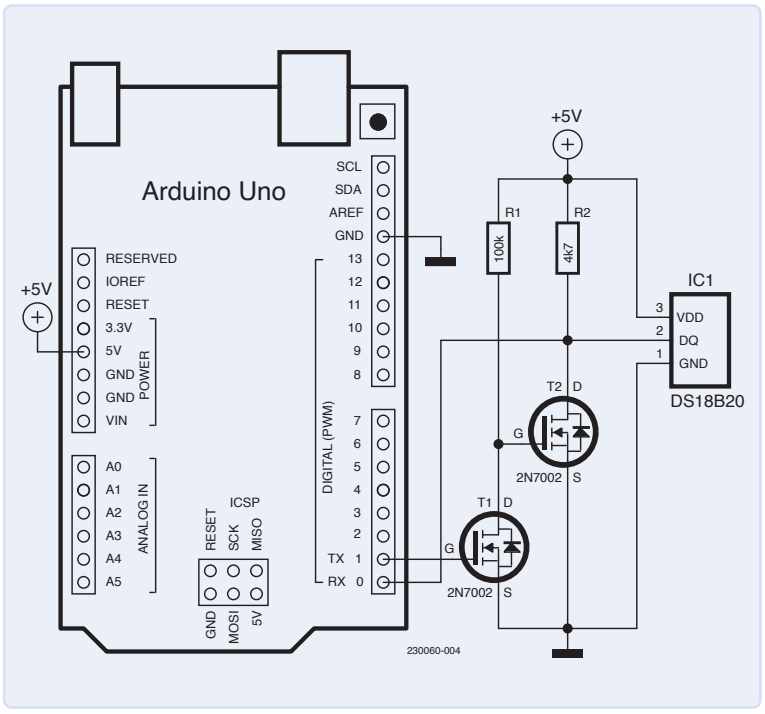


Bild 5. Schaltplan des DS18B20 am 1-Wire-Bus per UART.

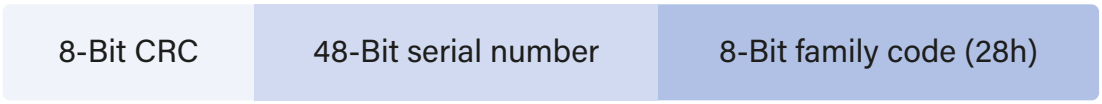


Bild 6. Aufbau der DS18B20-UUID.

sich aus einem 8-Bit-Code für die Familie, einer 48-Bit-Seriennummer und einer 8-Bit-Prüfsumme (CRC) zusammen (**Bild 6**). Der Code selber ist jedoch nicht auf dem Sensor aufgedruckt und muss daher durch eine Suchfunktion auf dem 1-Wire-Bus ermittelt werden. Der nötige Algorithmus für die Suche findet sich auf der Seite von Analog Devices [2].

Beispielcode für Arduino

Ausgehend vom Schaltplan aus Bild 4 ist der Aufbau sehr einfach gehalten. Der Sensor wird wie der Arduino Uno auch mit 5 V betrieben. Der Pin am Arduino Uno benötigt noch einen 4,7-kΩ-Pull-up-Widerstand. Die nötige Bibliothek für den 1-Wire-Bus ist in diesem Fall die *OneWire*-Bibliothek von Paul Stoffregen [3]. In dem Beispielcode der Bibliothek ist auch das Auslesen von DS18B20-Sensoren enthalten (**Listing 1**), was nun etwas näher betrachtet werden soll.

Um die Bibliothek zu nutzen, wird sie am Anfang des Sketches mit `#include <OneWire.h>` eingebunden. Danach kann mit `OneWire ds(10);` der 1-Wire-Bus an Pin 10 in Betrieb genommen werden. Mit `ds.search(addr)` wird nach dem nächsten Sensor auf dem Bus gesucht. Wurden alle Sensoren gefunden, wird die Suche beendet. Wenn ein Sensor gefunden wurde, so wird die ermittelte Sensoridentifikation in `addr` vermerkt.

DS18B20 - The Clone Wars

Der DS18B20 ist einer der Sensoren, den viele schon aus dem einen oder anderen Sensorkit oder Mikrocontroller-Starter-Kit kennen. Auch wenn man sich auf dem einschlägigen Marktplätzen umschaut, so gibt es einen DS18B20 schon um die 30 Cent, während Distributoren wie Mouser oder Farnell momentan für einen DS18B20 deutlich über 4 Euro aufrufen müssen.

Durch seine Popularität wurde der DS18B20 nachgebaut oder geklont. Während die Chips sich auf den ersten Blick wie das Original verhalten, weichen diese von der Charakteristik des Originals bei genauerer Betrachtung teilweise deutlich ab.

Auf der GitHub-Seite von Chris Petrich [4] gibt es neben Arduino-Sketchen zum Testen des eigenen Sensors auch weitere Informationen zu den einzelnen Klonen und deren Abweichungen vom Original.

Ist der Sensor so identifiziert, kann mit ihm interagiert werden. In dem ersten Byte, zu finden in `addr[0]`, ist der Code der Familie enthalten. Damit kann erkannt werden, ob ein DS18B20-, DS18S20- oder DS1822-Temperatursensor gefunden wurde, oder ob es sich um einen ganz anderen Typ von 1-Wire-Bus-Teilnehmer handelt. In Zeile 69 ist folgende Sequenz zu sehen:

```
ds.reset();
ds.select(addr);
ds.write(0x44, 1);
```

Hier wird einmal ein Reset aller Busteilnehmer mit `ds.reset();` durchgeführt. Anschließend wird mit `ds.select(addr);` der eben gefundene Sensor angesprochen. Um jetzt eine Temperatur vom Sensor zu erhalten, muss zuerst eine Messung gestartet werden. Mit dem entsprechenden Befehl `ds.write(0x44, 1);` wird 0x44 als Kommando gesendet. Mit dem zweiten Parameter (hier `1`) wird der I/O-Pin aktiv auf High getrieben, um Sensoren wie dnm DS18S20 mit parasitärer Stromversorgung zu betreiben. Das Ergebnis der Temperaturmessung steht nach 750 ms bis 1000 ms bereit. In diesem Beispiel wird einfach Rechenzeit vernichtet, um diese Zeitspanne zu überbrücken.

Ist die Temperaturmessung abgeschlossen und das Ergebnis steht bereit, kann dieses vom Sensor abgeholt werden. Dazu wird der Sensor wieder mit `ds.select(addr);` adressiert und ihm mitgeteilt, dass jetzt das RAM des Sensors, Scratchpad genannt, gelesen werden soll (`ds.write(0xBE);`). Jetzt können Sie neun Byte aus dem Sensor lesen, in dem sich die Temperaturdaten befinden.

Beim DS18B20 steht die Temperatur nach dem Auslesen als 16-Bit-Wert bereit, die aus zwei Registerbytes stammen. Sollte ein DS18S20 anstelle eines DS18B20 verwendet worden sein, muss der Wert noch umgerechnet werden. Das Codebeispiel berücksichtigt dieses und ergänzt zu einem 9-Bit-Wert noch weitere 3 Bit aus einer zusätzlichen Registerposition. Am Ende wird das Ergebnis passend geshiftet und bereitgestellt.

Der DS18B20 gibt die Temperatur mit 12 Bit aus und benötigt 750 ms für die Messung. Er kann optional auch 11-, 10- und 9-Bit-Werte ausgeben, bei einer sich verkürzenden Wandlungszeit. Damit ein stimmiges Ergebnis entsteht, müssen daher gegebenenfalls einzelne Bits auf 0 gesetzt werden.

Soll eine Temperatur in Fahrenheit ausgegeben werden, so ist eine Umrechnung durch den Mikrocontroller nötig, der Sensor selbst kann nur Daten in Celsiusgraden ausgeben.

Zusammenfassung

Mit dem DS18B20 und 1-Wire lassen sich einfach Temperatursensoren an Mikrocontroller anschließen. Für die Arduino-IDE gibt es passende Bibliotheken und Beispielquelltexte. Da nur ein Pin

der MCU benötigt wird, um diesen und auch mehrere Sensoren anzuschließen, kann eine kleine MCU mit wenigen Pins mehrere Messstellen bedienen.

RG — 230060-02

Sie haben Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Kontaktieren Sie Elektor unter redaktion@elektor.de!

Passende Produkte

- **Elektor 37-in-1 Sensorkit**
  SKU 16843: www.elektor.de/16843
- **Cytron Maker Uno**
  SKU 18634: www.elektor.de/18634
- **MakePython ESP32 Development Kit**
  SKU 20137: www.elektor.de/20137
- **Miroslav Cina:** ***Das 1-Wire-Praxisbuch***
  Buch (kartoniert, SKU 19263): www.elektor.de/19263
  E-Buch (PDF, SKU 19264): www.elektor.de/19264

WEBLINKS

[1] Using a UART to implement a 1-wire-bus-master, Analog Devices: https://www.analog.com/en/technical-articles/using-a-uart-to-implement-a-1wire-bus-master.html
[2] 1-Wire-Suchalgorithmus, Analog Devices: https://www.analog.com/en/app-notes/1wire-search-algorithm.html
[3] OneWire-Bibliothek von Paul Stoffregen auf GitHub: https://github.com/PaulStoffregen/OneWire
[4] Chris Petrich, „Your DS18B20 temperature sensor is likely a fake, counterfeit, clone…“, GitHub: https://github.com/cpetrich/counterfeit_DS18B20

Listing 1. Arduino-Beispielprogramm für den DS18B20

```
#include <OneWire.h>

// OneWire DS18S20, DS18B20, DS1822 Temperature Example
//
// http://www.pjrc.com/teensy/td_libs_OneWire.html
//
// The DallasTemperature library can do all this work for you!
// https://github.com/milesburton/Arduino-Temperature-Control-Library

OneWire  ds(10);  // on pin 10 (a 4.7K resistor is necessary)

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

void loop(void) {

  byte i;
  byte present = 0;
  byte type_s;
  byte data[9];
  byte addr[8];
  float celsius, fahrenheit;

  if ( !ds.search(addr)) {
    Serial.println("No more addresses.");
    Serial.println();
```

continued overleaf

```
028           ds.reset_search();
029           delay(250);
030           return;
031         }
032
033
034         Serial.print("ROM =");
035         for( i = 0; i < 8; i++) {
036           Serial.write(' ');
037           Serial.print(addr[i], HEX);
038         }
039
040         if (OneWire::crc8(addr, 7) != addr[7]) {
041             Serial.println("CRC is not valid!");
042             return;
043         }
044
045         Serial.println();
046         // the first ROM byte indicates which chip
047         switch (addr[0]) {
048
049           case 0x10:
050             Serial.println("  Chip = DS18S20");  // or old DS1820
051             type_s = 1;
052             break;
053
054           case 0x28:
055             Serial.println("  Chip = DS18B20");
056             type_s = 0;
057             break;
058
059           case 0x22:
060             Serial.println("  Chip = DS1822");
061             type_s = 0;
062             break;
063
064           default:
065             Serial.println("Device is not a DS18x20 family device.");
066             return;
067         }
068
069         ds.reset();
070         ds.select(addr);
071         ds.write(0x44, 1);        // start conversion, with parasite power on at the end
072         delay(1000);     // maybe 750ms is enough, maybe not
073         // we might do a ds.depower() here, but the reset will take care of it.
074
075         present = ds.reset();
076         ds.select(addr);
077         ds.write(0xBE);         // Read Scratchpad
078
079         Serial.print("  Data = ");
080         Serial.print(present, HEX);
081         Serial.print(" ");
082         for ( i = 0; i < 9; i++) {           // we need 9 bytes
083           data[i] = ds.read();
084           Serial.print(data[i], HEX);
085           Serial.print(" ");
086         }
087         Serial.print(" CRC=");
088         Serial.print(OneWire::crc8(data, 8), HEX);
```

```
089         Serial.println();
090         // Convert the data to actual temperature
091         // because the result is a 16 bit signed integer, it should
092         // be stored to an "int16_t" type, which is always 16 bits
093         // even when compiled on a 32 bit processor.
094
095         int16_t raw = (data[1] << 8) | data[0];
096         if (type_s) {
097           raw = raw << 3; // 9 bit resolution default
098           if (data[7] == 0x10) {
099             // "count remain" gives full 12 bit resolution
100             raw = (raw & 0xFFF0) + 12 - data[6];
101           }
102         } else {
103           byte cfg = (data[4] & 0x60);
104           // at lower res, the low bits are undefined, so let's zero them
105           if (cfg == 0x00) raw = raw & ~7;  // 9 bit resolution, 93.75 ms
106           else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms
107           else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms
108           //// default is 12 bit resolution, 750 ms conversion time
109         }
110
111         celsius = (float)raw / 16.0;
112         fahrenheit = celsius * 1.8 + 32.0;
113         Serial.print("  Temperature = ");
114         Serial.print(celsius);
115         Serial.print(" Celsius, ");
116         Serial.print(fahrenheit);
117         Serial.println(" Fahrenheit");
118       }
```

Rettet Matter das Smart Home?

Neue Standards zur Vereinfachung der Hausautomatisierung

Von Stuart Cording (Elektor)

Wischen, drücken, auswählen, zurück, auswählen, wischen - ah, jetzt ist das Licht an. Naja, meistens jedenfalls. Das Smart Home ist nicht immer so intelligent wie versprochen, dank zu vieler Apps, Kompatibilitätsproblemen und unterschiedlicher Standards. Jetzt gibt es eine Lösung - ein paar Standards mehr! Aber können und werden Thread und Matter all diese Probleme lösen?

Was gibt es Schöneres, als nach einem harten Arbeitstag nach Hause zu kommen und ein Heim vorzufinden, das für Sie bereit ist? Die Musikanlage spielt Ihre Lieblingsmusik, die Räume sind warm und gemütlich, und Ihr Essen ist fertig und kann aus dem Ofen genommen werden. Wenn Sie das Wohnzimmer betreten, schließen sich die Jalousien, um das direkte Sonnenlicht auszusperren, und die Stehlampe neben Ihrem bequemen Sessel leuchtet auf, damit Sie Ihren Roman auf Ihrem E-Reader weiterlesen können. Natürlich ist auch dieser zum Leben erwacht und zeigt die Seite, auf der Sie gestern Abend aufgehört haben.

Das ist der Traum vom Smart Home. Und für viele bleibt es ein Traum. Es gibt zwar jede Menge drahtloser und kabelgebundener Protokolle, aber jedes hat seinen eigenen Bereich, seine eigenen Anwendungen und Funktionen. Die Vision eines Hauses, das auf unsere Bedürfnisse reagiert, hat eine lange Geschichte. In den späten 1990er Jahren zeigte Microsoft ein Video, das ziemlich genau einen Großteil der heutigen Technologie vorhersagte. In dem Werbevideo aus dieser Zeit werden elektronische Türschlösser, Internet-Shopping, Sprachassistenten und vieles mehr gezeigt, alles auf der Basis von Windows XP und CE [1]. Und diese Ideen waren auch damals nicht neu. Ray Bradbury, der berühmte Science-Fiction-Autor des 20. Jahrhunderts, beschwor in seiner Kurzgeschichte aus den Mars-Chroniken „There Will Come Soft Rains“ [2] schon in den 1950er Jahren ein fast identisches Haus. Bevor wir jedoch erfahren, ob und wie Thread und Matter das heutige Wirrwarr lösen können, müssen wir herausfinden, wie es überhaupt dazu gekommen ist.

Die Geburt des drahtlosen Kurzstreckenfunks

Als das Internet der Dinge zu einem Ding wurde, war es klar, dass solche Dinge drahtlos miteinander verbunden werden mussten. In den frühen 2000er Jahren gab es dazu nicht viele Möglichkeiten. Man hatte Wi-Fi, ein GSM-Mobilfunkmodem und Bluetooth. Die ersten beiden Standards waren nicht gerade für einen stromsparenden Betrieb berühmt, so dass sie für batteriebetriebene Geräte nicht in Frage kamen. Damit blieb nur noch Bluetooth übrig, eine Technologie, die den Anforderungen des aufstrebenden Mobiltelefonmarktes entsprach und eine vielversprechende Alternative zu IrDA darstellte, der Infrarotverbindung, die bei PDAs (Personal Digital Assistants) für die Datensynchronisierung verwendet wurde.

Bluetooth nutzte den 100-MHz-Frequenzbereich um 2,4 GHz, der weltweit als Teil des ISM-Bandes (Industrial, Scientific and Medical) verfügbar war, und schien sich damit IoT zu eigen zu machen. Zu dieser Zeit gab es Internet-Gateways, drahtlose serielle Verbindungen und sogar Bluetooth-Zusatzgeräte für Drucker. Es wurde nicht einmal nativ von Windows unterstützt und erforderte eine Software, die auf einem Dongle geliefert wurde, um die Kommunikation zwischen einem PC und einem Telefon zu ermöglichen.

Bluetooth hat sich im Audiobereich etabliert und bietet eine robuste Schnittstelle für die Freisprecheinrichtung mit Ein-Ohr-Headsets und Fahrzeugzubehör. Autotelefone und sperrige Plastikadapter gehörten der Vergangenheit an, da die Fahrer einfach das Telefon, das sie besaßen, mit der Freisprecheinrichtung des Fahrzeugs koppeln konnten.

Das Protokoll war von Anfang an als Personal Area Network (PAN) konzipiert, mit der Verbindung von bis zu sieben Slave-Geräten an einem Master, die so ein Piconet bildeten (**Bild 1**). Es war sogar ein Scatternet vorgesehen, bei dem ein Master eines Piconets ein Slave eines anderen Piconets sein konnte oder ein Slave-Gerät Teil von zwei Piconets (**Bild 2**). Da damals jedoch nur ein Durchsatz von 720 kbit/s zur Verfügung stand, fiel die Datenrate erheblich ab, wenn das Netz wuchs und Master und Slave umgeschaltet werden sollten. Außerdem war nicht definiert, wie die Daten zwischen den Piconets weitergeleitet werden sollten - dies wurde dem Entwickler überlassen, der Bluetooth zu diesem Zweck implementierte. Der Autor war in dieser frühen Zeit an mehreren diesbezüglichen Projekten beteiligt, aber Scatternets schienen außerhalb eines praktischen Einsatzes zu sein.

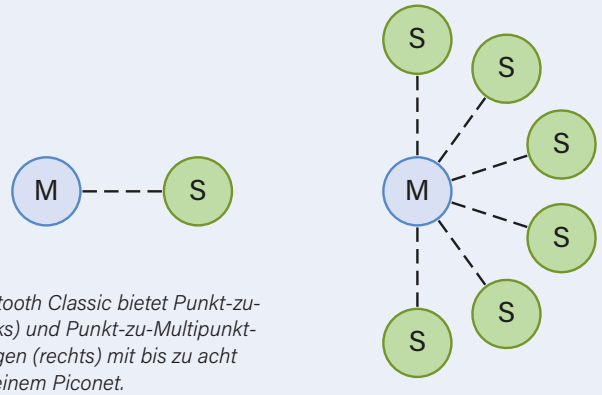


Bild 1. Bluetooth Classic bietet Punkt-zu-Punkt- (links) und Punkt-zu-Multipunkt-Verbindungen (rechts) mit bis zu acht Knoten in einem Piconet.

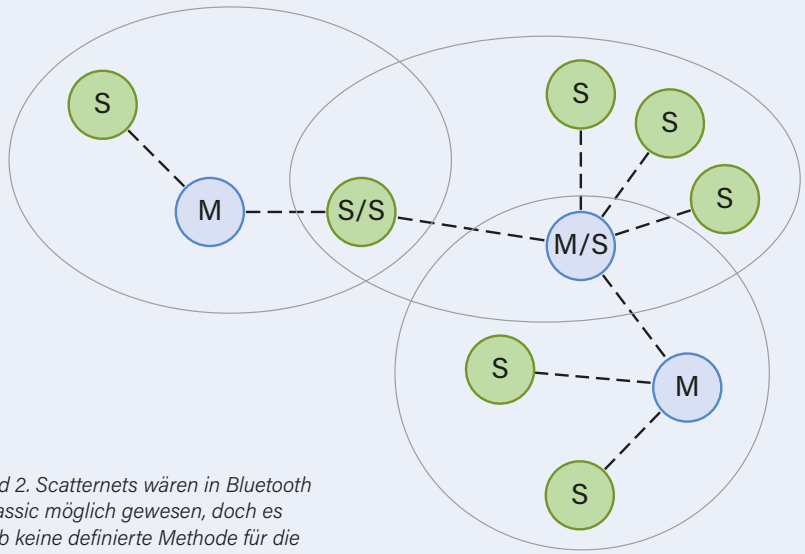


Bild 2. Scatternets wären in Bluetooth Classic möglich gewesen, doch es gab keine definierte Methode für die Datenübermittlung zwischen Piconets.

Bild 3. Die TRÅDFRI-Produktreihe von Ikea mit intelligenten Haushaltsgeräten wie Schaltern und Glühbirnen verwendet Zigbee.

Stromsparen

Die Notwendigkeit, den Transceiver mit Strom zu versorgen, machte Bluetooth ungeeignet für batteriebetriebene Sensoren. Nokia hatte eine 2,4-GHz-Technologie namens Wibree [3] entwickelt, um dieses Problem zu lösen. Mitte der 2000er Jahre boten Dual-Mode-Chipsätze Bluetooth und Wibree nebeneinander an und zielten damit auf intelligente Uhren, Sportsensoren und sogar drahtlose Tastaturen ab. Im Jahr 2007 wurde Wibree von der Bluetooth SIG, die die Bluetooth-Spezifikation verwaltet, übernommen und schließlich zu Bluetooth Low Energy (BLE). Heute sind Bluetooth Classic und BLE im selben Chipsatz untergebracht, teilen sich eine Antenne und einen Funkempfänger und haben Gemeinsamkeiten in ihrem Software-Stack, sind aber nicht miteinander kompatibel [4].

Trotz der geringen Stromaufnahme, der sie für batteriebetriebene IoT-Sensoren geeignet macht, wurde die Mesh-Networking-Funktion von Bluetooth LE erst im Jahr 2017 verfügbar.

Drahtlose Low-Power-Personal-Area-Networking-Alternativen

Zu diesem Zeitpunkt waren aber bereits alternative IoT-Funkstandards auf dem Markt. Zigbee, eine auf IEEE 802.15.4 basierende Technologie, gab es bereits seit 2003, und 2006 wurde eine überarbeitete Spezifikation vorgelegt (**Bild 3**). Auch sie arbeitete im 2,4-GHz-Band, doch konnten auch andere Frequenzen wie 784 MHz (China), 868 MHz (Europa) und 915 MHz (Australien und USA) genutzt werden. Und es gab Z-Wave, eine von Zensys entwickelte Sub-Gigahertz-Technologie, die schließlich von Silicon Labs aufgekauft wurde und ebenfalls um 2003 auf den Markt kam. Diese beiden Technologien bieten selbstheilende Mesh-Netzwerke, was heißt, dass selbst dann, wenn ein Knoten ausfällt, der Netzwerkverkehr einen alternativen Weg findet zwischen dem Gerät, das die Nachricht sendet, und dem Gerät, das sie empfangen soll. Und dies geschieht im Hintergrund, ohne dass der Systembenutzer davon etwas bemerkt. Diese beiden drahtlosen Systeme werden heute von tausenden Produkten genutzt.

Probleme mit stromsparenden Mesh-Netzwerken

Bei all diesen Technologien gibt es jedoch einige Probleme. Erstens stellt keine von ihnen von Haus aus eine Verbindung zum Internet her, was nützlich wäre, wenn man ein intelligentes Heimsystem einrichten möchte. Ein Bluetooth-Gerät könnte über ein Smartphone mit Cloud-Diensten interagieren, wenn es sich in der Nähe befindet, aber ansonsten benötigen sie alle ein Hardware-Gateway, das eine Verbindung zu einem Router mit Internetanschluss herstellt. Zweitens sind die konkurrierenden Technologien nicht so ohne weiteres interoperabel. Wenn Sie sich beispielsweise für Zigbee als Smart-Home-Technologie entscheiden und feststellen, dass ein Produkt, das Sie gerne verwenden möchten, nur mit Z-Wave verfügbar ist, dann ist das sehr ärgerlich. Es ist zwar möglich, mit Node-RED [5] Wege zur Überbrückung dieser Lücke zu entwickeln, aber dies ist für die meisten Nutzer keine praktikable Lösung.

Drittens kann sich die Steuerung der Geräte im Netzwerk als unhandlich erweisen. Möglicherweise sind mehrere Apps erforderlich, die alle unterschiedliche Steue-

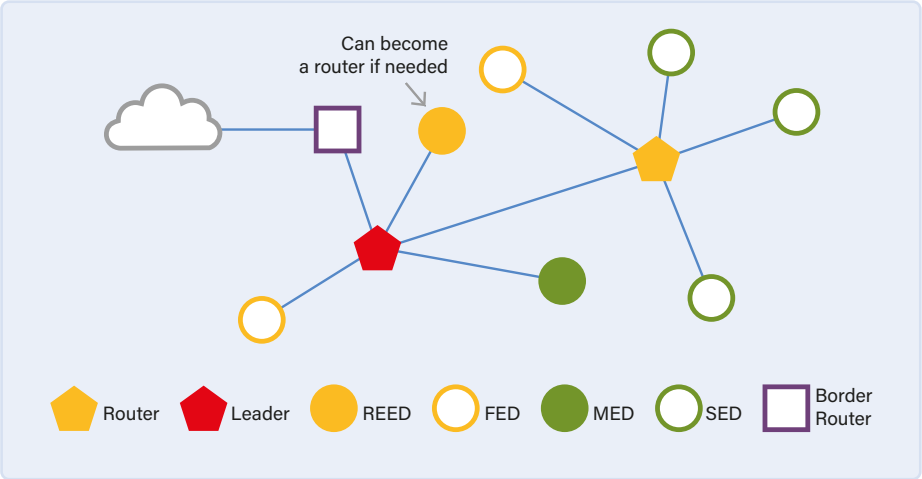


Bild 4. Thread bietet ein selbstheilendes Mesh, das im Gegensatz zu den Alternativen den IPv6-Standard 6LoWPAN verwendet.

Bild 5. Matter-Lösungen sind von Espressif erhältlich, wie mit diesem Demonstrationskoffer auf der embedded world 2023 gezeigt wurde.

Bild 6. Silicon Labs zeigte ebenfalls, dass Matter sowohl mit Zigbee- als auch mit Z-Wave-Netzwerken funktioniert.

rungsmethoden haben. Und auch die Verknüpfung von Sprachassistenten mit diesen Systemen kann eine Herausforderung darstellen. Amazon Alexa unterstützt weit mehr Geräte als Google Home, das wiederum mit mehr Geräten kompatibel ist als Apple HomeKit [6].

Um einige dieser Probleme zu lösen, hat die Technikbranche das getan, was sie am besten kann, nämlich - eine weitere Technologie einzuführen: Thread.

Alles miteinander verknüpfen

Die Thread Group Alliance wurde 2014 gegründet und hat sich auf die Entwicklung einer drahtlosen Netzwerktechnologie mit geringem Strombedarf und niedriger Latenz konzentriert, die mit den bereits erwähnten Lösungen konkurriert. Sie arbeitet ebenfalls mit 2,4 GHz, unterscheidet sich aber in mehrfacher Hinsicht.

Erstens nutzt sie die Kommunikation über das Internetprotokoll (IP), genauer gesagt eine Version von IPv6 namens 6LoWPAN. Damit wird IP an die Bedürfnisse von Geräten mit geringer Stromaufnahme und die verwendeten IEEE-802.15.4-Funknetze angepasst. Zu den verwendeten Mechanismen gehören Fragmentierung, Reassambling und Komprimierung, die bei IPv6 nicht notwendig wären.

Zweitens sind die Sicherheitsmaßnahmen robuster und obligatorisch. Die Daten werden wie in jedem anderen TCP/IP-Netz End-to-End verschlüsselt, das heißt, dass Geräte, die im Netz als Router fungieren, den Inhalt der sie durchlaufenden Daten nicht überprüfen können. Und schließlich ist die Mesh-Latenzzeit nur etwa halb so hoch wie bei den Alternativen wie Zigbee und etwa siebenmal niedriger als bei Bluetooth [7].

Thread-Knoten

Die Knoten eines Thread-Mesh-Netzwerks bestehen aus einem von zwei Gerätetypen [8]: einem Full Thread Device (FTD) oder einem Minimal Thread Device (MTD). MTDs können nur Endgeräte (ED) sein und benötigen einen Router-Knoten, um dem Thread-Netzwerk beizutreten. Ein einzelner Router unterstützt bis zu 511 EDs. Es gibt zwei Arten von MTDs: Das Minimal End Device (MED) verfügt über einen ständig eingeschalteten Transceiver, während das der Transceiver eines Sleepy End Device (SED) in regelmäßigen Abständen den Eingang von Nachrichten überprüft. SED-Knoten wie Sensoren werden in der Regel mit einer Batterie betrieben.

Transceiver von FTDs, die in zwei Varianten angeboten werden, sind immer eingeschaltet. Bei der ersten Variante handelt es sich um Router, die ihre Transceiver eingeschaltet lassen, um neue Knoten, die dem Netzwerk beitreten, zu unterstützen und Pakete weiterzuleiten. Der erste Knoten, der die Rolle des Routers übernimmt, wird als Leader bezeichnet und verwaltet zukünftige Knoten, die dem Netzwerk hinzugefügt werden und die Rolle des Routers übernehmen. Es kann nur einen Leader und bis zu 32 Router geben. Sollte der Leader ausfallen oder entfernt werden, übernimmt einer der anderen Router diese Rolle. Sollte die Router-Rolle nicht benötigt werden (da genügend Router im Netz vorhanden sind), kann das FTD zu einem Router-berechtigten Endgerät (REED) werden. Es fungiert so lange als ED, bis seine Router-Funktion benötigt wird, woraufhin es „befördert“ wird.

Das zweite ist das Full End Device (FED), das nur als ED fungieren kann, aber mehr Daten über das Netz im Auge behält als MEDs (zum Beispiel Multicasting und IPv6-Adresszuordnung). Dieses Profil ist für netzgespeiste Geräte geeignet.

Mit diesen Bestandteilen ist das Thread-Mesh voll funktionsfähig. Für die Verknüpfung mit Cloud-Diensten ist jedoch ein weiteres Element erforderlich - ein oder mehrere Border-Router. Solche Geräte verbinden das Thread-Netz entweder mit Ethernet oder WLAN (**Bild 4**). Sie könnten auch in Haushaltsgeräten wie Smart-TVs, WLAN-Router und Sprachassistenten integriert sein, die bereits mit dem Internet verbunden sind.

Der größte Vorteil besteht darin, dass Thread-Geräte genauso wie alle anderen

Ethernet-, WLAN- und Mobilfunkgeräte über sicheres IP kommunizieren. Im Vergleich dazu müssen Zigbee, Z-Wave und Bluetooth jede Internet/Cloud-Kommunikation in proprietäre drahtlose Netzwerkprotokolle übersetzen. Ein weiteres wichtiges Verkaufsargument ist jedoch, dass der Anwendungslayer, an den die Smartphone-Apps und Webschnittstellen in der Regel angeschlossen werden, unabhängig ist. Das bedeutet, dass sich theoretisch die Interoperabilität zwischen Thread-fähigen Geräten und Heimassistenten verschiedener Hersteller verbessern sollte. Außerdem unterstützt es eine weitere neue Technologie: Matter.

Interoperabilität von Geräten

Matter ist seit einigen Jahren in der Entwicklung und war ursprünglich als das Project *Connected Home over IP*, kurz CHIP, bekannt. Einer der Mitwirkenden ist die Connectivity Standards Alliance [9] (CSA), die ehemalige Zigbee-Alliance, die zusätzlich zu ihrem bestehenden Aufgabenbereich für Zigbee-basierte Geräte auch die Zertifizierung von Matter-Produkten übernimmt. Da Matter auf dem TCP/IP-Transportlayer aufsetzt, ist es auch bereit, mit Thread und jedem Ethernet- oder WLAN-Gerät im Haus zusammenzuarbeiten.

Matter ist eine Softwareschicht, die definiert, wie Geräte miteinander kommunizieren, so dass jedes Smart-Home-System oder jeder Sprachassistent verwendet werden kann. Der vielleicht größte Vorteil ist jedoch Multi-Admin, das die gleichzeitige Nutzung verschiedener Apps mit Geräten ermöglicht. So kann beispielsweise eine Glühbirne sowohl über einen Wandschalter als auch über die vom Hersteller bereitgestellte App auf dem Smartphone gesteuert werden, während sie gleichzeitig mit dem Sprachassistenten Ihrer Wahl verbunden ist und von diesem gesteuert wird. Das bedeutet auch, dass ein Google-Sprachassistent in der Lage sein sollte, Amazon- oder Apple-Geräte zu steuern. Es gibt jedoch keine Garantie dafür, wie weit die Kompatibilität in der Realität reichen wird. Wie Simon Hill von Wired in seinem Artikel über Matter anmerkt, besteht immer noch die Möglichkeit oder Gefahr, dass die Steuerung von erweiterten Funktionen oder Einstellungen die Verwendung eines Geräts oder einer App aus demselben Ökosystem erfordert [10].

Mehrere Halbleiterhersteller wie Espressif [11] und Silicon Labs [12] haben demonstriert, wie Matter mit Thread- und WLAN-Geräten funktioniert (**Bild 5** und **Bild 6**). Derzeit unterstützt die Spezifikation die meisten grundlegenden Smart-Home-Geräte, von Glühbirnen und Steckdosen bis hin zu Türschlössern und Thermostaten, und auch die großen Sprachassistenten sind dabei. Weiße Ware, Staubsaugerroboter und Kameras sind einige der Punkte auf der Liste für die nächste Version der Spezifikation [13].

Hat Matter eine Zukunft?

Nach einem Jahrzehnt Smart-Home-IoT befinden wir uns also immer noch in einer weiteren VHS-gegen-Betamax-Schlacht [14], die nicht klarer, sondern eher komplexer wird. Der Schlüssel scheint Thread zu sein, das das seit langem bestehende Problem mit Zigbee, Z-Wave und Bluetooth Mesh löst, die ein Gateway benötigen, das von IP in das jeweilige drahtlose Protokoll übersetzt. Matter bietet den Schmierstoff für Thread und sorgt dafür, dass die User und nicht die Elektronikriesen entscheiden, welche Apps und Sprachassistenten sie zur Steuerung ihrer Geräte verwenden. Matter verhindert auch, dass bestehende Installationen über Nacht obsolet werden. Natürlich kann es nur Schnittstellen zu den Gateways von Zigbee und Co. bieten, nicht zu den Netzwerken selbst, aber das sollte genügen. Und da Matter WLAN- und Ethernet-Geräte unterstützt, könnte dies ausschlaggebend für den Erfolg sein. Hersteller von hochwertigen, langlebigen Konsumgütern wie Waschmaschinen und Kochgeräten wissen, dass fast alle ihre Kunden WLAN haben und müssen nicht auf eine der drei anderen Lösungen setzen. Wird Matter also in zehn Jahren noch eine Rolle spielen? Es scheint auf jeden Fall mehr auf die Bedürfnisse der Nutzer ausgerichtet zu sein, so dass die Chancen gut stehen. Aber wie immer wird es nur die Zeit zeigen.

RG — 230226-02

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Senden Sie eine E-Mail an den Autor unter stuart.cording@elektor.com oder kontaktieren Sie Elektor unter redaktion@elektor.de.

WEBLINKS

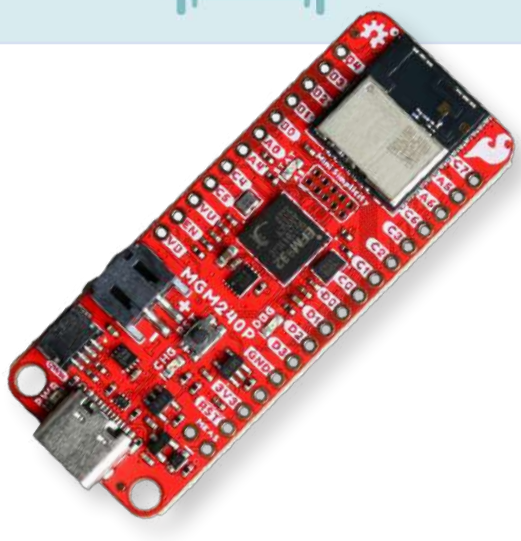
[1] „Microsoft Smart Home“, Microsoft 1999 (YouTube): https://bit.ly/3zoDVCv
[2] Ray Bradbury, „There Will Come Soft Rains (Kurzgeschichte)“, (Wikipedia): http://bit.ly/3ZR73x7
[3] E. Grabianowski, „Is Wibree going to rival Bluetooth?“, HowStuffWorks: http://bit.ly/3nHHTnb
[4] M. Afaneh, „Bluetooth vs. Bluetooth Low Energy: What's the Difference?“, April 2022: http://bit.ly/3KvmiaA
[5] R. Dullier, „Control a Z-Wave plug using a Zigbee button!“, März 2021: http://bit.ly/3nHRdHB
[6] M. Timothy, „Amazon Alexa vs. Google Home vs. Apple HomeKit: What's the Best Smart Home System?“, MakeUseOf, März 2023: http://bit.ly/40C3wDY
[7] „Benchmarking Bluetooth Mesh, Thread, and Zigbee Network Performance“, Silicon Labs: http://bit.ly/3Ge74UF
[8] „Node Roles and Types“, Google, Februar 2023: http://bit.ly/3m5ZmVN
[9] Website der Connectivity Standards Alliance (CSA): http://bit.ly/3nF6qcm
[10] S. Hill, „Here's What the 'Matter' Smart Home Standard Is All About“, Wired, Oktober 2022: http://bit.ly/3zrpWf7
[11] Espressifs Lösung für Matter: http://bit.ly/3zsb545
[12] Video: Matter over Wi-Fi and Thread Demo - Silicon Labs: http://bit.ly/3nFZ2gZ
[13] . P. Tuohy, „We're getting our first look at Matter devices today, and here's what's coming next“, The Verge, November 2022: http://bit.ly/410zh9D
[14] D. Owen, „The Betamax vs VHS Format War“, MediaCollege.com, Mai 2005: http://bit.ly/3U78JRD

Eine Frage der Zusammenarbeit

Entwickeln mit dem Board Thing Plus Matter und dem Simplicity Studio

Von Rob Reynolds (SparkFun)

Um in die Heimautomatisierung einzusteigen, musste man sich bisher für ein Ökosystem entscheiden. Nun, diese Zeiten sind vorbei, denn Matter will es jedem IoT-Gerät leicht machen, über dieses neue, quelloffene Protokoll zu kommunizieren. In diesem Artikel entwickeln wir eine kleine Matter-kompatible Demo-Anwendung mit dem neuen SparkFun-Entwicklungsboard *Thing Plus Matter* und der IDE *Simplicity Studio* von Silicon Labs.



Das Internet der Dinge oder IoT, wie es kurz und knapp genannt wird, ist uns allen mittlerweile gut bekannt. Aber selbst heutzutage gibt es wegen verschiedener Kommunikationsprotokolle reichlich Verwirrung. Dies zwingt sowohl Entwickler als auch Verbraucher zu der Entscheidung, wie ihre Geräte kommunizieren sollen, um sie an ein Netzwerk zu binden. Mit der Einführung des einheitlichen, quelloffenen Verbindungsstandards Matter, der es Entwicklern und Geräteherstellern ermöglichen soll, zuverlässige und sichere Ökosysteme aufzubauen und die Kompatibilität zwischen vernetzten Heimgeräten zu verbessern, werden diese Zeiten hoffentlich bald vorüber sein.

Eine kurze Geschichte von Matter

Matter begann im Jahr 2019 als das Project CHIP, was für *Connected Home over IP* steht. Große und zuvor konkurrierende Akteure wie Amazon, Apple und Google sowie die Zigbee-Allianz und eine Reihe anderer Unternehmen wie Nordic Semiconductor taten sich zusammen, um ein gemeinsames Kommunikationsprotokoll zu entwickeln, das zur Vereinheitlichung des gesamten Internets of Things verwendet werden sollte. Matter ist ein quelloffenes und lizenzfreies Protokoll, mit dem Geräte über WLAN, Ethernet, Bluetooth-Low-Energy- und Thread-Netzwerke kommunizieren können. Das bedeutet, dass Geräte, die Matter-zertifiziert sind, unabhängig von der verwendeten Funktechnologie nahtlos miteinander kommunizieren können.
Mit Matter müssen sich Entwickler, Hersteller und Verbraucher nicht mehr zwischen Apples HomeKit, Amazons Alexa oder den Weave-Komponenten von Google entscheiden. Für Hersteller von IoT-Geräten bedeutet dies eine vereinfachte Entwicklung, für die Verbraucher mehr Kompatibilität.
Einer der Hauptvorteile von Matter ist es, dass entsprechend zertifizierte Smart-Home-Geräte einfacher und schneller eingerichtet und verwaltet werden können, ohne dass spezielle Kenntnisse oder technische Fähigkeiten erforderlich wären. Und da Sicherheit von größter Bedeutung ist, unterstützt das Protokoll auch eine End-to-End-Verschlüsselung, die gewährleistet, dass die zwischen den Geräten übertragenen Daten sicher sind.
Ein weiterer, für die meisten von uns wichtiger Vorteil ist, dass Matter vollständig quelloffen ist. Alle Informationen über Matter sind in einem GitHub-Repository [1] verfügbar, einschließlich Quellcode, Dokumentation, Skripte, Beispiele und alles, was das Entwicklerherz benötigt, um Matter-kompatible Geräte zu entwerfen.

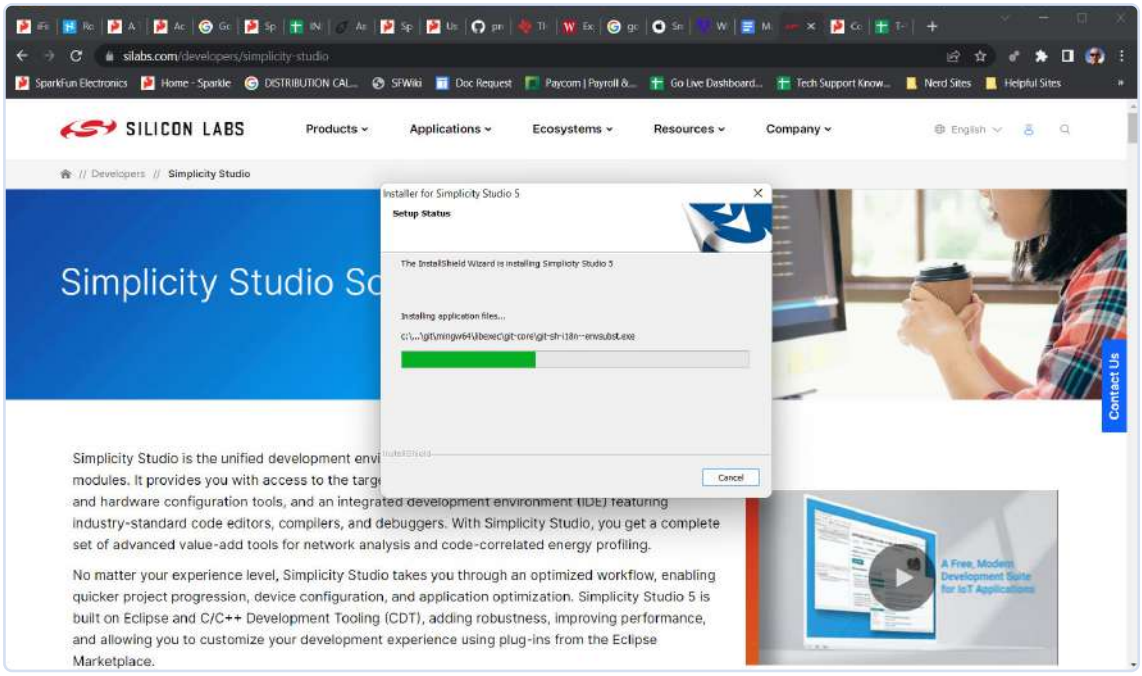


Bild 1. Installation der IDE Simplicity Studio.

So schön das auch ist, die meisten Menschen sind auf die Rolle des Verbrauchers beschränkt und haben keine Möglichkeit, Prototypen zu entwickeln, ohne ganz von vorne anzufangen. Glücklicherweise ändert sich dies mit dem kürzlich erschienenen *SparkFun Thing Plus Matter Development Board* von SparkFun Electronics [2], das im Elektor Store erhältlich ist (siehe **Passendes Produkt**). Es ist das erste leicht zugängliche Board seiner Art, das Matter und das Qwiic-Ökosystem von SparkFun für die flexible Entwicklung und das Prototyping von Matter-basierten IoT-Geräten kombiniert. Dieses MGM240P-Wireless-Modul von Silicon Labs bietet sichere Konnektivität sowohl für 802.15.4 mit Mesh-Kommunikation (Thread) als auch für Protokolle gemäß Bluetooth Low Energy 5.3. Das Modul kann leicht in das Matter-IoT-Protokoll von Silicon Labs für die Hausautomatisierung integriert werden. Die Thing-Plus-Entwicklungsboards von SparkFun sind Feather-kompatibel und besitzen einen Qwiic-Anschluss, wodurch lötfreie I²C-Verbindungen im Qwiic-Connect-System möglich sind.

Einrichtung des Simplicity Studios

Um unsere ersten Schritte als IoT-Entwickler mit Matter machen und das Thing Plus Matter Board mit einem Google Nest Hub verbinden zu können, müssen wir zunächst das Simplicity Studio von Silicon Labs installieren. Wir gehen die dazu erforderlichen Schritte hier kurz durch, aber auf der Website von SparkFun [3] finden Sie ein viel ausführlicheres Tutorial.

Die erste Maßnahme ist es, die aktuelle Version von Simplicity Studios (V 5, als dieser Artikels geschrieben wurde) für Ihr Betriebssystem von der Website von Silicon Labs herunterzuladen. Wenn Sie auf den Installer-Button für Ihr Betriebssystem klicken, werden Sie zu einer Anmeldeseite weitergeleitet. Wenn Sie noch kein Konto haben, sollten Sie jetzt eines erstellen, da Sie danach gefragt werden, wenn Sie das Simplicity Studio [4] öffnen. Sobald Sie das Programm heruntergeladen haben, führen Sie wie in **Bild 1** zu sehen den Installer aus.

Nachdem Sie das Simplicity Studio installiert und zum ersten Mal gestartet haben, werden Sie zum Installationsmanager weitergeleitet, der nach verfügbaren Updates sucht und diese gegebenenfalls auflistet. Hier können Sie einfach auf *Update All* klicken, damit Sie die neuesten Versionen von allem haben, was das Simplicity Studio zum Laufen braucht. Wenn keine weiteren Updates vorhanden sind oder Ihr System so konfiguriert ist, dass sie automatisch aktualisiert werden, springt der Installations-Manager sofort zum nächsten Schritt.

Nach dem (eventuellen) Neustart das Simplicity Studios gelangen Sie wieder zurück zum Installations-Manager, aber dieses Mal werden Sie gefragt, ob Sie Ihr Gerät durch Anschließen des Gerätes oder nach Technologietyp (Wireless, Xpress, MCU, Sensoren) installieren möchten. Hier wählen Sie *Install by connecting devices* (**Bild 2**) und schließen das Board SparkFun Thing Plus Matter über USB an.

Alsdann werden Sie gefragt, ob Sie die erforderlichen Pakete installieren möchten, was Sie natürlich tun wollen, also klicken Sie auf *Yes*. Sobald die Installation abgeschlossen ist, sollte der Installationsmanager *1 Device Found* anzeigen, mit einer Bezeichnung wie

☑ `J-Link (000449050174)(ID: 000449050174)`

Aktivieren Sie die Checkbox und klicken Sie auf *Next*, um zu den Paketinstallationsoptionen zu gelangen. Wenn Sie die Option *Auto* wählen, werden alle erforderlichen Pakete installiert, mit allem Zubehör, das Sie wollen. Es gibt auch eine Option *Advanced*, mit der Sie genau festlegen können, welche Pakete Sie installieren möchten und welche nicht, aber wenn Sie schon wissen, was Sie auswählen wollen, werden Sie diesen Artikel wahrscheinlich gar

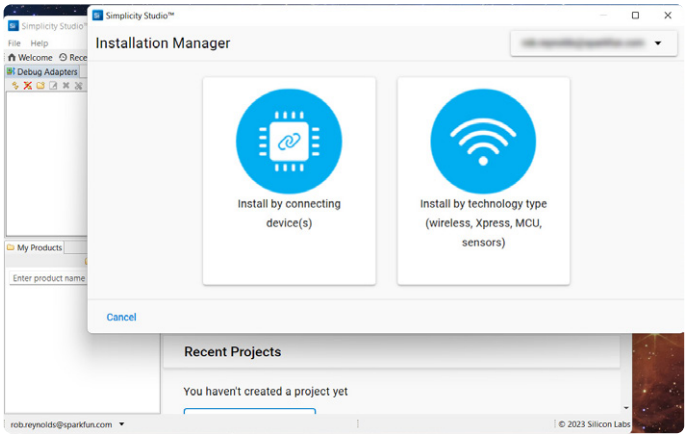


Bild 2. Installation durch Anschließen von Geräten im Simplicity Studio.

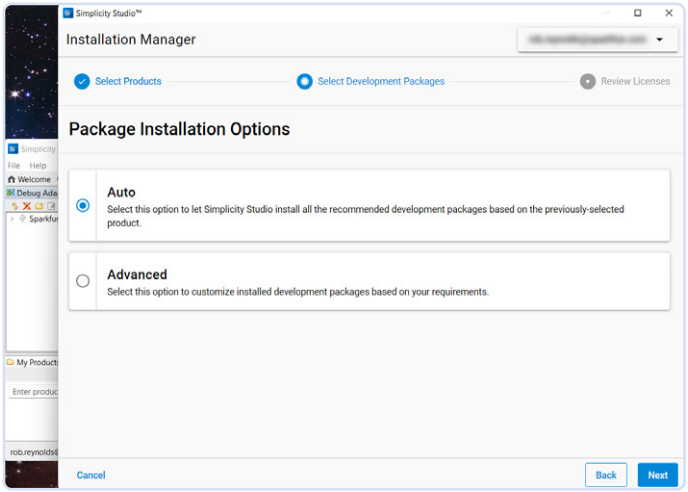


Bild 3. Optionen der Paket-Installation.

nicht lesen. Wählen Sie also die Option *Auto* und dann *Next*, wie in **Bild 3** dargestellt. Sie erhalten eine Master-Software-Lizenzvereinbarung, die Sie akzeptieren müssen, und dann beginnt endlich die Installation. Dies dürfte einige Zeit in Anspruch nehmen und erfordert am Ende einen Neustart, so dass es jetzt ein guter Zeitpunkt ist, ein Getränk oder einen Snack zu sich zu nehmen.

Wenn Sie zurückkommen und das Simplicity Studio neu starten, werden Sie von einer weiteren EULA begrüßt, und dann, nach den ganzen Vorarbeiten, mit einem *Welcome to Simplicity Studio*! Unter *Connected Devices* sollten Sie *SparkFun Thing Plus MGM240P* sehen. Klicken Sie auf *Start*, um die Thing-Plus-Informationsseite aufzurufen, auf der Sie einen Überblick über das Board sowie Beispielprojekte und Demos, Dokumentationen und kompatible Tools finden. Wenn Sie zur Registerkarte *Example Projects and Demos* navigieren, geben Sie in das Filterfeld das Stichwort „Blink" ein. Daraufhin sollte eine Reihe von Ressourcen angezeigt werden, von denen Sie die Ressource *Platform - Blink Bare-metal* wählen und auf die Schaltfläche *Create* klicken (**Bild 4**).

Daraufhin wird ein Fenster geöffnet, in dem Sie den Namen und/oder den Speicherort der Datei ändern können. Ich schlage vor, die Datei in etwas Sinnvolles umzubenennen, zum Beispiel *First-BlinkDemo*, um dann auf *Finish* zu klicken. Sobald das Projekt erstellt ist, sehen Sie auf der linken Seite des Simplicity Studios ein *Project-Explorer*-Fenster. Suchen Sie nach dem Hauptprojektordner, der den Namen *MatterBlinkExample* tragen sollte, klicken Sie rechts darauf und navigieren Sie dann zu *Run as/1 Silicon Labs ARM Program* (siehe **Bild 5**).

Wenn Sie darauf klicken, wird der Sketch kompiliert und auf Ihr Board Thing Plus Matter übertragen. Die blaue LED des Boards sollte nun im Halbsekundentakt blinken. Wenn Sie Ihr Thing-Plus-Board zum ersten Mal über USB angeschlossen haben, hat es wahrscheinlich schon im Halbsekundentakt geblinkt, bevor Sie überhaupt etwas getan haben. Um zu überprüfen, ob Ihr Code tatsächlich auf das Board geflasht wurde, können Sie das Blinkintervall ändern, indem Sie die Datei *blink.c* im *Project Explorer* aufrufen und das Intervall ändern. Etwa in Zeile 31 (**Bild 6**) sollten Sie Folgendes finden

```
#define TOOGLE_DELAY_MS 500
```

Ändern Sie den Wert einfach auf einen (deutlich) anderen, etwa 100, damit die LED extrem schnell blinkt, oder auf 3000, was ein sehr langsames Blinken zur Folge hat. Auf jeden Fall wissen Sie dann später, dass das Board tatsächlich mit Ihrem Code geflasht wurde. Sobald Sie diesen Wert geändert haben, können Sie erneut mit der rechten Maustaste auf den Ordner *MatterBlinkExample* klicken, zu *Run as/1 Silicon Labs ARM Program* navigieren, darauf klicken und beobachten, wie die blaue LED des Boards ihre Blinkfrequenz ändert.

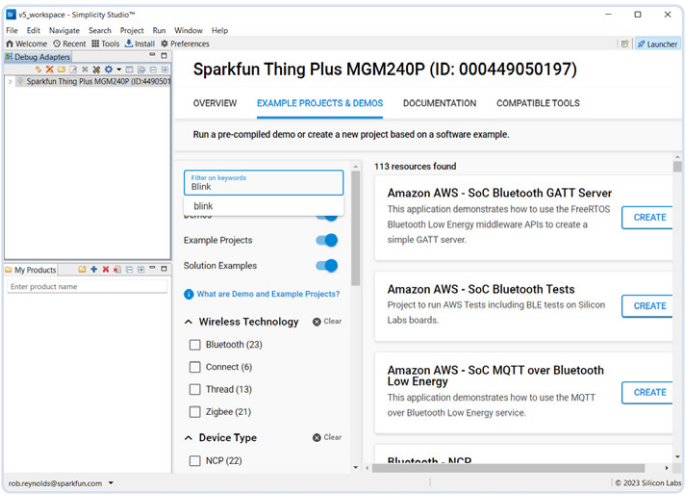


Bild 4. Der Tab „Example Projects and Demos".

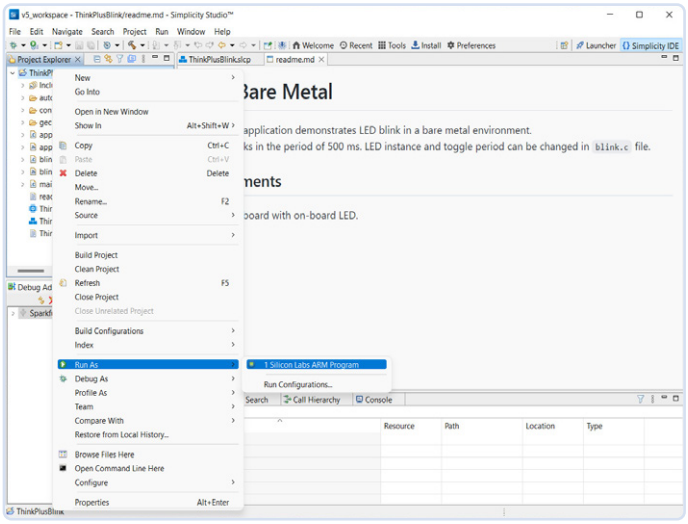


Bild 5. Start des Silicon-Labs-ARM-Programms.

Verbinden des Boards Thing Plus Matter mit dem Google Nest Hub (Tutorial)

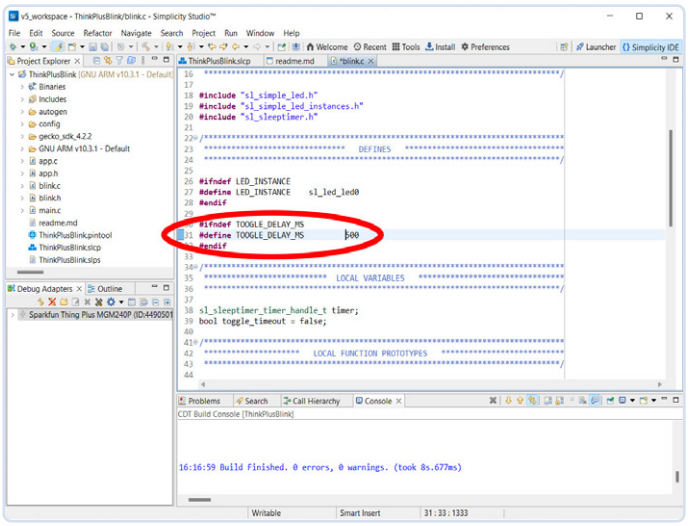


Bild 6. Ändern Sie den Wert, um die LED in einem anderen Tempo blinken zu lassen.

Verbinden des Boards mit Google Nest Hub

Herzlichen Glückwunsch, Sie kommunizieren jetzt mit Ihrem Board Thing Plus Matter über Matter mit dem Simplicity Studio von Silicon Labs. Von hier an werden die Dinge aber erst wirklich interessant. Die Kommunikation mit Google Hub ermöglicht es Ihnen, Ihre eigenen benutzerdefinierten Builds in Ihr Smart Home zu integrieren, mit Boards wie diesem Thing Plus Matter von SparkFun (**Bild 7**). Wenn Sie Lust haben, weiterzumachen, haben SparkFun-Ingenieur Drew und der Creative Technologist Mariah ein Video und eine Anleitung zusammengestellt, die genau erklären, wie man das macht [5].

Da diese Technologie noch sehr neu ist, gibt es nur wenige Beispiele und Tutorials, aber ihre Zahl wächst täglich. Wenn Sie jetzt mit Matter beginnen, sind Sie der Meute weit voraus, wenn sich dieses Protokoll über alle Plattformen hinweg durchsetzen und zur Vereinheitlichung der Smart-Home-Industrie beitragen wird.

RG — 230224-02

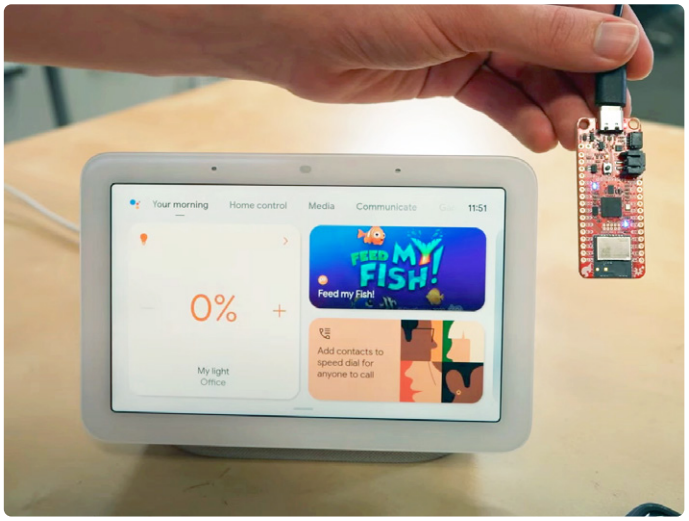


Bild 7. SparkFuns Entwicklungsboard Thing Plus Matter und Google Nest Hub.

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Schicken Sie eine E-Mail an das SparkFun-Supportteam unter support@sparkfun.com oder kontaktieren Sie Elektor unter redaktion@elektor.de.

Über den Autor

Rob Reynolds ist seit 2015 bei SparkFun und seit fünf Jahren in der Rolle des Creative Technologist tätig. Sein umfangreicher künstlerischer Hintergrund hilft ihm, Projekte, Tutorials und Videos zu erstellen, die in der Regel so unterhaltsam und amüsant wie informativ sind. Sie finden sie auf Twitter unter @thingsrobmade.

Passendes Produkt

- **SparkFun Thing Plus Matter (MGM240P)**
  https://elektor.de/20442

Sparkfun-Produktseite für weitere Informationen

WEBLINKS

[1] Matter auf GitHub: https://github.com/project-chip/connectedhomeip
[2] SparkFun-Entwicklungsboard Thing Plus Matter: https://www.sparkfun.com/products/20270
[3] Produktseite MGM240P: https://bit.ly/42NRVII
[4] Simplicity Studio: https://silabs.com/developers/simplicity-studio
[5] Verbindung von Thing Plus Matter mit Google Nest Hub: https://bit.ly/3VRcQCl

Elektor Infografik

IoT und Sensoren

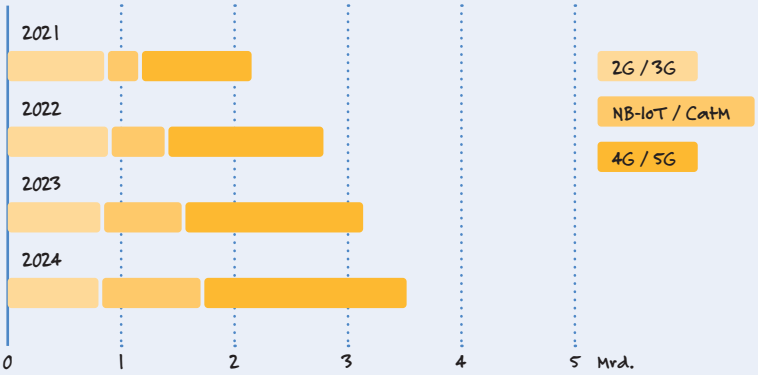
Das IoT im Blick

Das IoT ist alles andere als ein alter Hut. Man könnte sogar behaupten, dass die IoT-Innovation gerade erst in den Kinderschuhen steckt. Laut einer McKinsey-Studie könnte der Gesamtwert des IoT bis zum Jahr 2030 12,5 Billionen Dollar erreichen [1]. Sensoren sind die Schlüsselkomponenten des IoT und spielen auch in unzähligen Anwendungen, die von Mitgliedern der Elektor-Community entwickelt und eingesetzt werden, eine wesentliche Rolle. Der Gesamtmarkt für IoT-Sensoren wird von etwa 10,9 Milliarden Dollar im Jahr 2022 auf 22,1 Milliarden Dollar im Jahr 2027 ansteigen [2]. Dieses potenzielle Wachstum eröffnet neue Möglichkeiten für professionelle Ingenieure und Hersteller gleichermaßen.

Fünf IoT-Sensorik-Trends

- Intelligente Sensoren
- Energieeffiziente Sensoren
- Weiche und virtuelle Sensoren
- Sensor-Fusion
- Biosensoren

Zellulare IoT-Verbindungen nach Segment und Technologie



Quelle: Ericsson [3]

12,5 Billionen

McKinsey-Schätzung des Gesamtwerts in Dollar des IoT bis 2030.

Das Matter-Protokoll

Das Matter-Protokoll ermöglicht es Smart-Home-Geräten, unabhängig von ihrem Hersteller miteinander zu kommunizieren. Es standardisiert die Einrichtung von Geräten verschiedener Hersteller. Laut der **Connectivity Standards Alliance** „ermöglicht Matter auf der Grundlage des Internetprotokolls (IP) die Kommunikation zwischen Smart-Home-Geräten, mobilen Anwendungen und Cloud-Diensten und definiert einen spezifischen Satz von IP-basierten Netzwerktechnologien für die Gerätezertifizierung“ [4]. Von Matter unterstützte Geräte sind: Bridges, Controller, Steuerungen, Türschlösser, HLK-Steuerungen, Beleuchtung und Elektrik, Mediengeräte, Sicherheits- und Schutzsensoren sowie Fensterabdeckungen und Jalousien.

37%

Prozentualer Anteil der Haushalte mit Internetanschluss, die ein Smart Home-Gerät besitzen [6].

68%

Prozentualer Anteil der Gerätebesitzer (oder zukünftiger Besitzer), die eine Matter-Zertifizierung für wichtig halten [6].

Von Matter unterstützte Geräte

Steuerungen der Klimatechnik | Fensterabdeckungen (Rollos, Jalousien) | Sicherheits-technik-Sensoren | Beleuchtung und Elektrik | Türschlösser | Mediengeräte | Controller und Bridges

Quelle: CSA [5]

5G-Technologie

5G - die fünfte Generation der drahtlosen Mobilfunktechnologie - bietet Unternehmen und Verbrauchern zahlreiche Vorteile: immer schnellere und sicherere Konnektivität, geringere Latenzzeiten, längere Akkulaufzeiten und mehr. Seit ihrer Einführung im Jahr 2019 hat sich 5G zu einer der wichtigsten Technologien entwickelt, die die Branche beeinflussen.

2 Billionen

Potenzieller Anstieg in Dollar des weltweiten BIP, wenn 5G in den wichtigen Wirtschaftsbereichen Einzelhandel, Fertigung, Gesundheitswesen und Mobilität eingesetzt wird [8].

50%

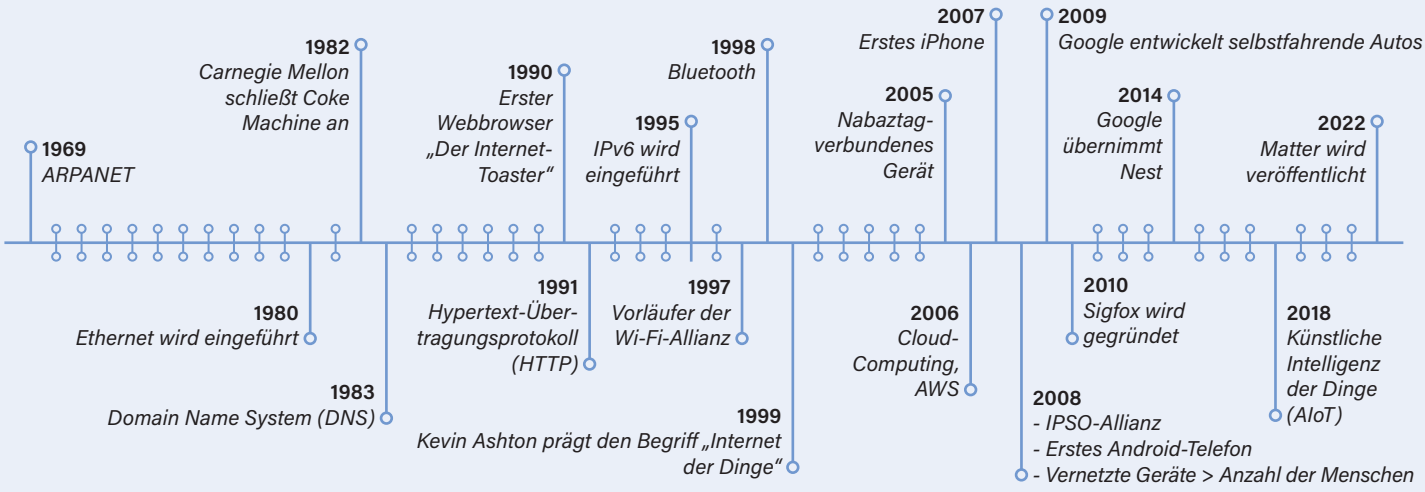
Prozentualer Anteil der Befragten einer IEEE-Umfrage, die 5G zu den fünf wichtigsten Technologien im Jahr 2022 zählen. [7]

Wo sehen vom IEEE Befragte [7] die Rolle von 5G?

- Fernlernen
- Telemedizin
- Unterhaltung
- Alltägliche Kommunikation
- Transport und Verkehrskontrolle
- Fertigung/Montage
- Energie-Effizienz

IoT-Zeitleiste

Der Technikpionier und Autor Kevin Ashton prägte Ende der 1990er Jahre den Begriff „Internet der Dinge“. Die folgende Zeitleiste zeigt einige Schlüsselmomente der Geschichte des Internets der Dinge.



WEBLINKS

[1] McKinsey & Company, „What is the Internet of Things?“, 17. August 2022: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-the-internet-of-things
[2] S. Sinha, „5 IoT sensor technologies to watch“, IoT Analytics, 4. Januar 2023: https://iot-analytics.com/5-iot-sensor-technologies/
[3] Ericsson, „IoT Connections Outlook“: https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/iot-connections-outlook
[4] CSA, „Matter: The Foundation for Connected Things“: https://csa-iot.org/all-solutions/matter/
[5] CSA, „Matter Executive Overview“: https://csa-iot.org/wp-content/uploads/2022/09/22-Matter-Executive-Overview-One-Pager.pdf
[6] C. White, „The Wait Is Over and Matter 1.0 Is Here“, Parks Associates, 6. Oktober 2022: https://www.parksassociates.com/blog/article/matter-is-here
[7] IEEE, „Advancing Connectivity in 2023“, IEEE Transmitter, 24. Oktober 2022: https://transmitter.ieee.org/advancing-connectivity-in-2023/
[8] McKinsey & Company, „What Is 5G?“, 7. Oktober 2022: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-5g

Q&A mit Amey Inamdar

Technical Marketing Director bei Espressif

Matter, ExpressLink, Rainmaker – Worum geht es eigentlich?

Fragen von Tam Hanna (Slowakei) und Jens Nickel (Elektor)

Auf der embedded world 2023 in Nürnberg hat Espressif, bekannt für den berühmten ESP32-Mikrocontroller, eine Menge Automatisierungslösungen vorgestellt, die das Leben von IoT-App-Entwicklern (und Nutzern) erleichtern sollen. Es ist jedoch nicht einfach zu verstehen, was sich hinter neuen Begriffen wie ExpressLink und RainMaker verbirgt. In diesem Interview klärt Amey Inamdar, Technical Marketing Director bei Espressif, darüber auf. Außerdem beantwortet er allgemeine Fragen zum Portfolio seines Unternehmens.

Elektor: Bitte erzählen Sie uns etwas über Espressif heute. Was ist der Schwerpunkt? Wie hat sich das Unternehmen seit seiner Gründung im Jahr 2008 verändert?

Amey Inamdar: Espressif konzentrierte sich auf die Demokratisierung des IoT-Segments mit innovativen, entwicklerbezogenen und erschwinglichen WLAN-Konnektivitätslösungen. Wir haben dafür gesorgt, dass unsere Hardware leicht zugänglich und unsere Software in der Open-Source-Community verfügbar ist. Dies ist auch heute noch eine Kernphilosophie von Espressif, doch unser Fokus hat sich mit den veränderten Marktanforderungen erweitert.

Espressif verbessert nicht nur weiterhin die WLAN- und BLE-Konnektivität, indem Unterstützung für Wi-Fi 6, Dualband und Bluetooth LE(5) zum Portfolio hinzugefügt werden, sondern geht auch auf die neuen Anforderungen des Marktes durch die Unterstützung von 802.15.4-Funk als Grundlage für Thread- und ZigBee-Protokolle ein. Wir haben kontinuierlich den Strombedarf und Aspekte der HF-Leistung der Konnektivität verbessert. Außerdem haben wir unsere Produkte mit modernster Schnittstellenperipherie ausgestattet. Darüber hinaus hat Espressif mit seinen ESP32- und ESP32-S3-Chips Pionierarbeit bei Multi-Core-MCUs geleistet. Die KI-Beschleunigungsunterstützung im ESP32-S3 ist für Anwendungen des ML-on-the-Edge von Vorteil. Hardware-Beschleuniger und Medien-Encoder wurden ebenfalls zu neueren Chipsätzen hinzugefügt. Darüber hinaus bieten die Espressif-SoCs Sicherheitsfunktionen, so dass alle damit aufgebauten Geräte die Sicherheitsanforderungen erfüllen. Die meisten Chips verfügen auch über innovative Sicherheitsperipherie, zum Beispiel für digitale Signaturen, eine integrierte Funktionalität in Form eines sicheren Hardwaremoduls.

Darüber hinaus hat sich Espressif zu einem Anbieter von Komplettlösungen entwickelt, der die Probleme seiner Kunden erkennt und sie mit Lösungen, die über Hardware und SDK hinausgehen, wirkungsvoll versorgt. ESP RainMaker, ESP Insights und ESP ZeroCode-Module sind gute Beispiele dafür.

Elektor: Auf der embedded world hat Espressif Lösungen für die Hausautomatisierung vorgestellt, bei denen ESP32-Boards mit AWS verbunden sind. Wir haben sowohl von *RainMaker* als auch von *ExpressLink* gehört. Können Sie uns mehr über diese beiden Lösungen erzählen? Sind sie unabhängig voneinander oder können sie zusammenarbeiten?

Amey Inamdar: ESP RainMaker ist eine IoT-Cloud-Implementierung, die Sie mit Ihrem eigenen AWS-Konto verknüpfen können. Es verfügt auch über ein Open-Source-Firmware-SDK, Smartphone- und Sprachassistentenanwendungen. ESP RainMaker basiert auf der serverlosen Architektur von AWS und nutzt intern AWS IoT Core und die zugehörigen Dienste.

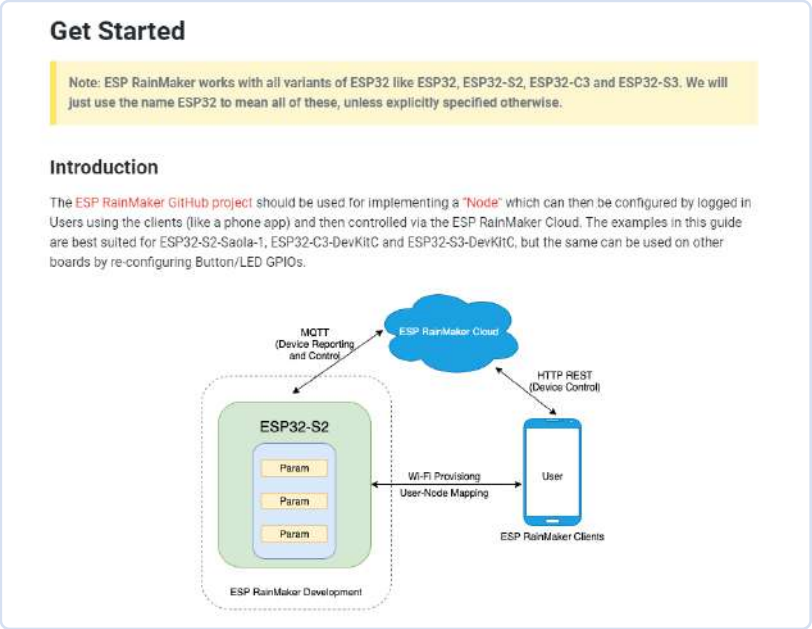
ExpressLink ist ein Konnektivitätsmodul, das eine einfache AT-Befehlsschnittstelle zur Host-MCU bereitstellt und eine nahtlose Verknüpfung zu AWS IoT Core und zugehörige Dienste wie OTA bietet. ExpressLink vereinfacht die geräteseitige Komplexität bei der Entwicklung und Verwaltung verbundener Geräte.

ESP RainMaker und ExpressLink ergänzen sich. Kunden können das eine oder andere oder beides zusammen verwenden, um ohne Schwierigkeiten vernetzte Geräte zu entwickeln.

Elektor: Lassen Sie uns mit RainMaker [1] beginnen. Laut Dokumentation sind die RainMaker-Funktionen über das ESP-IoT-Development-Framework ESP-IDF zugänglich. Seit Herbst gibt es auch eine Arduino-Schnittstelle. Würden Sie für den professionellen Einsatz immer noch das IDF empfehlen?
Amey Inamdar: ESP-IDF ist ein SDK für die Entwicklung von IoT-Anwendungen. Es ist nicht das einzige Software-Framework, aber ESP-IDF ist das Projekt, in dem wir als erstes die Unterstützung für neu veröffentlichte Produkte aufnehmen. ESP-IDF ist ein Open-Source-Projekt und bildet auch die Grundlage für viele andere Espressif-Software-Frameworks, Anwendungen und Lösungen.
Arduino bietet eine einfache Schnittstelle und ermöglicht die Nutzung vorhandener Bibliotheken und Peripherietreiber. ESP-IDF bietet dagegen mehr Flexibilität für Kunden, die Multithreading-Anwendungen mit Zugriff auf alle nativen SDK-APIs entwickeln möchten. Kunden können je nach Anwendung zwischen der Arduino-Schnittstelle und IDF wählen.

Elektor: Viele Anwender scheinen RainMaker zu mögen, aber sie zögern, es in der AWS-Cloud zu hosten. Kann Rainmaker auch ohne AWS genutzt werden oder ist so etwas für die Zukunft geplant?
Amey Inamdar: Es gibt mehrere Ansätze für den Aufbau von IoT-Cloud-Plattformen. Man könnte sie mit *Platform as a Service* (PaaS) aufbauen, indem man einfach Container oder virtuelle Maschinen von Cloud-Unternehmen verwendet. Solche Cloud-Implementierungen erfordern jedoch einen hohen technischen Aufwand, damit sie skalierbar und kostengünstig sind. Nur sehr wenige Kunden können solche DevOps verwalten. Daher haben wir uns entschieden, ESP RainMaker auf der AWS-Serverless-Architektur aufzubauen, die eine wartungsfreie Lösung mit Pay-as-you-go-Preisen bietet. So gesehen haben wir bewusst zu Gunsten der Kunden entschieden, weil es nicht einfach ist, RainMaker ohne AWS zu nutzen.

Elektor: Die RainMaker-Spezifikationen sehen Standard-Gerätetypen vor; die meisten davon sind im Bereich der Hausautomation angesiedelt (wie Jalousien, Ventilatoren und Einbruchmeldeanlagen) [2]. Das System könnte jedoch auch für ganz andere Anwendungen eingesetzt werden, zum Beispiel für Studiobeleuchtung und Videoausrüstung, ganz zu schweigen von typischen industriellen Produktionsanlagen. Planen Sie, den Anwendungsbereich zu erweitern, oder sehen Sie RainMaker nur im Bereich der Hausautomation?
Amey Inamdar: Das Cloud-Backend von ESP RainMaker ist völlig unabhängig von den Gerätetypen. Es fungiert als „Durchreiche“ für die Steuerung und als Zeitreihen-Datenspeicher. Es ist also nicht schwierig, einen bestimmten Gerätetyp zu unterstützen. Selbst die in den App-Stores angebotenen Smartphone-Anwendungen generieren die Benutzeroberfläche dynamisch auf der Grundlage der vom Gerät bereitgestellten Beschreibung. Wenn das Gerät zum Beispiel angibt, dass es sich um einen Soundmixer mit acht verschiedenen Frequenzbereichen handelt, die jeweils als Schieberegler von einem Minimal- bis zu einem Maximalwert eingestellt werden, wird die Benutzeroberfläche von den Telefonanwendungen automatisch gerendert, ohne dass eine Änderung auf der Cloud-Seite erforderlich wäre.



▲ *Bild 1. RainMaker verbindet Clients wie Smartphones und ESP32-basierte Geräte über ein Cloud-Backend, das auf AWS basiert.*

Elektor: Wäre es nicht eine gute Idee, Rainmaker für andere Mikrocontroller-Firmen zu öffnen, zum Beispiel für Microchip?
Amey Inamdar: Das ESP-RainMaker-Protokoll und das Geräte-SDK sind vollständig quelloffen, und es macht uns nichts aus, wenn Entwickler ESP RainMaker mit MCUs verwenden, die nicht von Espressif stammen.

Elektor: Kommen wir zu ExpressLink [3]. Im Internet ist zu lesen, dass „ExpressLink-kompatible Module eine einfache serielle Schnittstelle bieten, über die sich die Host-MCU mit AWS-IoT-Diensten verbindet und so jedes Offline-Produkt in ein mit der Cloud verbundenes Produkt verwandelt.“ Wie wir auf einer AWS-Webseite [4] von Espressif sehen können, scheint es aber nur ein kompatibles Board zu geben.
Amey Inamdar: Ja, und das ist nur eine Entwicklungsplatine für die Evaluierung und das Prototyping rund um das ExpressLink-kompatible Modul (*bei dem es sich im Grunde um ein ESP32-C3-Modul mit vorprogrammierter Firmware handelt, das die ganze AWS-Magie übernimmt - Anm. der Red.*). Wir haben einen Arduino-kompatiblen Formfaktor für dieses Entwicklungsboard ESP32-C3-AWS-ExpressLink-DevKit gewählt, und das Pin-Layout ist kompatibel mit dem Arduino-Zero-Board und kann direkt aufgesteckt werden. Es kann aber auch leicht an andere Host-MCUs wie den Raspberry Pi angeschlossen werden.

> *Die Smart-Home-Industrie benötigt einheitliche Standards*

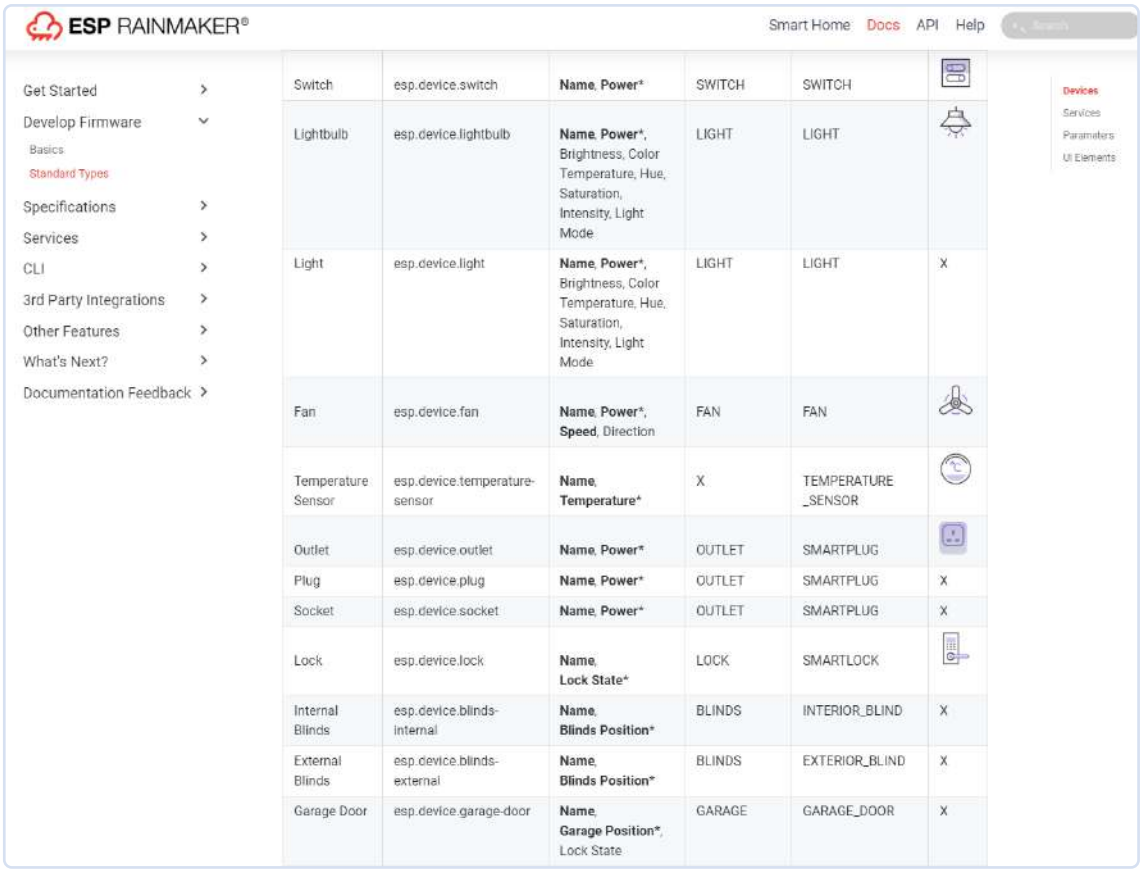


Bild 2. RainMaker verfügt über ein leistungsstarkes SDK und eine lange Liste vordefinierter Gerätetypen.

Elektor: Wenn wir über Matter [5] sprechen, ist Espressif einer der ersten und bekanntesten Anbieter. Wie sind Sie in eine solch einzigartige Position gekommen? Woran haben Sie den Nutzen dieses neuen Standards erkannt?

Amey Inamdar: Die Smart-Home-Branche brauchte eine Standardisierung - frühere Versuche waren erfolglos. Die Zersplitterung macht es den Verbrauchern schwer, die verbundenen Geräte zu nutzen, und den Herstellern schwer, sie zu bauen. Diesmal haben sich die größeren Akteure des Ökosystems unter dem Dach der *Connectivity Standards Alliance* zusammengeschlossen und ihren Willen für eine erfolgreiche Standardisierung gezeigt. Darüber hinaus haben die Überlegungen zu der Ausstattung des Protokolls wesentlich zu seinem Erfolg beigetragen. Bemerkenswerte Beispiele sind die Gewährleistung kryptografischer Sicherheit für die gesamte Kommunikation, Unterstützung von WLAN- und Thread-Transporten, Verwendung von Blockchain für die Authentifizierung, sichere OTA und einiges anderes.

Espressif ist in der einzigartigen Lage, eine umfassende Lösung für Matter mit Hardware, Software, gebrauchsfertigen Lösungen und Dienstleistungen anzubieten. Espressif bietet Chips und Module für den Aufbau von WLAN-Matter- und Thread-Matter-Zubehör, Thread-Border-Router sowie Matter-Bridges. *Esp-matter* ist ein Open-Source-SDK, das Tools und

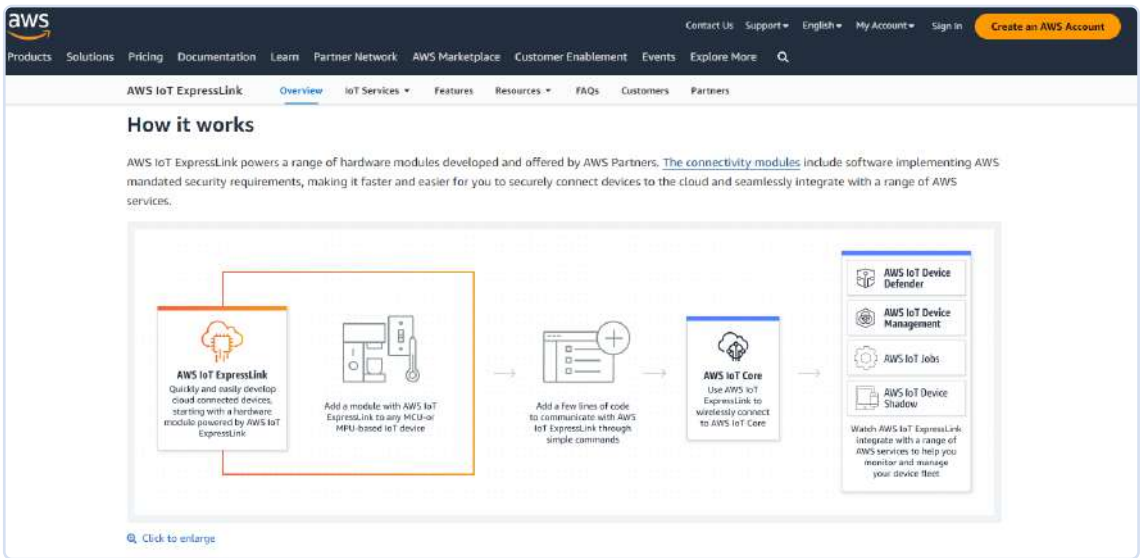


Bild 3. ExpressLink ist ein Konnektivitätsmodul, das eine einfache AT-Befehlsschnittstelle zur Host-MCU bietet.

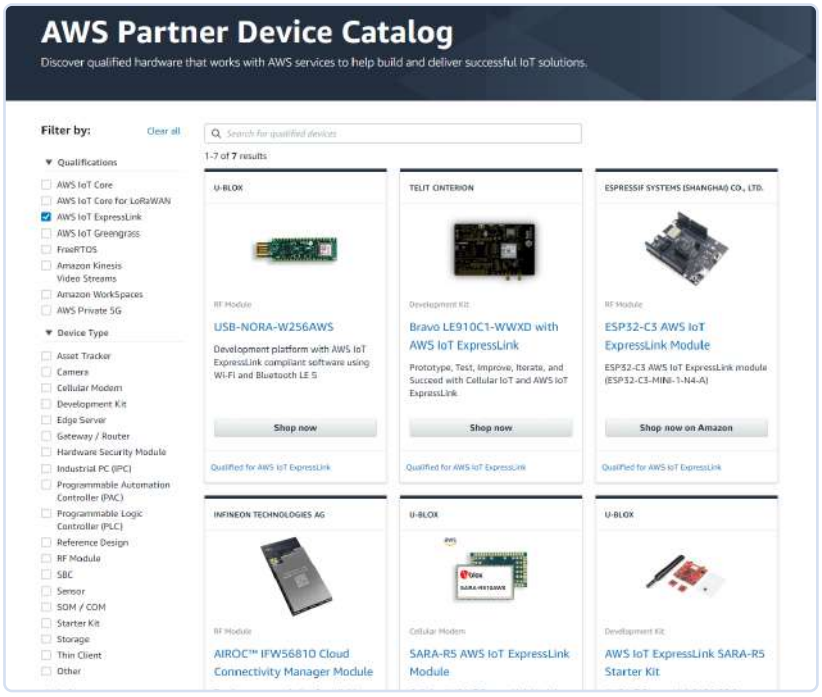


Bild 4. Derzeit sind nur wenige ExpressLink-Entwicklungsboards von Espressif und anderen Unternehmen verfügbar.

Beispiele zur Erstellung von Matter-Geräten bietet. Durch die Bereitstellung von Matters *Device Attestation Certificate* kann Espressif zertifizierte Module anbieten, so dass sich die Kunden nicht mehr um eine sichere und komplexe Herstellung kümmern müssen. Der Zertifizierungsservice von Espressif hilft Kunden, ihre Matter-kompatiblen Geräte zertifizieren zu lassen. Und ESP-ZeroCode-Module werden mit vorzertifizierter Firmware für einfache Geräte geliefert, so dass Kunden direkt und ohne den geringsten Entwicklungsaufwand Matter-kompatibles Zubehör bauen können.

Elektor: Mit dem ESP32-P4 bricht Espressif aus dem traditionellen Markt für Kommunikationsmodule aus. Wenn man bedenkt, dass Anbieter wie ST und Microchip ihren Kunden eine automatische Codegenerierung anbieten, welches Alleinstellungsmerkmal sehen Sie dann für den P4?
Amey Inamdar: Der ESP32-P4 verfügt über eine ganze Reihe guter Eigenschaften, etwa eine höhere Rechenleistung, verbesserte Peripherie und Speicherarchitektur. Der ESP32-P4 kann mit jedem anderen Espressif-Konnektivitäts-Chip gekoppelt werden und dann einige interessante Anwendungen bei höherwertigen IoT-Geräten ermöglichen. Das wichtigste Alleinstellungsmerkmal von ESP32-P4 ist die standardisierte Softwareunterstützung, bei der das gleiche ESP-IDF den ESP32-P4 unterstützt und es Entwicklern ermöglicht, ihre Lern- und Portierungsanwendungen nahtlos von anderen Espressif-SoCs auf den P4 zu übertragen. Darüber hinaus wird der Rest des Ökosystems - mit Hochsprachenineterpretern, RTOS-Unterstützung, SDKs und Softwarekomponenten - weiterhin auf dem ESP32-P4 nutzbar sein.

Elektor: Angesichts der Tatsache, dass immer mehr Anbieter in den Modulbereich drängen, was planen Sie, um langfristig auf dem Markt zu bleiben?
Amey Inamdar: Wir wollen weiterhin die richtigen Dinge für unsere Kunden tun, und wir glauben, dass uns das helfen wird, unsere führende Position langfristig zu halten. Ich glaube nicht, dass es dafür nur eine Richtung gibt. Ein hohes Maß an Integration in unseren SoC, Innovation und die Wahl der SoC- und Kommunikations-Subsystem-Architektur, Offenheit in Bezug auf Software und Informationen, keine Kompromisse bei den Sicherheitsmerkmalen, eine effiziente Lieferkette und eine flexible Fertigung sind einige dieser Unterscheidungsmerkmale.

RG — 230227-02

Über Amey Inamdar

Amey arbeitet als Direktor für technisches Marketing bei Espressif. Er verfügt über eine 20-jährige Erfahrung im Bereich eingebetteter Systeme und vernetzter Geräte und war in den Bereichen Technik, Produktmanagement und technisches Marketing tätig. Er hat mit vielen Kunden zusammengearbeitet, um erfolgreich vernetzte Geräte auf der Grundlage von WLAN- und Bluetooth-Konnektivität zu entwickeln.

WEBLINKS

[1] Webpage ESP RainMaker : https://rainmaker.espressif.com/
[2] Vordefinierte Standard-Gerätetypen von ESP RainMaker: https://rainmaker.espressif.com/docs/standard-types.html
[3] Webseite AWS IoT ExpressLink: https://aws.amazon.com/iot-expresslink/
[4] ExpressLink-Produkte im AWS-Partner-Device-Katalog: https://devices.amazonaws.com/search?page=1&sv=iotxplnk
[5] Espressif-Lösungen für Matter: https://espressif.com/en/solutions/device-connectivity/esp-matter-solution

Development-Kits für IoT- und IIoT-Applikationen

Über die Auswahl von Mikrocontroller-Entwicklungsboards

Von Mark Patrick (Mouser Electronics)

Das Internet der Dinge (Internet of Things, IoT) ist allgegenwärtig. Entwickler von Embedded-Systemen müssen bei der Erstellung neuer IoT-Designs zahlreiche Faktoren wie Stromverbrauch, Sensorik und drahtlose Konnektivität genau im Auge behalten. Der Zeitdruck bei der Markteinführung verschärft diese Situation noch. Hierfür bieten IoT-Development-Kits eine praktikable und bequeme Prototyping-Plattform als Basis für ein Design. Die Möglichkeiten von IoT-Development-Kits sind jedoch sehr unterschiedlich, sodass die Anforderungen der Applikation sowie die Funktionen und Möglichkeiten des Kits sorgfältig geprüft werden müssen. In diesem Artikel gehen wir auf einige Aspekte ein, die bei der Auswahl eines IoT-Development-Kits zu beachten sind.

Das Online-Zeitalter

Wir befinden uns zweifelsohne im Online-Zeitalter. Überall um uns herum befinden sich vernetzte Geräte. Einige tragen wir am Körper, andere helfen uns, unseren Stromverbrauch genau zu überwachen, und wieder andere benachrichtigen uns, wenn ein Gast vor unserer Tür steht. In der Industrieproduktion verändert das industrielle Internet der Dinge (IIoT) die Art und Weise, wie Fabriken arbeiten, und verbessert die Effizienz der Anlagen insgesamt. Innerhalb von nur einem Jahrzehnt haben wir die Interaktion und die Kontrolle über die Welt um uns herum grundlegend verändert. Wir haben uns gewundert, wie wir früher ohne Mobiltelefone zurechtkommen konnten, heute haben wir uns daran gewöhnt, dass wir stets sofortigen Zugang zu Informationen über viele Aspekte unseres Lebens und unserer Arbeit haben. Auch unsere Fahrzeuge erleben einen radikalen Wandel: Aktuelle Informationen über den Verkehrsfluss warnen uns vor möglichen Verspätungen. Dank der mit dem Internet verbundenen Überwachungsgeräte für die Gesundheitsversorgung können sich Patienten bequem zu Hause erholen und sich darauf verlassen, dass das Klinikpersonal sie überwacht und zur Stelle ist, falls ein Eingreifen erforderlich ist.

Das IoT wurde auch von der Industrie zügig angenommen, da aufgrund von politischen

Initiativen wie Industrie 4.0 der Bedarf an Automatisierung, Verbesserung der Prozesseffizienz und rationelleren Abläufen immer größer wurde. Mittlerweile überwachen unzählige Sensoren den Status jeder einzelnen Prozessphase und melden Daten an das Automatisierungssteuerungs- und -analysesystem zurück.
Die Vorteile von IoT/IIoT-Applikationen sind beträchtlich, aber aus Perspektive der Elektrotechnik sind sie auch mit vielen Herausforderungen bei der Entwicklung eines IoT-Geräts verbunden.

Erforschung der Anforderungen eines IoT-Geräts

IoT-Applikationen sind sehr unterschiedlich, aber die grundlegenden funktionalen Anforderungen sind in der Regel immer gleich, egal ob es sich um einen Drucksensor für einen industriellen Prozess oder einen Raumbelegungssensor in einem Büro handelt.
Bei der anfänglichen Bestandsaufnahme zur Erstellung der technischen Spezifikationen für ein neues IoT-Gerät sollten alle im Folgenden genannten Aspekte berücksichtigt werden, da sie die funktionale Architektur und das Design bestimmen.

Erkennung: Sensoren erfassen die Welt um uns herum, von der Temperatur über den Luftdruck bis hin zu den Bewegungen von Personen. Eine Kamera kann beispielsweise Daten in eine Anwendung für Machine Learning zur Objekterkennung übertragen und damit bestätigen, dass ein Etikett korrekt auf eine Flasche geklebt wurde. Mehrere technische Entscheidungen hängen davon ab, was und wie häufig es erkannt werden soll. Kosten, Größe und Komplexität der Sensoren sind weitere Faktoren. So sind für einen Thermistor, der zur Temperaturmessung verwendet wird, zusätzliche Komponenten für den analogen Bereich und eine gewisse Softwareverarbeitung vor der Umwandlung in eine digitale Form erforderlich. Ein weiterer Faktor betrifft die Anzahl der benötigten Sensoren und die Häufigkeit, mit der sie abgefragt werden sollen.

Konnektivität: Wie interagiert das IoT-Gerät mit einem übergeordneten Steuersystem? Ist in jedem Anwendungsszenario eine zuverlässige Wireless-Kommunikation verfügbar, oder wird eine kabelgebundene Kommunikation bevorzugt? Der Sensortyp bestimmt auch die Menge der zu übertragenden Daten und die Häufigkeit der Übertragung. Bei einem breiten Einsatz bietet die drahtlose Mesh-Technologie möglicherweise eine robustere Kommunikationsverbindung. Das setzt jedoch voraus, dass alle IoT-Geräte auf diese Art und Weise kommunizieren. Bei der drahtlosen Kommunikation steht man vor der Entscheidung, ein eigenständiges Design zu entwickeln oder sich für ein typgeprüftes Modul zu entscheiden.

Stromquelle: Welches Stromverbrauchsprofil wird das IoT-Gerät voraussichtlich haben? Einige Applikationen, Kommunikationsfrequenzen und Drahtlos-Protokolle stellen erhebliche Anforderungen an die Stromversorgung, die die Kapazität einer kleinen Batterie übersteigen. Steht für einige Einsatzszenarien möglicherweise eine Netzstromversorgung zur Verfügung? Der jüngste Trend bei IoT-Sensoren nutzt Technologien zur Energiegewinnung, um gänzlich auf eine Batterie zu verzichten. Stattdessen wird Energie aus der Umgebung, zum Beispiel durch Sonneneinstrahlung, Vibrationen und Wärme, gewonnen, um einen Superkondensator zu laden.

Benutzerschnittstelle: Ist für das IoT-Gerät eine Benutzerinteraktion erforderlich? Wie sieht es mit der Installation und dem Anschluss an das Host-System aus, wenn dieses nicht in Betrieb ist? Ist ein Display oder eine andere Form von Anzeige oder Status-LEDs erforderlich?

Cloud-Analyse und Steuerungsanwendungen: Das IoT zeichnet sich dadurch aus, dass die Geräte mit einem Steuerungs-Hostsystem verbunden sind. Dabei bestimmen die Konnektivitätsmethode und die Protokolle die Softwareanforderungen des Sensors und die Art und Weise, wie er mit dem Hostsystem interagiert. Ist eine ständige Datenverbindung erforderlich, um Daten zu streamen, oder können sie in regelmäßigen Abständen als Batch gesendet werden?

Hinweise und Tipps zur Auswahl von IoT-Development-Kits

Mit Development-Kits können Embedded-Entwickler schnell und einfach Prototypen eines Designs erstellen. In diesem Abschnitt des Beitrags stellen wir einige der Faktoren vor, die Entwickler bei der Auswahl eines geeigneten Kits berücksichtigen sollten. Die führenden Mikrocontroller-Hersteller bieten eine große Auswahl an IoT-Development- und -Evaluierungskits an. Daher ist es sinnvoll, eine fundierte Entscheidung auf der Grundlage der oben genannten Anwendungsanforderungen zu treffen. Im Folgenden sind einige Merkmale aufgeführt, auf die Sie bei der Auswahl einer Development-Kit-Plattform achten sollten.

Stromversorgung

- Wie wird das Board mit Strom versorgt? Über USB von einer Host-Workstation? Batterie? Kann es über die vorgesehene Stromquelle versorgt werden, und verfügt es über einen Schaltkreis zur Energieverwaltung (PMIC), auf den Sie zugreifen können, um andere Stromquellen auszuprobieren?
- Ist es möglich, einen Stromzähler zur Messung des Echtzeit-Stromverbrauchs zu Profilierungszwecken in das Board zu integrieren? Falls ja, umfasst diese Messung alle Komponenten auf dem Board und alle zusätzlichen Module, Sensoren und so weiter?

Sensoren

- Ist das Board mit den Sensortypen ausgestattet, die in Ihrer Applikation zum Einsatz kommen?
- Ist es möglich, zusätzliche Sensoren zu integrieren? Entweder über einen Peripherieanschluss oder über ein Industriestandard-Add-on-Format wie zum Beispiel mikroBUS Click?
- Welche Peripherieschnittstellen sind für den Zugriff verfügbar? I²C, UART, SPI, GPIO?
- Verfügt das Board oder der Mikrocontroller über einen ADC, der verwendet werden kann, und sind zusätzliche Signalaufbereitungskomponenten erforderlich?

Konnektivität

- Welche drahtgebundenen/drahtlosen Anschlussmöglichkeiten bietet das Board? Ethernet, WiFi, LoRa, BLE, ISM und so weiter.
- Wenn das Board keine Konnektivität bietet, kann sie leicht hinzugefügt werden? Empfiehlt und unterstützt

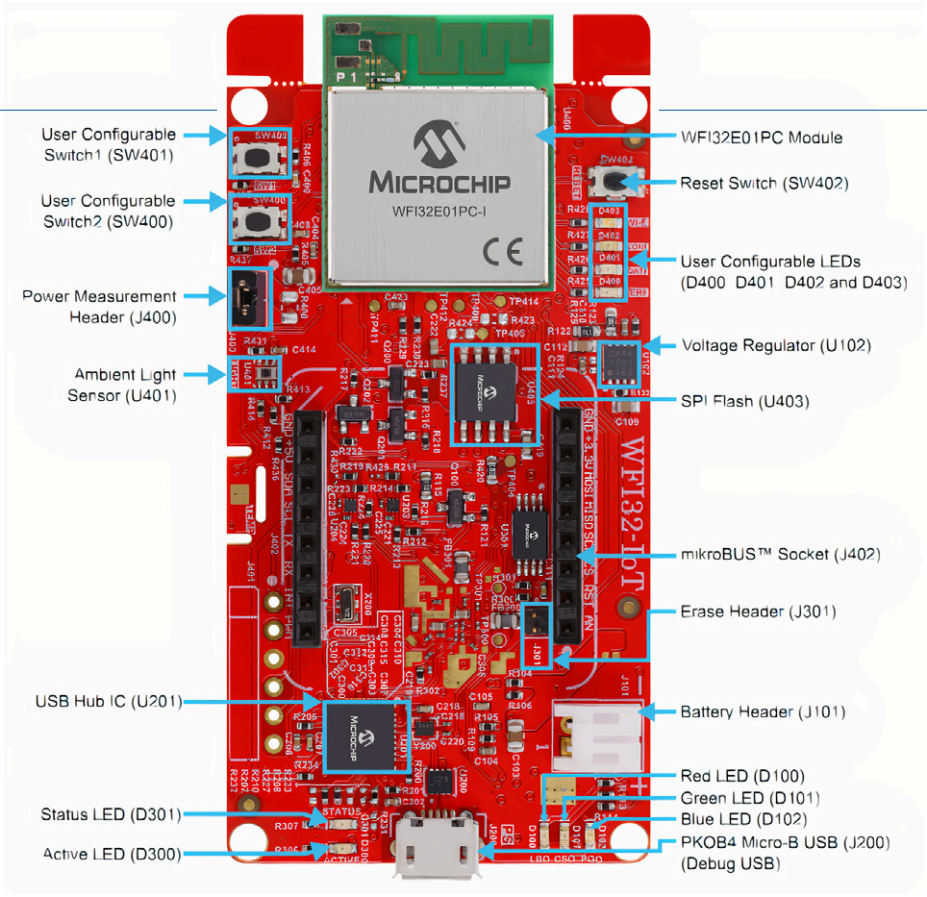


Bild 1. Das Microchip EV36W50A IoT-Development-Kit (Quelle: Microchip).

der Hersteller ein geeignetes Funkmodul oder ist eine Schnittstelle eines Drittanbieters (mikroBUS Click oder andere) vorhanden?
- Ist die Firmware des Boards in der Lage, Over-the-Air-Firmware-Updates durchzuführen?

Computing-Ressourcen

- Verfügt das Board über den Mikrocontroller, der verwendet werden soll? Haben Sie diesen schon einmal verwendet und verfügen Sie bereits über geeignete Entwicklungs-Toolchains?
- Sind die Computing-Ressourcen des Boards ausreichend, um die IoT-Applikation, die Host-Protokolle und alle Verbindungsprotokoll-Stacks auszuführen?
- Wenn der Mikrocontroller über einen integrierten Wireless-Transceiver verfügt, können Sie dessen Ruhemodus unabhängig steuern, um Strom zu sparen?
- Welche integrierten Sicherheitsfunktionen hat die MCU und sind diese für die geplante Applikation geeignet?

Benutzersteuerung

- Verfügt das Board über Bedientasten, berührungsempfindliche Schieberegler oder andere Hardwarefunktionen zur Benutzersteuerung?
- Ist ein Display vorhanden? Ist es für die Endanwendung erforderlich?
- Sind Benutzer-LEDs über den User-Code zugänglich? Sind genügend vorhanden, oder lassen sie sich schnell über einen freien GPIO-Anschluss hinzufügen?

Softwaresupport

- Welche ist die empfohlene Entwicklungs-Toolchain für dieses Board? Haben Sie diese bereits?
- Ist ein umfassendes Board-Support-Paket (BSP) enthalten?
- Welche zusätzlichen Treiber, Bibliothe-

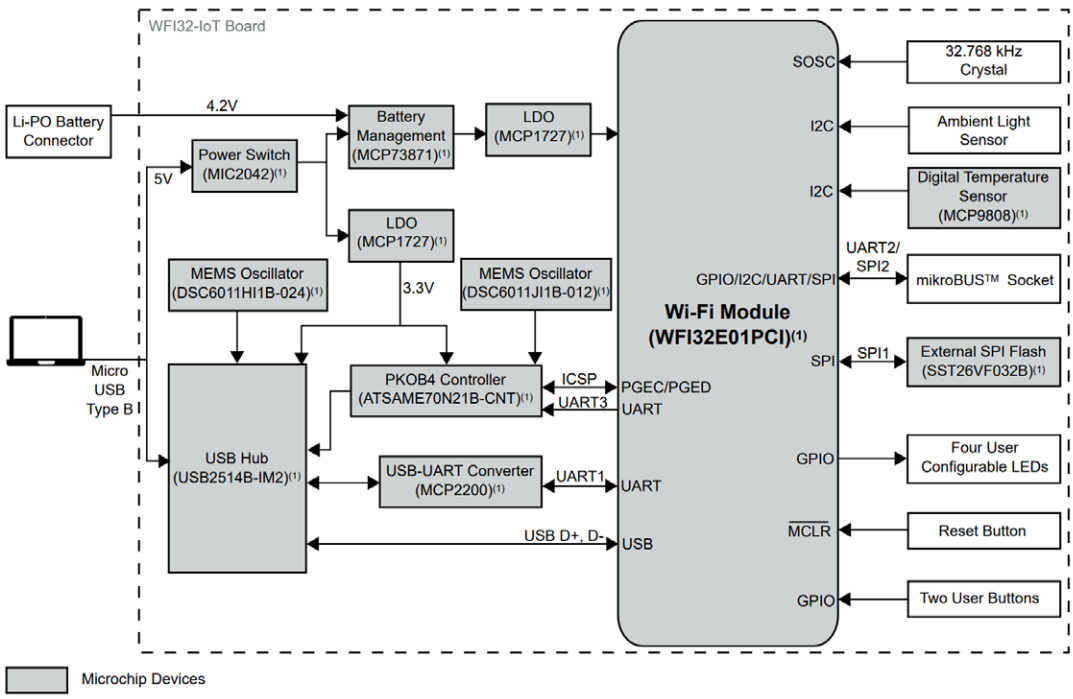


Bild 2. Funktionales Blockdiagramm des Micro WFI32-IoT EV36W50A Development Boards (Quelle: Microchip).

Bild 3. Das Asset Tracking Development Kit STEVAL-ASTRA1B von STMicroelectronics (Quelle: ST).

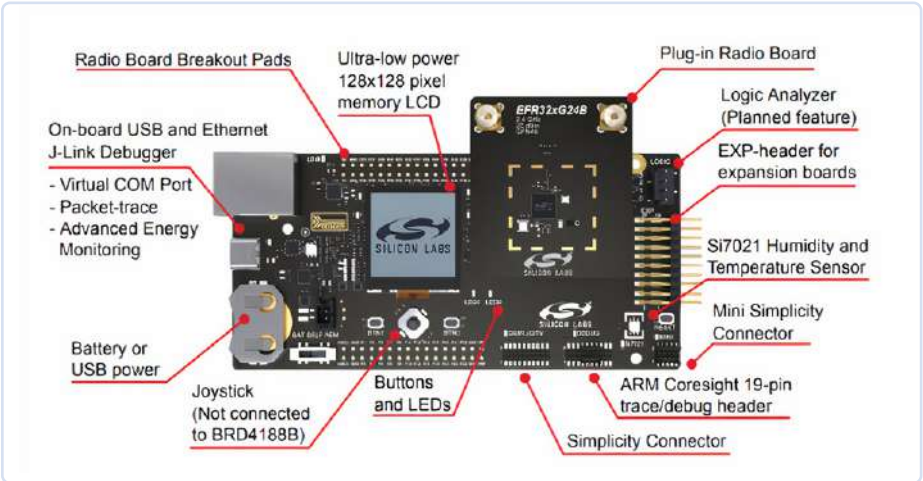


Bild 4. Das Antennen-Diversity-Modul xG24-RB4188A von Silicon Labs, aufgesetzt auf ein Wireless Kit Pro Mainboard von Silicon Labs (Quelle: Silicon Labs).

ken und Firmware sind erforderlich, und sind diese lizenzfrei?

- Prüfen Sie die Firmware- und Middleware-Lizenzierungsanforderungen beim Board-Hersteller.
- Wird das Board mit einer vorinstallierten Demo geliefert, mit der die Funktionen des Boards vorgestellt werden? Bietet es die Kommunikation mit gängigen Dienstanbietern wie Microsoft Azure oder Amazon AWS?
- Sind weitere Demo- und Code-Beispiele für das Board verfügbar? Gibt es ein Ökosystem von Bibliotheks- und Entwicklungspartnern?

IoT-Development-Boards im Schaufenster

Microchip WFI32-IoT development board

Das Microchip WFI32 (Artikelnummer EV36W50A [1]) ist ein umfassendes, voll integriertes, eigenständiges IoT-Development-Board **(Bild 1)**. Das WFI32-IoT beinhaltet ein Microchip WFI32E01PC Wi-Fi 802.11-Funkmodul, das auf der PIC-Mikrocontroller-Familie basiert. Zu den integrierten Sensoren zählen ein digitaler I²C-Temperatur-IC von Microchip und ein digitaler Umgebungslicht-IC. Entwickler können zusätzliche Sensoren oder Peripheriegeräte über einen MikroBUS-Anschluss anschließen. Das MCU-Funkmodul ist außerdem mit einer integrierten Antenne ausgestattet. Das Board kann über einen Workstation-Host oder einen LiPo-Akku mit Strom versorgt werden. Über einen integrierten PMIC kann der Akku per USB-Host geladen werden.

Bild 2 zeigt das funktionale Blockdiagramm des WFI32-IoT-Boards und hebt die auf dem Board integrierten Microchip-Komponenten hervor. Auf dem Board ist ein sofort einsatzbereites Demo-Image vorinstalliert, das die integrierten Sensoren ausliest und die Daten an Amazons AWS-Cloud sendet. Der Demo-Code und die vollständigen Anweisungen sind in einem GitHub-Repository verfügbar [2].

STMicroelectronics STEVAL ASTRA1B Referenzdesign zum Asset-Tracking mit mehreren Konnektivitäten

Bild 3 zeigt das STEVAL ASTRA1B [3] Development Kit und Referenzdesign. Es wurde speziell für das Prototyping und die Evaluierung von Asset-Tracking-Applikationen entwickelt und umfasst zwei drahtlose Konnektivitätsmodule. Dabei handelt es sich um ein drahtloses BLE/ZigBee-Mikrocontrollermodul STM32WB5MMG [4] mit geringer Leistung und kurzer Reichweite bei 2,4 GHz und ein drahtloses MCU-Modul STM32WL55JC mit großer Reichweite im Sub-GHz-Bereich für LPWAN-Kommunikation wie LoRa.

Das STEVAL ASTRA1B enthält zahlreiche Sensoren, die mehrere Umgebungs- und Bewegungsparameter messen können. Ein GNSS-Modul liefert Positionsdaten für den Outdoor-Einsatz. Außerdem verfügt das Board über ein STSAFE-Sicherheitselement, einen 480-mAh-Akku und eine OOB-Demo mit einem Asset-Tracking-Dashboard und einer Smartphone-App.

Bild 5. Ein Beispiel für die LR1120-Development-Kits von SEMTECH (Quelle: SEMTECH).

Silicon Labs xG24-RB4188A

Das Silicon Labs xG24-RB4188A [5] ist ein steckbares Antennen-Diversity-Modul für das Prototyping drahtloser 2,4-GHz-Applikationen **(Bild 4)**. Es wird an das Silicon Labs BRD4001 Wireless-Starterboard angeschlossen. Das Modul enthält ein Silicon Labs EFR32 Wireless Gecko System-on-Chip, einen HF-Schalter, ein Matching-Netzwerk und zwei SMA-Antennenanschlüsse. Das HF-Ausgangssignal des EFR32 liegt bei +20 dBm.

SEMTECH LR1120 development kits

Für das Prototyping von LoRa-LPWAN-Applikationen, die auf dem drahtlosen Mikrocontroller LR1120 von SEMTECH [6] basieren, bietet SEMTECH eine Reihe von LR1120-Development-Kits [7] an, wie in **Bild 5** dargestellt. Die Kits sind in regionalen Varianten entsprechend dem industriellen, wissenschaftlichen und medizinischen (ISM) Sub-GHz-Spektrum erhältlich. Das LR1120 eignet sich für multiregionale Applikationen in den Bereichen Asset Location, Bestandsmanagement und Diebstahlschutz.

In diesem Beitrag haben wir bereits auf die Möglichkeit hingewiesen, zusätzliche Sensoren oder Peripheriegeräte auf einem Development-Board zu integrieren. Wie bereits in der Beschreibung des Microchip-Boards erläutert, ist es mit einem mikroBUS-Anschluss ausgestattet. Der von Mikroe entwickelte mikroBUS hat sich schnell zu einem Industriestandard entwickelt, den viele Halbleiterhersteller für ihre Development- und Evaluation-Boards übernehmen. Der mikroBUS bringt serielle Konnektivität von SPI, UART und I²C zusammen mit Power-, Analog- und PWM-Signalen in ein kompaktes Socket-Format. Das Unternehmen Mikroe hat bereits Hunderte von Click-Boards [8] entwickelt, die diesen praktischen Formfaktor nutzen.

Ein Beispiel ist das Mikroe Ultra-Low Press Click [9]. Es wurde speziell für Niederdruckmessungen in der Pneumatik entwickelt und enthält den Drucksensor SM8436 von TE Connectivity, der über die I²C-Schnittstelle kommuniziert **(Bild 6)**.

Weiterentwicklung mit einem IoT-Development-Kit

Das Prototyping einer IoT-Applikation wird durch die Verfügbarkeit von Development-Boards erheblich vereinfacht. In diesem Beitrag haben wir einige Aspekte vorgestellt, die Embedded-Entwickler bei der Auswahl eines geeigneten Development-Boards beachten sollten. Neben den genannten Aspekten gibt es jedoch noch weitere Themen, die im Zusammenhang mit der jeweiligen Applikation zu berücksichtigen sind.

Und was möchten Sie als nächstes entwickeln?

230338-02

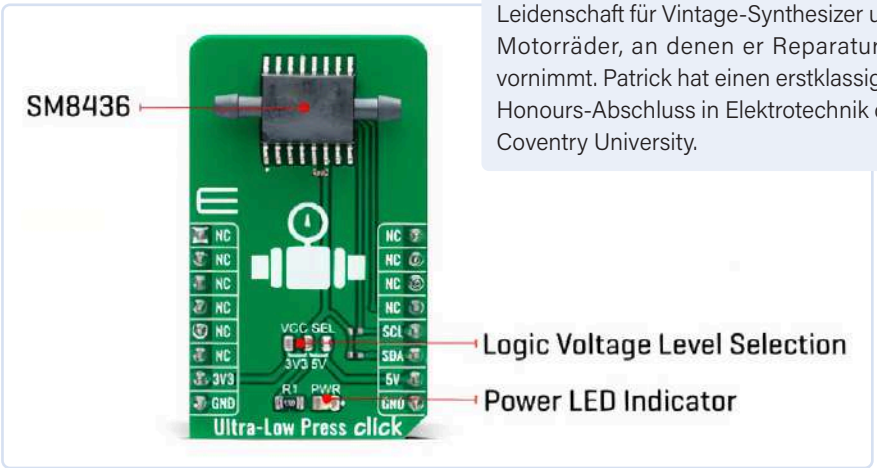


Bild 6. Das Board Ultra-Low Press Click von Mikroe (Quelle: Mikroe).

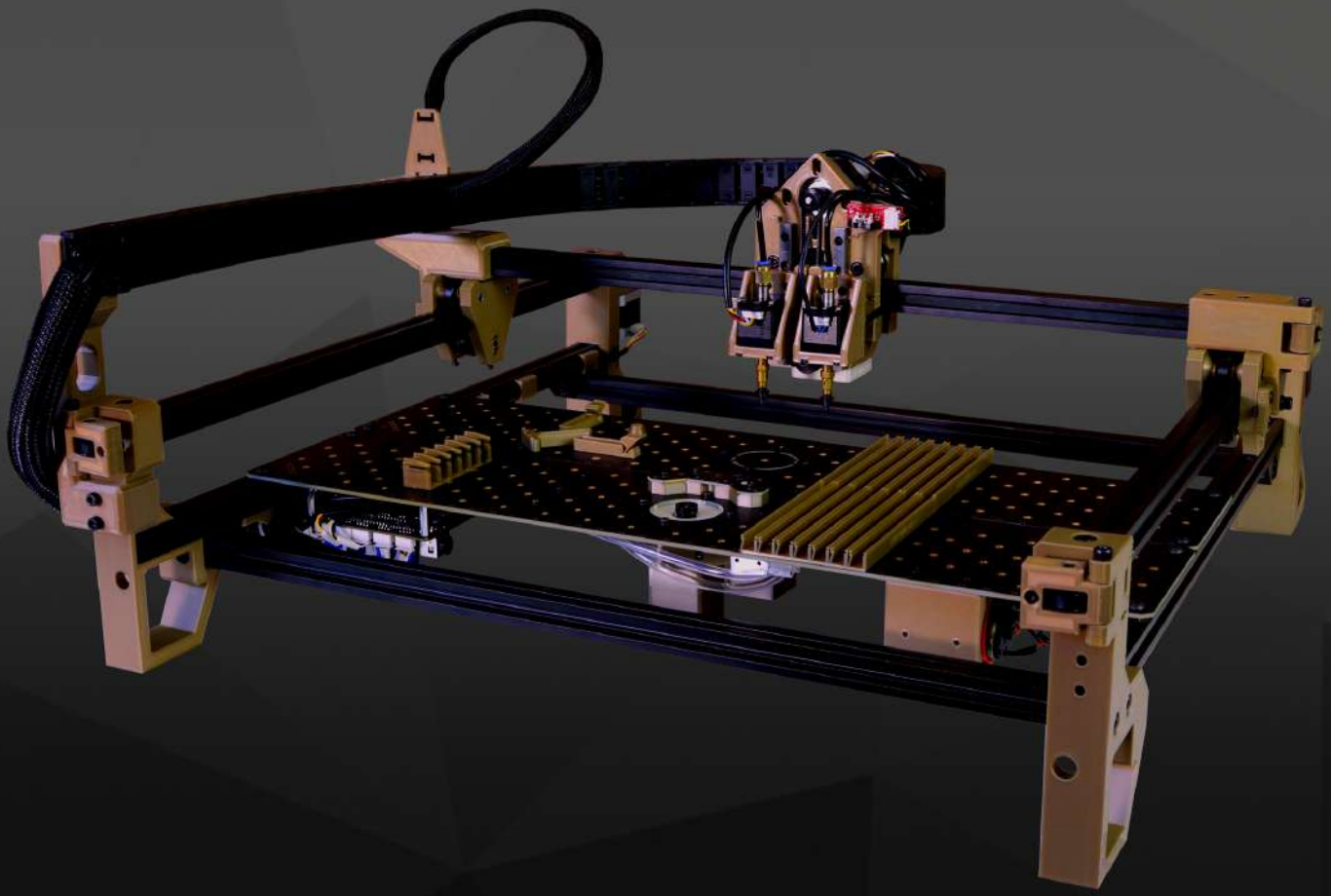
Über den Autor

Mark Patrick ist als technischer Marketingmanager von Mouser Electronics für EMEA für die Erstellung und Verbreitung technischer Inhalte in der Region verantwortlich – Inhalte, die für die Strategie von Mouser, sein technisches Publikum zu unterstützen, zu informieren und zu inspirieren, von entscheidender Bedeutung sind. Bevor er das technische Marketingteam leitete, war Patrick Teil des EMEA-Supplier-Marketing-Teams und spielte eine wichtige Rolle beim Aufbau und der Entwicklung von Beziehungen zu wichtigen Produktionspartnern. Zusätzlich zu verschiedenen technischen und Marketingpositionen war Patrick zuvor acht Jahre lang bei Texas Instruments im Anwendungssupport und im technischen Vertrieb tätig. Im Herzen ist er „praktizierender" Ingenieur mit einer Leidenschaft für Vintage-Synthesizer und Motorräder, an denen er Reparaturen vornimmt. Patrick hat einen erstklassigen Honours-Abschluss in Elektrotechnik der Coventry University.

WEBLINKS

[1] WFI32-IoT Board EV36W50A von Microchip Technology: https://bit.ly/3Vw4L5U
[2] WFI32-IoT auf GitHub: https://github.com/MicrochipTech/WFI32-IoT
[3] Asset Tracking Evaluation Board STEVAL-ASTRA1B von STMicroelectronics: https://bit.ly/3LwN4P6
[4] 2,4GHz-Wireless-Modul STM32WB5MMG von STMicroelectronics: https://bit.ly/3NFird0
[5] xG24-RB4188A von Silicon Labs: https://bit.ly/3LBRFQ5
[6] Wi-Fi/GNSS-Scanner und LoRa-Transceiver LR1120 von Semtech: https://bit.ly/428oET8
[7] Entwicklungskits LR1120: https://bit.ly/42rdGaO
[8] Click Boards™ von Mikroe: https://bit.ly/3LAaH9j
[9] Ultra-Low Press Click von Mikroe: https://bit.ly/3LXr2GH

Gegen



Das ist kein Wettbewerb.

Mit der LumenPnP Desktop Pick & Place Machine können Sie Ihre Leiterplatten selbst bestücken ohne eine zustäzliche Pinzette zu benötigen

-Radikal Open Source
-Angetriebene Feeder
-Doppeldüsen
-Bestückt 0402
-Erschwinglich

Opulo™

Opulo.io

Kondensatoren verhalten sich nicht immer kapazitiv

Von Dr. Rene Kalbitz (Würth Elektronik)

Kondensatoren verhalten sich kapazitiv, so zumindest die Theorie, wenn es um ideale Bauelemente geht. Das gilt allerdings nur unter bestimmten Betriebsbedingungen und hängt zudem vom Frequenzbereich ab. Dieser Artikel beleuchtet die Impedanzspektren unterschiedlicher Kondensatortechnologien und zeigt, wann kapazitives Verhalten zu erwarten ist und wann nicht.

Impedanz- und Kapazitätsspektren (oder S-Parameter) sind übliche Darstellungen der frequenzabhängigen elektrischen Eigenschaften von Kondensatoren. Die Interpretation solcher Spektren liefert eine Vielzahl von elektrochemischen, physikalischen und technisch relevanten Informationen. Diese müssen von stets vorhandenen Messartefakten sowie von parasitären Effekten getrennt werden. Da es manchmal nicht möglich ist, alle Daten im Datenblatt abzubilden, müssen Ingenieure unter Umständen auf gemessene Spektren zurückgreifen, um das geeignete Bauteil für den eigenen Schaltungsentwurf auszuwählen. Um eine bestmögliche Datengrundlage zu schaffen, hat *Würth Elektronik eiSos* das Online-Tool REDEXPERT [1] implementiert, in dem Spektren, aber auch andere Messungen zur Verfügung gestellt werden. Dieser Artikel beschreibt die Charakteristiken solcher Spektren und geht darauf ein, wie sich grundlegende elektrische Eigenschaften daraus ableiten lassen.

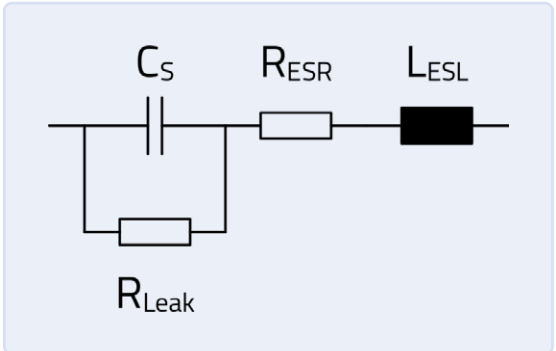


Bild 1. Standard-Ersatzschaltung für Kondensatoren, mit der Kapazität C_S, dem äquivalenten Serienwiderstand R_{ESR}, der äquivalenten Serieninduktivität L_{ESL} und dem Leck-Widerstand R_{Leak}.

Ersatzschaltung von Kondensatoren

Mit der in **Bild 1** gezeigten Schaltung ist es möglich, frequenzabhängige Impedanzspektren aller Kondensatortypen zu modellieren, vom Multilayer-Keramik-Chipkondensator (Multi Layer Ceramic Capacitors, MLCC) bis zum Superkondensator (Supercapacitor, SC).
Das Formelzeichen C_S steht für die reine Kapazität, die für sich genommen bei keinem elektrischen Bauteil existiert. Jeder reale Kondensator hat Verluste, die den Ladeprozess „verlangsamen". Dieses Phänomen wird durch den rein ohmschen äquivalenten Serienwiderstand R_{ESR} (Equivalent Series Resistance, ESR) beschrieben. Der Widerstand des Verbrauchers und der Leitungen tragen ebenfalls zum ESR bei.
Die reine verlustfreie Kapazität ist durch das Differential in **Gleichung 1** definiert:

$$C_S = \frac{dQ}{dV}$$

Dabei ist dQ die Änderung der Anzahl der Ladungen an der Kondensatoroberfläche und dV die Änderung der Spannung am Kondensator.
Jeder Wechselstrom in einem Metallleiter induziert ein magnetisches Feld, das dem Strom entgegenwirkt. In dem betrachteten Modell (auch L-C-R-Modell oder Standardmodell) wird diese Eigenschaft durch die äquivalente Serieninduktivität L_{ESL} (Equivalent Series Inductance, ESL) beschrieben.

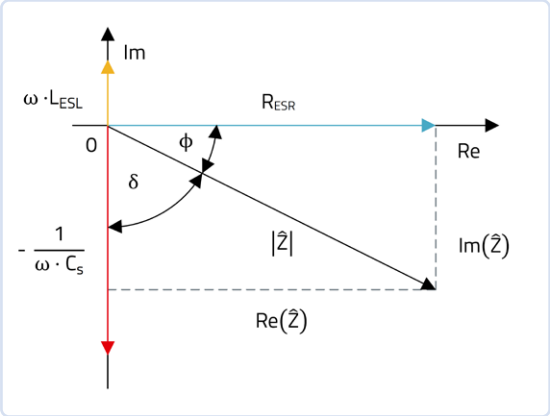


Bild 2. Vektorielle Darstellung der Impedanz in der komplexen Ebene. R_{Leak} wird der Einfachheit halber hier vernachlässigt.

C_S, R_{ESR} sowie L_{ESL} sind die wichtigsten Parameter, die zur Beschreibung der Mehrzahl aller Spektren erforderlich sind. Im einfachsten Ansatz sind sie Konstanten und ändern sich nicht mit der Frequenz, was für die Elektrotechnik ausreichend genau ist.
Der Langzeit-Ladungsverlust, also der Leckstrom, wird in guter Näherung durch den rein ohmschen Widerstand R_{Leak} beschrieben. Normalerweise ist R_{Leak} um Größenordnungen größer als R_{ESR} und kann oft vernachlässigt werden, das heißt $R_{Leak} \rightarrow \infty$. Seine Wirkung ist in den Spektren nur bei sehr niedrigen Frequenzen weit unterhalb von 1 Hz sichtbar [2].

Impedanz- und Kapazitätsspektren

Der folgende Abschnitt definiert häufig verwendete Begriffe und Messgrößen wie Kapazität und Impedanz. Die obige Schaltung lässt sich als frequenzabhängige komplexe Impedanz $\hat{Z}$, Kapazität $\hat{C}$, Streuparameter (S-Parameter) $\hat{S}$, Dielektrizitätskonstante $\hat{\varepsilon}$ oder jede andere messbare komplexe elektrische Größe beschreiben. Die Impedanz $\hat{Z} = Re(\hat{Z}) + i \cdot Im(\hat{Z})$ ist eine komplexe Größe, mit $Re(\hat{Z})$ und $Im(\hat{Z})$ als Real- beziehungsweise Imaginärteil. Die Impedanz wird häufig durch ihren Betrag $|\hat{Z}|$ und den Phasenwinkel ϕ ausgedrückt **(Gleichung 2)**.

$$\hat{Z} = |\hat{Z}| \cdot e^{i\phi}$$

In einer komplexen Ebene, wie in **Bild 2** dargestellt, beschreibt ϕ den Winkel zwischen $Re(\hat{Z})$ (Abszisse) und dem komplexen Vektor $\hat{Z}$. Physikalisch gesehen stellt $|\hat{Z}|$ das Verhältnis der Spannungsamplitude zur Stromstärkeamplitude dar, während ϕ die Phasendifferenz zwischen Spannung und Stromstärke bei einer bestimmten Frequenz angibt. Der Phasenwinkel ϕ ist mit dem Verlustwinkel verbunden gemäß **Gleichung 3.**

$$\arctan\left(\frac{Re(\hat{Z})}{|Im(\hat{Z})|}\right) = \delta = \frac{\pi}{2} - \phi$$

In der Elektrotechnik ist es ebenfalls üblich, die Größe $|\hat{Z}|$ und ihren äquivalenten Serienwiderstand $R_{ESR} = Re(\hat{Z})$ zu verwenden. Bei dem Beispiel in **Bild 1** ist der äquivalente Serienwiderstand der Realteil der Impedanz. Um die Beziehung zwischen dem Modell und der komplexen Größe $|\hat{Z}|$ grafisch darzustellen, sind alle Modellparameter (mit Ausnahme von R_{Leak}) auch in **Bild 2** angegeben. Die mathematische Beschreibung findet sich im Anhang von [2].
Die Impedanz lässt sich mit **Gleichung 4** auch in eine komplexe Kapazität umschreiben.

$$\hat{C} = \frac{1}{i \cdot 2 \cdot \pi \cdot f \cdot \hat{Z}} = Re(\hat{C}) + i \cdot Im(\hat{C})$$

All diese Größen wie $Re(\hat{Z})$, $Im(\hat{Z})$, $|\hat{Z}|$ oder δ können mit Impedanz- oder Netzwerkanalysatoren gemessen werden. Jedes elektronische Bauteil (nicht nur Kondensatoren) lässt sich durch einen Satz frequenzabhängiger Größen wie $Re(\hat{Z})$ und $Im(\hat{Z})$ oder $Re(\hat{C})$ und $Im(\hat{C})$ charakterisieren. Aber erst durch Ersatzschaltungen wie in Bild 1 werden Messergebnisse interpretierbar. Durch Veränderung von C_S, R_{ESR}, L_{ESL}, R_{Leak} ist es

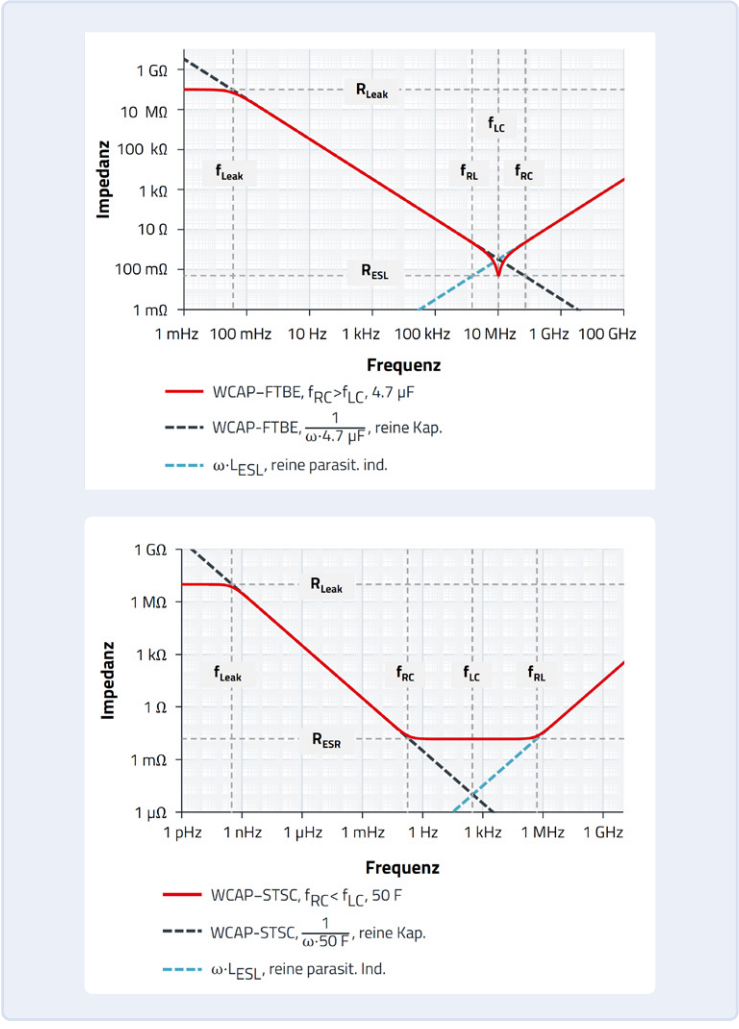
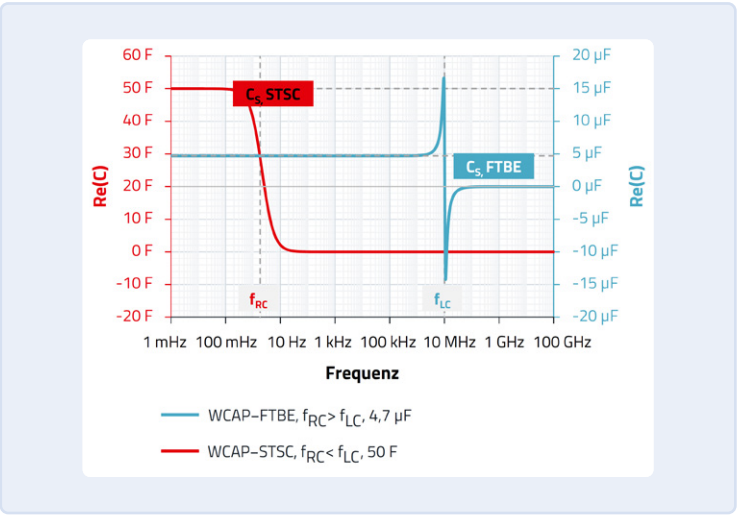


Bild 3. Impedanzspektren $|\hat{Z}|$ für WCAP- FTBE und WCAP-STSC, berechnet nach dem Standardmodell.



▲ *Bild 4. Kapazitätsspektren Re (Ĉ), berechnet anhand des Standardmodells. Das Diagramm für WCAP-STSC (rot) entspricht der linken Ordinate und das Diagramm für WCAP-FTBE (blau) entspricht der rechten Ordinate.*

möglich, das grundlegende Frequenzverhalten für alle Kondensatoren zu berechnen. Dies wird beispielhaft für die Impedanz- und Kapazitätsspektren eines 4,7-µF- und eines 50-F-Kondensators in **Bild 3** beziehungsweise **Bild 4** gezeigt. Die zugehörigen Phasen- und Verlustwinkel sind im Anhang von [2] dargestellt. Die Parameter für die beiden Beispiele lauten wie folgt:

- Superkondensator (WCAP-STSC) mit C_S = 50 F, R_{ESR} = 15 mΩ, L_{ESL} = 5 nH und R_{Leak} = 10 MΩ,
- Folienkondensator (WCAP-FTBE) mit C_S = 4,7 µF, R_{ESR} = 5 mΩ, L_{ESL} = 5 nH und R_{Leak} = 10 MΩ.

Die Parameter wurden so gewählt, dass sie zu den vorhandenen WE-eiSos-Produkten passen, die unter den Produktbezeichnungen für Folienkondensatoren WCAP-FTBE (4,7 µF) und SCs WCAP-STSC (50 F) zu finden sind. In diesen Diagrammen wurden C_S, R_{ESR}, L_{ESL} und R_{Leak} als Konstanten und unabhängig von der Frequenz angenommen **(Tabelle 1)**.

Elektrische Parameter	WCAP-FTBE	WCAP-STSC
C_S	4,7 µF	50 F
R_{ESR}	5 mΩ	15 mΩ
L_{ESL}	5 nH	5 nH
R_{Leak}	10 MΩ	10 MΩ

Tabelle 1. Für die Berechnung der Spektren verwendete elektrische Parameter.

Im Allgemeinen wird die Position der auffälligsten Stellen in den Spektren durch vier charakteristische Frequenzen beschrieben:

Charakteristische Frequenz des R_{ESR}-C-Glieds **(Gleichung 5)**:

$$f_{RC} = \frac{1}{2 \cdot \pi \cdot R_{ESR} \cdot C_S}$$

Charakteristische Frequenz des L-C-Glieds **(Gleichung 6)**:

$$f_{LC} = \frac{1}{2 \cdot \pi \cdot \sqrt{L_{ESL} \cdot C_S}}$$

Charakteristische Frequenz des R_{Leak}-C-Glieds **(Gleichung 7)**:

$$f_{Leak} = \frac{1}{2 \cdot \pi \cdot R_{Leak} \cdot C_S}$$

Charakteristische Frequenz des R_{ESR}-L-Glieds **(Gleichung 8)**:

$$f_{RL} = \frac{R_{ESR}}{2 \cdot \pi \cdot L_{ESL}}$$

In Bild 3 und Bild 4 sind zwei wesentliche Situationen zu erkennen:

- Lorentz-Oszillation: $f_{RC} > f_{LC}$ wie im Fall von C_S = 4,7 µF (blaue Kurve)
- Debye-Relaxation: $f_{RC} < f_{LC}$ wie im Fall von C_S = 50 F (rote Kurve).

Die schwarzen und blauen gestrichelten Linien in beiden Diagrammen von Bild 3 kennzeichnen den rein kapazitiven und induktiven Teil. Die charakteristische Frequenz f_{RC} des R-C-Glieds ist die Frequenz, mit der der Kondensator geladen und entladen werden kann. Der Kehrwert der Frequenz ist im Grunde die Ladezeit bei idealer konstanter Ladespannung. Oberhalb dieser Frequenz wird der Kondensator nicht mehr vollständig geladen (bezogen auf die maximale Spannung des Signals).
Bei f_{RC} zeigt das Kapazitätsspektrum (Bild 4) des Superkondensators eine Schulter. Unterhalb dieser Frequenz kann der Kapazitätswert aus dem Diagramm abgeleitet werden. Oberhalb von f_{RC} zeigt das Impedanzspektrum, das in Bild 3 (unten) dargestellt ist, ein Plateau bei R_{ESR}. Die charakteristische Frequenz des L-C-Glieds f_{LC} ist die Frequenz, bei der die Kopplung von parasitärer Induktivität und Kapazität zu einem Resonanzverhalten führt, wenn $f_{RC} > f_{LC}$, wie in Bild 3 (oben) gezeigt. Unterhalb dieser Frequenz verhält sich der Kondensator kapazitiv, das heißt, er kann elektrische Ladung speichern, oberhalb dieser Frequenz verhält sich der Kondensa-

tor induktiv. Die Eigenresonanz führt zu einem scharfen Minimum im Impedanzspektrum (WCAP-FTBE), wie in Bild 3 (oben) dargestellt. Am Minimum des Impedanzspektrums lässt sich der R_{ESR}-Wert ablesen. Kondensatoren sollten in Anwendungen nicht bei f_{LC} oder oberhalb dieser Frequenz betrieben werden.
Das Kapazitätsspektrum des 4,7-µF-Kondensators FTBE in Bild 4 zeigt einen Pol. Es handelt sich dabei um ein echtes physikalisches Verhalten und nicht nur um ein Mess-Artefakt. Das Messsystem, das aus dem Kondensator und der parasitären Induktivität besteht, verhält sich wie ein Oszillator beziehungsweise ein Schwingkreis. Die physikalischen Abläufe sind in [2] detailliert dargestellt. Die charakteristische Frequenz des R_{Leak}-C-Glieds ist f_{Leak}. Unterhalb dieser Frequenz verhält sich der Kondensator wie ein Widerstand mit dem Wert R_{Leak}. Normalerweise ist dieser Effekt in den Spektren kaum sichtbar. Er erfordert entweder Messungen bei Frequenzen unter 1 Hz oder einen recht niedrigen Wert von R_{Leak}.
Die charakteristische Frequenz f_{RL} des R_{ESR}-L-Glieds ist die Frequenz, oberhalb derer sich der Kondensator wie eine Induktivität mit dem Wert L_{ESL} verhält (Bild 3, unten). In Fällen, in denen $f_{RC} < f_{LC}$ gilt, kennzeichnet dies den Beginn des Anstiegs der Impedanz bei hohen Frequenzen.

Die beiden hier diskutierten Beispiele verdeutlichen, dass mit einem verhältnismäßig einfachen Modell das Verhalten von hoch- und niedrig-kapazitiven Kondensatoren beschrieben werden kann. Die berechneten Spektren zeigen grundsätzlich alle Charakteristika, die auch bei gemessenen zu finden sind. Zweifellos enthalten Letztere noch weitere Informationen, die eine Erweiterung des Modells erfordern; dennoch ermöglicht das L-C-R-Modell, die für den ingenieurstechnischen Bereich wichtigen Parameter zu bestimmen sowie die grundlegende Interpretation der Spektren. Die hier vorgestellten charakteristischen Frequenzen sind hierbei ein wichtiges analytisches Hilfsmittel, die quasi als Landmarken für die Interpretation von gemessenen Spektren verstanden werden können. Eine weiterführende Betrachtung von gemessenen Spektren verschiedener Kondensatortypen bietet die Application Note ANP109 [2].

Die Betrachtung umfasst folgende Kondensator-Typen:
- Superkondensatoren WCAP-STSC
- Aluminium-Elektrolytkondensatoren WCAP-AIGB
- Folienkondensatoren WCAP-FTBE
- Multilayer-Keramik-Chip-Kondensatoren WCAP-CSGP

230318-02

Über den Autor

Dr. René Kalbitz hat an der Universität Potsdam und an der Universität von Southampton (UK) Physik studiert. Nach absolviertem Diplom-Studium forschte und promovierte er über organische Halbleiter und Isolatoren an der Universität Potsdam. Weitere Erfahrungen im Bereich der angewandten Forschung konnte er am Fraunhofer Institut für angewandte Polymerforschung sammeln. Er ist seit 2018 bei Würth Elektronik als Produktmanager für Superkondensatoren tätig und betreut Forschungs- und Entwicklungsprojekte im Bereich Kondensatoren.

WEBLINKS

[1] REDEXPERT Online-Simulator: https://redexpert.we-online.com/redexpert/
[2] Kalbitz, René: Impedanzspektren unterschiedlicher Kondensatortechnologien. AppNote ANP109: http://www.we-online.com/ANP109

Eine NTP-Uhr mit CircuitPython

Warum Sie diese Programmiersprache verwenden sollten

Von Michael Bottin (Frankreich)

Wir werfen einen Blick auf eine der nützlichen Alternativen zu Python, die so leichtgewichtig ist, dass sie auf kleinen Mikrocontrollern funktioniert: CircuitPython. Hier realisieren wir ein Projekt rund um einen Raspberry Pi Pico W, der die Zeit von einem NTP-Zeitzeichenserver abruft und auf einem LCD anzeigt.

Python ist eine der beliebtesten Programmiersprachen. Da es sich um eine interpretierte Sprache handelt, ist die Entwicklungsphase im Gegensatz zu Sprachen wie C/C++, bei denen das Programm vor der Ausführung kompiliert werden muss, viel schneller durchlaufen. Andererseits bedeutet das Fehlen einer Kompilierung, dass zuvor eine Ausführungsumgebung auf dem Zielsystem installiert werden muss.

Seit über 30 Jahren macht Python im Bereich der Computer und des Cloud Computing Fortschritte. Es deckt verschiedene Bereiche wie wissenschaftliche Berechnungen, Webentwicklung (Back-End), System-Scripting, maschinelles Lernen und Big Data (Datenanalyse und -visualisierung) ab. Python wird von vielen Unternehmen verwendet, von Google („Python, wo wir können, C++, wo wir müssen"), YouTube, Instagram, Spotify, Intel, Facebook, Dropbox, Netflix, Pixar und Reddit und so fort.

Es ist daher nicht überraschend, dass Python seinen Weg auch in die Welt der eingebetteten Elektronik gefunden hat. Damit dies möglich ist, mussten jedoch abgespeckte Versionen entwickelt werden, die auf einem Mikrocontroller laufen konnten, die auch hardwareorientiert sein mussten, um die verschiedene Peripherie, die in einem Mikrocontroller verfügbar ist, nutzen zu können (GPIO, PWM, I²C, SPI, UART...).

Es gibt hauptsächlich zwei Versionen eines abgespeckten Pythons, nämlich:

- MicroPython, das von Damien George im Jahr 2013 entwickelt wurde [1]
- CircuitPython, das von Limor Fried (Adafruit CEO), Scott Shawcroft und vielen anderen Mitwirkenden entwickelt wurde [2]

Diese beiden Sprachen teilen sich die gleiche Implementierung der Programmiersprache Python (CPython). CircuitPython ist eine Open-Source-Abspaltung von MicroPython, so dass jede Weiterentwicklung von MicroPython direkte Auswirkungen auf CircuitPython hat. Soweit mir bekannt ist, wurden in Elektor bereits mehrere Artikel über MicroPython veröffentlicht, aber keiner über CircuitPython.

CircuitPython vs. MicroPython

Seien wir ehrlich: Wenn Sie ein IT-Spezialist sind, der bereits mit Python vertraut ist und diese Sprache in der Welt der eingebetteten Elektronik einsetzen möchte, dann ist CircuitPython für Sie uninteressant. In einem solchen Fall wäre MicroPython tatsächlich weitaus besser geeignet. Wenn Sie jedoch ein Maker, ein Student oder ein Lehrer sind, der kaum Kenntnisse in Python hat, könnte CircuitPython eine interessante und die geeignetere Wahl sein. CircuitPython bietet nämlich nicht nur eine benutzerfreundliche Entwicklungskette, sondern auch eine zusätzliche Abstraktionsebene. Es ist auch erwähnenswert, dass die meisten Informationen über diese Sprache online über Bibliotheken, Tutorials und Foren gefunden werden können.

Deshalb habe ich mich entschieden, CircuitPython für diesen Artikel zu verwenden. Ich wollte den Lesern, die mit Python auf Mikrocontrollern beginnen wollen, einen einfachen Zugang bieten. Außerdem ist die CircuitPython-Community sehr aktiv. Es werden regelmäßig neue Versionen (die aktuelle ist die Version 8.x) und neue Bibliotheken veröffentlicht, und viele mikrocontroller-basierte Boards unterstützen diese Sprache. CircuitPython kann mit Hilfe der Blinka-API auch auf SBCs wie dem Raspberry Pi, BeagleBone, Odroid oder Jetson ausgeführt werden.

Ziel dieses Artikels

Dieser Artikel soll Ihnen die Möglichkeit eröffnen, CircuitPython zu kennenzulernen und sich anhand einer einfachen Anwendung mit dieser Sprache vertraut zu machen.

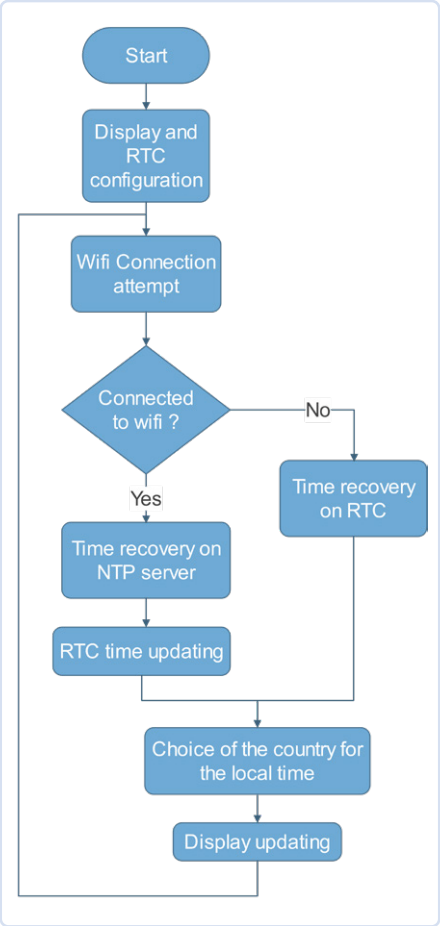


Bild 1. Prinzipielles Flussdiagramm für die NTP-Uhr.

Ich werde Ihnen zeigen, wie Sie mit CircuitPython sehr einfach eine internet-synchronisierte NTP-Uhr erstellen können. Mit diesem Projekt können Sie auch die Ortszeit für mehr als zwanzig Länder auf der ganzen Welt einstellen. Die Uhr benötigt eine WLAN-Verbindung (SSID und Passwort vorkonfiguriert), um die Zeit zu synchronisieren und einen dedizierten NTP-Server (Network Time Protocol), um den Zeitstempel abzurufen.

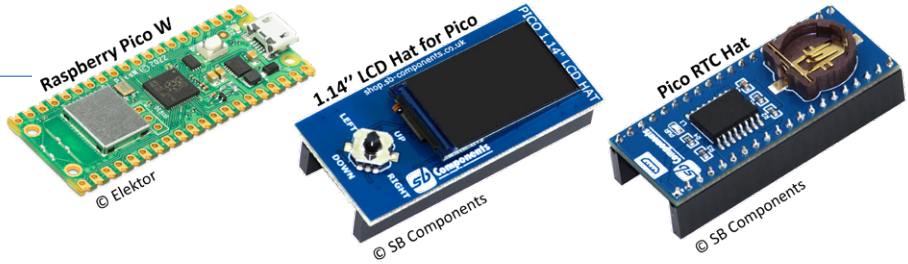


Bild 2. Die verwendeten Module.

Diese Uhr kann die Zeit dank einer eingebauten, synchron mit der NTP-Uhr laufenden Echtzeituhr (RTC) auch dann aktuell halten, wenn mal kein Internet verfügbar ist. Eine Backup-Batterie sorgt zudem dafür, dass die RTC auf dem neuesten Stand ist, auch wenn keine Stromversorgung vorhanden ist.
Das allgemeine Funktionsschema der Uhr ist in **Bild 1** dargestellt.

Präsentation der Hardware

Viele Entwicklungsplatinen (und damit auch Mikrocontroller) lassen sich leicht mit CircuitPython programmieren. Ich habe mich für den Raspberry Pico W entschieden, weil das Board bekannt und preiswert und bei den meisten Händlern, einschließlich Elektor [3], gut erhältlich ist, und weil es eine WLAN-Schnittstelle hat. Allerdings verfügt das Board weder über ein Display noch über eine genaue Echtzeituhr, was bedeutet, dass für unser Projekt zusätzliche Komponenten erforderlich sind.
Anstatt eine eigene Platine zu entwickeln, habe ich mich entschieden, kommerzielle Module für dieses Projekt zusammenzustellen. Drei gestapelte Module reichen für dieses Projekt aus (siehe **Bild 2**):

- ein Raspberry Pico W (achten Sie darauf, die W-Version zu kaufen, die die WLAN-Schnittstelle enthält)
- ein Display-Erweiterungsmodul mit einem LCD-Bildschirm mit einer Auflösung von 240 × 135 Pixel und einem Joystick-Controller für die Navigation (SB Components [4])
- ein Erweiterungsmodul mit einer DS3231-Echtzeituhr (SB Components [5])

Hinweis: Der RP2040-Mikrocontroller verfügt auch über eine interne Echtzeituhr, die anstelle der Pico-RTC-HAT verwendet werden könnte. Allerdings ist der RP2040 nicht so genau und verfügt auch nicht über eine Pufferbatterie.

Verdrahtung

Auch wenn wir kommerzielle Module verwenden, die die Hardware-Implementierung erleichtern, müssen wir die Verbindungen zwischen den Modulen kennen, um die Codezeilen für unsere Anwendung zu schreiben. **Bild 3** zeigt die Verbindungen zwischen den verschiedenen Modulen sowie die Bezeichnungen für diese Verbindungen, die auch im Programm verwendet werden. Beachten Sie, dass die in der GitHub-Dokumentation von SB Components angegebene Pinbelegung nicht korrekt ist, als dieser Artikel entstand, der Quellcode jedoch auf die in Bild 3 gezeigten korrekten Pinbelegungen verweist).

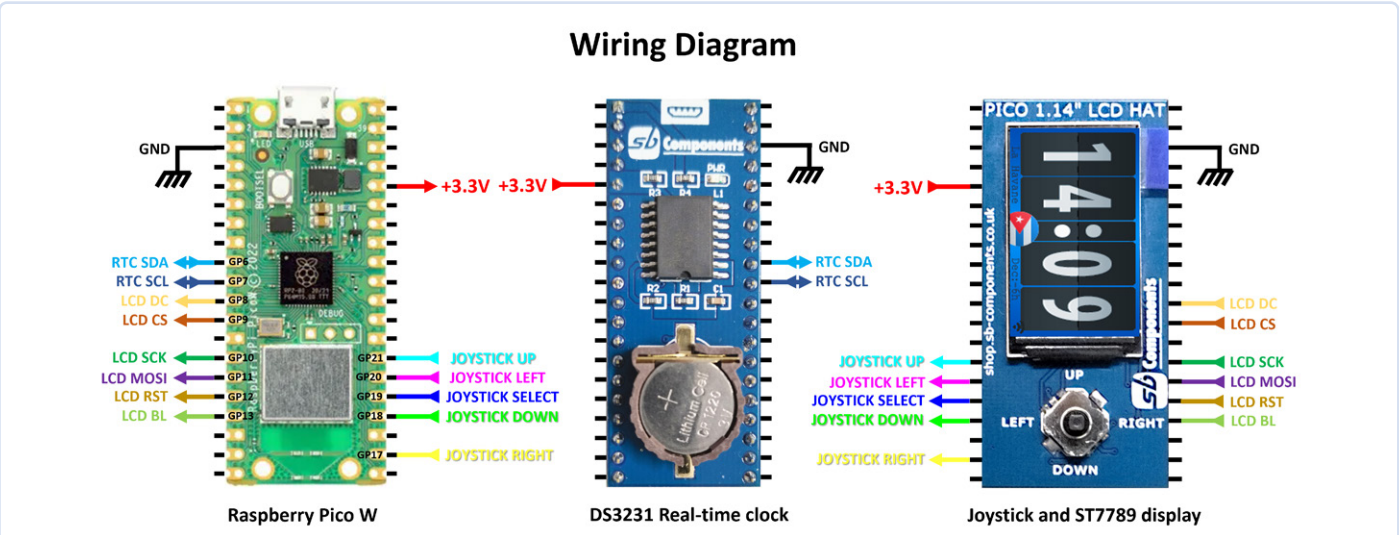


Bild 3. Verbindungen zwischen den Modulen.

Bild 4. CircuitPython-Webseite.

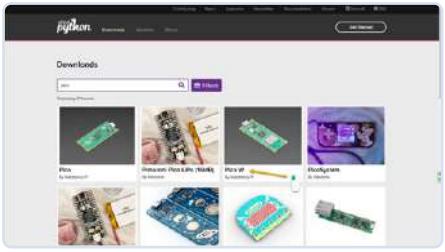


Bild 5. Auswahl des Pico W aus den CircuitPython-kompatiblen Boards.

Bild 6. Die .UF2-Datei wird heruntergeladen.

So installieren Sie CircuitPython

Der RP2040-Mikrocontroller des Raspberry PI Pico W verfügt beim Kauf des Moduls nicht über CircuitPython, so dass Sie es zunächst auf das Board „flashen" müssen. Genauso wie es verschiedene C-Compiler-Versionen je nach verwendetem Zielcontroller gibt, gibt es so viele CircuitPython-Versionen wie es unterstützte Boards gibt. Bis heute sind mehr als 380 CircuitPython-kompatible kommerzielle Boards zusammengekommen! Viele der neuesten Boards von Adafruit sind natürlich kompatibel, aber auch viele andere Boards, wie die von Pimoroni, Expressif, Seeedstudio, Waveshare, LilyGo, Cytron, DFRobot und Wiznet „passen".
Die meisten von ihnen enthalten Mikrocontroller von Microchip (SAMD21 / SAMD51), Nordic (nRF52840), Expressif (ESP32) oder von der Raspberry Pi Foundation (RP2040). Deshalb müssen Sie die für den Raspberry Pico W passende CircuitPython-Version herunterladen:

- Gehen Sie in Ihrem Browser auf die CircuitPython-Website [6] (**Bild 4**).
- Klicken Sie auf den Link *Downloads* und suchen Sie nach dem Pico-W-Board (**Bild 5**).
- Laden Sie die .UF2-Datei herunter, die auf der angezeigten Seite verfügbar ist (**Bild 6**).

Der RP2040-Mikrocontroller des Raspberry Pi Pico enthält bereits einen Bootloader, der die Installation von CircuitPython auf ihm erleichtert. Die folgende Anleitung zeigt Ihnen, wie Sie die von Ihnen heruntergeladene CircuitPython-Version installieren:

1. Trennen Sie das Board Raspberry Pico W von Ihrem Computer.
2. Halten Sie die BOOTSEL-Taste des Raspberry Pico W gedrückt, während Sie das Board über ein Micro-USB-Kabel (das Datenübertragung ermöglicht), wieder an Ihren Computer anschließen.
3. Im Dateiexplorer sollte ein neues Laufwerk mit dem Namen *RPI-RP2* angezeigt werden.
4. Laden Sie die heruntergeladene CircuitPython UF2-Datei auf das Laufwerk *RPI-RP2* hoch.

Das ist alles, was Sie tun müssen. Sobald CircuitPython erfolgreich auf das Raspberry Pico W geflasht wurde, sollte ein *CIRCUITPY*-Laufwerk erscheinen.

Wie man das CIRCUITPY-Laufwerk benutzt

Hier sind einige wichtige Punkte, die bei der Verwendung des CIRCUITPY-Laufwerks zu beachten sind:

- Dieses Laufwerk funktioniert wie ein USB-Laufwerk. Sie können Dateien/Ordner über einen Datei-Explorer hinzufügen oder löschen. Seine Kapazität entspricht der des Flash-Speichers des RP2040-Mikrocontrollers, um Ihren Code und Ihre Ressourcen (Bilder, Audio...) zu speichern.
- Sie können das Raspberry Pico W Board ohne besondere Vorkehrungen an Ihren Computer anschließen. **Es wird jedoch dringend empfohlen, das Board wie ein USB-Laufwerk auszuwerfen, wenn Sie es abziehen wollen, da sonst die Gefahr besteht, dass das Dateisystem beschädigt wird.**
- Sobald Sie die Datei *boot.txt* geöffnet haben, können Sie überprüfen, welche Version von CircuitPython auf dem Raspberry Pico W installiert ist.
- Sobald das Board Raspberry Pico W über den USB-Anschluss mit Strom versorgt wird (entweder von Ihrem Computer oder von einem Gleichstromnetzteil), läuft der CircuitPython-Code. Es werden jedoch nur Dateien mit dem Namen *code.py* oder *main.py* ausgeführt. Nennen Sie also Ihre Datei für dieses Projekt *code.py*.
- Es ist immer ratsam (aber nicht verpflichtend), die Bilddateien in ein *./images*-Verzeichnis zu kopieren, die Bibliotheksdateien in ein *./lib*-Verzeichnis, und so weiter. Dies wird Ihnen zu einer organisierten Baumstruktur für Ihre Projekte verhelfen.

Wie man die IDE benutzt

CircuitPython ist eine interpretierte Sprache, daher gibt es keine Kompilierung. Wenn eine *code.py*-Datei mit CircuitPython-Code auf dem *CIRCUITPY*-Laufwerk vorhanden ist, wird sie automatisch ausgeführt. Das bedeutet, dass wir diese Datei einfach in einem Texteditor bearbeiten und speichern könnten, bevor wir sie ausführen, aber denken Sie daran: Das Schreiben von Programmen geht oft Hand in Hand mit dem Debuggen. Wenn Sie eine richtige IDE verwenden, haben Sie Zugang zu bestimmten Werkzeugen wie einer Konsole, um Rückmeldungen über Fehler zu erhalten.
Die IDE, die wir für dieses Projekt verwenden werden, ist der Editor Mu [7]. Es handelt sich um eine kostenlose, einfache Softwarelösung, die für Windows, Mac OSX und Linux verfügbar ist. Sie können natürlich auch Thonny, VS Code, Atom oder PyCharm verwenden, wenn Sie die CircuitPython-Erweiterung installieren.
Bild 7 zeigt die Oberfläche der Mu Editor-IDE. Die Symbolleiste ist minimalistisch:

- **Mode**: Auswahl der Programmiersprache. Stellen Sie den Modus *CircuitPython* ein.
- **New**: Erstellen einer neuen, leeren Datei
- **Load**: Öffnet eine bestehende Datei. Wenn der *CircuitPython*-Modus ausgewählt ist und Ihr Raspberry Pico W korrekt angeschlossen ist, sollte der Ordner *CIRCUITPY* automatisch im Dialogfenster erscheinen.
- **Save**: Speichert die aktuelle Datei (das aktive Tab). Ein kleiner roter Punkt neben dem Dateinamen auf der Registerkarte zeigt, dass die Datei seit der letzten Sicherung geändert wurde.
- **Serial**: Blendet die Konsole ein/aus. Ich empfehle, die Konsole (REPL) immer einzublenden, da sie Fehler meldet und anzeigt, was Sie aus dem auf dem Pico laufenden Code ausgeben möchten.
- **Plotter**: Zeigt/verbirgt ein scrollendes Diagramm mit digitalen Daten, die Ihr Code erzeugen kann

Bild 7. IDE-Schnittstelle des Mu-Editors.

- **Zoom-in** und **Zoom-out**: Verkleinert oder vergrößert die Schrift
- **Theme**: Schaltet zwischen drei verschiedenen Anzeigethemen um: *Day* (helles Thema), *Night* (dunkles Thema) und *High-Contrast*.
- **Check**: Sucht nach Code-Fehlern, auch wenn sie keinen Einfluss auf den Betrieb haben
- **Tidy**: Bringt Ordnung in Ihren Code, indem zum Beispiel überflüssige Leerzeichen oder Leerzeilen gelöscht werden
- **Help**: Online-Hilfe
- **Quit**: Beendet den Editor

So, hier sind die verschiedenen Schritte, um ein Programm in der IDE zu entwickeln:

- Öffnen Sie die Datei *code.py* auf dem Laufwerk *CIRCUITPY*.
- Ändern Sie den Code im Editier-Bereich.
- Speichern Sie die Änderungen in Ihrer Datei.
- Beobachten Sie die Ergebnisse der Ausführung in der seriellen Konsole. Gehen Sie zurück zu Schritt 2, wenn bei der Ausführung ein Fehler aufgetreten ist.

Weitere Informationen finden Sie unter den folgenden Links:

- Hier starten! (über die Schnittstelle) [8]
- Was ist eine REPL? (die Konsole) [9]
- Plotten von Daten mit Mu [10]
- Tastaturkürzel [11]

Nun, da Sie alles haben, was Sie brauchen, um Ihr Projekt zu starten, lassen Sie uns zum Kern der Sache kommen: die NTP-Uhr.

WLAN-Verbindungstest

CircuitPython unterstützt natives WLAN auf dem Raspberry Pico W, so dass Sie keine zusätzlichen Bibliotheken installieren müssen. Um sich über WLAN zu verbinden, müssen Sie die SSID und das Passwort Ihres Netzwerks angeben. Um zu verhindern, dass diese Informationen direkt in den Code eingebettet werden, macht CircuitPython Gebrauch von Umgebungsvariablen.
Eine Datei namens *settings.toml* [12] befindet sich im Stammverzeichnis von *CIRCUITPY*, deren Daten „geheim“ bleiben. In unserem Fall enthält die TOML-Datei nur die SSID und das Passwort des Netzwerks, aber prinzipiell können dort alle vertraulichen Daten wie API-Schlüssel gespeichert werden (siehe **Bild 8**). Diese Datei lässt sich nicht in der IDE bearbeiten; Sie müssen einen Texteditor verwenden, um sie zu erstellen und zu vervollständigen.
Der Code für die Verbindung mit dem

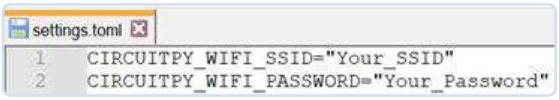


Bild 8. SSID und Passwort in settings.toml.

Listing 1: Code für die WLAN-Verbindung

```
# CircuitPython's own libraries
import os
import ipaddress
import wifi

print()
print("Connecting to Wi-Fi")

# connect to your Wi-Fi network with SSID/PASSWORD in 'settings.toml'
wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'), os.getenv('CIRCUITPY_WIFI_PASSWORD'))

print("Connected to Wi-Fi")

# pings Google DNS server
ipv4 = ipaddress.ip_address("8.8.8.8")
print("Ping google.com: %f ms" % (wifi.radio.ping(ipv4)*1000))
```

Bild 9. Herunterladen der Bibliotheken (das Datum im Dateinamen wird bei Ihnen natürlich anders sein).

Netzwerk ist in **Listing 1** dargestellt. Sie können auf den ersten Blick sehen, dass der Code sehr kompakt ist:

- **Zeile 2**: Import der *os*-Bibliothek, die es Ihnen ermöglicht, die Umgebungsvariablen der Datei *settings.toml* zu lesen
- **Zeile 4**: Import der *wifi*-Bibliothek, die die Netzwerkverbindung ermöglicht
- **Zeilen 6, 7, 12**: Diese Zeilen sind nicht notwendig, aber sie liefern dem Benutzer Informationen auf der seriellen Konsole
- **Zeile 10**: Diese eine Zeile reicht aus, um die physische Netzwerkverbindung herzustellen
- **Zeilen 3, 14…16**: Diese Zeilen sind nicht notwendig, aber sie ermöglichen einen Test (Ping) durch Abfrage eines Servers und damit die Überprüfung des WLAN-Verbindungsstatus und die Ausgabe des Ergebnisses auf der seriellen Konsole

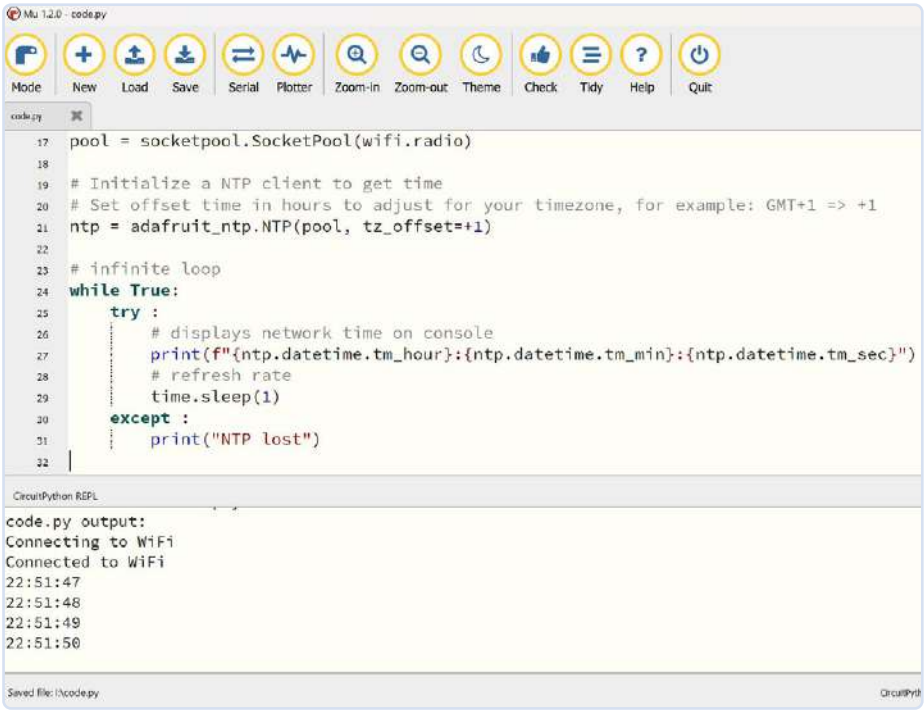


Bild 10. Die NTP-Zeit wird in der seriellen Konsole angezeigt.

Hinzufügen nützlicher Bibliotheken

Die meisten der grundlegenden Bibliotheken sind schon in CircuitPython untergebracht, nicht aber alle verfügbaren, da sonst der Flash-Speicher eines Mikrocontrollers aus allen Nähten platzen würde. Je nach Projekt müssen Sie deshalb externe Bibliotheken hinzufügen.

Die Installation zusätzlicher Bibliotheken in CircuitPython besteht einfach darin, die entsprechenden Dateien auf das Laufwerk *CIRCUITPY* zu kopieren. Damit Sie es nur einmal machen müssen, installieren Sie alle Bibliotheken, die für die endgültige Version des Projekts nützlich sein könnten, in einem Rutsch:

1. Falls noch nicht vorhanden, erstellen Sie einen Ordner *./lib* auf Ihrem Laufwerk *CIRCUITPY*
2. Alle verfügbaren Bibliotheken können als ein einziges Archiv von der CircuitPython-Website heruntergeladen werden, sie sind aber auch einzeln auf GitHub verfügbar)
3. Klicken Sie auf der Webseite auf den Link *Libraries*
4. Laden Sie das Bundle herunter, das der Version Ihres CircuitPython entspricht, in unserem Fall die Version 8.x (siehe **Bild 9**)
5. Entpacken Sie es an einem beliebigen Ort auf Ihrem Computer
6. Öffnen Sie im entpackten Bundle-Ordner den Ordner *lib*
7. Wählen Sie aus:
   a. Die Ordner
      i. *adafruit_register*
      ii. *adafruit_imageload*
      iii. *adafruit_Anzeige_Text*
   b. Die Dateien
      i. *adafruit_debouncer.mpy*
      ii. *adafruit_ds3231.mpy*
      iii. *adafruit_ntp.mpy*
      iv. *adafruit_st7789.mpy*
      v. *adafruit_ticks.mpy*
8. Kopieren Sie die Auswahl in den Ordner *./lib* auf dem Laufwerk *CIRCUITPY*

Hinweis: CircUp ist ein in Python geschriebenes Tool, das die Version Ihrer Bibliotheken prüfen, aktualisieren und ihre Abhängigkeiten installieren kann [13].

Zeitanzeige in der Konsole

Im vorherigen Code haben Sie Ihren Raspberry Pico W mit dem WLAN-Netzwerk verbunden. Jetzt werden Sie einen dedizierten NTP-Server abfragen, damit er Ihnen die Uhrzeit und das Datum anzeigt. Der Code, mit dem Sie die Zeit abrufen und in der Konsole anzeigen können (**Bild 10**), ist in **Listing 2** zu finden. Werfen wir einen Blick auf die hinzugefügten Zeilen:

- **Zeile 5 und Zeile 18**: Import der nativen *socketpool*-Bibliothek und Erstellung eines Sockets, der die Client-Server-Verbindung aufrechterhält
- **Zeile 22**: Instanziierung eines NTP-Clients über den Socket. Mit dem Parameter `tz_offset` wird der Zeitunterschied zwischen UTC (Coordinated Universal Time [14]) und Ihrer eigenen Zeitzone einzustellen. In Frankreich verwenden wir CET wie in Deutschland, was im Grunde UTC+1 ist, daher der Wert +1. Der Standard-Server ist der Adafruit-Server, was geändert werden kann, und der Standard-Timeout beträgt 10 s.
- **Zeilen 24…32**: In einer Endlosschleife wird die Zeit in der Konsole entsprechend der gewünschten Aktualisierungsrate angezeigt (in unserem Fall 1 s – Zeile 29). Dies wird mit dem Parameter `ntp.datetime` erreicht, der ein benanntes Tupel (eine Unterklasse der integrierten Tuples mit der zusätzlichen Möglichkeit, Elemente per Namen statt nur per Index aufzurufen) der Klasse `time` zurückgibt, das

Listing 2: Code für den NTP-Request

```
# CircuitPython own's libraries
import time
import os
import wifi
import socketpool

# CircuitPython external libraries
import adafruit_ntp

print("Connecting to WiFi")

# connect to your WiFi network with SSID/PASSWORD in 'settings.toml'
wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'), os.getenv('CIRCUITPY_WIFI_PASSWORD'))

print("Connected to WiFi")

# Setting up a socket
pool = socketpool.SocketPool(wifi.radio)

# Initialize a NTP client to get time
# Set offset time in hours to adjust for your timezone, for example: GMT+1 => +1
ntp = adafruit_ntp.NTP(pool, tz_offset=+1)

# infinite loop
while True:
    try :
        # displays network time on console
        print(f"::")
        # refresh rate
        time.sleep(1)
    except :
        print("NTP lost")
```

Bild 11. Anzeige der NTP-Uhr auf dem LCD.

die Datums- und Zeitinformationen des Servers enthält. Wir greifen auf jedes Element des Tupels über sein benanntes Feld zu [15]. Wir verwenden das Konstrukt `try...except`, um eine Ausnahme abzufangen, die bedeuten würde, dass die Daten nicht empfangen wurden, sobald die Zeitüberschreitung erreicht ist.

Zeitanzeige auf dem LCD

In diesem Schritt verwenden wir den Bildschirm anstelle der Konsole, um die Zeit anzuzeigen. CircuitPython verfügt über eine native Bibliothek namens *displayio*. Diese Bibliothek umfasst Methoden und gemeinsame Attribute für jedes von CircuitPython unterstützte Display. Das Farb-LCD-Modul, das wir verwenden, bietet die folgenden Funktionen:

- SPI-Protokoll
- Auflösung 240 × 135 Pixel
- ST7789-Treiber

Sie können den Code über den angegebenen Weblink herunterladen. Es würde zu lange dauern, dies im Detail zu erklären, aber hier sind die wichtigsten Schritte, um das gleiche Ergebnis wie in **Bild 11** zu erreichen. Sie müssen mehrere Bibliotheken importieren:

1. Die nativen CircuitPython-Bibliotheken:
   board: für den Zugriff auf die Pin-Namen des Raspberry Pico W
   displayio: zur Verwaltung der Grafik
   busio: für die SPI-Bus-Kommunikation
   terminalio: um eine Schriftart für den Bildschirm zu erhalten
2. Externe Bibliotheken:
   adafruit_st7789: zum Verwalten des LC-Displays
   adafruit_display_text: für die Anzeige des Textbereichs auf dem Display

Anschließend müssen Sie die Display-Einstellungen konfigurieren:

1. Erstellen Sie den SPI-Kommunikationsbus `SPI_bus`
2. Erstellen Sie den Bus `display_bus`, um den Raspberry Pi Pico W mit dem LCD zu verbinden. Er enthält den SPI-Bus und zwei zusätzliche Signale (*command* und *chip_select*).
3. Erstellen Sie das Display `display`, indem Sie den vorherigen Bus bereitstellen, seine Auflösung (`width=135, height=240`) definieren und die Offsets in x- (`rowstart=40`) und y-Richtung (`colstart=53`) einstellen. Die Werte der beiden letzten Parameter lassen sich dadurch erklären, dass die höchste Auflösung des ST7789-Treibers 320 × 240 beträgt, die Auflösung unseres Displays aber nur 240 × 135 (siehe **Bild 12**).
4. In CircuitPython muss jedes grafische Element zu einer Gruppe gehören. Erstellen Sie eine Gruppe mit dem Namen `display_group` und anschließend einen Textbereich `time_label`. Fügen Sie den Textbereich der Gruppe

Listing 3: RTC-Code

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple demo of reading and writing the time for the DS3231 real-time clock.
# Change the if False to if True below to set the time, otherwise it will just
# print the current date and time every second.  Notice also comments to adjust
# for working with hardware vs. software I2C.

import time
import board
import busio
import adafruit_ds3231

i2c = busio.I2C(scl=board.GP7, sda=board.GP6)
rtc = adafruit_ds3231.DS3231(i2c)

# Lookup table for names of days (nicer printing).
days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")

# pylint: disable-msg=using-constant-test
if True:  # change to True if you want to set the time!
    #                     year, mon, date, hour, min, sec, wday, yday, isdst
    t = time.struct_time((2023, 03, 09, 14, 58, 15, 3, -1, -1))
    # you must set year, mon, date, hour, min, sec and weekday
    # yearday is not supported, isdst can be set but we don't do anything with it at this time
    print("Setting time to:", t)  # uncomment for debugging
    rtc.datetime = t
    print()
# pylint: enable-msg=using-constant-test

# Main loop:
while True:
    t = rtc.datetime
    # print(t)     # uncomment for debugging
    print(
        "The date is {} {}/{}/{}".format(
            days[int(t.tm_wday)], t.tm_mday, t.tm_mon, t.tm_year
        )
    )
    print("The time is {}:{:02}:{:02}".format(t.tm_hour, t.tm_min, t.tm_sec))
    time.sleep(1)  # wait a second
```

hinzu. Schließlich zeigen Sie die Gruppe auf dem Display an.

5. Um die Zeitangabe auf dem Bildschirm zu aktualisieren, müssen Sie in der Endlosschleife nur das `text`-Attribut unseres Textbereich-Objekts `time_label` ändern, indem Sie die Zeichenkette kopieren, die wir für die Anzeige auf der Konsole verwendet haben.

Test der Echtzeituhr

Unsere web-synchronisierte Uhr funktioniert einwandfrei, können wir also Feierabend machen? Nein, wir sind noch nicht fertig, denn was würde bei unterbrochener Netzwerkverbindung passieren? Oder bei einem vorübergehenden Ausfall der Stromversorgung?

Um die aktuelle Zeit nicht zu verlieren, werden wir eine Echtzeituhr hinzufügen. Die RTC wird von einer Batterie CR1220 gepuffert, so dass die Zeit jahrelang auf dem neuesten Stand erhalten bleibt, auch wenn die Uhr nicht über den Micro-USB-Anschluss mit Strom versorgt wird. Diese RTC verwendet den bekannten DS3231 mit I²C-Bus zur Kommunikation mit dem Raspberry Pi Pico W.

Bevor Sie die RTC mit dem NTP-Server kombinieren, wollen wir sie alleine testen, und dazu benötigen Sie die *adafruit_ds3231*-Bibliothek. Wie die meisten CircuitPython-Bibliotheken bietet sie Online-Tutorials und Beispiele [16].

Bei dem in diesem Tutorial vorgeschlagenen Code müssen Sie nur die ersten Zeilen wie in **Listing 3** gezeigt ändern. Allerdings unterstützen viele Raspberry Pi Pico W das I²C-Protokoll nicht [17], so dass Sie genau wissen müssen (oder Bild 3 zu Rate ziehen), welche Anschlüsse physisch mit der DS3231-RTC verbunden sind. Bei Ausführung des Codes sollten Sie ein ähnliches Ergebnis wie in **Bild 13** erhalten.

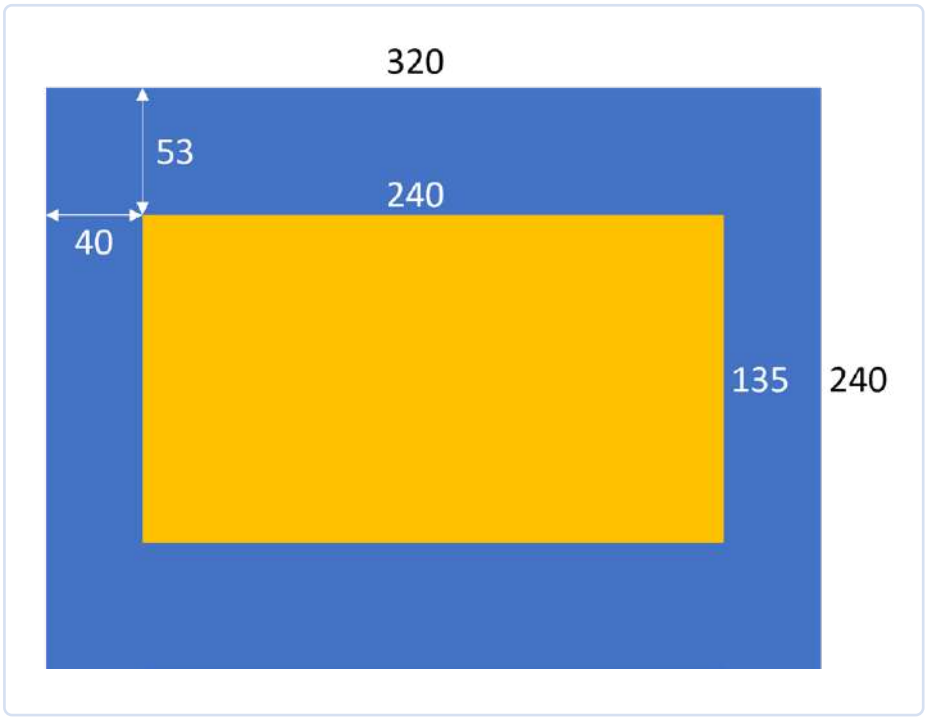


Bild 12. Offsets des ST7789-Displays.

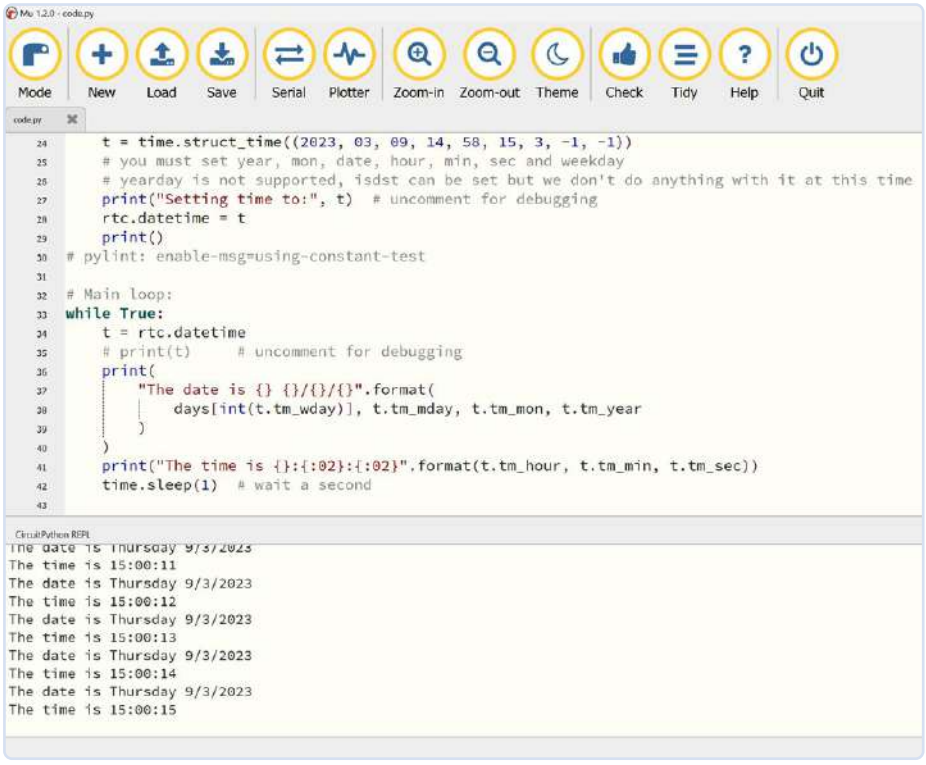


Bild 13. RTC-Ausgabe.

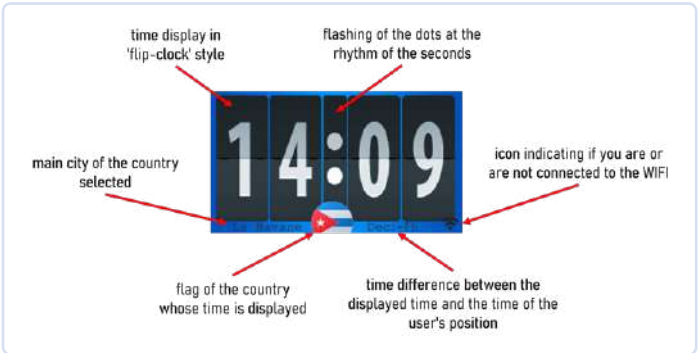


Bild 14. Endgültige Anzeige.

Bild 15. Spritesheet mit Ziffern.

Der finale Code

Die Hauptfunktion unseres Projekts ist erfüllt. Den Link zum Herunterladen des endgültigen Codes finden Sie unter [24]. **Bild 14** zeigt die endgültige Bildschirmanzeige mit einer verbesserten grafischen Darstellung und einigen neuen Elementen. Auch hier wäre eine ausführliche Erklärung mühsam, aber das Tutorial [18] klärt alle notwendigen Grundlagen. Was ist anders als in den vorherigen Listings?

- Der Code zur Verwaltung der Echtzeituhr (RTC) wurde mit dem Code verbunden, der die Zeit über den NTP-Server abruft
- Die Zeit wird nun dank Bitmap-Bildern angezeigt, die mit einer speziellen Software vorbereitet wurden. Wir müssen die *adafruit_imageload*-Bibliothek verwenden, um sie in den Speicher zu laden. Um zu vermeiden, dass bei jedem Ziffernwechsel ein neues Bild geladen wird (und so die Anzeige zu langsam werden könnte), müssen wir auf ein „Sprite Sheet" zurückgreifen (**Bild 15**), eine Art Verschiebebildchen, bei dem nur festgelegt werden muss, welcher Teil des Bildes angezeigt werden soll. Das gleiche Verfahren kann für die blinkenden Punkte angewendet werden.
- Mit dem Joystick auf dem HAT mit dem 1,14-Zoll-LCD können wir ein Land auswählen und die aktuelle Zeit in diesem Land sowie die Zeitdifferenz zur UTC anzeigen. Um zu verhindern, dass die Tasten des Joysticks prellen, ist die Bibliothek *adafruit_debouncer*

Bild 16. Spritesheet mit Flaggen.

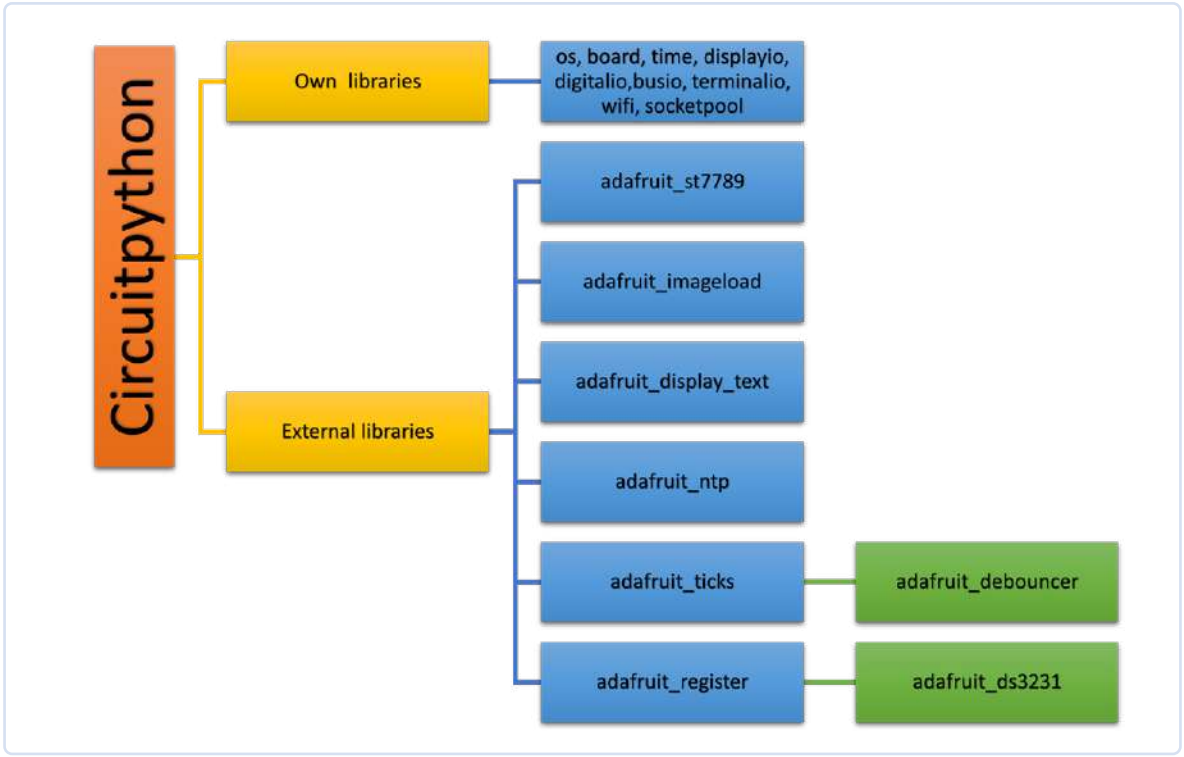


Bild 17. Baumstruktur der Bibliotheken.

erforderlich, die wiederum die *adafruit_ticks*-Bibliothek benötigt, um zu funktionieren.

- Die Flaggenanzeige verwendet ebenfalls eine Sprite-Sheets-Bitmap mit den Flaggen von 25 Ländern (**Bild 16**). Im Internet finden sich zahlreiche Webseiten mit kostenlosen Sprite Sheets mit anderen Flaggen.
- Die Beschriftung am unteren Bildrand (Stadt und Zeitdifferenz zur UTC) sind allerdings keine Sprite-Sheets, sondern normale Texte wie zuvor bei der Anzeige der Uhrzeit auf dem LCD.

Die Baumstruktur der in diesem Projekt verwendeten internen und externen Bibliotheken ist in **Bild 17** dargestellt.

Fazit

Es gibt natürlich viele Möglichkeiten, dieses Projekt zu verbessern, zum Beispiel die Verwendung einer LCD-Anzeige mit höherer Auflösung (die allerdings 320 × 240 nicht überschreiten sollte, da die Leistung des Raspberry Pi Pico W begrenzt ist), das Hinzufügen von akustischen Alarmen oder die Anzeige der aktuellen Wetterdaten für Ihren Standort… Diese Verbesserungen können dank der Ressourcen, die von Online-Bibliotheken zur Verfügung gestellt werden, oder dank der vielen Tutorials auf der Adafruit-Website leicht in ihr CircuitPython-Programm Aufnahme finden.
Und warum versuchen Sie nicht, Ihr eigenes CircuitPython-Projekt zu programmieren, um zu erkennen, wie einfach es ist, diese Sprache zu benutzen?

- Herunterladen: CircuitPython für den Raspberry Pi Pico W [18]
- Herunterladen: Externe Bibliotheken [19]
- Dokumentation: CircuitPython-eigene Bibliotheken [20]
- Dokumentation: Externe Bibliotheken [21]
- CircuitPython Essentials-Tutorials [22], [23]

Weitere Ressourcen finden Sie auch in einem (französischsprachigen) Buch, das ich über CircuitPython (Version 5.3) geschrieben habe (**Bild 18**). Es wurde zwar schon im Jahr 2020 von Elektor veröffentlicht, weshalb es natürlich viele Neuerungen gibt, aber die Grundlagen der Sprache sind gleich geblieben und der Code kann leicht auf andere Module wie den Raspberry Pi Pico W portiert werden (was eine der Hauptstärken von CircuitPython und seiner „unified hardware API“ ist). Und wenn Sie gar nicht mehr weiterkommen, können Sie Ihre Fragen jederzeit an meine E-Mail Adresse schicken!

Bild 18. Das Buch „Initiation au langage CircuitPython et à la puce nRF52840“

RG — 220633-02

Haben Sie Fragen oder Kommentare?

Wenn Sie technische Fragen haben oder einfach nur neue Artikelideen vorschlagen möchten, können Sie den Autor per E-Mail unter michael.bottin@univ-rennes.fr oder die Elektor-Redaktion unter redaktion@elektor.de kontaktieren.

Passende Produkte

- **Raspberry Pi Pico RP2040 W**
  https://elektor.de/20224
- **Michael Bottin, Initiation au langage CircuitPython et à la puce nRF52840**
  Buch (französisch), broschiert, SKU 19523: https://elektor.com/19523
  E-Buch (französisch), PDF, SKU 19524: https://tinyurl.com/4e5c7rj9

WEBLINKS

[1] MicroPython: https://de.wikipedia.org/wiki/MicroPython
[2] CircuitPython: https://en.wikipedia.org/wiki/CircuitPython
[3] Raspberry Pico W von Elektor: https://elektor.de/raspberry-pi-pico-rp2040-w
[4] LCD-HAT für Pico: https://shop.sb-components.co.uk/products/1-14-lcd-hat-for-pico
[5] RTC-HAT für Pico: https://shop.sb-components.co.uk/products/pico-rtc-hat
[6] Download CircuitPython: https://circuitpython.org/
[7] Download Mu-Editor: https://codewith.mu/en/download
[8] Tutorial Mu-Editor: https://codewith.mu/en/tutorials/1.2/start
[9] REPL-Konsole: https://codewith.mu/en/tutorials/1.2/repl
[10] Diagramm scrollen: https://codewith.mu/en/tutorials/1.2/plotter
[11] Tastatur-Shortcuts: https://codewith.mu/en/tutorials/1.2/shortcuts
[12] TOML-Konfigurationsdatei: https://de.wikipedia.org/wiki/TOML
[13] Utility CircUp: https://learn.adafruit.com/keep-your-circuitpython-libraries-on-devices-up-to-date-with-circup/
[14] UTC (Koordinierte Weltzeit): https://www.timeanddate.de/zeitzonen/utc-gmt
[15] Klasse des Zeitwertes: https://docs.python.org/3/library/time.html#time.struct_time
[16] Bibliothek adafruit_ds3231, Tutorials und Beispiele: https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout/circuitpython
[17] Pinbelegung Pico W: https://datasheets.raspberrypi.com/picow/PicoW-A4-Pinout.pdf
[18] CircuitPython-Version für den Raspberry Pico W: https://circuitpython.org/board/raspberry_pi_pico_w/
[19] Externe Bibliotheken: https://circuitpython.org/libraries
[20] Dokumentation der CircuitPython-Bibliotheken: https://docs.circuitpython.org/en/latest/shared-bindings/index.html#modules
[21] Dokumentation zu externen Bibliotheken: https://docs.circuitpython.org/projects/bundle/en/latest/drivers.html
[22] CircuitPython-Grundlagen, Tutorials 1: https://learn.adafruit.com/welcome-to-circuitpython/circuitpython-essentials
[23] CircuitPython-Grundlagen, Tutorials 2: https://learn.adafruit.com/welcome-to-circuitpython/
[24] Software-Download: https://elektormagazine.de/220633-02

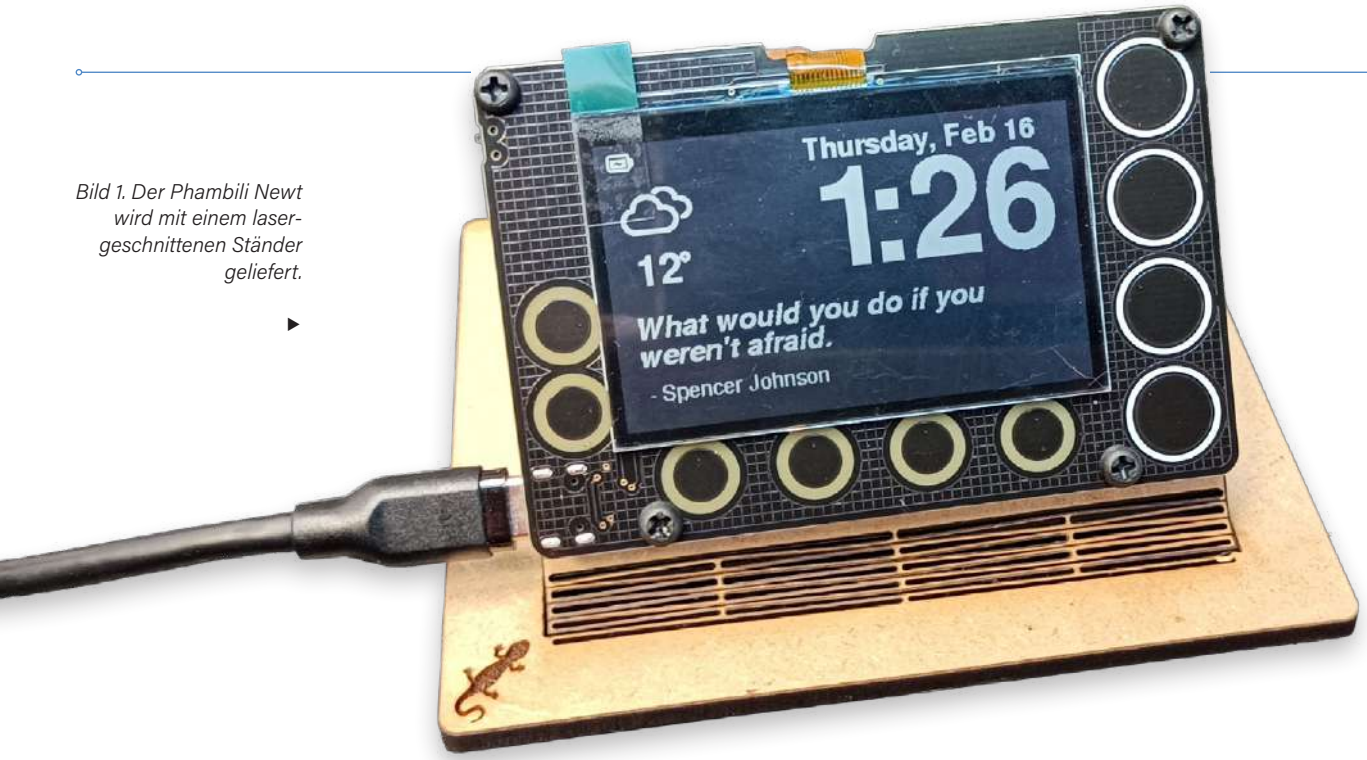


Bild 1. Der Phambili Newt wird mit einem lasergeschnittenen Ständer geliefert.

▶

Bauen Sie ein cooles **IoT-Display**

Mit dem Phambili Newt

Von Clemens Valens (Elektor)

Erforschen Sie den Phambili Newt, ein kompaktes und anpassbares Display-Moduls, das mehr bietet, als man auf den ersten Blick sieht! Werfen wir aber zunächst einen Blick auf seine einzigartigen Eigenschaften, von der Grundfunktionalität bis hin zu den spannenden benutzerprogrammierbaren Anwendungen, die es zu einem vielversprechenden Werkzeug für alle machen, die sich für IoT-Geräte interessieren.

Der Phambili Newt (Newt: Molch) ist ein batteriebetriebenes, immer eingeschaltetes und wandmontierbares Display, das online gehen kann, um Informationen aus dem Internet abzurufen und sie darzustellen. Das Modul ist etwas größer als eine Kreditkarte und verfügt über zehn Touchpads sowie eine 2,7-Zoll-Anzeige mit 240 x 400 Pixeln, die einem E-Ink-Display ähnelt. Hinter dem Display befindet sich ein ESP32-S2-Mikrocontroller, den man mit Arduino, CircuitPython, MicroPython oder dem ESP-IDF programmieren kann. Der Newt soll von einer Batterie versorgt werden, die aber nicht im Kit enthalten ist. Glücklicherweise können Sie ihn auch über ein Handy-Ladegerät mit USB-C betreiben. Laut Dokumentation soll eine Li-Po-Batterie mit mindestens 500 mAh das Gerät für bis zu zwei Monate zwischen den Ladevorgängen mit Energie versorgen. Die Batterie muss einen zweipoligen sogenannten JST-Anschluss haben. Obwohl der Newt für Dauerbetrieb gedacht ist, verfügt er über einen kleinen Ein-/Ausschalter, sodass er auch ausgeschaltet werden kann.

Der Phambili Newt wird mit einem lasergeschnittenen Holz-Ständer (mit Molch-Symbol) und einigem Montagematerial geliefert. Mit dem Ständer können Sie das Display zum Beispiel auf Ihrem Schreibtisch aufstellen, aber es ist etwas wackelig, wenn Sie die Tasten berühren. Eine stabilere Option ist es, die vier Magnetfüße zu montieren und damit das Newt an einen Kühlschrank oder eine andere Metallfläche zu „kleben".

Was kann der Phambili Newt?

Nach dem Auspacken macht der Newt nicht viel, da er erwartet, dass Sie zuerst eine WLAN-Verbindung konfigurieren. Nach dem Einschalten gibt das Modul daher Anweisungen, wie dies zu bewerkstelligen ist. Das ist zwar einfach, aber etwas langsam, und das Gerät startete mehrmals neu, bevor die Verbindung zu meinem Netzwerk stand. Als es dann endlich funktionierte, zeigte es Datum und Uhrzeit an und blieb dann stecken.

Ich stellte fest, dass die Firmware auf meinem Newt die frühe Versionsnummer v0.0.11 hatte, also suchte ich nach einer neueren Version. Nachdem ich die neuste Version [2] (v1.1.15) geladen hatte, funktionierte der Newt wie erwartet. Die Aktualisierung der Firmware ist ganz einfach: Schließen Sie den Newt an einen Computer an, sodass er sich als externer Datenträger anmeldet, und kopieren Sie die neue Firmware-Datei darauf.

Mit der aktuellen Firmware verband sich der Newt dann schnell mit meinem Netzwerk und zeigte sofort Zeit, Datum und Wetterinformationen für meinen Standort an. Die Wetterdaten (und ein Sinnspruch darunter) wechseln alle drei Minuten.

Durch Berühren der oberen Taste auf der rechten Seite öffnet sich ein Menü am unteren Rand des Bildschirms. Es gibt drei „Seiten“ mit Wecker und Timer, Wetter- und Luftqualitätsinformationen, einem Kalender und anderen nützlichen oder unterhaltsamen Dingen. Über das Menü haben Sie auch Zugriff auf die Einstellungen und die Firmware-Aktualisierung. Leider ist die Befestigungsschraube der Menütaste ein wenig im Weg (ebenso wie die Schraube in der Nähe der Taste unten).

Eigene Anwendungen

Obwohl die grundlegende Funktionalität des Newt schön ist, ist es wahrscheinlich nicht der Grund, warum Sie einen kaufen würden. Seine wahre Stärke liegt darin, dass er sozusagen hackbar ist. Der Quellcode der Firmware ist auf GitHub verfügbar, zusammen mit Anweisungen zur Einrichtung der Arduino-IDE für das Schreiben eigener Newt-Anwendungen. Ein I²C-Anschluss im Qwiic-Format (SparkFun) ermöglicht das Anschließen von Sensoren und anderen Erweiterungen an den Newt, was ihn damit zu einem echten IoT-Gerät anstatt nur einer vernetzten Uhr macht.

Fazit

Der ESP32-S2 ist ein leistungsstarker Mikrocontroller, obwohl er kein Bluetooth, sondern nur WLAN-Fähigkeiten hat. Das Schwarz-Weiß-Display sieht sehr schön aus und ist im Gegensatz zu normalen E-Ink-Displays schnell, so dass das Zeichnen auf dem Display unmittelbar erfolgt. Die Kombination von ESP32-S2 und Display ergibt ein starkes Duo mit vielen Anwendungsmöglichkeiten, insbesondere im Bereich des Low-Power-IoT. Der I²C-Erweiterungsanschluss bietet noch mehr Möglichkeiten.

230345-02

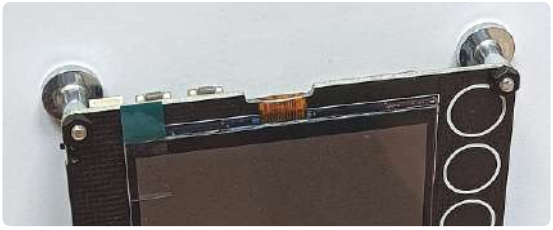


Bild 2. Mit den Magnetfüßen kann der Newt an einen Kühlschrank oder eine andere ferromagnetische Oberfläche angebracht werden.

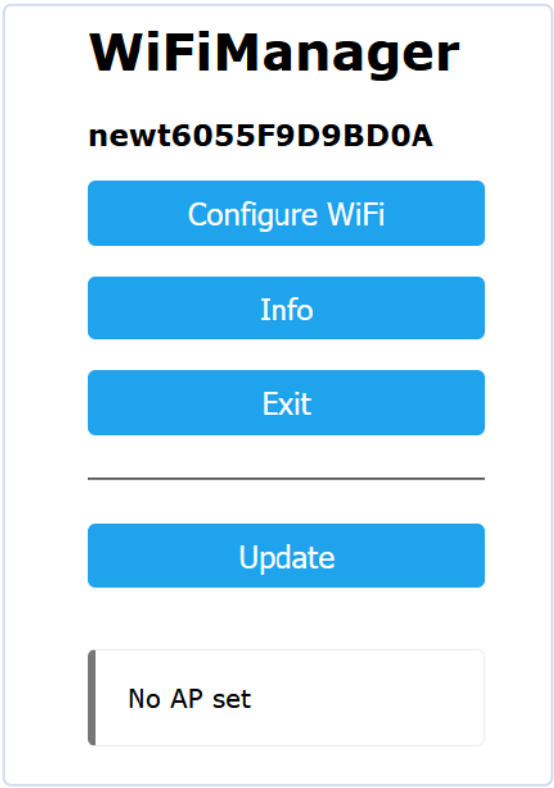


Bild 3. Menü für die Konfigurierung des Phambili Newt.

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Senden Sie eine E-Mail an clemens.valens@elektor.com, um mit dem Autor in Kontakt zu treten.

Passende Produkte

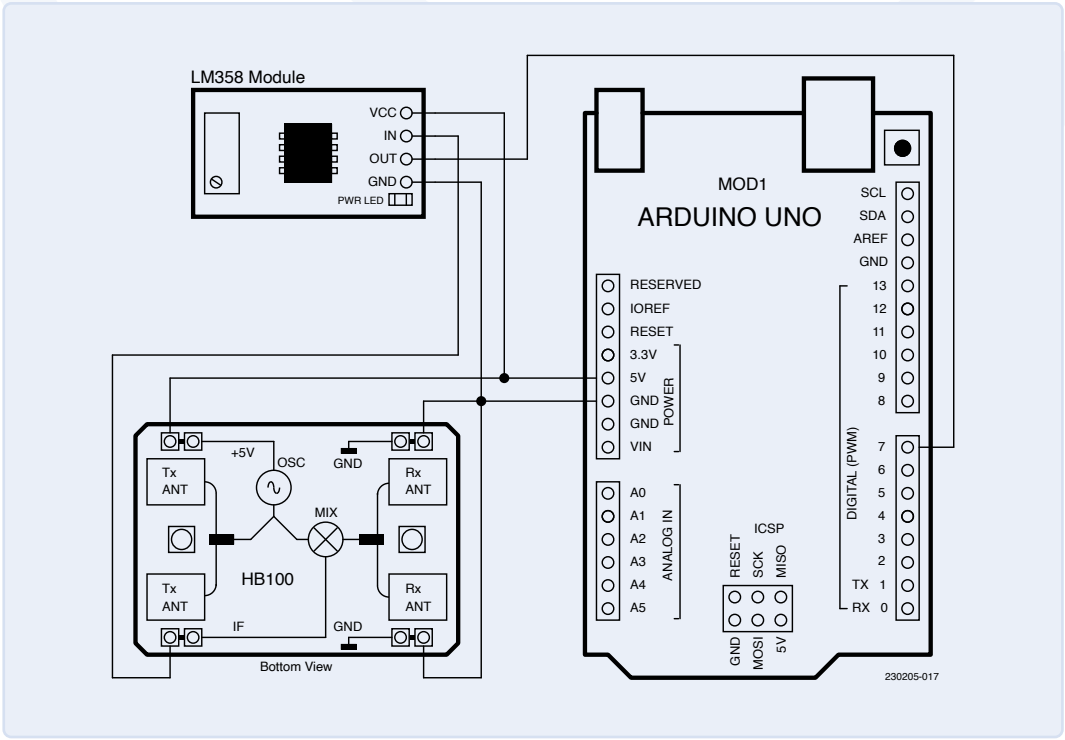
- **2,7″-IoT-Display Phambili Newt (mit ESP32-S2)**
  https://elektor.de/20230
- **SparkFun Umweltsensor-Kombi CCS811/BME280 auf Breakout-Board mit Qwiic**
  https://elektor.de/19580

WEBLINKS

[1] Günter Spanner, „MicroPython für den ESP32 und Co. - Teil 1“: https://www.elektormagazine.de/articles/micropython-fur-den-esp32-und-co
[2] Neuste Firmware-Version für den Newt: https://phambili-pub.s3.amazonaws.com/Newt.ino_latest.bin
[3] Einrichten der Arduino-IDE: https://github.com/Phambili-Tech/Newt_Display/wiki/Arduino-Setup

Der Doppler-Bewegungssensor HB100

Theorie und Praxis eines ungewöhnlichen Sensors



Von Stefano Lovati (Italien)

Um die Bewegung von Menschen, anderen Tieren oder verschiedenen Objekten zu erkennen, bedarf es des Einsatzes geeigneter Bewegungssensoren. Dieser Artikel befasst sich mit einem speziellen Sensortyp, der aufgrund seiner einzigartigen Fähigkeiten und seiner pädagogischen Bedeutung geschätzt wird: dem Mikrowellensensor. Dieses Bauteil zur Bewegungserkennung nutzt den Dopplereffekt, das gleiche Prinzip, das auch in modernen Radarsystemen verwendet wird.

Im Gegensatz zu Infrarot-Bewegungsmeldern erkennen Mikrowellensensoren die Bewegung eines Objekts durch die Analyse der vom Objekt reflektierten Mikrowellen. Im Vergleich zu anderen Sensortypen sind die Arten von Objekten, die erfasst werden können, nicht auf den menschlichen Körper beschränkt. Außerdem ist der Mikrowellensensor unabhängig von der Umgebungstemperatur, besitzt einen großen Messbereich und weist eine hohe Empfindlichkeit auf. Aufgrund dieser Eigenschaften wird er häufig in industriellen Geräten, im Transportwesen, bei der automatischen Türkontrolle, als Parksensor und als Geschwindigkeitsmesser eingesetzt. Aufgrund seiner Fähigkeit, verschiedene Arten von Objekten zu erkennen, wird dieser Sensor in vielen realen Anwendungen mit einem anderen Sensortyp kombiniert, um eine gezielte und ausfallsichere Erkennung zu erzielen. So kann der Mikrowellensensor beispielsweise mit einem Infrarot-Anwesenheitssensor (Infrared Presence Sensor, PIR) kombiniert werden, um den Durchgang einer Person durch eine Schranke oder deren Anwesenheit sicher(er) zu bestimmen und mögliche Störquellen auszuschalten.

Der Doppler-Effekt

Der Doppler-Effekt bezieht sich auf die Änderung der Frequenz beziehungsweise der Tonhöhe, wenn sich eine Schallquelle dem Hörer nähert oder sich von ihm entfernt, oder umgekehrt, wenn sich der Hörer der Schallquelle nähert oder sich von ihr entfernt. Jeder von uns hat diesen Effekt bereits praktisch erfahren können: Der von einem Martinshorn erzeugte Ton (nehmen wir der Einfachheit halber an, dass er monoton ist und eine feste Frequenz hat) verändert sich, wenn sich der Rettungswagen auf uns zubewegt oder von uns wegfährt. Dieses vom österreichischen Physiker Christian Doppler beschriebene Prinzip gilt nicht nur für Schall-, sondern auch für Radiowellen oder elektromagnetische Strahlung allgemein. Diese scheinbare Frequenzänderung zwischen der Quelle einer Welle und dem Empfänger wird genau durch die relative Bewegung zwischen der Quelle und dem Empfänger bestimmt.

Um den Doppler-Effekt besser zu verstehen, nehmen wir an, dass die Frequenz beziehungsweise Wellenlänge eines von einer Quelle kommenden Schalls konstant bleibt. Wenn sowohl die Quelle als auch der Empfänger stationär sind, hört der Empfänger immer dieselbe Schallfrequenz, die von der Quelle erzeugt wird. Das liegt daran, dass der Empfänger die gleiche Anzahl von Wellen pro Sekunde wahrnimmt, die von der Quelle erzeugt werden. Bewegen sich dagegen die Quelle, der Empfänger oder beide aufeinander zu, nimmt der Empfänger einen Ton mit höherer Frequenz wahr. Dies liegt daran, dass der Empfänger mehr Schallwellen pro Sekunde wahrnimmt und dies als einen Schall mit höherer Frequenz interpretiert. Wenn sich Quelle und Empfänger voneinander entfernen, nimmt der Empfänger weniger Schallwellen pro Sekunde wahr und empfindet einen Ton mit niedrigerer Frequenz. Eine grafische Darstellung dieses Prinzips liefert **Bild 1**. Wenn sich das Fahrzeug der Quelle (dem Sender) nähert, erhöht sich die Frequenz der reflektierten Welle, wenn sich das Fahrzeug jedoch entfernt, verringert sich die Frequenz der reflektierten Welle gegenüber der ursprünglichen Welle.

Einige Gleichungen

Für ein Radarsystem lässt sich der Wert in Hertz der Frequenzverschiebung f_D, die so genannte Dopplerfrequenz, nach folgender Formel berechnen:

$$f_D = \frac{2 \cdot v}{\lambda} \quad \textbf{(Gleichung 1)}$$

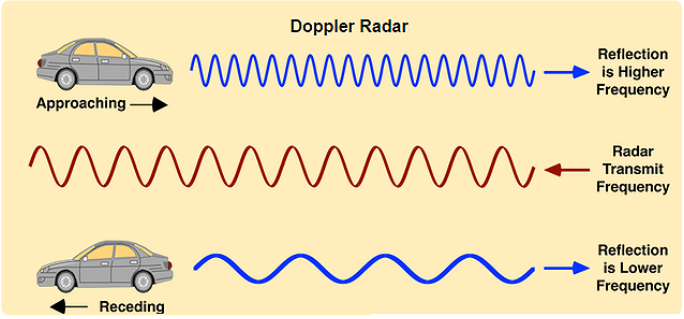


Bild 1. Beispiel für die Anwendung des Dopplereffekts. (Quelle: physicsopenlab.org, CC BY 4.0)

wobei *v* die Geschwindigkeit des erfassten Objekts in m/s und λ die Wellenlänge in Metern ist. Diese Formel ist gültig, wenn die Geschwindigkeit *v* eindeutig radial ist, also keine transversalen Komponenten hat, wie es bei einer seitlichen Bewegung des zu erfassenden Objekts (das das Signal reflektiert) und der Quelle (dem Sender) der Fall ist. Im allgemeinen Fall hat die obige Gleichung die Form:

$$f_D = \frac{2 \cdot v}{\lambda} \cdot \cos \propto \quad \textbf{(Gleichung 2)}$$

wobei α der Winkel ist, den die Richtung des gesendeten/reflektierten Signals mit der Bewegungsrichtung des zu erfassenden Objekts bildet. Man beachte, dass in dem Fall, in dem sich das zu erfassende Objekt ausschließlich senkrecht zur Richtung des gesendeten Signals bewegt, der Dopplereffekt sich aufhebt (f_D=*0*). Dies erklärt, warum bei Luftkämpfen im Stil von *Top Gun*, bei denen hochentwickelte Dopplereffekt-Radargeräte eingesetzt werden, Piloten, die vom Radar bedroht werden, immer danach streben, ihre Flugbahn senkrecht zur Flugbahn des Gegners zu kreuzen, was die Radarerfassung und die Verfolgung ihrer Position und Geschwindigkeit erschwert.

Im speziellen Fall eines Radars, das elektromagnetische Wellen aussendet (wie beim verwendeten Sensor), lautet die Formel, die den Absolutwert der Dopplerfrequenz f_D ausdrückt:

$$|f_D| = \frac{2 \cdot v_r}{\lambda} = \frac{2 \cdot v_r \cdot f_{TX}}{c_0} \quad \textbf{(Gleichung 3)}$$

wobei v_r die Radialgeschwindigkeit, f_{TX} die Frequenz des übertragenen Signals und c_0 die Lichtgeschwindigkeit ist.

Das HB100-Modul

In diesem Artikel werden wir die Eigenschaften eines der preiswertesten auf dem Dopplereffekt basierenden Anwesenheitserkennungssensoren kennenlernen, des HB100. Es handelt sich im Grunde um ein integriertes Modul, das einen X-Band-Mikrowellensender, eine Empfängerschaltung, einen Mischer und den gesamten HF-Teil umfasst und auf einer äußerst kompakten Leiterplatte aufgebaut ist. Um die Leser zu beruhigen, möchte ich gleich darauf hinweisen, dass trotz der hohen Frequenzen von etwa 10 GHz keine Gesundheitsrisiken bestehen, da die Sendeleistung sehr gering ist. Das Funktionsprinzip dieses Moduls, das bei den großen Elektronikhändlern für ein paar Euro erhältlich ist, ist nicht nur sehr interessant, sondern auch aus pädagogischer Sicht wichtig. Die von dem Modul verwendete Superüberlagerungsschaltung (Superheterodynschaltung) bildet die Grundlage moderner Radarsysteme und folgt dem klassischen Prinzip vieler Hochfrequenzempfänger. **Bild 2** zeigt das äußere Erscheinungsbild des sehr kompakten Moduls mit dem Metallgehäuse, das als Schutzschild für den HF-Teil dient. In **Bild 3** ist die Platine auf der Rückseite des Moduls zu sehen, die im Wesentlichen als Sende- und Empfangsantenne dient.

Der Miniatur-Bewegungsmelder HB100 ist ein Doppler-Transceiver-Modul, das im Mikrowellenbereich (X-Band) bei einer Frequenz von 10,525 GHz arbeitet. Im Inneren befinden sich ein dielektrischer

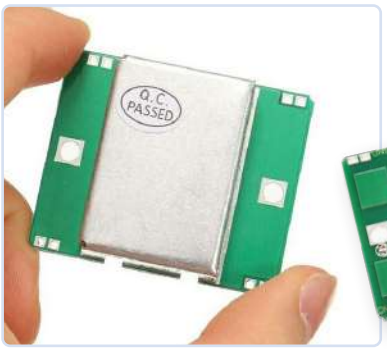


Bild 2. Oberseite des Doppler-Moduls HB100. (Quelle: [6])

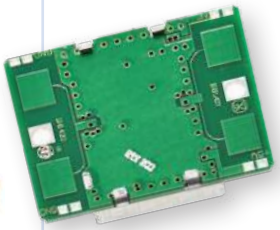


Bild 3. Ansicht des HB100-Moduls von unten. (Quelle: [7])

Resonator-Oszillator (DRO) und ein Antennenpaar, das direkt auf die Platine geätzt ist. Dieses Modul ist bestens geeignet, um die häufigen Fehlalarme in Einbruchmeldeanlagen zu unterbinden, insbesondere in Kombination mit einem herkömmlichen Passiv-Infrarot-Sensor (PIR). Der Sensor kann auch in automatischen Türöffnungssystemen und Tachometern in Fahrzeugen eingesetzt werden. Die wichtigsten Vorteile des HB100-Moduls sind:

- Berührungslose Bewegungserfassung
- Messung unbeeinflusst von Temperatur, Feuchtigkeit, Lärm, Luft und Staub. Der Sensor ist für besonders raue Umgebungsbedingungen geeignet.
- Ausgezeichnete Immunität gegen HF-Störungen
- Niedrige Ausgangsleistung. Das Gerät stellt, wenn man der US-amerikanischen Federal Communications Commission (FCC) Glauben schenken mag, auch in Gebäuden keine Gefahr für Menschen dar.
- Hohe Erfassungsreichweite (bis zu 20 Meter)
- Fähigkeit, die Bewegung verschiedener Objekte, nicht nur von Menschen, zu erkennen
- Hohe Richtwirkung der Funkwellen
- Niedrige Stromaufnahme
- Fähigkeit, sowohl im CW-Betrieb (Continuous Wave, das heißt kontinuierliche Übertragung des Funksignals) als auch im Impulsmodus (Übertragung kurzer, zeitlich aufeinander folgender Impulse) zu arbeiten
- Kompakte Abmessungen und sehr dünn

In **Bild 4** ist das Blockdiagramm des Moduls zu sehen, das wir nun im Detail analysieren wollen. Beginnen wir mit dem DRO-Oszillator, der auf die Frequenz von 10,525 GHz abgestimmt ist. Eine Keramikscheibe, in der Regel aus Bariumtitanat ($Ba_2Ti_9O_{20}$), dient als Resonanzkammer für die HF-Energie. Die Resonanzfrequenz hängt von der Größe und Form des Materials ab. Dies ist genau die Frequenz des Signals, das vom Modul durch die beiden rechteckigen Platinenantennen gesendet wird, die auf der linken Seite von Bild 4 zu sehen sind. Jedes Hindernis, das sich dem gesendeten Signal in den Weg stellt, erzeugt eine reflektierte Welle, deren Frequenz sich je nach der Bewegung des Objekts relativ zum Sender ein wenig (oder gar nicht) unterscheidet. Der Umgang mit einem HF-Signal mit diesen Frequenzen wäre sowohl auf der Hardware- als auch vor allem auf der Software-Ebene eine gewaltige Aufgabe. Die Lösung für dieses Problem stellt die Superüberlagerungsschaltung, genauer gesagt der HF-Mischer in Bild 4 dar. Seine Aufgabe ist es, das gesendete Signal mit dem über die beiden rechteckigen Antennen (rechts in Bild 4) empfangenen Signal zu „überlagern". Im Allgemeinen empfängt ein Mischer zwei Eingangssignale und erzeugt zwei Ausgangssignale, die jeweils die Summe und die Differenz der beiden Eingangssignale sind. In unserem Fall würden also zwei Signale erzeugt werden, eines mit einer Frequenz in der Größenordnung von 20 GHz (Summe des gesendeten und des empfangenen Signals) und das andere mit einer Frequenz von einigen Hertz (Differenz zwischen dem gesendeten und dem empfangenen Signal). Dieser Mechanismus löst unser Problem auf elegante Weise, da er das Problem der Messung eines Mikrowellensignals in die Messung eines niederfrequenten Signals (bis zu einigen hundert Hertz) verlagert. Und dies kann leicht von einem gewöhnlichen, kostengünstigen Mikrocontroller wie dem Arduino erledigt werden.

Das als Zwischenfrequenz (Intermediate Frequency, IF) bezeichnete Signal, das vom Mischer ausgegeben wird, ist genau das Differenzsignal. Dieses Verfahren kommt in vielen HF-Schaltungen zum Einsatz und wird auch schlicht als „Abwärtswandlung" bezeichnet. Sinn und Zweck bestehen in der Tat darin, eine hohe Frequenz abwärts zu wandeln, um ihre Handhabung zu vereinfachen. Die so erhaltene Frequenz wird als „Zwischenfrequenz" bezeichnet, um sie von der Original- oder „Basisband"-Frequenz zu unterscheiden.

In Bild 4 sind nur vier Pins zu sehen, die für den Anschluss des Moduls an einen Mikrocontroller oder eine Messschaltung erforderlich sind: zwei Masseanschlüsse, ein Anschluss für die positive 5-V-Versorgung und ein Anschluss für das niederfrequente ZF-Ausgangssignal. Das ist sehr schön, denn es geht nur darum, ein Signal mit einer Frequenz von nur wenigen Hertz zu verarbeiten. Die ganze unangenehme Verarbeitung des Mikrowellensignals bleibt der integrierten Hardware des HB100-Moduls zu überlassen. Allerdings gibt es immer noch ein Problem, das mit der eher geringen Amplitude des am Sensorausgang erzeugten Signals zu tun hat. Wir werden später sehen, wie dieses Problem zu lösen ist.

Für den praktischen Einsatz sollte das Modul so montiert werden, dass die Antennen (siehe Bild 3) auf den Bereich gerichtet sind, den Sie mit dem abgestrahlten Signal abdecken wollen, und dass sie so ausgerichtet sind, dass die beste Abdeckung erzielt wird. Die Antennen-Strahlungsdiagramme, sowohl in Azimut als auch in Elevation, sind in **Bild 5** zu sehen.

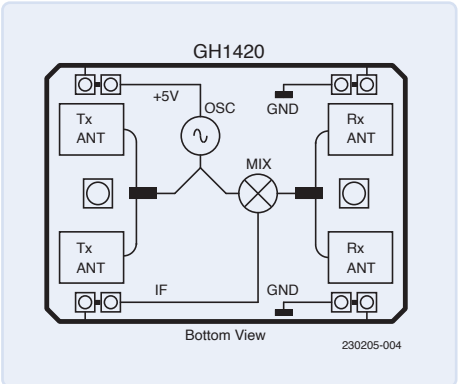


Bild 4. Blockschaltbild des Sensors. (Quelle: [5])

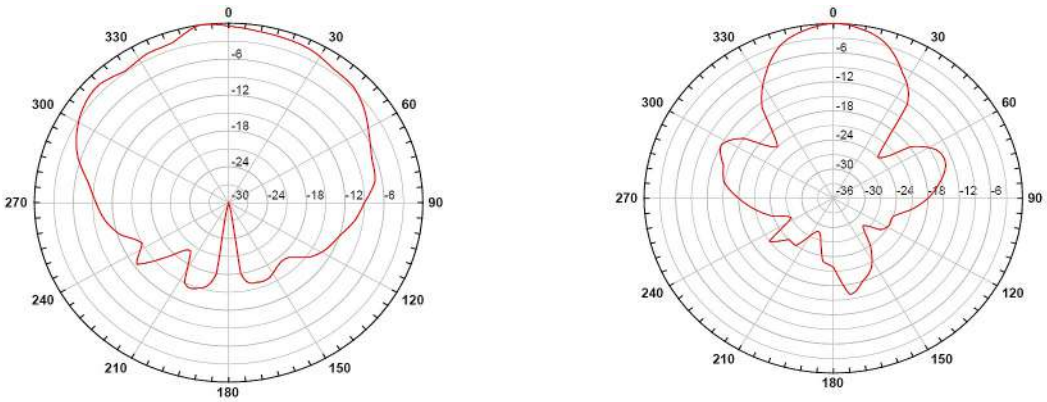


Bild 5. Strahlungsdiagramme der Antenne. (Quelle: [1])

Ausgangssignal

Der ZF-Pin liefert das Ausgangssignal des Moduls, das der Frequenzverschiebung entspricht, die durch die Größe der Bewegung eines von der Antenne beleuchteten Objekts bestimmt wird. Die Amplitude dieser Frequenzverschiebung, die durch den Dopplereffekt beeinflusst wird, ist proportional zur Reflexion der übertragenen Energie und liegt in der Größenordnung von einigen Mikrovolt (µV). Aus diesem Grund wird normalerweise ein Niederfrequenzverstärker mit hoher Verstärkung an den ZF-Pin angeschlossen, um eine Verstärkung des Ausgangssignals zu ermöglichen. Genauer gesagt, ist die Frequenz der Dopplerverschiebung proportional zur Geschwindigkeit der Bewegung. Nehmen wir an, das menschliche Gehtempo erzeugt typisch eine Frequenzverschiebung von weniger als 100 Hz. Die Empfangssignalstärke (RSS) entspricht der am ZF-Ausgang gemessenen Spannung. Die Dopplerfrequenz kann mit den oben genannten Gleichungen berechnet werden. Bewegt sich das vom Sensor beleuchtete Objekt nur entlang der radialen Linie und nähert oder entfernt es sich, so wird die vorstehende Formel vereinfacht und erhält folgende Form, wobei v_r die radiale Geschwindigkeit in km/h ist:

$$f_D = 19.49 \cdot v_r \quad \textbf{(Gleichung 4)}$$

Die technischen Daten des Moduls lassen sich wie folgt zusammenfassen:

- Betriebsspannung: 5 V ± 0,25 V
- Stromaufnahme (im Dauerübertragungsmodus): 60 mA maximal, 37 mA typisch
- Größe: 61,2 x 61,2 mm
- Erfassungsabstand: zwischen 2 m und 16 m
- Sendefrequenz: 10,525 GHz
- Frequenzgenauigkeit: 3 MHz
- Minimale Ausgangsleistung: 13 dBm EIRP
- Oberwellenemission: < -10 dBm
- Gemittelter Strom: 2 mA typisch
- Minimale Sendeimpulsbreite: 5 µs
- Minimales Tastverhältnis: 1 %

Schaltungen zur Signalaufbereitung

Am ZF-Ausgang liefert der HB100-Sensor ein niederfrequentes Signal, das der durch das empfangene Signal verursachten Frequenzverschiebung entspricht, allerdings nur mit einer sehr geringen Amplitude von einigen Mikrovolt. Daher ist in praktischen Anwendungen eine angemessene Verstärkung erforderlich, vorzugsweise in Verbindung mit einem Tiefpassfilter, um Störfrequenzen oberhalb von einigen hundert Hertz zu eliminieren. Sowohl das Datenblatt [1] als auch der Anwendungshinweis [2] des Sensors sind in dieser Hinsicht sehr nützlich. Insbesondere letzterer zeigt zwei mögliche Schaltungen für die Signalaufbereitung. Die erste in **Bild 6** gilt für den kontinuierlichen Übertragungsmodus (Continuous Wave, CW) und basiert auf dem hochverstärkenden Operationsverstärker LM324 von Texas Instruments. Die zweite Schaltung in **Bild 7** ist für den Impulsmodus vorgesehen, bei dem der Sender eine Folge von Impulsen mit einer bestimmten Frequenz (bekannt als Pulse Repetition Frequency, PRF) und einem

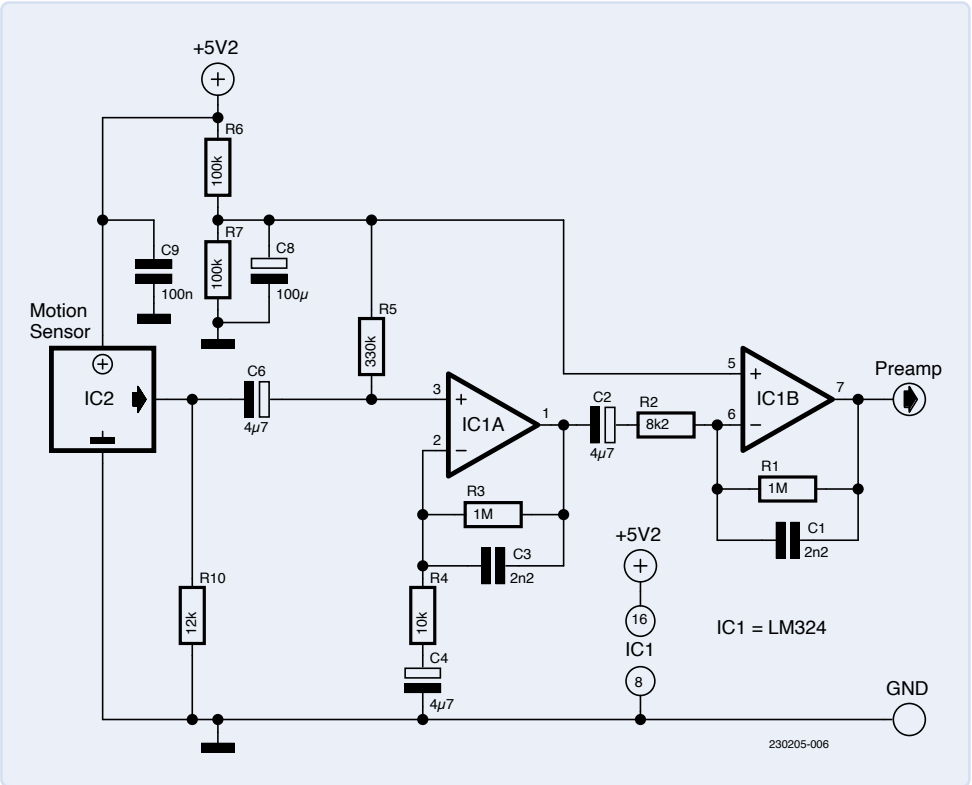


Bild 6. Signalaufbereitung für den CW-Betrieb. (Quelle: [5])

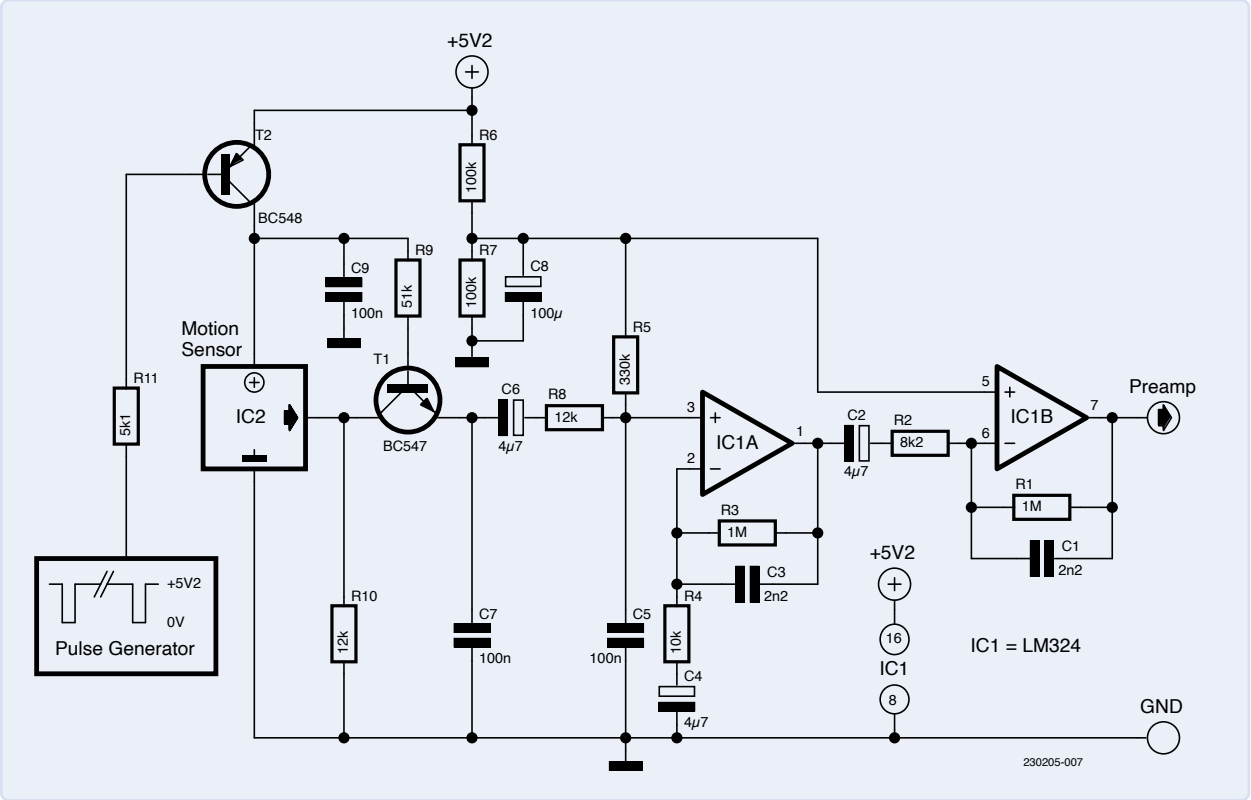


Bild 7. Signalaufbereitung für den Pulsbetrieb. (Quelle: [5])

bestimmten Tastverhältnis aussendet. Der Hersteller des HB100-Moduls empfiehlt eine PRF von 2 kHz und ein Tastverhältnis von 4 %.
In dieser Schaltung werden neben dem hochverstärkenden Operationsverstärker LM324 zwei Transistoren eingesetzt. Wie unten links in Bild 7 zu sehen ist, kann der Impulsmodus durch direkte Beeinflussung der Versorgungsspannung des Moduls aktiviert werden, wobei die vom Hersteller festgelegte (und empfohlene) PRF und das Tastverhältnis einzuhalten sind. Mit anderen Worten, im Impulsmodus schaltet der Transistor T2 alle 0,5 ms (2 kHz) den Sensor mit einem Tastverhältnis von 4 % ein.
Um jedoch die Tests mit dem Sensor zu vereinfachen, wurde ein Allzweck-Operationsverstärker gewählt, der bereits auf einem Breakout-Board mit Stiftleiste und Trimmpoti für die Verstärkungseinstellung aufgebracht ist. Das sehr preiswerte und leicht erhältliche Modul in **Bild 8** basiert auf dem LM358, der eine einkanalige, variable Verstärkung zwischen 1 × und 100 × bietet, die mit dem mitgelieferten Potentiometer eingestellt werden kann. Dreht man die Schraube im Uhrzeigersinn, verringert sich die Verstärkung (und umgekehrt natürlich). Außerdem verfügt das Modul über eine LED, die die korrekte Stromversorgung anzeigt. Der LM358-Opamp kann Signale mit Frequenzen bis zu 700 kHz verarbeiten (also weit über den uns interessierenden Bereich hinaus) und kann mit einer Spannung zwischen 3 V und 32 V versorgt werden. In Bild 8 sind die Anschlüsse des Moduls gekennzeichnet: V_{CC}, GND, Signal In und Signal Out.

Testen mit Arduino

Zum Testen des HB100-Dopplersensors können wir ein gewöhnliches Arduino-UNO-Board und das LM358-Verstärkermodul oder alternativ jeden anderen einkanaligen, rauscharmen Verstärker mit hoher Verstärkung und einfacher Versorgung verwenden. Auf einem Arduino-UNO-Board wird ein spezieller Sketch ausgeführt, der die Frequenz des vom Modul ausgegebenen Signals erfasst und daraus die Geschwindigkeit ableitet, mit der sich das potenzielle „Ziel" bewegt. Der betreffende Sketch verwendet eine Bibliothek, die speziell für die genaue Frequenzmessung entwickelt wurde, denn für Messungen von Frequenzen im Audiobereich oder darunter muss die Periodenlänge des Eingangssignals sehr genau bestimmt werden. Zu diesem Zweck bedient sich die Bibliothek eines hochgenauen „Counter and Capture"-Moduls, das in der Hardware-Architektur des ATmega-Mikrocontrollers verfügbar ist. Die Bibliothek liefert die gemessene Periodendauer, ausgedrückt als Ganzzahl mit einer Auflösung von 1/16 µs. Um die Frequenz abzuleiten, dividieren Sie einfach die Taktfrequenz durch den von der Bibliothek zurückgegebenen Wert. In unserem Fall wird die Taktfrequenz auf 16.000.400 gesetzt, um eventuelle Ungenauigkei-

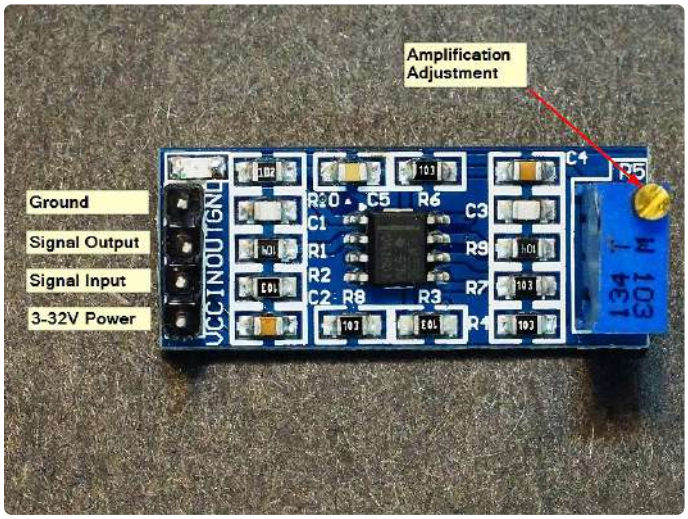


Bild 8. Das Verstärkermodul LM358.

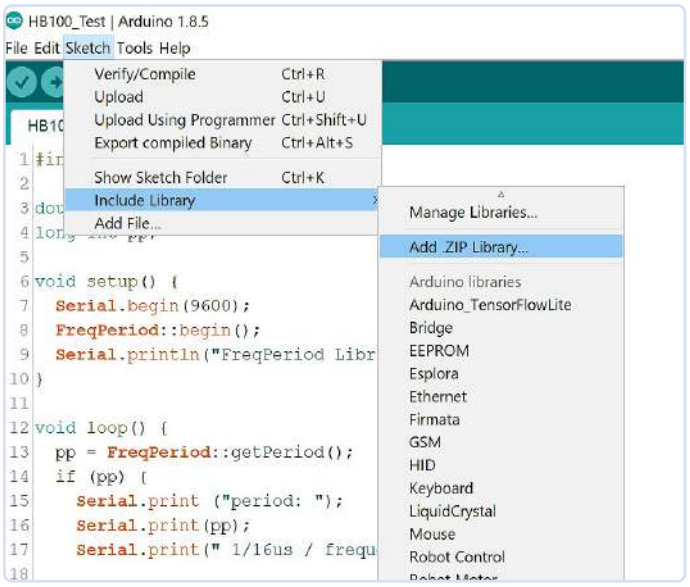


Bild 9. Installation der FreqPeriod-Bibliothek.

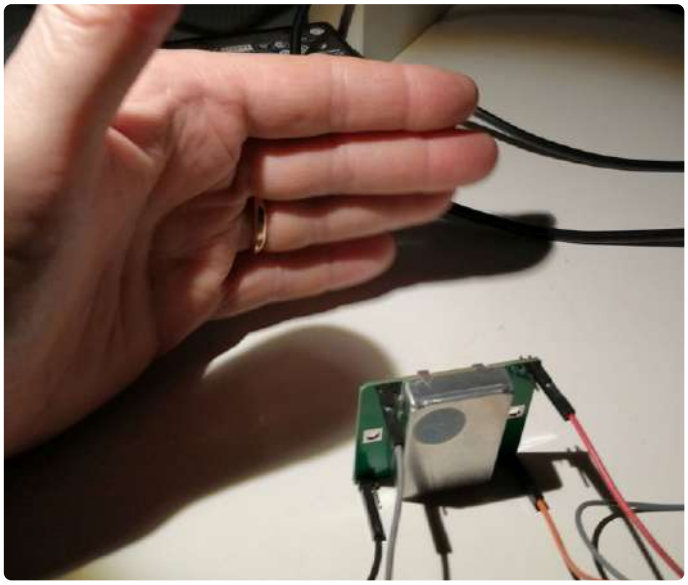


Bild 10. Schnelle Handbewegungen vor der Sensorantenne …

ten auf dem Arduino-Board auszugleichen. Die Bibliothek ist derzeit nicht in der offiziellen Arduino-Bibliothek enthalten, aber sie kann leicht in der IDE installiert werden. Nachdem Sie die .zip-Datei von [3] heruntergeladen haben, öffnen Sie die Arduino-IDE und wählen den Menüpunkt *Sketch -> Bibliothek einbinden -> .ZIP-Bibliothek hinzufügen…*, wie in **Bild 9** gezeigt. Wählen Sie dann die *.zip*-Datei der zuvor heruntergeladenen Bibliothek, sagen Sie *installieren* und schließen Sie die IDE. Beim nächsten Start ist die neue Bibliothek in der Arduino-Umgebung verfügbar.

Der vollständige Sketch ist in **Listing 1** zu sehen. Er ist sehr kompakt und relativ einfach. In der `setup()`-Funktion werden die serielle Leitung (mit 9600 bps) und die *FreqPeriod*-Bibliothek über die `begin()`-Methode initialisiert. In der `loop()`-Funktion wird zunächst der Periodenwert des Eingangssignals über den Aufruf der `getPeriod()`-Methode wiederholt erfasst. Anschließend werden die entsprechenden Werte von Frequenz (in Hertz) und radialer Geschwindigkeit (in km/h) berechnet und angezeigt.

Um die Funktionsweise des Sketches zu überprüfen, müssen wir den HB100-Sensor und das LM358-Verstärkermodul mit dem Arduino-Board verbinden. Diese Anschlüsse sind im **Titelbild** des Artikels zu sehen.

Dann schalten wir die Schaltung ein und aktivieren den *Seriellen Monitor* der Arduino-IDE, um die erzeugten Protokolle zu beobachten. Wenn wir ein Objekt, zum Beispiel die Hand wie in **Bild 10** vor der Antenne des HB100-Sensors (die sich, wie wir uns erinnern, auf der Rückseite der Platine befindet) bewegen, sollten wir eine Änderung der vom Sketch berechneten Geschwindigkeit sehen. Ein Beispiel für eine solche über die serielle Schnittstelle gesendete Ausgabe finden Sie in **Bild 11**.

Mit diesem Geschwindigkeitswert lassen sich dann auch komplexere Algorithmen mit Anwendungen wie Türbetätigung (einschließlich Schiebetüren), automatisches Schalten von Treppenhauslicht, Öffnen

Listing 1: Arduino-Sketch

```
#include <FreqPeriod.h>
double lfrq;
long int pp;

void setup() {

   Serial.begin(9600);
   FreqPeriod::begin();
   Serial.println("FreqPeriod Library Test");

}

void loop() {

   pp = FreqPeriod::getPeriod();

   if (pp) {
     Serial.print ("period: ");
     Serial.print(pp);
     Serial.print(" 1/16us / frequency: ");
     lfrq = 16000400.0 /pp;
     Serial.print(lfrq);
     Serial.print(" Hz ");
     Serial.print(lfrq/19.49);
     Serial.println( " km/h ");
   }

}
```

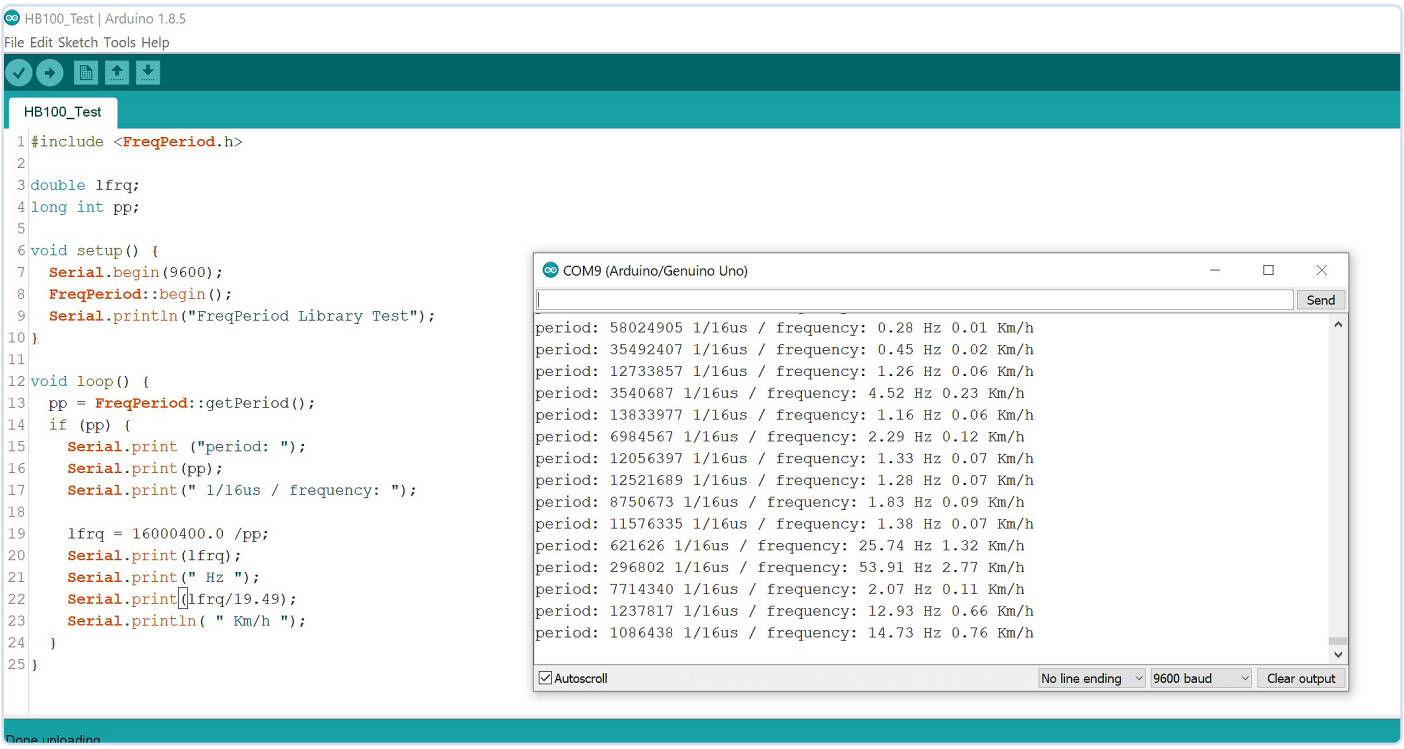


Bild 11. … und die Ausgabe im Seriellen Monitor.

von Fußgängertoren, Einbruchmeldeanlagen, Videoüberwachung und so weiter konstruieren. Die Empfindlichkeit des Moduls selbst ist ausreichend, vorausgesetzt, es arbeitet innerhalb seiner maximalen Reichweite von etwa 20 Metern. Die Leistung kann durch eine Verstärkerschaltung mit höherer Verstärkung als der des LM358 (100x) noch verbessert werden.

Schlussbetrachtung

Ziel dieses Artikels war es, einen sehr interessanten und wenig bekannten Sensor vorzustellen, den Bewegungs-Doppler-Sensor HB100. Der HB100 ist besonders preiswert und lässt sich leicht mit einem Mikrocontroller verbinden. Er bietet die Möglichkeit, wichtige Konzepte und Techniken der Signaldetektion zu erlernen und zu erproben, die anspruchsvollen und teuren kommerziellen Radargeräten zugrunde liegen. An Anwendungen für den Sensor mangelt es nicht und zweifellos denken viele Hersteller bereits über weitere, zukünftige Entwicklungen nach.

RG — 230205-02

Haben Sie Fragen oder Kommentare?

Bei technischen Fragen können Sie sich gerne per E-Mail an die Elektor-Redaktion wenden: redaktion@elektor.de.

Über den Autor

Nach seinem Abschluss in Elektrotechnik am Politecnico di Milano begann Stefano seine Karriere als Firmware- und Softwareentwickler. Im Laufe der Jahre hat er die Fortschritte der Embedded-Welt miterlebt und mitgemacht, von den ersten 8-Bit-Mikroprozessoren, die nur in Assembler programmierbar waren, bis hin zu den neuesten Soc, FPGA, DSP und programmierbarer Logik mit herausragender Leistung und Funktionen. Er hat ein umfassendes Interesse an Technologie und Elektronik und widmet einen Teil seiner Freizeit dem Studium neuer Bauteile und realisiert kleine Projekte.

Passende Produkte

- **YDLIDAR X2 Lidar - 360-Grad-Laserentfernungsmesser**
  SKU 18941: www.elektor.de/18941
- **Arduino Uno Rev3**
  SKU 15877: www.elektor.de/15877
- **Arduino Uno Mini (Limitierte Auflage)**
  SKU 20098: www.elektor.de/20098

WEBLINKS

[1] Datenblatt zum HB100-Modul: https://limpkin.fr/public/HB100/HB100_Microwave_Sensor_Module_Datasheet.pdf
[2] Application Note HB100: https://limpkin.fr/public/HB100/HB100_Microwave_Sensor_Application_Note.pdf
[3] Bibliothek zur Frequenzmessung: https://github.com/Jorge-Mendes/Agro-Shield/tree/master/OtherRequiredLibraries/FreqPeriod
[4] Software auf der Webseite dieses Projekts: https://elektormagazine.de/230205-02
[5] Schaltpläne: https://mantech.co.za/Datasheets/Products/MSAN-001_AGILSENSE.pdf
[6] Quelle des Fotos (Draufsicht): https://mantech.co.za/Datasheets/Products/HB100_RADAR.pdf
[7] Quelle des Fotos (Ansicht von unten): https://kuongshun-ks.com/uno/uno-sensor/hb100-microwave-doppler-radar-wireless-module-moti.html

Eine Anleitung zur Bare-Metal-Programmierung

Teil 1: STM32 und andere Controller

Von Sergey Lyubka (Irland)

Möchten Sie Mikrocontroller auf ihrer untersten Ebene programmieren, wo Sie ein tieferes Verständnis dafür bekommen können, wie sie tatsächlich funktionieren? Diese Anleitung für Entwickler hilft Ihnen bei den ersten Schritten, auf denen nur der GCC-Compiler und ein Referenzhandbuch verwendet werden. Die hier gelernten Grundlagen werden Ihnen helfen, besser zu verstehen, wie Frameworks wie Cube, Keil und Arduino ihre Arbeit verrichten. In dieser zweiteiligen Anleitung verwenden wir den STM32F429-Controller auf einem Nucleo-F429ZI-Board, aber was Sie hier lernen, kann leicht auch bei anderen Mikrocontrollern genutzt werden.

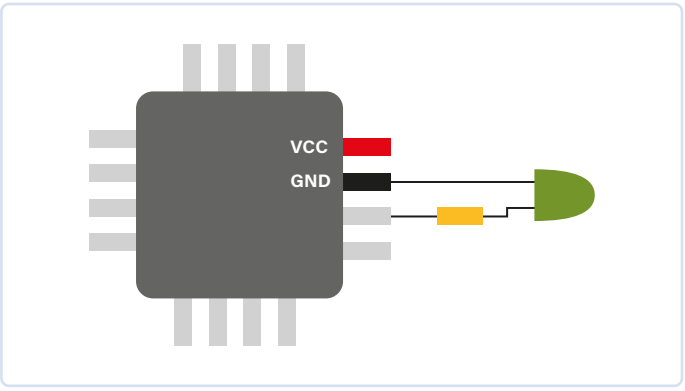


Bild 1. Der Firmware-Code kann eine hohe oder niedrige Spannung an einem Signal-Pin setzen, so dass eine LED blinkt.

Ein Mikrocontroller (µC oder MCU) ist ein kleiner Computer. Er verfügt in der Regel über eine CPU, einen Arbeitsspeicher (RAM), einen Flash-Speicher zum Speichern des Firmware-Codes und eine Reihe von Anschlüssen, auf die man von außen zugreifen kann. Einige dieser Anschlüsse sind für die Stromversorgung der MCU vorgesehen und in der Regel als GND (Masse) und VCC gekennzeichnet. Andere Pins werden zur Kommunikation mit der MCU verwendet, wobei eine hohe oder niedrige Spannung an diese Anschlüsse gelegt wird. Eines der einfachsten Kommunikationsmittel ist eine an einen Pin angeschlossene LED: Der eine LED-Kontakt ist mit dem Masse-Pin (GND) verbunden, der andere Kontakt über einen Strombegrenzungswiderstand mit einem Signal-Pin der MCU. Der Firmware-Code kann eine hohe oder niedrige Spannung an diesen Signal-Pin einstellen, wodurch die LED blinkt (**Bild 1**).

Erforderliche Hardware und Softwaretools

In dieser Anleitung verwenden wir ein Nucleo-F429ZI Entwicklungsboard (erhältlich bei Mouser und anderen Händlern). Um diesem Tutorial zu folgen, laden Sie das Referenzhandbuch der STM32F429-MCU [1] und dann das Benutzerhandbuch für das Entwicklungsboard [2] herunter.

Weiterhin werden die folgendenSoftwaretools benötigt:
ARM GCC, https://launchpad.net/gcc-arm-embedded - zum Kompilieren und Linken
GNU make, https://gnu.org/software/make - für die Build-Automatisierung
ST link, https://github.com/stlink-org/stlink - zum Flashen

Wir zeigen hier die Installationsanweisungen für Linux (Ubuntu).

Starten Sie ein Terminal, dann führen Sie aus:

```
$ sudo apt -y update
$ sudo apt -y install gcc-arm-none-eabi make
      stlink-tools
```

Um die Tools auf einem Mac oder Windows-PC einzurichten, siehe [3].

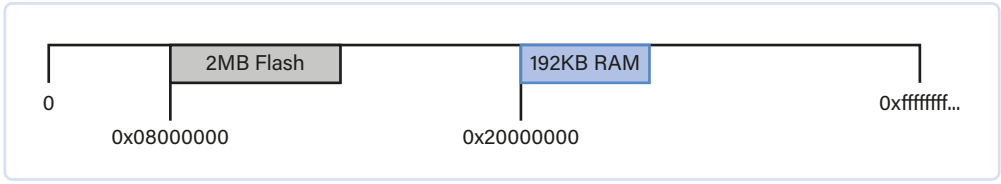


Bild 2. Flash- und RAM-Speicherbereiche des STM32F429.

Speicher und Register

Der Adressraum einer 32-Bit-MCU wie dem STM32F429 von STMicroelectronics ist in Bereiche unterteilt. Ein Speicherbereich (an einer bestimmten Adresse) ist beispielsweise auf den internen MCU-Flashspeicher abgebildet. Firmware-Anweisungen werden aus diesem Speicherbereich gelesen und ausgeführt. Ein weiterer Bereich ist das RAM, das ebenfalls einer bestimmten Adresse zugeordnet ist. Wir können beliebige Werte im RAM-Bereich lesen und dorthin schreiben.

Im Referenzhandbuch des STM32F429-[1] können wir in Abschnitt 2.3.1 nachsehen, dass der RAM-Bereich bei Adresse 0x20000000 beginnt und 192 KB groß ist. Aus Abschnitt 2.4 erfahren wir, dass der Flash-Speicher an der Adresse 0x08000000 abgebildet wird. Unsere MCU hat 2 MB Flash, also sind die Flash- und RAM-Bereiche wie in **Bild 2** angeordnet.

Aus dem Handbuch erfahren wir auch, dass es noch viele weitere Speicherbereiche gibt. Ihre Adressbereiche sind im Abschnitt 2.3, „Memory map" angegeben. So gibt es zum Beispiel einen „GPIOA"-Bereich, der bei 0x40020000 beginnt und 1 KB groß ist. Diese Speicherbereiche entsprechen verschiedenen „Peripheriegeräten" innerhalb der MCU. Das sind Teile der Siliziumschaltung, die bestimmte Pins auf eine besondere Weise reagieren lässt. Ein peripherer Speicherbereich ist eine Ansammlung von 32-Bit-Registern. Jedes Register ist ein 4-Byte-Speicherbereich an einer bestimmten Adresse, die einer bestimmten Funktion der jeweiligen Peripherie zugeordnet ist. Durch das Schreiben von Werten in ein Register - mit anderen Worten, durch das Schreiben eines 32-Bit-Wertes an eine bestimmte Speicheradresse - können wir steuern, wie sich ein bestimmtes Peripheriegerät verhalten soll. Durch das Lesen dieser Register können wir die Daten oder die Konfiguration eines Peripheriegeräts zurückerhalten.

Es gibt viele verschiedene Peripheriegeräte. Eines der einfacheren ist GPIO (General Purpose Input Output), das es dem User ermöglicht, MCU-Pins in einen „Ausgabemodus" zu versetzen und eine hohe oder niedrige Spannung an ihnen einzustellen. Oder er kann für Pins in einen „Eingangsmodus" einstellen und Spannungswerte von ihnen lesen. Es gibt einen UART-Peripheriebaustein, der Daten gemäß eines seriellen Protokolls über zwei Pins senden und empfangen kann.

Oft gibt es mehrere „Instanzen" desselben Peripheriegeräts, zum Beispiel *GPIOA*, *GPIOB* und so weiter, die verschiedene Gruppen von MCU-Pins steuern. Ebenso kann es *UART1*, *UART2* und so weiter geben, die die Implementierung mehrerer UART-Kanäle ermöglichen. Auf dem STM32F429 gibt es mehrere GPIO- und UART-Peripherieinstanzen.

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

pin 15 … pin 8

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

pin 7 … pin 3 … pin 2 … pin 1 … pin 0 (A0, B0, …)

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

Bild 3. Die Beschreibung der GPIO-Register finden Sie im Referenzhandbuch. Ein MODER-Register steuert 16 physikalische Pins. (Quelle: [1])

Die GPIOA-Peripherie zum Beispiel beginnt bei 0x40020000, und die Beschreibung der GPIO-Register finden Sie in Abschnitt 8.4 [1]. Das Referenzhandbuch sagt, dass das `GPIOA_MODER`-Register den Offset 0 hat, was bedeutet, dass seine Adresse `0x40020000 + 0` ist. In **Bild 3** können wir das Format des Registers sehen.
Das Handbuch zeigt, dass das 32-Bit-*MODER*-Register eine 16 Bit breite Sammlung von 2-Bit-Werten ist. Ein `MODER`-Register steuert 16 physische Pins: Die Bits 0...1 steuern Pin 0, Bits 2...3 steuern Pin 1 und so weiter. Der 2-Bit-Wert kodiert den Pin-Modus: 0 bedeutet Eingang, 1 bedeutet Ausgang, 2 bedeutet „alternative Funktion" - ein bestimmtes Verhalten, das an anderer Stelle beschrieben wird, und 3 bedeutet analog. Da der Name des Peripheriegeräts *GPIOA* lautet, werden die Pins als „A0", „A1" ... „Ax" bezeichnet. Für die *GPIOB*-Peripherie würde die Pin-Bezeichnung „B0", „B1" ... lauten. Wenn wir den 32-Bit-Wert 0 in das *MODER*-Register schreiben, versetzen wir alle 16 Pins von A0 bis A15 in den Eingangsmodus:

```
* (volatile uint32_t *) (0x40020000 + 0) = 0;
      // Set A0...A15 to input mode
```

Beachten Sie Qualifizierer `volatine`, dessen Bedeutung später behandelt wird. Durch das Setzen einzelner Bits können wir bestimmte Pins selektiv auf einen gewünschten Modus einstellen. Folgendes Snippet setzt zum Beispiel Pin A3 in den Ausgabemodus:

```
* (volatile uint32_t *) (0x40020000 + 0) &= ~(3 << 6);
  // Clear bit range 6...7
* (volatile uint32_t *) (0x40020000 + 0) |= 1 << 6;
  // Set bit range 6...7 to 1
```

Lassen Sie mich diese Bitoperationen erklären. Ziel ist es, die Bits 6...7, die für Pin 3 der GPIOA-Peripherie zuständig sind, auf einen bestimmten Wert (in unserem Fall 1) zu setzen. Dies geschieht in zwei Schritten. Zunächst muss der aktuelle Wert der Bits 6...7 gelöscht werden, da sie ja bereits einen Wert enthalten könnten. Dann müssen wir die entsprechenden Bits von 6...7 setzen, um den gewünschten Wert zu erhalten.
Zuerst müssen wir also den Bitbereich 6...7 (zwei Bits an Position 6) auf 0 setzen. Wie man eine Anzahl von Bits auf 0 setzt, zeigen die vier Schritte in **Tabelle 1.**

Tabelle 1. Setzen bestimmter Bits auf 0

Aktion	Ausdruck	Bits (erste 12 von 32)
Holt eine Zahl, bei der N zusammenhängende Bits gesetzt sind: 2^N-1, hier mit N=2	`3`	`000000000011`
Verschiebe die Zahl um X Stellen nach links	`(3<<6)`	`000011000000`
Invertiere die Zahl: Verwandle Nullen in Einsen und Einsen in Nullen	`~(3<<6)`	`111100111111`
Logisches UND mit bestehendem Wert	`VAL &= ~(3<<6)`	`xxxx00xxxxxx`

Beachten Sie, dass die letzte Operation, das logische UND, N Bits an der Position X auf 0 setzt (weil sie mit 0 UND-verknüpft werden), aber den Wert aller anderen Bits beibehält (weil sie mit 1 UND verknüpft werden). Die Beibehaltung des bestehenden Wertes ist wichtig, weil wir die Einstellungen in anderen Bitbereichen nicht ändern wollen. Wenn wir also im Allgemeinen N Bits an der Position X löschen wollen, können wir folgendes schreiben:

```
REGISTER &= ~((2^N - 1) << X);
```

Und schließlich wollen wir einen bestimmten Bitbereich auf den gewünschten Wert setzen. Wir verschieben diesen Wert um X Positionen nach links und verknüpfen ihn mit dem aktuellen Wert des gesamten Registers (um die Werte der anderen Bits zu erhalten):

```
REGISTER |= VALUE << X;
```

Von Menschen lesbare Programmierung von Peripheriegeräten

Im vorigen Abschnitt haben wir gelernt, dass wir ein Peripherieregister lesen und schreiben können, indem wir direkt auf bestimmte Speicheradressen zugreifen. Schauen wir uns das Snippet an, das Pin A3 in den Ausgabemodus versetzt:

```
* (volatile uint32_t *) (0x40020000 + 0) &= ~(3 << 6);
// Clear bit range 6...7
* (volatile uint32_t *) (0x40020000 + 0) |= 1 << 6;
// Set bit range 6...7 to 1
```

Das ist ziemlich kryptisch. Ohne ausführliche Kommentare wäre ein solcher Code ziemlich schwer zu verstehen. Wir können diesen Code in eine viel besser lesbare Form umschreiben. Die Idee dabei ist, das gesamte Peripheriegerät als eine Struktur darzustellen, die 32-Bit-Felder enthält. Schauen wir uns an, welche Register für die GPIO-Peripherie in Abschnitt 8.4 des Referenzhandbuchs vorhanden sind. Es sind *MODER, OTYPER, OSPEEDR, PUPDR, IDR, ODR, BSRR, LCKR, AFR*. Ihre Offsets sind 0, 4, 8, und so weiter. Das bedeutet, dass wir sie als eine Struktur mit 32-Bit-Feldern darstellen und ein `#define` für GPIOA erstellen können:

```
struct gpio {
  volatile uint32_t MODER, OTYPER, OSPEEDR,
       PUPDR, IDR, ODR, BSRR, LCKR, AFR[2];
};
#define GPIOA ((struct gpio *) 0x40020000)
```

Dann können wir für die Einstellung des GPIO-Pin-Modus eine Funktion definieren:

```
// Enum values are per reference manual: 0, 1, 2, 3
enum {GPIO_MODE_INPUT, GPIO_MODE_OUTPUT,
GPIO_MODE_AF, GPIO_MODE_ANALOG};

static inline void gpio_set_mode
  (struct gpio *gpio, uint8_t pin, uint8_t mode) {
  gpio->MODER &= ~(3U << (pin * 2));
```

```
  // Clear existing setting
  gpio->MODER |= (mode & 3) << (pin * 2);
  // Set new mode
}
```

Jetzt können wir das Snippet für A3 wie folgt umschreiben:

```
gpio_set_mode(GPIOA, 3 /* pin */, GPIO_MODE_OUTPUT);
  // Set A3 to output
```

Unsere MCU hat mehrere GPIO-Peripheriegeräte (auch „Bänke“ genannt): A, B, C, … K. Aus Abschnitt 2.3 wissen wir, dass sie 1 KB voneinander entfernt sind: GPIOA befindet sich an der Adresse 0x40020000, GPIOB an 0x40020400, und so weiter:

```
#define GPIO(bank) ((struct gpio *)
  (0x40020000 + 0x400 * (bank)))
```

Wir können eine Pin-Nummerierung definieren, die die Bank und die Pin-Nummer enthält. Dazu verwenden wir einen 2-Byte-Wert `uint16_t`, wobei das obere Byte die GPIO-Bank und das untere Byte die Pin-Nummer angibt:

```
#define PIN(bank, num) ((((bank) - 'A') << 8) | (num))
#define PINNO(pin) (pin & 255)
#define PINBANK(pin) (pin >> 8)
```

Auf diese Weise können wir Pins für jede GPIO-Bank festlegen:

```
uint16_t pin1 = PIN('A', 3); // A3 - GPIOA pin 3
uint16_t pin2 = PIN('G', 11); // G11 - GPIOG pin 11
```

Schreiben wir die Funktion `gpio_set_mode()` neu, um unsere Pin-Spezifikation zu übernehmen:

```
static inline void gpio_set_mode(uint16_t pin, uint8_t
mode) {
  struct gpio *gpio = GPIO(PINBANK(pin));
      // GPIO bank
  uint8_t n = PINNO(pin); // Pin number
  gpio->MODER &= ~(3U << (n * 2));
      // Clear existing setting
  gpio->MODER |= (mode & 3) << (n * 2);
      // Set new mode
}
```

Der Code für A3 ist nun selbsterklärend:

```
uint16_t pin = PIN('A', 3); // Pin A3
gpio_set_mode(pin, GPIO_MODE_OUTPUT); // Set to output
```

Wir haben hiermit eine nützliche erste API für die GPIO-Peripherie geschaffen. Andere Peripheriegeräte wie der UART für serielle Kommunikation und andere können auf ähnliche Weise implementiert werden. Dies ist eine gute Programmierpraxis, da es den Code selbsterklärend und für den Menschen lesbar macht.

MCU-Boot und Vektortabelle

Wenn eine ARM-MCU bootet, liest sie eine sogenannte „Vektortabelle“ vom Anfang des Flash-Speichers. Eine Vektortabelle ist ein gemeinsames Konzept für alle ARM-MCUs, ein Array von 32-Bit-Adressen von Interrupt-Handlern. Die ersten 16 Einträge sind von ARM reserviert und gelten für alle ARM-MCUs. Die restlichen Interrupt-Handler sind spezifisch für die jeweilige MCU, und dabei handelt es sich um Interrupt-Handler für Peripheriegeräte. Einfachere MCUs mit wenigen Peripheriegeräten haben wenige Interrupt-Handler, komplexere MCUs haben viele.
Die Vektortabelle für den STM32F429 ist in Tabelle 62 des Referenzhandbuchs [1] dokumentiert. Dort erfahren wir, dass es zusätzlich zu den 16 Standard-Handlern 91 Peripherie-Handler gibt.
Jeder Eintrag in der Vektortabelle ist eine Adresse einer Funktion, die die MCU ausführt, wenn ein Hardware-Interrupt (IRQ) ausgelöst wird. Eine Ausnahme sind die ersten beiden Einträge, die eine Schlüsselrolle im Boot-Prozess der MCU spielen. Bei diesen beiden ersten Werten handelt es sich um einen initialen Stapel-Pointer und die Adresse der Boot-Funktion (ein auszuführender Einstiegspunkt der Firmware).
Wir wissen nun, dass unsere Firmware so aufgebaut sein muss, dass der zweite 32-Bit-Wert im Flash die Adresse unserer Boot-Funktion enthält. Wenn die MCU bootet, liest sie diese Adresse aus dem Flash und springt zu unserer Boot-Funktion.

Minimale Firmware

Erstellen wir eine Datei *main.c* und spezifizieren unsere Boot-Funktion, die zunächst nichts tut (in eine Endlosschleife fällt), und zusätzlich eine Vektortabelle, die 16 Standardeinträge und 91 STM32-Einträge enthält. Erstellen und öffnen Sie in einem Editor Ihrer Wahl die Datei *main.c* und geben Sie Folgendes ein:

```
// Startup code
__attribute__((naked, noreturn)) void _reset(void) {
  for (;;) (void) 0; // Infinite loop
}

extern void _estack(void); // Defined in link.ld

// 16 standard and 91 STM32-specific handlers
__attribute__((section(".vectors")))
      void (*tab[16 + 91])(void) = {_estack, _reset};
```

Für die Funktion `_reset()` haben wir die GCC-spezifischen Attribute `naked` und `noreturn` verwendet, was bedeutet, dass die Standardfunktionen `prologue` und `epilogue` vom Compiler nicht erstellt werden sollen und dass die Funktion nicht zurückkehrt.
Der Ausdruck `void (*tab[16 + 91])(void)` bedeutet: Definiere ein Array von 16 + 91 Pointern auf Funktionen, die nichts zurückgeben `(void)` und zwei Argumente annehmen. Jede solche Funktion ist ein IRQ-Handler (Interrupt ReQuest Handler). Ein Array dieser Handler wird als Vektortabelle bezeichnet.
Die Vektortabelle `tab` legen wir in einem separaten Abschnitt namens `.vectors` ab, weil wir ihn später brauchen, um dem Linker mitzuteilen, dass er diesen Abschnitt direkt an den Anfang der generierten Firmware setzen soll, und zwar fortlaufend, am Anfang

des Flash-Speichers. Die ersten beiden Einträge sind der Wert des Stack-Pointer-Registers und der Einstiegspunkt der Firmware. Den Rest der Vektortabelle lassen wir mit Nullen gefüllt.

Kompilierung

Lassen Sie uns unseren Code kompilieren. Starten Sie ein Terminal (oder einen Eingabe-Prompt in Windows) und führen Sie aus:

```
$ arm-none-eabi-gcc -mcpu=cortex-m4 main.c -c
```

Das funktioniert! Die Kompilierung hat eine Datei *main.o* erzeugt, die unsere minimale Firmware enthält, die nichts tut. Die Datei *main.o* liegt im ELF-Binärformat vor und enthält mehrere Abschnitte. Sehen wir uns diese in **Listing 1** an.
Die VMA/LMA-Adressen für die Abschnitte sind auf 0 gesetzt, was bedeutet, dass *main.o* noch keine vollständige Firmware ist, da sie keine Informationen darüber enthält, wo diese Abschnitte in den Adressraum geladen werden sollen. Wir müssen einen Linker verwenden, um aus *main.o* die vollständige Firmware-Datei *firmware.elf* zu erzeugen.
Der Abschnitt `.text` enthält Firmware-Code, in unserem Fall nur eine `_reset()`-Funktion, die zwei Bytes lang ist - eine Sprunganweisung zu ihrer eigenen Adresse. Es gibt weder einen leeren `.data`-Abschnitt noch einen leeren `.bss`-Abschnitt [8] (für nicht initialisierte, aber deklarierte Variablen wird dieser Abschnitt normalerweise mit Nullen gefüllt). Unsere Firmware wird in den Flash-Bereich an Offset 0x8000000 kopiert, aber unser Datenbereich sollte sich im RAM befinden - daher sollte unsere `_reset()`-Funktion den Inhalt des `.data`-Bereichs ins RAM kopieren. Außerdem muss sie Nullen in den gesamten `.bss`-Abschnitt schreiben. In unserem Fall sind die Abschnitte `.data` und `.bss` leer, aber wir müssen unsere `_reset()`-Funktion trotzdem modifizieren, um sie richtig anzuwenden.
Dazu müssen wir wissen, wo der Stack beginnt, und wo die Abschnitte data und bss beginnen. Dies können wir im *Linker*-Skript angeben, einer Datei mit Anweisungen an den Linker, wo verschiedene Abschnitte im Adressraum zu platzieren und welche Symbole zu erstellen sind.

Linker-Skript

Erstellen Sie eine Datei namens `link.ld` und kopieren Sie den Inhalt aus [4] hinein. Unten finden Sie die Erklärung:

```
ENTRY(_reset);
```

Diese Zeile teilt dem Linker den Wert des „entry point"-Attributs im generierten ELF-Header mit - dies ist also ein Duplikat dessen, was eine Vektortabelle enthält. Dies ist ein Hilfsmittel für einen Debugger (wie Ozone, beschrieben im zweiten Teil dieser Artikelreihe), das uns hilft, einen Haltepunkt am Anfang der Firmware zu setzen. Ein Debugger kennt keine Vektortabelle, daher ist er auf den ELF-Header angewiesen.

```
MEMORY {
flash(rx) : ORIGIN = 0x08000000, LENGTH = 2048k
sram(rwx) : ORIGIN = 0x20000000, LENGTH = 192k
       /* remaining 64k in a separate address space */
}
```

Diese Zeilen teilen dem Linker mit, dass wir zwei Speicherbereiche sowie deren Adressen und Größen im Adressraum haben.

```
_estack = ORIGIN(sram) + LENGTH(sram);
                    /* stack points to end of SRAM */
```

Dies weist den Linker an, ein Symbol `_estack` zu erstellen, das einen Wert am Ende des RAM-Speicherbereichs enthält. Das wird unser anfänglicher Stack-Wert sein!

```
.vectors : { KEEP(*(.vectors)) } > flash
.text : { *(.text*) } > flash
.rodata : { *(.rodata*) } > flash
```

Diese Zeilen weisen den Linker an, zuerst die Vektortabelle im Flash abzulegen, gefolgt von der `.text`-Sektion (Firmware-Code), gefolgt von den Nur-Lese-Daten `.rodata`.
Als nächstes kommt die `.data`–Sektion:

Listing 1: Kompilieren von main.o

```
$ arm-none-eabi-objdump -h main.o
 ...
 Idx Name          Size      VMA       LMA       File off  Algn
   0 .text         00000002  00000000  00000000  00000034  2**1
                   CONTENTS, ALLOC, LOAD, READONLY, CODE
   1 .data         00000000  00000000  00000000  00000036  2**0
                   CONTENTS, ALLOC, LOAD, DATA
   2 .bss          00000000  00000000  00000000  00000036  2**0
                   ALLOC
   3 .vectors      000001ac  00000000  00000000  00000038  2**2
                   CONTENTS, ALLOC, LOAD, RELOC, DATA
 ...
```

Listing 2: Startup-Code

```
int main(void) {
   return 0; // Do nothing so far
}

// Startup code
__attribute__((naked, noreturn)) void _reset(void) {
   // memset .bss to zero, and copy .data section to RAM region
   extern long _sbss, _ebss, _sdata, _edata, _sidata;
   for (long *src = &_sbss; src < &_ebss; src++) *src = 0;
   for (long *src = &_sdata, *dst = &_sidata; src < &_edata;) *src++ = *dst++;

   main();             // Call main()
   for (;;) (void) 0;  // Infinite loop in the case if main() returns
 }
```

```
.data : {
  _sdata = .; /* .data section start */
  *(.first_data)
  *(.data SORT(.data.*))
  _edata = .; /* .data section end */
} > sram AT > flash
_sidata = LOADADDR(.data);
```

Wir weisen den Linker an, die Symbole `_sdata` und `_edata` zu erstellen, die wir verwenden, um den Datenbereich in der `_reset()`-Funktion in das RAM zu kopieren.
Dasselbe gilt für den `.bss`-Abschnitt:

```
.bss : {
  _sbss = .; /* .bss section start */
  *(.bss SORT(.bss.*) COMMON)
  _ebss = .; /* .bss section end */
} > sram
```

Startup-Code

Jetzt können wir unsere Funktion `_reset()` aktualisieren. Wir kopieren den `.data`-Abschnitt in das RAM und initialisieren den `.bss`-Abschnitt mit Nullen. Dann rufen wir die Funktion `main()` auf - und fallen in eine Endlosschleife, wenn `main()` zurückkehrt; siehe **Listing 2**. Das Diagramm in **Bild 4** zeigt, wie `_reset()` sowohl `.data` als auch `.bss` initialisiert.

Die Datei *firmware.bin* ist lediglich eine Verkettung der drei Abschnitte `.vectors` (IRQ-Vektortabelle), `.text` (Code) und `.data` (Daten). Diese Abschnitte wurden gemäß dem Linker-Skript aufgebaut: `.vectors` liegt ganz am Anfang des Flash-Speichers, `.text` folgt unmittelbar danach, und `.data` liegt weit oben. Die Adressen in `.text` liegen im Flash-Speicherbereich, die Adressen in `.data` im RAM-Bereich. Wenn eine Funktion eine Adresse hat, zum Beispiel `0x8000100`, dann befindet sie sich genau an dieser Adresse im Flash. Wenn der Code jedoch auf eine Variable im `.data`-Bereich über die Adresse zugreift, beispielsweise `0x20000200`, dann befindet sich an dieser Adresse nichts, denn beim Booten befindet sich der `.data`-Bereich in der Datei *firmware.bin* im Flash! Deshalb muss der Startup-Code den `.data`-Abschnitt aus dem Flash-Speicherbereich in den RAM-Bereich verschieben.
Jetzt sind wir so weit, um die vollständige Firmware-Datei *firmware.elf* zu erzeugen:

```
$ arm-none-eabi-gcc -T link.ld -nostdlib main.o -o
firmware.elf
```

Schauen wir uns in **Listing 3** die Abschnitte in *firmware.elf* an. Wir können sehen, dass der Abschnitt `.vectors` ganz am Anfang des Flash-Speichers an der Adresse 0x8000000 liegt, und der Abschnitt `.text` direkt danach an 0x80001ac. Unser Code erzeugt keine Variablen, daher gibt es auch keinen Datenbereich.

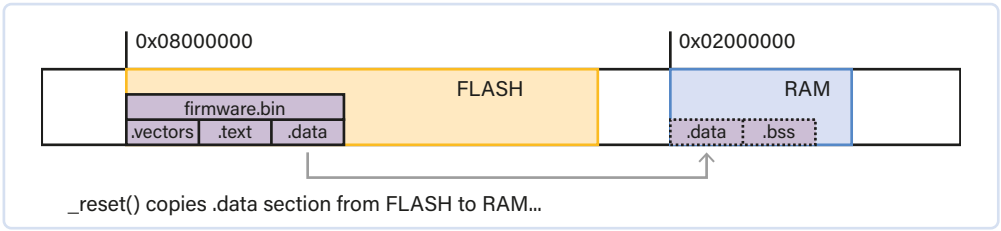


Bild 4. Das Diagramm veranschaulicht, wie `_reset()` `.data` und `.bss` initialisiert.

Listing 3: Bereiche in firmware.elf

```
$ arm-none-eabi-objdump -h firmware.elf
 ...
 Idx Name          Size      VMA       LMA       File off  Algn
   0 .vectors      000001ac  08000000  08000000  00010000  2**2
                   CONTENTS, ALLOC, LOAD, DATA
   1 .text         00000058  080001ac  080001ac  000101ac  2**2
                   CONTENTS, ALLOC, LOAD, READONLY, CODE
 ...
```

Firmware flashen

Nun können wir die Firmware zu flashen! Zuerst extrahieren wir die Abschnitte aus *firmware.elf* in einen einzelnen, zusammenhängenden Binärblob:

```
$ arm-none-eabi-objcopy -O binary firmware.elf firmware.bin
```

Schließen Sie Ihr Board an den USB an und verwenden Sie das Dienstprogramm *st-link*, um *firmware.bin* zu flashen:

```
$ st-flash --reset write firmware.bin 0x8000000
```

Geschafft! Wir haben eine Firmware geflasht, die nichts tut.

Build-Automatisierung mit Makefile

Anstatt all diese Befehle zum Kompilieren, Linken und Flashen einzugeben, können wir das Kommandozeilenwerkzeug `make` verwenden, um den gesamten Prozess zu automatisieren. Das Dienstprogramm `make` verwendet eine Konfigurationsdatei namens Makefile, aus der es Anweisungen für die Ausführung von Aktionen liest. Diese Automatisierung ist großartig, weil sie auch den Prozess der Firmware-Erstellung, die verwendeten Kompilierungsflags und so weiter dokumentiert.
Es gibt für alle, die neu in `make` sind, ein großartiges Makefile-Tutorial [5]. Im Folgenden führe ich die wichtigsten Konzepte auf, die zum Verständnis unseres einfachen Bare-Metal-`Makefiles` erforderlich sind. Wer mit `make` bereits vertraut ist, kann ruhigen Gewissens diesen Abschnitt überspringen. Das `Makefile`-Format ist einfach:

```
action1:
    command ... # Comments can go after hash symbol
    command .... # IMPORTANT: command must be preceded
                 with the TAB character
action2:
    command ... # Don't forget about TAB. Spaces won't
                work!
```

Jetzt können wir `make` mit dem Namen der Aktion (auch *target* genannt) aufrufen, um eine entsprechende Aktion auszuführen:

```
$ make action1
```

Es ist möglich, Variablen zu definieren und sie in Befehlen zu verwenden. Außerdem können Aktionen die Namen von Dateien sein, die erstellt werden müssen:

```
firmware.elf:
    COMPILATION COMMAND .....
```

Außerdem kann jede Aktion eine Liste von Abhängigkeiten haben. Zum Beispiel hängt *firmware.elf* von unserer Quelldatei *main.c* ab. Immer wenn sich die Datei *main.c* ändert, wird *firmware.elf* mit dem Befehl `make build` neu erstellt:

```
build: firmware.elf
firmware.elf: main.c
    COMPILATION COMMAND
```

Schreiben wir nun also ein *Makefile* für unsere Firmware. Wir definieren eine `build`-Aktion/Target:

```
CFLAGS ?= -W -Wall -Wextra -Werror -Wundef -Wshadow
-Wdouble-promotion \ -Wformat-truncation -fno-common
-Wconversion \ -g3 -Os -ffunction-sections -fdata-sec-
tions -I. \ -mcpu=cortex-m4 -mthumb -mfloat-abi=hard
-mfpu=fpv4-sp-d16 $(EXTRA_CFLAGS)
LDFLAGS ?= -Tlink.ld -nostartfiles -nostdlib --specs nano.
specs -lc -lgcc -Wl,--gc-sections -Wl,-Map=$@.map
SOURCES = main.c
build: firmware.elf
firmware.elf: $(SOURCES)
    arm-none-eabi-gcc $(SOURCES) $(CFLAGS)     $(LDFLAGS)
-o $@
```

Hier definieren wir Flags zum Kompilieren. Das `?=` steht für einen Standardwert, der auf der Kommandozeile wie folgt überschrieben werden kann:

```
$ make build CFLAGS="-O2 ...."
```

Wir geben die Variablen `CFLAGS`, `LDFLAGS` und `SOURCES` an. Dann teilen wir `make` mit: Wenn dir gesagt wird, dass du „builden" sollst, dann erstelle eine Datei *firmware.elf*. Sie hängt von der Datei *main.c* ab, und um sie zu erstellen, starte den Compiler `arm-none-eabi-gcc` mit den angegebenen Flags. Die spezielle Variable `$@` expandiert zu einem Zielnamen - in unserem Fall *firmware.elf*.
Rufen wir nun `make` auf:

```
$ make build arm-none-eabi-gcc main.c -W -Wall -Wextra
-Werror -Wundef -Wshadow -Wdouble-promotion -Wformat-trun-
cation -fno-common -Wconversion -g3 -Os -ffunction-sec-
tions -fdata-sections -I. -mcpu=cortex-m4 -mthumb -mfloat-
abi=hard -mfpu=fpv4-sp-d16 -Tlink.ld -nostartfiles -nostd-
lib --specs nano.specs -lc -lgcc -Wl,--gc-sections
-Wl,-Map=firmware.elf.map -o firmware.elf
```

Wir führen es erneut aus:

Listing 4: Ausschnitt der main.c-Datei Blinky LED

```
#include <inttypes.h>
#include <stdbool.h>
#define BIT(x) (1UL << (x))
#define PIN(bank, num) ((((bank) - 'A') << 8) | (num))
#define PINNO(pin) (pin & 255)
#define PINBANK(pin) (pin >> 8)

struct gpio {
   volatile uint32_t MODER, OTYPER, OSPEEDR, PUPDR, IDR, ODR, BSRR, LCKR, AFR[2];
};
#define GPIO(bank) ((struct gpio *) (0x40020000 + 0x400 * (bank)))

// Enum values are per datasheet: 0, 1, 2, 3
enum { GPIO_MODE_INPUT, GPIO_MODE_OUTPUT, GPIO_MODE_AF, GPIO_MODE_ANALOG };

static inline void gpio_set_mode(uint16_t pin, uint8_t mode) {
   struct gpio *gpio = GPIO(PINBANK(pin));  // GPIO bank
   int n = PINNO(pin);                      // Pin number
   gpio->MODER &= ~(3U << (n * 2));         // Clear existing setting
   gpio->MODER |= (mode & 3) << (n * 2);    // Set new mode
}
```

```
$ make build
make: Nothing to be done for 'build'.
```

Das Dienstprogramm `make` prüft die Änderungen bei der Abhängigkeit *main.c* und *firmware.elf* - und unternimmt nichts, wenn *firmware.elf* auf dem neuesten Stand ist. Aber wenn wir *main.c* ändern, kompiliert das nächste `make build` neu:

```
$ touch main.c # Simulate changes in main.c
$ make build
```

Jetzt ist nur noch das `flash`-Target übrig:

```
firmware.bin: firmware.elf
    $(DOCKER) $(CROSS)-objcopy -O binary $< $@ flash:
firmware.bin
    st-flash --reset write $(TARGET).bin 0x8000000
```

Das war's! Jetzt erstellt der Terminalbefehl `make flash` eine Datei *firmware.bin* und flasht sie auf das Board. Die Firmware wird neu kompiliert, wenn *main.c* geändert wird, weil *firmware.bin* von *firmware.elf* abhängt, und diese wiederum von *main.c*. Der Entwicklungszyklus würde also aus diesen beiden Aktionen in einer Schleife bestehen:

```
# Develop code in main.c
$ make flash
```

Es ist eine gute Idee, einen Clean-Befehl hinzuzufügen, um Build-Artefakte zu entfernen:

```
clean:
    rm -rf firmware.*
```

Den vollständigen Quellcode des Projekts finden Sie im Ordner *Step 0 minimal* [6].

Blinkende LED

Nun, da wir die gesamte Build-/Flash-Infrastruktur eingerichtet haben, können wir unserer Firmware auch etwas Nützliches beizubringen. Etwas sehr Nützliches ist natürlich das Blinken einer LED ;-) Ein Nucleo-F429ZI Board besitzt drei LEDs. Im Nucleo-Board-Benutzerhandbuch [2] wird in Abschnitt 6.5 gezeigt, an welchen Pins diese LEDs angeschlossen sind:

- PB0: grüne LED
- PB7: blaue LED
- PB14: rote LED

Ändern wir nun die Datei *main.c* und fügen unsere Definitionen für PIN `gpio_set_mode()` hinzu. In der Funktion `main()` setzen wir PB7 mit der blauen LED in den Ausgabemodus und starten eine Endlosschleife. Kopieren wir zunächst die Definitionen für die Pins und GPIO, die wir zuvor besprochen haben. Fügen wir auch zur Bequemlichkeit das Makro `BIT(x)` hinzu - siehe **Listing 4**.
Bei einigen Mikrocontrollern werden alle Peripheriegeräte automatisch mit Strom versorgt und aktiviert, wenn sie eingeschaltet werden. Bei STM32-MCUs sind die Peripheriegeräte jedoch standardmäßig deaktiviert, um Strom zu sparen. Um ein GPIO-Peripheriegerät zu aktivieren, muss es über die RCC-Einheit (Reset and Clock Control) aktiviert (getaktet) werden. Im Referenzhandbuch-Abschnitt 7.3.10 erfahren wir, dass das *AHB1ENR* (AHB1 peripheral clock enable register) für das Ein- und Ausschalten von GPIO-Bänken verantwortlich ist. Zunächst fügen wir eine Definition für die gesamte RCC-Einheit hinzu:

```
struct rcc {
    volatile uint32_t CR, PLLCFGR, CFGR, CIR, AHB1RSTR,
AHB2RSTR, AHB3RSTR, RESERVED0, APB1RSTR, APB2RSTR, RESER-
VED1[2], AHB1ENR, AHB2ENR, AHB3ENR, RESERVED2, APB1ENR,
APB2ENR, RESERVED3[2], AHB1LPENR, AHB2LPENR, AHB3LPENR,
RESERVED4, APB1LPENR, APB2LPENR, RESERVED5[2], BDCR, CSR,
RESERVED6[2], SSCGR, PLLI2SCFGR;
};
#define RCC ((struct rcc *) 0x40023800)
```

In der Dokumentation des *AHB1ENR*-Registers sehen wir, dass die Bits 0...8 den Takt für die GPIO-Bänke GPIOA-GPIOE einstellen:

```
int main(void) {
  uint16_t led = PIN('B', 7); // Blue LED
  RCC->AHB1ENR |= BIT(PINBANK(led));
  // Enable GPIO clock for LED
  gpio_set_mode(led, GPIO_MODE_OUTPUT);
  // Set blue LED to output mode
  for (;;) asm volatile("nop"); // Infinite loop
  return 0;
}
```

Jetzt müssen wir nur noch herausfinden, wie man einen GPIO-Pin ein- oder ausschaltet, und dann die Hauptschleife modifizieren, um einen LED-Pin einzuschalten, zu verzögern, auszuschalten und zu verzögern . Ein Blick in das Referenzhandbuch, Abschnitt 8.4.7, zeigt, dass das Register *BSRR* für das Setzen der Spannung auf high oder low verantwortlich ist. Die unteren 16 Bits werden verwendet, um das *ODR*-Register zu setzen (das heißt, den Pin auf High zu setzen), und die oberen 16 Bits, um das *ODR*-Register zurückzusetzen (also den Pin auf Low zu setzen). Definieren wir eine API-Funktion für diese Aufgabe:

```
static inline void gpio_write(uint16_t pin, bool val) {
  struct gpio *gpio = GPIO(PINBANK(pin));
  gpio->BSRR |= (1U << PINNO(pin)) << (val ? 0 : 16);
}
```

Als nächstes müssen wir eine Verzögerungsfunktion implementieren. Da wir im Moment keine genaue Verzögerung benötigen, definieren wir eine Funktion `spin()`, die einfach eine bestimmte Anzahl von NOPs ausführt:

```
static inline void spin(volatile uint32_t count) {
  while (count--) asm("nop");
}
```

Zum Schluss können wir unsere Hauptschleife ändern, um das Blinken der LEDs zu implementieren:

```
for (;;) {
  gpio_write(pin, true);
  spin(999999);
  gpio_write(pin, false);
  spin(999999);
}
```

Der vollständige Quellcode des Projekts befindet sich im Ordner Step 1 blinky [7]. Starten Sie `make flash` und genießen Sie das Blinken der blauen LED!

Im zweiten Teil dieses Artikels werden wir uns die UART-Ausgabe, das Debugging, eine Webserver-Implementierung, automatische Tests und mehr ansehen. Bleiben Sie dran!

RG — 220665-02

Über den Autor

Sergey Lyubka ist ein Ingenieur und Unternehmer. Er hat einen MSc in Physik von der Staatlichen Universität Kyjiw, Ukraine. Sergey ist Direktor und Mitbegründer von Cesanta, einem Technologieunternehmen mit Sitz in Dublin, Irland (Embedded Web Server for electronic devices: https://mongoose.ws). Seine Leidenschaft ist die Programmierung von eingebetteten Bare-Metal-Netzwerken.

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Senden Sie eine E-Mail an den Autor unter sergey.lyubka@cesanta.com oder kontaktieren Sie Elektor unter redaktion@elektor.de.

WEBLINKS

[1] Referenzhandbuch RM0090 für STM32F429: https://bit.ly/3neE7S7
[2] Benutzerhandbuch Nucleo-144-Board (UM1974): https://bit.ly/3oIBXKZ
[3] Artikel auf GitHub: https://github.com/cpq/bare-metal-programming-guide
[4] Inhalt der Datei link.ld: https://github.com/cpq/bare-metal-programming-guide/blob/main/step-0-minimal/link.ld
[5] Makefile-Tutorial: https://makefiletutorial.com/
[6] Demoprogramm Step 0 minimal : https://github.com/cpq/bare-metal-programming-guide/blob/main/step-0-minimal
[7] Demoprogramm Step 1 blinky: https://github.com/cpq/bare-metal-programming-guide/blob/main/step-1-blinky
[8] .bss (Wikipedia): https://en.wikipedia.org/wiki/.bss

Siglent-**Multimeter** SDM3045X

Von Philippe Demerliac (Frankreich)

Ein typisches Handmultimeter erfüllt seine Aufgabe gut und liegt immer bequem in der Hand. Ein Tischmultimeter bietet jedoch viel mehr Möglichkeiten, die das Leben im Labor viel einfacher machen. Das Tischmultimeter SDM3045X von Siglent ist ein solches Gerät, und ich stelle es hier vor.

Ich besitze bereits mehrere Geräte von Siglent, die ich aufgrund ihrer hervorragenden Qualität sehr schätze. Mir fehlte jedoch ein Tischmultimeter in der großen Palette meiner Instrumente. Jetzt, wo ich diese Lücke mit dem SDM3045X geschlossen habe, kann ich meine Eindrücke teilen.

Siglent bietet drei Multimeter an: SDM3045X, SDM3055 und SDM3065X. Diese Modelle haben ähnliche Funktionen, unterscheiden sich aber in der Auflösung. Das 3045 hat 4½ Ziffern (60.000 Werte), das 3055 hat 5½ Ziffern (240.000 Werte) und das 3065 hat 6½ Ziffern (2.200.000 Werte). 3055 und 3065 bieten auch eine bessere Leistung bei niedriger Empfindlichkeit: 200 mV / 200 µA, gegenüber 600 mV / 600 µA beim 3045. Darüber hinaus bieten die Modelle 3055 und 3065 die SC-Option, die mehrere programmierbare Messeingänge ermöglicht. Alle Modelle stecken im gleichen Gehäusetyp und weisen die gleiche Ergonomie auf, aber die Preise variieren natürlich mit der Auflösung.

Auflösung und Messgenauigkeit

Lassen Sie uns ein wenig über die Auflösung sprechen. Sie wird als Anzahl signifikanter Ziffern oder Zählwerte ausgedrückt. Der Zählwert gibt die Anzahl der unterschiedlichen Werte an, die in einem bestimmten Bereich angezeigt werden können. Im 600-mV-Bereich mit der höchsten Auflösung löst beispielsweise das 3045 Inkremente von 10 µV auf, was bei 600 mV 60.000 möglichen Werten entspricht. Jeder weitere Bereich – 6 V, 60 V, 600 V und 1000 V - bringt eine Verringerung der Auflösung mit sich. Im 1000-V-Bereich (der höchste Bereich liegt aus Sicherheitsgründen bei 1000 V und nicht bei 6000 V) beträgt die Auflösung 100 mV.

Bei Modellen mit höherer Auflösung gibt es mehr signifikante Ziffern. Achten Sie darauf, dass Sie Auflösung und Genauigkeit oder besser den Fehler nicht verwechseln. Wenn wir einen Messwert von 1,0000 V auf dem Display ablesen, sind wir dann sicher, dass die Spannung 1 V mit einem möglichen Fehler von 100 µV beträgt? Nein, denn selbst ordnungsgemäß kalibrierte Messgeräte weisen einen maximalen Fehler von 0,06 % ±8 Stellen auf. Die tatsächliche Spannung könnte also zwischen 0,9986 V und 1,0014 V liegen, wobei es sich bei diesen Grenzen um Extremwerte handelt. Bedeutet dies aber, dass die niedrigsten Werte nutzlos sind? Wieder nein, denn sie ermöglichen es, Messungen zu vergleichen und ihre Entwicklung im Laufe der Zeit zu beobachten.

Zurück zu unserem SDM3045X: Reichen 60.000 Werte aus, oder müssen wir ein anderes Modell wählen? Die Antwort auf diese Frage hängt natürlich von der Anwendung ab, aber im Allgemeinen ist diese Auflösung für einen Amateur oder sogar einen Profi mehr als genug, oft sogar überflüssig genau. Ich habe Multimeter mit niedrigeren Auflösungen verwendet und verwende sie immer noch, weil die angezeigten Werte in 90 % aller Messungen völlig ausreichend genau sind.

Sie können das Datenblatt des Modells von Siglent [1] leicht herunterladen und daraus die maximale Auflösung und den Fehler für jede Art von Messung sowie für jeden Bereich die vom Hersteller garantierte maximale Auflösung und den Fehler ablesen.

Apropos Genauigkeit: Die Multimeter werden mit einem Kalibrierungszertifikat geliefert, das beweist, dass das Gerät bei Auslieferung innerhalb der angegebenen Grenzwerte lag, oft sogar besser. Siglent sagt diesbezüglich:

Review im Video [Französisch]:
https://youtu.be/iTW9O_JqmM4

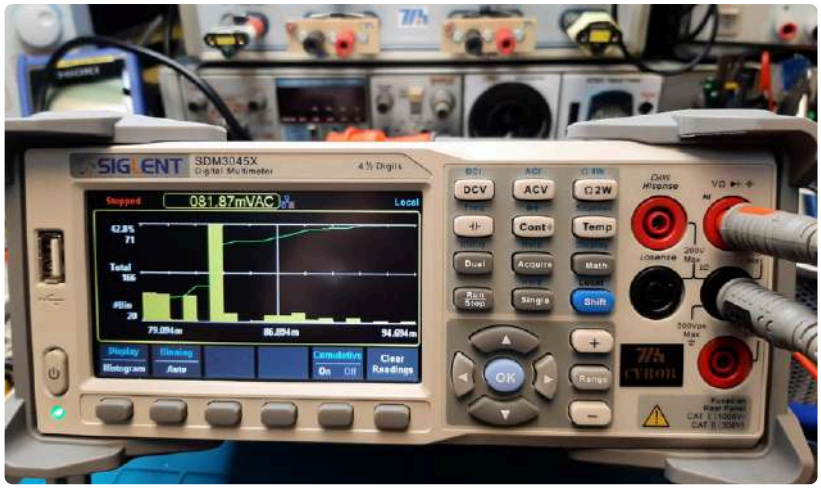
„SIGLENT hat festgestellt, dass die Werkskalibrierung dieses Geräts durch eine Lagerung von bis zu 180 Tagen vor dem ersten Gebrauch nicht wesentlich beeinträchtigt wird. Das Kalibrierungsintervall beginnt, wenn das Gerät in Betrieb genommen wird oder 180 Tage nach dem Kalibrierungsdatum auf dem dem Gerät beiliegenden Zertifikat."

Sollten diese Multimeter regelmäßig neu kalibriert werden? In einem professionellen Umfeld ist es ratsam, dies zu tun, wenn man bestimmte Standards erfüllen will, aber für einen Amateur ist eine häufige Kalibrierung nicht erforderlich. Die Erfahrung hat gezeigt, dass sich moderne Materialien mit der Zeit nicht mehr so stark ändern. Wenn Sie dazu in der Lage sind, ist es ratsam, die Genauigkeit des Geräts nur ab und zu in regelmäßigen Abständen zu überprüfen.
Es gibt zugelassene Einrichtungen (zum Beispiel Eichämter), die solche Kalibrierungen durchführen können, aber es ist ein relativ teures Vergnügen im Vergleich zum Preis des Geräts. Auch wenn Siglent in Bezug auf eine Kalibrierung nicht sehr ausführlich ist, gibt es Hacks im Internet, um dies zu tun, aber es setzt immer noch voraus, dass man die entsprechenden Referenzquellen besitzt. Zusammenfassend lässt sich sagen, dass die Kalibrierung normalerweise kein Problem darstellt und Ihr Multimeter Ihnen viele Jahre lang treue Dienste leisten wird.

Ein erster Blick auf das SDM3045X

Das SDM3045X ist ein Tischmultimeter und daher, obwohl es transportabel ist, nicht wirklich für den mobilen Einsatz im Feld gedacht. Es benötigt eine Netzspannugsversorgung und muss im Einsatz auf einen Tisch oder zumindest eine stabile Unterlage gestellt werden. Im Vergleich zu Stand-alone-Modellen bieten Tischmultimeter aber in der Regel bessere und zusätzliche Funktionen, etwa eine umfangreicherer Konnektivität, 4-Leiter-Messung, reichhaltigere Anzeige und so weiter. Außerdem kann man sie sicher direkt vor sich aufstellen, was ihre Bedienung und das Ablesen des Displays deutlich komfortabler macht, und die Netzversorgung erspart ein lästiges Wechseln der Batterien im falschen Moment oder ein regelmäßiges Aufladen von Akkus.
Das SDM3045X kann Spannungen und Ströme in DC und AC_{RMS} bis zu 100 kHz (perfekt für NF!), Widerstände (zwei Drähte oder vier Drähte für sehr niederohmige Widerstände), Durchgang, Diodentest, Kondensatoren und Temperatur mit verschiedenen Sensoren messen (es enthält eine Kaltstellenkompensation für Thermoelemente). Dies deckt die meisten aktuell üblichen Anforderungen ab. Außerdem ist es möglich, die gemessenen Werte zu speichern, Statistiken zu erstellen und Alarme im Falle von Grenzwertüberschreitungen zu setzen und anzuzeigen.
Wie andere Siglent-Geräte wird es in einer gut durchdachten Verpackung geliefert, die es beim Transport perfekt vor Stößen schützt. Ich empfehle Ihnen, sie gut aufzubewahren! Das Gerät wird mit einem Netzkabel, zwei flexiblen Messleitungen von hervorragender Qualität, einem USB-A/B-Kabel für den Anschluss an einen PC, einem Kalibrierungszertifikat und einem einfachen Benutzerhandbuch in englischer Sprache geliefert. Ausführliche Handbücher zu verschiedenen Themen können von der Siglent-Website heruntergeladen werden.
Das Gerät strahlt Qualität aus, und die Verarbeitung ist tadellos. Die Softkeys bieten eine angenehme Haptik. Die Einschalttaste ist allerdings kein „echter" Schalter, so dass sich das ausgeschaltete Gerät immer im Standby-Modus befindet. Ich persönlich ziehe eine echte Trennung vom Netz vor und trenne alles, wenn ich diese Geräte nicht benutze, über eine Steckdosenleiste mit Hauptschalter.
Der Bildschirm ist sehr gut ablesbar. Beim Einschalten dauert es einige zehn Sekunden, bis das Gerät einsatzbereit ist. Auf jeden Fall ist es ratsam, solche Geräte nicht für kurze Zeit ein- und auszuschalten, da die maximale Genauigkeit erst nach einer Aufwärmphase (10…30 Minuten) gewährleistet ist. Aber man kann sie natürlich auch schon vorher benutzen. Im Gegensatz zu den besseren Modellen mit den etwas lauten Lüftern schweigt sich das SDM3045X, abgesehen von ein paar diskreten Relaisklicks, aus, da es gar keinen Lüfter besitzt.
Auf der Rückseite (**Bild 1**) befinden sich zwei BNC-Buchsen, ein Eingang zum Triggern der Messungen durch ein externes Signal und ein Ausgang, der das Ende der Messung anzeigt. Diese Buchsen sind vor allem für automatische Messungen nützlich, und werden im täglichen Gebrauch eher selten verwendet. Außerdem gibt es einen RJ45-LAN-Anschluss und einen USB-B-Stecker, um das Gerät mit einem PC zu verbinden. Das SDM3045X verfügt außerdem über einen Kensington-Sicherheitsschlitz zur Befestigung

Bild 1. Rückansicht mit Anschlüssen.



▲ *Bild 2. Histogramm-Modus.*

eines diebstahlsichernden Drahtes. Die 10-A-Sicherung des Amperemeters ist ebenfalls dort zugänglich und kann im Falle eines Überstromereignisses leicht ausgewechselt werden.

Zunächst sollten Sie prüfen, ob die Firmware auf dem neuesten Stand ist. Siglent aktualisiert seine Geräte häufig, und die Updates können leicht von der Website heruntergeladen werden. Die Aktualisierung erfolgt über einen USB-Stick, der an der Frontplatte eingesteckt wird.

Grundlegende Bedienung

Die grundlegende Bedienung ist einfach und intuitiv. Über die Tasten auf der Vorderseite kann man leicht von einem Messmodus zum anderen wechseln. Es ist möglich, die Messgeschwindigkeit zu ändern, was sich auf die zeitliche Auflösung auswirkt. In vielen Fällen dürfte der schnelle Modus ausreichen, der eine nahezu sofortige Reaktionszeit bietet. Auch die automatische Bereichsumschaltung reduziert die notwendigen Eingriffe in den Messvorgang erheblich. Der Bereich kann jedoch jederzeit problemlos auch manuell gewählt werden.

Für die Messung von Wechselspannungen und -strömen bietet es eine Bandbreite von 100 kHz und True-RMS-Messungen. Wie bei den meisten Multimetern muss jedoch bei asymmetrischen Signalen eine manuelle Korrektur des Spitzenfaktors vorgenommen werden; das Handbuch enthält alle Einzelheiten zu diesem Punkt.

Im Durchgangs- oder Diodentestmodus können wir den Schwellwert des Signaltons sowie glücklicherweise seine Lautstärke einstellen. Bei allen Messungen kann man einen „relativen“ Modus einstellen, um die Messungen in Bezug auf einen vordefinierten Wert zu sehen. Sie können die Erfassung auch manuell stoppen und die letzte Messung jederzeit speichern.

Interessante Eigenschaften

Das SDM3045X verhält sich wie ein einfaches Multimeter, bietet aber zusätzliche Funktionen, die das Elektronikerleben leichter machen. Hier sind die wichtigsten davon:

- Verschiedene Anzeigemodi: Im Histogramm-Modus (**Bild 2**) wird die statistische Verteilung der Messwerte grafisch dargestellt, so dass Sie die am häufigsten auftretenden Werte sofort erkennen können, und im Kurven-Modus wird die zeitliche Entwicklung des Messwerts angezeigt, selbst für Zeiträume von nur einer Sekunde. Dies ist sehr praktisch, um die Veränderung der Werte im Laufe der Zeit zu beobachten. In all diesen Modi können die Skalen- und Grenzwerte automatisch berechnet oder manuell eingestellt werden.
- Ein statistischer Modus zeigt den Mittelwert, das Minimum, das Maximum, die Standardabweichung und andere Statistikparameter der Messungen an
- Die Dualanzeige (**Bild 3**) ermöglicht die gleichzeitige Überwachung von zwei Parametern, zum Beispiel Spannung und Frequenz
- Ein dB/dBm-Modus ermöglicht relative Messungen in dB oder Leistungsmessungen in dBm für eine einzustellende Lastimpedanz. ACHTUNG: Diese Einstellung ändert nicht die Eingangsimpedanz des Multimeters; Sie müssen die gewünschte Last extern anschließen.
- Der Grenzwertmodus (Limits) (**Bild 4**) zeigt visuell an, ob der gemessene Wert zwischen zwei einstellbaren Grenzwerten liegt, was sehr praktisch für sich wiederholende Schaltungseinstellungen ist, ohne die Werte interpretieren zu müssen
- Der Modus *Probe Hold* (**Bild 5**), einer meiner Favoriten, speichert automatisch aufeinanderfolgende stabile Messungen, was für die Sortierung von Bauteilen sehr praktisch ist

Bild 3. Duale Anzeige.

Bild 4. Limits-Modus.

Bild 5. Probe-Hold-Modus.

- Sie können den Erfassungsmodus, die Abtastgeschwindigkeit und die Wahl zwischen automatischer, manueller oder durch ein externes Triggersignal mit definierbarem Wert und Polarität ausgelöster Erfassung fein einstellen
- Messwerte und/oder aktuelle Einstellungen können auf einem USB-Laufwerk gespeichert werden, um sie einfach abrufen zu können
- Der Thermometermodus akzeptiert verschiedene Arten von gängigen Thermoelementen (mit Kaltstellenkompensation) und Widerstandssensoren. Hinweis: Im Lieferumfang des Geräts ist kein Temperaturfühler enthalten.
- Die integrierte Hilfe erinnert Sie an die richtige Verwendung und die Anschlüsse für die Messungen

Verwendung mit einem PC

Das Gerät kann über SPI, USB oder LAN-Ethernet gesteuert werden. LabVIEW-Treiber sind auf der Siglent-Website verfügbar. Für den Betrieb des Geräts wird eine Protokolldokumentation bereitgestellt. Darüber hinaus bietet Siglent die kostenlose Windows-Anwendung EasyDMM an (**Bild 6**), mit der sich alle Siglent-Multimeter einfach steuern, Messungen anzeigen und in CSV-Dateien exportieren lassen. Diese Anwendung funktioniert trotz ihres etwas veralteten Aussehens sehr gut.

Fazit

Ich bin von Siglent-Messgeräten noch nie enttäuscht worden, und auch dieses Multimeter hat bei mir insgesamt einen guten Eindruck hinterlassen. Es ist praktisch und deckt wahrscheinlich die meisten Bedürfnisse eines Labors für Fehlersuche oder Forschung und Entwicklung ab. Der Preis ist etwas höher als bei asiatischen Einsteigermodellen, aber die Qualität des Produkts rechtfertigt ihn. Das Preis/Leistungs-Verhältnis ist ausgezeichnet, und wenn Sie in ein praktisches und effizientes Tischmultimeter investieren wollen, kann ich es nur empfehlen.

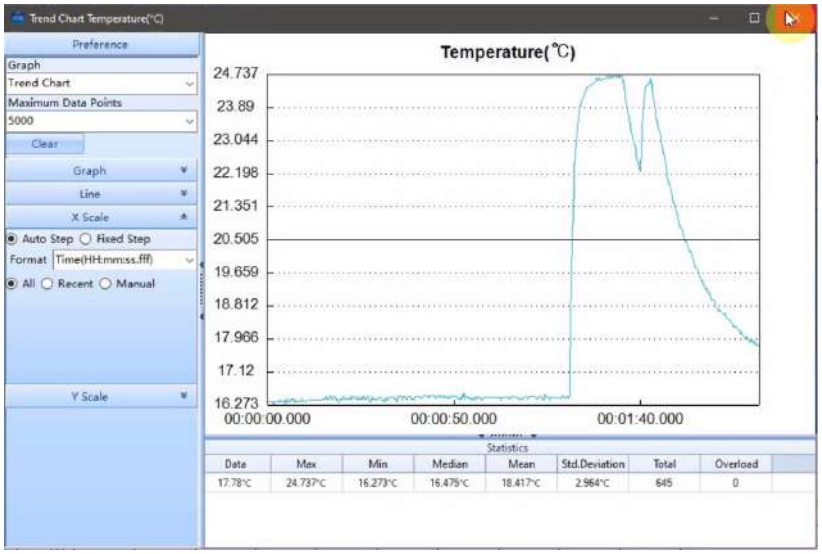


Bild 6. Windows-Anwendung EasyDMM.

Was mir besonders gefallen hat:

- Die Benutzerfreundlichkeit
- Die Gesamtqualität und das gut lesbare Display
- Die Fülle der angebotenen Funktionen
- Geräuscharm (kein Lüfter)
- Gutes Preis/Leistungs-Verhältnis
- Leicht verfügbare Updates

Was mir nicht gefallen hat:

- „Weicher" Netzschalter, der nur in den Schlafmodus schaltet
- Das Wartungshandbuch: Es fehlen klare Informationen oder nützliche Anleitungen zur Kalibrierung - es gibt ein Verfahren, das PHP-Skripte nutzt, aber es ist sehr komplex. Es gibt Anweisungen zum Öffnen des Geräts und zur Überprüfung der Spannungen, aber keinen Schaltplan. Bei Problemen muss man sich an Siglent wenden.

RG — 230126-02

Über den Autor

Philippe Demerliac, geboren 1962, ist Entwickler elektronischer Schaltungen für wissenschaftliche Zwecke und begeistert sich sein Leben lang für Präzisionsmechanik, Messgeräte, Metallverarbeitung und Wissenschaft im Allgemeinen. Obwohl er sich beruflich der Software zugewandt hat, bleibt er der „harten" Elektronik treu. In dem Bestreben, der Community etwas zurückzugeben, hat Philippe die Website *cyrob.org* und später einen französischen YouTube-Kanal eingerichtet: *youtube.com/@Cyrob-org*

Haben Sie Fragen oder Kommentare?

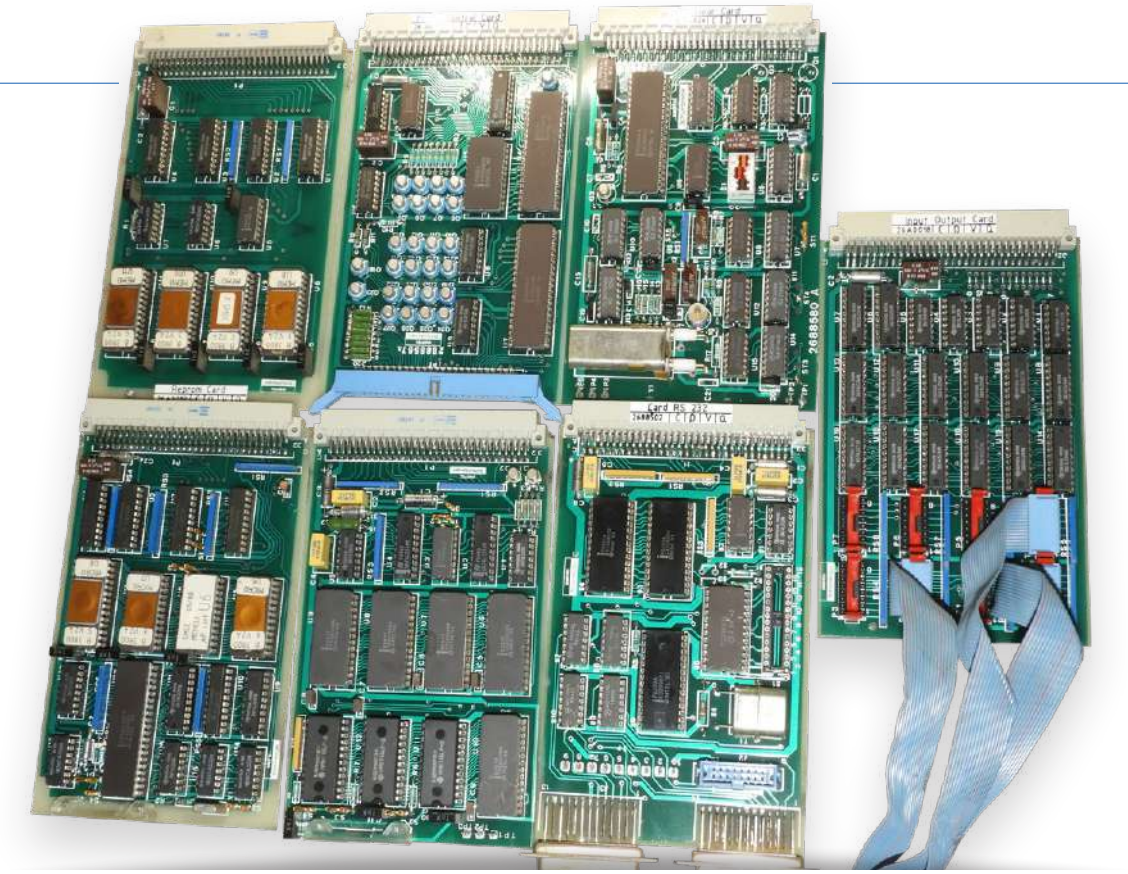
Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Kontaktieren Sie den Autor unter info@cyrob.org oder das Elektor-Team unter redaktion@elektor.de.

Passendes Produkt

- **Digitales Multimeter SDM3045X von Siglent**
  https://elektor.de/17892

WEBLINK

[1] Siglent, Datenblatt SDM3045X: https://bit.ly/40iCh00



Mikroprozessoren für eingebettete Systeme

Bemerkenswerte Bauteile

Von David Ashton (Australien)

Embedded-Ingenieure und Hobby-Elektroniker haben heutzutage dank des riesigen Angebots an Mikrocontrollern die Qual der Wahl, wenn es um Rechenleistung geht. Aber das war nicht immer so! Lassen Sie uns einen Blick auf die eingebetteten Systeme der Vergangenheit werfen...

Heutzutage verfügen selbst Low-End-Mikrocontroller wie der 8-polige AVR ATtiny über bis zu 8 KB Flash-Speicher und bis zu 512 Byte EEPROM und SRAM. Hinzu kommt eine Vielzahl an Peripherie: zwei Timer/Zähler, eine serielle Schnittstelle, ein Vier-Kanal-ADC, ein Watchdog-Timer und ein Analogkomparator. Der Controller führt seinen Code mit 10 MHz aus und unterstützt einen großen Eingangsspannungsbereich. Im Vergleich dazu war das Leben der frühen Entwickler eingebetteter Systeme hart. Sie waren auf Mikroprozessoren ohne On-Chip-Speicher (abgesehen von den Registern) oder Peripherie angewiesen. Stattdessen mussten all diese guten Dinge extern hinzugefügt werden, um ein komplettes System zu erhalten.

Das bedeutete, dass andere ICs an den Bus des Mikroprozessors angeschlossen werden mussten - sehr eigenartig. Normalerweise hatte man 16 Bits für Adressen, die Platz für 64 KB Speicher boten, und 8 Bits für Daten. Hinzu kamen verschiedene Lese-/Schreib- und Timing-Steuersignale. Alle angeschlossenen Peripheriebausteine hatten meist nur eine einzige Funktion. Der Speicher bestand aus einem ROM (vorprogrammiert oder als EPROM) und einem RAM, das dynamisch sein musste, wenn man einen vernünftig großen Speicher haben wollte. Dies alles erforderte teuflisch komplizierte Aktualisierungszeiten, für die oft ein separates IC benötigt wurde. Auch die Peripherie wurde separat hinzugefügt: Timer, Zähler, Eingangs-/Ausgangsports, UARTs für die serielle Kommunikation, ADCs, CRT-Controller (mit Ausgängen zur Ansteuerung von Bildröhren) und so weiter. Jeder Chip war ein 24- bis 40-poliges Dual-Inline-Monster, das in der Regel auf den Mikroprozessor desselben Herstellers abgestimmt war und mit maximal 4 MHz lief. Aber damit hatte der Spaß noch kein Ende, denn als nächstes kamen die grundlegenden Logik-ICs für die Dekodierung und

Bild 1. Diese Platine stammt aus der Mitte der 1980er Jahre und enthält einen 8085-Mikroprozessor. Kombiniert mit sechs anderen Platinen erreichte sie die Leistung und Ausstattung eines modernen Mikrocontrollers!

Pufferung, die so genannte Glue-Logik. Programmierbare Logikchips wie GALs, PALs und ULAs waren für diese Aufgaben sehr beliebt, aber manchmal gossen die Hersteller diese Funktionalität auch in spezielle kundenspezifische Chips.

Intel brachte den Ball im Jahr 1974 mit dem ehrwürdigen 8080 ins Spiel, was sozusagen den wahren Beginn des Mikroprozessorzeitalters einläutete. Der 8080 benötigte mindestens zwei unterstützende Chips, den 8224-Taktgenerator und den 8228-Buscontroller. Ungewöhnlich für die heutigen Ingenieure war auch, dass er sowohl eine Spannungsversorgung von ±5 V als auch +12 V benötigte. Erst spätere Versionen wie der 8085 (**Bild 1**) benötigten nur noch eine +5-V-Versorgung. Allerdings war der 8085 aufgrund des seltsamen Bus-Multiplexing-Schemas komplexer. Dies erforderte zusätzliche Glue-Logik, obwohl eine Reihe von dedizierten 8085-Peripheriechips verfügbar war.

Die Programme wurden während der Entwicklung in UV-EPROMs gespeichert (Ultraviolet-erasable programmable Read-only-Memory). Dank eines kleinen Fensters oben im Chipgehäuse konnte der Inhalt mit einem EPROM-Löschgerät - einem Kasten mit einer UV-Lampe, einem Timer und einem ausziehbaren Fach für das EPROM - in etwa 20...30 Minuten gelöscht werden. Danach konnte das EPROM mit einem EPROM-Programmiergerät an der seriellen oder parallelen Schnittstelle des PCs neu programmiert werden. Wie das vor dem Aufkommen der PCs gemacht wurde, entzieht sich meiner Kenntnis, aber ich erinnere mich dunkel, dass ich Programmiergeräte mit Hex-Tastaturen gesehen habe, auf denen die Daten für jeden Speicherplatz eingetippt wurden...

Programmiert wurde in Maschinencode, wobei in jeden Speicherplatz hexadezimale Werte eingegeben wurden. Die andere (komfortablere) Möglichkeit war die Assemblersprache, die Mnemonics für die Operationen und Bezeichnungen für Variablen und Codeabschnitte verwendete. Anweisungen wie „ADC B“ bedeuteten „addiere den Inhalt von Register B zu Register A, mit Übertrag“. Sprachen wie C waren nur ein ferner Traum, aber wenn man Glück hatte, bekam man vielleicht einen BASIC-Interpreter. Wenn Ihr Programm nicht funktionierte oder umgeschrieben werden musste, wurde das EPROM gelöscht und neu programmiert. Und wenn man sein endgültiges fehlerfreies Programm hatte, konnte man es auf ROMs brennen lassen. Diese waren zwar pro Gerät billiger als EPROMs, aber man konnte sie nicht löschen!

Auch andere Hersteller stiegen auf den Mikroprozessor-Zug auf. Motorola hatte den 6800, MOS Technology den 6502 und National Semiconductor den SC/MP. Einige ehemalige Intel-Entwickler gründeten die Firma Zilog, die den Z80 (**Bild 2**) anbot, eine Art Luxus-8080. Die meisten dieser Chips wurden in den ersten Heimcomputern wie dem Sinclair ZX80/81, dem Spectrum (Z80), dem Commodore 64 (6502-Variante) verwendet. Das SC/MP-Projekt von Elektor aus dem Jahr 1978 war und ist dafür ein weiteres gutes Beispiel [1].

Im Jahr 1980 führte Intel den 8051 ein. Dieser hatte ein kleines ROM (manchmal auch EPROM) und ein wenig RAM an Bord, aber auch vier 8-Bit-Ports, einen UART und zwei Zähler/Timer. Da alles in einem einzigen Chip untergebracht war, könnte man sagen, dass mit diesen Chips die Ära der Mikrocontroller begann. Der Rest ist, wie man so schön sagt, Geschichte. Die 8051-Architektur ist auch heute noch im Einsatz, allerdings mit viel mehr Peripheriefunktionen als ihre Vorgänger und ohne die ganzen Busse und die Verbindungslogik der Vergangenheit.

Der 8051 und die 68xx-Bausteine von Motorola brachten das umfangreiche und vielseitige Angebot an Mikrocontrollern hervor, das wir heute kennen und lieben: AVRs, PICs und eine Vielzahl von Arm-basierten Mikrocontrollern, aus denen Boards wie Arduino und Raspberry Pi entstanden sind. Intel entwickelte den 8085 zum 16-Bit-8086, der im ersten IBM PC XT verwendet wurde, und dann zum 80286, 80386, 80486 und den Pentium-Prozessoren, die noch heute in PCs eingesetzt werden. Single Board Computer auf Mikroprozessor- und nicht Mikrocontrollerbasis waren bis in die 1990er Jahre üblich.

Als der 8080 auf den Markt kam, war ich gerade 18 Jahre alt und erinnere mich, dass ich in den damaligen Hobby-Elektronikmagazinen darüber gelesen hatte. Ich hatte das Glück, schon früh in meiner Karriere an Mikroprozessoren arbeiten zu können, was sehr spannend war. Meine Aufgaben konzentrierten sich auf ein wenig Programmierung und viel Fehlersuche. Ich hatte das Glück, diese Technologie während meiner gesamten Laufbahn verfolgen zu können, und sie hat mich immer fasziniert. Im Vergleich zu den heutigen Mikrocontrollern waren die alten Mikroprozessoren definitiv etwas Besonderes, aber es hat trotzdem viel Spaß gemacht, an ihnen zu arbeiten. So verklärt man die Vergangenheit!

RG — 230047-02

Bild 2. Ein weiteres Arbeitspferd der frühen eingebetteten Systeme war der Z80 von Zilog. Hier wird er von einem seriellen I/O-Controller (SIO) und einem Zähler/Timer-Schaltkreis (CTC) begleitet.

WEBLINK

[1] J. Buiting, „Der Elektor SC/MP Computer“, Elektor Summer Circuits 2022: https://www.elektormagazine.de/magazine/elektor-262/60768

Mikrocontroller-Dokumentation verstehen

Teil 3: Blockdiagramme und mehr

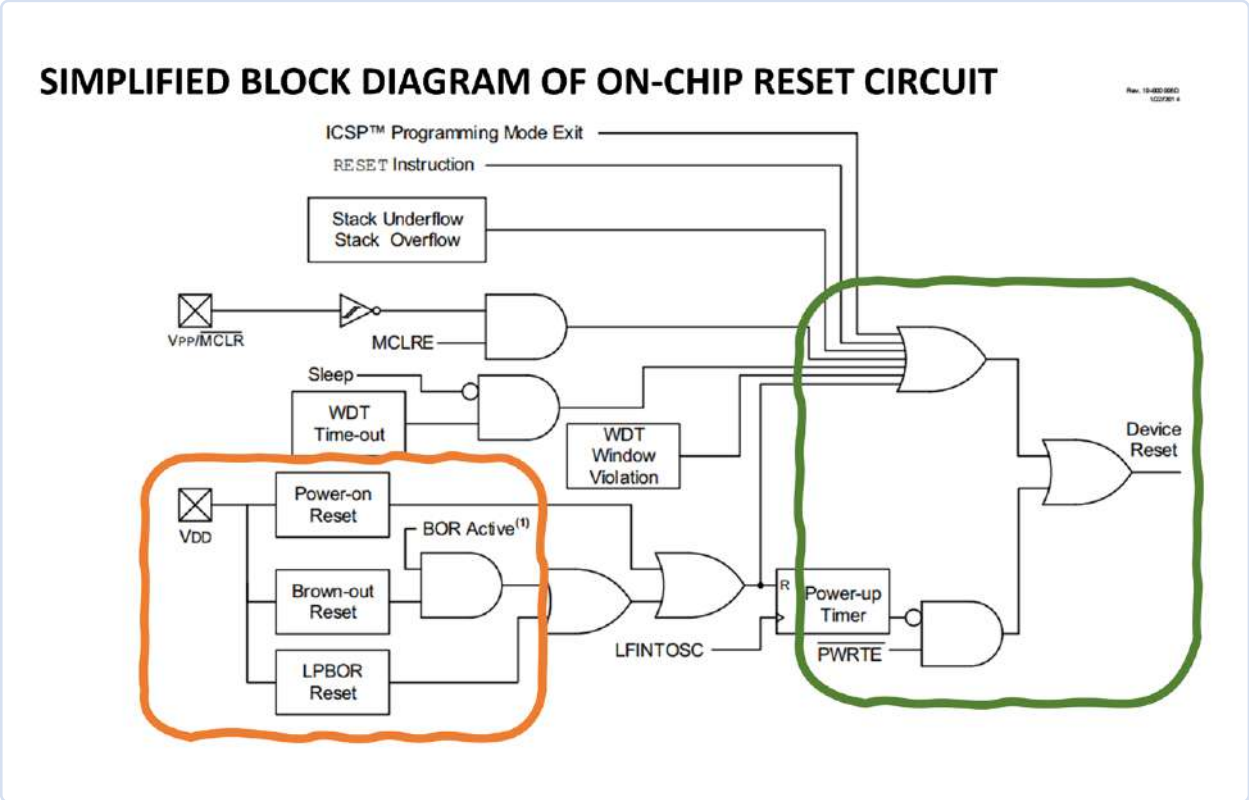


Bild 1. Die Reset-Schaltung bietet nur sehr wenige Konfigurationsmöglichkeiten, wobei einige Reset-Quellen mit dem VCC-Stromversorgungspin verbunden sind. (Quelle: Microchip Technology)

Von Stuart Cording (Elektor)

In den vorangegangenen Teilen dieser Reihe [1] haben wir uns mit der Funktionalität von Registern beschäftigt und einen Blick auf das Blockdiagramm des Taktgebers geworfen. Hier stellen wir weitere Blockdiagramme vor und erläutern, wo Sie den Rest der erforderlichen Informationen in der Dokumentation finden können.

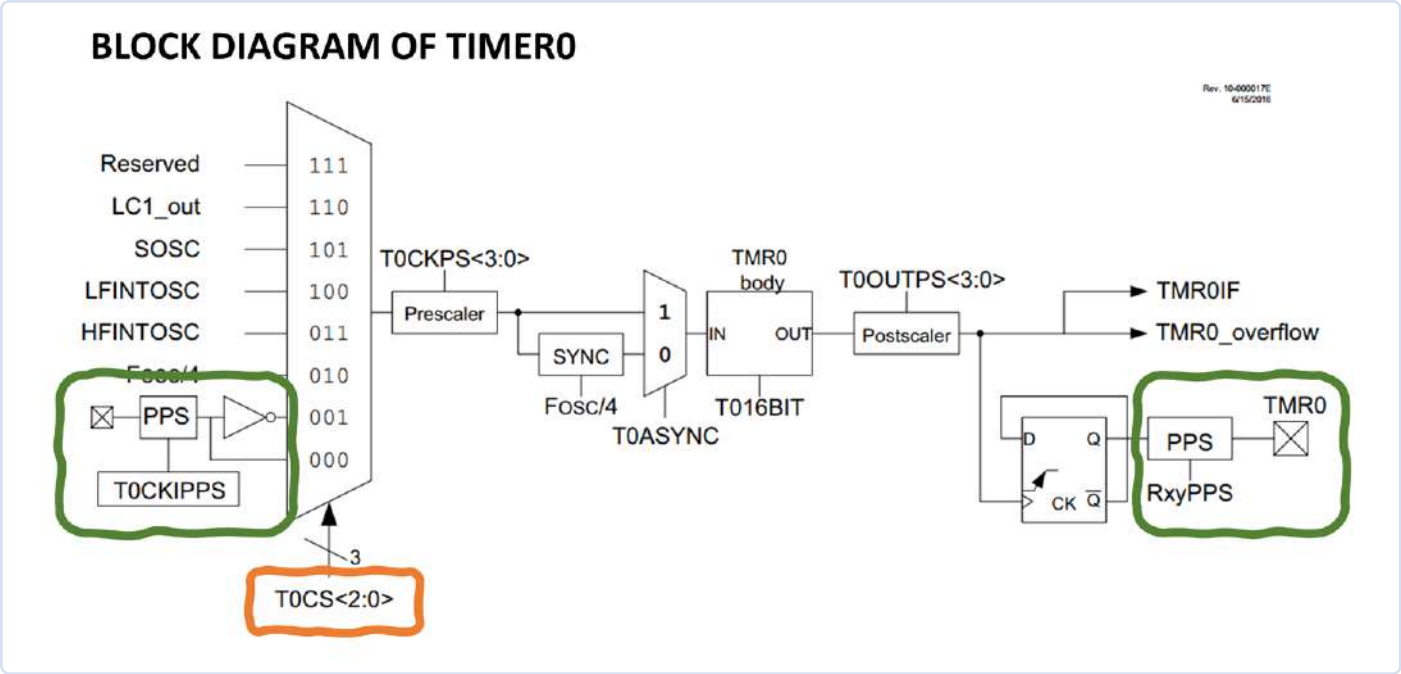


Bild 2. Das Blockdiagramm für `TIMER0` *zeigt, dass er über eine Reihe von Taktquellen verfügt und zusätzlich zu den internen Interrupt-Signalen auch ein Ausgangssignal erzeugen kann. (Quelle: Microchip Technology)*

Auch hier steht der 8-Bit-Mikrocontroller PIC16F18877 von Microchip Technology im Mittelpunkt, und Sie sollten das Datenblatt [2] herunterladen, wenn Sie den Ausführungen folgen wollen.

Neustart mit Reset

Wenn die Taktperipherie der wichtigste Teil eines Mikrocontrollers ist, so kann man die Reset-Schaltung als zweitwichtigsten Block bezeichnen. Wie der Taktgeber muss er nur einmal konfiguriert werden (falls es Konfigurationsoptionen gibt), aber es erspart später eine Menge Ärger, wenn Sie gut verstehen, was einen Reset auslösen kann.

In dem Beispiel von *Seite 100* (**Bild 1**) können wir sehen, dass zwei Pins (markiert als Quadrate mit einem Kreuz, links) einen Reset auslösen können. Dies geschieht zusätzlich zu einer Vielzahl anderer interner Reset-Quellen. Eine davon ist der `RESET`-Befehl, zwei kommen vom Stack-Unter- und -Überlauf, während andere mit dem Watchdog-Timer verbunden sind.

Die orangefarbene Markierung zeigt, dass eine Reihe von Reset-Mechanismen wie Power-on-Erkennung und Brown-out mit dem Versorgungs-Pin des Mikrocontrollers verbunden ist. Dieser Block bietet nur wenig Eingriffsmöglichkeiten, da im Wesentlichen alle Interrupt-Quellen potenzielle Reset-Signale sind (grün markiert). Wie wir es bereits beim `STATUS`-Register kennengelernt haben, kann in einigen Fällen die Ursache für einen erfolgten Reset ermittelt werden.

Peripherie-Blockdiagramme

Die Blockdiagramme für Peripheriefunktionen sind ebenso vielfältig wie komplex. Hier haben wir einen Teil eines einfachen Timers (`TIMER0`) von *Seite 396* ausgewählt (**Bild 2**). Die orangefarbene Markierung zeigt wieder die Verwendung einer Gruppe von Bits in (vermutlich) einem Register zur Auswahl unter einer Reihe von Quelleingängen für diese `TIMER0`-Peripherie. Die grüne Markierung links zeigt auch, dass ein Pin des Controllers eine Quelle sein kann, und rechts, dass der Block auch ein Signal an einen Pin ausgeben kann.

Während Sie vielleicht das D-Flip-Flop [3] auf der rechten Seite erkennen, das mit dem Ausgangspin verbunden ist, sind viele der Blöcke im Diagramm kaum mehr als Quadrate. Einige, wie der Vorskalierer und der Nachskalierer, sind selbsterklärend. Auch der `SYNC`-Block scheint selbsterklärend zu sein, aber wahrscheinlich muss der Entwickler den Begleittext im Detail lesen, um seine Funktion vollständig zu erfassen. Daher gehen Diagramme und Text in Datenblättern oft Hand in Hand. Manchmal ist es notwendig, etwas Testcode zu schreiben, um wirklich nachzuvollziehen, wie einige Peripherieblöcke funktionieren.

Entwurf einer Platine

Irgendwann muss der Entwurf auf einer Platine aufgebaut werden, wenn Sie planen, Ihr Gerät in Serie zu produzieren. Die meisten computergestützten Designprogramme (CAD) enthalten die Schaltplanblöcke für Ihren Prozessor sowie die Platinen-Footprints für verschiedene Gehäusebauformen. Fehlen sie jedoch, kann man sich wahrscheinlich mit den technischen Zeichnungen für jedes Gehäuse und die zugehörenden Leiterbahnmuster behelfen, die höchstwahrscheinlich im Datenblatt enthalten sind. In unserem Fall sind sie auf *Seite 639* zu finden.

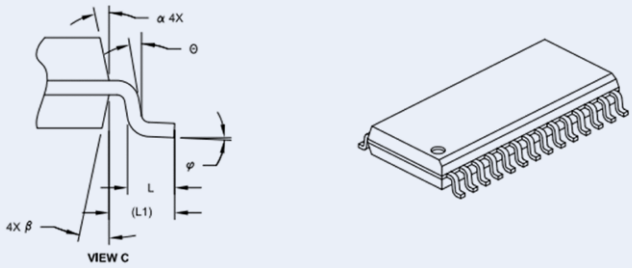
Das SOIC-Gehäuse in **Bild 3** ist auf *Seite 645* und *Seite 646* zu finden. Für den Platinenentwickler ist heutzutage auch die Höhe des Gehäuses eine nützliche Angabe, wenn das Volumen des Geräts begrenzt ist (Breite und Länge sind ja bereits bekannt). Viele Gehäuse verfügen heute auch über ein integriertes „Metallplättchen", das auf die Leiterplatte gelötet werden muss, um die Wärmeabfuhr zu unterstützen. Dies ist hier nicht der Fall, aber wenn es so wäre, würde eine Anleitung erläutern, wie man es anschließt. Möglicherweise müssen Sie auch überprüfen, ob diese Metallfläche mit der Masse des Geräts oder vielleicht mit einem anderen Pin verbunden ist.

Was steht nur manchmal im Datenblatt?

Wie Sie nun wissen, enthält ein Datenblatt eine Menge Informationen. Es gibt jedoch einige Dinge, die weggelassen werden können, weil das Datenblatt dadurch zu lang würde oder weil die Informationen für viele Controller gelten und in einer eigenen Dokumentation enthalten sind. Dazu könnten gehören:

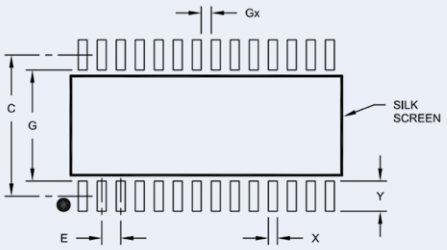
- Der Befehlssatz des Prozessors: Wenn die Anzahl der Befehle gering ist (weniger als 60...70), könnten sie alle mit einer kurzen Erläuterung im Datenblatt enthalten sein. Ist dies

28-Lead Plastic Small Outline (SO) - Wide, 7.50 mm Body [SOIC]



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		28	
Pitch	e		1.27 BSC	
Overall Height	A	-	-	2.65
Molded Package Thickness	A2	2.05	-	-
Standoff §	A1	0.10	-	0.30
Overall Width	E		10.30 BSC	
Molded Package Width	E1		7.50 BSC	
Overall Length	D		17.90 BSC	
Chamfer (Optional)	h	0.25	-	0.75
Foot Length	L	0.40	-	1.27
Footprint	L1		1.40 REF	
Lead Angle	θ	0°	-	-
Foot Angle	φ	0°	-	8°
Lead Thickness	c	0.18	-	0.33
Lead Width	b	0.31	-	0.51
Mold Draft Angle Top	α	5°	-	15°
Mold Draft Angle Bottom	β	5°	-	15°

28-Lead Plastic Small Outline (SO) - Wide, 7.50 mm Body [SOIC]



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Contact Pitch	E		1.27 BSC	
Contact Pad Spacing	C		9.40	
Contact Pad Width (X28)	X			0.60
Contact Pad Length (X28)	Y			2.00
Distance Between Pads	Gx	0.67		
Distance Between Pads	G	7.40		

Bild 3. Die Gehäusezeichnungen enthalten alle Abmessungen des Gehäuses (links) sowie den empfohlenen „Fußabdruck" für die Platine (rechts). (Quelle: Microchip Technology)

nicht der Fall, gibt es wahrscheinlich ein separates Dokument, in dem jede Anweisung im Detail erklärt wird.

- Code-Beispiele: Diese erscheinen oft, wenn der Mikrocontroller üblicherweise in Assembler programmiert wird. Beispiele in Hochsprachen wie C haben wenig Sinn, da der Compiler die verwendeten Anweisungen genau definiert.
- Schaltungsbeispiele: Diese sind am häufigsten im Zusammenhang mit Funktionen zu finden, die für den gewählten Mikrocontroller einzigartig sind, zum Beispiel den Oszillator oder die Stromversorgung betreffend, oder mit etwas komplexeren Schnittstellen, die besondere Balastungs-Anforderungen an die Signale stellen, wie zum Beispiel USB. Es ist jedoch wahrscheinlicher, dass sie in einem Anwendungshinweis zu diesem Thema ausführlicher behandelt werden.

Was fehlt im Datenblatt?

Die folgenden Punkte sind normalerweise nicht im Datenblatt enthalten und werden stattdessen in anderen Dokumentationen behandelt. Dies liegt in der Regel daran, dass es sich um ein Thema handelt, das für eine Vielzahl von Mikrocontroller gilt.

- Flashen des Mikrocontroller-Speichers: Dieses Thema wird normalerweise gesondert behandelt, wenn es um die Programmierung von Mikrocontrollern in der Massenproduktion geht.
- Verwendung von Entwicklungswerkzeugen: Für den Compiler, die Verwendung einer integrierten Entwicklungsumgebung (IDE) und die Debug-Tools gibt es eine eigene Dokumentation.
- Detaillierte Erläuterung des Prozessorkerns: Dies wird oft in einem separaten Dokument behandelt, insbesondere für 16-Bit- und 32-Bit-Cores.
- Ausführliche Erläuterung komplexer Peripherieblöcke: USB, Grafikschnittstellen und Ethernet-Peripherie werden in der Regel in separaten Dokumenten behandelt, da sich der Umfang des Mikrocontroller-Datenblatts durch die Einbeziehung jedes einzelnen Blocks leicht verdoppeln könnte.
- Errata: Alle Fehler in der Dokumentation oder Umgehungslösungen zur Behebung von Siliziumfehlern, die nicht behoben werden können.

Andere Mikrocontroller-Dokumentationen

Neben dem Datenblatt werden in der Regel die folgenden Dokumente zur Verfügung gestellt:

- Family Reference Manual: Während das Datenblatt genau angibt, was ein Mikrocontroller kann, bieten solche Dokumente einen übergeordneten Überblick darüber, was alle Mikrocontroller einer Familie können.
- Application Notes: In diesen Dokumenten wird sehr detailliert beschrieben, wie bestimmte Peripherieblöcke oder eine Gruppe von Peripherieblöcken verwendet werden, um eine Anwendung oder Schnittstelle zu implementieren. Das Datenblatt kann erklären, wie der CAN-Peripherieblock (Controller Area Network) zu verwenden ist, aber die Anwendungshinweise erklären, wie er als Teil eines CAN-Netzwerks zu verwenden ist, und bieten Anleitungen zu übergeordneten Softwareprotokollen und zur Auswahl geeigneter Transceiver.
- Programming Specification: Für die Programmierung in Massenproduktionsumgebungen werden hier die erforderlichen Spannungen und Timings sowie das für die Programmierschnittstelle verwendete Protokoll im Detail erläutert.

Wie man viel schriftliches Material konsumiert

Leider gibt es keine Möglichkeit, alle Daten, die mit einem Mikrocontroller und all seinen Werkzeugen verbunden sind, schnell zu erfassen. Wenn Sie Anfänger sind, ist es wahrscheinlich am besten, das Datenblatt zusammen mit einem Buch oder Artikel über den Mikrocontroller zu lesen, den Sie verwenden möchten. Elektor hat nicht nur Bücher für diejenigen, die sich für PIC-Mikrocontroller interessieren; es gibt auch Einsteigerbücher für den STM32 [4] und ARM-basierte Mikrocontroller [5] im Allgemeinen. Oder warum nicht einmal ein einfaches MSP430-Projekt [6] ausprobieren? Die Kombination aus praktischen Beispielen und zwei Arten von schriftlichen Erklärungen (ein Buch/Artikel und das Datenblatt selbst) wird Ihnen helfen. Wenn Sie fortgeschrittener sind,

konzentrieren Sie sich am besten auf die Abschnitte, die den Prozessorkern, den Taktbaum und den Reset-Block behandeln, gefolgt von den Peripherieblöcken, die Sie verwenden wollen. Dann müssen Sie sich mit der Dokumentation der Compiler-Toolchain vertraut machen. Nutzen Sie auch die Softwarebibliotheken und Beispiele, um Ihr Verständnis zu vertiefen, und stellen Sie Fragen in Foren, wenn Sie weitere Hilfe benötigen.
Datenblätter können schwierig zu lesen und zu verstehen sein und sind nicht die spannendste Lektüre. Sie sind jedoch (meistens) genau und stellen eine technische Beschreibung der Funktionalität dar. Anhand der Errata können Sie bestimmen, wie viel Vertrauen Sie in das Datenblatt setzen dürfen. Bleiben Sie dabei, und mit der Zeit werden Sie ein gutes Verständnis dafür entwickeln, wie Mikrocontroller-Datenblätter aufgebaut und formuliert sind.

RG – 230286-02

Haben Sie Fragen oder Kommentare?

Haben Sie Fragen oder Kommentare zu diesem Artikel? Schreiben Sie dem Autor eine E-Mail an stuart.cording@elektor.com.

Passende Produkte

- **T. Hanna, Mikrocontroller-Basics mit PIC, Elektor 2019**
  Buch, kartoniert, SKU 18945: https://elektor.de/18945
  E-Buch, PDF, SKU 18946: https://elektor.de/18946

- **A. Pratt, Programming the Finite State Machine, Elektor 2020**
  Buch, Paperback, SKU 19327: https://elektor.de/19327
  E-Buch, PDF, SKU 19328: https://elektor.de/19328

WEBLINKS

[1] Mikrocontroller-Dokumentation verstehen: https://www.elektormagazine.de/search?query=Mikrocontroller-Dokumentation
[2] Datenblatt PIC16F18877 von Microchip Technologies: https://microchip.com/en-us/product/PIC16F18877
[3] Das D-Flip-Flop: https://de.wikipedia.org/wiki/Flipflop#D-Flipflop
[4] Einführung in STM32 Nucleo Boards: https://elektor.de/programming-with-stm32-nucleo-boards-e-book
[5] ARM-basierte Mikrocontroller: https://elektor.de/embedded-in-embedded
[6] Einfaches MSP430-Projekt: https://elektormagazine.de/labs/elektorpost-no-1-led-earring

LoRa-Wetterstation mit niedriger Stromaufnahme

Bauen Sie Ihre entfernte Wetterstation selbst!

Von Edward Ringel (USA)

Als meine in die Jahre gekommene entfernte Temperatur- und Luftfeuchtigkeitsmessstation irgendwann ausfiel, ersetzte ich sie durch ein selbst entworfenes System: ein batteriebetriebener Außensensor und ein batteriebetriebenes Innendisplay, über LoRa verbunden. Es war nicht einfach, ein zuverlässiges Gerät zu entwickeln, das einem Winter im Nordosten der USA standhält! Aber lesen Sie selbst…

Bild 1. Ein schneebedeckter Sensor hängt in der Nähe unserer Einfahrt.

Die Mindestanforderung an das System bestand in der Entwicklung eines batteriebetriebenen Außensensors (**Bild 1**) und eines ebenso batteriebetriebenen Displays für Innenräume, das ebenfalls mit einem Sensor ausgestattet sein sollte. Im weiteren Verlauf des Projekts fügte ich einen sehr einfachen Webserver hinzu.

Dazu wollte ich die Arduino-Entwicklungsumgebung mit ihren zahlreichen tollen Bibliotheken nutzen. Um das Projekt einfach zu gestalten, plante ich, handelsübliche Bauteile und -gruppen zu verwenden, die von Hand gelötet werden konnten. Schließlich war die Minimierung der Stromaufnahme eine wichtige Eigenschaft. Den Außensensor für die kalten Winter in Maine anzupassen, erwies sich dabei als unerwartet schwierig.

Allgemeine Überlegungen führten zur Auswahl der Hardware und der Boards - lesen Sie unten mehr darüber und über die Schwierigkeiten, auf die ich stieß. Zunächst einmal werden eine Außenstation, eine Innenstation und eine Basisstation benötigt, wobei letztere die Daten der beiden Messstationen sammelt und an einen Computer weiterleitet. Daher benötigen alle drei Einheiten mindestens eine Mikrocontroller-Platine und eine Funkeinheit, um miteinander zu kommunizieren. Die beiden Messstationen für außen und innen benötigen außerdem Sensoren und eine Zeitsteuerung, auf die später noch eingegangen wird. Im Folgenden finden Sie eine kurze Übersicht über die entscheidenden „Bausteine“ der drei Stationen:

- Außenstation: Artemis Nano, LoRa-Funkmodul, Zeitschaltuhr, Sensoren
- Innenstation: Artemis Thing Plus, LoRa-Funkmodul, Timer, Sensoren, Display
- Basis- oder Heimstation: Raspberry Pi Pico, Arduino Mini, LoRa-Funkmodul

Überlegungen zur Stromversorgung

Messungen von Temperatur, Luftfeuchtigkeit und Luftdruck werden regelmäßig durchgeführt. Der Strombedarf wird minimiert, indem das entfernte Messgerät in einen stromsparenden oder Schlaf-Mo-

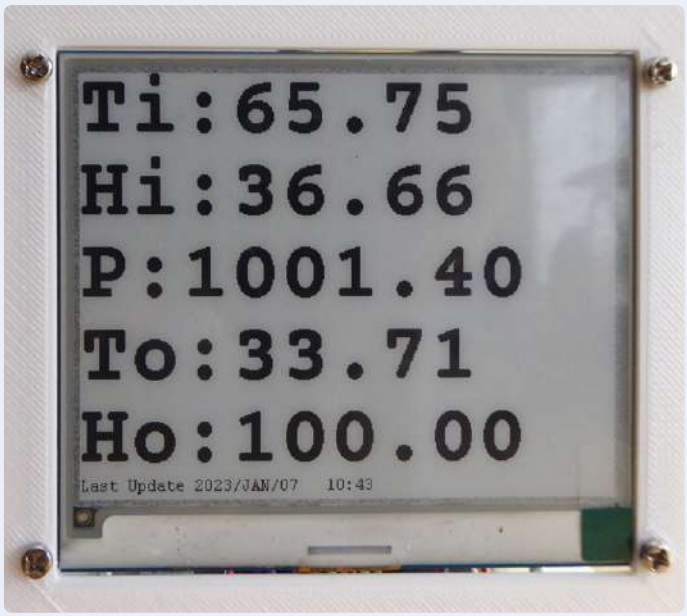


Bild 2. Mein ePaper-Display zeigt die Temperatur (in°F), die relative Luftfeuchtigkeit und den äußeren Luftdruck (unkorrigiert/nicht normiert) an. Die Indizes „i" und „o" beziehen sich auf innen und außen.

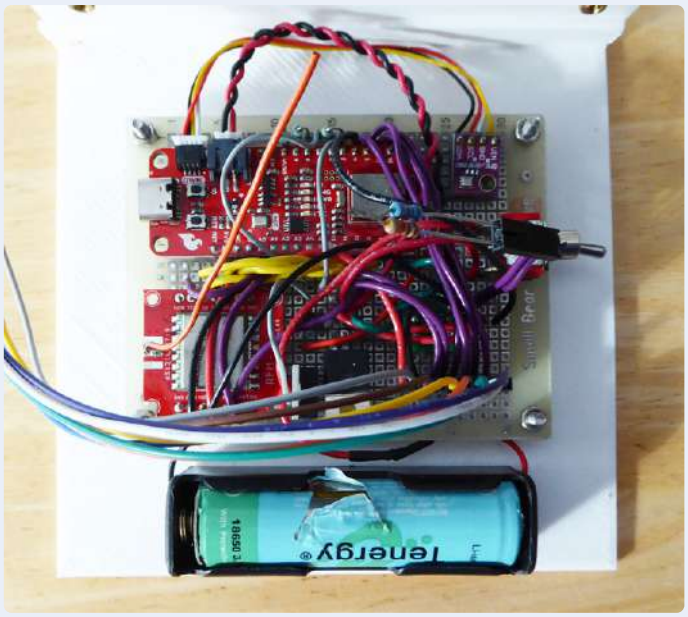


Bild 3. Frühe Version des Display-Boards, vor der Änderung der Artemis-Platine und dem Hinzufügen des TPL5111. Optokoppler, MOSFETs, TPL5110, Schalter zum Aufladen.

dus versetzt wird, aufwacht, Messwerte erfasst und überträgt und dann in den stromsparenden Zustand zurückkehrt. Der Zyklus der Stromaufnahme wird durch T_{ON} / T_{TOT} dargestellt. Die effektive Stromaufnahme hängt ab von:

- der Stromaufnahme des Geräts, während es eingeschaltet ist und Anweisungen oder Daten verarbeitet
- dem Anteil der Zeit, in der das Gerät eingeschaltet ist, um Daten zu sammeln und zu senden (oder anzuzeigen) . Eine Minimierung des Arbeitszyklus wirkt sich erheblich günstig auf die Gesamtstromaufnahme aus.
- der Stromaufnahme des Geräts zwischen den Messungen. Im Idealfall würde zwischen den Messungen gar kein Strom aufgenommen, aber das ist keine realistische Erwartung.

Ich habe die Stromaufnahme mehrerer MCU-Platinen getestet, indem ich sie eingeschaltet habe, aber ohne Programme auszuführen oder Peripherie anzuschließen. Die Ergebnisse finden Sie in **Tabelle 1**, wobei sie möglicherweise nicht der Stromaufnahme nach einer Feinabstimmung entsprechen, also beispielsweise nach dem Abschalten von Peripherieschnittstellen oder der Verringerung der Systemfrequenz. In Anbetracht dieser Ergebnisse entschied ich mich für die Artemis-Boards von Sparkfun.

Die Stromaufnahme war auch ein Faktor bei der Entwicklung der Indoor-Display-Station, aber die Wahl des Displays war einfach: ePaper haben den geringsten Strombedarf, denn solche Display benötigen nur Energie, wenn sie aktualisiert werden (weshalb ein E-Reader im ausgeschalteten Zustand eine Grafik anzeigen kann). WaveShare bietet ein schönes 4,2"-ePaper-Display an. Die Anweisungen waren schwer zu befolgen und ich musste einige Schriftarten erstellen, aber wie **Bild 2** zeigt, funktionierte das Display schließlich gut.

Für die Messungen habe ich BME280-Sensoren von Bosch Sensortec verwendet. Die T/H/P-Ausführung misst Temperatur, Luftfeuchtigkeit und Luftdruck und es stehen robuste I²C-Bibliotheken zur Verfügung. Ihre Stromaufnahme ist vernachlässigbar.

Ein Timing-Mechanismus steuert das Tastverhältnis. Die MCU-Platine, das Funkmodul und der Sensor müssen ein- und ausgeschaltet werden - es wäre nicht sinnvoll, eine effiziente MCU zu verwenden, die mit 20 µA schläft, während das Funkmodul beständig 20 mA zieht. Anstatt den Ruhezustand der MCU und des Funkmodul per Software zu steuern, schaltete ich lieber das gesamte Gerät ein und aus.

Der 3,3-V-Regler auf der Artemis-Platine hat eine maximale Ausgangsleistung von 600 mA, genug, um alle Komponenten des Innendisplays und des Außensensors samt Funkmodule zu versorgen. Um die Stromzufuhr zu den Platinen abzuschalten, war ein Ultra-Low-Power-Timer-Chip die beste Lösung. Diese Bauteile haben eine interne Zeitschaltung, deren Intervall durch einen einzigen externen Widerstand eingestellt wird, und werden mit einer MCU verwendet. Wenn der Timer auf EIN schaltet, fließt Strom, und wenn die MCU ihre Aufgabe erfüllt hat, sendet sie eine „Erledigt"-Nachricht an den Timer, und der Stromfluss wird unterbrochen. Ich habe mit dem TPL5110-Chip auf einer selbstentworfenen Platine und auf dem Breakout-Board von Adafruit gearbeitet, beides in Kombination mit einer

Tabelle 1. Boards und ihre Stromaufnahme

Board	Stromaufnahme (mA)
Teensy 3.5	74,5
Raspberry Pi Pico	20,4
Teensy 4.1	92,6
ESP32 Adafruit Huzzah Feather	125,0
Artemis Thing Plus	8,5
Generisches ESP32-WROOM	70,3

Stromaufnahme der verschiedenen MCUs im eingeschalteten Zustand, versorgt mit 5 V an Vin, aber ohne laufende Programme. Die Stromaufnahme wird nicht nur von der MCU bestimmt, sondern auch von anderen Komponenten auf der Platine (LEDs, Spannungsregler, externer Speicher, drahtlose Konfiguration und so weiter).

frühen Version des Display-Boards (**Bild 3**). Meine erfolglosen Abenteuer mit diesem kleinen Kerl sind in [1] dokumentiert.
Letztendlich habe ich einen TPL5111 verwendet und ihn mit dem Enable-Pin des 3,3-V-Reglers auf den Artemis-Platinen verbunden (**Bild 4**). Dazu musste der SMD-Pull-up-Widerstand R1 auf dem Artemis Nano (**Bild 5**) beziehungsweise R3 auf dem Artemis Thing Plus (**Bild 6**) entfernt werden, was mir aber eine vollständige Kontrolle der gesamten Stromversorgung auf den Boards ermöglichte. Diese Widerstände setzen den ENABLE-Pin des Spannungsreglers auf „high“, solange der Pin nicht direkt mit Masse verbunden ist. Letztendlich war die Modifikation der Platinen eine elegante Lösung und die gesamte Schaltung zieht im Ruhezustand nur noch 20 µA.
Sowohl der TPL5110 als auch der TPL5111 sind interessante Chips, die es wert sind, genauer betrachtet zu werden, insbesondere im Bereich der IoT-Fernerfassung. Die Adafruit-Platinen sind großartige Implementierungen und bieten eine ausgezeichnete Flexibilität des Designs.

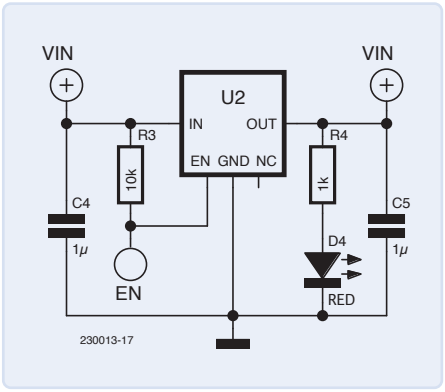


Bild 4. Der ENABLE-Pin des Artemis-Spannungsreglers wird von R3 (R1 beim Artemis Nano) auf High gezogen. Dieser Widerstand muss entfernt werden. Der ENABLE-Pin betrifft nur den Spannungsregler der Platine, nicht die MCU.

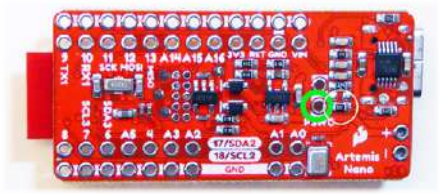


Bild 5. Artemis-Nano-Platine. R1 ist weiß eingekreist. Anders als beim Thing Plus gibt es keinen EN-, sondern einen PSWC-Anschluss. TPL5111-ENABLE sollte mit dem grün eingekreisten Anschluss verbunden werden, nachdem R1 entfernt wurde. Beachten Sie, dass sich R1 auf der Rückseite der Platine befindet und deshalb vor dem Zusammenbau entfernt werden sollte.

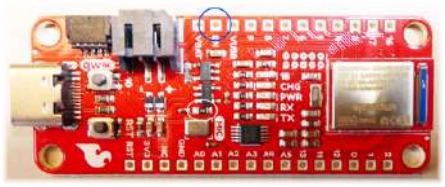


Bild 6. Die Artemis-Thing-Plus-Platine. R3 ist weiß eingekreist. TPL5111-ENABLE kann direkt an den EN-Pin (blau eingekreist) angeschlossen werden, nachdem der Widerstand entfernt wurde.

Datenübertragung

Für die Datenübertragung habe ich LoRa-Boards von HopeRF mit 915 MHz gewählt, in Europa ist eine Frequenz von 868 MHz erlaubt. Für die Arduino-Umgebung gibt es mehrere Bibliotheken, die die Implementierung vereinfachen. Die Sub-Gigahertz-Trägerfrequenz ermöglicht eine gute Durchdringung von Wänden und Gebäuden sowie längere Übertragungsstrecken. Die Übertragungsprotokolle sind für kleine Datenpakete optimiert; sie verlangen keine zeitaufwändigen Handshakes und keinen Sicherheits-Overhead. Dies bedeutet jedoch auch, dass der Programmierer für eine sichere Datenübertragung verantwortlich ist und nicht die integrierten Übertragungsprotokolle.
An dieser Stelle werden Sie sich vielleicht fragen, wie Sensor- und Anzeigestation kommunizieren, denn schließlich überschneiden sich die T_{ON}-Zeiten der beiden Geräte nur selten. Wenn der Sensor hochfährt und Daten überträgt, ist die Anzeigestation wahrscheinlich ausgeschaltet. Die Lösung war eine dritte, ständig eingeschaltete Station auf der Basis eines Raspberry Pi Pico, der keine Sensoren enthält, sondern als Datensammler und -weiterleiter fungiert und an einen Computer angeschlossen ist. Ich will und werde diese Station als Home-Station bezeichnen. Die einzige Peripherie der Home-Station sind das Funkmodul und ein paar LEDs, die beim Debuggen nützlich werden können. Der Spannungsregler des Pico kann ein paar hundert Milliampere Strom liefern, so dass ich das Funkmodul über ihn versorgen konnte. Der Pico wiederum wird über den USB-Anschluss eines PCs mit Strom versorgt und kommuniziert seriell mit einem Processing-Programm. Nur ein paar

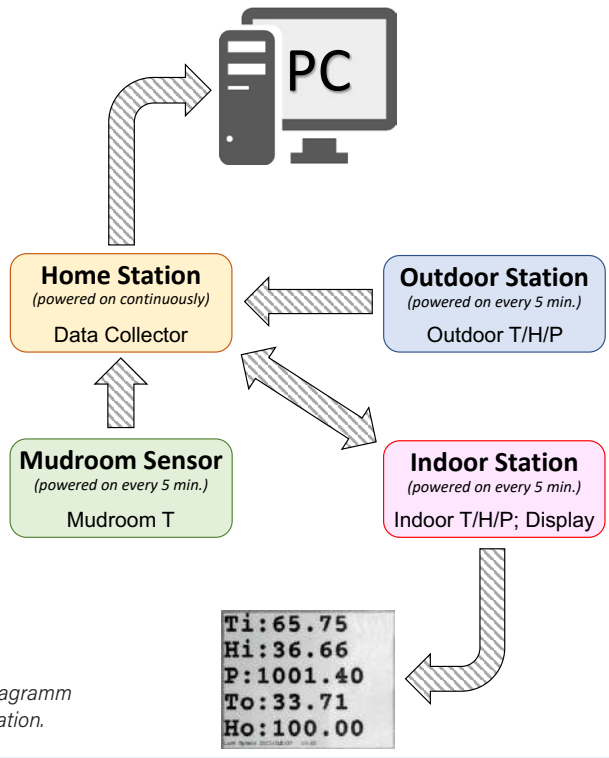


Bild 7. Datenflussdiagramm der Wetterstation.

Bild 8. Die in Epoxid eingebettete Sensorplatine. Beachten Sie das kleine Plastikquadrat in der oberen linken Ecke, das verhindert, dass das Epoxidharz in den Sensor läuft. Der USB-C-Dongle links in der Mitte ermöglicht den Zugriff auf die Platine zum Aufladen und für Software-Updates.

Bild 9. Detail des Sensorgehäuses. Alle Öffnungen sind abgeschirmt, um biologische statt Software-Bugs fernzuhalten ;-).

hundert Zeilen Code reichen aus, um den Luftdruck aufzuzeichnen und eine Tabelle mit den Messwerten auf einer Webseite zu veröffentlichen. **Bild 7** veranschaulicht den Datenfluss wie folgt:

- 1. Der Außensensor wacht alle fünf Minuten auf und sendet Werte von Temperatur, Luftfeuchtigkeit und Luftdruck an die Home-Station. Es findet keine Kommunikation zur Bestätigung von der Home-Station zurück zum Außensensor statt. Der Außensensor schaltet sich nach der Übertragung ab.
- 2. Die Home-Station speichert die letzten Messwerte des Außenfühlers im RAM. Die Außenmesswerte werden auch an den Computer zur weiteren Verarbeitung gesendet.
- 3. Der Innensensor und die Anzeigeeinheit wachen alle fünf Minuten auf. Er misst die T/H/P-Werte im Innenbereich, sendet diese Informationen an die Home-Station und speichert eine Kopie der Messungen im RAM.
- 4. Wenn die Home-Station eine Aktualisierungsanforderung vom Innensensor und der Anzeigeeinheit erhält, sendet sie diese Informationen an den Computer. Anschließend sendet sie die neusten Außen-Messwerte an die Innenstation. Danach aktualisiert die Innenstation die ePaper-Anzeige und schaltet sich ab.

Im Laufe der Zeit stellte ich fest, dass die Home-Station wiederholt blockierte, was dazu führte, dass die Anzeige und das Verarbeitungsprogramm nicht mehr aktualisiert werden konnten. Beiträge in verschiedenen Support-Foren deuten darauf hin, dass der LoRa-Transceiver-Chip SX1276 „einfrieren“ und dieses Verhalten verursachen könnte, indem er sowohl das Funkgerät als auch die MCU blockiert, wenn er über längere Zeit läuft. Dieses Verhalten wurde bei den Display- oder Sensorstationen aber nie festgestellt, da diese Module ja häufig ein- und ausgeschaltet werden.

Die Lösung für den Pico bestand darin, eine zweite MCU, einen Arduino Mini, als intelligenten Timer zu verwenden. Der Ausgang des Mini wurde mit dem Enable-Pin des Pico verbunden und schaltete den Spannungsregler jede Stunde für 500 ms ab. Ein Ausgang des Pico wurde mit dem Mini verbunden, um anzuzeigen, wann der Datenfluss am wenigsten anfällig für die Auswirkungen eines Resets ist: Der Hard-Reset würde nur direkt nach einer Datenaktualisierung erfolgen, wenn der Pico keine neuen Informationen empfängt. Auch wenn dies eine extravagante Verwendung einer MCU zu sein scheint, sind diese Boards so billig, dass sie gerechtfertigt ist. Ich bitte die Elektor-Community um Anregungen und Ideen, wie dieses Problem besser gelöst werden könnte. Es werden zwei Versionen der Home-Station-Software zur Verfügung gestellt [2], eine für die Verwendung mit dem Processing-Programm und eine, die unabhängig von einer Computerverbindung ist und nur eine Stromversorgung benötigt, zum Beispiel ein altes Handy-Ladegerät mit dem entsprechenden USB-Anschluss. Der Processing-Code ist noch in Arbeit und wird zurzeit nicht als vollständige, fehlerfreie Lösung zur Verfügung gestellt.

Bewitterung

Anfänglich fiel mein Außensensor bei schnellen Temperatur- und Feuchtigkeitsschwankungen bisweilen aus, und ich vermutete Kondensation als Fehlerquelle, nachdem ich alle Lötstellen und Kabel untersucht und keine Fehler gefunden hatte. Ich beschloss deshalb, meine Außenplatine zu vergießen. Ich druckte einen 3D-Behälter für meine Platine und einen kleinen „Damm“ um die Sensorplatine herum, damit sie nicht in das Epoxidharz tauchen würde (**Bild 8**). Ich schloss zudem einen 90°-USB-C-Dongle an, der mir einen Zugriff auf den USB-C-Anschluss der Artemis-Platine gestattet.

Ich habe preiswertes, klares Bastel-Epoxid verwendet, das bei Amazon erhältlich ist. Das Harz härtet sehr langsam aus und gibt dem Benutzer ein großes Zeitfenster für die Arbeit mit dem Material. Da ich kein ästhetisch perfektes Ergebnis anstrebte und auch nicht wollte, dass meine idiotischen Katzen sich mit Epoxidharz bekleckern, stellte ich den Behälter bei etwa 65°C auf einen elekt-

Stückliste

Widerstände:
R1 = siehe Text
R2…R4 = 1 M; 1/4 W
R5 = 680 Ω; 1/4 W
R6…R8 = 360 Ω; 1/4 W

Halbleiter:
D1 = LED blau
D2 = LED rot
D3 = LED gelb
D4 = LED grün

Boards:
2 × TPL5111-Breakout-Platinen (Adafruit Industries)
1 × Raspberry Pi Pico
1 × Artemis Thing Plus (Sparkfun)
1 × Artemis Nano (Sparkfun)
2 × BME280-Breakout-Boards (ASIN B08DHTGNHR *)
1 × FTDI-Basis-Breakout 3,3 V (Sparkfun)
3 × FCC-zertifizierter LoRa-Transceiver HopeRF RFM95CW 868 MHz (Lynx-dev.com oder ineltek.com)
1 × Arduino Pro Mini, 3,3 V, 8 MHz (siehe Text)

Außerdem:
2 × ST 4-Pin-Stecker (ASIN B01DUC1M2O*; siehe Baubeschreibung)
1 × 4,2"-ePaper-Display (WaveShare; ASIN B074NR1SW2*)
3 × doppelseitige FR4-Leiterplatten 7 × 9 cm (ASIN B08F7X8JHV*)
2 × wiederaufladbare 18560 LiPo-Akkus
2 × Batteriehalter für 18650er-Akkus
Epoxidharz-Kit (ASIN B07TVWTG829*)
USB-C 90°-Adapter (ASIN B0BBVWF54L*)
Header und Drähte nach Bedarf

* ASIN-Codes sind durchsuchbare Amazon-Inventarnummern

rischen Lebensmittelwärmer. Das Epoxidharz war innerhalb einer Stunde katzenpfotenfest! Dann ließ ich es über Nacht bei Raumtemperatur (18…20°C) aushärten. Die Platine befindet sich nun in einem belüfteten, weiß gestrichenen Plastikgefäß (**Bild 9**), und die Belüftungslöcher sind abgeschirmt, um den Sensor vor Ungeziefer (damit meine ich NICHT die Katzen!) zu schützen.
Bevor ich die Schaltung vergoss, habe ich mich natürlich der hundertprozentig korrekten Funktion der Platine versichert und auch durch Tests bestätigt, dass das Epoxidharz weder blanke Drähte kurzschließen noch elektrische Verbindungen unterbrechen würde. Seit ich den Schaltkreis vergossen habe, arbeitet die Außeneinheit zufriedenstellend und zeigt keinerlei Ausfallserscheinungen mehr.

Software und Daten

Die Software für alle drei Stationen verwendet LoRa-Bibliotheken und WaveShare-Code. Für die Innen- und Außenstation werden Bibliotheken für den BME280-Sensorchip benötigt. Sie können die kompletten Pakete unter [2] kostenlos herunterladen. Der Programmieraufwand für den TPL5111 ist gering. Ich habe eine garantierte Datenkommunikation gegen die Einfachheit des Codes und verkürzte T_{ON}–Zeiten eingetauscht. Nach der Messung wird ein kurzer String erstellt, der die Sensormesswerte und die Identifikationsdaten der Sendestation enthält. Dieser String wird zweimal gesendet (siehe unten). Die Übertragungen von der Home-Station zur Display-Station enthalten auch Zeitdaten, so dass auch der Zeitpunkt der letzten Aktualisierung angezeigt werden kann.
Der Code könnte Ihnen unvollständig erscheinen. Bedenken Sie jedoch, dass die Codeausführung angehalten wird, wenn der Spannungsregler ausgeschaltet wird. Um dies zu erreichen, sendet die MCU ein *DONE*-Signal an den Timer. Dadurch werden alle „Lücken" in der Software geschlossen. Mit Ausnahme des WaveShare-Codes habe ich keine externen Bibliotheken in den Software-Download aufgenommen, so dass diese über die Arduino-IDE zu Ihrer Umgebung hinzugefügt werden müssen. Ich habe die BME280-Bibliothek von Sparkfun verwendet, weil es außer der Datei *Wire.h* keine weiteren Code-Abhängigkeiten gibt. Die LoRa-Bibliothek von *Sandeep Mistry* bietet alle notwendigen Funktionen und hat außer der Datei *SPI.h* keine weiteren Code-Abhängigkeiten.
Da die Home-Station die Übertragungen zwischen den Sensoren und der Anzeigestation nicht koordinieren kann, sind Datenkollisionen mit daraus resultierendem Datenverlust unvermeidlich. Der Datenstring wird zweimal mit einer kleinen Pause zwischen den Übertragungen gesendet, um das Risiko von verlorenen Daten zu verringern. Letztendlich ist der Verlust von ein paar Datenpunkten pro Tag in dieser unkritischen Anwendung akzeptabel.
Während der Entwicklung wurde mir klar, dass die Home-Station mit einem Computer kommunizieren und die Wetterbeobachtungen an eine Datenbank, eine Webseite oder ähnliches senden könnte. Daran wird derzeit gearbeitet. Ich habe einen noch „groben" Webserver in Processing geschrieben, einer von Java abgeleiteten Sprache mit einer breiten Benutzerbasis und vielen guten Bibliotheken. Wie gesagt, der Processing-Code wird irgendwann zur Verfügung gestellt werden, ebenso wie die Arduino-Sketches. Obwohl es einige oberflächliche Ähnlichkeiten geben mag, ist dies aber kein LoRaWan-Projekt, und die Heimstation ist kein LoRa-Gateway.
In meiner Konfiguration werden die Außenbedingungen, die Bedingungen in meiner Küche (wo sich das Display befindet) und ein dritter Sensor in unserem Windfang (Mudroom sagt man bei uns) überwacht, in dem im Winter Frostgefahr besteht. Die Sensordaten aus dem Windfang werden nur auf der Webseite angezeigt. Es gibt Obergrenzen für die Anzahl der Sensoren und Anzeigen, die von dieser Anwendung unterstützt werden können, denn wenn der Netzwerkverkehr zunimmt, werden Datenverluste aufgrund von Kollisionen irgendwann inakzeptabel groß. Dies hängt von vielen Faktoren ab, unter anderem von der Genauigkeit des Timing-Widerstands des TPL5111, von Umgebungsfaktoren und vor allem von der Häufigkeit der Sampling-

Hinweise zum Bau

1. **R1 und R3 entfernen**: Siehe Bilder zur Identifizierung der Widerstände R1 und R3, die vom Nano beziehungsweise Thing entfernt werden müssen. Andernfalls funktionieren die Schaltungen nicht.

2. **ENABLE-Anschluss am Artemis-Nano**: Ich habe ein Bild 5 von der Unterseite der Platine gemacht. Sie sollten den Pin NICHT auf Massepotential legen!

3. **Hauptplatine**: Es kann jede beliebige Lochrasterplatine verwendet werden, aber meine war vorverzinnt, FR4, ausreichend groß und hielt einer Nachbearbeitung stand.

4. **Funkfrequenz**: Kaufen Sie eine für Ihr Land zugelassene Funkplatine.

5. **Anschluss an das Qwiic-System**: Der erwähnte JST-Stecker passt perfekt zum QWIIC-System, aber die Farben entsprechen NICHT der Sparkfun-Konvention. Für diese Stecker gilt: weiß = GND, gelb = 3,3 V, schwarz = SDA und rot = SCL.

6. **BME280-Platinen**: Seien Sie vorsichtig beim Kauf von BME-Boards. Einige Amazon-Verkäufer ersetzen (wissentlich oder unwissentlich) BM**E**280- durch BM**P**280-Chips. Die BMP-Version ist billiger und sieht fast genauso aus, aber die Bibliotheken für BME funktionieren nicht und Sie erhalten keine Feuchtigkeitsdaten.

7. **Das WaveShare-Display** wird mit einem einfach anzuschließenden Kabel geliefert. Lila = BUSY, Weiß = RESET, Grün = DC, Orange = CS, Gelb = CLK, Blau = DIN (MOSI), Braun = GND, Grau = Vcc. Das Display besitzt keinen MISO-Anschluss.

8. **Zeitintervall des TPL5111**: Das Timer-Intervall des TPL5111 wird entweder mit einem On-Board-Poti oder einem externen Widerstand programmiert. Auf der Adafruit-Website und im IC-Datenblatt finden Sie eine Tabelle mit Widerstandswerten für verschiedene Zeitintervalle. Wenn Sie einen externen Widerstand verwenden, müssen Sie eine Leiterbahn auf der Rückseite der Platine auftrennen. Sollte die Stromversorgung problematisch sein, kann die Aktivitäts-LED aus der Schaltung entfernt werden, indem eine Leiterbahn aufgetrennt wird.

9. **Batterieanschluss an der Artemis-Platine**: Der Akku wird mit einem handelsüblichen 2-adrigen JST-Kabel an die Platine angeschlossen.

10. **Funk-Breakout-Board**: Ich habe ein Breakout Board für die Funkeinheit von Diycon.nl verwendet (LoRa Node PCB 100 Shield Only for RFM92/RFM95). Es ist einfach, daran entweder eine Drahtantenne oder einen Antennenstecker anzubringen.

11. **Funkantenne**: Ich habe eine einfache λ/4-Drahtantenne verwendet. Die richtige Drahtlänge für 915 MHz ist 78 mm, für 868 MHz 83 mm, angelötet an den mittleren Anschluss auf dem Funk-Breakout-Board.

12. **Epoxid**: Sehr schmutzig. Handschuhe, Einwegbehälter zum Abmessen und Mischen, Einweg-Mischstäbchen und ein Tuch sind erforderlich. Vergewissern Sie sich, dass die Platine perfekt funktioniert, bevor sie eingegossen wird. Der 90°-USB-C-Adapter muss angeschlossen sein und der Timer sollte den richtigen Widerstand haben. Das flache Gefäß muss etwas größer als die Platine inklusive der Batterie sein. Tauchen Sie den BME280 nicht in Epoxid ein. Ich habe einen „Damm" um den Sensor gelegt (das Epoxid kommt durch die Perforationslöcher hoch, also auch den Boden schützen). Alternativ können Sie den Sensor über Drähte mit dem QWIIC-Anschluss verbinden und die Sensorplatine über das Epoxidharz halten, während es aushärtet. Ein leichtes Erwärmen des Epoxidharzes beschleunigt die Aushärtungszeit erheblich. Die drei wichtigsten Punkte sind, dass der Sensor nicht mit Epoxid bedeckt ist, der USB-C-Anschluss durch den Adapter zugänglich bleibt und dass mit Ausnahme der Sensorplatine alle Komponenten auf der Platine bedeckt sind.

13. **Das Gehäuse**: Es gibt mehrere beachtenswerte Details: Die Öffnungen für die Belüftung sollten so angeordnet und geschützt sein, dass Wasser entgegen der Schwerkraft nach oben fließen müsste, um in das Innere des Gehäuses zu gelangen. Zweitens sollte das Gehäuse reflektierend oder zumindest weiß gestrichen sein, um den Treibhauseffekt zu minimieren. Drittens sollte das Gehäuse leicht sein. Die thermische Trägheit eines großen, schweren Gehäuses macht Ihre Messungen langsam und ungenau. Und schließlich sollte das Gehäuse in irgendeiner Weise insektengeschützt sein.

14. **18560er-Batterien**: Auf Amazon und eBay sind Batterien allgegenwärtig, die mit einer unmöglichen Energiedichte zu lächerlich niedrigen Preisen prahlen. Seien Sie skeptisch gegenüber diesen Behauptungen, Markenzellen mit 2.500…3.500 mAh sind eine sichere Wahl!

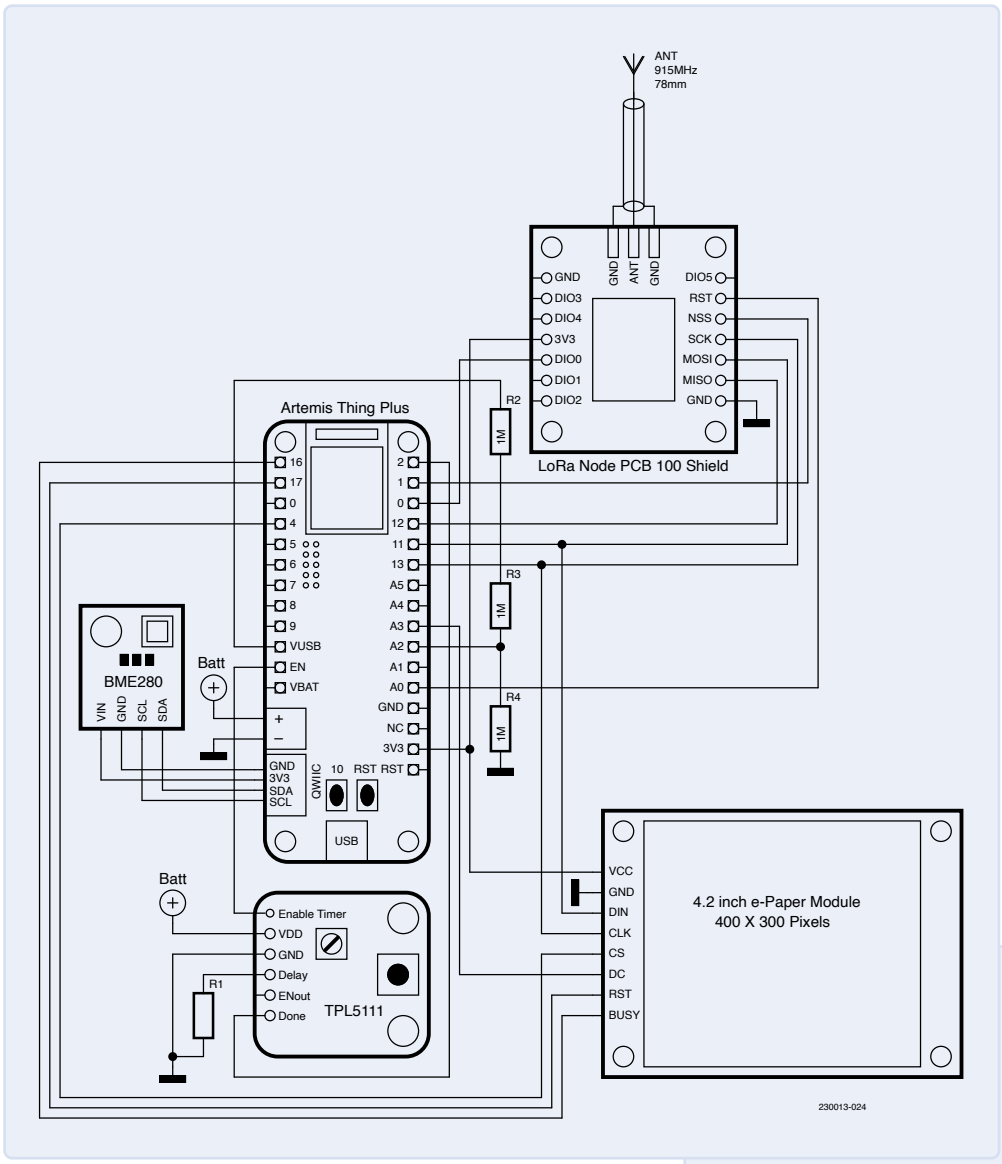


Bild 10. Schaltung der Display-Station.

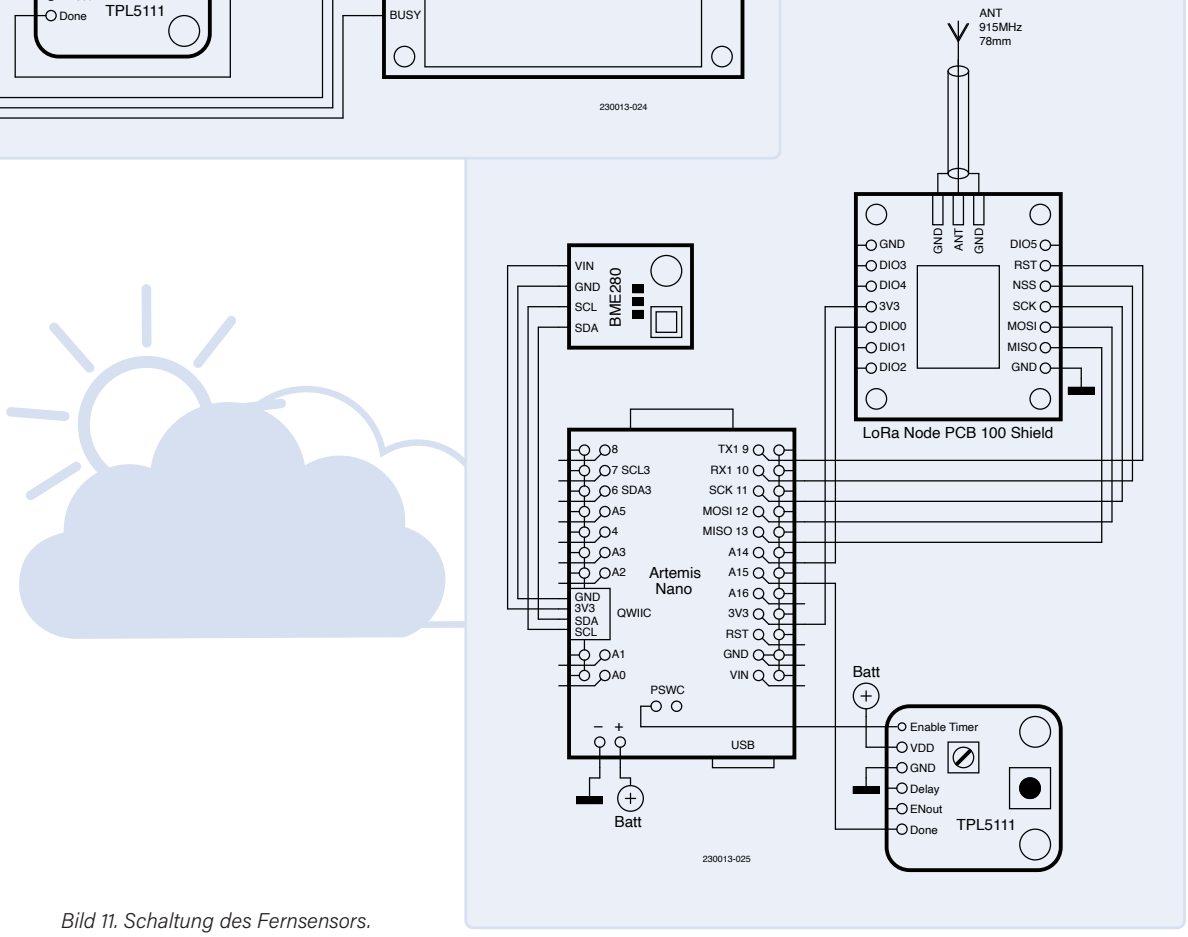


Bild 11. Schaltung des Fernsensors.

und/oder Aktualisierungsanfragen. Es gibt aber keine Obergrenze für die Anzahl der Displays, die ja keine Datenanfragen stellen, denn nur Geräte, die Datenanforderungen stellen, tragen zum Funkverkehr bei. Der gesamte Funkverkehr und nicht die Übertragungen von Sensormessungen begrenzen das Netzwerk. Die Berechnung der theoretischen Grenzen verschiedener Konfigurationen bleibt Ihnen als Übung überlassen.

Praktische Überlegungen

Die Hardware ist handverdrahtet. Lesen Sie die Hinweise zum Bau im **Kasten Hinweise zum Bau** sorgfältig durch. Die Sensoren kommunizieren über I²C. Ich habe das praktische Qwiic-System von Sparkfun verwendet, das mit Qwiic-kompatiblen Steckern von Amazon oder eBay mit den kleinen BME280-Platinen verbunden werden kann. Achten Sie darauf, dass Sie einen BM**E**280 und nicht einen BM**P**280 kaufen, wenn Sie Feuchtigkeitsinformationen wünschen. Das HopeRF-Modul hat ein Raster von 2 mm und nicht von 2,54 mm, was ein Breakout-Board oder sorgfältiges Löten erfordert. Ich verwende eine einfache ¼-Drahtantenne.

Ich habe mit *FontEdit* eine 36-Punkt-Schrift für das ePaper-Display erstellt. Die Anweisungen von WaveShare sind nur schwer zu verstehen, aber letztendlich habe ich einen funktionierenden Code entwickelt. Wenn Sie eine andere MCU als die Artemis-Boards verwenden, sollten Sie berücksichtigen, dass die Schriftarttabellen und der Bitmap-Puffer eine Menge RAM verbrauchen.

Ich habe eine wiederaufladbare Lithium-18560-Batterie als Stromquelle verwendet. In den Bauanmerkungen finden Sie Hinweise zur Beschaffung von 18560ern. Andere Stromversorgungskonfigurationen sind möglich, aber Alkalibatterien werden für Temperaturen weit unter dem Gefrierpunkt ausdrücklich nicht empfohlen.

Die Schaltpläne (siehe **Bilder 10...12**) sind nur ein Vorschlag. Die beliebten neuen 32-Bit-Boards sind flexibel, besitzen viele Pins, die Interrupts unterstützen und verfügen über alternative SPI-, I²C- und UART-Schnittstellen. Daher wurden einige Verbindungen so wie gezeigt gewählt, um das physischen Layout der Gesamtschaltung zu optimieren.

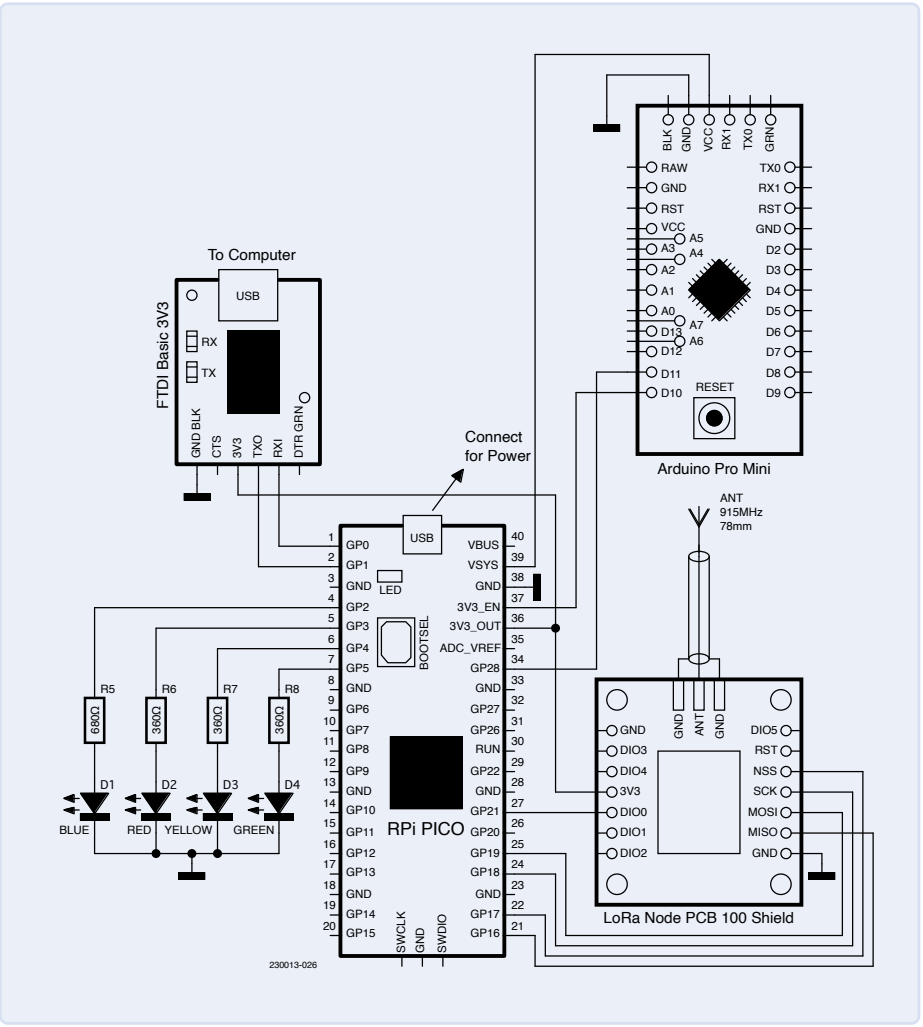


Bild 12. Schaltung der Home-Station.

Viel Spaß beim Nach- und Ausbau des Systems!

RG — 230013-02

Über den Autor

Ed Ringel ist ein fast-pensionierter Arzt für Atemwegserkrankungen und Intensivmedizin. Er genießt in Maine die Natur mit seiner Frau, schreibt Science-Fiction, bastelt mit seinem 3D-Drucker und entwickelt Elektronikprojekte mit MCUs.

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Schreiben Sie dem Elektor-Team bitte eine E-Mail an redaktion@elektor.de.

Passende Produkte

- **SparkFun Sensor-Kit**
  https://elektor.de/19620
- **Raspberry Pi Pico RP2040**
  https://elektor.de/19562

WEBLINKS

[1] Erfolglose Abenteuer: https://forums.adafruit.com/viewtopic.php?p=927526
[2] Dieses Projekt auf Elektor Labs: https://elektormagazine.de/labs/low-power-lora-weather-station

Transverter für das 70-cm-Band

Von Jan Buiting PE1CSI (Elektor)

1981 lockte Elektor die Amateurfunkgemeinde mit einem schön konstruierten Sender-Empfänger-Umsetzer (Transverter) für das 70-cm-Band, damals ein ziemlich freier Bereich des Funkspektrums, der von echten Experimentatoren für die Kommunikation über den Äther genutzt wurde - ohne Handys, können Sie sich das vorstellen?

Prototyp des 70-cm-Transverters, der 1981 von Gerrit, PA0HKD und Ed, PE1CJP im Elektor-Labor gebaut wurde. Herz des Transverters ist ein Icom IC211 2-m Allmode-Transceiver.

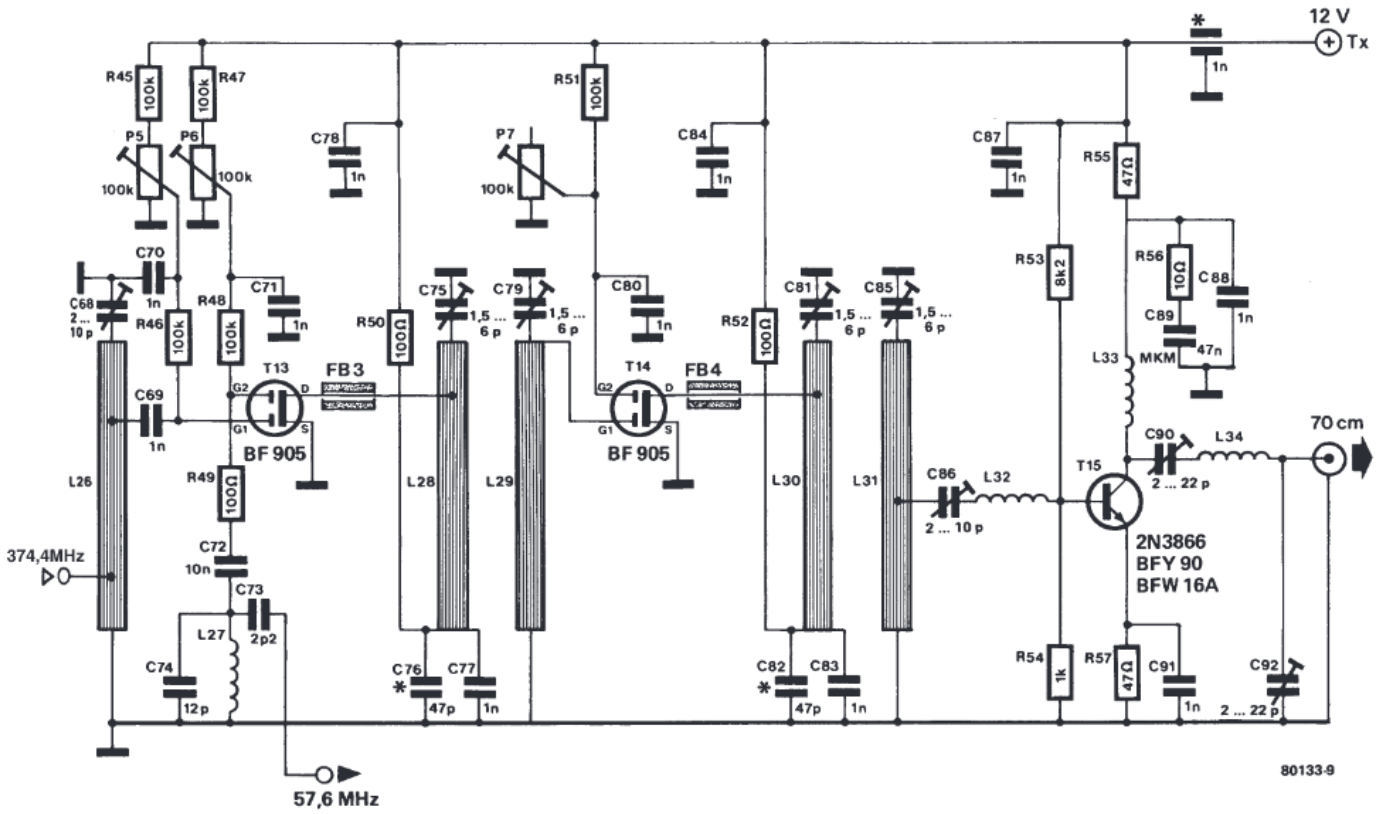
Der 70-cm-Transverter (430...440 MHz), an den ich hier erinnere, wurde in zwei Teilen in Elektor Juni 1981 veröffentlicht [1]. Er ist ein gutes Beispiel für eine Veröffentlichung, die sich an Funkamateure richtete, die nicht bereit waren, die damals exorbitanten Preise für kommerzielle Geräte zu zahlen. Sie wollten einfach SSB (Single-Sideband) auf 70 cm, wie sie es schon seit vielen Jahren auf Kurzwelle und 2 m (144-146 MHz) genießen konnten. Im Gegensatz zu UKW ist SSB eine lineare Betriebsart, die eine perfekte Linearität aller Senderstufen bis hin zum Antennenanschluss erfordert.

Vorsicht, Amateure am Werk!

Der ursprüngliche Entwurf für den Transverter stammt von J. de Winter PE0PJW. Im Elektor-Labor haben Mitarbeiter Gerrit Dam PA0HKD und Laborpraktikant Ed Warnier (PE1CJP, jetzt PA1AW) den Entwurf so weit optimiert und elektorisiert, dass er für Elektor-Leser reproduzierbar war und die gesetzlichen Bestimmungen in Bezug auf Störaussendungen und Oberwellen erfüllte. Ed erinnerte sich daran, dass es eine „Herausforderung" war, ein Ultrahochfrequenz-Design (UHF) auf einer Platine zu realisieren. Er musste sich mit den Platinendesignern bei Elektor herumschlagen, die mit den Unwägbarkeiten von Sub-pF-Streukapazitäten und Nanospulen nicht vertraut waren und sich nur mit Gleichstromzeugs wie Audio, Mikrocontrollern und Netzteilen auskannten. Letztendlich fand man eine zufällige Lösung in der Verwendung von auf die Platine geätzten Mikrostreifenleitungsdrosseln im UHF-Bereich der Schaltung, nicht zu vergessen eine versilberte Platine und natürlich verzinnte Blechabschottungen auf der Platine, um die Störstrahlung auf ein Minimum zu reduzieren.

Laut und deutlich

In den frühen 1980er Jahren hatte das 70-cm-Band eine besondere Anziehungskraft, nicht nur als Treffpunkt für Funkamateure mit zu 100% selbstgebauten Anlagen (einschließlich

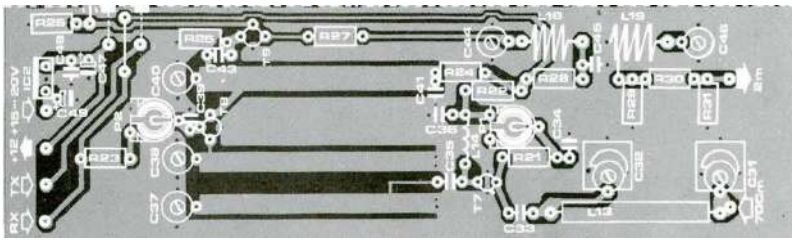


Schaltplan nur für den UHF-Teil des Transverters. Diese „Mikrostreifenleitungsdrosseln" erforderten 1981 die Erfindung eines neuen Symbols durch die damalige Elektor-Zeichenabteilung sowie die Aufnahme eines Wortes in das technische Wörterbuch der Redaktion.

Amateurfernsehen, ATV), sondern auch für die Satellitenkommunikation, die kontinentübergreifende QSOs in CW und SSB mit relativ geringen Sendeleistungen und stark gerichteten Antennen ermöglichte.

Der 70-cm-Transverter von Elektor erreichte einen hohen Q-Faktor bei den Funkamateuren und wurde genau auf ihre Bedürfnisse abgestimmt. Das Projekt wurde brillant ausgeführt und in zwei Artikeln in einer Elektor-Ausgabe glorreich dokumentiert, was zu einer UFB-Publikation (*Ultra-fine Business*) führte.

220214-02



Ein reproduzierter Ausschnitt der Transverterplatine. Die rechteckigen Bereiche zwischen den dicken schwarzen Linien sind keine großen, fetten Kurzschlüsse, sondern abgestimmte Induktivitäten im „Microstrip-Stil", die bei etwa 430 MHz arbeiten. So etwas konnte man damals nicht einfach drag & drop und automatisch routen!

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Schicken Sie dem Autor eine E-Mail an jan.buiting@elektor.com.

WEBLINKS

[1] J. de Winter, „70-cm-Transverter", Elektor 6/1981
Teil 1: https://www.elektormagazine.de/magazine/elektor-198106/46257
Teil 2: https://www.elektormagazine.de/magazine/elektor-198106/46265

Climate Calling Engineers

Move Fast and Fix Things

By Priscilla Haring-Kuipers

The sixth synthesis report by the Intergovernmental Panel on Climate Change (IPCC) is clear: "Human activities, principally through emissions of greenhouse gases, have unequivocally caused global warming." What are we going to do about it?

The sixth synthesis report by the Intergovernmental Panel on Climate Change (IPCC) is the collective scientific wisdom on climate change and how to fix it. [1] They inform the UN. The main message of their latest report has hope: "There are multiple, feasible and effective options to reduce greenhouse gas emissions and adapt to human-caused climate change, and they are available now," [2] but currently, we are not applying the technical solutions we have with enough vigour, scale or speed.

Calling all engineers! There is a lot to do, so let me give you some highlights:

Current Climate

We are now at 1.1°C global warming and will likely reach 1.5°C in the early 2030s and shoot up to 3.5°C this century if we don't change drastically. "There is a rapidly closing window of opportunity to secure a liveable and sustainable future for all." [1] We have already caused a lot of damage across ecosystems. More than climate scientists estimated earlier. We have lost many species, nearly 50% of coastal wetlands, and we are impacting ecosystems in ways that are not reversible. Cities have become hotter and the air we breathe more polluted.

Our supply chain has already been impacted by the more frequent occurring extreme weather, making factories freeze or catch fire. Water is fast becoming a contested resource, and factories should look into either recycling or using seawater.

We have not done nothing. Agreements made at Kyoto and Paris have helped. Social movements have accelerated climate action. We can still save ourselves with climate resilient development based on science, indigenous knowledge and local context. High-tech and low-tech solutions working together.

"Individuals with high socio-economic status contribute disproportionately to emissions, and have the highest potential for emissions reductions, e.g., as citizens, investors, consumers, role models, and professionals." [1] That means us. What we do and what we demand of our governance makes a big difference. What you choose to work on as an engineer will either contribute to a liveable world or to further heating up the place. When we support developing regions with our technological development, they can leapfrog to low-emissions solutions with us.

Stop That

If we are ever to stay under 2°C of global warming, a lot of fossil fuels are going to have to stay in the ground. Today, new fossil fuel developments are still being funded, and the fossil fuel industry receives more money in private investments, public subsidies and tax breaks than developments tackling climate adaptation and mitigation. [1] Simply ending fossil fuel subsidies would lower greenhouse gas emissions with 10% by 2030, while improving public revenue that could be redirected to our necessary transition. If your work or your pension funds are connected to the fossil fuel industry, you might want to start looking for a way to decouple before the well is shut down. Our electronics are heavy on petrochemicals and will be looking to shift to bio-based alternatives. Opportunities abound.

By Priscilla Haring-Kuipers, made with DALL-E: Electronics engineer soldering a product to achieve a future save from climate change.

Carbon pricing such as carbon taxes or emission trading have led to some low-cost emission reduction measures but have not been very successful to promote the higher-cost measures that are necessary to shift an industry. We need more. Luckily, climate laws are increasing, and they are helping to fight climate change causes and effects. Climate-related litigation is growing and has already had an effect on the "outcome and ambition of climate governance." [1] It is likely that climate law will grow in the near future, both internationally and on regional levels. The WEEE regulations and the Supply Chain Act are early versions of what is coming. Your efforts and your company can be ahead of the curve, riding the green wave, or you can be dragged along by legislation, but everyone is coming eventually.

Technology for the Win

"If all technically available options were used, global emissions could be at least halved by 2030, at manageable costs." [1] We need tons of engineers to roll out, scale up, improve and adapt to local circumstances many of the already available and proven solutions.

In the last decade, the cost of solar energy has dropped by 85%, wind energy by 55% and lithium-ion batteries by 85%. Meanwhile, deployment has increased over tenfold for solar and over a hundredfold for electric vehicles. In some areas and industries, keeping the old is becoming more expensive than changing to the new. Work on whatever you can to push, develop and spread this development.

Green energy will not only curb our emission, but the economic benefit in air quality alone would offset the cost of the transition. Co-development of energy efficiency and renewable energy will create a happy feedback loop of improvement. Work on big renewables and small-scale nets, smart-grids, transmission and capacity is very much needed.

Cities are critical in this transition. We can build or retrofit to match our new low-emission lifestyles and make space for cycling and walking, teleworking and electric public transport. More plants and water in cities would help cool during heatwaves, process heavy rainfall slower and keep moist during droughts while benefiting the health and well-being of all who live there. Engineers should work on building materials and practices, sustainable urban planning and maintenance, digital communities and smart transport solutions. Many cities have already announced a net-zero emissions target. My city started a green jobs market for all the technical roles we are going to need to develop, install and maintain this bright new future. Your city might have a similar initiative.

Carbon capture is most reliably done by reforestation, improved forest management, soil carbon sequestration, peatland restoration and coastal blue carbon management. Protecting high-carbon ecosystems would have an immediate impact. Globally, we need to protect 30-50% of our land and water to maintain a resilient biodiversity. Throw your skill set behind any project that supports conservation and restoration.

Your time is now.

230265-01

No Geo-Engineering

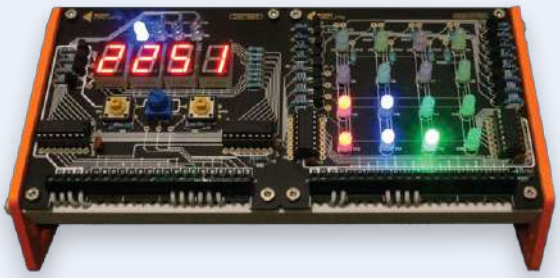
Blocking the sun with solar shields or sulfur is a no-go. Short-term and local cooling effects are likely, but the amount of green house gases would still grow and the acidification of our oceans would continue. We don't know enough about the effects on the targeted region nor our global ecology. Once it is up and running, turning it off could cause "rapid climate change." The risks are too great, and the reward is too uncertain.

WEB LINKS

[1] IPCC, "AR6 Synthesis Report: Climate Change 2023," 2023: https://www.ipcc.ch/report/ar6/syr/

[2] IPCC, "Urgent climate action can secure a liveable future for all," March 20, 2023: https://www.ipcc.ch/2023/03/20/press-release-ar6-synthesis-report/

Short Circuits: Das 4er-Pack (Arduino-kompatible Elektronik-Plattform)

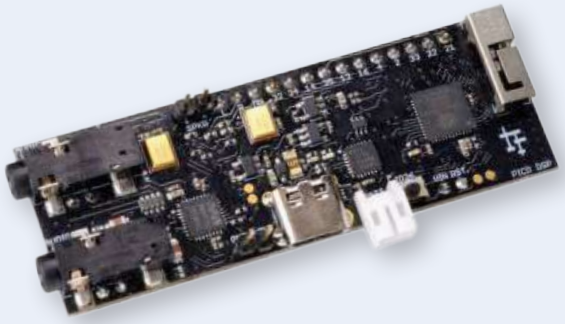


Preis: 99,95 €

Mitgliederpreis: 89,96 €

www.elektor.de/20474

PÚCA DSP ESP32 Development Board

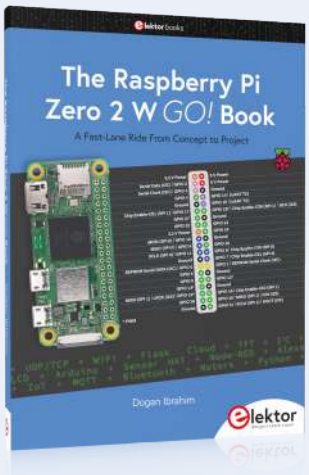


Preis: 69,95 €

Mitgliederpreis: 62,96 €

www.elektor.de/20504

The Raspberry Pi Zero 2 W GO! Book



Preis: 34,95 €

Mitgliederpreis: 31,46 €

www.elektor.de/20445

The Elektor Power Supply Collection (USB-Stick)

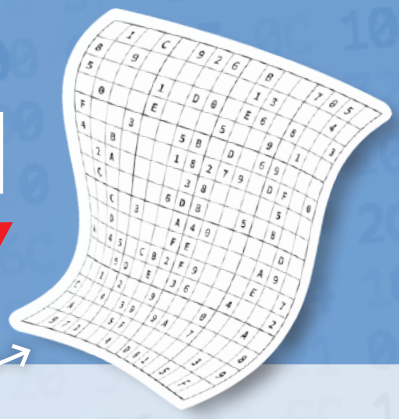
Preis: ~~49,95 €~~

Sonderpreis: 34,95 €

www.elektor.de/20451

Hexadoku

Sudoku für Elektroniker



Wie in jeder Ausgabe finden Sie auch in diesem Heft unser ganz spezielles Sudoku. PC, Oszilloskop und Lötkolben können sich erholen, während Ihre kleinen grauen Zellen auf Hochtouren arbeiten. Wenn Sie alle Hex-Ziffern in den grauen Kästchen herausgefunden haben, sollten Sie uns diese gleich zumailen – denn hier warten fünf Elektor-Gutscheine!

Die Regeln dieses Rätsels sind ganz einfach zu verstehen: Bei einem Hexadoku werden die Hexadezimalzahlen 0 bis F verwendet, was für Elektroniker und Programmierer ja durchaus passend ist. Füllen Sie das Diagramm mit seinen 16 x 16 Kästchen so aus, dass alle Hexadezimalzahlen von 0 bis F (also 0 bis 9 und A bis F) in jeder Reihe, jeder Spalte und in jedem Fach mit 4 x 4 Kästchen (markiert durch die dickeren schwarzen Linien) **genau einmal** vorkommen. Einige Zahlen sind bereits eingetragen, was die Ausgangssituation des Rätsels bestimmt. Wer das Rätsel löst – sprich die Zahlen in den grauen Kästchen herausfindet – kann einen von fünf Gutscheinen im Wert von 50 Euro gewinnen!

EINSENDEN

Schicken Sie die Lösung (die Zahlen in den grauen Kästchen) per E-Mail oder Post an:

Elektor Redaktion
Lukasstraße 1
52070 Aachen

E-Mail: hexadoku@elektor.de

Als Betreff bitte nur die Ziffern der Lösung angeben!

Einsendeschluss ist der 15. August 2023.

DIE GEWINNER DES HEXADOKUS AUS DER AUSGABE MAI/JUNI STEHEN FEST!

Die richtige Lösung ist: **EB0C8**.

Aus allen Einsendungen mit der richtigen Lösung haben wir die fünf Gewinner eines Elektor-Wertgutscheins über je 50 € gezogen.

Die Namen der Gewinner werden unter www.elektormagazine.de/hexadoku bekannt gegeben.

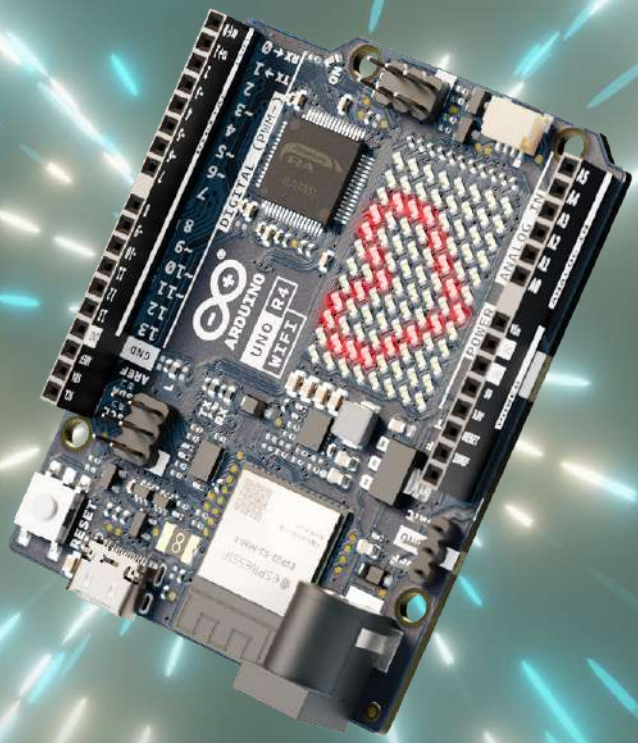
Herzlichen Glückwunsch!

D		1	6	0		2	7		A						B
	3							0	1	8	2	7			
5		F					4	3	B		D			0	A
A					3	5					F			4	
3				4			1	9	C			E			
			1		B			6					C	8	
6			5			9	0	A	8				D	F	
0		A		C		D				1		4	5	6	7
	D	6		8	4	F			E			2			
2	E	8		5		0		7	4			6	1		
	F						6			2	B			5	
	5	0	4							9		C		E	8
	6			9			3	C	7		0				
					5	4	2		9						
		5	2		0	E	A			D	8			B	
8		9					F				4				6

E	2	5	A	0	3	7	B	1	F	8	C	6	4	9	D
6	C	D	F	1	5	8	9	3	2	4	0	E	B	A	7
B	0	1	7	6	E	4	F	D	9	5	A	C	8	3	2
8	3	4	9	2	A	C	D	6	7	B	E	F	0	1	5
D	5	F	8	4	B	A	7	C	1	6	3	0	E	2	9
7	4	2	3	E	0	1	C	F	5	9	B	D	6	8	A
9	6	B	0	3	D	F	8	2	A	E	7	5	1	4	C
A	1	C	E	5	2	9	6	4	8	0	D	7	3	B	F
C	7	8	2	A	9	D	3	0	E	1	6	B	F	5	4
F	D	A	1	7	6	0	2	5	B	3	4	8	9	C	E
3	E	9	6	B	8	5	4	A	C	7	F	1	2	D	0
4	B	0	5	C	F	E	1	8	D	2	9	3	A	7	6
0	9	E	B	D	1	2	A	7	3	F	5	4	C	6	8
1	A	7	D	9	4	6	E	B	0	C	8	2	5	F	3
2	8	6	C	F	7	3	5	E	4	A	1	9	D	0	B
5	F	3	4	8	C	B	0	9	6	D	2	A	7	E	1

ARDUINO

UNO R4



Die neue Dimension für ‘Makers’

Erweitern Sie mit diesem benutzerfreundlichen und leicht zugänglichen Mikrocontroller die Grenzen des Machbaren!

Schnellerer und größerer Speicher - 48 MHz, 256 kB Flash, 32 kB RAM - perfekt, um präzisere Berechnungen durchzuführen und komplexere Projekte zu bearbeiten.

Neue eingebaute Peripheriegeräte - Der RA4M1 (Arm® Cortex®-M4 Core mit FPU) verfügt über einen DAC, CAN-BUS, HID-Unterstützung und OpAMP Embedded, was fortgeschrittenere Projekte ermöglicht.

Erweitern Sie Ihre Projekte mit dem kompatiblen UNO Shield (5 V kompatibel) und dem riesigen Katalog kompatibler *Qwiic-Knoten.

Konnektivität *ESP32-S3 Coprozessor - Wi-Fi® und Bluetooth® Low Energy Konnektivität, wodurch der RA4M1 Mikrocontroller frei bleibt.

Robustes Design - mit einem größeren Spannungsbereich, 6 - 24 V, können Sie den UNO R4 jetzt in Kombination mit Motoren, LED-Streifen oder anderen Aktoren über eine einzige Stromversorgung verwenden.

*Hinweis - diese Funktionen sind nur für das UNO R4 WiFi Modell verfügbar

UNO R4 WiFi

UNO R4 Minima

Die größte Auswahl elektronischer Bauelemente™

Auf Lager und versandfertig

mouser.de

MOUSER ELECTRONICS