G3078 E (D) € 14,90 | CHF 27,00 | (A, B, L) € 16,00

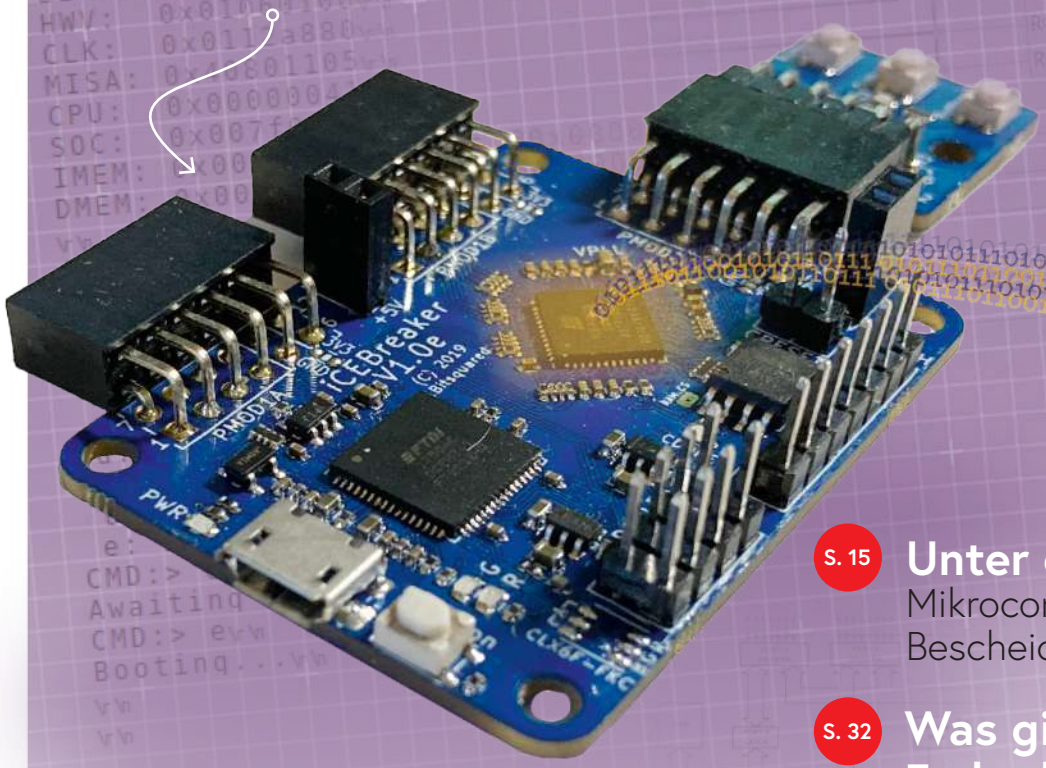
elektorMAG
design > share > earn

embedded world Special
* SEIT 1961 *

JUNI/JULI 2022
ELEKTORMAGAZINE.DE

embedded world Special

Mein eigener RISC-V-Controller



IM FOKUS Embedded-Entwicklung und IoT

embeddedworld2022
Exhibition&Conference

IMPRESSUM

53. Jahrgang, embedded world Special
Juni/Juli 2022
ISSN 0932-5468

Erscheinungsweise: 8x jährlich

Verlag
Elektor Verlag GmbH
Kackertstraße 10
52072 Aachen
Tel. 0241 95509190

Technische Fragen bitten wir per E-Mail an redaktion@elektor.de zu richten.

Hauptsitz des Verlags
Elektor International Media
Postbus 11, 6114 ZG Susteren
Niederlande

Anzeigen
Raoul Morreau (Leitung)
Mobil: +31 6 440 399 07
E-Mail: raoul.morreau@elektor.com

Büsra Kas
Tel. 0241 95509178
E-Mail: busra.kas@elektor.com

Es gilt die Anzeigenpreisliste ab 01.01.2022.

Distribution
IPS Pressevertrieb GmbH
Postfach 12 11, 53334 Meckenheim
Tel. 02225 88010
Fax 02225 8801199

Druck
Senefelder Misset, Doetinchem (NL)

EDITORIAL

Von Jens Nickel

Chefredakteur ElektorMag

Durch nichts zu ersetzen

Nach zwei Jahren einschneidender Pandemie-Maßnahmen geht es nun endlich weiter mit „richtigen" Messen. Auf den ersten Events wie der PCIM und der Sensor+Test wurde deutlich, dass das Ganze eher sanft wiederanläuft. Über Gedränge in den Hallen konnte sich niemand beklagen. Auch habe ich dort an vielen Ständen zu hören bekommen, dass Unternehmen ihr Messeprogramm in Zukunft stärker fokussieren wollen: auf eine, höchstens zwei Messen pro Jahr.

Für Elektor und viele unserer Partnerfirmen - die oft aus den Bereichen Halbleiter, Software und Distribution kommen - steht die **embedded world** (neben der electronica) ganz oben auf der Liste. Und so könnte diese Messe zu einem Lackmus-Test werden, wie viel Wert Unternehmen einem Messeauftritt und -Besuch noch beimessen, der stets eine Menge Zeit und Geld kostet.

Wir selbst haben uns in dieser Frage entschieden - genauso wie die Firmen, die in diesem Special vertreten sind. Eine Messe ist durch nichts zu ersetzen! Denn nur hier kann man sich in kurzer Zeit einen Markt-Überblick verschaffen, geniale neue Lösungen entdecken, sich nach Bezugsquellen und Kunden umschauen, Trends ausmachen und Stimmungen einfangen.

Meine Kollegen und ich werden hier in Nürnberg genau dies tun, mit vielen Video-Interviews und einem Newsroom an unserem eigenen Stand. An diesen möchte ich Sie herzlich einladen - wir sind in Halle 4A (4A.646) zu finden!

Unser Team

INHALT

53. Jahrgang
embedded world Special - Juni/Juli 2022

Erste Schritte ins IoT – mit dem ESP32-C3

WLAN-Taster und -Relais

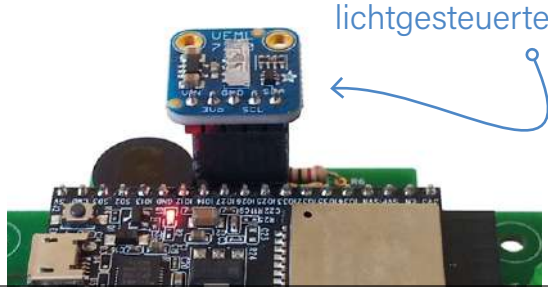
42

Rubriken und Hintergrund

Industry

Lichtschalter DeLux
Lösung für hochpräzises lichtgesteuertes Schalten

66



98

WinUI 3: Neues Grafik-Framework für Windows-Apps

6

Projekte

Vorschau

Elektor Juli/August 2022
Das nächste Heft ist wie immer randvoll gefüllt mit Schaltungsprojekten, Grundlagen sowie Tipps und Tricks für Elektroniker.

Aus dem Inhalt:

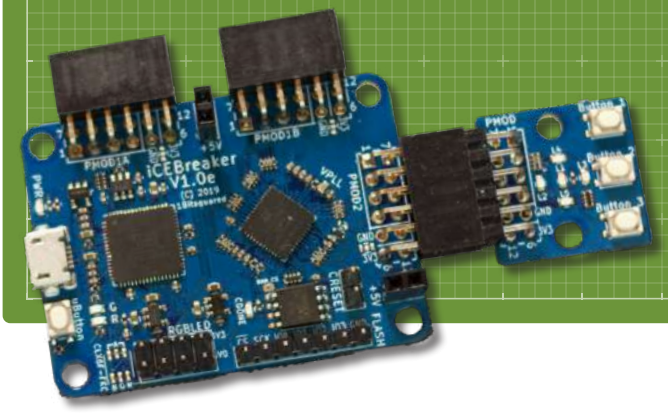
- Autonomes Induktivitäts-Messgerät
- CO_2-Meter mit Sigfox
- Smarte Steckdosen: Ein neugieriger Blick ins Innere
- Einfaches analoges ESR-Meter
- Richtig messen mit dem Oszilloskop
- Raspberry Pi Pico als MSF-SDR
- Netzspannungs-Frequenzmeter

Und vieles mehr!

Elektor Juli/August 2022 erscheint am 7. Juli 2022.
Änderungen vorbehalten.

Mein eigener RISC-V-Controller

Erste Schritte mit dem NEORV32 RISC-V Softcore für preiswerte FPGAs



Von Mathias Claußen (Elektor)

Möchten Sie mit RISC-V experimentieren? Dazu bedarf es keines dedizierten Controllers, es ist auch mit einem preiswerten FPGA mit dem NEORV32-RISC-V-Softcore möglich!

Wer mit RISC-V-Mikrocontrollern anfangen möchte, hat inzwischen die Wahl zwischen einer Vielzahl von Prozessoren, wie zum Beispiel dem neuen ESP32-C3. Doch es muss nicht immer ein fest verdrahteter Chip sein: Das NEORV32-Projekt bietet einen RISC-V-Softcore für FPGAs. Das Ganze ist etwas weniger performant, aber deutlich flexibler, denn es lassen sich unterschiedliche Designs testen und selbst entwickelte Peripherie einbinden. Nebenbei lernt man eine Menge, zum Beispiel über das Innenleben einer CPU.

Auch wer sich schon mit FPGAs auskennt, wird beim Erschaffen eines eigenen kleinen Prozessor-Systems schnell auf Hürden stoßen. Das Ganze stellt auch Experten immer wieder vor Herausforderungen, die gemeistert werden müssen, wie unsere Serie über das SCCC-Projekt von Martin Oßmann [1] zeigt.

Doch man muss nicht bei Null anfangen, sondern kann auch fast schlüsselfertige Lösungen verwenden. Eine solche, noch dazu unter Open-Source-Lizenz stehende Lösung soll hier vorgestellt werden. Sicher eine gute Gelegenheit, das eine oder andere herumliegende FPGA-Board wieder einmal aus der Ecke zu holen. Dieser Artikel kann natürlich bei weitem kein kompletter RISC-V- oder FPGA-Kurs sein, sondern soll die Hürden beim Einrichten der nötigen Tools und dem Erstellen eines ersten Projektes nehmen.

FPGA, Synthese, Softcore, RISC-V und Compiler

Wer eine klassische Mikrocontrolleranwendung entwickelt, sucht für sein Projekt einen Chip eines Herstellers aus, der einen Satz der benötigten Peripherie enthält. Alle Funktionen sind fest im Chip verankert und können nicht verändert werden. Das ermöglicht kostengünstige Chips mit einer optimierten Performance. Bei einem FPGA ist dies anders.

Ein FPGA selbst besteht aus Logikzellen, den Lookup Tabels (LUT), die flexibel miteinander durch eine Matrix verschaltet werden können. Die Blöcke, die in einer solchen LUT vorhanden sind, werden in **Bild 1** gezeigt. Als Beispiel dient hier ein LUT-4-Element mit vier Eingangssignalen, einer Wahrheitstabelle, einem Flip-Flop und einen Multiplexer am Ausgang. Durch die Wahrheitstabelle kann man so ein beliebiges Logisches Element wie zum Beispiel UND, ODER, NICHT oder EXCLUSIV ODER formen. In Verbindung mit der Matrix innerhalb des FPGA lassen sich aus diesen Bausteinen komplexere Gebilde, wie Speicher, Addierer oder Multiplexer schaffen, die wiederum zu einem noch komplexeren System wie einem Prozessor oder einem kompletten System-on-Chip zusammengefügt werden können. Der FPGA lässt sich mit einer Bausteinkiste vergleichen, in der eine definierte Menge an Bausteinen vorhanden sind, die sich immer wieder zu neuen Formen zusammensetzen lassen.

Damit der FPGA eine bestimmte Funktion ausführen kann, muss er passend konfiguriert werden. Es ist jedoch nicht nötig, jede einzelne LUT von Hand anzulegen und in der Matrix zu verbinden, das ist Aufgabe der Synthesetools für den FPGA. Die gewünschte Funktionalität wird in Sprachen wie Verilog oder VHDL beschrieben. Die Synthesetools verstehen in der Regel beide Beschreibungssprachen. Das Synthesetool kennt die Eigenheiten des FPGA und erstellt aus

der Beschreibungssprache am Ende einen Bitstrom, der zur Konfiguration des FPGA verwendet wird. **Bild 2** zeigt den groben Ablauf einer Synthese für einen FPGA. Die meisten Hersteller von FPGAs bieten kostenfreie Tools an, die in der Regel unter Windows und Linux verwendet werden können. Für einige FPGAs gibt es auch Open-Source-Lösungen, die diese Synthese durchführen können und bevorzugt unter unixoiden Betriebssystemen wie Linux oder MacOS laufen.

Ein FPGA mit genug LUTs kann nicht nur einfache Logikfunktionen abbilden, sondern ganze Prozessoren oder Prozessorsysteme, diese werden ebenfalls mit Verilog oder VHDL beschrieben. Da dieser Prozessorkern jedoch nicht fest im Silizium des FPGA verdrahtet ist, sondern sich seine Funktion oder Verhalten durch Modifikation der Hardwarebeschreibung anpassen lässt, wird das Ganze als Softcore bezeichnet. Solche Softcores gibt es für unterschiedliche Prozessorarchitekturen; teilweise enthalten diese Softcores zusätzlich noch Peripherie wie zum Beispiel Bus-Interfaces und dergleichen.
Mehr und mehr kommt bei Softcores die freie Prozessorarchitektur RISC-V zum Einsatz. Da momentan die Auswahl an fest verdrahten RISC-V MCUs noch überschaubar ist, lassen sich auf diese Weise erste Erfahrungen mit der Architektur sammeln. Man erstellt sich so seine eigene RISC-V-MCU und kann diese testen und studieren.

Falls man RISC-V nutzt, fallen keine Lizenzgebühren an und es müssen auch keine NDA oder anderweitige Lizenzabkommen (wie bei anderen Architekturen) abgeschlossen werden. RISC-V bedeutet auch, dass schon Compiler für C-Quelltext vorhanden sind, unter anderem in Form des GNU C-Compilers (GCC). Damit stehen auch grundlegende Bibliotheken zur Verfügung.

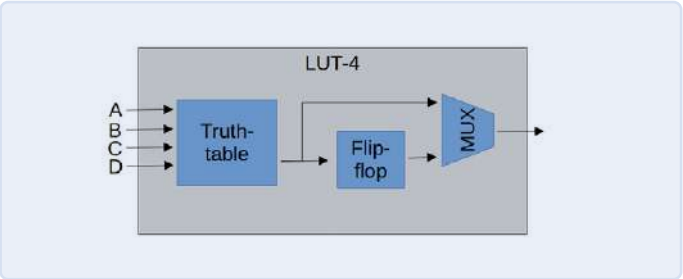


Bild 1. Aufbau eines LUT-4-Elements.

NEORV32

Dass der Weg zum eignen SoC (System on Chip) auf einem preiswerten FPGA nicht schwierig sein muss, zeigt das Projekt NEORV32 von Stephan Nolting [4]. Der NEORV32 ist eine RISC-V-kompatible CPU, die zusätzlich noch Peripherie mitbringt, um als Mikrocontroller-artiges SoC auf einem FPGA zu laufen. Das Projekt ist komplett in plattformneutralen VHDL realisiert und damit nicht an einzelne FPGA-Hersteller gebunden. Der NEORV32 ist nicht nur vollständig open-source, sondern bringt dazu noch eine umfangreiche Dokumentation, ein Software-Framework und Tools mit.

Die realisierten Peripheriebausteine sieht man in **Bild 3**. SPI, I²C und UARTs sind ebenso vorhanden wie GPIOs, PWM-Einheiten und ein WS2812-Interface. So erhalten Einsteiger und Fortgeschrittene ein komplettes System, das eine komplette Entwicklungsumgebung für

Bild 2. Grober Ablauf einer Synthese.

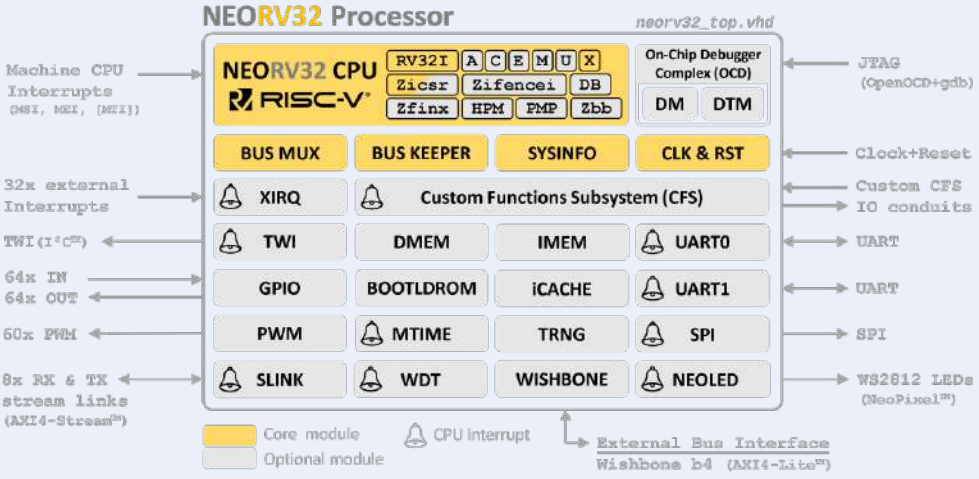


Bild 3. Übersicht der NEORV32-Funktionsblöcke (Quelle: Github / Nolting, S. [20]).

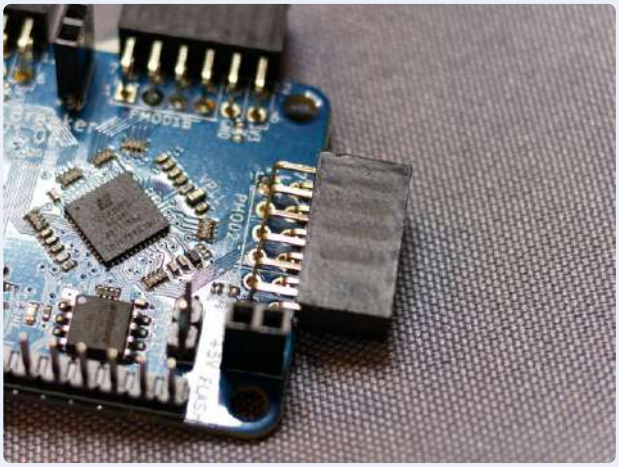


Bild 4. iCE40UP5K als QFN mit 7x7 mm Kantenlänge.

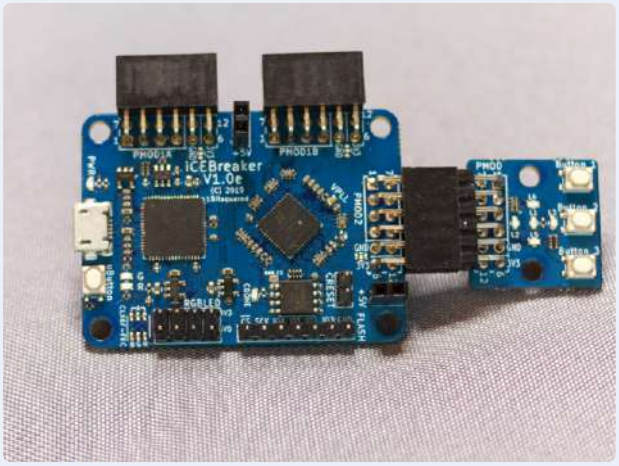


Bild 5. iCEBreaker-Board mit PMOD-Header.

den NEORV32 inklusive der nötigen Bibliotheken für die Hardware und Peripherie mitbringt. Zusätzlich sind für einige FPGA-Boards schon Beispielkonfigurationen vorhanden, so dass der Einstieg ohne größere Probleme starten kann. Doch welche Boards laufen „Out-of-the-Box" und wie schwierig ist es, den NEORV32 auf ein FPGA-Board zu bekommen, das nicht direkt unterstützt wird?

FPGA-Auswahl

Grundsätzlich kann jedes vorhandene Board mit FPGA genutzt werden, das genug Ressourcen bietet, da der NEORV32 keine herstellerspezifischen Erweiterungen nutzt. Ein FPGA, das auf mehreren preiswerten Boards zu finden ist, ist das Lattice iCE40UP5K [5].
Beim Lattice iCE40UP5K FPGA handelt es sich um die größte Ausbaustufe der iCE40 Ultra-Plus-Varianten. 5280 LUTs, 120 kBit (15 kByte) EBR RAM, 1024 kBit (128 kByte) SPRAM und fest verdrahtete Funktionseinheiten für SPI und I²C bilden eine solide Grundlage für die ersten eigenen Projekte. Nicht nur die Ausstattung des FPGAs macht dieses für eigene Projekte interessant, sondern auch das Package und der Preis. Ein QFN-48 mit 7x7 mm Kantenlänge (**Bild 4**) ist deutlich besser zu handhaben als ein Chip in BGA-Ausführung und mit etwa 5 € pro Chip liegt das Ganze auch noch in einem interessanten Preisbereich. Aktuell (Oktober 2021) ist das FPGA mit etwa 5 € bis 6 € pro Chip bei den meisten Distributoren leider ohne Bestand gelistet, mit Lieferzeiten von bis zu 46 Wochen.

Doch man muss nicht selbst ein passendes Board mit einem FPGA designen. Das iCEBreaker FPGA-Board (**Bild 5**) und das iCEBreaker Bitsy (**Bild 6**) von 1BitSquard [6] sind zwei Open-Hardware-Boards, auf dem das iCE40UP5K FPGA verbaut ist. Eine weitere Option und ebenfalls offene Hardware ist das UPduino V3.0 [7] von tinyVision.ai (**Bild 7**).
Das NEORV32-Projekt unterstützt das UPduino V3.0 direkt und bringt Beispielprojekte mit. Eine Anpassung für das iCEBreaker FPGA-Board ist recht schnell durchzuführen. Es wird hier erst einmal das UPduino V3.0 verwendet und anschließend gezeigt, was an einem Beispielprojekt angepasst werden muss, damit dieses auf dem iCEBreaker Board lauffähig ist.

In jedem Fall bekommt man in dieses FPGA ein SoC mit 64 kB Platz für Anwendungen, 64 kB RAM, SPI-Interface, I²C-Interface, 4 Input- und 4 Output-Pins, 3 PWM-Pins, ein UART; nicht zu vergessen natürlich der RV32IMAC-Kern mit 18 MHz Takt.

Toolchain

Bei der Toolchain für den Lattice iCE40up5k gibt es zwei Wege, die eingeschlagen werden können, die Nutzung der Tools von Lattice [8] und die Verwendung der OSS CAD Suite von YosysHQ [9], eine Toolchain, die komplett auf Open-Source-Tools basiert. Für dieses Projekt wird der Open-Source-Weg eingeschlagen. In diesem Fall bedeutet dies Ubuntu 20.04 LTS als Betriebssystem und die OSS CAD Suite, um den NEORV32 für den iCE40UP5K zu synthetisieren.

Neben den Tools für den FPGA soll später auch noch ein Demo-Programm für den synthetisierten RISC-V Prozessor übersetzt werden: Eine klassische Ausgabe von „Hello World" über den UART. Auch hier werden Open-Source-Tools eingesetzt und eine passende Toolchain (ähnlich wie schon für den Kendryte K210 [10]) generiert. Damit stehen dann ein GNU C-Compiler (GCC) und zusätzlich passende Bibliotheken für die Peripherie des NEORV32 bereit.

Mise en Place

Bei der Arbeit mit FPGAs ist es wie beim Kochen, wie mein Kollege Clemens Valens schon in seinem Video demonstriert hat [11]. Die Vorbereitung der Werkzeuge und Zutaten ist der erste Schritt, der zügig erledigt werden kann. Wer seine eigene Betriebssysteminstallation nicht beschädigen möchte, kann auch eine Virtuelle Maschine benutzen; hier wird von einem frisch installierten Ubuntu 20.04 auf einem AMD64-System ausgegangen. Die Verwendung eines Raspberry Pi sollte möglich sein, da sowohl Ubuntu 20.04 wie auch das Raspberry Pi OS auf Debian basieren, es kann aber zu kleinen Unterschieden durch die Architektur kommen. Ich selbst habe das Setup nur auf einem Ubuntu 20.04 und einer AMD64-Maschine getestet.

Um die „Hardware" für den FPGA zu synthetisieren, wird das aktuelle Release der OSS CAD Suite [12] in den Home-Ordner heruntergeladen, die Datei sollte *oss-cad-suite-linux-x64-xxxxxxxx.tgz* heißen. In einem Terminal wird diese nun mit `tar -xvzf oss-cad-suite-linux-x64-xxxxxxxx.tgz` entpackt und anschließend mit `sudo mv ~/oss-cad-suite /opt/` nach */opt* verschoben. Damit später auf den Ordner zugegriffen werden kann, werden mit einem `chmod 777 /opt/oss-cad-suite -R` die Rechte gesetzt. Es wird noch eine weitere Bibliothek benötigt, *libgnat-9*, die mit `sudo apt install libgnat-9` installiert wird. Damit wären die Tools für den FPGA vorbereitet.

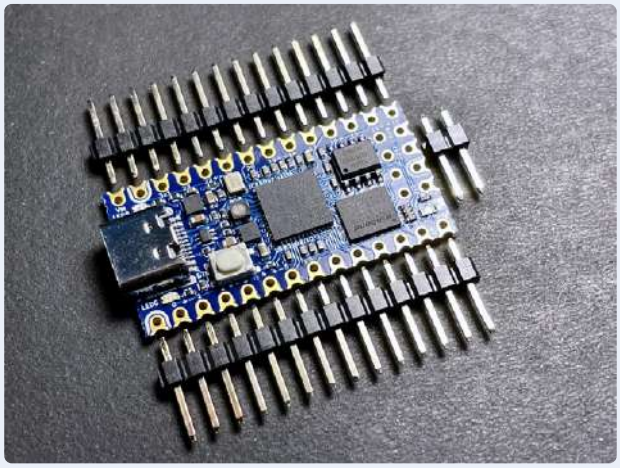


Bild 6. iCEBreaker Bitsy (Quelle: https://cdn.shopify.com/s/files/1/1069/4424/products/IMG_3859_large.jpg / 1BitSquare).

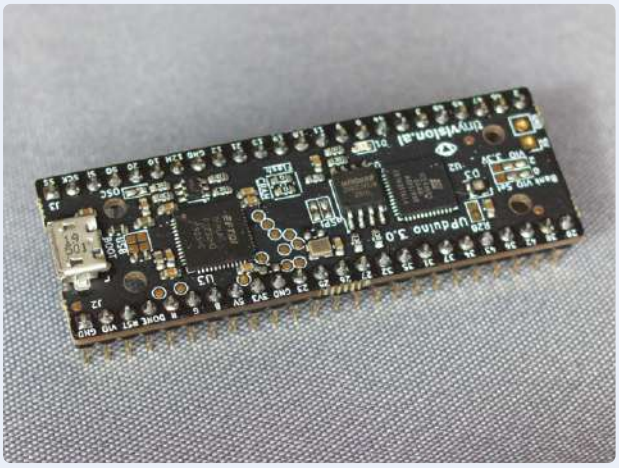


Bild 7. UPduino V3.0 mit Pinheadern.

Compiler

Als Nächstes werden die Daten aus dem GitHub-Repository des NEORV32 [4] benötigt. Dazu muss *git* mit `sudo apt install git` installiert werden. Anschließend wird mit `git clone https://github.com/stnolting/neorv32.git ~/neorv32` das Repository des NEORV32 geklont.
Um später mit dem NEORV32 zu kommunizieren, wird noch ein Terminalprogramm benötigt. Hier hat sich HTerm von Tobias Hammer bewährt, das Programm kann von seiner Homepage [13] mit `wget https://www.der-hammer.info/terminal/hterm-linux-64.tgz` heruntergeladen werden. Mit `mkdir ~/hterm && tar -xvf hterm-linux-64.tgz -C ~/hterm` wird der Inhalt der tgz-Datei im Ordner *~/hterm* entpackt. Da als normaler Benutzer später auf die seriellen Schnittstellen zugegriffen werden soll, muss der aktuelle Benutzer noch der Gruppe *dailout* hinzugefügt werden. Dazu wird in einem Terminal `sudo adduser $(whoami) dialout` eingegeben.

Nun soll auch einmal der C-Compiler selbst übersetzt werden. In den Beispielen für den iCE40up5k ist der NEORV32 als RV32IMAC konfiguriert, damit stehen zum Beispiel Befehle für Integer-Multiplikationen und -Divisionen zur Verfügung (siehe dazu auch den **Kasten RISC-V Namensschema**). Der Compiler muss für diese Architektur und die Befehls-Erweiterungen übersetzt werden; in der Dokumentation des NEORV32 [14] ist nachzulesen, warum diese Übereinstimmung wichtig ist.
In einem Terminal wird dazu zuerst mit cd ~ in den Home-Ordner gewechselt und anschließend mit `git clone https://github.com/riscv/riscv-gnu-toolchain --recursive` die RISC-V Toolchain geklont. Zusätzlich werden noch ein paar Pakete benötigt die mit

```
sudo apt-get install autoconf automake autotools-
   dev curl python3 libmpc-dev libmpfr-dev libgmp-
   dev gawk build-essential bison flex texinfo gperf
   libtool patchutils bc zlib1g-dev libexpat-dev
```

installiert werden müssen. Anschließend steht das Kompilieren des Compilers und der Bibliotheken an.
Bei einer RISC-V CPU gibt es einen kleinsten gemeinsamen Nenner, was die unterstützten Befehle angeht. Das bedeutet, dass Code, der für einen RV32I kompiliert wurde, auch auf einem RV32IMAC lauffähig ist, aber nicht umgekehrt. Daher ist die Empfehlung, erst einmal die Toolchain für den kleinsten gemeinsamen Nenner zu übersetzen. Mit `cd ~/riscv-gnu-toolchain` wird in einem Terminal in das gerade geklonte Repository gewechselt und anschließend mit `./configure --prefix=/opt/riscv --with-arch=rv32i --with-abi=ilp32` die Toolchain für das Kompilieren vorbereitet. Mit sudo make wird das Kompilieren gestartet. Die Toolchain steht danach in */opt/riscv* bereit. Damit jeder auf den Compiler zugreifen kann, müssen noch die Rechte von */opt/riscv* angepasst werden. In einem Terminal wird mit dem Befehl `chmod 777 /opt/riscv -R` jedem der Zugriff gewährt. Für die OSS CAD Suite und für die RISC-V GCC Toolchain muss noch eine Pfadvariable in */etc/enviroment* angepasst werden. Mit `sudo nano /etc/enviroment` wird die Datei zum Editieren geöffnet. Hinter `PATH="` wird `/opt/oss-cad-suite/bin:/opt/riscv/bin:` eingefügt und die Datei gespeichert.

Bei den meisten FPGA-Boards wird ein FT232H-Chip von FTDI zum Programmieren verwendet. Damit ein Benutzer ohne Root-Rechte darauf zugreifen kann, muss noch eine passende udev-Regel für den FTDI-Chip angelegt werden. In einem Terminal wird mit `sudo nano /etc/udev/rules.d/53-lattice-ftdi.rules` eine neue Datei zum Schreiben geöffnet. In dieser Datei muss dann

```
ACTION=="add", ATTR{idVendor}=="0403",
   ATTR{idProduct}=="6010", MODE:="666"
 ACTION=="add", ATTR{idVendor}=="0403",
   ATTR{idProduct}=="6014", MODE:="666"
```

eingefügt und die Datei gespeichert werden. Damit sind die Vorbereitungen abgeschlossen und es kann mit der Synthese und anschließendem Upload eines ersten Testprogramms losgehen. Damit alle Einstellungen wirksam werden, sollte ein Reboot durchgeführt werden. Eine Empfehlung ist noch, einen Editor mit Syntax-Highlighting für VHDL und Verilog zu installieren.

NEORV32 für den FPGA

Im stromlosen Zustand ist der FPGA nicht konfiguriert. Der iCE40UP5K liest, nachdem er mit Spannung versorgt wurde, die Verbindungsbeschreibung von einem externen SPI-Flash. Diese Beschreibung der internen Verbindungen muss dafür in das SPI-Flash auf dem UPDuino V3.0 oder dem iCEBreaker geschrieben werden.
In diesem Artikel erstellen wir ein Beispielprojekt für das UPduino

```
MEMORY
{
/* section base addresses and sizes have to be a multiple of 4 bytes */
/* ram section: first value of LENGTH => data memory used by bootloader (fixed!); second value of LENGTH => *physical* size of data memory */
/* adapt the right-most value to match the *total physical data memory size* of your setup */

  ram  (rwx) : ORIGIN = 0x80000000, LENGTH = DEFINED(make_bootloader) ? 512 : 8*1024

/* rom and iodev sections should NOT be modified by the user at all! */
/* rom section: first value of ORIGIN/LENGTH => bootloader ROM; second value of ORIGIN/LENGTH => maximum *logical* size of instruction memory */

  rom   (rx) : ORIGIN = DEFINED(make_bootloader) ? 0xFFFF0000 : 0x00000000, LENGTH = DEFINED(make_bootloader) ? 32K : 2048M
  iodev (rw) : ORIGIN = 0xFFFFFE00, LENGTH = 512

}
/* ************************************************************************** */
```

Bild 8. Anpassungen für die konfigurierte RAM-Größe.

```
user@user-VirtualBox:~/neorv32/sw/example/hello_world$ make exe
Memory utilization:
   text    data     bss     dec     hex filename
   5576       0     116    5692    163c main.elf
Executable (neorv32_exe.bin) size in bytes:
5588
user@user-VirtualBox:~/neorv32/sw/example/hello_world$
```

*Bild 9. **NEORV32_exe.bin** erstellt.*

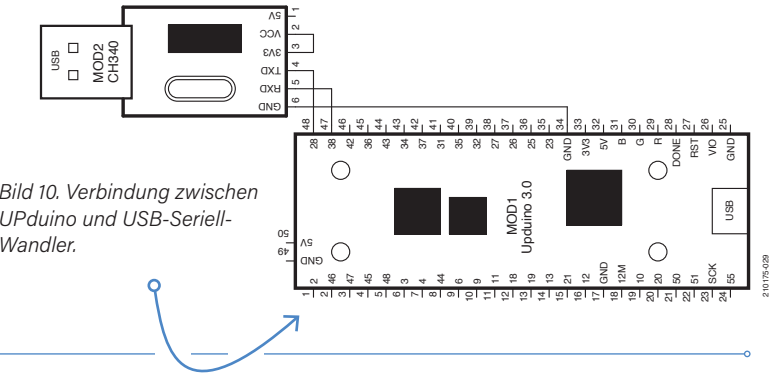


Bild 10. Verbindung zwischen UPduino und USB-Seriell-Wandler.

V3.0 Board, das aus dem Prozessor samt Peripherie besteht. Damit entsteht ein System, das auf dem FPGA direkt lauffähig sein sollte.

In einem Terminal muss nun in das Beispielverzeichnis für die Open Source Tools mit `cd ~/neorv32/setups/osflow/` gewechselt werden. Um nun die Synthese und damit die Erstellung des Bitstroms für den FPGA zu beginnen, reicht es aus, `make BOARD=UPduino UP5KDemo` als Befehl einzugeben, danach kann es ein wenig dauern, bis die Synthese abgeschlossen ist. Im Ordner *~/neorv32/setups/osflow/* ist nun unter anderem eine *neorv32_UPduino_v3_UP5KDemo.bit* Datei entstanden: Der Bitstrom beschreibt, wie im FPGA die logischen Grundbausteine verschaltet werden müssen. Dieser Bitstrom muss nun in das SPI-Flash auf dem FPGA-Board geschrieben werden.

Das UPduino-Board muss per USB mit dem PC verbunden werden; im Terminal müssen wir `iceprog ~/neorv32/setups/osflow/neorv32_UPduino_v3_UP5KDemo.bit` eingeben. Damit startet das Programmieren des externen SPI-Flashs und die Konfiguration wird anschließend in den FPGA geladen. Damit steht im UPduino V3.0 nun ein RISC-V System bereit, das nun mit Software versorgt werden kann.

Wir eingangs schon erwähnt, sind 64 kB für Anwendungen vorhanden und 64 kB stehen als RAM bereit. An Peripherie gibt es eine SPI-Schnittstelle, I²C, einen UART, vier Eingänge und vier Ausgänge, sowie drei PWM-Ausgänge. Die CPU, die hier synthetisiert wurde, ist ein RV32IMAC, der mit 18 MHz laufen wird. Das SoC in dem FPGA verfügt auch über einen kleinen Bootloader, der per UART bedient werden kann.

Hallo Welt

Da der FPGA nun mit dem NEORV32 ausgestattet ist, kann die erste *Hello World* Demo für den RISC-V kompiliert und hochgeladen werden. Da der NEORV32 konfigurierbar ist, muss im Linker-Skript die aktuelle Größe des RAM passend konfiguriert werden. Dazu wird in einem Terminal `nano ~/neorv32/sw/common/neorv32.ld` eingegeben und in der Zeile 62

```
ram (rwx) : ORIGIN = 0x80000000, LENGTH =
   DEFINED(make_bootloader) ? 512 : 8*1024
```

gegen

```
ram (rwx) : ORIGIN = 0x80000000, LENGTH =
   DEFINED(make_bootloader) ? 512 : 64*1024
```

ausgetauscht (**Bild 8**). Nun ist der RISC-V Compiler schon einsatzbereit. In einem geöffneten Terminal kommt man mit `cd ~/neorv32/sw/example/hello_world` in den Ordner des *Hello World* Programms. Um eine Executable (ausführbare Datei) zu erhalten, die in den NEORV32 geladen werden kann, reicht es aus, `make exe` einzugeben, damit *neorv32_exe.bin* generiert wird (**Bild 9**). Damit der NEORV32 das Programm ausführen kann, muss dieses hochgeladen werden. Dazu wird der integrierte Bootloader genutzt, der über den UART Daten entgegennimmt (19200 Baud, 8 Datenbits, 1 Stoppbit, keine Parität, kein Flusskontrolle). Wenn ein UPduino V3.0 verwendet wird, ist noch ein zusätzlicher USB-Seriell-Wandler nötig, so wie der CH340-basierende aus dem Elektor Store (siehe **Kasten Passende Produkte**). Dieser muss wie in **Bild 10** zu sehen verbunden werden.

Für den Upload selbst wird HTerm genutzt. Dieses kann aus einem Terminal mit `~/hterm/hterm` gestartet werden; es sollte ein Fenster wie in **Bild 11** erscheinen. Als Port muss nun der USB-Seriell-Wandler ausgewählt werden, der sich in der Regel als */dev/ttyUSB0* meldet; je nach Hardwarekonfiguration und gewähltem Adapter kann dies aber unterschiedlich sein.

Nach dem Verbinden mit dem USB-Seriell-Adapter kann der UPDuino mit Spannung versorgt werden und es sollte die Meldung des Bootloaders zu sehen sein (**Bild 12**). Wird nicht rechtzeitig ein Zeichen an den Bootloader gesendet, versucht dieser ein Autoboot vom SPI-Flash, das im Moment noch keine Software beinhaltet. Es reicht aus, nach dem Start des Bootloaders innerhalb von 8 Sekunden ein beliebiges Zeichen zu senden, um in den Kommandomodus zu gelangen. Anschließend muss ein u gesendet werden, um den Upload in Bootloader zu aktivieren, und in HTerm die Schaltfläche *Select File* angeklickt werden. Wie in

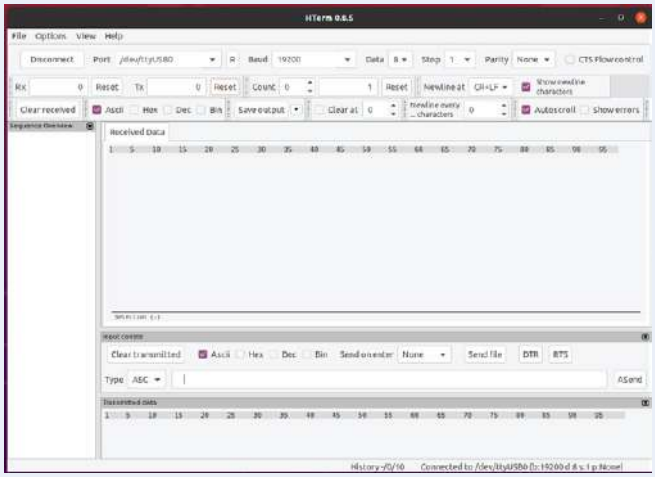


Bild 11. Geöffnetes HTerm-Fenster.

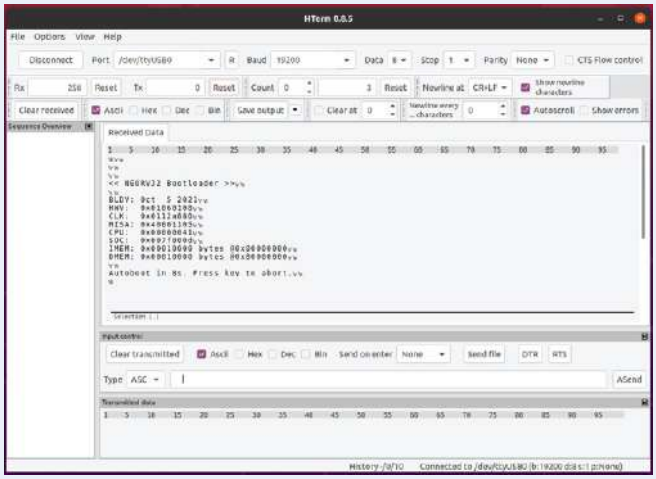


Bild 12. Bootloader des NEORV32.

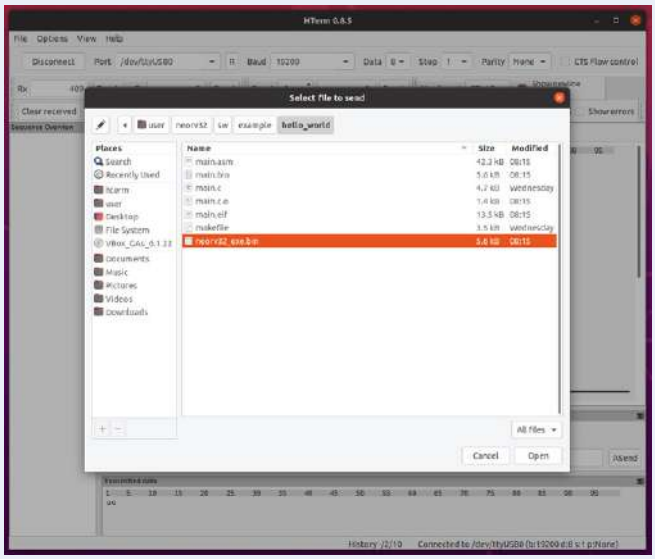


*Bild 13. Dialog für den Upload der **neorv32_exe.bin**.*

```
Available CMDs:\r\n
 h: Help\r\n
 r: Restart\r\n
 u: Upload\r\n
 s: Store to flash\r\n
 l: Load from flash\r\n
 e: Execute\r\n
CMD:> u\r\n
Awaiting neorv32 exe.bin... OK\r\n
CMD:>
```

Bild 14. Erfolgreicher Upload.

```
<< NEORV32 Bootloader >>\r\n
\r\n
BLDV: Oct  5 2021\r\n
HWV:  0x01060108\r\n
CLK:  0x0112a880\r\n
MISA: 0x40801105\r\n
CPU:  0x00000041\r\n
SOC:  0x097f009d\r\n
IMEM: 0x00010000 bytes @0x00000000\r\n
DMEM: 0x00010000 bytes @0x80000000\r\n
\r\n
Autoboot in 8s. Press key to abort.\r\n
Aborted.\r\n
\r\n
Available CMDs:\r\n
 h: Help\r\n
 r: Restart\r\n
 u: Upload\r\n
 s: Store to flash\r\n
 l: Load from flash\r\n
 e: Execute\r\n
CMD:> u\r\n
Awaiting neorv32 exe.bin... OK\r\n
CMD:> e\r\n
Booting...\r\n
\r\n
\r\n
 ##      ##   #########   ########    ########   ##       ##   ########    ########
####     ##   ##          ##      ##  ##      ##  ##       ##   ##      ##  ##      ##
## ##    ##   ##          ##      ##  ##      ##  ##       ##   ##      ##  ##      ##
##  ##   ##   #########   ##      ##  ########    ##       ##   ##      ##  ########
##   ## ##   ##          ##      ##  ##  ##      ##       ##   ##      ##  ##  ##
##    ####   ##          ##      ##  ##   ##     ##       ##   ##      ##  ##   ##
##     ##    #########   ########    ##    ##    ########    ########    ##    ##

Hello world! :)\r\n
```

Bild 15. „Hello World" vom NEORV32.

RISC-V Namensschema

RISC-V ist ein Oberbegriff für unterschiedliche Architektur-Varianten. Unser Kollege Stuart Cording hat einen schönen Artikel [2] verfasst, in dem die Details genauer erklärt werden. RISC-V beschreibt eine ISA, bei dem die Prozessoren als 32-, 64- oder 128-Bit-Ausführungen definiert sind. Ein 32-Bit-Prozessor hat eine Bezeichnung, die mit RV32 für 32 Bit anfängt, so wie ein RV64 auf einen 64-Bit-Prozessor hinweist. Hinzu kommen nach RV32 oder RV64 noch eine Reihe von Buchstaben, die angeben, welche Befehle und Erweiterungen der Prozessor beherrscht. Diese Buchstaben reichen von A bis Z; ihre Bedeutung kann in der aktuellen RISC-V-Spezifikation [3] nachgeschlagen werden. Je nach den unterstützten Befehlen und Erweiterungen unterscheidet sich die Leistungsfähigkeit der Prozessoren.

Bild 13 zu sehen, ist die Datei *neorv32_exe.bin* im Ordner *~/neorv32/sw/example/hello_world* auszuwählen und anschließend hochzuladen. Wenn alles beendet ist, meldet der Bootloader wie in **Bild 14** ein `OK` und das Programm kann mit e gestartet werden.
Das Ergebnis ist in **Bild 15** zu sehen. Das Programm wird nicht in das SPI-Flash geschrieben, sondern in das RAM-basierte „ROM" des NEORV32. Damit wird die Lebensdauer des SPI-Flashs durch Reduzierung der Schreibvorgänge verlängert. Der Nachteil ist so jedoch, dass bei jedem Neustart des NEORV32 auch wieder ein Upload des Programms nötig ist. Soll das Programm automatisch geladen werden, so muss durch den Bootloader ein Upload in das SPI-Flash erfolgen. Neben der „Hello World"-Demo gibt es noch weitere Beispiele zu entdecken, bis zu einem kompletten FreeRTOS, das auch schon Thema [15] in Elektor war. Dazu lohnt ein Blick in den Ordner *~/neorv32/sw/example*, in dem jedes Beispiel in einem eigenen Ordner zu finden ist.

Ein neues iCE40UP5K-Board hinzufügen

Das iCEBreaker Board soll hier demonstrieren, wie man den NEORV32 auch an andere iCE40up5k-Boards anpassen kann. Die Features des SoCs selbst werden hier nicht verändert, aber das Pinout am FPGA so angepasst, das ein iCEBreaker-Board verwendet und ein passender Bitstrom generiert werden kann.
Dazu muss das Makefile in *~/neorv32/setup/osflow* angepasst werden. Die Datei wird mit einem Texteditor der Wahl geöffnet und das neue Board als Target hinzugefügt. Für den iCEBreaker schreiben wir in Zeile 72 folgendes:

```
iCEBreaker:
$(MAKE) \
BITSTREAM=neorv32_$(BOARD)_$(DESIGN).bit \
NEORV32_MEM_SRC="devices/ice40/neorv32_imem.ice40up_
   spram.vhd devices/ice40/neorv32_dmem.ice40up_
   spram.vhd" \
run
```

Damit ist im primären Makefile schon einmal das Board bekannt, jedoch muss nun auch in *~/neorv32/setup/osflow/boards* eine *iCEBreaker.mk* mit folgendem Inhalt angelegt werden:

```
.PHONY: all

all: bit
echo "! Built $(IMPL) for $(BOARD)"
```

Damit wären die Makefiles vorbereitet, jedoch fehlen noch zwei VHDL-Dateien. Unter *~/neorv32/setup/osflow/board_tops/* muss eine *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* und eine *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd* erzeugt werden. Da der Inhalt fast mit dem der *neorv32_UPduino_BoardTop_UP5KDemo.vhd* und *neorv32_UPduino_BoardTop_MinimalBoot.vhd* identisch ist, können diese Dateien kopiert und umbenannt werden. Dazu sollte in einem Terminal

```
cp ~/neorv32/setups/osflow/board_tops/neorv32_UPduino_
BoardTop_MinimalBoot.vhd ~/neorv32/setups/osflow/board_
tops/neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd
```

und

```
cp ~/neorv32/setups/osflow/board_tops/neorv32_UPduino_
BoardTop_UP5KDemo.vhd ~/neorv32/setups/osflow/board_tops/
neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd
```

eingegeben werden. In den beiden Dateien *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* und *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd* müssen nun noch ein paar Anpassungen durchgeführt werden. In der Datei *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* muss die Zeile 42 in `entity neorv32_iCEBreaker_BoardTop_UP5KDemo is` und Zeile 68 in `architecture neorv32_iCEBreaker_BoardTop_UP5KDemo_rtl of neorv32_iCEBreaker_BoardTop_UP5KDemo is` geändert werden. In der Datei *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd* ist Zeile 42 in `entity neorv32_iCEBreaker_BoardTop_MinimalBoot is` und Zeile 54 in `architecture neorv32_iCEBreaker_BoardTop_MinimalBoot_rtl of neorv32_iCEBreaker_BoardTop_MinimalBoot is` zu ändern.
Der letzte Schritt ist das Constraints-File *iCEBreaker.pcf,* das unter *~/neorv32/setups/osflow/constraints/* angelegt werden muss. Der Inhalt der Datei ist in **Listing 1** zu sehen. Mit *iCEBreaker.pcf* wird festgelegt, welche Funktion auf welchen Pin des FPGA geroutet werden soll. Damit wird auch der USB-Seriell-Wander, der sich auf dem iCEBreaker befindet, direkt mit eingebunden, sowie die Taster und LEDs auf die IO-Pins des NEORV32 gelegt. Etwas das noch stört, ist der fehlende Reset-Taster; auch wenn dieser schon in der Constraints-Datei definiert ist, hat er hier noch keine Funktion.

Mit den aktuellen Anpassungen kann nun genau wie für den UPduino ein Bitstrom erstellt werden. Dazu muss ein Terminal geöffnet werden und mit `cd ~/neorv32/setups/osflow` in den Ordner *osflow* gewechselt werden. Mit `make BOARD=iCEBeaker UP5KDemo` lässt sich das Erstellen starten. Das Programmieren des Bitsteams in den iCEBreaker erfolgt (wie beim UPduino) anschließend mit `iceprog ~/neorv32/setups/osflow/neorv32_iCEBreaker_UP5KDemo.bit`. Das Programmieren der Software für den NEORV32 geschieht auch hier über den integrierten Bootloader des NEORV32; nur dass hier kein externer USB-Seriell-Wandler verwendet werden muss, sondern der zweite Kanal des auf dem iCEBreaker integrierten Konverters genutzt wird.

Listing 1. iCEBreaker.pcf

```
#UART (uart0)
ldc_set_location -site {9} [get_ports uart_txd_o]
ldc_set_location -site {6} [get_ports uart_rxd_i]

#SPI - on-board flash
ldc_set_location -site {14} [get_ports flash_sdo_o]
ldc_set_location -site {15} [get_ports flash_sck_o]
ldc_set_location -site {16} [get_ports flash_csn_o]
ldc_set_location -site {17} [get_ports flash_sdi_i]

#SPI - user port
ldc_set_location -site {43} [get_ports spi_sdo_o]
ldc_set_location -site {38} [get_ports spi_sck_o]
ldc_set_location -site {34} [get_ports spi_csn_o]
ldc_set_location -site {31} [get_ports spi_sdi_i]

#TWI
ldc_set_location -site {2} [get_ports twi_sda_io]
ldc_set_location -site {4} [get_ports twi_scl_io]

#GPIO - input
ldc_set_location -site {18} [get_ports {gpio_i[0]}]
ldc_set_location -site {19} [get_ports {gpio_i[1]}]
ldc_set_location -site {20} [get_ports {gpio_i[2]}]
ldc_set_location -site {28} [get_ports {gpio_i[3]}]

#GPIO - output
ldc_set_location -site {25} [get_ports {gpio_o[0]}]
ldc_set_location -site {26} [get_ports {gpio_o[1]}]
ldc_set_location -site {27} [get_ports {gpio_o[2]}]
ldc_set_location -site {23} [get_ports {gpio_o[3]}]

#RGB power LED
ldc_set_location -site {39} [get_ports {pwm_o[0]}]
ldc_set_location -site {40} [get_ports {pwm_o[1]}]
ldc_set_location -site {41} [get_ports {pwm_o[2]}]

#Reset
ldc_set_location -site {10} [get_ports
   {user_reset_btn}]
```

Ein Reset-Taster für den NEORV32

Um den NEORV32 neu zu starten musste aktuell immer die Spannungsversorgung kurz getrennt und wieder verbunden werden. Das ist auf Dauer etwas lästig und da der iCEBreaker genug Tasten hat, kann eine davon als Reset benutzt werden. Dazu wird der uButton nahe der Micro-USB Buchse verwendet, welcher an Pin 10 des FPGA geführt ist.

Der NEORV32 hat intern einen Eingang für einen Reset, an dem aber die PLL zur Erzeugung des Prozessortaktes angeschlossen ist. Die sauberste Lösung wäre es, mit etwas Logik das externe Resetsignal und das Resetsignal der PLL korrekt logisch zu verknüpfen und dem NEORV32 bereitzustellen. Die einfachere Lösung ist es, den Reseteingang der PLL zu nutzen. **Bild 16** zeigt die Verbindung des uButton mit diesem Eingang. Wird die PLL resettet, so wird auch der NEORV32 mit resettet, da das *lock_o* Signal den Zustand wechselt, wenn die PLL im Reset ist.

Um das UP5K-Demo-Projekt entsprechend zu erweitern, müssen in der *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* in dem Ordner ~/neorv32/setups/board_tops eine Zeile eingefügt und eine Zeile angepasst werden. Ab Zeile 44 wird `user_reset_btn : in std_ulogic;` als eigene Zeile eingefügt. Zeile 130 wird in `RESETB => user_reset_btn,` geändert, das Komma am Ende sollte nicht vergessen werden, da es sonst zu einem Syntaxfehler kommt. Damit wäre der uButton als Reset funktional. Es muss abschließend der Bitstrom einmal mit den aktuellen Anpassungen in der UP5K-Demo neu generiert und in den FPGA des iCEBreaker geladen werden.

Ausblick

Über das Thema RISC-V, FPGA und auch den NEORV32 könnte man mehr als nur ein Buch füllen. Der iCE40UP5K ist für Einsteiger eine preiswerte Wahl, wenn der Chip denn lieferbar wäre. Die beiden im

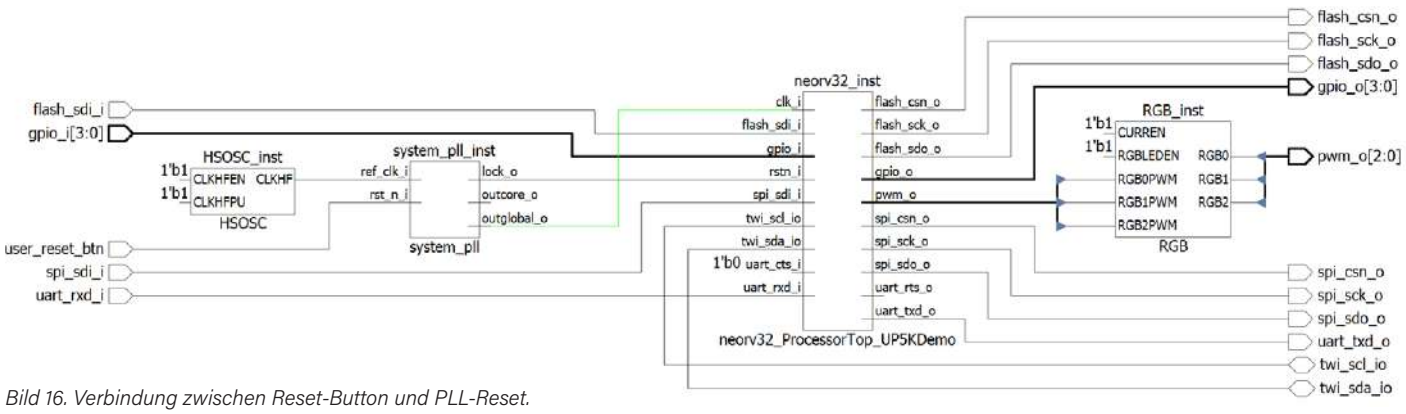


Bild 16. Verbindung zwischen Reset-Button und PLL-Reset.

Artikel vorgestellten Boards sind nicht die einzigen, die diesen FPGA nutzen, es gibt noch mindestens eine Handvoll weiterer Plattformen; vom Raspberry Pi Add-On bis hin zur kompletten Handheld-Konsole ist alles vertreten. Auch die Tools und Projekte, in denen dieser FPGA verwendet werden kann, sind einen Blick wert. Beispiele sind LiteX [16], mit dem das eigene SoC wie in einem Baukasten zusammengestellt werden kann und nicht zwangsweise auf RISC-V beschränkt ist, der RISCboy [17] von Luke Wren, der ein Gameboy-artiges System im FPGA bereitstellt sowie der OK-ice40-PRO [18], eine Gamepad-Konsole. Sollte die Beschaffungssituation besser werden, kommen sicherlich noch weitere Projekte und Ideen für den iCE40UP5K hinzu. Eine Mischung aus Raspberry Pi RP2040 und iCE40UP5K als kleine Spielekonsole scheint schon in der Konzeption zu sein [19].

210175-02

Ein Beitrag von

Text und Bilder: **Mathias Claußen**
Redaktion: **Jens Nickel**
Layout: **Harmen Heida**

Sie haben Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Schicken Sie eine E-Mail an den Autor unter mathias.claussen@elektor.com oder kontaktieren Sie Elektor unter editor@elektor.com.

PASSENDE PRODUKTE

- **CH340 USB/TTL-UART Konverter-Modul CH340G (3,3 V/5,5 V) (SKU 19151)**
  www.elektor.de/19151
- **Alchitry Cu FPGA Development Board (Lattice iCE40 HX) (SKU 19640)**
  www.elektor.de/19640
- **Buch: *Inside an Open-Source Processor* (SKU 19826)**
  www.elektor.de/19826

WEBLINKS

[1] Martin Oßmann, „Das SCCC-Projekt (1)", ElektorMag 3-4/2019: www.elektormagazine.de/magazine/elektor-87/42467
[2] Stuart Cording, „What Is RISC-V?", elektormagazine.com 04/2021: www.elektormagazine.com/articles/what-is-risc-v
[3] RISC-V Spezifikationen: https://riscv.org/technical/specifications/
[4] NEORV32 GitHub Repository: https://github.com/stnolting/neorv32/
[5] Lattice iCE40UltraPlus Produktseite: www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus
[6] iCEBreaker GitHub Repository: https://github.com/icebreaker-fpga/icebreaker
[7] UPduino GitHub Repository: https://github.com/tinyvision-ai-inc/UPduino-v3.0
[8] Lattice Radiant: www.latticesemi.com/LatticeRadiant
[9] YosysHQ oss-cad-suite-build : https://github.com/YosysHQ/oss-cad-suite-build
[10] Setup a toolchain for the Kendryte K210, Elektor Labs: www.elektormagazine.de/labs/setup-a-toolchain-for-the-kendryte-k210-1
[11] Linux-flavored Snickerdoodles with Zynq, ElektorTV: www.youtube.com/watch?v=EE4yYZ-FEoQ
[12] YosysHQ oss-cad-suite-build Releases: https://github.com/YosysHQ/oss-cad-suite-build/releases
[13] HTerm: www.der-hammer.info/
[14] NEORV32 - Build the Toolchain from scratch: https://stnolting.github.io/neorv32/ug/#_building_the_toolchain_from_scratch
[15] Warren Gay, „Praktisches ESP32-Multitasking", ElektorMag 1-2/2020: www.elektormagazine.de/magazine/elektor-138/56969
[16] LiteX: https://github.com/enjoy-digital/litex
[17] RISCBoy : https://github.com/Wren6991/RISCBoy
[18] OK-iCE40Pro Handheld: https://github.com/WiFiBoy/OK-iCE40Pro
[19] PicoStation3D: https://github.com/Wren6991/PicoStation3D
[20] Nolting S., The NEORV32 RISC-V-Processor, GitHub repository 2020: https://raw.githubusercontent.com/stnolting/neorv32/master/docs/figures/neorv32_processor.png

Unter dem Radar

Unbekannte Mikrocontroller, über die Sie Bescheid wissen sollten

Von Clemens Valens (Elektor)

Der globale Markt für Mikrocontroller ist vielfältiger, als viele Menschen denken. Werfen wir einen Blick auf einige der Mikrocontroller und Hersteller, die nicht so oft in Elektor zu finden sind. Vielleicht finden Sie den einen oder anderen davon bei einem zukünftigen Projekt nützlich.

Die Wahl des Mikrocontrollers für eine Elektor-Schaltung basiert meist auf der Verfügbarkeit von kostengünstigen Software-Entwicklungswerkzeugen und Programmiergeräten sowie der Möglichkeit, diese zu kaufen oder zu bauen. Daher haben viele Elektronik-Enthusiasten einen eher begrenzten Blick auf den weltweiten Mikrocontroller-Markt. Es gibt viel mehr als PIC, AVR, ARM oder ESP, Controller, die wir so oft in DIY-Projekten sehen. Werfen wir einen Blick auf einige der MCUs, die in Ihrem blinden Fleck leben.

Alles begann mit vier Bits

Der 1971 vorgestellte Intel 4004 gilt als der erste kommerziell hergestellte Mikroprozessor (mehr dazu in unserem Elektor-Industry-Spezial über 60 Jahre Elektronik [1]). Der 4-Bit-Prozessor war Bestandteil der MCS-4-Familie. Ihm folgte die MCS-40-Familie mit der 4040-CPU. Der erste erfolgreiche Mikrocontroller (kein Mikroprozessor) war der TMS1000 von Texas Instruments aus dem Jahr 1974, ebenfalls ein 4-Bit-Baustein, der wie der 4040 seinen Weg in viele Taschenrechner fand.

In einer Welt, in der die Hersteller von Mikrocontrollern anscheinend nach möglichst hohen Datenwortbreiten streben - 64 Bit sind keine Seltenheit -, dürfte es Sie überraschen, wie viele 4-Bit-Mikrocontroller noch heute im Einsatz sind. Aber warum? Die Antwort ist wahrscheinlich eine Mischung aus Legacy-, Strombedarfs- und Kostengründen.

Eine 4-Bit-MCU kann mit weniger Transistoren gebaut werden als Bausteine mit einer größeren Wortbreite. Daher benötigen sie unter sonst gleichen Randbedingungen weniger Strom, was zu einer längeren Batterielebensdauer beiträgt. Weniger Transistoren bedeuten auch weniger Platz, und so kann ein 4-Bit-Kern in einer Ecke des Chips untergebracht werden oder der Chip kann kleiner sein, was wiederum Kosten spart (auch wenn man sich fragen mag, wie viel). Großvolumige Anwendungen wie Taschenrechner, Timer, Uhren, Fahrradcomputer, Spielzeug und Fernbedienungen verwenden 4-Bit-MCUs, und das schon seit vielen Jahrzehnten. Da die Hersteller es in der Regel vermeiden, Produkte zu ändern, die sich in der Praxis bewährt haben - wenn es läuft, lass es laufen - erklärt dies gut, warum es immer noch einen Markt für solche kleinen MCUs gibt.

Falls Sie einen 4-Bit-Mikrocontroller ausprobieren möchten, werfen Sie einen Blick auf die NY-Familien des taiwanesischen Unternehmens Nyquest. Die modernen Entwicklungswerkzeuge können kostenlos heruntergeladen werden (**Bild 1**). Weitere Hersteller sind EM Microelectronic aus der Schweiz, CR Micro aus China und Tenx Technology aus Taiwan.

8051

Bevor ARM zum wichtigsten MCU-Core-Lieferanten für fast alle Halbleiterhersteller der Welt wurde, gab es den 8-Bitter 8051. Der 1980 von Intel als MCS-51 entwickelte Kern (**Bild 2**) wurde an mehrere Wettbewerber lizenziert und fand seinen Weg in eine Vielzahl von Produkten. Viele dieser Produkte oder ihre Derivate, Varianten und Geschwister werden auch heute noch hergestellt, und auch 40 Jahre nach der Markteinführung werden 8051-Derivate aktiv in neue Produkte eingesetzt.

Ein weiterer Grund dafür ist, dass Legionen

von 8051-Anwendern in dieser Zeit viel Software und Know-how entwickelt haben, und eine solche Ressource wirft man nicht leichtfertig weg, nur weil ein besserer Mikrocontroller auf den Markt kommt.
Schließlich ist der 8051-Kern sehr billig, wenn nicht sogar kostenlos geworden. Das macht ihn zu einer interessanten Option für Halbleiterhersteller, die extrem preiswerte Geräte entwickeln. Nicht immer wird dies im Datenblatt erwähnt, aber wenn dort so etwas wie „1T instruction cycle“ steht, können Sie mit Sicherheit davon ausgehen, dass es sich um ein 8051-Derivat handelt. Der ursprüngliche 8051 benötigte für die meisten Befehle zwölf Taktzyklen (genannt „12T“), während die modernsten Versionen nur einen benötigen (daher „1T“). Abgesehen davon, dass weniger Taktzyklen pro Befehl benötigt werden, ist die Programmausführung auch (viel) schneller, zumal einige dieser modernen Derivate mit Frequenzen von bis zu 450 MHz anstelle von 12 MHz des Originals laufen. Daher sind moderne 8051-MCUs preiswerte und dennoch leistungsfähige 8-Bit-Mikrocontroller.
Die wohl größte Unannehmlichkeit des 8051 aus heutiger Sicht ist wahrscheinlich seine Inkompatibilität mit modernen Programmiersprachen wie C und C++, was in seiner seltsamen Speicherstruktur begründet liegt. Um das Beste aus dem 8051 herauszuholen, bedarf es einer kommerziellen Toolchain von Keil oder IAR oder ähnlichen Compiler-Profis. Die populäre GCC-Toolchain, die auf alle möglichen Mikrocontroller portiert wurde, bietet leider keine Unterstützung für den 8051. Die freie Toolchain SDCC macht einen einigermaßen vernünftigen Job, ist aber bei weitem nicht perfekt. Die Programmierung des 8051 in Assembler ist natürlich auch eine Option. Oder bevorzugen Sie Pascal? Dann werfen Sie einen Blick auf Turbo51 [2]!
Sie finden 8051-basierte MCUs in preisgünstigen Produkten mit hohen Stückzahlen wie Spielzeug, PC-Tastaturen und -Mäusen, Zahnbürsten, Haushaltsgeräten, Fernbedienungen und so weiter, hergestellt hauptsächlich in Asien, wo der 8051 besonders beliebt zu sein scheint. Silan, SiGma Micro, SinoWealth, Silicon Laboratories, Sonix, STC und SyncMOS (um nur einmal die mit „S“ beginnenden Firmen zu nennen) stellen alle Produkte mit 8051ern her.

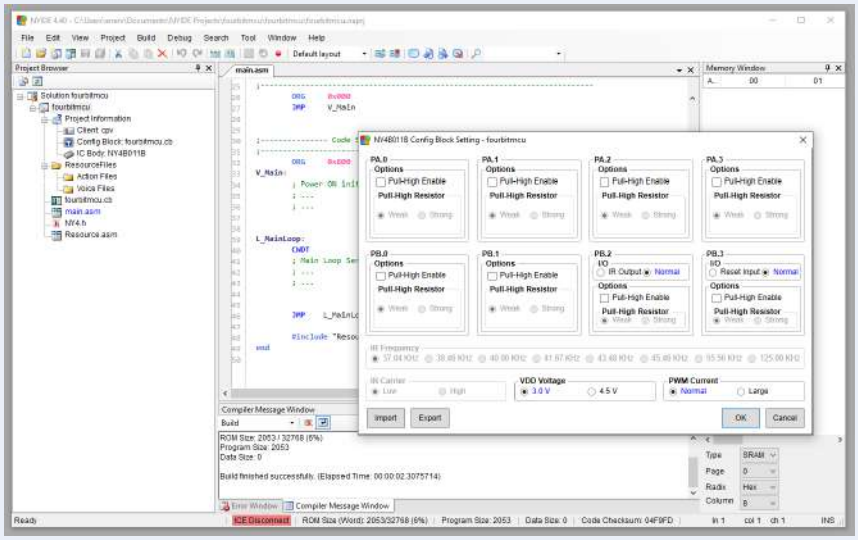


Bild 1. NYIDE 4.40 von Nyquest zeigt, dass auch 4-Bit-Mikrocontroller moderne IDEs haben können.

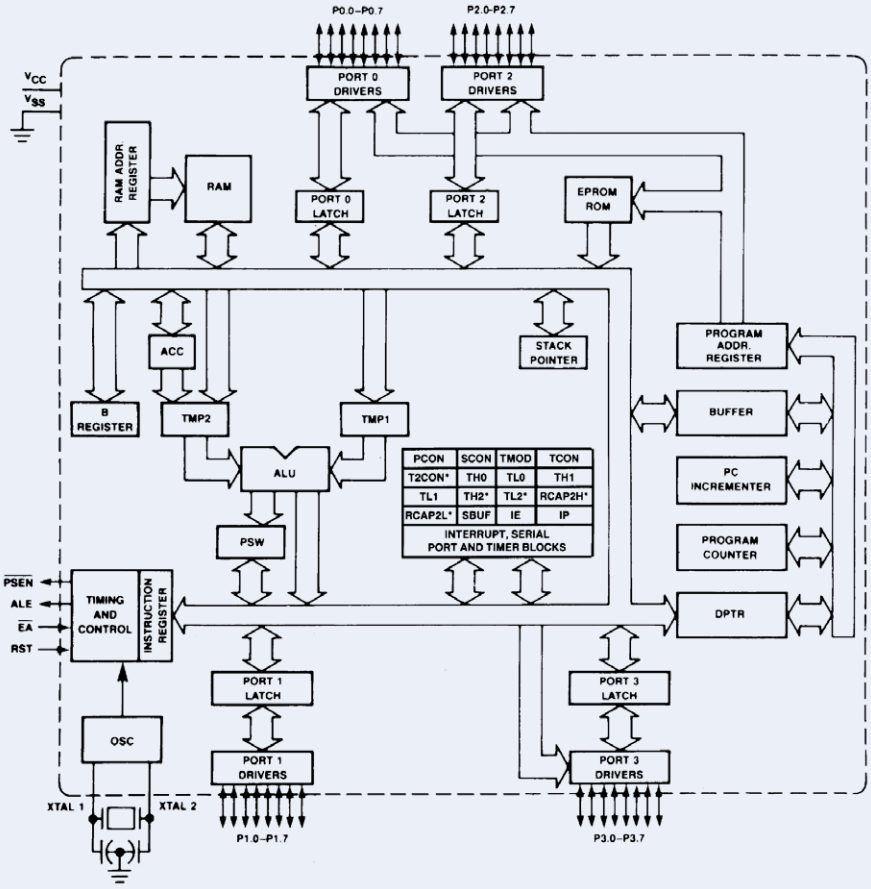


Bild 2. Die grundlegende Architektur des weltweit vielleicht am häufigsten verwendeten Mikrocontroller-Kerns 8051. (Quelle: Intel)

Wenn auch Sie mit einem modernen 8051-Derivat experimentieren möchten, sollten Sie sich die CH55x-Familie von WCH ansehen. Sie sind zwar nur auf Chinesisch dokumentiert, aber billig und verfügen über eine USB-Schnittstelle, die eine einfache Programmierung ermöglicht. Und sie werden von Open-Source-Projekten unterstützt. Einen 8051 in einen Arduino verwandeln? Kein Problem, wie zum Beispiel in [3] zu sehen.

Bild 3. Im Commodore 64, einem der berühmten Heimcomputer aus den 1980er Jahren, schlägt ein Herz aus 6502. (Quelle: ralfsfotoseite bei Pixabay)

Evergreens und Ewiggestrige

Neben dem 8051 aus dem Jahr 1980 gibt es noch einige andere Prozessorkerne aus dieser Zeit, vor allem den 6502 und den Z80, aber auch den etwas jüngeren 68000 von Motorola (jetzt NXP). Der Z80 und die CMOS-Version des 6502, der W65C02, werden immer noch von ihren Entwicklern Zilog (6502) beziehungsweise WDC(W65C02) hergestellt und aktiv unterstützt. Der 68000 hingegen scheint hauptsächlich zur Unterstützung älterer Anwendungen am Leben erhalten zu werden (aber wer weiß, wie viele andere Hersteller eine Lizenz erworben haben?).

6502

Der 6502 wurde in mehreren berühmten frühen Computern wie dem Commodore 64, dem Apple II und dem BBC Micro eingesetzt und war tatsächlich sehr erfolgreich (**Bild 3**). Seine verbesserte und energiesparende CMOS-Variante ist immer noch aktuell, auch wenn sie recht teuer ist. Der Grund dafür ist, dass die meisten Anwender den Kern nur für die Verwendung in FPGAs, ASICs und ähnlichen kundenspezifischen Chips lizenzieren. Da dies alles „confidential" ist, fällt es schwer herauszufinden, um welche Beuteile es sich handelt.
WDC stellt jedoch echte 65er-ICs her, die Sie ausprobieren können, zum Beispiel den Mikrocontroller W65C265S mit einer 16-Bit-CPU W65C816S, die vollständig mit dem 8-Bit-Chip W65C02S kompatibel ist und bereits ab 1,8 V läuft. Es gibt auch Controller-Module und sogar ein „Companion's Board" mit Seeed-Studio-Grove-, Sparkfun-QWIIC- und MikroE-Click-Anschlüssen.

Z80

Der Z80 ist ein weiterer sehr erfolgreicher Prozessor aus den späten siebziger und frühen achtziger Jahren des vorigen Jahrhunderts. Der Kern wurde 1975 von Zilog entwickelt und wird immer noch hergestellt. Er wurde an viele andere Hersteller weltweit lizenziert und von diesen kopiert und geklont, was zu einer riesigen Anwenderbasis geführt hat. Es gibt mehrere Familien auf dem Markt, zum Beispiel den eZ80 mit Taktfrequenzen bis zu 50 MHz oder den Z8 und den eZ8 Encore! Letzterer ist beispielsweise in der ZMOTION-Produktlinie zu finden, einer MCU-Familie, die für PIR-Bewegungserkennung optimiert ist.
Wenn Sie experimentierfreudig sind, sollten Sie den Z8FS040BSB mit seinen 4KB-Flash-Speicher und fünf GPIO-Pins in einem praktischen 8-poligen SOIC-Gehäuse ausprobieren. Der kostenlose Compiler SDCC unterstützt mehrere Z80-basierte MCUs wie die von Rabbit (jetzt Digi) und den Nintendo Gameboy.

Ultra-Low-Cost

Viele elektronische Produkte mit Mikrocontrollern werden in großen Mengen hergestellt. Denken Sie an viele Haushaltsgeräte, Uhren, elektrische Zahnbürsten, E-Zigaretten, Corona-Virentester, Chipkarten, Rauchmelder, Spielzeug und so weiter und so fort (**Bild 4**). Um eine Vorstellung von den Zahlen zu bekommen: Spielkonsolen wie der Nintendo-Gameboy, -Wii und -Switch haben jeweils die 100-Millionen-Marke verkaufter Einheiten überschritten. Stellen Sie sich vor, wie hoch die Zahlen beispielsweise bei Smartcards sind. Winzige Einsparungen bei einem Exemplar senken die Kosten für das Massenprodukt enorm! Und so gibt es halt einen großen Markt für extrem billige

Bild 4. Beispiele für Anwendungen, die durch extrem preisgünstige Mikrocontroller ermöglicht werden. (Quelle: Holtek.com)

Bild 5. Das Uno-Board NuMaker von Nuvoton besitzt ein abnehmbares NuLink Programmier/Debug-Modul. (Quelle: https://danchouzhou.blogspot.com)

Mikrocontroller. Einige dieser Mikrocontroller-Hersteller, auf die Elektronik-Bastler und -Maker aufmerksam geworden sind, sind Padauk, MDT und Holtek.

Padauk

Padauk stellt 3-Cent-MCUs her, die einmalig programmierbar (OTP) und flashbasiert sind. Die Besonderheit dieser Bausteine ist die auf FPPs (Field-Programmable Processing Units) basierende Architektur. Dabei handelt es sich um Registerbänke mit einem Programmzähler, einem Stapelzeiger, einem Akkumulator und einem Flagregister, die eine schnelle Kontextumschaltung ermöglichen. Dies ist zum Beispiel für die Interrupt-Verarbeitung und für Multitasking nützlich. Sie erinnern ein wenig an die vier Registerbänke des 8051. Da die meisten ihrer Produkte jedoch nur ein FPP haben (einige haben zwei, der PFC460 hat vier und der MCS11 acht), sind sie sehr einfache MCUs.

Der PMS150 ist ein guter Startpunkt. Eine exzellente Beschreibung dieser Bausteine finden Sie unter [4]. SDCC unterstützt den PDK14 und den PDK15, während an der Unterstützung für den PDK13 gearbeitet wird.

MDT

Wie bereits erwähnt, stellt MDT a.k.a. Micon Design Technology Klone oder Derivate von PIC-Controllern von Microchip her. Da PIC-MCUs bei vielen Herstellern sehr beliebt sind, haben auch MDT-ICs einige Aufmerksamkeit auf sich gezogen. Während der Recherchen für diesen Artikel ging die MDT-Website jedoch plötzlich offline. Bei der Internet-Suche nach MDT-Bausteinen fand ich jedoch mehrere Ergebnisse von MCU-Cracking- und Reverse-Engineering-Diensten unter anderem für MDT-MCUs, was darauf schließen lässt, dass sie weit verbreitet sind.

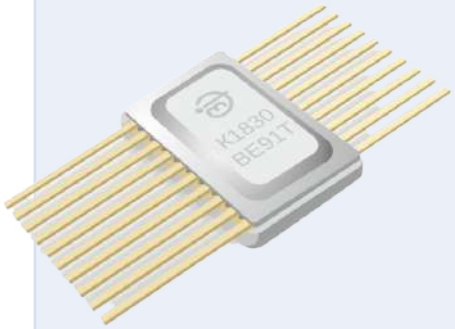


Bild 6. Der K1830BE91T von NIIET besitzt einen 8051-Kern und entspricht funktionell dem AT89C2051 von Microchip. (Quelle: https://niiet.ru)

Holtek

Einige Holtek-Produkte wurden in der Vergangenheit in Elektor-Projekten eingesetzt. Dabei handelte es sich aber nicht um Mikrocontroller, sondern um Tastatur- und RC-Kanaldecoder. Holtek stellt aber auch MCUs her, und zwar viele, von 8051-Typen über ARM-Cortex-M0 und -M3 bis hin zu Bausteinen, die auf einem eigenen proprietären Kern basieren. Wie viele der Low-Cost-MCU-Anbieter ist auch die Produktlinie von Holtek in anwendungsspezifische und Allzweck-MCUs („I/O-Typ“) unterteilt. Die Dokumentation ist gut und die HT-IDE3000-IDE (Assembler und C) ist kostenlos (wenn auch etwas schwer zu finden), aber ein Holtek-Programmiergerät ist erforderlich.

Eine MCU, den ich interessant fand, ist der analoge HT66F4550 mit zwei Operationsverstärkern und einem Audioausgang „on chip“.

Und die Großen aus Asien?

Elektor hat hunderte von mikrocontrollerbasierten Projekten veröffentlicht, und in den meisten Fällen enthielten sie einen PIC- oder AVR-Baustein von Microchip (und früher Atmel), einen ESP-Baustein von Espressif oder eine MCU mit einem ARM-Kern von NXP oder ST. Auch der MSP430 von Texas Instruments tauchte einige Male in Projekten auf. Mit Ausnahme von Espressif sind all diese Hersteller in Europa und den USA beheimatet. Das zeigt, wie voreingenommen wir sind, denn es gibt natürlich viele große asiatische Unternehmen, die MCUs herstellen.

Renesas

Einer der größten, wenn nicht sogar der größte asiatische Halbleiterhersteller ist Renesas, der sich aus Abteilungen von NEC, Hitachi und Mitsubishi zusammensetzt. Manche behaupten sogar, dass Renesas weltweit die Nummer eins unter den MCU-Anbietern ist. Langjährige Elektor-Leser erinnern sich vielleicht noch an die R8C- und R32C/111-Artikelserie von vor etwa 15 Jahren [5]. Die aktuelle RX671-Familie von 32-Bit-Mikrocontrollern ist auf schnelle Echtzeitsteuerung und berührungslose Mensch-Maschine-Schnittstellen (HMI) in Verbindung mit Näherungsschaltern und Spracherkennung spezialisiert und eignet sich perfekt für moderne, hygienische HMI-Designs. Renesas bietet auch eine große Anzahl anderer Entwicklungs- und Evaluierungsboards an, und ich möchte Sie ermutigen, diese auszuprobieren und über Ihre Ergebnisse zu berichten.

Nuvoton

Nuvoton wurde 2008 aus Winbond ausgegliedert und übernahm 2020 die Chip-Sparte von Panasonic. Nuvoton hat ein großes Angebot an 8051- und ARM-Core-MCUs und auch einige proprietäre Core-Bausteine. Im Gegensatz zu vielen Mitbewerbern verfügt Nuvoton nicht über eine eigene Toolchain. Für die 8051-Bausteine setzen sie auf Keil

und IAR, für die ARM-Bausteine auf Eclipse. Auf GitHub findet man Unterstützung für die Verwendung von SDCC bei einigen Nuvoton-Bausteinen.
Der NuMaker Uno (**Bild 5**) ist eine gute Möglichkeit, mit Nuvoton-Controllern loszulegen. Es handelt sich offensichtlich um ein Arduino-kompatibles Board mit einem NUC131-ARM-Cortex-M0 Controller. Enthalten ist ein abnehmbares Debugger/Programmiermodul namens Nu-Link, das auch mit anderen Boards verwendet werden kann. Software-Unterstützung findet man auf GitHub (OpenNuvoton). Im NuMaker-Repository finden Sie Unterstützung für mbed, Arduino, MicroPython und mehr.

Russische MCUs?

Um diesen Artikel zu vervollständigen, wollte ich einige Informationen über russische Mikrocontroller hinzufügen. Leider kann ich kein Russisch, und die meisten Websites sind (nur) auf Russisch, was die Suche nach nützlichen Informationen erschwert. Ich bin auf Milandr, Mikron und die „fabless" Firma Syntacore gestoßen, die alle RISC-V-basierte Controller entwickeln. Laut [6] hat Milandr auch eine Lizenz für ARM-Cores, aber auf ihrer Website werden solche Bauteile nicht erwähnt.
NIIET stellt den K1921VK01T her, der auf Motorsteuerungs- und Smart-Metering-Anwendungen ausgerichtet ist und auf einem ARM-Cortex-M4F-Kern basiert. OpenOCD bietet Unterstützung für diese MCU. Im Oktober 2021 kündigte NIIET einen RISC-V-basierten Controller an, der die STM32- und MSP430-Bauteile ersetzen soll, die derzeit in „zivilen Geräten" (wie sie es nennen) in Russland verwendet werden. Außerdem gibt es 8- und 16-Bit-RISC-MCUs sowie einige MCS-51- (Intel 8051) und MCS-96- (Intel 80196) Bausteine (**Bild 6**).

Eine Welt voller Möglichkeiten

In diesem Artikel wurden einige Mikrocontroller und Hersteller vorgestellt, die nicht oft in Elektor-Projekten zu sehen sind, die aber einen wichtigen Teil des weltweiten MCU-Marktes darstellen. Natürlich ist dieser Artikel bei weitem nicht vollständig, und es kann sein, dass ich einige interessante MCUs oder Hersteller übersehen habe. Während der Recherche zu diesem Artikel habe ich eine Liste von mehr als 50 aktiven Mikrocontroller-Herstellern zusammengestellt, und ich bin sicher, dass es noch viele mehr gibt. Wenn Sie weitere unbekannte, aber interessante Mikrocontroller (-Familien) kennen, die Sie den anderen Lesern vorstellen möchten, lassen Sie es mich bitte wissen.

210630-02

QUIZ

Sowohl der 4004 als auch der 4040 wurden von Federico Faggin entworfen. Welche anderen berühmten Prozessoren stammen ebenfalls von ihm?

Intel 8080, Zilog Z8 & Z80

Ein Beitrag von

Idee und Text: **Clemens Valens**
Redaktion: **Jens Nickel** und **C. J. Abate**
Übersetzung: **Rolf Gerstendorf**
Layout: **Harmen Heida**

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.

WEBLINKS

[1] Stuart Cording, „Die Geburt des Mikroprozessors", Elektor Industry 11/2021: https://www.elektormagazine.de/magazine/elektor-242/60078
[2] Pascal für 8051: https://turbo51.com/
[3] CH55xduino: https://github.com/DeqingSun/ch55xduino
[4] Padauk PMS150: https://jaycarlson.net/2019/09/06/whats-up-with-these-3-cent-microcontrollers/
[5] Gunther Ewald, „Der R8C und seine Familie", Elektor 11/2005: https://www.elektormagazine.de/magazine/elektor-200511/2270
[6] Russische Mikrocontroller: https://geek-info.imtqy.com/articles/M4836/index.html

Wie funktioniert ein **E-Paper-Display?**

Funktionsweise und Eigenschaften

Ein Beitrag von Ynvisible

Ein E-Paper-Display – Elektronisches Papier – hat viele Vorteile gegenüber herkömmlichen Display-Technologien wie LCD und LED. Daher sind E-Paper-Displays in verschiedenen Bereichen schon recht verbreitet. Aber was genau ist die Technologie hinter E-Paper-Displays? In diesem Artikel wird beschrieben, wie Elektronisches Papier funktioniert und welche Vorteile es bietet.

Bei E-Paper-Displays können verschiedene Technologien genutzt werden, z.B. Elektrophorese, Elektrobenetzung und Elektrochromie [1]. Bei Ynvisible kommt Elektrochromie zum Einsatz, der einen hoch skalierbaren, flexiblen und kosteneffizienten Siebdruck-Produktionsprozess ermöglicht. Andere Display-Hersteller wie E Ink setzen auf Elektrophorese (siehe unten).

Bei E-Paper handelt es sich um einen Display-Typ, der das Aussehen von gewöhnlicher Tinte auf Papier nachahmt und sehr energiesparend ist. E-Paper-Displays verfügen in der Regel über ein sogenanntes Bildgedächtnis – es braucht daher nicht viel Strom, um die Anzeige aufrechtzuerhalten. Strom wird vor allem benötigt, wenn sich die Anzeige ändert. Daher eignen sich E-Paper-Displays am besten für Anzeigen, die nur selten verändert werden, wie z.B. Beschilderungen und Etiketten.

E-Paper-Displays reflektieren Licht, anstatt selbst welches abzustrahlen. Daher lassen sie sich auch bei direkter Sonneneinstrahlung sehr gut ablesen. Aufgrund des geringen Stromverbrauchs, hohen Reflexionsvermögens, hohe Kontraste, der guten Lesbarkeit in hellem Umgebungslicht, geringer Dicke und des großen Betrachtungswinkels eignen sie sich für viele Anwendungen. E-Paper-Displays werden z.B. häufig in E-Book-Readern oder Echtzeit-Informationen für Busse, elektronische Regaletiketten (ESL = Electronic Shelf Label), digitale Speisekarten, Verkehrsschildern und bei der Distributionslogistik eingesetzt.

Wie funktionieren E-Paper-Displays?

Die verschiedenen Technologien für E-Paper-Displays basieren auf unterschiedlichen Funktionen.

Elektrophorese-Displays (z.B. E Ink)

Elektrophorese-Displays (z.B. EInk) Elektrophorese-Displays wie etwa das von E Ink enthalten Millionen von winzigen, mit einer klaren Flüssigkeit gefüllten Mikrokapseln. Enthalten sind kleine Partikel mit unterschiedlicher Farbe und elektrischer Ladung. Wenn an einer Displayfläche eine positive Ladung angelegt wird, werden negativ geladene Teilchen angezogen und positiv geladene Teilchen abgestoßen. Diese Bewegung der Teilchen zur Oberfläche oder davon weg bewirkt, dass eine bestimmte Farbe sichtbar wird.

Elektrobenetzungs-Displays (z.B. Etulipa)

Der Hersteller Etulipa baut Displays auf der Basis von Elektrobenetzungs-Technologie. Hierbei wird Oberflächenspannung einer Flüssigkeit genutzt. Ein sogenanntes Electrowetting-Display enthält winzige Zellen aus einer transparenten polaren Flüssigkeit und einem farbigen Öl, das eine hydrophobe Oberfläche bedeckt. Durch Anlegen einer niedrigen Spannung an die Zellen zieht sich das Öl zu einem kleinen Tröpfchen zusammen. Dadurch entstehen Pixel in der Form von offenen oder geschlossenen optischen Schaltern. Diese Displays eignen sich nicht nur für die Anzeige von Texten, sondern auch für Bilder, Fotos und sogar Videos.

Elektrochromatische Displays (z.B. Ynvisible)

Elektrochromatische Displays basieren auf einem Verfahren, bei dem ein

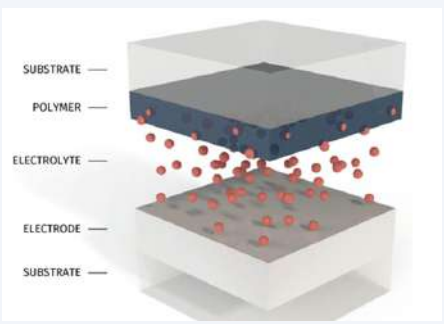


Bild 1. Die Displays von Ynvisible werden in einem hoch skalierbaren und kostengünstigen Verfahren hergestellt.

Bild 2. Die Displays von Ynvisible sind flexibel und leicht.

Bild 3. Die prominenteste Anwendung von E-Paper-Displays sind E-Reader.

Material seine Farbe ändert, wenn elektrischer Strom fließt. Das elektrochromatische Material unterliegt einer Oxidation bzw. Reduktion beim Anlegen einer Spannung in Gegenwart eines Elektrolyten. Dies führt zu einem fein abstimmbaren Farbwechsel. Die elektrochromatischen Displays von Ynvisible benötigen äußerst wenig Strom und könnten mit der Energie einer Knopfzelle etwa 50 Jahre lang betrieben werden. Ynvisible stellt Displays im hochgradig skalierbaren und preiswerten Siebdruckverfahren her (siehe **Bild 1**), was sie preiswerter als die mit anderen E-Paper-Technologien hergestellten Alternativen macht.

Hauptmerkmale von E-Paper-Displays

Die 5 wichtigsten Eigenschaften der E-Paper-Technologie sind:

1. Bistabilität

E-Paper-Displays verfügen über verschiedene Arten von Bistabilität (= das Display behält ein Bild auch ohne Stromquelle bei). Ynvisible-Displays können ein Bild bis zu 15 Minuten lang anzeigen, bevor ein kleiner Auffrischungsimpuls erforderlich ist. Dies nennt sich Semi-Bistabilität.

2. Geringer Stromverbrauch

Da E-Paper-Displays weder eine Hintergrundbeleuchtung noch Strom zum halten eines Bildes benötigen, haben sie einen extrem niedrigen Stromverbrauch. Dies prädestiniert sie besonders für netzunabhängige Anwendungen.

3. Reflexionsvermögen

E-Paper-Displays reflektieren Licht und strahlen keines ab - ein weiterer Grund für geringen Energieverbrauch. Außerdem kann man diese Displays im Gegensatz zu anderen Displaytypen auch bei direkter Sonneneinstrahlung sehr gut ablesen.

Bild 4. Elektronisches Papier eignet sich perfekt für digitale Beschilderung und öffentliche Hinweistafeln.

Bild 5. E-Paper-Displays werden häufig im Logistik-Management eingesetzt, um Waren in Echtzeit zu verfolgen und zu überwachen.

Bild 6. E-Papier-Displays helfen auch bei der Bekämpfung von Produktpiraterie.

Bild 7. E-Papier-Displays sind eine kostengünstigere Alternative zu herkömmlichen Papieretiketten.

4. Lesbarkeit

E-Paper-Displays sind aufgrund ihres großen Sichtwinkels und des hohen Kontrastes sehr gut lesbar und eignen sich daher ideal für wichtige Mitteilungen wie etwa Verkehrshinweise.

5. Flexibilität

Einige E-Paper-Displays überzeugen durch eine hohe Flexibilität (Ynvisible-Displays sind so flexibel, dass sie sich um einen Bleistift wickeln lassen) und durch ein geringes Gewicht (**Bild 2**). Dies sind gerade beim Versand wichtige Faktoren.

Die E-Paper-Displays [2] von Ynvisible sind vollständig an Kundenwünsche anpassbar und können so konstruiert werden, dass sie auch anspruchsvolle Anforderungen von Geräteherstellern erfüllen.

E-Paper-Displays gibt es schon seit etlichen Jahren. Sie wurden zunächst vor allem in E-Book-Readern eingesetzt, aber aufgrund ihrer stromsparenden Eigenschaften, ihres geringen Gewichts und ihrer Flexibilität (**Bilder 3** bis **7**) [3][4] sind sie inzwischen in vielen weiteren Anwendungen zu finden.

Der beste Weg, um herauszufinden, ob diese Displays für Ihre Anwendungen geeignet sind, ist das praktische Ausprobieren. Das E-Paper-Display-Kit [5] von Ynvisible eignet sich besonders gut für erste Praxis-Erfahrungen. Interessierte Entwickler finden weiterführende Informationen zur Ansteuerung der Displays unter [6].

220273-02

WEBLINKS

[1] Elektrochromatische Displays: www.ynvisible.com/news-inspiration/what-is-an-electrochromic-display
[2] Kundenspezifische Prototypen: www.ynvisible.com/news-inspiration/cost-effective-custom-display-prototypes
[3] Echtheitsnachweise für Waren: www.ynvisible.com/news-inspiration/authentication-solutions-printed-electronics
[4] Neue Anwendungen für preiswerte Displays: www.ynvisible.com/solutions
[5] E-Paper-Display-Kit: www.ynvisible.com/product/e-paper-display-kit
[6] Infos zur Ansteuerung: www.ynvisible.com/news-inspiration/how-to-drive-ynvisible-e-paper-display

CLUE - ein Clou von Adafruit?

Angriff auf den micro:bit

Von **Tam Hanna**

Der BBC micro:bit war ein großer Erfolg - weit über den edukativen Markt hinaus, für den er gedacht war. Mit einem vollwertigen Display und weit mehr Speicher tritt nun der CLUE von Adafruit an. Dank Bluetooth LE und einer Vielzahl von integrierten Sensoren empfiehlt er sich vor allem für kleinere IoT-Projekte

Spätestens seit dem Erfolg von Arduino und Raspberry Pi ist offensichtlich, dass man mit „Schulungscomputern" aller Herren Couleur Geld verdienen kann. Die BBC schickte im Jahr 2016 mit dem micro:bit einen Einplatinencomputer ins Rennen, der statt eines vollwertigen Linux-fähigen Prozessors „nur" ein Bluetooth-SoC aus dem Hause Nordic Semiconductor mitbrachte.

Der Rest ist eine für den Autor dieser Zeilen nicht wirklich verständliche Geschichte: Mittlerweile gibt es zum Beispiel in der Slowakei Unternehmen, die sich ausschließlich über den Vertrieb des micro:bit-Ökosystems ernähren [1].

Der alte Kalauer, dass man an einem „gut gefüllten" Topf nicht allzu lang alleine frisst, gilt auch im Bereich des Prozessrechnerwesens. Die Weiterentwicklung im Bereich der Bluetooth-SoCs hat dazu geführt, dass der mit 16 MHz arbeitende Prozessor des micro:bit (insbesondere auch mit seinen nur 16 KB SRAM) heute wie ein Methusalem aussieht. Zudem ist das aus 5×5 Leuchtdioden bestehende Display für die Ausgabe komplizierterer Grafiken nicht geeignet.

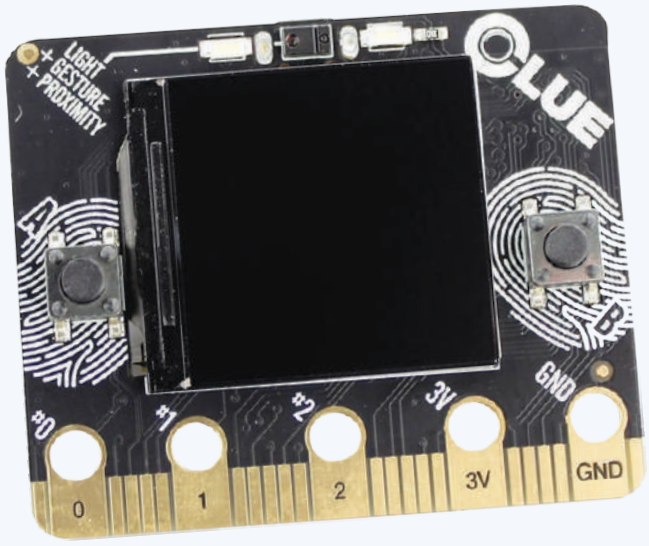


Bild 1. Der Angreifer von vorne ...

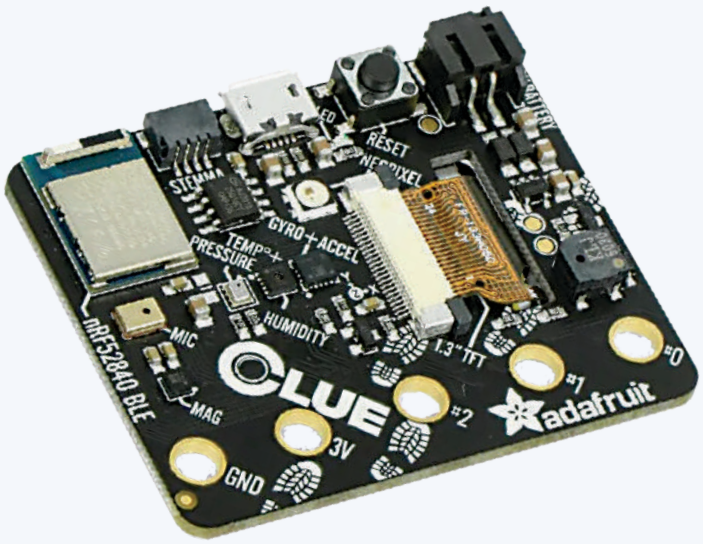


Bild 2. ... und von hinten.

Eigenschaften des CLUE

- Nordic nRF52840 Bluetooth LE Prozessor: 1 MB Flash, 256 KB RAM, 64 MHz Cortex M4 Prozessor
- 1,3" 240×240 Color IPS-TFT-Display für Texte und Grafiken
- Stromversorgung über eine beliebige 3...6-V-Quelle (interner Regler und Schutzdioden)
- Zwei Benutzertasten und eine Reset-Taste
- Bewegungssensor, Accelerometer/Gyrometer, Magnetometer
- Näherungs-, Licht-, Farb- und Gestensensor
- Mikrofon/Schallsensor
- SHT-Sensor für Luftfeuchtigkeit
- BMP280-Sensor für Temperatur und barometrischer Druck/Höhe
- RGB-NeoPixel-Anzeige-LED
- 2 MB interner Flash-Speicher für Daten, Bilder, Schriftarten oder CircuitPython-Code
- Summer/Lautsprecher
- Zwei helle weiße LEDs an der Vorderseite zur Beleuchtung / Farberkennung
- Qwiic / STEMMA QT-Anschluss für das Hinzufügen weiterer Sensoren, Motorsteuerungen oder Displays über I²C. GROVE-I²C-Sensoren lassen sich über ein Adapterkabel anschließen.
- Programmierbar mit Arduino IDE oder CircuitPython

Attacke von Adafruit

Spätestens als Nordic Semiconductor den nRF52840 vorstellte - ebenfalls ein einkerniges Bluetooth-SoC, dessen ARM-Prozessor allerdings bis zu 64 MHz erreicht und 256 KB RAM hat - war der „Angriffsvektor" klar. Die **Bilder 1** und **2** zeigen das Resultat - ein System, das dem BBC micro:bit durchaus ähnlich sieht.

Neben dem auf diesen Fotos nicht direkt erkennbaren SoC fällt auf der Vorderseite der wesentlich größere Bildschirm auf - statt mit Leuchtdioden bekommen wir es mit einem 240 × 240 Pixel großen Farbdisplay zu tun, das allerdings nicht auf organischer, sondern auf klassischer IPS-LCD-Technologie basiert.

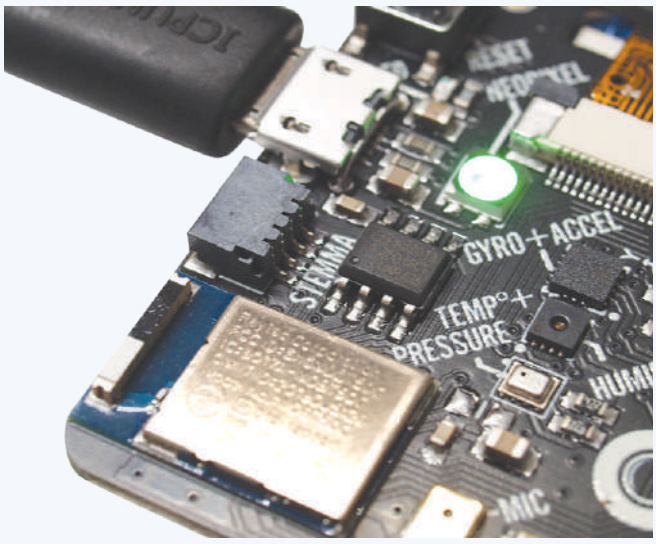


Bild 3. Der STEMMA-Port erlaubt dem Adafruit CLUE die Kontaktaufnahme zu Erweiterungen.

Ein „netter Touch" des Moduls ist der auf der Rückseite befindliche und in **Bild 3** gezeigte Steckverbinder. Er exponiert im hauseigenen Format einen I²C-Bus, über den sich unbürokratisch weitere Sensoren anschließen lassen. Zudem gibt es einen Adapter auf das von Seeed betriebene Grove-Format, für das verschiedene fair bepreiste Sensoren im Handel angeboten werden.

Beachten Sie, dass der CLUE nur teilweise mit dem großen Vorbild kompatibel ist. Der Verbinder auf der Unterseite ist physikalisch identisch - ob der Nutzung eines anderen Displays gilt allerdings, dass der „Gutteil" der Gehäuse für den BBC micro:bit nicht auf den Adafruit CLUE passt.

Der Autor testete diese Hypothese mit dem unter [2] bereitstehenden ThingiVerse-Gehäuse von domw. Die Vorderseite passte offensichtlich nicht, da das Display des CLUE wesentlich größer war als die LED-Matrix des Originals der BBC. Besonders überraschend empfand der Autor in diesem Zusammenhang, dass die Rückwand des Gehäuses trotz der zusätzlichen Stecker problemlos passte - dies lag allerdings daran, dass das Gehäusedesign vergleichsweise großzügig gestaltet war. Hat Ihr Gehäusedesigner „knapper" konzipiert, so wäre wahrscheinlich schon an dieser Stelle komplettes Ende.

Eine Frage der Programmierung

Als „Ausbildungssystem" ist der micro:bit fernab von klassischen Embedded-Entwicklungsumgebungen wie ARM Keil angesiedelt - dies mag für Embedded-Puristen ärgerlich sein, ist in der Praxis aber notwendig, weil an vielen Hochschulen nicht ausreichend kompetentes Personal zum Debuggen von C++-Programmfehlern zur Verfügung steht (glauben Sie dem Autor: Kadetten finden magisch die debilsten Wege).

Stattdessen wird im Allgemeinen auf ein Quadriumvirat aus Arduino-IDE, CircuitPython, MakeCode und Scratch gesetzt. Im Fall des

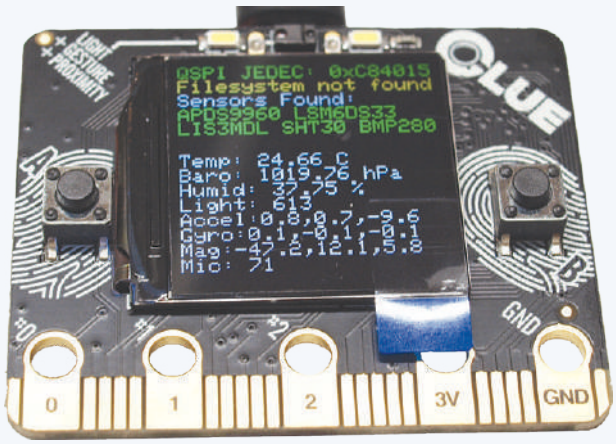


Bild 4. Diese App zeigt die von den Sensoren zurückgelieferten Informationen.

Bild 5. Die Python-Runtime exponiert ebenfalls ein virtuelles Laufwerk.

CLUE gilt, dass nur zwei der Systeme zur Verfügung stehen: An MakeCode wird ohne Liefertermin gearbeitet, zu Scratch gibt es keine Informationen.
Wichtig ist, dass Adafruit den CLUE mit einem seriellen Bootloader ausstattet, um das „Deployment" von Code - ganz analog zum Raspberry Pi Pico - zu ermöglichen.
Für einen ersten kleinen Versuch starten wir zunächst einmal CircuitPython.
Verbindet man ein jungfräuliches Board über die auf der Rückseite befindlichen MicroUSB-Buchse mit dem Rechner, so zeigt das Display eine Statusseite (**Bild 4**), die Informationen über den Betriebszustand anbietet.
Doppeltes Drücken des auf der Rückseite befindlichen Reset-Tasters sorgt im ersten Schritt dafür, dass die Bildschirmanzeige ob des im Displaycontroller enthaltenen Framebuffers einfriert. Die angeschlossene Workstation (beim Autor läuft sie unter Linux) sieht das Aufscheinen eines neuen USB-Laufwerks, auf dem sich Kompilate ablegen lassen.
Interessanterweise ist der Adafruit CLUE für den Rechner „immer" sichtbar - ist er nicht im Bootloader-Modus, so erkennt ihn *dmesg* folgendermaßen:

```
tamhan@TAMHAN18:~$ dmesg
. . .
[28292.202193] usb 1-2.7: Manufacturer: Adafruit LLC
[28292.202195] usb 1-2.7: SerialNumber: 7687A137B6FDB874
[28292.204040] cdc_acm 1-2.7:1.0: ttyACM0: USB ACM device
```

Nach dem Doppel-Druck auf den Button sehen wir stattdessen nach folgendem Schema ein USB-Laufwerk:

```
tamhan@TAMHAN18:~$ dmesg
. . . .
[28371.624193] sd 10:0:0:0: Attached scsi generic sg6 type 0
```

Wichtig ist, dass dieses Laufwerk nicht „ewig" aktiviert bleibt - die Firmware setzt sich nach etwa 30 Sekunden Totzeit wieder in den normalen Betrieb zurück.

Dateien suchen

Wir besuchen im ersten Schritt die URL *https://circuitpython.org/board/clue_nrf52840_express/*, um die Datei *adafruit-circuitpython-clue_nrf52840_express-en_US-6.1.0.uf2* herunterzuladen. Sie enthält die Runtime, die Sie auf dem USB-Laufwerk ablegen.
Interessant ist in diesem Zusammenhang, dass Sie im Laufwerk zudem eine Datei namens *CURRENT.UF2* finden - sie erlaubt das einfache Abernten des Kompilats, dass gerade im Speicher des Prozessrechners liegt.
Lustigerweise wird die Runtime nicht mit einem vollständigen Bibliotheks-Set ausgeliefert, das alle enthaltenen Sensoren abdeckt. Wir müssen stattdessen die URL *https://circuitpython.org/libraries* aufrufen, wo wir das Archiv *adafruit-circuitpython-bundle-6.x-mpy-20210329.zip* herunterladen und es in einen bequemen Ordner im Dateisystem extrahieren.
Spätestens an dieser Stelle sollten Sie einen weiteren Blick auf den Bildschirm des CLUE werfen - die Runtime gibt die Inhalte der Konsole permanent am Bildschirm aus. Ein netter Touch ist, dass das Gerät auf dem PC - wie in **Bild 5** gezeigt - den internen Speicher der Python-Arbeitsumgebung exponiert.
Wichtig ist, dass Sie im *Libs*-Ordner einerseits die folgenden Ordner aus dem Archiv unterbringen:

adafruit_apds9960
adafruit_bus_device
adafruit_display_shapes
adafruit_display_text
adafruit_lsm6ds
adafruit_register

Als wäre dies noch nicht genug der Arbeit, mutet Adafruit den Käufern auch noch das Zusammensuchen der folgenden

Einzeldateien zu - warum man nicht ein „schlüsselfertiges“ Archiv anbietet, ist dem Rezensenten nicht klar:

adafruit_bmp280.mpy
adafruit_clue.mpy
adafruit_lis3mdl.mpy
adafruit_sht31d.mpy
adafruit_slideshow.mpy
neopixel.mpy

Code-Beispiel

Für einen ersten kleinen Gehversuch mit der Python-Umgebung bietet sich das unter *https://learn.adafruit.com/adafruit-clue/clue-spirit-level* bereitstehende Beispiel an - es realisiert einen Lagemesser, nutzt dabei aber eine Reihe CLUE-spezifischer Idiome.
Die erste Amtshandlung des Codes besteht darin, eine Gruppe von Bibliotheken einzubinden:

```
import board
import displayio
from adafruit_display_shapes.circle import Circle
from adafruit_clue import clue
```

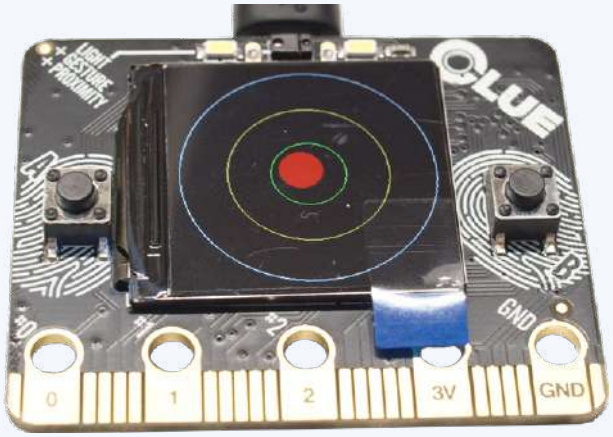


Bild 6. Eine Wasserwaage auf Basis eines Eval-Boards.

Neben dem `clue`-Objekt, das diverse boardbezogene Funktionen zur Verfügung stellt, ist hier auch der Import der `Circle`-Klasse interessant. Der GUI-Stack erlaubt nicht nur das „direkte" Zeichnen in einen Framebuffer, sondern unterstützt auch die Arbeit mit Objekten, die von der Firmware in am Bildschirm erscheinende Elemente umgesetzt werden.
Im nächsten Akt kümmert sich die Firmware darum, einen Verweis auf das Display zu verschaffen und ein Display-Group-Objekt zusammenzustellen:

```
display = board.DISPLAY
clue_group = displayio.Group(max_size=4)
```

Das `clue_group`-Objekt ist insofern interessant, als es ein an einen DOM-Baum erinnerndes Eltern-Element generiert, in das unser Code danach mehr oder weniger beliebige Objekte für die Anzeige schreibt.
Aus der Logik folgt, dass die nächste Amtshandlung des Programms darin besteht, drei für die Darstellung der Stärke der Auslenkung zuständigen Kreise zu erzeugen und für die Ausgabe anzumelden:

```
outer_circle = Circle(120, 120, 119, outline=clue.WHITE)
middle_circle = Circle(120, 120, 75, outline=clue.YELLOW)
inner_circle = Circle(120, 120, 35, outline=clue.GREEN)
clue_group.append(outer_circle)
clue_group.append(middle_circle)
clue_group.append(inner_circle)
```

Als Nächstes folgen noch einige weitere Housekeeping-Tasks, deren Sinn sich am einfachsten beim Blick auf das im Artikel weiter unten gezeigte Beispiel-Rendering erschließt:

```
x, y, _ = clue.acceleration
bubble_group = displayio.Group(max_size=1)
level_bubble = Circle(int(x + 120), int(y + 120), 20,
   fill=clue.RED, outline=clue.RED)
bubble_group.append(level_bubble)

clue_group.append(bubble_group)
display.show(clue_group)
```

Zu guter Letzt benötigen wir eine Arbeitsschleife, die die von der Adafruit-Bibliothek über das Attribut `acceleration` ausgegebenen Lagewerte analysiert und in die Koordinaten-Eigenschaften des „Ziel-Objekts" weiterschreibt:

```
while True:
    x, y, _ = clue.acceleration
    bubble_group.x = int(x * -10)
    bubble_group.y = int(y * -10)
```

Der bequemste Weg zur „schnellen" Ausführung vom Code am CLUE ist die in Bild 5 gezeigte Datei *code.py*. Sie wird von der CircuitPython-Firmware automatisch im Rahmen jedes Starts ausgeführt. **Bild 6** zeigt, was Sie sich erwarten dürfen.
Ob des Vorhandenseins eines Bluetooth-Transmitters ist das System auch zur Kommunikation mit dem Rechner befähigt. Unter [3] stellt Adafruit ein sehr lustiges Beispiel zur Verfügung, das die Verwendung einer in Google Chrome implementierten Web-Bluetooth-API illustriert.

Und jetzt mit C

Python mag ein schneller Weg sein, um „unbürokratisch" Ergebnisse aus einem Prozessrechner zu bekommen. Die maximale Performance erreicht man aber durch C.
Insbesondere auf einem Einkern-Funksystem ist das Realisieren von kommunikativem Code schwierig, wenn es nicht zu Timing-Problemen kommen soll. Damit zwingt Adafruit Entwicklern die Nutzung der Arduino-IDE mehr oder weniger auf. Im Hintergrund arbeitet dann ein Echtzeit-Betriebssystem, das sich um die Zuweisung von Rechenleistung an die verschiedenen Aufgaben kümmert.
Unter Linux ist im ersten Schritt ein Erweiterungspaket erforderlich, das der Arduino-IDE (ab Version 1.8.6) die Kommunikation mit dem nicht-standardisierten Bootloader des CLUE ermöglicht:

```
tamhan@TAMHAN18:~$ pip3 install --user adafruit-nrfutil
Collecting adafruit-nrfutil
  . . .
Successfully installed adafruit-nrfutil-0.5.3.post13
```

Als Nächstes müssen wir im Board Manager die URL *https://www.adafruit.com/package_adafruit_index.json* einpflegen, um das Board-Package *Adafruit nRF52* zum Download verfügbar zu machen. Nach getaner Arbeit steht das Board unter *Tools* > *Board* > *Adafruit CLUE* zur Verfügung.
Wie im Fall von CircuitPython gilt leider auch hier, dass die

WEBLINKS

[1] StreamIT, s. r. o.: https://edutronik.sk/v2/
[2] Gehäuse: https://www.thingiverse.com/thing:1767446
[3] Demo-Applikation: https://learn.adafruit.com/bluefruit-dashboard-web-bluetooth-chrome
[4] Infos: https://learn.adafruit.com/adafruit-clue?view=all

Einrichtung von Bibliothek & Co. ein arbeitsintensiver Prozess ist. Weitere Informationen dazu finden sich unter [4].

Lohnt es sich?

Wer den CLUE in die Hand nimmt, bekommt eine durchaus attraktive Evaluationsplattform, die - insbesondere dank des Bildschirms - im Bereich Interfacing angenehm zu bedienen ist. Andererseits steht der im Vergleich zum BBC micro:bit vergleichsweise hohe Preis - das „Original" kostet wesentlich weniger. Als wäre dies noch nicht genug, ist der CLUE eben doch nicht zu 100% mit dem micro:bit kompatibel - die traurige Lebenserfahrung lehrt, dass ein „insignifikantes Detail", das von 99% der Anwendungen nicht tangiert wird, genau im vorliegenden Projekt für Probleme sorgt.
Wer eine „saubere" Nordic-basierte Plattform realisieren möchte, ist mit einem klassischen Evaluationsboard besser bedient. Unterm Strich ist der CLUE also ein liebenswertes Produkt für jene, die einer kompatiblen Kleinstserie von BBC micro:bit-Projekten zu mehr Leistungsfähigkeit verhelfen wollen.

210395-02

Haben Sie Fragen oder Kommentare?

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

Ein Beitrag von

Text und Bilder: **Tam Hanna**
Redaktion: **Jens Nickel**
Layout: **Giel Dols**

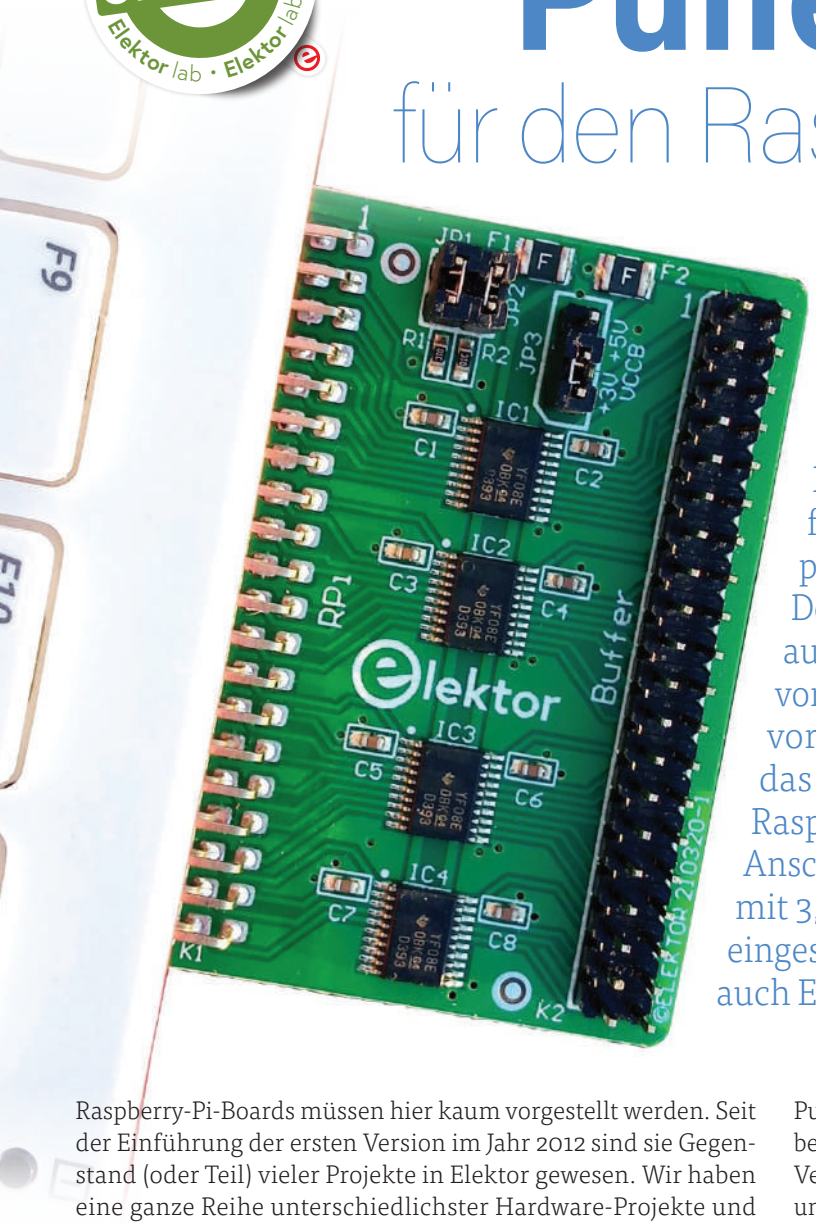
PASSENDE PRODUKTE

- Adafruit CLUE
  www.elektor.de/19512

Puffer-Board

für den Raspberry Pi 400

Schützen Sie die I/Os!



Von **Ton Giesberts** (Elektor)

Der im November 2020 in den Markt eingeführte Raspberry Pi 400 ist eigentlich ein Raspberry Pi 4, der in einer Tastatur eingebaut ist. Der GPIO-Erweiterungsanschluss befindet sich auf der Rückseite des Gehäuses. Der Anschluss von zusätzlicher Hardware birgt immer Risiken, vor allem beim Prototyping. Das Pufferboard, das wir hier vorstellen, wurde speziell für den Raspberry Pi 400 entwickelt. Es ermöglicht den Anschluss externer Logik für alle 26 GPIOs, sowohl mit 3,3-V- als auch mit 5-V-Versorgung. Die dafür eingesetzten Puffer/Pegelwandler bieten zusätzlich auch ESD-Schutz.

Raspberry-Pi-Boards müssen hier kaum vorgestellt werden. Seit der Einführung der ersten Version im Jahr 2012 sind sie Gegenstand (oder Teil) vieler Projekte in Elektor gewesen. Wir haben eine ganze Reihe unterschiedlichster Hardware-Projekte und Erweiterungsplatinen für diese Prozessorboards vorgestellt. Die jetzt hier gezeigte Schaltung ist nicht neu; die erste Version wurde bereits 2015 auf Elektor Labs [1] besprochen. Im Jahr 2018 wurde dieser Entwurf an den erweiterten 40-poligen I/O-Anschluss angepasst, der seit der Einführung des Raspberry Pi B+ [2] Standard für den Anschluss zusätzlicher Hardware an den Raspberry Pi ist. Und nun wird die letztere Version an eines der neueren Mitglieder der Raspberry-Pi-Produktfamilie angepasst - den Raspberry Pi 400. Ganz einfach gesagt, ist der 400er ein Raspberry 4, der in eine Tastatur eingebaut ist. Er ähnelt damit den einst berühmten Computern der 1980er Jahre wie dem Commodore 64, Sinclair Spectrum, Acorn BBC und ähnlichen. Von den Spezifikationen her ist der Raspberry Pi natürlich um ein Vielfaches leistungsfähiger als seine mittlerweile antiken Vorgänger, aber es gibt auch eine auffällige Ähnlichkeit, nämlich einen GPIO-Erweiterungsanschluss auf der Rückseite des Gehäuses, an den der Benutzer externe (zugekaufte oder selbst entworfene) Hardware anschließen kann. Der Anschluss von zusätzlicher Hardware birgt aber immer Risiken, vor allem in der Entwicklungsphase. Die hier vorgestellte Pufferplatine vermindert das Risiko, dass der Raspberry Pi dabei beschädigt wird. An die Pufferplatine kann externe Logik mit einer Versorgungsspannung von 3,3 V oder 5 V angeschlossen werden, und die dafür verwendeten Puffer/Pegelwandler bieten einen eingebauten ESD-Schutz.

Die Hardware

Der Schaltplan des Projekts (**Bild 1**) ist exakt derselbe wie im Artikel aus dem Jahr 2018. Die verwendeten 8-Bit-Puffer/Spannungswandler TXS0108E sind bidirektional, und jeder A-Port- und B-Port-Pin verfügt über einen Pull-up-Widerstand. Der Pull-Down-Widerstand eines GPIOs des Raspberry Pi liegt typischerweise in der Größenordnung von 40...60 kΩ, was zu hoch ist, um den I/O richtig „herunterzuziehen“, wenn die Pufferplatine eingesteckt ist. Bei dieser Eingangskonfiguration ist also der logische Pegel nicht richtig low; der Pull-Down funktioniert nicht wie vorgesehen. Die I/Os haben separate Versorgungspins zur Raspberry-Pi-Seite (VCCA) und zur Außenwelt (VCCB). Jeder A-Port-I/O des TXS0108E verfügt über einen Pull-up-Widerstand zu VCCA, der mit der +3,3-V-Spannungsversorgung des Raspberry Pi 400 verbunden ist, und jeder B-Port-I/O besitzt einen Pull-up-Widerstand zu VCCB. VCCB für den Pegel der I/Os auf K2 kann am Jumper JP3 entweder auf +3,3-V- oder auf +5-V-Logik eingestellt werden. Die Pull-ups der Puffer haben einen Wert von 40 kΩ, wenn

der Ausgang auf Low steht, und einen Wert von 4 kΩ, wenn der Ausgang auf High steht. Es handelt sich also faktisch um Open-Drain-Ausgänge. Wenn zum Beispiel eine LED vom Ausgang des Puffers nach Masse angeschlossen wird, entsteht ein Spannungsteiler, wenn ein zusätzlicher Vorwiderstand verwendet wird. Eine ohmsche Last am Ausgang führt zu einer Verringerung des logischen High-Pegels, etwas, das Sie beachten sollten!

Es gibt zwei rücksetzbare 0,5-A-PPTC-Sicherungen (F1 und F2) in den Versorgungsleitungen zwischen K1 und K2 auf der Pufferplatine, um die +5-V- und +3,3-V-Stromversorgung des Raspberry Pi 400 zu schützen. Für die Implementierung eines I²C-Busses zur Kommunikation mit externer Hardware (GPIO2 ist SDA und GPIO3 ist SCL) können die zusätzlichen Pull-up-Widerstände R1 und R2 durch die Jumper JP1 und JP2 aktiviert werden.

Während des Hochfahrens des Raspberry Pi werden GPIO0 (ID_SD) und GPIO1 (ID_SC) verwendet, um das EEPROM eines I²C-HAT (Hardware Attached on Top) zu lesen. Nach dem Booten können diese GPIOs wie die 26 anderen verwendet werden, aber es muss darauf geachtet werden, dass das System nicht beim Booten beeinträchtigt wird, wenn ein I²C-HAT montiert ist. Um das Lesen von GPIO0 und GPIO1 während des Bootens zu verhindern, fügen Sie den folgenden Eintrag in die Konfigurationsdatei */boot/config.txt* ein:

```
force_eeprom_read=0
```

Weitere Informationen über die Datei config.txt finden Sie in der Raspberry-Pi-Dokumentation [3].

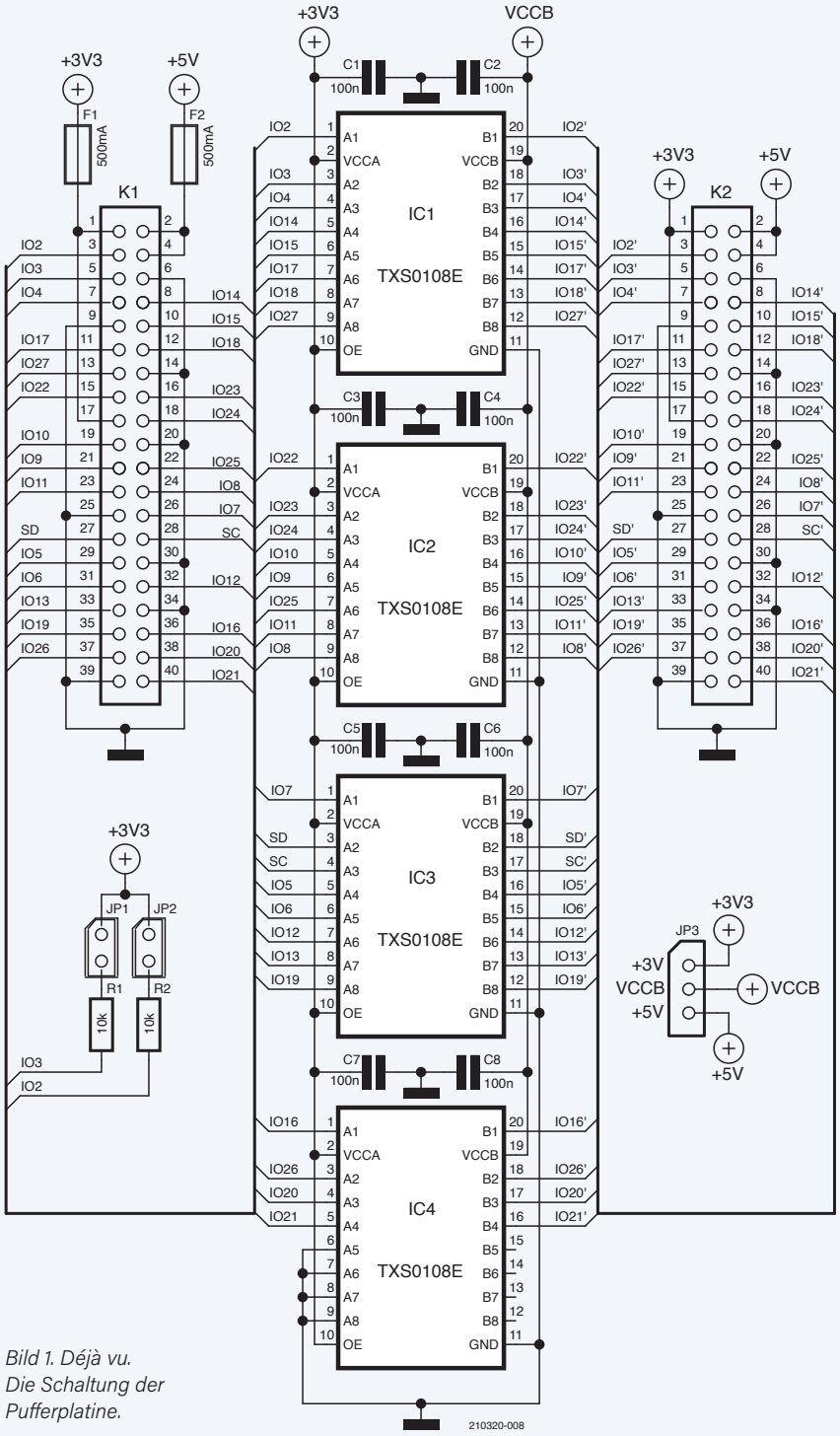


Bild 1. Déjà vu. Die Schaltung der Pufferplatine.

Das Platinenlayout

Der Schaltplan mag alt sein, aber die Platine (**Bild 2**) wurde speziell für den Raspberry Pi 400 angepasst. Die Gerber-Dateien für diese neue Platine stehen zum Download bereit, so dass Sie sie beim Platinenhersteller Ihrer Wahl bestellen können. Viel bequemer ist es natürlich, die komplett bestückte Pufferplatine im Elektor Store [4] zu kaufen.

Für den Stecker auf der Raspberry-Pi-400-Seite (K1) wird eine rechtwinklige Buchse verwendet, so dass die Pufferplatine in den GPIO-Header auf der Rückseite des Gehäuses eingesteckt werden kann (**Bild 3**). Beim Anschluss K2, der die gepufferten I/Os bereitstellt, handelt es sich um einen vertikalen 40-poligen Standardheader (K2). Das Board ist mit 55 x 44 mm einschließlich der Buchse K1, die über die Kante der Platine herausragt, etwas kleiner als die ursprüngliche Pufferplatine aus dem Jahr 2018. Im Vergleich zur Originalplatine 150719-1 sind die beiden Stiftreihen von K1 vertauscht, da hier eine Buchse verwendet wird. Das Anbringen einer vertikalen 40-poligen Stiftleiste für K1, um diese Pufferplatine wie bei der älteren Version des Puffers über ein Flachkabel mit einem „normalen" Raspberry Pi zu verbinden, funktioniert hier zwar nicht, man kann, wie **Bild 4** zeigt, dieses Modul aber einfach sekrecht direkt auf den GPIO-Header eines Raspberry Pi 2...4 aufstecken.

STÜCKLISTE

Widerstände:

R1,R2 = 10 k, 100 mW, 1 %, SMD 0603

Kondensatoren:

C1…C8 = 100 n, 50 V, 10 %, X7R, SMD 0603

Halbleiter:

IC1…IC4 = TXS0108EPWR, SMD TSSOP-20

Außerdem:

K1 = 2x20-polige Stiftleiste, rechtwinklig, Raster 2,54 mm
K2 = 2x20-polige Stiftleiste, gerade, Raster 2,54 mm
JP1,JP2 = 1x2-polige Stiftleiste, gerade, Raster 2,54 mm
JP3 = 1x3-polige Stiftleiste, gerade, Raster 2,54 mm
JP1,JP2,JP3 = Jumper, Raster 2,54 mm
F1,F2 = Zurücksetzbare PPTC-Sicherung, SMD
(Polyfuse, 1210L050YR Littelfuse)
Platine 210320-1 v1.0

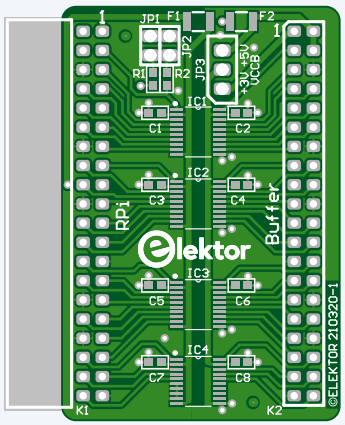


Bild 2. Das Layout der neuen Pufferplatine.

Der Ausgangsanschluss K2 kann über ein kurzes 40-poliges Flachbandkabel mit zwei 2x20-Buchsen mit externen Schaltungen verbunden werden, oder nur eine einzelne Buchse verwenden, die mit angelöteten kurzen Käbelchen angeschlossen wird, oder gar nur über einzelne Buchsen mit Drähten. Seien Sie jedoch vorsichtig, wenn Sie eine 40-polige Buchse auf K2 drücken oder sie von der Platine abziehen. Tun Sie dies nicht, solange die Pufferplatine noch im Raspberry Pi 400 steckt, da dies einen gewissen Kraftaufwand erfordert und beim Herumwerkeln der GPIO-Header des Raspberry Pi 400 beschädigt werden könnte.

Testen der Pufferplatine

Zwei sehr einfache Python-Programme zum Testen der Pufferplatine - entlehnt aus dem älteren Projekt - stehen auf der Elektor-Labs-Seite dieses Projekts als Download *210320-11.zip* bereit [5]. Eines testet alle GPIOs als Output (*Check_all_GPIOs_as_output.py*), das andere testet alle GPIOs als Input (*Check_all_GPIOs_as_input.py*). Im Raspberry Pi OS (aka Raspbian) genügt ein Doppelklick auf eine der Dateien, um die Standard-IDE für Python zu öffnen. Dann wählen Sie RUN, um den Test zu starten.

Wenn man die GPIOs als Ausgang testet, braucht man nur eine einzelne Low-Current-LED zwischen einem Pin und GND, um zu sehen, ob ein Ausgang funktioniert oder nicht. Als Vorwiderstand für die LED kann ein 1,8-kΩ-Widerstand verwendet werden, wobei der Wert nicht wirklich kritisch ist. Er begrenzt den Strom durch die LED, wenn diese direkt an die positive Versorgungsspannung angeschlossen ist. Die Ausgänge werden nacheinander in vier Gruppen IOA bis IOD von jeweils maximal acht Pins getestet. Aufgrund des Open-Drain-Ausgangs beträgt die Spannung über einer (roten) LED plus Widerstand etwa 2,6 V, wenn 5 V als Versorgungsspannung für die Ausgänge gewählt ist (JP3). Schließen Sie den Widerstand plus LED an einen der gewählten Ausgänge an, und er wird für 0,2 s eingeschaltet. Die Wiederholrate dieses Impulses hängt von der Größe der Gruppe ab: 1,6 s für die Gruppen A…C mit jeweils acht Ausgängen und nur 0,4 s für Gruppe D, die nur zwei Ausgänge besitzt. In der Zeile

```
for i in IOA:            # leds blink 0.2 s in IOx group
```

ist „IOA“ für den Test der anderen Gruppen entsprechend zu ändern. Natürlich können auch GPIO0 und GPIO1 (ID_SD und ID_SC) zu einer der Gruppen hinzugefügt werden.

Das Programm zum Testen der GPIOs als Eingang verwendet als Anzeige des Testprogramms einen I/O als Ausgang, und zwar standardmäßig GPIO3. Schließen Sie dazu einen 1,8-kΩ-Widerstand und eine LED zwischen diesem Pin 5 von K2 und GND an. Vom Quellcode wird jeweils nur ein Eingang getestet, um sicherzustellen, dass auch nur dieser als Eingang funktioniert. Ändern Sie die Zahl in der folgenden Zeile, um einen anderen GPIO als Eingang zu testen:

```
IN1 = 2  #selected GPIO to test as input
```

Das Programm gibt den ausgewählten GPIO und seinen Eingangspegel aus. Die Pull-ups der Eingänge sind aktiviert, sodass der aktuelle Eingangspin mit Masse verbunden werden muss, damit die angeschlossene LED leuchten kann. Wählen Sie am Ende der Prüfungen einen anderen als GPIO3 für den Ausgang, damit Sie auch diesen IO3 als Eingang testen können. Natürlich gibt es auch zahlreiche andere Möglichkeiten, die GPIOs zu testen. Wenn jemand also einen effizienteren und/oder schnelleren Weg entdeckt: bitte mitteilen!

Mit diesem Puffer können Sie neue Hardware unbesorgter an den Raspberry Pi 400 anschließen, da die Gefahr, dass der Pi bei Experimenten beschädigt wird, deutlich geringer ist. Dass das Risiko geringer ist, heißt nicht, dass die Pufferplatine garantiert, dass gar nichts mehr schief gehen kann. In vielen Fällen könnte es nicht schaden, der Elektronik auch etwas logisches Denken hinzuzufügen!

210320-02

WEBLINKS

[1] Raspi Buffer Board bei Elektor Labs: www.elektormagazine.de/labs/raspi-buffer-board
[2] Guy Weiler: „Puffer-Board für Raspberry Pi“, Elektor 11-12/2018 : www.elektormagazine.de/magazine/elektor-51/42024
[3] Mehr zur config.txt: www.raspberrypi.com/documentation/computers/config_txt.html
[4] Buffer-Board für den Raspberry Pi 400 im Elektor-Store: www.elektor.de/raspberry-pi-400-buffer-board
[5] Projektseite bei Elektor Labs: https://bit.ly/3GdRJ4G

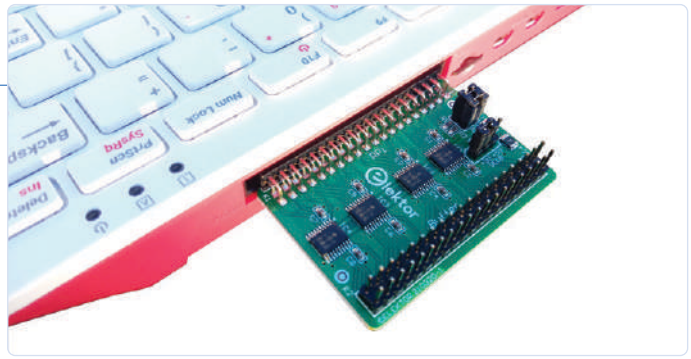


Bild 3. Die Pufferplatine, eingesteckt in den Raspberry Pi 400.

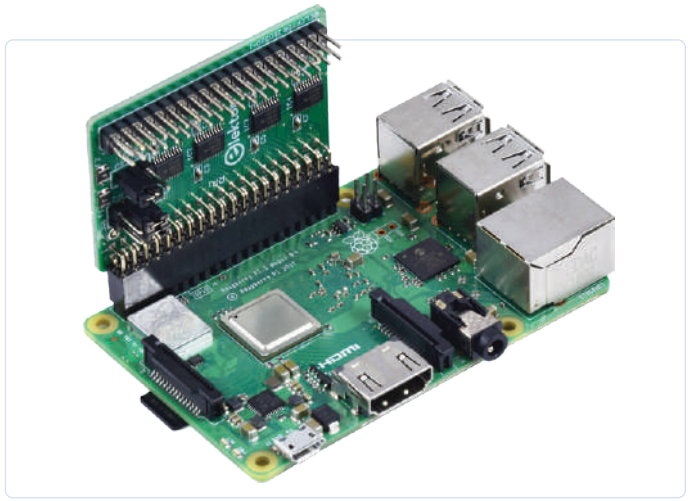


Bild 4. Die Platine kann auch mit „klassischen" Raspberry-Pi-Boards verwendet werden.

Ein Beitrag von

Entwurf, Text und Illustrationen: **Ton Giesberts**
Redaktion: **Luc Lemmens**
Übersetzung: **Rolf Gerstendorf**
Layout: **Giel Dols**

Haben Sie Fragen oder Kommentare?

Wenn Sie technische Fragen oder Kommentare zu diesem Artikel haben, können Sie sich per E-Mail an die Elektor-Redaktion wenden: redaktion@elektor.de.

PASSENDE PRODUKTE

- **Raspberry Pi 400 Puffer-Board (SKU 19884)**
  www.elektor.de/19884
- **Raspberry Pi 400 Kit – Raspberry Pi 4-basiertes PC-Kit (DE) + GRATIS GPIO-Header (QUERZ-Tastatur, SKU 19432)**
  www.elektor.de/19432
- **Raspberry Pi 400 Kit – Raspberry Pi 4-based PC Kit (EU) + FREE GPIO Header (QUERTY-Tastatur, SKU 19431)**
  www.elektor.com/19431

Was gibt‘s Neues in der Embedded-Entwicklung?

Rust und die Aktualisierung von IoT-Implementierungen

Embedded Development

Von **Stuart Cording** (Elektor)

Obwohl die Technologie für eingebettete Systeme scheinbar schnell voranschreitet, ist die Branche selbst im Vergleich dazu eher langsam. Deshalb war es fast ein kleiner Schock, als ein bekannter Hersteller von Einplatinencomputern, die Raspberry-Pi-Foundation, den RP2040-Mikrocontroller mit zwei Cortex-M0+-Kernen und ohne Onboard-Flash herausbrachte. Ein Dual-Core-Prozessor in dieser Geräteklasse ist ein absolutes Novum, aber der RP2400 ist auch schon das Spannendste auf dem Markt, dessen Fortschritt ansonsten messbar, sinnvoll und überlegt ist. Es deuten sich aber Fortschritte an, die die Entwicklung eingebetteter Systeme im kommenden Jahrzehnt verändern könnten.

Die Entwicklung eingebetteter Software ohne C ist kaum vorstellbar. Als Assembler für die Entwicklung vollständiger Anwendungen zu umständlich wurde, verdrängte C diese Sprache (außer wenn ein hochoptimierter, von Hand geschriebener Assembler die einzige Option war). Die von Dennis Ritchie [1] in den Bell Labs entwickelte Sprache ist ausreichend flexibel für komplexe Anwendungen und bietet gleichzeitig einen einfachen Zugriff auf Register. Dies ist entscheidend für einen kompakten Mikrocontroller-Code, der Registerzugriffe in Interrupt-Routinen verarbeitet. Auch Aufgaben wie die Bitmanipulation von Registern lassen sich damit leicht umsetzen. Und im Gegensatz zu Assembler-Code ist C-Code leichter zu lesen. Außerdem rangiert C in Umfragen und Marktanalysen stets unter den drei am häufigsten verwendeten Programmiersprachen (**Bild 1**) [2][3].

C ist alt

Allerdings ist C, das im Jahr 1972 entwickelt wurde, inzwischen 50 Jahre alt. Die Programmiersprache hat eine Reihe von bekannten Einschränkungen, von denen viele mit der Verwendung von Pointern zusammenhängen. Pointer erleichtern Entwicklern von eingebetteten Systemen zwar den Zugriff auf Register, sie können aber auch zu unerwünschten Speicherzugriffen außerhalb des zulässigen Bereichs führen. Außerdem führen C-Compiler im Vergleich zu moderneren Programmiersprachen vergleichsweise wenige Codeprüfungen durch. So werden ungenutzte Variablen einfach ignoriert, was ein Zeichen für einen Fehler im Code sein kann.

Um unsicheren C-Code in eingebetteten Systeme zu vermeiden, verwenden Entwickler Codierungsstandards wie MISRA C [4]. Dieser Standard entstand, als C in der Automobilindustrie als Programmiersprache für eingebettete Systeme an Bedeutung gewann. C++ löste

einige der Probleme mit Pointern durch *references* [5], die nicht geändert werden können, um auf ein anderes Objekt zu verweisen, nicht NULL sein können und bei der Erstellung initialisiert werden müssen. Trotzdem hat AUTOSAR, eine Partnerschaft von Entwicklern von Automobilsystemen, in einem mehrere hundert Seiten umfassenden Dokument Richtlinien für die Verwendung von C++ für sicherheitsrelevante Anwendungen herausgegeben [6]. Obwohl also die Beherrschung dieser etablierten Sprachen für Entwickler von eingebetteten Systemen unabdingbar ist, weist jede dieser Sprachen so viel Schwächen auf, dass Richtlinien erforderlich sind, um häufige Programmierfehler zu vermeiden.

Einführung von Rust

Rust hat sich als potenzieller Herausforderer herauskristallisiert und wirbt mit der Eignung zur Entwicklung sicherer Systeme. Rust begann 2006 als privates Projekt von Graydon Hoare und wurde 2010 zu einem von seinem Arbeitgeber Mozilla Research geförderten Projekt. Als 2021 Umstrukturierungen des Unternehmens Auswirkungen auf das Rust-Entwicklungsteam hatten, wurde die Rust Foundation [7] gegründet.
Das Besondere an Rust ist, dass viele Probleme schon bei der Kompilierung erkannt und markiert werden, oft auch solche, die C/C++-Compiler ignorieren würden. Es wurde auch ein *Ownership*-System für Variablendeklarationen implementiert, das einen *Borrow-Checker* verwendet, um eine missbräuchliche Anwendung von Variablen bereits während der Kompilierung auszuschließen. Außerdem muss der Lese- und Schreibzugriff auf Variablen, die per Referenz übergeben werden, explizit deklariert werden. Die Syntax von Rust ähnelt weitgehend C und C++, wobei geschweifte Klammern um Funktionen und die bekannten Steuer-Keywords verwendet werden.
Da der Schwerpunkt auf sicherem Code liegt, wurde Rust besonders für die Bare-Metal-Community zur Entwicklung eingebetteter Software interessant. Aufgrund der Natur der eingebetteten Programmierung kann die vom Compiler eingesetzte statische Prüfung jedoch bei der Implementierung einiger Codetypen zu Problemen führen. Einige Codeabschnitte können als *unsafe* markiert werden, um zum Beispiel die De-Referenzierung von Pointern zu ermöglichen. Durch die explizite Definition von Codeabschnitten als *unsafe* kann die Umgehung der Rust-Regeln verdeutlicht werden.

Rust auf den Prüfstand gestellt

Um ein Gefühl für Rust zu bekommen, installiert man Rust am besten auf einem Raspberry Pi. Die Installation ist einfach, wenn man den Hinweisen auf der Website *rustup.rs* [8] folgt. Geben Sie auf der Kommandozeile einfach den folgenden Befehl ein und folgen Sie den Anweisungen:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.
   rs | sh
```

Im Gegensatz zu C/C++, das Binärdateien erzeugt, wird die Ausgabe von Rust als *Crate* bezeichnet. Der Paketmanager Cargo sorgt für eine einfache Befehlszeilenkompilierung. Er enthält die Daten, die für die Erzeugung des *Crate* benötigt werden und ermöglicht es dem Entwickler, die für die Erstellung des *Crate* erforderlichen Pakete zu definieren. Durch den Aufruf von Cargo auf der Kommandozeile wird ein neues Rust-Projekt wie folgt erzeugt:

```
cargo new rust_test_project
```

Wenn Sie das Verzeichnis des neuen Projekts öffnen, sehen Sie eine Datei namens *Cargo.toml*. Diese Datei muss unter Umständen geändert werden. Für eine einfache Raspberry-Pi-Anwendung, die eine an einen

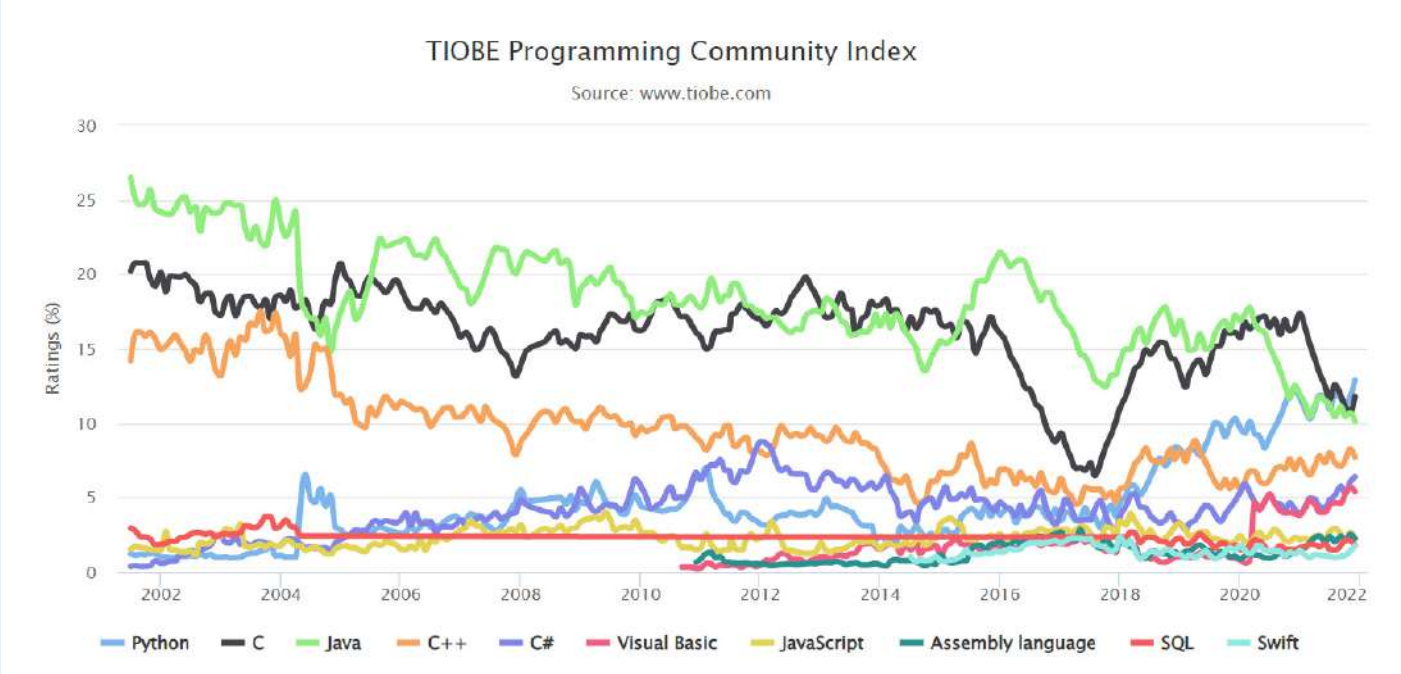


Bild 1. C ist als Programmiersprache nach wie vor sehr beliebt und belegt in Umfragen und Marktanalysen regelmäßig einen der ersten drei Plätze. (Quelle: www.tiobe.com)

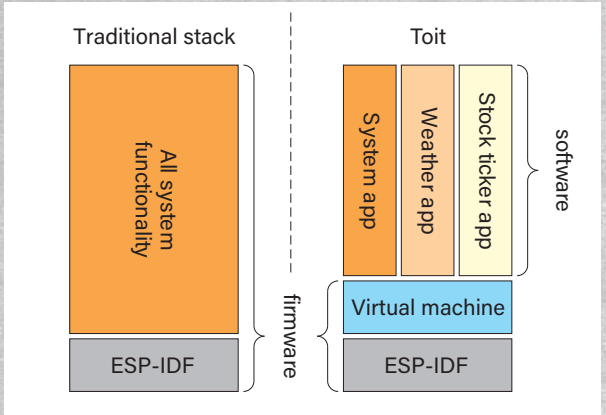


Bild 2. Toit führt IoT-Code als Apps auf einer virtuellen Maschine aus, die auf einem ESP32 läuft. (Quelle: Toit)

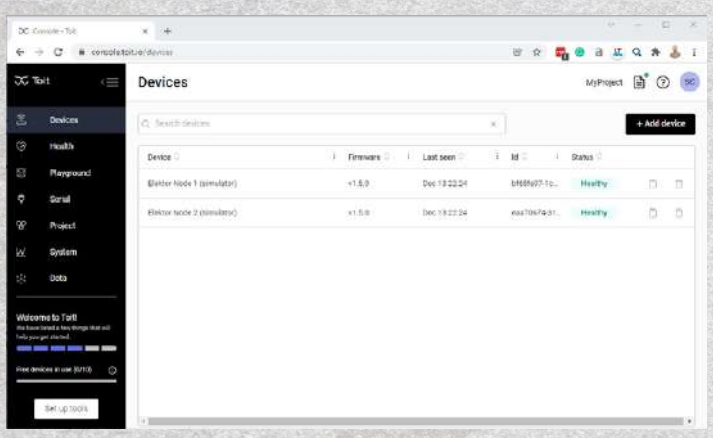


Bild 3. Die Toit-Konsole in einem Browser, hier mit zwei simulierten ESP32-Knoten.

GPIO-Pin angeschlossene LED blinken lässt, muss eine geeignete Crate-Abhängigkeit für den Zugriff auf die GPIO-Pins definiert werden. Crates werden auf der Crate-Plattform der Rust-Community crates.io [9] zur Verfügung gestellt. Ein geeigneter Crate ist *rppal*, der über die Suchfunktion gefunden werden kann. Die Plattform gibt die Versionsnummer an und stellt eine Dokumentation und Beispielcode zur Verfügung. Der Name des Crate und die Versionsnummer werden dann als *dependencies* in *Cargo.toml* hinzugefügt (**Listing 1**).

Listing 1. Hinzufügen der rppal-Abhängigkeit zu Cargo.toml in einem Rust-Projekt.

```
[package]
name = "rust_test_project"
version = "0.1.0"
edition = "2021"
[dependencies]
rppal = "0.13.1"
```

Im *src*-Verzeichnis findet der Entwickler ein einfaches *Hello-World*-Beispielprojekt in der Rust-Quellcodedatei *main.rs*. Wenn man diesen Code durch **Listing 2** ersetzt, kann man eine LED an GPIO 23 (Pin 16 des Raspberry-Pi-Headers) zehnmal blinken lassen. Die Kompilierung erfolgt über die Kommandozeile mit `cargo build`, das Crate wird mit `cargo run` ausgeführt.

Um Rust auf Mikrocontrollern verwenden zu können, ist eine LLVM-Toolchain erforderlich. Wenn diese vorhanden ist, können Sie mit Hilfe eines der verschiedenen Online-Tutorials loslegen. Ein Beispiel für den BBC micro:bit [10] zeigt, dass das Schreiben von Code, hat man erst einmal die Rust-Syntax im Griff, dem von C/C++ sehr ähnlich ist (**Listing 3** [11]). Ein weiteres Beispiel für den STM32 [12] ist ebenfalls enthalten.

Gehört Rust die Zukunft?

Was steht also der Einführung von Rust für Embedded-Anwendungen im Wege? Wenn Sie nach einer robusten Programmiersprache für den

Listing 2. Blinken einer LED in Rust.

Das Blinken einer LED in Rust ähnelt dem in C geschriebenen Code.
Dieses Beispiel basiert auf dem Code, der im rppal-Crate enthalten ist.

```
use std::error::Error;
use std::thread;
use std::time::Duration;
use rppal::gpio::Gpio;
use rppal::system::DeviceInfo;
// Gpio uses BCM pin numbering. BCM GPIO 23 is tied to physical pin 16.
const GPIO_LED: u8 = 23;
fn main() -> Result<(), Box<dyn Error>> {
    let mut n = 1;
    println!("Blinking an LED on a {}.", DeviceInfo::new()?.model());
    let mut pin = Gpio::new()?.get(GPIO_LED)?.into_output();
    while n < 11 {
        // Blink the LED by setting the pin's logic level high for 500 ms.
        pin.set_high();
        thread::sleep(Duration::from_millis(500));
        pin.set_low();
        thread::sleep(Duration::from_millis(500));
        n += 1;
    }
    Ok(())
}
```

Listing 3. Vereinfachter Codeschnipsel, der den Zustand eines Eingangspins des BBC micro:bit mit Rust überprüft.

```
#![no_std]
#![no_main]

extern crate panic_abort;
extern crate cortex_m_rt as rt;
extern crate microbit;

use rt::entry;
use microbit::hal::prelude::*;

#[entry]
fn main() -> ! {
     if let Some(p) = microbit::Peripherals::take() {
        // Split GPIO
        let mut gpio = p.GPIO.split();

        // Configure button GPIO as input
        let button_a = gpio.pin17.into_floating_input();

        // loop variablew
        let mut state_a_low = false;

        loop {
             // Get button state
             let button_a_low = button_a.is_low();

            if button_a_low && !state_a_low {
                 // Output message
             }

            if !button_a_low && state_a_low {
                 // Output message
             }

            // Store button states
             state_a_low = button_a_low;
         }
     }
     panic!("End");
 }
```

Einsatz in sicherheitskritischen Systemen suchen, ist doch Ada [13] eigentlich eine gute Wahl. Doch obwohl Ada bereits 40 Jahre alt ist, hat sie C nicht verdrängen können, obwohl sie speziell für den Einsatz in eingebetteten Echtzeitsystemen entwickelt wurde. Ein weiterer Aspekt ist der jahrelange Erfolg von C, mit unzähligen Entwicklern, Werkzeugen und Code-Bibliotheken.

Der Crate-Paketmanager könnte jedoch dazu beitragen, die Verbreitung von Rust zu beschleunigen. Den Überblick über die in C/C++ geschriebenen Peripherie-Bibliotheken der Halbleiterhersteller zu behalten, kann schwierig und undurchsichtig sein, so dass die explizite Definition der Version der in *Cargo.toml* verwendeten Crates als bedeutende Verbesserung angesehen werden könnte. Es könnte auch das Leben der MCU-Hersteller erleichtern, die ihre riesigen Produktportfolios unterstützen wollen. Ada hat im Jahr 2020 bereits mit der Veröffentlichung des Paketmanagers *Alire* reagiert [14] und unterstützt genau wie Rust den BBC micro:bit. Damit können Sie beide Sprachen auf der gleichen MCU vergleichen [15].

IoT-Geräte auf dem neuesten Stand halten

Da immer mehr eingebettete Geräte mit Netzwerken verbunden sind, besteht die größte Herausforderung darin, sie mit der neuesten Version der Firmware auf dem neuesten Stand zu halten. Traditionell müssen Firmware-Updates über Funk heruntergeladen werden, wobei die Binärdaten über einen On-Chip-Bootloader im geschützten Bereich des Geräts in den Flash-Speicher programmiert werden. Dabei besteht das Risiko, dass die neue Codeversion nicht korrekt installiert wird, so dass das Produkt nicht mehr funktioniert und unbrauchbar wird. Alternativ kann der Code zwar funktionieren, aber ein Softwarefehler im neuen Code könnte verhindern, dass künftige Updates eingespielt werden, so dass die Geräte in der Praxis anfällig bleiben. Auf diese Weise wird das Gerät vielleicht nur aufgrund eines Fehlers in einer einzigen Codezeile unbrauchbar, obwohl der Großteil der Anwendung korrekt funktioniert.

Virtuelle Maschinen sind seit Jahren eine Standardmethode, um Anwendungen oder Betriebssysteme auf Servern bereitzustellen. Die virtuelle Maschine gaukelt einem Betriebssystem vor, es würde auf dedizierter Hardware ausgeführt. Sollte das Betriebssystem schwerwiegend versagen, ist nur die betroffene virtuelle Maschine davon betroffen, nicht aber die übrigen auf dem Server laufenden Systeme. Desktop-Benutzer kennen Virtualisierungssoftware wie VirtualBox, VMware und Parallels, mit der sie Software oder alternative Betriebssysteme testen können, ohne ihren primären Rechner einem Risiko auszusetzen.

Aktualisierung der IoT-Knoten-Firmware: Der Weg zu Toit

Das Team von Toit, einem dänischen Unternehmen, das von ehemaligen Google-Ingenieuren gegründet wurde, fragte sich, warum die Virtualisierung nicht auch auf Mikrocontrollern eingesetzt wird, auf denen IoT-Anwendungen laufen. Schließlich müssen diese regelmäßig aktualisiert werden, um Fehler zu beheben und möglicherweise um Verbesserungen in der Cloud zu unterstützen.

Für die ersten Schritte haben sie sich für den ESP32 von Espressif entschieden, genauer gesagt, den ESP32-WROOM-32E mit einem 32-bit-Dual-Core Xtensa LX6 Mikroprozessor, 520 KB SRAM und 4 MB Flash. Für diese Plattform gibt es bereits das ESP-IDF (Espressif IoT Development Framework), so dass sie für den Einsatz als IoT-Knoten gut vorbereitet ist. Toit hat dann eine virtuelle Maschine (**Bild 2**) entwickelt, auf der Anwendungen sicher ausgeführt werden können. Die Anwendungen sind in Toit geschrieben, einer objektorientierten und sicheren Hochsprache.

Testen von Toit mit simulierten ESP32-Knoten

Die Verwaltung und Bereitstellung von Anwendungen auf einer Ansammlung von IoT-Knoten erfolgt über die Toit-Konsole in Verbindung mit Visual Studio Code von Microsoft. Für diejenigen, die nur daran interessiert sind, die Plattform zu testen, ermöglicht die Toit Console die Verwendung von simulierten ESP32-Geräten. Nach der Installation der Toit-Erweiterung für Visual Studio Code können Anwendungen geschrieben und lokal simuliert oder an Geräte verteilt werden, die mit der Konsole verbunden sind.

Toit-Apps werden von einer YAML-Datei begleitet. In dieser Markup-Language-Datei werden die Namen der Toit-App und der Quellcodedatei deklariert und Trigger definiert, die die App nach dem

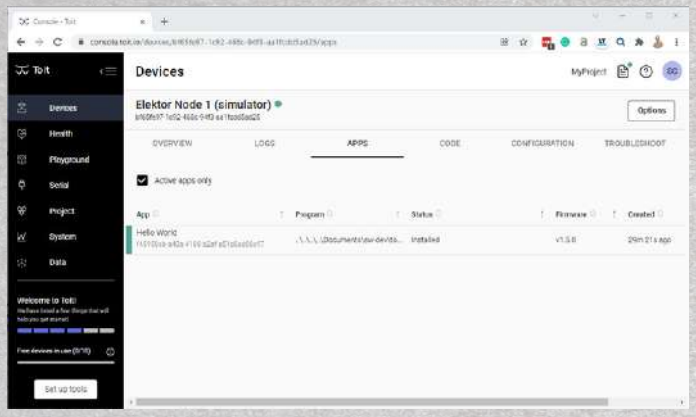


Bild 4. Die Hello-World-App wurde erfolgreich auf einem simulierten Knoten installiert

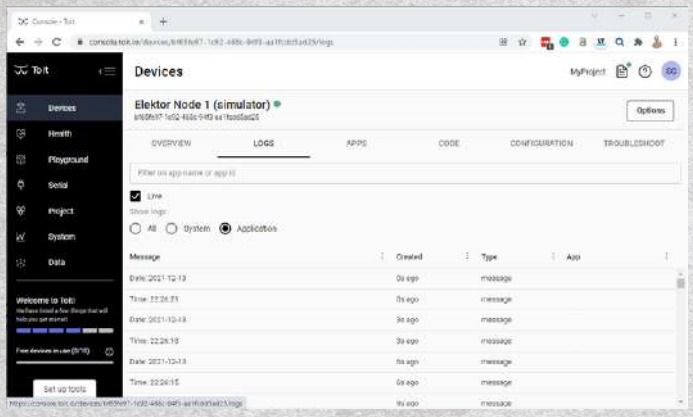


Bild 5. Die LOGS-Ausgabe zeigt die Uhrzeit und das Datum, die von der Toit-App alle drei Sekunden ausgegeben werden, wie in der YAML-Datei definiert.

Booten oder nach bestimmten Zeitintervallen ausführen.
Die Bereitstellung von Apps oder Updates erfordert nicht, dass das ESP32-Zielgerät deaktiviert oder die Stromversorgung unterbrochen werden muss. Stattdessen aktualisiert die virtuelle Maschine die App an Ort und Stelle und triggert sie gemäß den in der YAML-Datei angegebenen Einstellungen. Wenn die App in irgendeiner Weise versagt, zum Beispiel durch einen Speicherüberlauf, werden die übrigen Apps ohne Probleme weiterhin ausgeführt [16]. Solche Fehler können über die Konsole diagnostiziert werden, indem man sich das Protokoll ansieht.
Listing 4 zeigt eine einfache *Hello-World*-App, die Uhrzeit und Datum ausgibt, **Listing 5** die zugehörige YAML-Datei. In **Bild 3** sind zwei simulierte Knoten in der Konsole zu sehen, während **Bild 4** die erfolgreich installierte *Hello-World*-App zeigt. **Bild 5** zeigt die Ausgabe von Datum und Uhrzeit in Drei-Sekunden-Intervallen, wie sie mit dem Trigger `on_interval: "3s"` in der YAML-Datei festgelegt wurden.
Der Projektcode wurde in Visual Studio Code erstellt und mit der Toit-Erweiterung auf dem ausgewählten Knoten bereitgestellt, der mit dem Konsolen-Konto des Users verbunden ist (**Bild 6**).
Auf den ersten Blick mag der Toit-Ansatz ein wenig restriktiv erscheinen. Die Virtualisierungsumgebung erlaubt nur die Steuerung der GPIOs, der seriellen Schnittstelle (UART), SPI oder I²C. Da jedoch viele IoT-Knoten Sensordaten sammeln und diese an die Cloud melden, dürfte dies für die meisten Anwendungen ausreichend Flexibilität bieten. Toit betreibt auch einen eigenen Paketmanager (Registry) [17], der Treiber für eine Reihe von Sensoren, Eingabegeräten, LCDs und anderen Dienstprogrammen zur Verfügung stellt. Die Umgebung unterstützt auch den Stromsparmodus des ESP32, der es angeblich ermöglicht, das Gerät jahrelang mit zwei AA-Batterien zu betreiben [18]. Wenn eine App-Aktualisierung erfolgt, während sich der IoT-Knoten im Tiefschlafmodus befindet, stellt die Konsole sie dann bereit, wenn der Knoten das nächste Mal aufwacht.

Rust und Toit - die Zukunft von Embedded?

Zwei Dinge stechen bei Rust und Toit hervor. Beide sind einfach in Betrieb zu nehmen und beide verwenden Paketmanager, um die Low-Level-Treiber zu verwalten. Dadurch können sich die Entwickler auf ihre eigentliche Aufgabe konzentrieren, nämlich das Erstellen von Anwendungen. Da die Anwendungen immer komplexer werden, ist eine solche Wiederverwendung von Code unerlässlich, um die Entwicklungszeit zu verkürzen und Produkte schneller auf den Markt zu bringen.
Rust steht vor einer gewaltigen Aufgabe: Da C und C++ in der Welt der Embedded-Entwicklung so fest verankert sind, dürfte es schwierig sein, eine Lücke zu finden, in der Rust einen so gewichtigen Vorteil bietet, dass er Entwickler anlockt. Und da sich Ada bereits als die

Listing 4. Eine einfache Toit-Anwendung, die formatierte Zeit- und Datumsstrings in der Log-Konsole ausgibt.

```
main:
  time := Time.now.local
  print "Time: $(%02d time.h):$(%02d time.m):$(%02d time.s)"
  print "Date: $(%04d time.year)-$(%02d time.month)-$(%02d time.day)"
```

Listing 5. Die zugehörige YAML-Datei, die der Toit-Konsole mitteilt, wie die Anwendung bereitgestellt werden soll.

```
name: Hello World
entrypoint: hello_world.toit
triggers:
  on_boot: true
  on_install: true
  on_interval: "3s"
```

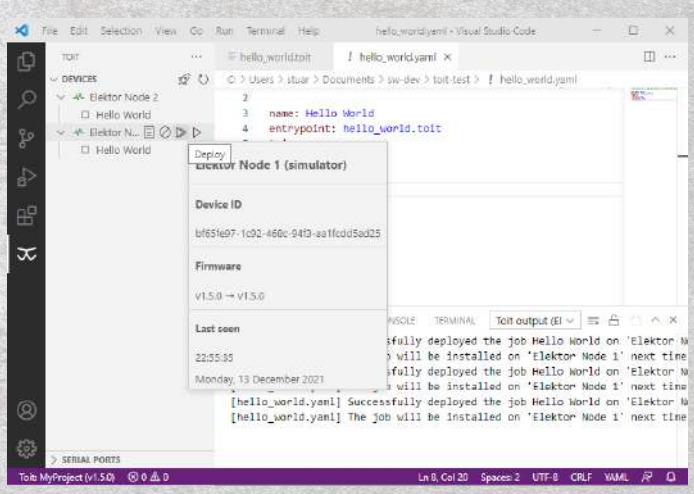


Bild 6. Mit der Toit-Erweiterung in Visual Studio Code können Änderungen an Anwendungen auf IoT-Knoten bereitgestellt werden, ohne dass die Geräte vor Ort aus dem Netz entnommen werden müssen.

Sprache für sicherheitskritische Systeme etabliert hat, verschwinden die Vorteile von Rust.

Toit hingegen löst ein großes Problem: Es hält entfernte IoT-Knoten auf dem neuesten Stand, stellt neue Funktionen zur Verfügung und das alles, ohne die Geräte aus dem System zu entfernen. Einige Entwickler dürften über nur 4 MB Flash und die Tatsache, dass derzeit nur der ESP32 unterstützt wird, die Stirn runzeln, sollte jedoch eine Nachfrage nach anderen MCUs entstehen, dürfte es keinen technischen Grund zu geben, warum andere Plattformen in Zukunft nicht unterstützt werden sollten.

210652-02

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Dann wenden Sie sich bitte an den Autor unter stuart.cording@elektor.com oder an die Elektor-Redaktion unter redaktion@elektor.de.

Ein Beitrag von

Text und Illustrationen: **Stuart Cording**
Redaktion: **Jens Nickel, CJ Abate**
Übersetzung: **Rolf Gerstendorf**
Layout: **Harmen Heida**

PASSENDE PRODUKTE

- Buch: D. Ibrahim, BBC micro:bit (Elektor, 2016, SKU 17972) www.elektor.de/bbc-micro-bit-book
- Joy-IT BBC micro:bit Go Set (SKU 18930) www.elektor.de/18930

WEBLINKS

[1] Dennis Ritchie, Computer History Museum: https://bit.ly/3oOXG1A
[2] P. Jansen, „TIOBE Index for Dezember 2021", TIOBE Software BV, Dezember 2021: https://bit.ly/3EXx8Ri
[3] S. Cass, „Top Programming Languages 2021", IEEE Spectrum, 2021: https://bit.ly/3oPJq8z
[4] Webseite MISRA: https://bit.ly/3s1Mv7D
[5] „C++ References", Tutorials Point: https://bit.ly/31Y6k53
[6] „Guidelines for the use of the C++14 language in critical and safety-related systems" AUTOSAR, Oktober 2018: https://bit.ly/3dO3zWl
[7] Webseite Rust Foundation: https://bit.ly/3DNgO45
[8] Webseite rustup: https://bit.ly/3EUTiDR
[9] Webseite Rust Crate Registry: https://bit.ly/3dLKmor
[10] droogmic, „MicroRust", April 2020: https://bit.ly/3EUWFdM
[11] droogmic, „MicroRust: Buttons", April 2020: https://bit.ly/3DWes30
[12] bors and NitinSaxenait, „Discovery", Dezember 2021: http://bit.ly/3GERDT7
[13] Webseite Get Ada Now: https://bit.ly/3GHyikw
[14] F. Chouteau, „First beta release of Alire, the package manager for Ada/SPARK", AdaCore, Oktober 2020: https://bit.ly/3EUXOSC
[15] F. Chouteau, „Microbit_examples", Dezember 2021: https://bit.ly/3mnH7bx
[16] „Watch us turn an ESP32 into a full computer!", Toit, März 2021: https://bit.ly/3IM0WT8
[17] Webseite Toit Package Registry: https://bit.ly/3yokpo7
[18] Webseite Toit-Docs, Voraussetzungen: https://bit.ly/3oNSECq

Die Miniaturisierung **elektronischer Bauteile und industrieller Sensoren**

Von Richard Woodward, Distrelec

Die elektronischen Komponenten werden immer kleiner. Dieser Trend kann auf mehrere zusammenhängende Faktoren zurückgeführt werden. Einerseits stehen die Hersteller von Endprodukten unter dem Druck, ihre Produkte zu verkleinern und gleichzeitig deren Leistungsfähigkeit und Funktionalität zu erhöhen. Andererseits müssen auch die Komponentenhersteller ihre Technologien ständig erneuern, damit die Komponenten in die kleineren Endprodukte passen. Das ist nur durch die technologische Forschung und Entwicklung möglich. Ein herausragendes Beispiel dafür sind Halbleiter, denn sie können mehrere Milliarden Transistoren auf immer kleinerer Fläche unterbringen. Doch sie sind nicht die einzigen elektronischen Bauteile, die kleiner werden müssen. Heute enthalten die meisten elektronischen Endprodukte nur noch weniger als fünf Halbleiter, obwohl ihre Leistungsfähigkeit enorm gestiegen ist. Es werden jedoch Hunderte unterstützender und ebenso wichtiger Komponenten benötigt, wie z. B. passive Bauelemente (Induktivitäten, Kondensatoren und Widerstände).

Die Anforderungen an die Miniaturisierung von Bauteilen beschränken sich jedoch nicht nur auf elektronische Komponenten. In der Industrie ist der Platz in der Fabrikhalle sehr knapp bemessen, und der verfügbare Platz für die einzelnen Produktionsanlagen, Sensoren und Aktoren ist ebenfalls begrenzt. Folglich muss die nächste Generation jedes Produktionsmittels kleiner sein und mehr Funktionen bieten.

Die Miniaturisierung von Komponenten wird immer wichtiger

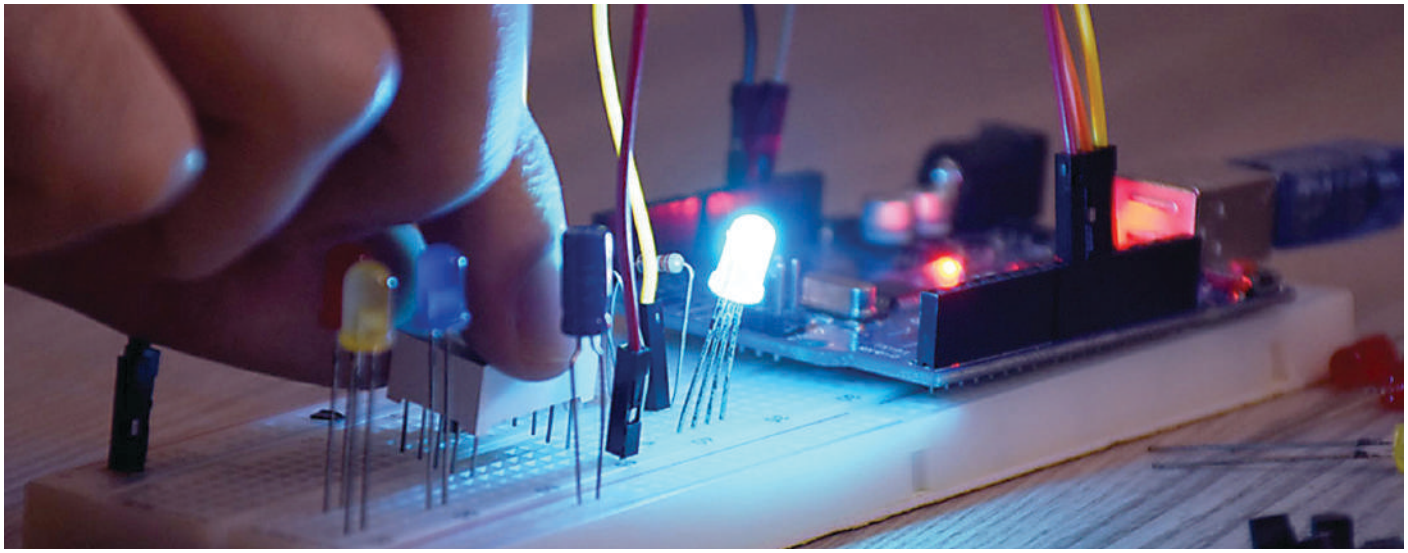
Die Elektronikbranche befindet sich ständig im Wandel. Seit der Erfindung des Transistors in den späten 1940er Jahren lag der Fokus dabei stets auf der Miniaturisierung. Ein Jahrzehnt später wurde der erste integrierte Schaltkreis (IC) mit vier Transistoren von Robert Noyce bei Fairchild Semiconductor entwickelt. Heutzutage finden sich in den modernsten Prozessoren mehrere Milliarden Transistoren. Das bedeutet, dass sich das Verfahren zur Herstellung von Halbleitern auf immer kleinerer Fläche deutlich verbessert hat. Die Fortschritte bei der Entwicklung und Herstellung von Komponenten sind jedoch auch der Elektronikindustrie im Allgemeinen zugute gekommen.

Die Auswirkungen des technologischen Fortschritts bei elektronischen Bauteilen

Der erste Transistor-Prototyp war im Vergleich zu den heutigen Geräten sehr groß. Allerdings war er wesentlich kleiner als die damals üblichen Thermionischen Ventile. Der Transistor war nicht nur kleiner, sondern auch die Anordnung der Versorgungsspannung war weniger komplex als bei Röhren und es war kein Heizelement erforderlich. Schon früh erkannten die Ingenieure das Potenzial, das die Integration von Transistoren in einen IC bieten kann. Dies war der Startpunkt für die Entwicklung immer kleinerer und leistungsfähigerer ICs. Gordon Moore, der Mitbegründer von Intel, prognostizierte mit seinem „Mooreschen Gesetz", dass sich die „Anzahl der Transistoren in einem IC alle zwei Jahre verdoppeln" würde.

Die Forschung und Entwicklung im Bereich des Designs und der Entwicklung von Halbleiter-ICs ist auch heute noch führend in der Elektronikindustrie. Die Verkleinerung elektronischer Komponenten stellt die Hersteller von automatisierten Produktionsanlagen vor die Herausforderung, sich auf kleinere Größen einzustellen. Die physischen Dimensionen, um die es jetzt geht, sind erstaunlich. Ein moderner Halbleiterprozessknoten beispielsweise hat derzeit eine Größe von 5 nm. Dieses Maß bezieht sich nicht auf die tatsächliche Transistorgröße, sondern wird von Halbleiterherstellern zur Angabe der Transistordichte verwendet. Viele Smartphones verwenden ICs, die auf dem 5-nm-Verfahren basieren, und die Rechenleistung der 30 Milliarden Transistoren ermöglicht den Betrieb des Telefons und all unserer beliebten Apps.

Transistoren und Halbleiter-ICs werden nach den Standardspezifikationen für die Oberflächenmontage (SMT) verpackt, die von der JEDEC Solid State Technology Association verwaltet werden. Das Gleiche gilt für oberflächenmontierte passive Komponenten wie Kondensatoren, Widerstände und Induktivitäten. Da Halbleiter immer kleiner und leistungsfähiger werden, müssen auch die passiven Komponenten an die Größe der Halbleiter angepasst werden. Deshalb findet man heute in jedem beliebigen Embedded-System nur ein paar komplexe ICs. Um die ICs herum befinden sich jedoch viele



Hunderte von passiven Bauteilen, die für ihren Betrieb von grundlegender Bedeutung sind.

Ein Beispiel für ein Transistorgehäuse ist das Format SOT23-3 (**S**mall **O**utline **T**ransistor). Er wird in der Regel für Kleinsignal-Allzwecktransistoren verwendet, hat drei Anschlüsse und misst 3 mm x 1,75 mm x 1,3 mm. Einige ICs verwenden auch das SOT-23-Gehäuse, wobei das „-x" die Anzahl der verwendeten Pins angibt. Ein SOT23-6 bezeichnet zum Beispiel ein IC mit sechs Pins.

Für integrierte Schaltungen gibt es eine Vielzahl unterschiedlicher Gehäuseformen, einige mit, andere ohne Verbleiung. In jedem Fall sind sie alle oberflächenmontiert. Einige Sensoren, wie z. B. mikroelektromechanische Systeme (MEMS), sind in gängigen Halbleitergehäusen untergebracht.

Beispiele hierfür sind:

- SSOP (Shrink Small Outline Package) – bedrahtet – mit einem Pinabstand von 0,635 mm
- TSSOP (Thin Shrink Small Outline Package) – bedrahtet – mit einem Pinabstand von 0,65 mm
- QFN-Gehäuse (Quad Flat Non-leaded) – dieses Gehäuse ist in einer Vielzahl verschiedener Elektrodenanschlüsse (Pins) erhältlich - von 14 bis 100 und im Rastermaß von 0,5 mm bis 1,65 mm

Die meisten oberflächenmontierten passiven „Chip"-Komponenten verwenden die EIA-Codes zur Angabe der Komponentengröße. Beliebte Beispielgrößen sind:

- 0805 mit 2,0 mm x 1,30 mm (0,08 Zoll x 0,05 Zoll)
- 0603 mit den Abmessungen 1,5 mm x 0,80 mm (0,06 Zoll x 0,03 Zoll)
- 0402 mit den Abmessungen 1,0 mm x 0,50 mm (0,04 Zoll x 0,02 Zoll)

Die Miniaturisierung von Komponenten in der Industrie

Die Miniaturisierung von Komponenten ist besonders für Hersteller von Endprodukten wichtig. Denn vor allem in der Industrie hat der Automatisierungsgrad in den letzten zehn Jahren deutlich zugenommen. Durch die Industrie 4.0 und das industrielle Internet der Dinge (IIoT) konnte die Effizienz deutlich gesteigert werden. Das bedeutet auch, dass immer mehr elektronikbasierte Sensoren, Steuergeräte und maschinelle Lernsystemen verwendet werden. Gleichzeitig wird der verfügbare Platz in den Fabriken immer knapper, was die Nachfrage nach kompakten, energieeffizienten und hochmodernen Komponententechnologien erhöht.

Daher ist das Streben nach Miniaturisierung von Bauteilen nicht auf elektronische Komponenten beschränkt. Auch die Hersteller von Automatisierungsgeräten und -baugruppen sind von der Miniaturisierung betroffen. Denn Fortschritte im 3D-Druck, in der Materialforschung und ein besseres Verständnis der Finite-Elemente-Analyse in der mechanischen Konstruktion tragen zur Gewichts-, Größen- und Kostenreduzierung bei.

Die Vorteile der Miniaturisierung von Komponenten

Durch die Miniaturisierung können Entwickler und Hersteller fortschrittlichere Funktionen in ihre Endprodukte einbauen und gleichzeitig deren Platzbedarf verringern. Sowohl bei einem IIoT-Sensor als auch bei einer industriellen speicherprogrammierbaren Steuerung kann an Platz innerhalb des Produktgehäuses gespart werden. Somit können durch die Miniaturisierung elegantere und platzsparendere Produkte entworfen werden. Darüber hinaus eröffnet sie die Möglichkeit, die Merkmale und die Funktionalität neuer Produkte zu erhöhen, ohne deren Abmessungen zu vergrößern.

Weitere Informationen erhalten Sie unter *www.distrelec.com*.

220286-01

Über Distrelec

Die Distrelec Gruppe ist ein führender Online-Distributor für elektronische und technische Komponenten im B2B-Bereich mit rund 500 Mitarbeitern. Neben den Hauptabsatzmärkten Deutschland, Schweiz und Schweden, hat das Unternehmen eine starke Marktposition in insgesamt 17 europäischen Ländern. Distrelec zielt auf Kunden in den Bereichen Instandhaltung, Kleinserie, Prototyping sowie Forschung und Entwicklung.

Neue Wege für **Industrie** und **Automobilbranche** mit **5G**

Von Mark Patrick (Mouser Electronics)

5G ist mehr als nur ein Fortschritt für den Mobilfunk, wo dank höherer Download-Geschwindigkeiten das Surfen auf einem Smartphone spürbar verbessert wird. Die eigentlichen Auswirkungen von 5G werden aber eher von Applikationen ausgehen, die erst noch entwickelt werden müssen. Werfen wir dazu einen Blick auf die Industrie- und Automobilbranche und darauf, wie Sie hier mit 5G neue Wege gehen können.

Warum 5G im Bereich der industriellen Fertigung?

Die kabelgebundene Architektur vieler intelligenter Fabriken stellt ein Hemmnis für die Weiterentwicklung dar. Die verschiedenen Sensoren, Aktoren und Steuerungen in automatisierten Anlagen sind über durchaus bewährte Netzwerktechnologien wie Industrial Ethernet, Profinet und CANbus miteinander verbunden, allerdings sind selbst kleinste Änderungen an den Produktionsanlagen wegen der fest definierten Konnektivität zeitaufwändig und kostspielig. Die bisherigen Generationen drahtloser Netzwerke, selbst wenn sie auf der 4G/LTE-Technologie basieren, sind nicht in der Lage, die für die Autonomie erforderliche Echtzeit-Reaktionsfähigkeit und niedrige Latenzen zu bieten. Außerdem sind Fertigungsbereiche eine schwierige Betriebsumgebung, in der die Kommunikation mit den bisherigen Funktechnologien durch ein hohes Maß an elektrischem Rauschen und Störungen in ihrer Leistung beeinträchtigt ist. Die erweiterten Netzwerkfähigkeiten von 5G können einige dieser Probleme lösen und die Systemeffizienz und -flexibilität erhöhen.

Eine der Schlüsselfunktionen jeder automatisierten Fabrik ist das Monitoring. 5G ermöglicht mMTC (Massive Machine-Type Communications) und wird damit den Anforderungen umfangreicher drahtloser Sensornetzwerke (WSN) gerecht. 5G ist auch energieeffizienter als seine Vorgänger, was entscheidend für längere Akkulaufzeiten der angeschlossenen Geräte ist und somit zur Minimierung des Wartungsaufwands beiträgt.

Die für Bewegungssteuerung und Industrierobotik erforderliche Präzision und Echtzeit-Empfindlichkeit wurde bisher über Time-Sensitive Networking (TSN) mit kabelgebundenem Industrial Ethernet als bevorzugter Netzwerktechnologie umgesetzt. Mit seiner ultrazuverlässigen Kommunikation mit geringer Latenz (URLLC) ist 5G eine effiziente drahtlose Alternative und ermöglicht zudem den Einsatz von Cloud-Robotik.

Drei verwandte Technologien, die zunehmend in Fertigungsumgebungen eingesetzt werden, sind Virtual Reality (VR), Augmented Reality (AR) und Künstliche Intelligenz (KI). Dank hoher Geschwindigkeit und URLLC ermöglicht 5G die Edge-Verarbeitung, sodass energieintensive Berechnungen an die Cloud übergeben werden können, was den Einsatz von weniger komplexen und somit kostengünstigeren Geräten in der Fertigung ermöglicht.

Chancen und Herausforderungen bei der Implementierung von 5G

Um bisherige Investitionen in kabelgebundene und drahtlose Netzwerktechnologien zu schützen, müssen sich 5G-Projekte nahtlos in die bestehende Infrastruktur integrieren. Eine der größten Herausforderungen besteht in der Komplettabdeckung in Gebäuden, die für Mobilfunknetzbetreiber (MNOs) bisher keine Priorität hatte. Neue Entwicklungen bei Open-RAN-Technologien senken die Betriebskosten von 5G-Funkzugangsnetzen (5G RAN), sodass private 5G-Netze eine realistische Option darstellen. Für Unternehmen, die diese Non-Public Networks (NPN) bevorzugen, stellen Regulierungsbehörden weltweit ein dediziertes, kosteneffizientes Frequenzspektrum zur Verfügung. Darüber hinaus können private 5G-Netze je nach den betrieblichen Anforderungen des Unternehmens entweder vollständig vom öffentlichen Netz isoliert oder mit diesem gemeinsam genutzt werden.

5G und die Ära der vernetzten Fahrzeuge

Der Automobilsektor wird den Prognosen zufolge ebenfalls eine Vorreiterrolle bei der Einführung von 5G spielen, auch wenn es noch einige Jahre dauern könnte, bis der

Automatisierungsgrad Level 5 (Vollautomatisierung) kommerzielle Relevanz erreicht. Es ist jedoch wahrscheinlich, dass das nächste Auto, das Sie kaufen, internetfähig ist, um Telematik, C-V2X (Cellular Vehicle-to-Everything) und Infotainment zu verwalten.

Das vernetzte Auto von heute kann bis zu 4 Terabyte an Daten pro Tag generieren, was etwa 500 Filmen entspricht. Neueste Entwicklungen bei C-V2X-Kommunikationstechnologien nutzen diese Daten bereits auf vielfältige Weise. So werden Daten aus den Motormanagementsystemen zur vorausschauenden Wartung an entfernte Servicezentren gesendet. Lokale Informationen zur Verkehrssituation und zum Wetter können zudem in öffentliche Sicherheitssysteme einfließen. Sogar das Fahrerverhalten und der Kilometerstand des Fahrzeugs lassen sich in Datenbanken für nutzungsbasierte Versicherungstarife nutzen.

In den letzten 5 Jahren hat das 3GPP (3rd Generation Partnership Project), ein weltweites Standardisierungsgremium für Mobilfunktechnologien wie Funkzugang, Kernnetz und Dienstfunktionen zur Bereitstellung vollständiger Systemspezifikationen für die mobile Telekommunikation, die Funktionalität von C-V2X auf Basis neuester Entwicklungen im Mobilfunk erweitert. Die in Release 16 vereinbarten Standards ebnen den Weg für fortgeschrittene Fahrerassistenzsysteme (FAS).

Auch wenn die breite Verfügbarkeit von selbstfahrenden Kraftfahrzeugen noch Zukunftsmusik ist, gab es bereits einige viel beachtete Versuche. Tesla, Google und BMW sorgen für Schlagzeilen, schüren die Erwartungen der Öffentlichkeit und bringen Schwung in die Sache. Viele Fahrzeuge der Oberklasse verfügen bereits über ein gewisses Maß an Autonomie, einige bis Level 3 (Automatisierter Modus), die ebenfalls auf C-V2X-Technologien beruhen.

Obwohl 4G/LTE-Netze viele der oben genannten Applikationen unterstützen, wird die verfügbare Bandbreite durch das steigende Volumen der gemeinsam genutzten Daten immer stärker beansprucht. Da die kritischen Sicherheits- und Energiemanagementsysteme in den Fahrzeugen immer komplexer werden, werden niedrige Latenzen dringend notwendig. Um ein höheres Maß an Autonomie zu erreichen, müssen die Latenzen der Netzwerke und der Cloud-Edge-Verarbeitungsfunktionen das Niveau menschlicher Reflexe erreichen. Auch für anspruchsvollere FAS muss das vernetzte Fahrzeug in Echtzeit auf Ereignisse in der Umgebung reagieren. Die derzeitigen drahtlosen Netzwerke stoßen hier an ihre Grenzen und werden immer mehr zu einem Hindernis – ohne 5G wird es keine selbstfahrenden Autos geben.

Fazit

Hauptaugenmerk beim Ausbau der 5G-Netze wurde zunächst auf die Verbesserung von 4G/LTE durch Einführung der 3GPP-Spezifikationen für New Radio Non-Standalone (5G NR NSA), Release 15, gelegt, was die Einführung einer begrenzten Anzahl von 5G-Diensten ermöglichte. Das wahre Potenzial von 5G wird jedoch erst bei Einführung von 3GPP, Release 16 und dem angekündigten Release 17, zum Tragen kommen. Applikationen wie autonome Fahrzeuge und autonome Fabriken lassen sich nur dann umsetzen, wenn sie einfachen Zugang zu Netzen dieses Leistungsniveaus haben. Die Einführung von 5G verlief anfangs nur zögerlich, was vor allem an den Auswirkungen der weltweiten Pandemie lag. Die zweite Phase der Einführung des 5G-Netzes wird sicherlich die Nachfrage nach einem breiten Spektrum von noch zu entdeckenden Applikationen beschleunigen.

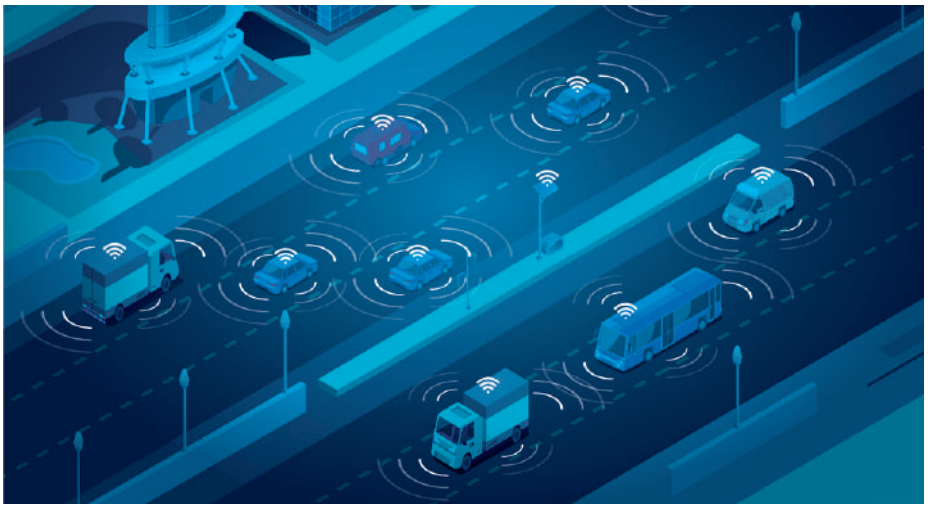
220061-02

Über den Autor

Als Technical Marketing Manager für EMEA bei Mouser Electronics ist Mark Patrick für die Erstellung und Verbreitung von technischen Inhalten in der Region verantwortlich - Inhalte, die für Mousers Strategie zur Unterstützung, Information und Inspiration der Elektronik-Branche von zentraler Bedeutung sind.

Bevor er das Technische Marketing Team leitete, war Patrick Teil des EMEA-Lieferanten-Marketing-Teams und spielte eine wichtige Rolle beim Aufbau und der Entwicklung von Beziehungen zu wichtigen Produktionspartnern. Zusätzlich zu einer Vielzahl von technischen und Marketing-Positionen war Patrick acht Jahre lang bei Texas Instruments in den Bereichen Anwendungsunterstützung und technischer Vertrieb tätig.

Weitere Informationen zu 5G finden Sie auf der Mouser-Website zu „Empowering Innovation Together“ unter **www.mouser.com/empowering-innovation/5G**



Erste Schritte ins **IoT** - mit dem **ESP32-C3**

WLAN-Taster und -Relais

Von Mathias Claußen (Elektor)

Das IoT ist kein Buch mit sieben Siegeln. Clevere Controller wie der neue ESP32-C3 und die einsteigerfreundliche Arduino-IDE machen das Entwickeln von kleinen Projekten einfach.

Wer vom Internet of Things (IoT) redet, meint damit, dass immer mehr Dinge unseres täglichen Lebens an das Internet angeschlossen werden. Das beginnt mit Leuchten, Heizungen und Sensoren in der Wohnung und setzt sich fort über Autos, Verkehrsampeln, Seecontainer und vieles mehr. In jedem der angeschlossenen Dinge sind kleine netzwerkfähige Komponenten verbaut, die den Austausch von Daten ermöglichen.
Wie man eigene Elektronik mit dem IoT verbinden kann, lernt man am besten anhand eines praktischen Beispiels. In diesem Artikel soll ein WLAN-fähiger Taster und ein WLAN-fähiges Relais entstehen; das Relais lässt sich mit dem Taster fernschalten und meldet seinen Status zurück.

Zutatenauswahl

Wie bei allen Projekten müssen zuerst passende Zutaten ausgewählt werden. Hier kommt das ESP-C3-12F-Kit (**Bild 1**) aus dem Elektor-Shop zum Einsatz. Auf diesem Board sitzt ein WLAN-fähiger Mikrocontroller ESP32-C3 von Espressif. Der ESP32-C3 ist als Ersatz für den bewährten ESP8266 angetreten, neben einem modernen CPU-Kern bietet der Chip eine gute Mischung aus integrierter Peripherie, die sowohl einsteigerfreundlich als auch leistungsfähig ist (siehe unser Review des ESP32-C3 [1]). Eine Übersicht der integrierten Hardwareblöcke ist in **Bild 2** zu sehen. Neben dem ESP32-C3 sind auch eine RGB-LED und ein USB-Seriell-Wandler auf dem Board integriert. Für unser Projekt

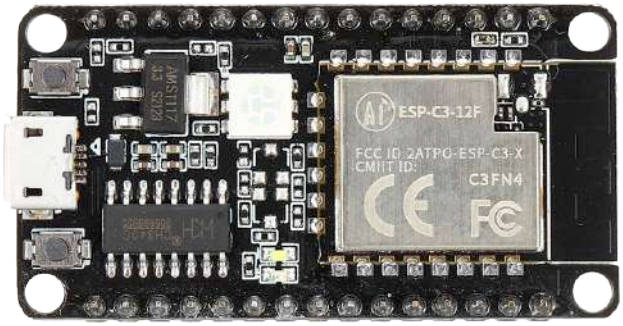


Bild 1. ESP32-C3-12F Kit.

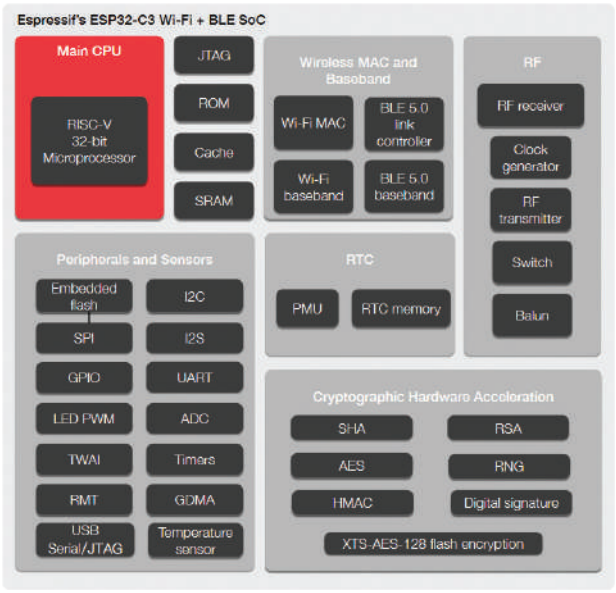


Bild 2. ESP32-C3 Funktionsblöcke (Quelle: ESP32-C3 Datenblatt).

brauchen wir natürlich zwei ESP-C3-12F-Kits.
Neben den ESP-C3-12F-Kits werden auch noch ein Sensor und ein Aktor benötigt. Hier kommt das 37-in-1-Sensor-Kit von Elektor zum Einsatz, in dem 35 Sensoren enthalten sind (von den einst 37 Sensoren wurden zwei quecksilberhaltige Typen inzwischen aus Sicherheitsgründen aus dem Kit entfernt). Eine Übersicht der Sensoren des Kits (**Bild 3**) kann man **Bild 4** und dem Infodokument [2] entnehmen. Wir benötigen zuerst einmal den Jostick, der hier als Taster verwendet wird, und das Relais, das als Aktor zum Einsatz kommt.
Für das Verbinden der Module werden noch ein paar Steckbrücken (weiblich/weiblich) benötigt. Diese sind unter anderem im Maker-Kit „Mini Breadboards & Jumper" von Pimoroni (siehe Kasten **Passende Produkte**) enthalten.
Hinzu kommt noch ein Rechner, zum Beispiel ein Raspberry Pi, der später als Server agiert, über den die IoT-Geräte ihre Daten austauschen können. Ein ausrangierter Raspberry Pi 1 ist hierfür ausreichend, mindestens ein Raspberry Pi 2 wird aber empfohlen. Wer auf der Suche nach einem kleinen, aber leistungsfähigen Raspberry Pi ist, sollte sich einmal das *Raspberry Pi Zero 2 W Bundle* (Kasten) ansehen. Es muss aber kein Raspberry Pi sein, ein PC mit einer Linux-Distribution wie Ubuntu [3] ist auch vollkommen ausreichend.
Bevor wir mit dem eigentlichen Projekt starten, wollen wir uns noch ein klein wenig damit beschäftigen, wie wir Daten übertragen können.

MQTT

IoT-Geräte, ob Sensor oder Aktor, müssen Daten austauschen. Hierzu können entweder proprietäre Protokolle zum Einsatz kommen, bei dem jeder Entwickler das Rad komplett neu erfindet, oder es werden standardisierte Protokolle verwendet. Ein standardisiertes Verfahren, das große Verbreitung gefunden hat, ist MQTT. Ursprünglich war das die Abkürzung von *Message Queuing Telemetry Transport*, doch 2013 wurde offiziell beschlossen, dass MQTT nur noch für sich steht [4].
MQTT ist ein Protokoll, das für den Nachrichtenaustausch durch einen Broker (Server) sorgt, ohne dabei festzulegen, wie die Nachrichten aussehen müssen. Man kann das mit dem Versand eines Briefes vergleichen: Die Logistik und das Format des Umschlages wird durch das Postunternehmen vorgegeben, was aber in dem Brief steht und in welcher Sprache dieser geschrieben ist, kann jeder selbst entscheiden.

Bild 3. Elektor 37-in-1 Sensorkit.

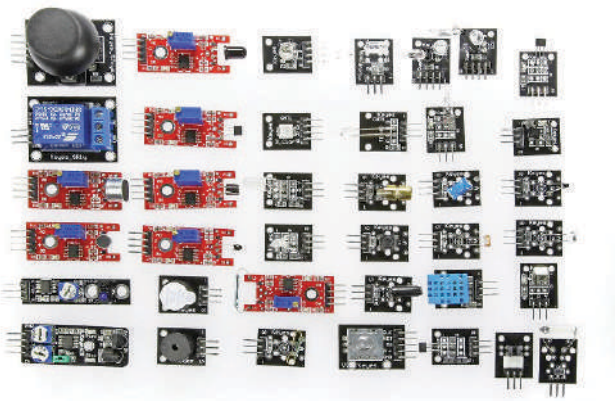


Bild 4. Übersicht der enthaltenen Sensoren.

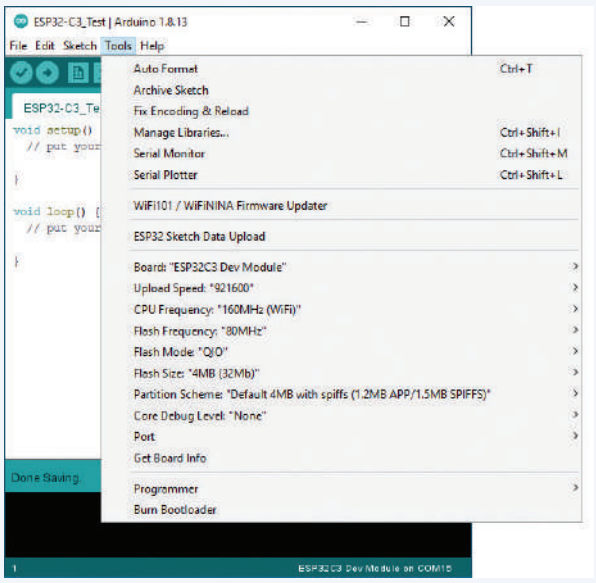


Bild 5. Einstellungen für den ESP32-C3.

Doch welche „Sprache" soll nun für unsere Nachrichten verwendet werden? Auch hier gibt es mehrere Optionen. Eine, die nicht nur für MQTT genutzt wird, ist JSON.

JSON – Java Script Object Notation

JSON ist ein leichtgewichtiges Format für den Datenaustausch, auch kleinere Mikrocontroller können solche Nachrichten leicht generieren und aufschlüsseln. Hinzu kommt, dass JSON auch für Menschen nicht nur einfach zu lesen, sondern auch einfach zu schreiben ist. Eine Übersicht über die JSON-Spezifikation findet man auf der Webseite des JSON-Standards [5]. JSON wird nicht nur in Verbindung mit MQTT verwendet, sondern auch in vielen anderen Bereichen. JavaScript, die Programmiersprache, von der JSON abgeleitet wurde, ist eine der Kerntechnologien, auf denen heute das World Wide Web basiert. Eine gute Einführung in JSON mit praktischen Beispielen findet man auf der Site von Mozilla [6].

Vorbereitung der IoT-Umgebung: Der MQTT-Broker

Wie bei allen Projekten sorgt eine passende Vorbereitung später für weniger Überraschungen. MQTT benötigt einen Broker, der auf einem alten PC oder einem Raspberry Pi installiert werden kann, um Nachrichten auszutauschen. Mit Node-RED steht eine kompletter Werkzeugkoffer für das Entwickeln von Netzwerk-Anwendungen bereit, der nicht nur auf das Verarbeiten von MQTT-Nachrichten beschränkt ist. Node-RED [7] wurde bei Elektor schon häufig eingesetzt, um MQTT-Daten zu verarbeiten, und lässt sich dank der Anleitungen auf dem Raspberry Pi [8] oder dem PC [9] zügig installieren.

Die Arduino-IDE

Als Entwicklungsumgebung kommt die Arduino-IDE zum Einsatz. Der Editor der IDE ist nicht der beste in seiner Klasse, bietet aber momentan die stabilste Unterstützung für den ESP32-C3. Die Arduino-IDE [10] kann kostenfrei von der Arduino-Homepage heruntergeladen und installiert werden. Anschließend muss die Arduino-ESP32-Unterstützung (so wie in der Espressif-Dokumentation [11] beschrieben) installiert werden. Die Einstellungen für das Board sind wie in **Bild 5** vorzunehmen.

Neben der Arduino-ESP32-Unterstützung werden für die ersten Schritte auch noch ein paar Bibliotheken benötigt. Damit der ESP32-C3 Daten per MQTT/JSON versenden kann, wird der *PubSubClient* von Nick O'Leary benötigt und die *ArduinoJson*-Bibliothek von Benoit Blanchon. Diese lassen sich über den Bibliotheks-Manager der Arduino-IDE installieren (**Bild 6** und **Bild 7**).

Zusammenbau der Hardware

Die beiden ESP32-C3-basierten Module werden gemäß den Schaltplänen in **Bild 8** und **Bild 9** zusammengesetzt. Für den Taster und das Relais sind jeweils nur drei Verbindungen herzustellen, wobei zu beachten ist, dass das Relais aus 5 V gespeist wird. Ist alles zusammengesetzt, sieht der Aufbau wie in **Bild 10** aus.

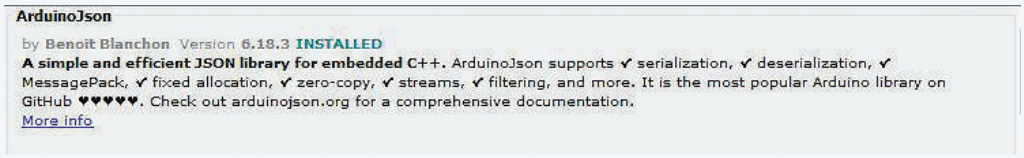


Bild 6. PubSubClient im Arduino Bibliotheks-Manager.

Bild 7. ArduinoJson im Arduino Bibliotheks-Manager.

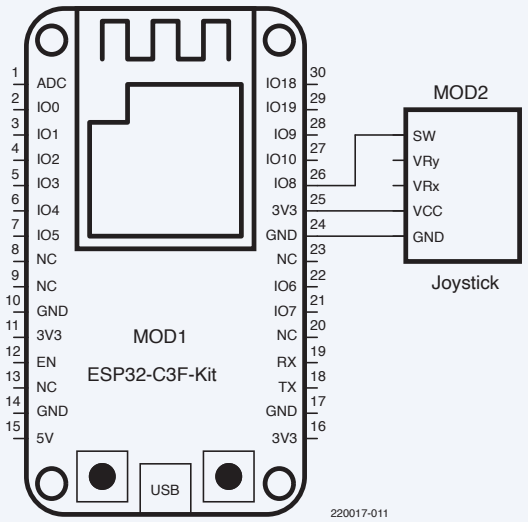


Bild 8. Schaltplan ESP32-C3 und Joystick.

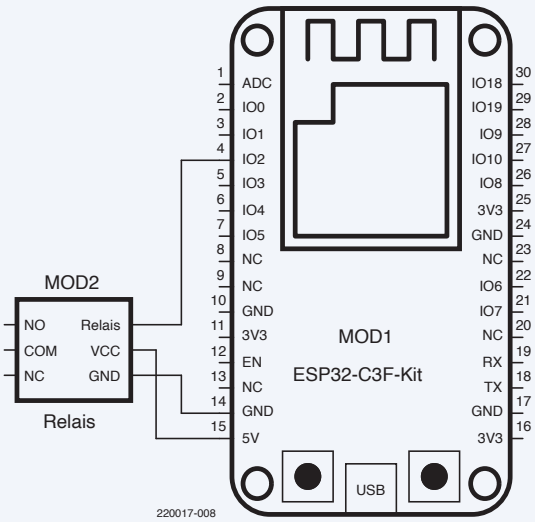


Bild 9. Schaltplan ESP32-C3 und Relais.

Einrichtung der Software

Auch bei diesem Projekt steht der Quelltext wieder auf GitHub [12] bereit. Von dort können die Sketche für die beiden ESP32-C3-Controller heruntergeladen werden. Bevor diese jedoch einfach auf die ESP32s hochgeladen werden können, bedarf es noch etwas Konfiguration. Damit die beiden Controller Daten mit einem MQTT-Broker austauschen können, muss alles passend eingestellt werden. Dazu sind in den beiden Arduino-Sketches am Anfang `#defines` vorhanden.

```
#define WIFI_SSID "changeme"
#define WIFI_PASS "changeme"
#define MQTT_SERVER "test.mosquitto.org"
```

Diese drei `#defines` am Anfang des Arduino-Sketches müssen für das eigene Netzwerk angepasst werden. Man beginnt mit der WLAN-SSID und dem Passwort. Für den MQTT-Server wird die IP-Adresse des Node-RED-Computers im eigenen Netzwerk angegeben. Sind beide Sketche (sowohl für das Relais als auch für den Taster) vorbereitet, dann können diese auf den jeweiligen ESP32-C3 hochgeladen werden. Beide ESP32s sollten danach starten und die große LED auf den Boards jeweils anfangen, weiß zu blinken. Dies zeigt an, dass der ESP32-C3 versucht, sich mit dem WLAN zu verbinden. Ist die Verbindung hergestellt, dann leuchtet die LED dauerhaft. Die Farbe hängt von der Funktion des Boards ab: Weiß leuchtet die LED auf der Platine, an die das Relais angeschlossen ist. Bei der Platine, an die der Taster angeschlossen ist, leuchtet die LED rot (Relais aus) oder grün (Relais an), also je nach Zustand des Relais.

Damit sind beide ESP32-C3 erfolgreich in Betrieb genommen. Ein Druck auf den Taster wird nun für eine Zustandsänderung des Relais sorgen; die Farbe der LED wird von Rot nach Grün oder Grün nach

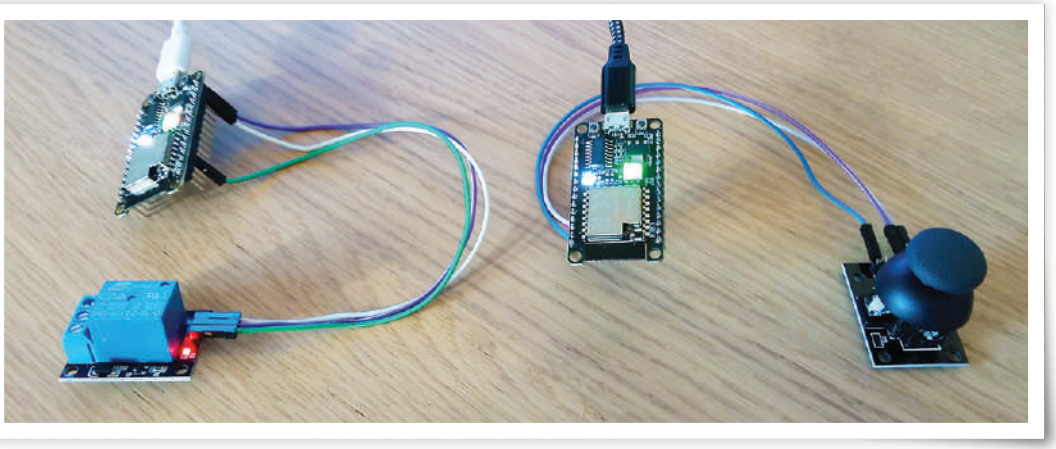


Bild 10. Fertiger Hardwareaufbau.

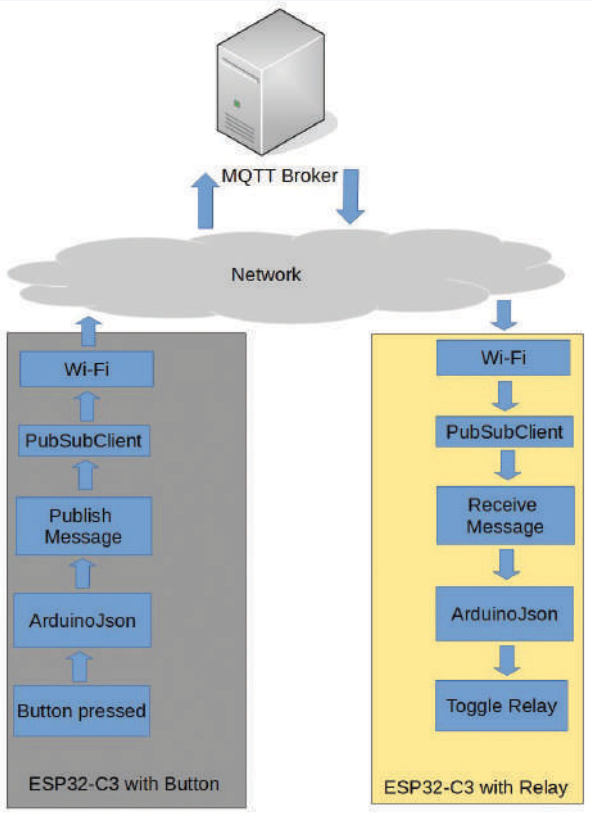


Bild 11. Datentransport zum Broker und zurück.

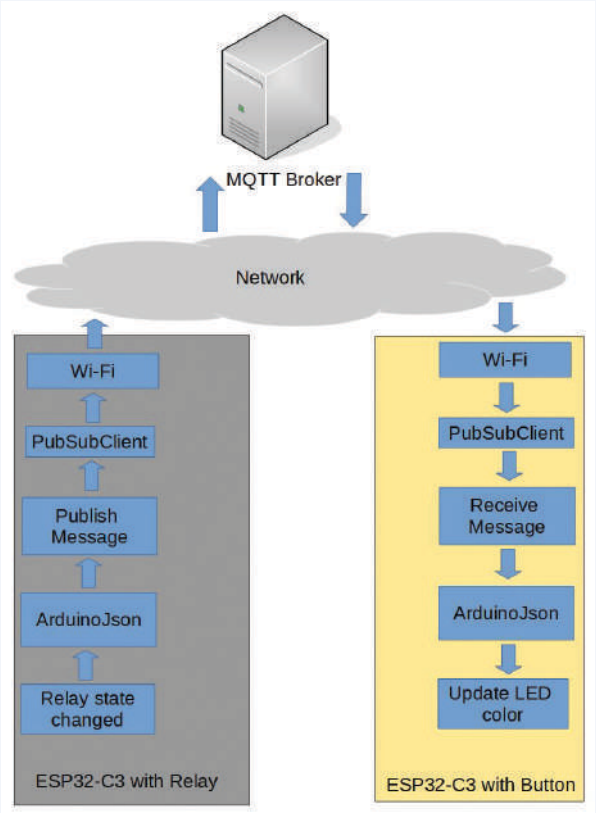


Bild 12. Feedback vom Relais.

Rot wechseln. Der Taster des einen ESP32-C3 kann so erfolgreich das Relais des anderen ESP32-C3 steuern und bekommt den Zustand des Relais auch zurückgemeldet. Die erste eigene IoT-Anwendung läuft! Doch wie funktioniert der Datenaustausch?

Einmal Relais und zurück

Zuerst wollen wir uns den Weg ansehen, den ein Tastendruck bis zum Relais nimmt. In **Bild 11** ist zu sehen, wie die Nachricht Schicht für Schicht verpackt und anschließend per WLAN an den Broker gesendet wird. Im Quelltext geschieht dies durch `client.publish(MQTT_TOPIC_OUT, (const uint8_t*)buffer, n, true);`.

Doch warum heißt die Funktion `publish` und nicht einfach `send`? Dies liegt an der Art, wie bei MQTT die Daten später verteilt werden. Auf einem MQTT-Broker werden Nachrichten anhand eines Topics (Thema) verteilt; in diesem Fall ist das Topic (`MQTT_TOPIC_OUT`) *„BUTTON“*. Beim Verbinden mit dem MQTT-Broker kann der Client, also der ESP32-C3 unseres Relais, mitteilen, welche Themen für ihn interessant sind, diesen Nachrichtenkanal also „abonnieren“.

Jeder Teilnehmer, der dem MQTT-Broker mitgeteilt hat, das er sich für ein bestimmtes Thema interessiert, bekommt die unter diesem Thema gesendeten Nachrichten zugestellt. Der Sender andererseits muss sich nicht um die Verteilung kümmern, sondern seine Nachrichten nur an den Broker senden (sie „veröffentlichen“).

Der ESP32-C3 mit dem Relais abonniert das Thema *BUTTON* in seinem Code mit `client.subscribe(MQTT_TOPIC_IN);` , wobei hier `MQTT_TOPIC_IN` *„BUTTON“* ist. Jedes Mal, wenn nun durch den Taster eine Nachricht ausgelöst wird, geht diese zum MQTT-Broker. Sie wird dann dem ESP32-C3 mit dem Relais zugestellt, was das Relais zum Umschalten bringt.

Wenn das Relais seinen Zustand ändert, wird auch eine Nachricht an den MQTT-Broker unter dem Topic *„RELAIS“* gesendet, in dem der nun aktuelle Zustand (ob an oder aus) festgehalten ist. Der ESP32-C3 mit Taster wiederum hat beim MQTT-Broker das Topic *„RELAIS“* abonniert und bekommt damit, wie in **Bild 12** zu sehen, die Nachricht mit dem neuen Zustand zugestellt. Dies bewirkt, dass die Farbe der LED auf Rot oder Grün gesetzt wird.

Das Schöne an diesem Aufbau ist, dass sich so auch ein zweiter Schalter integrieren lässt, in dem dieser wie der erste Schalter Nachrichten unter dem Topic *„BUTTON“* sendet. Der ESP32-C3 mit dem Relais wird dann die Daten von beiden Tastern entgegennehmen und entsprechende Schalthandlungen durchführen. Sollten Sie Lust bekommen haben, weiter mit MQTT zu experimentieren, dann schauen Sie doch auch einmal die anderen Projekte von Elektor an, bei denen per MQTT Daten versendet werden. Beispiele sind die Wetterstation [13] oder

die Monster-LED-Uhr mit externem Temperatursensor [14]. Mehr zu MQTT und wie man damit zum Beispiel Daten an Cloud-Plattformen senden kann, findet man auch in der Serie „Mein Weg in das IoT" [15].

Zusammenfassung

IoT muss nicht kompliziert sein: Mit einfachen Mitteln und dem Wissen um die passenden Zutaten lassen sich schnell IoT-Geräte entwickeln. Diese können komplexer sein als nur ein Taster oder ein Relais. Von der Heizungssteuerung bis zur Türklingel: Der Fantasie sind keine Grenzen gesetzt. Ein ESP32-C3 ist in jedem Fall eine preiswerte Basis für eigene Versuche!

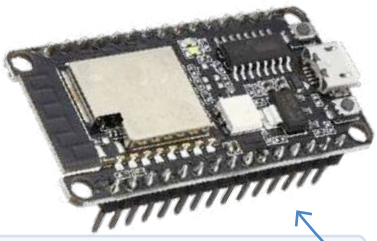
220017-02

Ein Beitrag von

Entwurf und Text: **Mathias Claußen**
Redaktion: **Jens Nickel**
Layout: **Giel Dols**

Sie haben Fragen oder Kommentare?

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



WEBLINKS

[1] Mathias Claußen, „ESP32-C3: 32-Bit-RISC-Einkerner", Elektormagazine.de: www.elektormagazine.de/news/esp32c3-32bitrisceinkerner
[2] Elektor 37-in-1 Sensor-Kit Dokumentation: www.elektor.com/amfile/file/download/file/1170/product/6171/
[3] Ubuntu Linux Distribution: https://ubuntu.com/
[4] OASIS MQTT TC Minutes vom 25.04.2013: www.oasis-open.org/committees/download.php/49028/OASIS_MQTT_TC_minutes_25042013.pdf
[5] JSON.org: www.json.org/json-de.html
[6] Mozilla Web Docs: Working with JSON: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON
[7] Node-RED: https://nodered.org/
[8] Node-RED Installation auf dem Raspberry Pi: https://nodered.org/docs/getting-started/raspberrypi
[9] Node-RED Installation auf dem PC: https://nodered.org/docs/getting-started/local
[10] Arduino-IDE Download: www.arduino.cc/en/software
[11] Espressif Arduino-ESP32 Installationsanleitung: https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html
[12] Elektor GitHub Repository: https://github.com/ElektorLabs/220017-ESP32-C3-and-IoT_First-steps
[13] Roy Aarts, „Wetterstation mit ESP32", Elektor 1-2/2019: www.elektormagazine.de/magazine/elektor-69/42263
[14] Mathias Claußen, „Monster-LED-Uhr mit WLAN und Temperatur-Anzeige", Elektor 5-6/2019: www.elektormagazine.de/magazine/elektor-95/42563
[15] Jens Nickel, „Mein Weg in das IoT", Elektormagazine.de: www.elektormagazine.de/tags/mein-weg-ins-iot

Ein genauerer Blick auf das **WizFi360-Modul** von **WIZnet**

Ein Beitrag von WIZnet

WizFi360 ist ein Wi-Fi-Modul für das Internet der Dinge (IoT), das von WIZnet Co. Ltd Südkorea hergestellt wird. Aufgrund der niedrigen Kosten, der geringen Größe und der Anpassungsfähigkeit an eingebettete Geräte wird das WizFi360 mittlerweile in vielen privaten und industriellen IoT-Geräten eingesetzt. In diesem Artikel erläutern wir die Hauptmerkmale der WizFi360-Module und Evaluierungsboards.

Was ist das WizFi360?

Das WizFi360 ermöglicht es Mikrocontrollern, sich mit 2,4-GHz-WiFi unter Verwendung des IEEE 802.11b/g/n Standards zu verbinden. Das Modul wurde entwickelt, um Entwicklern eine einfache Wi-Fi-Lösung zu bieten. Für eine einfache Migration von ESP8266-Projekten unterstützt das WizFi360 einen Espressif-ähnlichen AT-Befehlssatz.

Die folgenden Eigenschaften des WizFi360 wurden dem Datenblatt entnommen [1]:

- WiFi 2,4 GHz, 802.11 b/g/n
- Betriebsarten Station / SoftAP / SoftAP+Station
- UART / SPI-Schnittstellen
- Übertragungs-Modi "Data pass-through" und "AT-Befehle"
- Baudrate bis zu 2 Mbps mit 16 häufig genutzten Übertragungsgeschwindigkeiten
- Unterstützt Firmware-Upgrade über UART/OTA (über WLAN)
- Industrietauglich (Betriebstemperaturbereich: -40 °C bis +85 °C)
- CE, FCC, KC, K-MIC (TELEC), RoHS, REACH Zertifizierung

Das WizFi360 ist in zwei Varianten erhältlich: mit integrierter Antenne oder mit u.Fl-Anschluss. Außerdem ist das Modul in verschiedenen Formfaktoren erhältlich (**Bild 1**). Das WizFi360io-C ist ein Modul mit SMW200-06-Anschluss und 5V-Unterstützung, das WizFi360io-H ist ein Modul mit 2,00-mm-Stiftleiste.
WIZnet bietet auch verschiedene Evaluierungsboards für einfaches Projekt-Prototyping zum Kauf an (**Bild 2**). Das Arduino-kompatible WizFi360-EVB-Shield kann für Experimente, Tests und Überprüfungen des WizFi360 verwendet werden. Kürzlich hat WIZnet das RP2040-basierte WizFi360-EVB-Pico herausgebracht - ein Raspberry Pi Pico-Klon mit zusätzlicher WiFi-Konnektivität.

Zusammenarbeit mit Microsoft, um Lösungen für das IoT zu beschleunigen

Das WizFi360 ist ein kleines, aber leistungsstarkes Modul, das SSL/TLS mit der MbedTLS-Bibliothek unterstützt. Mit seiner Hilfe können Nutzer das WizFi360 einfach mit der Cloud oder einem MQTTS-Broker verbinden. Im Oktober 2019 gab WIZnet bekannt, dass es bei Microsoft als „Azure Certified for Internet of Things (IoT)" zertifiziert wurde. Durch vorhergehende Tests der Hard- und Software wird die Kompatibiität zu Microsoft Azure IoT-Diensten sichergestellt, so dass Kunden ihre IoT-Lösungen schnell in Betrieb nehmen können.

WizFi360 in verschiedenen Ökosystemen

DIY-Enthusiasten und Bastler können ihre Anwendungen für das WizFi360 leicht mit der WizFi360 Arduino-Bibliothek entwickeln. Diese Bibliothek wird derzeit von unseren Freunden bei Kocoafab gepflegt [2].

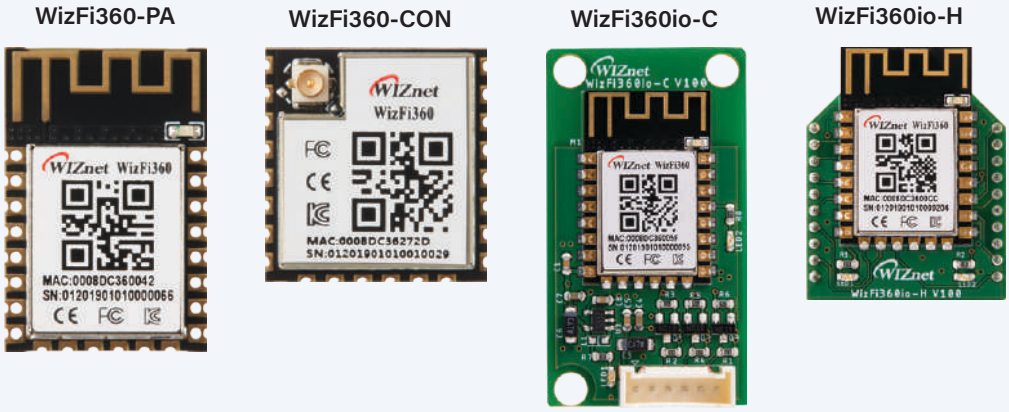


Bild 1. Das WizFi360 ist in zwei Varianten erhältlich: mit integrierter Antenne oder mit u.Fl-Anschluss. Außerdem ist das Modul in verschiedenen Formfaktoren erhältlich.

Bild 2. WIZnet bietet auch verschiedene Evaluierungsboards für einfaches Projekt-Prototyping zum Kauf an.

Da WIZnet Espressif-ähnliche AT Kommandos implementiert hat, können Nutzer außerdem die WiFiEsp Bibliothek nutzen. Weitere Details und Anleitungen finden sich unter [3].

Für Boards der RP2040-Familie hat WIZnet bereits ein C/C++ SDK und Beispielcode veröffentlicht [4]. CircuitPython und MicroPython SDK sind ebenfalls in Vorbereitung.

Bereits lange vor der Zertifizierung des WIZwiki-W7500 durch Arm Mbed im Jahr 2015 ist WIZnet dem ARM Ökosystem beigetreten. Um die langjährige Partnerschaft zwischen den beiden Unternehmen fortzuführen und zu vertiefen, wurde das WizFi360 als eines der offiziellen Shields für die „Wi-Fi Shield on Arm Open-CMSIS-Pack"- und die Keil Studio Cloud-Entwicklung ausgewählt.

Wir bei WIZnet versuchen, möglichst viele Tools und Beispielcodes zur Verfügung zu stellen, um die Arbeit der Entwickler zu erleichtern.

Wie kann man es beschaffen?

Trotz der weltweiten Silizium- und Halbleiterknappheit tut WIZnet sein Bestes, um die Preise für seine Produkte nicht zu erhöhen. Im Gegensatz zu vielen anderen Herstellern wurden im März 2022 die Preise für WizFi360 weltweit um bis zu 20% gesenkt. Außerdem führt WIZnet einen neuen Direktvertrieb [5] für VAR-Partner (Value-Added-Resellers) und UCC-Maker (User-Created-Content) ein, bei dem Sie das WizFi360 zu einem noch günstigeren Preis bestellen können.

Sind Sie daran interessiert, das WizFi360 zu testen?

WIZnet veranstaltet im Sommer einen WizFi360 Design-Wettbewerb. Um die Einführung des WizFi360-EVB-Pico zu feiern, bieten wir kostenlose Muster der Produkte WizFi360-Module, WizFi360-EVB-Mini oder WizFi360-EVB-Pico für alle Teilnehmer an. Dies ist eine großartige Möglichkeit Ihre eigenen Ideen mit Hilfe einer Vielzahl von Open Source-Beispielen umzusetzen. Es werden unter anderem 30 iPads der neuesten Generation unter den Gewinnern verschenkt. Alle Details sind unter dem Link [6] zu finden.

Zum Schluss

Dieser Artikel beschreibt das WiFi-Modul WizFi360 im Detail und wie es für die Entwicklung verschiedener Anwendungen in unterschiedlichen Ökosystemen eingesetzt werden kann. Seit seiner Einführung hat WIZnet bereits mehr als 1 Million Module weltweit verkauft.

Das WiFi-Modul WizFi360 ist die perfekte Wahl für mobile drahtlose Anwendungen wie Fernüberwachung und Sensoranwendungen. Die einfache Integration und Programmierung kann die Entwicklungszeit erheblich reduzieren und die Systemkosten minimieren. Abhängig von Ihren Projektanforderungen bietet WIZnet einen Service zur Anpassung der Firmware an. Egal ob maßgeschneiderte AT-Kommandos oder ein anwendungsspezifisches Programm: Wir sind bereit, Sie bei Ihrem Projekt zu unterstützen.

220277-02

Andere WIZnet Wi-Fi-Module

Seit 2015 hat WIZnet einige Wi-Fi-Module mit Chips von verschiedenen Herstellern herausgebracht. Im aktuellen Lineup sind dies:

- WizFi630S [7] – ein Gateway-Modul basierend auf Linux, das 1T1R 802.11 WiFi, eine 580 MHz CPU, Ethernet PHY, USB2.0 Host, SD-XC, I2S, I2C und weitere GPIOs integriert hat.
- WizFi310 [8] - ein kleines drahtloses Modul für die höchste Integrationsstufe mit 802.11b/g/n. Dieses Modul ist in den von Ubidot unterstützten Boards enthalten.

Informationen über andere Module finden Sie auf der WIZnet-Website www.wiznet.io.

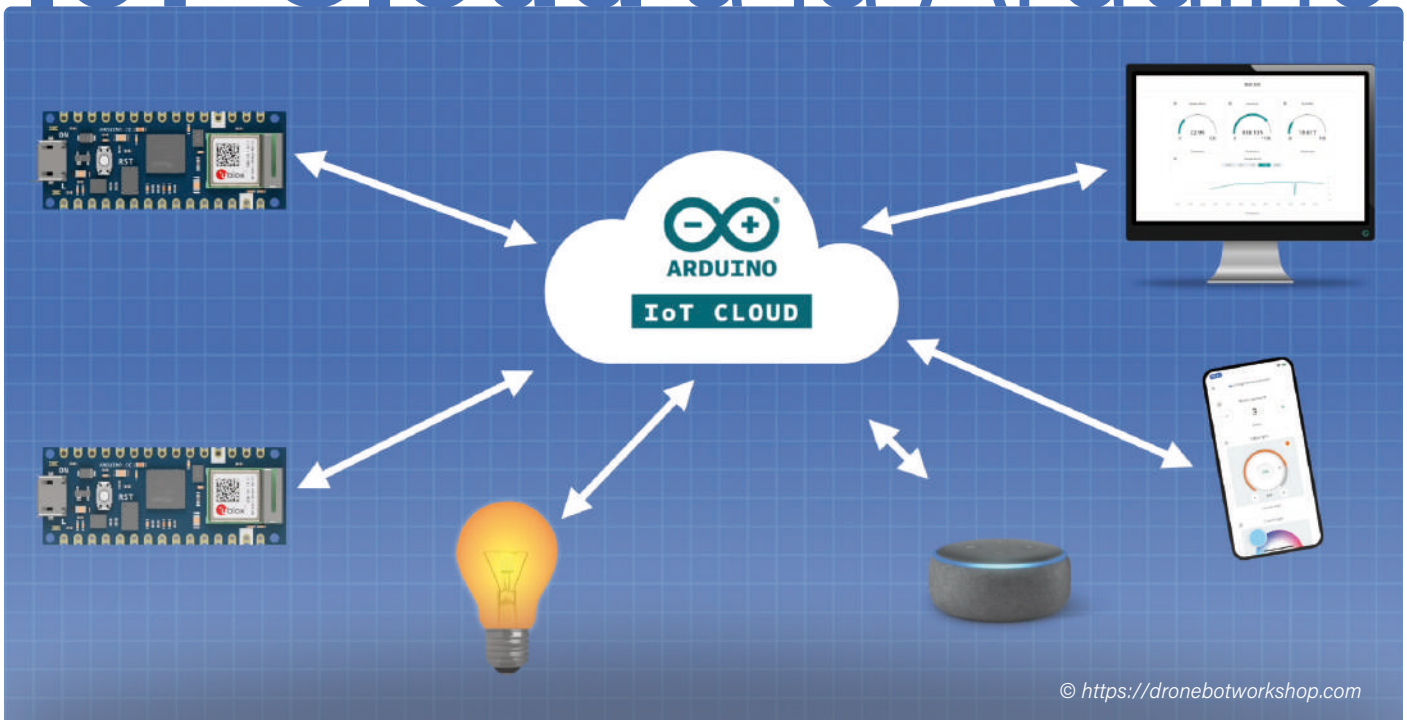
Über WIZnet

WIZnet ist ein IT-Unternehmen ohne eigene Produktionsstrecken (fabless), das netzwerkfähige Prozessoren für das Internet der Dinge (IoT) anbietet. WIZnet ist der einzige Innovator, der 2001 die festverdrahtete TCP/IP-Technologie in einem Mikroprozessor-Chip patentieren ließ. Seitdem werden jährlich durchschnittlich 10 Millionen WIZnet-Chips in verschiedenen eingebetteten Netzwerkgeräten weltweit eingesetzt. Seit der Etablierung eines loyalen Markenimages bei den Kunden kooperiert WIZnet mit 70 Distributoren weltweit mit Niederlassungen in den USA, China und Indien, um wettbewerbsfähiges Projektmarketing und schnelleren technischen Support zu gewährleisten.

WEBLINKS

[1] WizFi360 Datenblatt: https://docs.wiznet.io/Product/Wi-Fi-Module/WizFi360/documents
[2] Arduino-Bibliothek: www.arduino.cc/reference/en/libraries/wizfi360/
[3] WiFiEsp-Bibliothek: https://github.com/wizfi/WizFi360EVB-Arduino
[4] C/C++ SDK und Beispielcode: https://docs.wiznet.io/Product/Open-Source-Hardware/wizfi360-evb-pico/
[5] Direktvertrieb: https://direct.wiznet.io
[6] WizFi360 Design-Wettbewerb: https://maker.wiznet.io
[7] WizFi630s: www.wiznet.io/product-item/wizfi630s
[8] WizFi310: www.wiznet.io/product-item/wizfi310

IoT-Cloud à la Arduino



Von Tam Hanna (Slowakei)

Wer einen IoT-Clouddienst realisiert, verwendet einen MQTT-Broker. Diese Binsenweisheit gilt zumindest für alle Cloudsysteme der „großen“ Anbieter. Mit der Arduino-Cloud schickt das Arduino-Entwicklerteam nun eine Cloud-Variante ins Rennen, die durch eine andere Herangehensweise beeindruckt. Mit einer kleinen Anwendung wollen wir einen ersten Kontakt mit der Wolke aufnehmen: Wir senden einen Variablenwert an die Cloud und lassen eine LED blinken.

Die Basis der Arduino-Cloud ist das *Thing*. Darunter verstehen Massimo Banzis Leute ein Objekt der realen Welt wie einen Server, ein Controllerboard oder eine ähnlich intelligente Einrichtung. Analog zum in der „großen Cloud-Welt“ verbreiteten mentalen Konstrukt des Digital Twin [1] gilt auch im Fall des Things, dass dieses eine Reihe von Eigenschaften bereitstellt, die in der Arduino-Cloud *Variables* genannt werden.

Wer profitiert von der Arduino-Cloud?

Schon an dieser Stelle sei angemerkt, dass die Arduino-Cloud explizit kein Ersatz für Amazon AWS IoT Core, Microsoft IoT Hub oder Yandex IoT Core ist. Sofern Sie große Mengen von Geräten zu verwalten haben, ist die Nutzung der alteingesessenen IoT-Cloud-Dienste anzuraten.

Das Alleinstellungsmerkmal der Arduino-Cloud besteht darin, dass die in den diversen Things angelegten Variablen nicht nur für dieses Thing sichtbar sind - ein Webdienst erlaubt je nach kommerzieller Ausbaustufe mehr oder weniger Interoperabilität. Zudem gibt es einen nach Ansicht des Autors sehr leistungsstarken Web-Editor zur Erzeugung von Dashboards. Die Arduino-Cloud kann ihre Stärken immer dann ausspielen, wenn es um einfache Entwicklungen mit hohen Ansprüchen sowohl an einfache Programmierbarkeit als auch an die Attraktivität der Dashboards geht. Dagegen ist eine bequeme Provisionierung der Endgeräte kein Ziel der Arduino-Cloud, die hier immer „von Hand“ durch den Entwickler zu erfolgen hat.

Einrichtung der Hardware

Arduino-Cloud ist per se kein kostenloser Dienst. Die Preisstaffel in **Bild 1** informiert darüber, dass ein Großteil der interessanten Funktionen, zum Beispiel die Interaktion mit Fremd-Diensten, bezahlt werden muss. Logischerweise ist ein derartiger Dienst immer dann besonders profitabel, wenn er für möglichst viele Kunden benutzbar ist. Im Fall der Arduino-Cloud wirkt sich dies unter anderem dadurch aus, dass der Service nicht nur für hauseigene Boards (**Tabelle 2**) verfügbar ist, sondern auch viele andere Plattformen unterstützt, wie ein Blick auf die Kompatibilitäts-Liste [2] beweist. Neben Boards auf Basis des ESP8266 und des ESP32 gibt es Treiberbibliotheken, die verschiedenen Linux-basierten Systemen das Hoch- und Herunterladen von Informationen in die Arduino-Cloud erlauben.

Der Autor dieser Zeilen wird - schon aus reiner Bequemlichkeit - in den folgenden Schritten auf einen Nano RP2040 Connect setzen: Dieses auf dem Pico-Mikrocontroller der Raspberry Pi Foundation

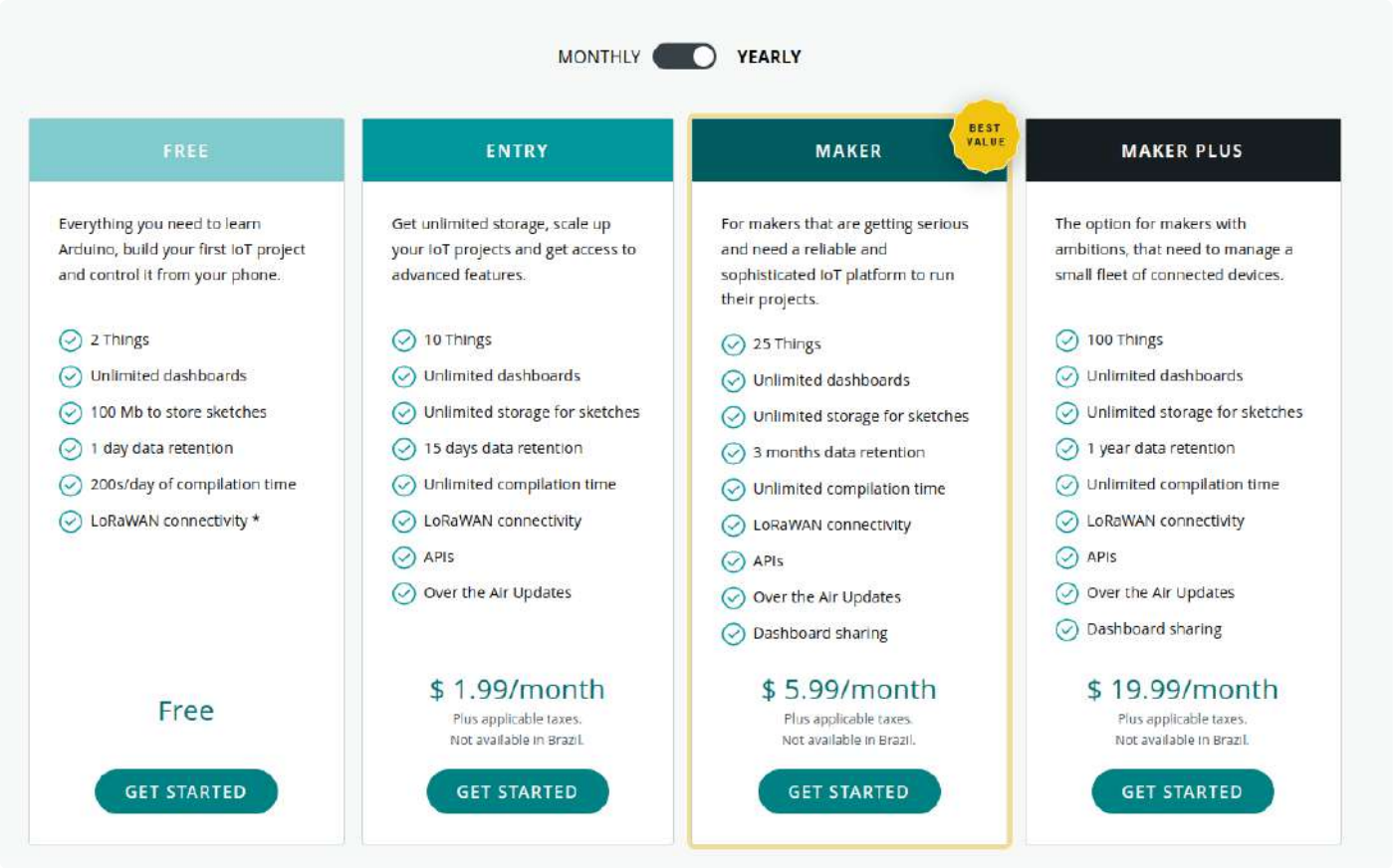


Bild 1. Mit der Arduino-Cloud unternimmt Arduino erste Schritte in Richtung „Recurring Revenue" (Stand 20.1.2022).

basierende Board bringt ein WLAN- Modul aus dem Hause u-blox mit, was die Inbetriebnahme erleichtert. Bevor es mit der Einrichtung der Arduino-Cloud losgeht, wird das Board per USB-Kabel am Computer angeschlossen.

Der erste Schritt besteht darin, die Webseite [3] zu besuchen und ein neues Arduino-Konto zu beantragen. Für unsere kleinen Experimente reicht das kostenlose Konto aus. Die Arduino-Cloud ist explizit an Entwickler gerichtet, die in dieser Beziehung über keine große Erfahrung verfügen. Zunächst klicken wir auf den Knopf *Create Thing*, um ein neues Thing anzulegen.

Lohn ist die Übersicht in **Bild 2**, deren dreigeteilte Konfiguration die fehlenden Elemente sichtbar macht. Da der Autor für dieses Experiment sein Arduino-Konto von allen Konfigurationsparametern befreit hat, können wir mit der „frischen" Einrichtung eines Geräts beginnen. Die folgenden Schritte erfolgen unter Windows 10, wobei die Arbeit unter Ubuntu Linux identisch ist; unter Unix funktioniert die Hardwareerkennung aber meist besser.

Zunächst klicken wir auf das Verknüpfungs-Symbol in der Rubrik *Device* und entscheiden uns danach für die Rubrik *Set Up an Arduino Device*. Einige Sekunden nach dem Anklicken dieser Option weist das Backend darauf hin, dass die als *Arduino Create Agent* bezeichnete Komponente fehlt - klicken Sie auf den Download-Knopf, um die Software herunterzuladen und wie gewohnt zu installieren.

Beachten Sie, dass der *Create Agent* browserspezifisch ist: Wenn Sie die Installation unter Chrome durchführen, müssen sie eine abermalige Installation vornehmen. Eventuell aufpoppende Firewall-Warnungen werden abgenickt. Achten Sie dabei darauf, den Zugriff auf private und auf öffentliche Netzwerke gleichermaßen zu erlauben. In der Folge nistet sich der *Arduino Create Agent* in ihrer Task-Leiste ein - in manchen Fällen muss er danach nochmals aus dem Startmenü heraus aufgerufen werden, was die

Tabelle 1. Arduino-Boards für die Arduino Cloud.

WLAN

- MKR 1000 WiFi
- MKR WiFi 1010
- Nano RP2040 Connect
- Nano 33 IoT
- Portenta H7

LoRaWAN

- MKR WAN 1300
- MKR WAN 1310

GSM/NB-IoT

- MKR GSM 1400
- MKR NB 1500

ESP32/ESP8266

- große Anzahl von Boards

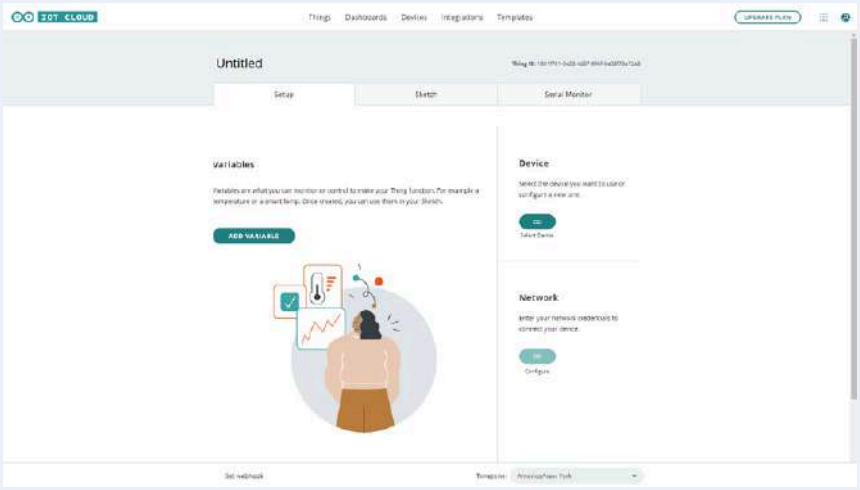


Bild 2. Die IoT-Cloud führt den Entwickler Schritt für Schritt ans Ziel.

Bild 3. Dieser Arduino ist mit der Cloud verbunden.

Treiber-Installation abschließt.
Im nächsten Schritt aktualisieren wir die Ansicht, bis uns die Arduino-Cloud darüber informiert, dass unser Nano RP2040 Connect erkannt ist. Klicken Sie danach auf den *Configure*-Knopf, um den Konfigurationsassistenten zu starten - er fragt nach einem „freundlichen" Namen und initialisiert danach das Secure-Element des Zielsystems mit der grundlegenden Kommunikations-Software.
Im Rahmen der Provisionierung kommt es unter Windows manchmal zu Problemen. Die zuverlässigere und erfolgreiche Methode ist es, den Provisionierungs-Schritt stattdessen unter Linux durchzuführen. Die Rubrik *Network* wird dabei von der Arduino-Cloud übrigens nicht automatisch freigegeben. Sie steht erst dann zur Verfügung, wenn Sie eine der für den Datenaustausch vorgesehenen Variablen erzeugen.
Deshalb klicken wir zunächst auf *Variables*, was den Dialog zum Hinzufügen einer neuen Variable öffnet. Als erstes vergeben wir den Namen `ledIntenBool` und entscheiden uns im Datentyp-Feld für den Wert *Boolean*. Die Arduino-Cloud unterstützt lustigerweise nicht nur C-Programmiereinheiten, sondern realisiert auch „Wrapper" um Größen der realen Welt.
Möchten Sie sich aber ausschließlich auf C beschränken, so empfiehlt sich die Auswahl der Option *Basic Types*. In den Feldern *Variable Permission* und *Variable Update Policy* könnten wir dann theoretisch Einstellungen vornehmen, aber die Voreinstellungen reichen für uns aus, weshalb wir den Dialog schließen. Im nächsten Schritt legen wir dann noch ein ein Feld vom Typ *Integer Number* an, dem wir den Namen `ledIntenInt` verleihen.

Bearbeitung des Codes

Nach dem Anlegen der Variablen sehen Sie im Sketch-Tab rote Bubbles, die sie über Änderungen an der Programmstruktur informieren. Jetzt ist es möglich, auf das Verknüpfungs-Symbol in der Rubrik *Network* zu klicken, um die Einstellungen für das WLAN festzulegen. Der Autor empfiehlt Ihnen hier dringlich, die Werte auf einem Linux-Laptop mit einem Kommandozeilen-Tool wie *iwlist* zu ermitteln und über die Zwischenablage zu übernehmen.
Im nächsten Schritt wechseln wir auf den Reiter *Sketch* und klicken auf das *Verify and Upload*-Symbol. Die Arduino-Cloud beginnt daraufhin mit der Kompilation des Codes und schickt ihn danach unter Nutzung des *Arduino Cloud Agent* auf den angeschlossenen RP2040. Bei erfolgreicher Auslieferung des Kompilats wird in der Cloud jedenfalls die Meldung „*Untitled_dec25a uploaded successfully on board Arduino Nano RP2040 Connect (/dev/ttyACM0)*" angezeigt. Spätestens nach dem obligatorischem Reset wird der RP2040 damit beginnen, über seinen WLAN-Transmitter nach Hause zu telefonieren. Mehrfaches Drücken von *F5* führt nach einiger Zeit dazu, dass das Gerät wie in **Bild 3** gezeigt mit „*Status: online*" erscheint. Besitzer eines kostenpflichtigen Abonnements können Software-Updates auch direkt per WLAN ausliefern; für unsere kleinen Experimente mit dem kostenlosen Account ist eine Kabelverbindung erforderlich.

Analyse des Codes, zum Zweiten

Der im *Sketch*-Tab eingeblendete Editor ist nicht wirklich komfortabel. Der Button *Open full editor* führt zu einer vollwertigen cloudbasierten IDE, mit der sich kleinere Projekte komfortabel bearbeiten lassen. Als Erstes wollen wir uns den Inhalt der Datei *thingProperties.h* ansehen, die strukturale Elemente enthält.
Zuerst sehen wir die folgenden Deklarationen, die die für den WLAN-Zugriff erforderlichen Elemente bereitstellen:

```
const char SSID[] = SECRET_SSID; // Network SSID (name)
const char PASS[] = SECRET_PASS; // Network password
// (use for WPA, or use as key for WEP)
```

Den Eintrag von Namen und Passwort erledigt die Arduino-Cloud über die schon in der Rubrik *Network* eingepflegten Einstellungen. Der nächste Teil betrifft die beiden folgenden Variablen, deren Namen von der Einrichtung aus der *Variable*-Rubrik bekannt erscheinen sollten:

```
int ledIntenInt;
bool ledIntenBool;
```

Die Arduino-Cloud realisiert die Variablen „im Backend" als gewöhnliche C-Variablen, die mit zusätzlicher Intelligenz ausgerüstet werden. Die Intelligenz findet sich unter anderem in der Methode `initProperties`, die sich nach folgendem Schema um die Einrichtung der für die Cloud-Kommunikation notwendigen Primitiva und Strukturen kümmert:

```
void initProperties() {
ArduinoCloud.setThingId(THING_ID);
ArduinoCloud.addProperty(ledIntenInt, READWRITE,
ON_CHANGE, onLedIntenIntChange);
```

```
  ArduinoCloud.addProperty(ledIntenBool, READWRITE,
ON_CHANGE, onLedIntenBoolChange);
}
```

Besonders interessant ist hier die Methode `addProperty`, die sich um die „Anmeldung" der Attribute kümmert. Beachten Sie insbesondere das Übergeben der Funktionspointer `onLedIntenIntChange` und `onLedIntenBoolChange` - sie werden später noch eine wichtige Rolle spielen.
Die eigentliche Logik der Applikation findet sich dann in dem Sketch, der mit der (hier nicht abgedruckten) Inklusion der Header beginnt. Im nächsten Schritt findet sich die Initialisierung, die nach folgendem Schema abläuft:

```
void setup() {
 Serial.begin(9600);
 delay(1500);

 initProperties();

 ArduinoCloud.begin(ArduinoIoTPreferredConnection);
 setDebugMessageLevel(2);
 ArduinoCloud.printDebugInfo();
}
```

Aus Sicht der Arduino-Programmierumgebung ist die Arduino-Cloud ein Hardwaretreiber wie jeder andere. Das globale Objekt `ArduinoCloud` exponiert dabei eine Gruppe von Funktionen, über die Ihr Code mit dem Cloud-Treiber kommuniziert. Von besonderer Wichtigkeit ist hier der Aufruf von `setDebugMessageLevel`, der die „Gesprächigkeit" des Treibers festlegt - desto höher der numerische Wert, desto mehr Debugger-Informationen gibt der Cloud-Treiber über die serielle Schnittstelle des Boards aus.
Bei „komplizierten" Treibern stellt sich immer auch die Frage, wie die Bereitstellung der Rechenleistung erfolgt. Das von der Arduino-Cloud für uns angelegte Projektskelett beantwortet diese Frage in der Methode `loop`, die sich nach folgendem Schema um die Zuweisung von Rechenleistung kümmert:

```
void loop() {
 ArduinoCloud.update();
}
```

Von Haus aus spendiert uns die Arduino-Cloud außerdem noch die drei folgenden Listener-Methoden:

```
void onTestScheduleChange() {
}
void onLedIntenBoolChange() {
}
void onLedIntenIntChange() {
}
```

`onLedIntenBoolChange` und `onLedIntenIntChange` sind dabei für die weiter oben im Backend angelegten Variablen verantwortlich, während `onTestScheduleChange` „interne" Funktionen der Arduino-Cloud realisieren hilft.

Im nächsten Schritt wollen wir uns darum kümmern und der Cloud-Lösung Eigenintelligenz einzuschreiben. Der verwendete Arduino hilft uns dabei insofern, als dass er neben der gewöhnlichen (hier roten) LED an Pin 13 auch eine RGB-LED mitbringt, die per Pulsbreitenmodulation ansprechbar ist (Kanäle LEDR, LEDG und LEDB).
Als Nächstes kehren wir deshalb in die Funktion `setup` zurück, in der wir die verschiedenen Pins initialisieren:

```
void setup() {
...
 ArduinoCloud.printDebugInfo();

 pinMode(LED_BUILTIN, OUTPUT);
 pinMode(LEDB, OUTPUT);
}
```

Von der Cloud ausgelöste Änderungen führen logischerweise zur Aktivierung der Listener, in denen wir die angelieferten Werte in Richtung der Hardware weiterschreiben:

```
void onLedIntenBoolChange() {
 digitalWrite(LED_BUILTIN, ledIntenBool);
}
void onLedIntenIntChange() {
 analogWrite(LEDB, ledIntenInt);
}
```

An dieser Stelle können Sie den Sketch abermals auf die Platine übertragen. Die RGB-LED spielt insofern eine Sonderrolle, als dass sie durch eine Ausgabe einer Null aktiviert wird. Da unsere frisch initialisierten Variablen den in **Bild 4** gezeigten Zustand aufweisen, leuchtet nach der erfolgreichen Initialisierung die blaue Diode der RGB-LED auf.

Modifizieren des Variableninhalts

Böse Zungen behaupten, dass der Grad beziehungsweise die Geschwindigkeit der Management-Adoption eines IoT-Systems zu einem wesentlichen Teil von der grafischen „Qualität" von Dashboards und Co. beeinflusst wird. Der Autor möchte sich

Bild 4. Die Arduino-IoT-Cloud informiert detailliert über den aktuellen Zustand der in einer Variable gespeicherten Informationen.

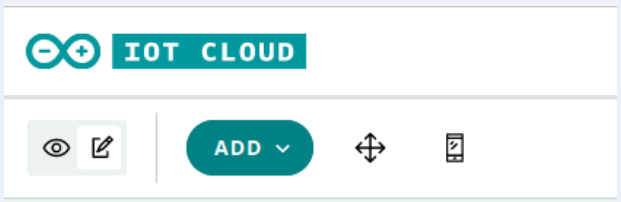


Bild 5. Die oben links eingeblendeten Bearbeitungs-Steuerelemente legen den Betriebsmodus fest.

Bild 6. Die Bearbeitungs-Oberfläche für Steuerelemente ist „modal".

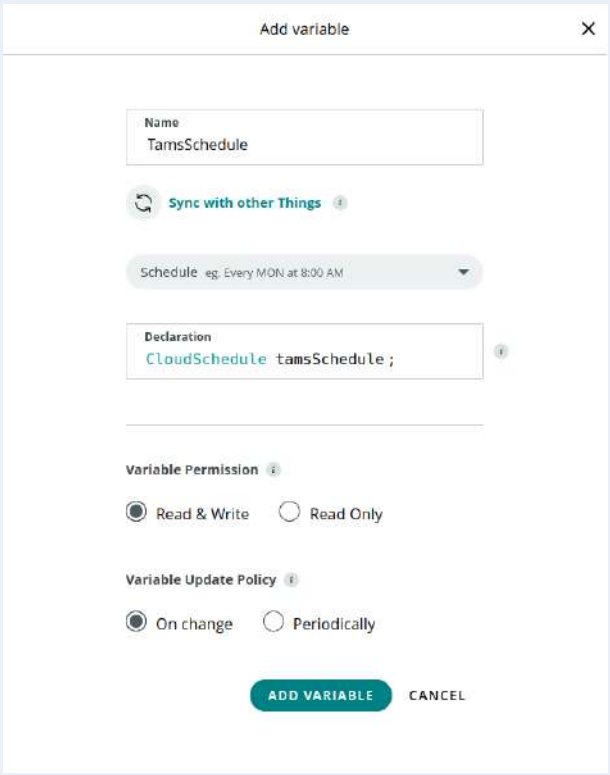


Bild 7. Der Datentyp Cloud Scheduler ist eine „normale" Variable wie jede andere.

an dieser Stelle zu diesem häufig geäußerten Vorurteil nicht weiter äußern und empfiehlt stattdessen, in das Haupt-Backend der Arduino-Cloud zurückzukehren. Klicken Sie dort auf das *Dashboards*-Tab, das sie im Fall eines jungfräulichen Accounts auffordert, ein neues Dashboard anzulegen. Klicken Sie den *Build Dashboard*-Knopf an, um den Editor zu starten, was auch bei schneller Internetverbindung die eine oder andere Gedenksekunde in Anspruch nimmt.

Am wichtigsten ist dabei der in **Bild 5** gezeigte und oben links eingeblendete Moduswechsler – er erlaubt die Festlegung der Funktion der Weboberfläche. Zum Hinzufügen neuer Widgets müssen Sie logischerweise im Bearbeitungsmodus sein, der durch das Bleistift-Piktogramm dargestellt ist. Im nächsten Schritt können Sie auf das *Add*-Button klicken, um sich für verschiedene Steuerelemente zu entscheiden. Als Erstes klicken wir einen *Switch* an, der daraufhin in der in **Bild 6** gezeigten Bearbeitungsoberfläche erscheint.

Wichtig ist vor allem die Rubrik *Linked Variable*, in der sich abermals das vom Einrichten der Netzwerk-Zugangskonfiguration bekannte Verbinden-Symbol verbirgt. Klicken Sie es an, um eine Liste aller im Cloud-Konto enthaltenen Things und der in ihnen angelegten Variablen zu aktivieren. Daraus folgt hier, dass wir uns für die Variable `ledIntenBool` entscheiden und sie durch die *Link Variable*-Option verbinden. Danach fehlt noch ein Klick auf *DONE*, um die multimodale Bearbeitungsoberfläche zu schließen und in das Dashboard zurückzukehren. Danach klicken wir auf das Augen-Symbol, um den Schalter zur Aktivierung freizugeben. Das Ein- und Ausschalten des Switch sorgt dafür, dass die rote LED neben der MicroUSB-Buchse aufleuchtet beziehungsweise erlischt.

Um die Helligkeit der blauen Leuchtdiode einstellen zu können, müssen wir den Dashboard-Editor wieder in den Bearbeitungsmodus zurückversetzen und über *Add -> Widgets* abermals ein neues Steuerelement hinzufügen. Der Autor entscheidet sich dieses Mal für den Typ *Slider*. In seiner Bearbeitungsoberfläche legen wir als *Value Rang* den Wert von *0 - 255* fest. Die Verknüpfung erfolgt durch die Variable `letIntenInt`, die ja für die „Helligkeitssteuerung" von RGB-LEDs verantwortlich ist. Zu guter Letzt wechseln wir auch hier in den Aktivierungsmodus und stellen erfreut fest, dass Änderungen des Sliders die Helligkeit der blauen Leuchtdiode beeinflussen.

Eile mit Weile

Zu der Zeit, als der Artikel verfasst wurde, war die Funktion zur Einrichtung von Web-Cronjobs brandneu: Dahinter versteht Arduino die Möglichkeit, „Aufgaben" am IoT-Endgerät programmatisch auszulösen. Zur Vorführung dieser Möglichkeiten wollen wir eine neue Variable vom Typ *Schedule* einrichten. **Bild 7** zeigt die gewünschte Konfiguration. Vorrangiger Lohn unserer Arbeit ist, dass wir im Code nun die folgende neue Variable vorfinden:

```
CloudSchedule tamsSchedule;
```

Interessant ist, dass die Einstellung der Arbeitsdauer über das Dashboard erfolgt - die in den **Bild 8** gezeigte Einstellungsseite verrät mehr über das dazu vorgesehene Steuerelement.

Im folgenden Schritt müssen wir uns darum kümmern, die „lokale" Verarbeitung der in `tamsSchedule` enthaltenen Werte durchzuführen:

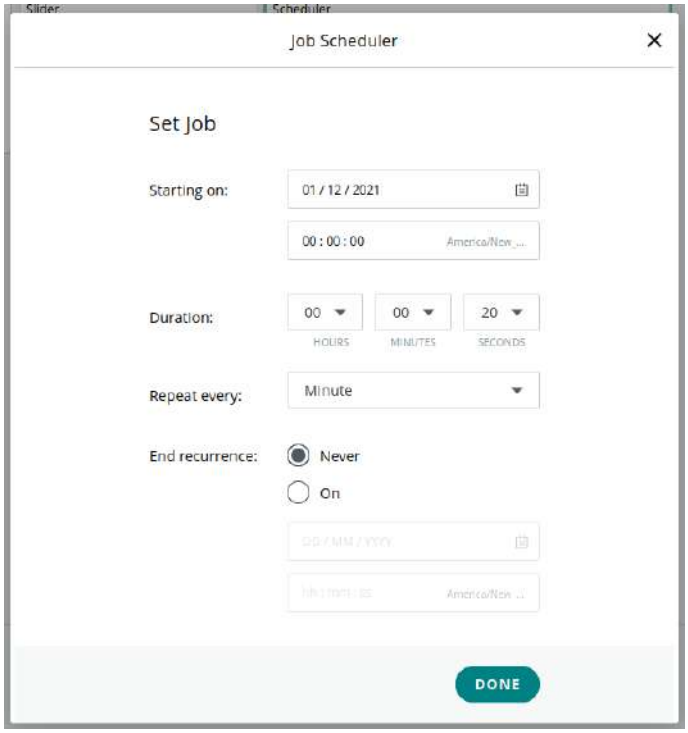


Bild 8. Die Konfiguration des Schedulers erfolgt per Dashboard.

```
void setup() {
...

 pinMode(LED_BUILTIN, OUTPUT);
 pinMode(LEDB, OUTPUT);
 pinMode(LEDR, OUTPUT);
}

void loop() {
 ArduinoCloud.update();
 // Your code here
 if(tamsSchedule.isActive()){
  digitalWrite(LEDR, HIGH);
 }
 else{
  digitalWrite(LEDR, LOW);
 }
}
```

Wichtig ist hier, dass das „Herausschreiben“ der in `tamsSchedule` angelieferten Informationen die ausschließliche Aufgabe des Entwicklers ist. Die Cloud beschränkt sich darauf, den in `tamsSchedule` enthaltenen Wert periodisch anzupassen. Die hier gezeigte permanente Polling-Prozedur in der Methode `loop` mag aus ressourcentechnischer Sicht nicht optimal sein, funktioniert aber problemlos. Schicken Sie das Programm anschließend zum Arduino und erfreuen Sie sich des periodischen Ein- und Ausschaltens der roten Komponente der RGB-LED.

An dieser Stelle konnte sich der Autor einen weiteren kleinen Versuch nicht verkneifen. Er schaltete das „Labor-WLAN“ ab und überprüfte das Systemverhalten. Der Arduino reagierte darauf zuerst mit „chaotischem“ Ausschalten der blauen RGB-Komponente und der LED an Pin 13, um einige Sekunden später komplett neu zu starten.

Zum Zeitpunkt der Drucklegung dieses Artikels war nicht wirklich klar, wie die Arduino-Cloud auf den Verlust der Funkverbindung vom Endgerät zum Server reagiert - nach Ansicht des Autors wäre ein Nachbessern an dieser Stelle wünschenswert.

Fazit

Sofern die zu realisierende Aufgabe nicht extrem sicherheitskritisch ist (Stichwort: Keine Turbinen- oder Heizungssteuerungen!), bietet die Arduino IoT-Cloud schon in der vorliegenden Version eine bequeme und niederschwellige Möglichkeit, Cloud-Backends zu realisieren, ohne sich auf den Kampf mit MQTT und Co. einlassen zu müssen. Nach Ansicht des Autors ein durchaus empfehlenswertes Produkt!

210550-02

Ein Beitrag von

Idee, Illustrationen und Text: **Tam Hanna**
Redaktion: **Rolf Gerstendorf**
Layout: **Harmen Heida**

Sie haben Fragen oder Kommentare?

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

WEBLINKS

[1] Digitaler Zwilling: https://de.wikipedia.org/wiki/Digitaler_Zwilling
[2] Liste kompatibler Boards: https://bit.ly/3t8Vl3W
[3] Arduino Things: https://create.arduino.cc/iot/things

Arduino Shield **für zwei Geiger-Müller-Zählrohre**

Ein hochempfindlicher, sehr stromsparender Strahlungsmesser

Gabriele Gorla (Italien/USA)

Das Geiger-Müller-Zählrohr-Shield verwandelt Ihren Arduino Uno in ein Instrument zur Messung und Aufzeichnung von radioaktiver Strahlung. Es bietet sogar Platz für zwei Rohre, um eine besonders hohe Empfindlichkeit zu erzielen. Das Shield kann mit einem Dragino-Shield für LoRa-Konnektivität kombiniert werden. Sammeln Sie Strahlungsdaten „im Feld“ - und greifen Sie von überall auf der Welt darauf zu!

Wie viele andere Hobbyelektroniker war ich schon immer von Radioaktivität und den Sensoren, mit denen sie nachgewiesen werden kann, fasziniert. Geiger-Müller-Rohre [1] sind eine gängige und relativ kostengünstige Methode zur Messung von Strahlung. Mein Projekt habe ich auf den Namen GRAD getauft. Es ist eine Komplettlösung für die Strahlungsmessung im Formfaktor eines Arduino-Shields. Seine Hauptmerkmale sind eine hohe Empfindlichkeit durch die Verwendung zweier Zählrohre und ein sehr geringer Strombedarf.

Was ist ein Geiger-Müller-Zähler?

Jeder Geiger-Müller-Zähler benötigt vier wesentliche Funktionsblöcke.

- **Geiger-Müller-Rohr**: Das Rohr hat zwei Anschlüsse und ist mit einem Niederdruck-Gasgemisch gefüllt. Wenn es mit der entsprechenden Spannung vorgespannt wird, ionisiert das Gas und leitet kurzzeitig Strom, sobald es von Strahlung getroffen wird. Je nach Rohrtyp ist es möglich, Alpha- und Betateilchen sowie Gammastrahlung nachzuweisen. Das im GRAD verwendete SBM-20 ist besonders empfindlich für Gamma- und hochenergetische Betastrahlung.
- **Hochspannungsnetzteil**: Das verwendete Rohr wird im so genannten Geiger-Müller-Bereich betrieben. Dies ist ein Vorspannungsbereich, in dem die Impulszahl nahezu unabhän-

gig von der Vorspannung ist. Bei herkömmlichen Röhren liegt die Vorspannung zwischen 400 V und 500 V. Die optimale Spannung für die SBM-20 beträgt etwa 400 V.

- **Impulsdetektor**: Die aus dem Rohr kommenden Impulse sind sehr kurz und von unterschiedlicher Spannungsamplitude. Der Impulsdetektor bereitet das Signal so auf, dass es von der folgenden digitalen Stufe leicht gezählt werden kann.
- **Impulszählung**: Die Impulse werden über ein festes Zeitintervall gezählt, um einen CPS- (counts per second) oder CPM-Wert (counts per minute) zu berechnen. Dieser Wert kann mit den im Datenblatt des Rohrs angegebenen Parametern grob in eine Dosisleistung umgerechnet werden. Für die Impulszählung und deren Visualisierung und/oder Protokollierung wird ein Arduino-Board wird verwendet.

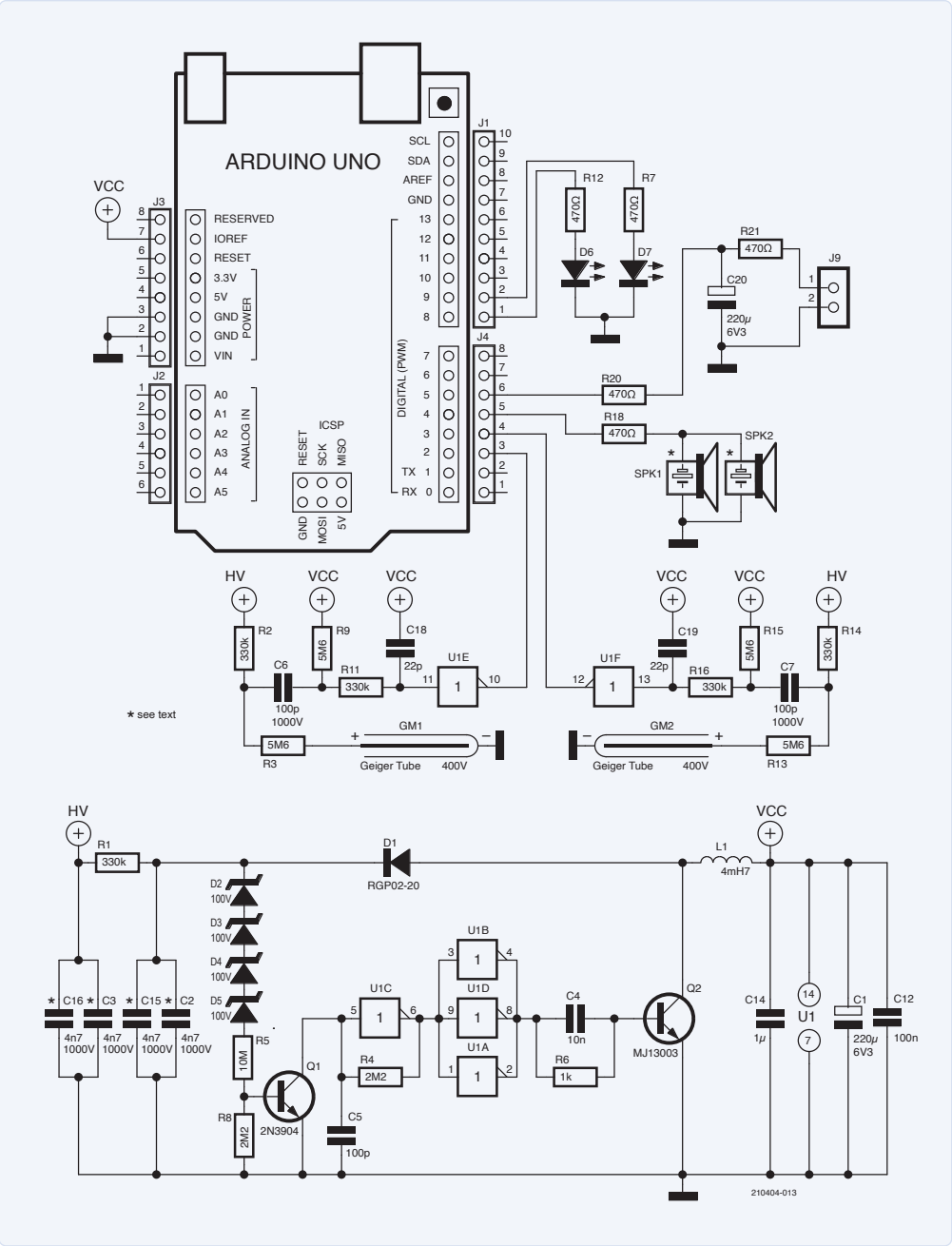


Bild 1. Hochspannend, aber einfach: Die Schaltung des GRAD03-Projekts [3].

Hochspannung

Im Internet gibt es viele Schaltungen, um die hohe Spannung für den Betrieb von Geiger-Müller-Rohren zu erzeugen. Viele davon sind Aufwärtswandler, die mit einem 555-Timer aufgebaut sind, einige mit offenem Regelkreis, andere mit Rückkopplung. Die Entwürfe mit offenem Regelkreis wurden nicht in Betracht gezogen, da sie eine individuelle Abstimmung jedes Geräts erfordern und keine stabile Spannung bei hohen Impulszahlen liefern. Ausführungen mit geschlossenem Regelkreis sind wegen ihrer Stabilität besser geeignet. Um die Gesamtstromaufnahme niedrig zu halten, sollte jedoch der Rückkopplungsschleife besondere Aufmerksamkeit gewidmet werden (12 µA Strom bei 400 V sind etwa 5 mW).

Um die Leistungseinbußen bei der Hochspannungsrückkopplung zu vermeiden, verwenden die elegantesten Lösungen einen nicht isolierten Aufwärtsregler wie den LT3420 von Analog Devices, der die Spannung auf der Primärseite erfasst.

Wir haben stattdessen einen einfachen Aufwärtswandler mit einem sehr geringen Strom-Feedback implementiert. Unser Entwurf in **Bild 1** basiert stark auf den Theremino-Geiger-Adaptern in [2]. Wir haben die dortigen drei Vorschläge (SMD, DIY und „Flintstones“) zu einer Schaltung mit Z-Dioden-Rückkopplung kombiniert, die ausschließlich bedrahtete Bauteile verwendet.

Der Schmitt-Trigger-Inverter U1C bildet mit R4 und C5 einen Oszillator, der die Impulse zur Ansteuerung des Schalttransistors Q2 erzeugt. U1A, U1B und U1D sind parallel geschaltet, um den Basisstrom von Q2 zu erhöhen.

Die Rückkopplung wird mit einer Reihe von Z-Dioden mit sehr geringem Leckstrom (D2…D5) realisiert. Wenn die Ausgangsspannung die resultierende Z-Spannung übersteigt, schaltet Q1 durch und stoppt den Oszillator. Wenn die Ausgangsspannung wieder abfällt, sperrt Q1 wieder und gibt den Oszillator frei. R1 und C3 reduzieren die Welligkeit der Ausgangsspannung. Zusätzliche Vorwiderstände versorgen jede Röhre separat (R2 und R3 für GM1, R13 und R14 für GM2).

Zählung

Die Zählimpulse werden auf der Hochspannungsseite der Rohre über Koppelkondensatoren (C6 und C7) gleichspannungsfrei abgegriffen und von den beiden noch zur Verfügung stehenden Schmitt-Trigger-Invertern „digitaltauglich“ geformt (**Bild 2**).

J1 bis J4 sind die Standard-Arduino-Anschlüsse, von denen das Shield

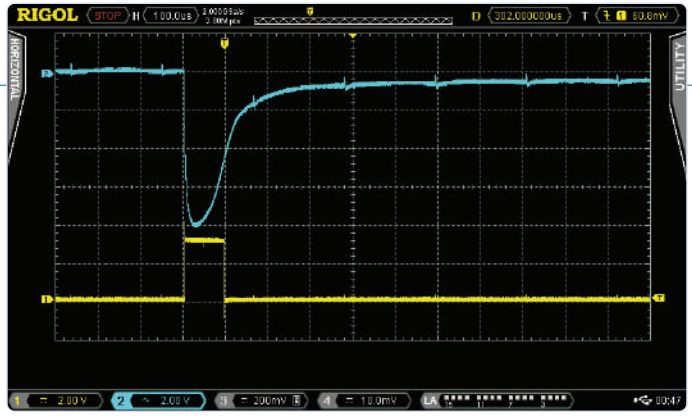


Bild 2. Oszillogramm eines ionisierenden Teilchens, das auf ein Rohr trifft, und der resultierende Impuls.

nur wenige Pins verwendet. Die digitalen Eingänge D2 und D3, an denen die beiden Rohre angeschlossen sind, sind die Interrupt-Pins des Arduino Uno. Über sie kann der Arduino im Hintergrund Impulse zählen, während er andere Aktionen durchführt.
Die digitalen Ausgänge D8 und D9 des Arduino sind mit einem LED-Paar verbunden, um anzuzeigen, wenn ionisierende Strahlung auf das jeweilige Rohr trifft. Pin D4 ist mit dem Piezo-Lautsprecher SPK1 verbunden, um eine akustische Rückmeldung zu erhalten.
Schließlich gibt es noch einen digitalen Ausgang D5/J9 mit einem Tiefpassfilter (R20, R21 und C20), an den ein einfaches analoges 10-mA-Panelmeter angeschlossen werden kann.

Platinenentwurf

Der Schaltplan und die Platine (**Bild 3**) wurden in KiCad entworfen. Die Designdateien stehen auf der Elektor-Labs-Seite [3] dieses Projekts im Bereich *Elements* zum Download bereit. Die Gerber- und Bohrdateien der Leiterplatte sind ebenfalls dort zu finden. Sie können diese Dateien verwenden, um eine Platine bei Ihrem bevorzugten PCB-Lieferanten zu bestellen. Zu guter Letzt finden Sie dort eine Excel-Tabelle mit einer sehr detaillierten Stückliste, die vor allem die kritischeren (Hochspannungs-!) Bauteile dokumentiert, die für dieses Projekt benötigt werden, einschließlich Angaben zu Herstellern, Typennummern und sogar Bestellcodes.
Bild 4 zeigt die bestückte Leiterplatte mit eingebauten SBM-20-Rohren. Die Bauteilpaare SPK1/SPK2, C3/C16 und C2/C15 sind jeweils nur alternative Fußabdrücke auf der Platine; es wird also nur SPK1 oder SPK, C3 oder C16 und C2 oder C15 installiert. J5 und J6 auf der Platine bezeichnen zwei zusätzliche Anschlussflächen für die Minuspole kürzerer Geiger-Müller-Rohre.

Zählrohroptionen

Die Platine ist für den Betrieb mit den russischen 105 mm langen Zählrohren SBM-20, STS-5, der chinesischen J305 oder der 90-mm-Rohre J305 und M4011 ausgelegt. Jedes andere 400-V-Rohr funktioniert ebenfalls, wenn es mechanisch anschließbar ist. In allen Fällen muss der Anodenanschluss so kurz wie möglich gehalten werden, um die parasitäre Kapazität zu minimieren. Bei Zählrohren, die andere Spannungen benötigen, sollte(n) die Zenerdiode(n) ausgetauscht werden, um die erforderliche Spannung zu erhalten. Jede Diode mit einem Leckstrom von 0,5 µA oder weniger sollte funktionieren.

Leistung

Die Stromversorgung ist auf 65 µA bei 5 V, also 325 µW ausgelegt. Bei batteriebetriebenen Geräten kann die Schaltung auch mit 3,3 V versorgt werden, so dass die Leistungsaufnahme auf nur 150 µW (45 µA) sinkt.

STÜCKLISTE

Widerstände:
R1,R2,R11,R14,R16 = 330 k
R3,R9,R13,R15 = 5M6
R4,R8 = 2M2
R5 = 10 M
R6 = 1 k
R7,R12,R18,R20,R21 = 470 Ω

Kondensatoren:
C1,C20 = 220 µ, 6,3 V
C2,C3 = 4700 p, 1000 V
C4 = 10 n, 6,3 V
C5 = 100 p, 50 V
C6,C7 = 100 p, 1000 V
C12 = 100 n, 50 V
C14 = 1 µ, 6,3 V
C15,C16 = 4700 p, 1000 V
C18,C19 = 22 p, 50 V

Induktivität:
L1 = 4m7

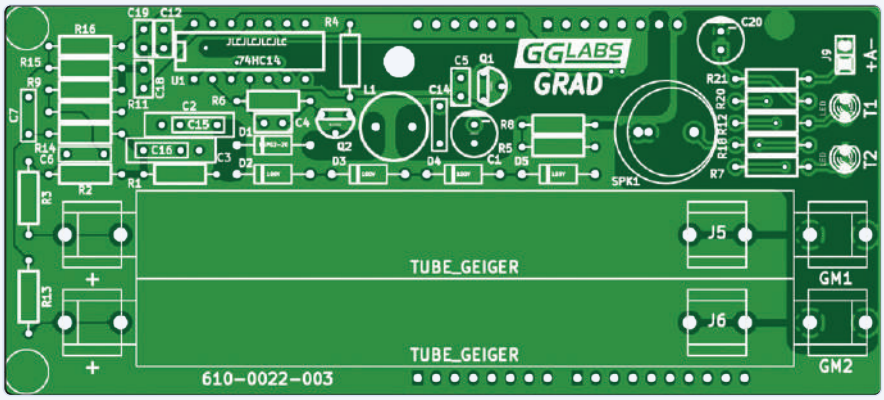


Bild 3. Das Platinenlayout des GRAD03-Shields.

Halbleiter:
D1 = Diode 1 A, 800 V
D2,D3,D4,D5 = Z-Diode 100 V, 1,5 W
D6, D7 = rote LED, 3 mm (T1, T2 auf der Platine)
Q1 = 2N3904
Q2 = MJ13003
U1 = 74HC14

Außerdem:
GM1,GM2 = Geiger-Müller-Zählrohr, z.B. SBM-20
J1,J2,J3,J4 = Anschlussleisten-Satz für Arduino-Uno-Shield
SPK1 = AC-Piezo-Summer, z.B. AC-1205G-N1LF

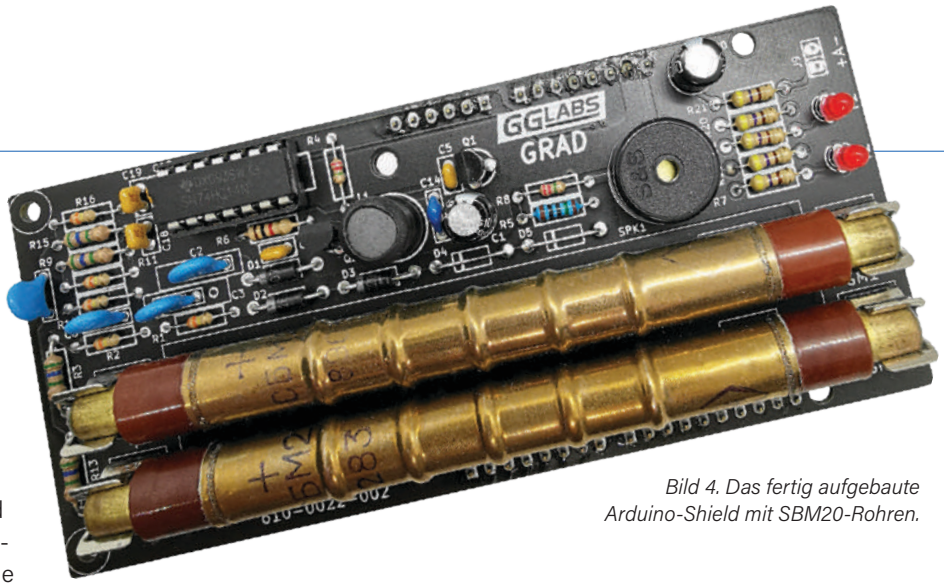


Bild 4. Das fertig aufgebaute Arduino-Shield mit SBM20-Rohren.

Arduino Beispielcode

Ein einfacher Arduino-Sketch zur Ansteuerung des Zählers ist unter *Project Elements* auf der Elektor Labs-Seite verfügbar. Alle 60 Sekunden gibt die Software über die serielle Schnittstelle eine kommagetrennte Zeile von Werten (CSV) aus mit: einer fortlaufenden Nummer, dem Rohwert für jedes Rohr, einem gleitenden Mittelwert von `cnt1 + cnt2` und einer auf Grundlage dieses Mittelwerts berechneten Dosisleistung in µSv/h. Das folgende Datenfragment zeigt den Beginn einer Messung der Hintergrundstrahlung in Santa Clara mit zwei SBM-20-Rohren auf dem GRAD-Shield. In Kalifornien strahlt also nicht nur die Sonne!

```
Seq , cnt1 , cnt2 , avg10 , µSv/h
1   , 14   , 19   , 33.0  , 0.075
2   , 10   , 17   , 32.3  , 0.073
3   , 16   , 12   , 31.8  , 0.072
4   , 16   , 24   , 32.5  , 0.074
5   , 13   , 11   , 31.6  , 0.072
```

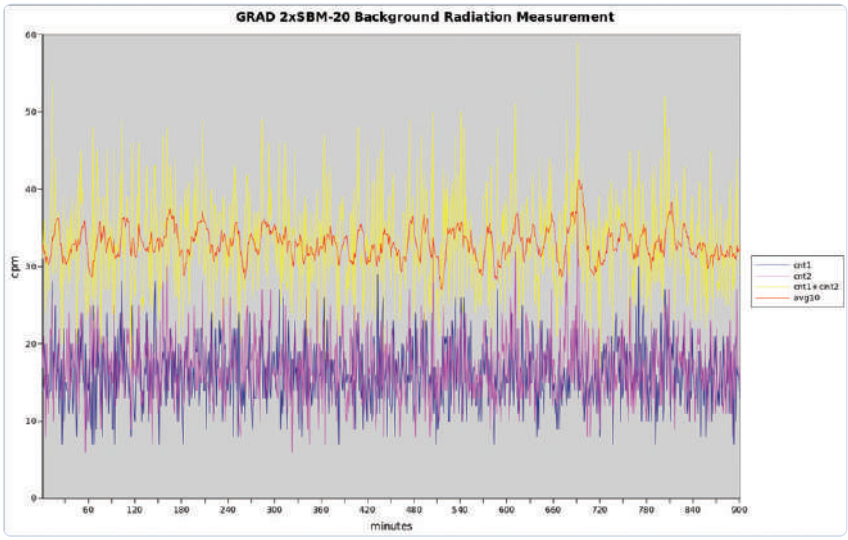


Bild 5. Hintergrundstrahlung im kalifornischen Santa Clara.

Die Grafik in **Bild 5** zeigt die gesamte Messung über einen längeren Zeitraum.
Am Anfang des Sketches gibt es einige Definitionen, die entsprechend der Boardkonfiguration und/oder den Benutzerwünschen eingestellt werden können. Der erste Satz konfiguriert die Parameter der Geiger-Müller-Rohre und die Größe des Fensters für die Messwertausgabe.

```
#define CPM2USV 220  // tube CPM to
   µSv/h conversion
                     // factor
#define TUBES 2      // number of tubes installed
#define WSIZE 10     // moving average window (in
                     // minutes)
```

Der Parameter `CPM2USV` ist die Zählung pro Minute für 1 µSv/h für das eingesetzte Zählrohr. Leider ist in den veröffentlichten Datenblättern keine „richtige" Zahl für diesen Parameter angegeben. Begeisterte Geiger-Müller-Bastler im Netz verwenden für die SBM-20-Rohre Werte zwischen 130 und 220 (Umrechnungsfaktor von 0,0075...0,0045).
Der Parameter `TUBES` gibt, wie der Name schon sagt, die Anzahl der installierten Zählrohre an. Gültige Werte sind 1 und 2. Der Parameter `WSIZE` schließlich bestimmt die Größe des Fensters, in dem der gleitende Mittelwert der Zählungen dargestellt wird. Der Standardwert von 10 bedeutet ein 10-Minuten-Fenster.
Der zweite Satz von Parametern legt fest, wie die optionalen Hardware-Funktionen angeschlossen werden:

```
#define LED1_PIN     8  // pin for TUBE1 LED
#define LED2_PIN     9  // pin for TUBE2 LED
#define SPKR_PIN     4  // pin for speaker connection
#define LED_BLINK_MS 20 // duration of LED blink for
                        // each count
```

Alle drei `_PIN`-Definitionen wählen den Arduino-Pin aus, an dem die LED oder der Lautsprecher angeschlossen ist. `LED_BLINK_MS` definiert die Zeit in Millisekunden, die die LED für jeden Zählimpuls aufleuchtet.

LoRaWAN-Vernetzung

Der grundlegende Sketch erfordert, dass das Gerät über USB mit einem Computer verbunden ist. Oft aber wünscht man sich, den Sensor weit ab vom Computer zu platzieren, so dass keine USB-Verbindung möglich ist.
Auf den Arduino Uno und das GRAD-Shield kann ein Dragino-LoRa-Shield aufgesteckt werden, um eine Verbindung zum LoRaWAN zu erreichen. Leider verwendet das Dragino-Schild auch den Arduino-Pin D2, so dass es zwei Optionen gibt:
Option mit einem Zählrohr: In diesem Fall kann nur das Rohr GM2 verwendet werden. Pin 10 des Schmitt-Triggers U1 muss aufgetrennt werden, um Störungen der LoRa-Kommunikation zu vermeiden. Die Fotos in **Bild 6** zeigen den vollständigen Stapel mit einer einzigen russischen SBM-19.
Option mit zwei Zählrohren: In diesem Fall müssen sowohl das Dragino-Shield als auch die GRAD-Platine modifiziert werden. Beim Dragino-Shield müssen R5 und J_DIO0 entfernt und die Verbindung

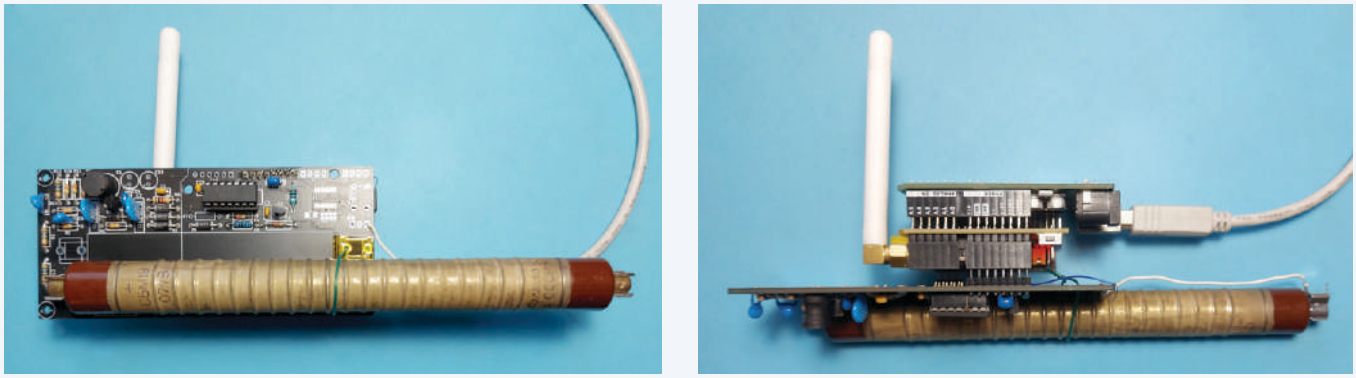


Bild 6. GRAD03-Platine, Arduino Uno und das Dragino-LoRa-Shield kombiniert.

zwischen Funk-Pin DIO0 und Arduino-Pin D7 mit einem Draht realisiert werden.

Bei der GRAD-Platine müssen R7, R12, C20 und die beiden LEDs entfernt werden. R21 sollte durch einen Draht ersetzt und eine einzelne LED an J9 (ersetzt den analogen Zählerausgang) angebracht werden.

Ein Sketch, der mit *The Things Network* (TTN) kommuniziert und alle 60 Sekunden die Zählwerte der Rohre anzeigt, steht ebenfalls zum Download bereit. Es verwendet die Arduino-Bibliothek LMIC, um das Dragino-LoRa-Shield zu steuern und verbindet sich mit dem TTN mittels *Over The Air Activation* (OTAA). Zur Anzeige der Daten genügt ein einfacher Node-RED-Flow, wie in **Bild 7** zu sehen.

Messung der Strahlung

Um die korrekte Funktion des Geigerzählers zu überprüfen, wird eine Strahlungsquelle benötigt. So etwas kann man online bei speziellen Anbietern erwerben, alternativ aber bestimmte alte Schätzchen in einem Secondhandladen oder bei Ebay erstehen, die kleine Mengen radioaktiver Materialien enthalten: Uranglas aus Böhmen, einen alten (und mittlerweile in der EU verbotenen) Thorium-Glühstrumpf, bestimmtes farbig glasiertes Keramikgeschirr aus China oder den USA.

In Ermangelung solcher Gegenstände kann man auch einfach die Zerfallsprodukte von Radon untersuchen, die vom Luftfilter einer Klimaanlage oder eines Luftreinigers aufgefangen werden (**Bild 8**). Diese haben eine relativ kurze Halbwertszeit (in der Größenordnung von einigen zehn Minuten). Lassen Sie die Klimaanlage ein paar Stunden lang laufen und messen dann sofort am Filter. Wenn das Filter fein genug ist, dürfte er Strahlung mit einer Rate emittieren, die um ein Vielfaches höher ist als die lokale Hintergrundstrahlung. Die Grafik zeigt,

PASSENDE PRODUKTE

- **Arduino Uno SMD Rev3 SKU 19938**
  www.elektor.de/19938
- **MightyOhm Geigerzähler-Kit (inkl. Gehäuse) SKU 18509**
  www.elektor.de/18509

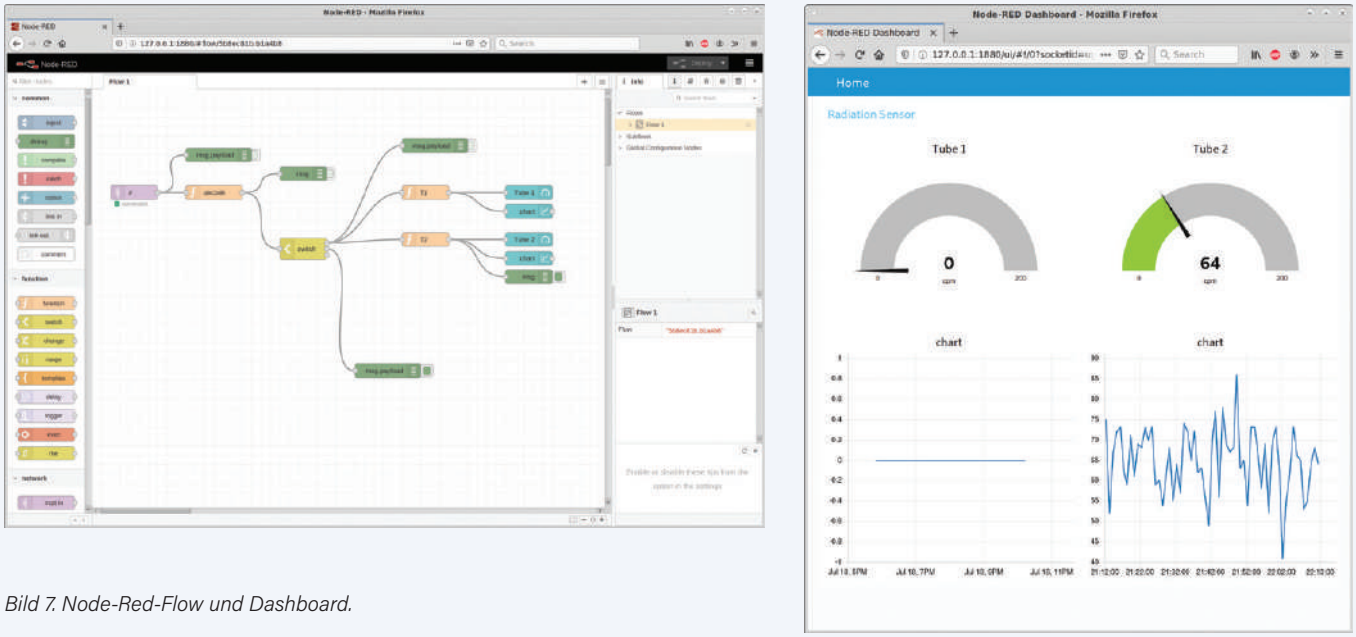


Bild 7. Node-Red-Flow und Dashboard.

dass die Strahlung fast das Zehnfache der Hintergrundstrahlung erreicht und dann der für den radioaktiven Zerfall typischen Exponentialkurve folgt.

210404-02

Dieser Artikel basiert auf dem Material, das auf der Elektor Labs-Seite dieses Projekts [3] vorgestellt wird. Dort finden Sie alle Downloads für GRAD03 sowie Diskussionen und Anmerkungen zu diesem Thema.

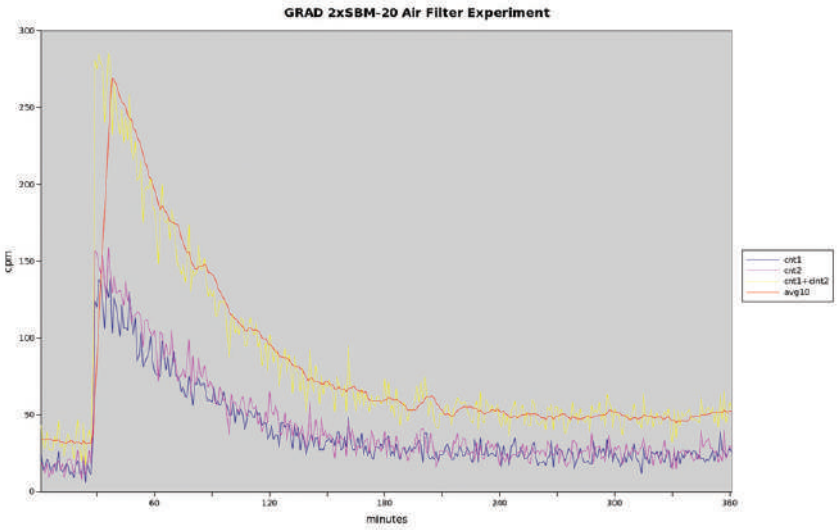


Bild 8. Durch Zerfall von Radon verursachte Strahlung aus einem Klimaanlagenfilter.

Ein Beitrag von

Entwurf: **Gabriele Gorla**
Text: **Gabriele Gorla, Luc Lemmens**
Illustrationen: **Gabriele Gorla, Patrick Wielders**
Redaktion: **Jens Nickel, C. J. Abate**
Übersetzung: **Rolf Gerstendorf**
Layout: **Giel Dols**

Haben Sie Fragen oder Kommentare?

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

WEBLINKS

[1] Geiger-Müller-Zählrohr: https://de.wikipedia.org/wiki/Z%C3%A4hlrohr
[2] Theremino Geiger-Adapter (siehe die drei letzten Entwürfe auf dieser Seite): www.theremino.com/de/technical/schematics
[3] Dieses Projekt auf Elektor Labs: https://bit.ly/3uZlqmU

MonkMakes: Luftqualitäts-Messgerätekit für den Raspberry Pi

Misst Temperatur und CO_2e

Von Luc Lemmens (Elektor)

Weil die meisten von uns heutzutage viel in (privaten) Zimmern sitzen, werden Module zur kostengünstigen Messung der Luftqualität immer beliebter. Das *MonkMakes Air Quality Kit* misst den CO_2e-Gehalt und die Temperatur. Es ist speziell für den Raspberry Pi 400 konzipiert, kann aber dank der mitgelieferten Jumper-Drähte und einer GPIO-Schablone auch an andere Raspberry-Pi-Modelle angeschlossen werden.

Bekanntermaßen misst man mit Thermometern die Raumtemperatur, aber in den letzten Jahren sind CO_2(e)-Messgeräte zur Überwachung der Luftqualität immer beliebter geworden. Zu viel Kohlendioxid (CO_2) wirkt sich negativ auf die Konzentrationsfähigkeit aus, und bei noch höheren Werten ist es sogar gesundheitsschädlich. Dieses Kit misst die Qualität der Raumluft sowie die Temperatur. Es ist vorrangig für den Raspberry Pi gedacht, kann aber auch als eigenständiges Gerät eingesetzt werden. Das Board verfügt über einen Summer und eine Leiste mit sechs LEDs (zwei grüne, zwei orange und zwei rote), die die Luftqualität anzeigen. Die Messwerte für Temperatur und Luftqualität können von einem Raspberry Pi verarbeitet werden, der ebenfalls den Summer und die LED-Anzeige ansteuern kann.

Der Bausatz wird ohne gedruckte Dokumentation geliefert, aber er enthält einen Link zur MonkMakes-Website, wo das Datenblatt und die Anleitung heruntergeladen werden können [1]. Diese Dokumente enthalten alle relevanten Informationen und helfen dem Benutzer, das Board anzuschließen und zu verwenden. Beispielanwendungen in Python stehen auf Github [2] zum Download bereit.

Die Hardware

Neben der Anzeige mit sechs LEDs und dem großen quadratischen Summer in der Mitte der Platine in **Bild 1** enthält die Platine eine Power-LED, einen Temperatursensor, einen eCO2-Sensor, einen Mikrocontroller und natürlich einen 40-poligen Steckverbinder, der direkt auf den Erweiterungsstecker eines Raspberry Pi 400 passt (**Bild 2**). Vielleicht überflüssig zu sagen: Andere Raspberry-Pi-Boards können nicht direkt angeschlossen werden, dafür sind Jumper-Drähte im Bausatz enthalten. Die vier benötigten Anschlüsse (zwei für die Stromversorgung, zwei für die serielle Verbindung) sind im Aufdruck auf der MonkMakes-Platine und auf der mitgelieferten Schablone so dargestellt, dass sie zu den entsprechenden Pins des GPIO-Anschlusses des Raspberry Pi

passen, wie in **Bild 3** dargestellt. Die Power-LED leuchtet auf, sobald die 3,3-V-Versorgungsspannung eingeschaltet wird, ebenso wie eine der CO_2e-Pegel-LEDs.
Der Temperatursensor TMP235 stammt von Texas Instruments [3]. Seine Ausgangsspannung ist proportional zur Temperatur. Für die CO_2-Messung verwendet das MonkMakes-Board einen TVOC-Sensor (Total Volatile Organic Compounds) des Typs CCS811 [4], der nicht direkt den CO_2-Gehalt misst, sondern den Gehalt einer Gruppe von Gasen, die als flüchtige organische Verbindungen (VOCs) bezeichnet werden. In Innenräumen steigt der Gehalt dieser Gase in einer Kurve an, die mit der von CO_2 vergleichbar ist, so dass die Sensorwerte auch zur Abschätzung des CO_2-Gehalts verwendet werden kann. Dies wird als CO_2-Äquivalent oder CO_2e bezeichnet.
Der eingebaute ATtiny1614-Mikrocontroller liest beide Sensoren aus und steuert die LED-Balkenanzeige und den Summer. Über ein serielles Protokoll kann ein Host-System wie der Raspberry Pi die Sensorwerte abfragen oder die LEDs und den Summer ein- und ausschalten. Das Datenblatt des Kits dokumentiert das einfache Protokoll, so dass es nicht allzu schwierig sein dürfte, eine eigene Software zur Unterstützung des Air Quality Kits zu schreiben. Wie der Name des Kits schon sagt, ist es für den Raspberry Pi konzipiert, aber es gibt keinen Grund, warum man es nicht mit anderen Boards oder Systemen mit einem 3,3-V-UART verwenden könnte.
Die Firmware des ATtiny bietet auch einen automatischen Modus, der standardmäßig eingeschaltet ist. In diesem Modus wird der CO_2e-Pegel ohne eine externe Steuerung auf der LED-Leiste anzeigt. Alles, was in diesem Modus dann zusätzlich zum Kit benötigt wird, ist eine 3,3-V-Stromversorgung. So kann das Air Quality Kit auch ohne Host-System als CO_2e-Monitor verwendet werden.

Software

Wie bereits erwähnt, stehen auf der Github-Seite von MonkMakes einige Python-Beispielprogramme zur Verfügung, mit denen das Air Quality Kit gesteuert werden kann, um alle seine Funktionen zu testen und zu demonstrieren. Im Abschnitt *Getting Started* der Dokumentation zeigen die Anweisungen deutlich, wie die Software auf einem Raspberry-Pi-Board verwendet werden kann, um das Kit in ein CO_2e-Messgerät, in ein CO_2e-Messgerät mit akustischem Alarm (**Bild 4**) und in einen Datenlogger zu verwandeln. Wenn man sich die Python-Beispiele (**Bild 5**) ansieht, wird man feststellen, dass der ATtiny und die API das Abrufen und die Auswertung der Sensordaten vollständig übernehmen. Ein einfacher Befehl vom Host (Raspberry Pi) veranlasst das Air Quality Board, die aktuelle Umgebungstemperatur (in °C) beziehungsweise den CO_2e-Gehalt (in ppm) auszugeben. Ähnliche Befehle werden verwendet, um den Summer ein- und auszuschalten und die LEDs der CO_2e-Balkenanzeige zu steuern.

Ein schönes Design

Um dieses Luftqualitäts-Kit in Betrieb zu nehmen, sind nur einige wenige Grundkenntnisse über den Raspberry Pi erforderlich. Was für die einen ein großer Vorteil ist, dürfte für die anderen weniger attraktiv sein: Kenntnisse über die Sensoren und die Steuerung von Buzzer und LEDs sind nicht erforderlich. Zudem ist der Quellcode der Firmware des ATtiny1614 nicht veröffentlicht, was heißt, dass wir nicht

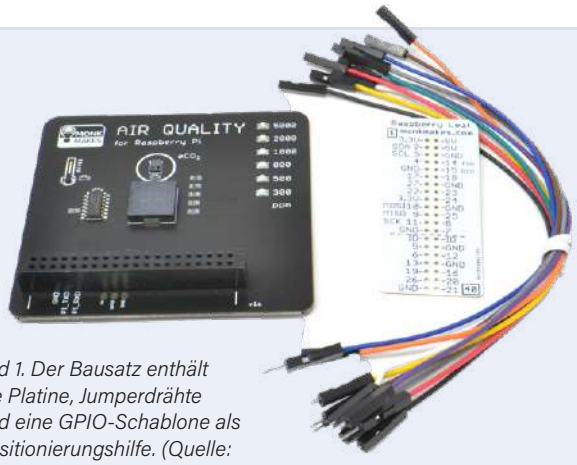


Bild 1. Der Bausatz enthält die Platine, Jumperdrähte und eine GPIO-Schablone als Positionierungshilfe. (Quelle: MonkMakes)

Bild 2. Air Quality Kit, angeschlossen an einen Raspberry Pi 400. (Quelle: MonkMakes)

Bild 3. Bei anderen Raspberry-Pi-Modellen als dem 400 wird die Platine nicht aufgesteckt, sondern mit Jumper-Drähten angeschlossen. (Quelle: MonkMakes)

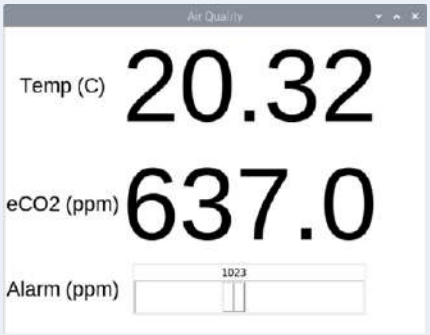


Bild 4. Raspberry-Pi-Displayausgabe für eines der Beispielprogramme. (Quelle: MonkMakes)

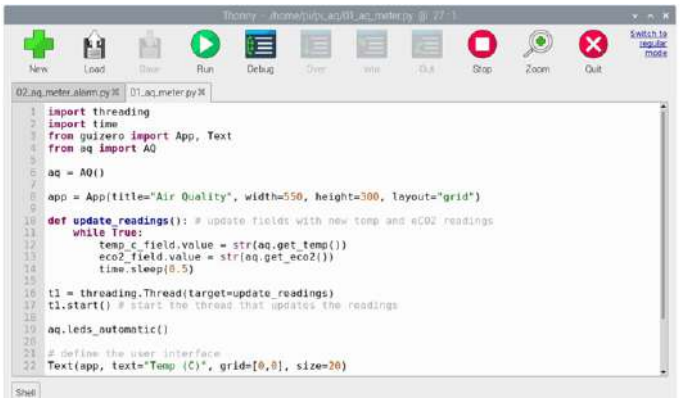


Bild 5. Der Python-Quelltext zeigt, dass nur einfache, kurze Anweisungen erforderlich sind, um mit dem Air Quality Kit zu kommunizieren.

wissen, was genau in diesem Mikrocontroller passiert. Das Protokoll zur Kommunikation mit dem Board ist jedoch einfach und gut dokumentiert und die Entwicklung eigener Anwendungen - auch für andere Zielsysteme als Raspberry-Pi-Boards – dürfte relativ einfach sein. Das Air Quality Kit für den Raspberry Pi von MonkMakes ist ein schön gestaltetes, gut dokumentiertes Board, das zusammen mit den Beispielen auch für Anfänger geeignet ist, die sich mit Luftqualitätsmessungen beschäftigen wollen.

210681-02

Haben Sie Fragen oder Kommentare?

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

Ein Beitrag von

Text: **Luc Lemmens**
Illustrationen: **MonkMakes, Luc Lemmens**
Redaktion: **Jens Nickel, C. J. Abate**
Übersetzung: **Rolf Gerstendorf**
Layout: **Harmen Heida**

PASSENDE PRODUKTE

- **MonkMakes Air Quality Kit for Raspberry Pi (SKU 19913)**
  www.elektor.de/19913
- **Raspberry Pi 400 – Raspberry Pi 4-based PC (US) + FREE GPIO Header (SKU 19429)**
  www.elektor.de/19429
- **Raspberry Pi 4 B (1 GB RAM) (SKU 18966)**
  www.elektor.de/18966

Eigenschaften des Sensors

CO_2e-Mindestwert	400	ppm
CO_2e-Höchstwert	4095	ppm
CO_2e-Auflösung	1	ppm
CO_2e-Fehler	nicht angegeben	
Temperatur minimaler Messwert	–10	°C
Temperatur Höchstwert	100	°C
Temperatur Fehler	±2	°C

Über CO_2-Konzentrationen

Der CO_2-Gehalt in der Luft, die wir einatmen, hat einen direkten Einfluss auf unser Wohlbefinden und ist aus Sicht der öffentlichen Gesundheit von besonderem Interesse. Einfach ausgedrückt ist er ein Maß dafür, wie stark wir die Luft anderer Menschen einatmen. Wir Menschen atmen CO_2 aus, und wenn sich mehrere Personen in einem schlecht belüfteten Raum aufhalten, steigt der CO_2-Gehalt allmählich an, ebenso wie die Konzentration von Aerosolen, die Erkältungen, Grippe und Coronaviren verbreiten. Eine weitere wichtige Auswirkung des CO_2-Gehalts ist die kognitive Funktion, das heißt, wie gut man sich konzentrieren und nachdenken kann.

Die nachstehende Tabelle zeigt, bei welchen Werten CO_2 ungesund werden kann. Die CO_2-Werte sind in ppm (parts per million) angegeben.

CO_2 -Gehalt

250…400	Normale Konzentration in der Umgebungsluft
400…1000	Konzentration, die für bewohnte Innenräume mit gutem Luftaustausch typisch ist
1000…2000	Beschwerden über Schläfrigkeit und schlechte Luft
2000…5000	Kopfschmerzen, Schläfrigkeit und das Gefühl von abgestandener und stickiger Luft, Konzentrationsschwäche, Aufmerksamkeitsverlust, erhöhte Herzfrequenz und leichte Übelkeit können ebenfalls auftreten
5000	In den meisten Ländern Grenzwert für die gemittelte Exposition am Arbeitsplatz
>40.000	Exposition kann zu ernsthaftem Sauerstoffmangel führen, der dauerhafte Hirnschäden, Koma und sogar Tod zur Folge haben kann

WEBLINKS

[1] MonkMakes-Webseite mit Anleitungen: http://monkmakes.com/pi_aq
[2] Software auf Github: https://github.com/monkmakes/pi_aq
[3] TMP235-Datenblatt: www.ti.com/product/TMP235
[4] CCS811-Datenblatt: www.sciosense.com/products/environmental-sensors/ccs811-gas-sensor-solution/

Development Tools alle an einem Ort

Tausende Tools von hunderten zuverlässigen Herstellern

Wählen Sie Ihr Produkt aus unserer breiten Palette auf **mouser.de/dev-tools**

MOUSER ELECTRONICS

Lichtschalter DeLux

Eine Lösung für hochpräzises lichtgesteuertes Schalten

Von Clemens Valens (Elektor)

Lichtgesteuerte Schalter gibt es in Hülle und Fülle und sie kosten im Einzelhandel um die 10 €, aber die meisten von ihnen ändern ihren Zustand nur „irgendwann“ in der Dämmerung. Doch manchmal benötigt man eine höhere Präzision und mehr Kontrolle, als diese billigen Schalter bieten können. Benötigen Sie auch solch eine lux-genaue Lichtsteuerung? Wenn ja, dann ist dieses Projekt genau das Richtige für Sie!

Es gibt viele, viele Entwürfe für lichtgesteuerte Schalter, und die meisten von ihnen funktionieren hervorragend für die Aufgaben, für die sie entworfen wurden. Diese bestehen in der Regel darin, eine Lampe einzuschalten, wenn die Umgebungshelligkeit unter einen bestimmten Schwellenwert fällt, und das Licht auszuschalten, wenn es wieder hell wird. Manchmal ist auch eine Zeitschaltuhr hinzugefügt.

Man könnte meinen, dass diese Konstruktionen alle möglichen Anwendungen abdecken, doch das stimmt nicht. Der Grund dafür ist die mangelnde Präzision, denn sie basieren auf einem LDR oder einem Fototransistor und schalten in der Regel irgendwann in der Abenddämmerung ein und in der Morgendämmerung aus. Die Umgebungshelligkeit variiert in der Realität jedoch viel stärker und viel unregelmäßiger.

Helligkeit ist subjektiv

Für den Menschen wirkt die Helligkeit des Tageslichts oder, wie es fachlich korrekt genannt wird, die Beleuchtungsstärke ziemlich konstant. Natürlich nehmen wir Schwankungen aufgrund von Wolken und Sonne wahr, aber wir sind nicht sehr empfindlich dafür. Der Grund dafür ist die logarithmische Reaktion des Auges auf Helligkeit. An einem bewölkten Tag kann die Helligkeit zwischen 5.000 lx und 10.000 Lux schwanken, und doch sieht sie für uns fast gleich aus. Direktes Sonnenlicht erreicht Werte von über 25.000 lx, was wir natürlich als besonders hell wahrnehmen, aber wir empfinden es nicht als dreimal oder mehr so hell.

Pflanzen hingegen reagieren viel empfindlicher auf die Beleuchtungsstärke als Menschen, ist doch Licht ihr Lebenselixier. Landwirte wissen das und bestrahlen besonders lichthungrige Pflanzen auch tagsüber mit künstlichem Licht, um den Ertrag zu steigern. An sonnigen Tagen ist dies in der Regel nicht nötig, aber an bewölkten Tagen kann es helfen. Wenn dies schon nicht besonders ökologisch ist, so benötigen Landwirte exakte lichtgesteuerte Schalter, die Unterschiede in der Beleuchtungsstärke mit größerer Präzision erkennen können als ein LDR oder ein Fototransistor, damit wenigstens die Ökonomie stimmt.

Heutzutage gibt es Lichtsensoren, die die Beleuchtungsstärke direkt in einen Wert in Lux umwandeln, und dies mit einer Auflösung von bis zu 16 Bit. Einige dieser Sensoren messen nicht nur sichtbares Licht, sondern auch UV-A und UV-B. Mit einem solchen Sensor ist es recht einfach, einen hochpräzisen lichtgesteuerten Schalter zu bauen.

Hochpräziser Umgebungslichtsensor

Ein beliebter Lichtsensor ist der VEML7700 von Vishay. Dabei handelt es sich um einen hochpräzisen Umgebungslichtsensor mit I²C-Schnittstelle, der auf einem kleinen Modul für ein paar Euro zu haben ist. Aus der gleichen Familie stammt auch der VEML6075, ein UV-A- und UV-B-Lichtsensor, ebenfalls mit I²C-Schnittstelle.

Dank seiner digitalen I²C-Schnittstelle benötigt der Sensor keinen externen Analog-Digital-Wandler und kann stattdessen direkt an die meisten Mikrocontroller angeschlossen werden. Seine Messdaten stehen in zwei 16-Bit-Registern zur Verfügung, eines für Umgebungslicht (ambient light, AL) und eines für Weißes Licht. Das Umgebungslicht ist das vom Menschen wahrgenommene Licht im spektralen Bereich von etwa 450...650 nm, Weißes Licht deckt ein breiteres Spektrum von 250...950 nm ab, umfasst also auch den

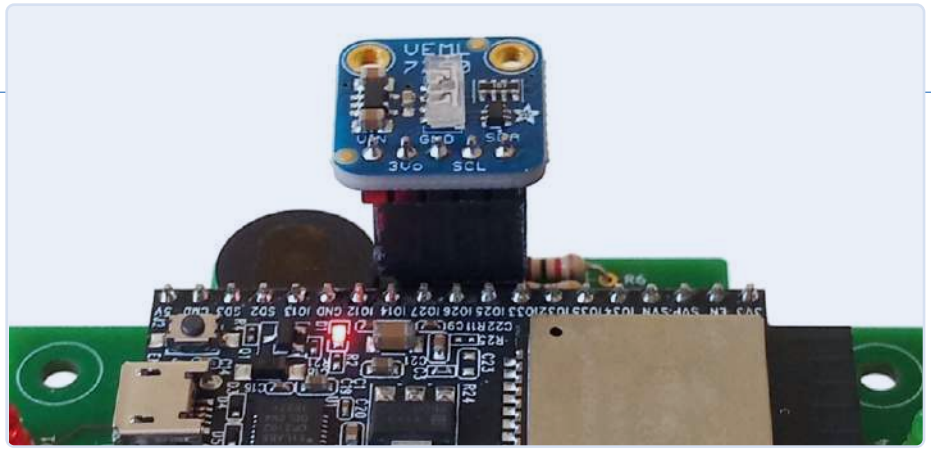


Bild 1. Das 5-polige Modul VEML7700 wird auf den 4-poligen Steckverbinder K9 gesteckt. Der VIN-Pin ist nicht angeschlossen, auch wenn es so aussehen mag, sondern hängt links in der Luft!

gesamten ultravioletten und den nahen Infrarot-Bereich. Pflanzen sind keine Menschen, daher hängt das zu verwendende Ausgangsformat von der jeweiligen Anwendung ab.

Auch die Sensorempfindlichkeit ist wichtig. Der VEML7700 besitzt eine Auflösung von 0,005 lx pro LSB und misst maximal 167.000 lx (Minimum ist 0,01 lx). Für einen solch großen Bereich bei gegebener Auflösung wären 25-bit-Werte erforderlich. Da aber der Sensorspeicher auf 16 Bit beschränkt ist, können Empfindlichkeitsbereiche (gain) angegeben werden, um die 16-bit-Werte in den richtigen Bereich zu bringen. Dank seiner hohen Empfindlichkeit kann der Sensor auch hinter Oberflächen mit geringer Lichtdurchlässigkeit (das heißt, dunklen Oberflächen) eingesetzt werden und dennoch brauchbare Ergebnisse liefern. Für schwache Lichtverhältnisse bietet der Sensor eine Integrationszeit von bis zu 800 ms. Schließlich können niedrige und hohe Schwellenwerte eingestellt werden, die Interrupts auslösen, was die Erstellung von Alarmen oder automatisches Schalten erleichtert.

Einsatz am ESP32-Thermostat

Das für meine Experimente verwendete VEML7700-Modul habe ich bei Adafruit gekauft. Eine gute Plattform für das Modul ist das ESP32-verbundene Thermostat [1] (auch bekannt als Automator, siehe [2]). Das Modul hat fünf Anschlüsse, aber da es zwei Stromversorgungspins gibt, brauchen wir nur vier davon. Diese vier Pins haben die gleiche Reihenfolge wie der Anschluss des OLED-Displays am Thermostat, so dass wir das Modul auf K9 stecken können. Achten Sie darauf, dass der VIN-Pin ganz links nicht angeschlossen ist und das Modul nach oben zeigt (im Gegensatz zum OLED-Display, siehe **Bild 1**).

Software mit ESPHome

Nachdem der Sensor mit dem ESP32 verbunden ist, müssen wir eine Software erstellen, damit alles funktioniert. Da das Ziel eine Art lichtgesteuerter Schalter ist, wäre es eine logische Wahl, eine Heimautomatisierungsplattform zu verwenden, und mein Favorit ist bekanntermaßen ESPHome. Elektor hat mehrere Projekte veröffentlicht, bei denen ESPHome zum Einsatz kam [3][4], so dass Sie vielleicht schon wissen, wie man es benutzt und konfiguriert, aber dieses Projekt führt ein Konzept ein, das ich bisher nicht behandelt habe: die Erstellung eines benutzerdefinierten Sensors. Wenn Sie mit ESPHome noch nicht vertraut sind, empfehle ich Ihnen, zuerst [3] und [4] anzusehen.

Ein benutzerdefinierter Sensor

ESPHome unterstützt viele Sensoren, aber (zum Zeitpunkt, als ich dies geschrieben habe) noch nicht den VEML7700. Es kennt zwar andere Sensoren für Beleuchtungsstärke, aber leider nicht diesen. Das ist jedoch kein Problem, denn ESPHome bietet eine Methode, einen eigenen Sensor hinzuzufügen. Dazu muss man allerdings etwas C++-Code schreiben, was die Angelegenheit ein wenig komplizierter macht.

Wie zuvor (siehe [3] und [4]) müssen wir in der ESPHome-Konfigurationsdatei eine *sensor*-Sektion für den VEML7700 deklarieren. Die *platform* des Sensors muss als *custom* definiert sein. Damit wird ESPHome mitgeteilt, dass Sie alle Details für den Sensor zur Verfügen stellen wollen.

Als nächstes folgt ein *lambda*-Abschnitt aus ein paar Zeilen C++-Code, um ESPHome anzuweisen, einen von uns entwickelten Sensor zu registrieren (den wir *veml7700* nennen). Wir müssen auch die Datenströme angeben, die unser Sensor erzeugt. Neben Registern für Weißes Licht und AL übernimmt die zum Adafruit-Modul gehörende Bibliothek die Berechnung des direkten Lux-Wertes. Die Reihenfolge der drei Ausgänge: ALS, Lux und Weißes Licht ist wichtig und muss beachtet werden, wenn an anderer Stelle in der YAML-Datei auf sie verwiesen wird.

Wir fahren mit normalen YAML-Anweisungen fort, um die Sensorausgänge weiter zu spezifizieren. Dies geschieht in einem Abschnitt *sensors* (Plural!), in dem wir für jeden Datenstrom den Namen, die Einheiten und die Anzahl der zu verwendenden Dezimalstellen angeben können. Die Reihenfolge ist dieselbe wie in der *return*-Anweisung im Lambda-Abschnitt. Nur die Lux-Daten haben eine Einheit (lx), und es hat keinen Sinn, bei irgendeinem Wert Dezimalstellen anzugeben, so dass wir deren Anzahl auf Null setzen.

Als letztes muss angegeben werden, wo ESPHome den Treiber für den benutzerdefinierten Sensor finden kann. Dies geschieht in der Sektion *esphome* am Anfang der Konfigurationsdatei, wo wir eine Datei (*veml7700.h*) zum *includes*-Unterabschnitt hinzufügen. Auf dem System, auf dem ESPHome läuft, sollte sich diese Datei im selben Ordner befinden wie die entsprechende YAML-Datei.

Der schwierige Teil

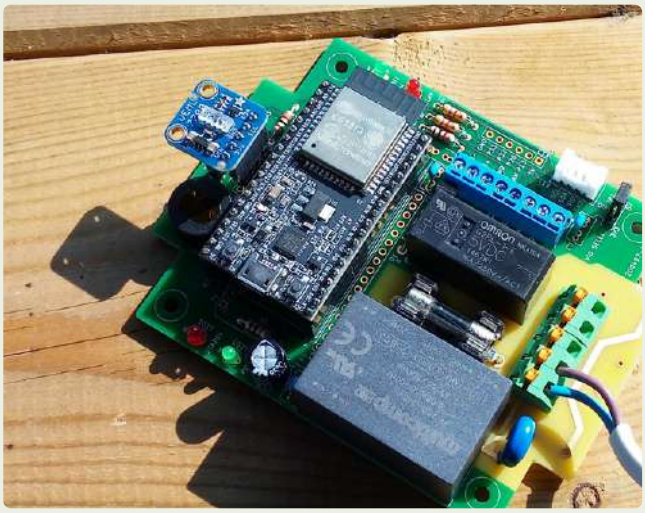
Geschafft? Nicht ganz. Den schwierigen Teil haben wir uns für das Ende aufgespart, nämlich die Erstellung des Treibers für den benutzerdefinierten Sensor. Dieser Treiber muss den ESPHome-Standards für *components* entsprechen (ein Sensor ist eine solche Komponente), das heißt, er muss bestimmte Funktionen bereitstellen, die ESPHome von seinen Komponenten erwartet, und er muss mit dem Sensor selbst kommunizieren.

Für die zweite Aufgabe können wir uns ganz auf Adafruit verlassen, die nicht nur das VEML7700-Modul herstellen und anbieten, sondern auch eine Arduino-Bibliothek dafür entwickelt haben [5]. Unser Treiber muss dagegen die Schnittstelle zwischen ESPHome und der Arduino-Bibliothek bereitstellen.

Installation einer Bibliothek eines Drittanbieters

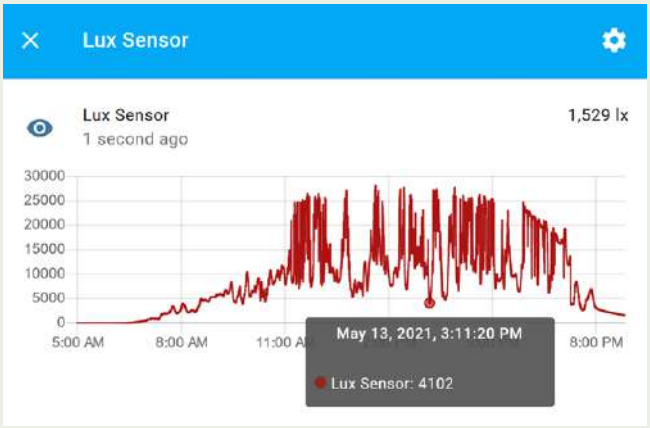
ESPHome bietet einen Mechanismus zum Hinzufügen von Open-Source-Community-Bibliotheken: Fügen Sie einfach den genauen (!) Namen der Bibliothek im esphome-Bereich in die Untersektion *libraries* ein. In unserem Fall ist dies „Adafruit VEML7700 Library“. Wenn ESPHome die Konfigurationsdatei verarbeitet, installiert es die Bibliothek, wenn noch nicht gesche-

Einige Ergebnisse



Zur Mittagszeit, mit einer Empfindlichkeit von 0,125 und einer Integrationszeit von 100 ms, meldete der Sensor Werte von etwa 48.000 für ALS und etwas mehr als 30.000 für weißes Licht. Diese Werte entsprechen einer Lichtintensität von etwa 22.000 lx. Um 10:30 Uhr habe ich unter den gleichen Bedingungen 16.800 lx gemessen, eine Differenz von 5.200 lx. Meine subjektiven Augen konnten so gut wie keinen Helligkeitsunterschied erkennen.

Entwicklung der Helligkeit an einem sonnigen Tag im Juli. Das „Prellen“ auf der linken Seite ist auf den Schatten eines Baumes zurückzuführen. Von Mittag bis 18 Uhr nimmt die Helligkeit langsam ab, was man nicht wirklich merkt. Die kurzen Einbrüche werden von Wolken verursacht. Der starke Abfall ab 18 Uhr wird durch das Haus verursacht, der den Sensor für etwa eine Stunde abschattet. Danach nimmt die Lichtintensität bis zur Dunkelheit kontinuierlich ab. Menschen empfinden bereits 5.000 lx als ziemlich hell.



An einem teilweise bewölkten Tag im Mai schwankte die Beleuchtungsstärke zwischen etwa 4.000 lx und 27.000 lx.

Alle Diagramme wurden mit Hilfe von Home Assistant auf einem Breitengrad von lat = 47°N erstellt.

PASSENDE PRODUKTE

- ESP32 DevKitC SKU 18701
  www.elektor.de/18701
- I²C-OLED-Display 0,96“ 128×64 SKU 18747
  www.elektor.de/18747
- Koen Vervloesem, Getting Started with ESPHome
  Buch, SKU 19738: www.elektor.de/19738
  E-Buch, SKU 19739: www.elektor.de/19739

hen (**Bild 2**), bevor es an die Kompilierung geht. In Ihrem eigenen Treiber können Sie die Bibliothek so einfach wie jede andere Bibliothek einbinden.
Ich habe mich mit dieser Funktion noch nicht tiefgreifend beschäftigt und kenne die Kriterien nicht, die eine Bibliothek eines Drittanbieters erfüllen muss, um auf diese Weise verwendet zu werden. Sie sollten die Dokumentation *platform.io* konsultieren, wenn Sie weitere Details erfahren möchten, da dies die von ESPHome verwendete Toolchain ist.

Ein gekapselter Arduino-Sketch

Unser Treiber ist eine C++-Klasse mit mindestens einem Konstruktor und einer Funktion namens *update*. Wir brauchen auch eine Funktion *setup*, damit wir den Adafruit-Treiber initialisieren können. Die Funktionen *setup* und *update* der Klasse tun das, was normalerweise in den Funktionen *setup* und *loop* eines typischen Arduino-Sketches für den Adafruit-Treiber geschehen würde. Im Grunde kapselt unsere Klasse einen Arduino-Sketch einschließlich globaler Variablen und fügt ihm einige ESPHome-spezifische Dinge hinzu. Für ESPHome müssen wir Calls zu *publish_state* für jeden Datenstrom (ALS, Lux und Weißlicht) in die Funktion *update* einfügen. Dadurch werden die Daten für den Rest der Welt verfügbar gemacht. Die Reihenfolge der Calls muss dieselbe sein wie in der Return-Anweisung in der *lambda*-Untersektion der *sensor*-Sektion.
Da unsere Klasse von der *PollingComponent*-Klasse erbt, die ihrerseits von der *Component*-Klasse erbt, stehen weitere Funktionen zur Verfügung, die Sie vielleicht verwenden möchten. Eine Polling-Komponente wird periodisch mit festgelegten Intervallen aufgerufen, zum Beispiel in unserem Konstruktor (siehe [6] für weitere Details).

Hinzufügen des I²C-Busses

Jetzt müssen wir nur noch den I²C-Bus des Sensors in der Software verbinden, um neben der pysikalischen auch eine logische Verbindung herzustellen. Die Adafruit-Bibliothek verwendet zu diesem

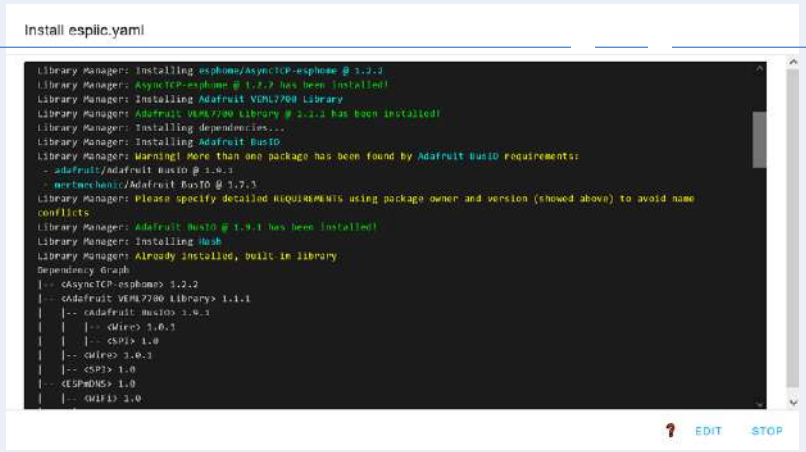


Bild 2. ESPHome installiert die Adafruit-Bibliothek automatisch.

Zweck die Wire-Bibliothek von Arduino. In der ESPHome-YAML-Datei fügen wir daher eine *i2c*-Sektion hinzu, damit ESPHome weiß, dass der I²C-Bus verwendet wird. Der ESP32 verfügt über zwei I²C-Busse mit SDA und SCL, die an fast jeden Pin des Chips angeschlossen werden können. ESPHome unterstützt daher mehrere I²C-Busse mit frei zuweisbaren Pins, so dass man nur noch angeben muss, wohin was gehört. Allerdings unterstützt die Arduino-Wire Bibliothek nur einen I²C-Bus. Wie kann man ihr mitteilen, dass sie den von Ihnen gewünschten Bus verwenden soll? Gar nicht! Die Bibliothek will immer den Standard-Bus verwenden. In ESPHome ist dieser Default-I²C-Bus der erste Bus, der in der *i2c*-Sektion angegeben ist. Es wäre schön gewesen, wenn man einen I²C-Bus über seine ID angeben könnte, und ESPHome wäre wahrscheinlich mehr als glücklich, wenn man das tun könnte, aber die zugrundeliegende Arduino-Wire Bibliothek erlaubt das nicht.

Fertig!

Hier endet dieser Artikel. Wenn Sie den Automator von ESPHome mit dem oben beschriebenen Code programmieren, erhalten Sie ein Gerät, das von einer Hausautomatisierungssteuerung wie Home Assistant gesteuert werden kann. In der YAML-Datei von ESPHome sind keine Automatisierungsregeln enthalten, so dass das Gerät ohne eine Steuerung nur die Lichtintensität der Umgebung misst. Automatisierungsregeln können der YAML-Datei selbst hinzugefügt oder im Home Assistant erstellt werden. Eine Beschreibung des restlichen, nicht besonders aufregenden YAML-Codes finden Sie in [3] und [4]. Die YAML-Konfigurationsdatei und der C++-Code stehen auf der Webseite dieses Artikels zum Download bereit [7].

210190-02

Ein Beitrag von

Gestaltung, Text und Fotografien: **Clemens Valens**
Redaktion: **Jens Nickel, C. J. Abate**
Übersetzung: **Rolf Gerstendorf**
Layout: **Harmen Heida**

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.

WEBLINKS

[1] Y. Bourdon, „ESP32-verbundenes Thermostat“, ElektorMag 9/2021: www.elektormagazine.de/magazine/elektor-182/59862
[2] Elektor Automator: www.elektormagazine.de/labs/automator
[3] ESPHome starts here: www.elektormagazine.de/labs/how-to-home-assistant-esphome
[4] C. Valens, „Hausautomation leicht gemacht“, ElektorLab 9/2020: www.elektormagazine.de/magazine/elektor-154/58936
[5] Bibliothek Adafruit VEML7700: https://github.com/adafruit/Adafruit_VEML7700
[6] ESPHome auf GitHub: https://github.com/esphome
[7] Downloads zu diesem Artikel: www.elektormagazine.de/210190-02

Herausforderungen bei der Markteinführung von **IoT-Lösungen**

Sorgen um Sicherheit, Skalierbarkeit und Wettbewerb

Bild 1. Die Europäische Kommission hat untersucht, ob die großen Akteure im Bereich der Sprachassistenten, der beliebtesten Benutzerschnittstelle für das intelligente Zuhause, den Wettbewerb behindern. (Quelle: ShutterStock/Gorodenkoff)

Von Stuart Cording (Elektor)

Das Internet der Dinge (IoT) gibt es seit mehr als 20 Jahren, und eine Vielzahl von drahtlosen Technologien ist entstanden, um seine Einführung zu unterstützen. Im privaten Bereich haben sich Sprachassistenten als wichtigste Benutzerschnittstelle für eine Reihe intelligenter Geräte etabliert. Trotz dieser Fortschritte schaffen es mehr als ein Drittel der IoT-Projekte nie über die Phase des Proof-of-Concept hinaus. Außerdem ist die Europäische Kommission besorgt, dass ein Mangel an Wettbewerb in einigen Anwendungsbereichen den Markteintritt von EU-Unternehmen behindern könnte. Was sind also die wirklichen Herausforderungen, und wie sieht es mit der Einführung einer IoT-Lösung in Europa aus?

Wenn Sie ein Gefühl für die Technologien bekommen wollen, die hinter dem Internet der Dinge (IoT) stehen, finden Sie schnell eine ganze Reihe von geeigneten Projekten. Suchen Sie einfach nach „IoT“ und der von Ihnen bevorzugten Prototyping-Plattform, und Sie werden mit Projekten, Cloud-Plattformen, Technologievergleichen und Listen von Ideen überschüttet. Auch Elektor ist eine hervorragende Informationsquelle: Seit der Begriff IoT vor mehr als 20 Jahren geprägt wurde, hat unsere Website mehr als 600 Artikel zum Thema veröffentlicht [1].

Zwischen einer Prototyp-Plattform, die das grundlegende Konzept einer IoT-Lösung veranschaulicht, und der tatsächlichen Realisation einer solchen Lösung klafft jedoch eine beträchtliche Lücke. Der Bericht *IoT Insights* von Microsoft [2] befragte im Jahr 2021 mehr als 3.000 IoT-Experten. Sie fanden heraus, dass 35 % der IoT-Projekte in der Test- oder der Proof-of-Concept-Phase scheitern, 5 % mehr als bei der gleichen Umfrage ein Jahr zuvor. Als Grund für das Scheitern wurden meist die hohen Kosten für die Skalierung genannt. Weitere Gründe sind die Unmengen zu testender Plattformen, zu prüfender Anwendungsfälle und auch ein Mangel an Ressourcen. Eine andere Studie von Cisco [3] berichtet, dass nur 26 % der befragten Unternehmen ihre IoT-Initiativen für erfolgreich hielten. Die Antworten in dieser Studie zeigten, dass die meisten IoT-Projekte zwar auf dem Papier gut aussehen, sich aber letztendlich als komplexer erweisen als ursprünglich angenommen. Trotz dieser düsteren Rückmeldungen zum IoT insgesamt gibt es Bereiche, in denen das IoT große Fortschritte macht und wachsende Umsätze bringt.

EU prüft das IoT für Verbraucher

Viele EU-Bürger haben in den letzten Jahren das Angebot an IoT-Lösungen für Verbraucher begrüßt, und zwar so sehr, dass ein Bericht von Statista [4] über die Umsätze im Bereich *Smart Home* vorhersagt, dass sie sich von rund 17 Milliarden Euro im Jahr 2020 auf rund 38,1 Milliarden Euro im Jahr 2025 mehr als verdoppeln werden. Aus Sorge um den freien Wettbewerb in diesem Bereich hat die Europäische Kommission im Rahmen ihrer Digitalen Strategie eine Untersuchung durchgeführt [5]. Der Bericht dazu, der im Januar 2022 veröffentlicht wurde, enthält Beiträge von Herstellern portabler Geräte, von vernetzten Verbrauchergeräten, die im intelligenten Heim verwendet werden, und von Anbietern von Diensten über diese intelligente Geräte. Außerdem bat die Kommission um Beiträge von Organisationen, die Normen und Standards festlegen. Es fällt auf, dass ein Großteil der Beiträge Sprachassistenten (Voice Assistants) als Benutzerschnittstelle für IoT-Produkte und -Dienste gewidmet sind (**Bild 1**). In der Analyse der Smart-Home-Landschaft macht der Bericht [6] deutlich, dass Googles *Google Assistant*, Amazons *Alexa* und Apples *Siri* die führenden Allzweck-Sprachassistenten in Europa sind. Es gibt weitere Sprachassistenten, die jedoch in ihrer Funktionalität eher beschränkt sind und sich auf die Unterstützung eines einzelnen Produkts oder einer App eines Dienstleisters konzentrieren. Laut ZDNet bietet unter den drei großen Anbietern Amazons *Alexa* den höchsten Grad an Kompatibilität und unterstützt rund 7.400 Marken [7]. Im Vergleich dazu unterstützt Google etwa 1.000 Marken, während Apple mit etwa 50 Marken die größte „Exklusivität" aufweist.

Wenig Chancen für Neueinsteiger

Bei so mächtigen und auf dem Markt etablierten Akteuren bleibt wenig Platz für Neueinsteiger, und der technologische Aufwand zur Entwicklung eines wettbewerbsfähigen Sprachassistenten ist beträchtlich. Wenn Sie also eine Sprachsteuerung für Ihre IoT-Lösung nutzen wollen, müssen Sie nach der Musik der großen Drei tanzen. Ein alternativer Ansatz wäre die Lizenzierung eines Sprachassistenten, doch einige befragte Hersteller berichteten, dass die Lizenzbedingungen ihre Wahlmöglichkeiten stark einschränken. Diese reichten von Exklusivität oder Beschränkungen, die den gleichzeitigen Einsatz von mehr als einem Sprachassistenten verhinderten, bis hin zu einer Lizenzierung, die die Einbeziehung anderer Arten von Software oder Anwendungen erzwang, was bedeutete, dass die Sprachassistent-Technologie nicht eigenständig verwendet werden konnte.

Ein weiteres großes Problem ist der Zugang zu den Daten. Als „Third Party", die auf einen Sprachassistenten aufbaut, haben Sie nur begrenzten Zugriff auf die erfassten Daten. Der Anbieter des Sprachassistenten selbst hat aber Zugang zu den Audioaufzeichnungen und weiß auch, wie viele fehlgeschlagene Versuche es gab, die für Ihr Gerät ausgewählten Befehle zu erteilen. Sie und Ihr Team haben jedoch keinen Zugang zu den Audioaufzeichnungen, so dass Sie eher auf das Feedback der Endanwender warten müssen, um festzustellen, dass die von Ihnen gewählten Sprachbefehle in der rauen Praxis nicht so funktionieren wie in einem kleineren Test. Da der Sprachassistent-Anbieter außerdem alles analysieren kann, was in das Gerät gesprochen wird, könnte er diese Daten nutzen, um eine Lösung zu entwickeln, die mit der Ihren konkurriert, oder um die von Ihrer IoT-Lösung gelieferte Benutzererfahrung zur Verbesserung seiner Dienste zu nutzen.

Die IoT-Landschaft zeigt zahlreiche Geschäftsmöglichkeiten, unabhängig davon, ob es Lösungen für Verbraucher oder die Industrie sind.

Ein weiteres Problem ergibt sich, wenn der Sprachassistent-Anbieter auch Werbedienste anbietet. Theoretisch könnten ihm die von Ihren Nutzern bereitgestellten Spracheingaben dabei helfen, die Werbung genauer auf die durch Ihren Kundenstamm repräsentierte gesellschaftliche Gruppe auszurichten.

Und schließlich geht auch der Wiedererkennungswert und die Erfahrung mit der Marke verloren. Ihre sorgfältig ausgearbeitete Lösung ist den Launen und der Gnade des Sprachassistenten ausgeliefert. Sollte sich der Anbieter dazu entschließen, eine wesentliche Änderung vorzunehmen, zum Beispiel die verwendete Stimme, das Weckwort oder sogar die Einführung von Funktionen zu ändern, die zu einem Rückgang der Nutzerzahlen führen, werden Sie unweigerlich auch diese Folgen zu spüren bekommen.

In dem Bericht werden auch viele andere relevante Bereiche untersucht, darunter Anwendungsprogrammierschnittstellen (API), Standards, Interoperabilität, das Ungleichgewicht der „Macht" von Drittentwicklern von IoT-Geräten und den großen Anbietern von Cloud-Plattformen sowie Vertragskündigungsklauseln.

Der Bericht enthält zwar keine Empfehlungen, doch in den Schlussfolgerungen heißt es, dass der Inhalt des Berichts zur Standardisierungsstrategie der Europäischen Kommission beiträgt und in die legislative Debatte über das Gesetz über digitale Märkte (DMA) einfließen wird.

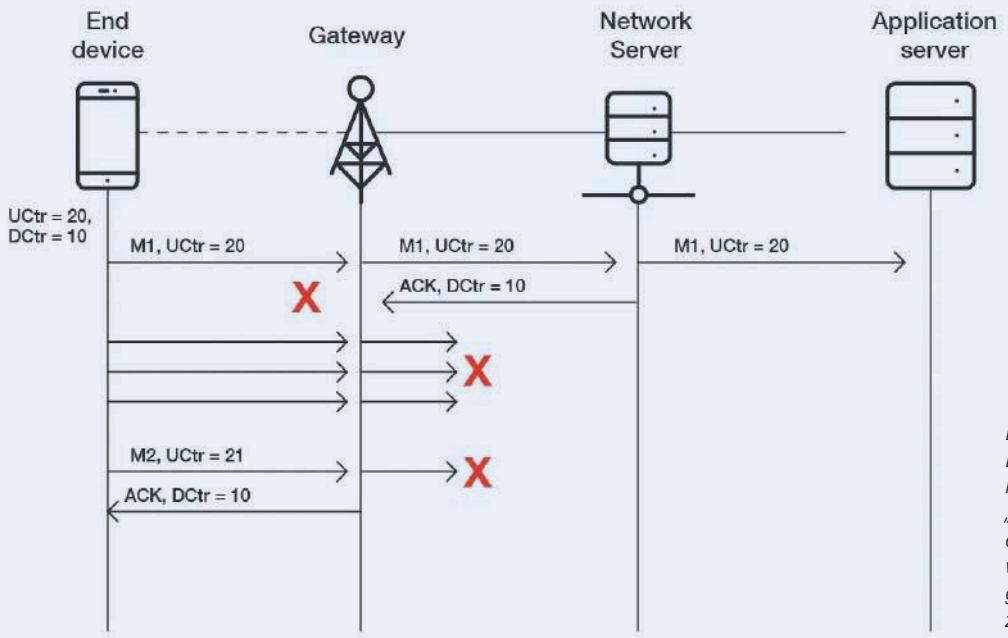


Bild 2. Forscher entdeckten einen Denial-of-Service–Angriff (DoS) in LoRa 1.0. Durch das erneute „Abspielen" einer vorherigen erfolgreichen Datenübertragung wird ein LoRaWAN-Knoten daran gehindert, weitere Datenpakete zu senden. (Quelle: Trend Micro)

Sicherheitsprobleme beunruhigen IoT-Implementierer

Bei den Daten im IoT-Insights-Bericht von Microsoft steht die IoT-Sicherheit ganz oben auf der Kummerkasten-Liste. 29 % der Befragten gaben an, dass die damit verbundenen Sicherheitsrisiken sie davon abhalten, das IoT verstärkt einzusetzen. Der Bericht zeigt auch, dass etwa ein Drittel der Unternehmen über die Sicherheitsrisiken des IoT besorgt ist, insbesondere über Verletzungen des Datenschutzes. Um dem entgegenzuwirken, wird Outsourcing als beste Möglichkeit beschrieben, den „Seelenfrieden" zu verbessern. Zwar kennen viele Ingenieure den Satz „Sicherheit durch Unklarheit" (beziehungsweise das Gegenteil), doch nur wenige sind in diesem Bereich erfahren genug, um eine durchgängige Lösung für den Schutz gegen Angreifer zu entwickeln. Und obwohl die Halbleiterhersteller eine Reihe von Single-Chip-Sicherheitslösungen anbieten, müssen die Entwickler lernen, diese richtig einzusetzen, um nicht versehentlich neue Sicherheitslücken zu eröffnen. In den letzten Jahrzehnten haben sich Low-Power-Wide-Area-Networks (LPWAN)-Lösungen wie LoRa und Sigfox als wichtige drahtlose IoT-Technologien etabliert, die eine Kommunikation mit großer Reichweite ermöglichen. Sie erreichen Reichweiten von mehreren zehn Kilometern und sind dank ihrer überlegenen Low-Power-Leistung bei geringen Datenmengen eine Alternative zu Mobilfunknetzen wie LTE Cat-M1 und NB-IoT [8]. Aber wie sicher sind sie?

LoRaWAN auf dem Prüfstand

LoRa und LoRaWAN wurden von der Sicherheits- und Hackergemeinde besonders unter die Lupe genommen. Sébastien Dudek von Trend Micro, einem Unternehmen für IT-Sicherheitslösungen, ist einer von mehreren Forschern, die ausführlich über potenzielle Probleme veröffentlicht haben. In einer Reihe von drei technischen Kurzberichten [9][10][11] skizziert er Probleme bei der Implementierung und möglichen Angriffen, angefangen von Denial-of-Service- (DoS) (**Bild 2**) und Lauschangriffen, Bitflipping (**Bild 3**) bis zu Spoofing von Bestätigungen (ACK) (**Bild 4**). Die Folgen solcher Angriffe reichen von der Unfähigkeit, mit einem Knoten zu kommunizieren, über die gewollte Verkürzung der Batterielebensdauer bis hin zur Veränderung von Anwendungsdaten.

Viele der aufgezeigten Schwachstellen wurden zwischen Version 1.0.2 und 1.1 des LoRaWAN-Standards behoben. Weitere Herausforderungen ergeben sich jedoch, wenn LoRaWAN-Knoten an Gateways betrieben werden, die unterschiedliche Versionen der Spezifikation verwenden. In solchen Fällen müssen Änderungen vorgenommen werden, um eine gesicherte Abwärtskompatibilität zwischen Endgeräten und Backend zu gewährleisten, wie in einem Papier von Tahsin C. M. Dönmez aus dem Jahr 2018 hervorgehoben wird [12].

Neben dem Hacken von drahtlosen Verbindungen gibt es auch das Problem, dass bösartige Akteure die Hardware stehlen und direkt angreifen können. Sébastien Dudek untersucht auch diesen Sicherheitsaspekt. Im Falle von LoRaWAN verwenden viele Entwickler einen Mikrocontroller (MCU) und ein drahtloses Modul von Semtech, die über SPI verbunden sind. Daten, die zwischen den beiden ausgetauscht werden, können leicht erfasst und analysiert werden.

Darüber hinaus stellt sich auch die Frage nach der Sicherheit der MCU selbst. Bei einer Angriffsmethode wird einfach die Firmware aus dem Flash-Speicher extrahiert, so dass der Code analysiert werden kann. Wenn sich die Sicherheitsschlüssel auch in der Firmware befinden, kann ein Angreifer diese verwenden.

Darüber hinaus stellt sich auch die Frage nach der Sicherheit der MCU selbst. Ein Angreifer kann einfach die Firmware aus dem Flash-Speicher auslesen und den Code analysieren. Wenn sich Sicherheitsschlüssel in der Firmware befinden, kann ein Angreifer sie verwenden, um Knoten zu entwickeln, die authentische Endgeräte vortäuschen. Als Gegenmaßnahme wird die Verwendung von Secure Elements (SE) empfohlen, Einzelchip-Authentifizierungsbausteine, die Verschlüsselungsschlüssel sicher speichern.

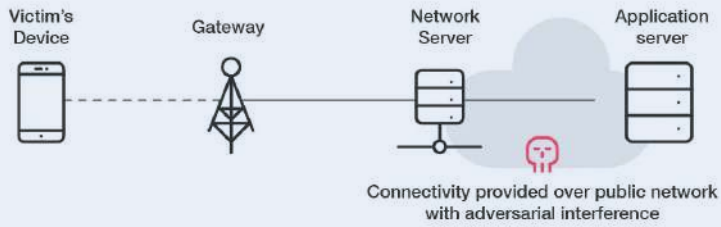


Bild 3. Aufgrund der bekannten festen Struktur von LoRaWAN-Datenpaketen ist es möglich, den Inhalt von Bits umzudrehen (Bit-Flipping-Angriff), ohne die Nachricht entschlüsseln zu müssen. Dazu ist der Zugriff auf den Netzwerkserver erforderlich, der die Daten empfängt. (Quelle: Trend Micro)

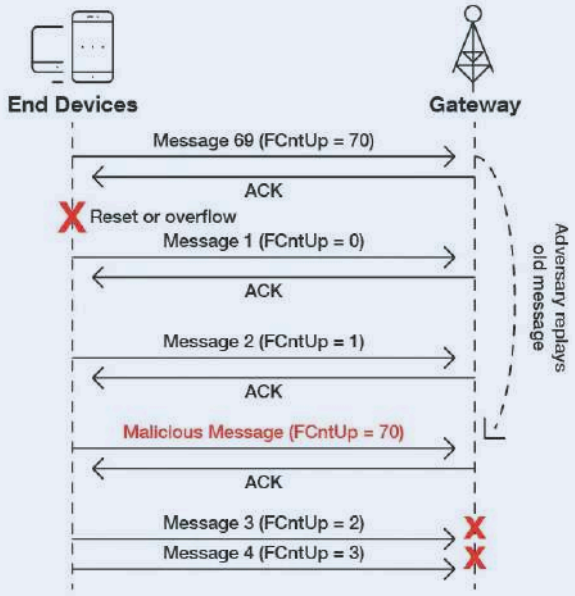


Bild 4. Das Stören der drahtlosen Verbindung führt dazu, dass der LoRaWAN-Knoten die Paketübertragung bis zu sieben Mal wiederholt, was die Batterielebensdauer verkürzt. Wenn der Angreifer auch Acknowledgment-Pakete (ACK) abfängt und wiedergibt, denkt der Knoten, dass die Verbindung noch funktioniert, da ACK-Pakete die Nachricht, die sie bestätigen, nicht deklarieren. (Quelle: Trend Micro)

Ein Ansatz mit dem ATECC608A von Microchip [13] ist einer von mehreren, für die Beispielcode verfügbar ist. Während aber die Beispielprojekte zeigen, wie die kryptografischen Schlüssel geschützt werden können, wird die sichere Boot-Funktion dieses Authentifizierungsbausteins jedoch nicht behandelt. Würde aber dieser Ansatz für ein Produkt verwendet, könnte der Authentifizierungsbaustein entfernt und als SE mit einer anderen MCU und neuer Firmware verwendet werden.

Sicherheitsprobleme, wohin man schaut

LoRaWAN-Endgeräte bieten nur eine begrenzte Datenbandbreite und sind nicht adressierbar, da sie keine IP-Adresse wie ein WLAN-Modul haben. Daher stellen sie ein minimales Risiko für Unternehmensnetzwerke dar. Potenzielle Risiken bestehen jedoch bei Anwendungen, die auf solchen drahtlosen Technologien basieren. Diese können Auswirkungen auf Leben und Umwelt haben, wenn etwas schief geht. Beispielsweise können LoRa-basierte Sensoren als Teil einer intelligenten Überwachung des Wasserstands zuständig sein, um Überschwemmungen zu vermeiden. Wenn die Daten der Sensoren blockiert werden, können die Hochwasserschutzsysteme möglicherweise nicht reagieren. Umgekehrt könnten falsch eingespeiste Daten dazu führen, dass der Hochwasserschutz auf ein Ereignis reagiert, das gar nicht stattfindet, mit möglicherweise ebenso katastrophalen Folgen.

LoRa und LoRaWAN sind keine Einzelfälle. Viele Forscher untersuchten auch andere LPWAN-Technologien wie Sigfox und NB-IoT. In einem Papier von Florian Laurentiu Coman et al. [14] werden mehrere Proof-of-Concept-Angriffe auf andere drahtlose Netzwerke beschrieben. In einem technischen Brief der Deutschen Telekom [15] heißt es, dass die Implementierung von Sigfox und LoRaWAN „ohne Sicherheitselement selbst eine Ende-zu-Ende-Verschlüsselung wirkungslos sein kann." Im Gegensatz dazu profitiert NB-IoT von den seit langem bewährten LTE-Sicherheitsmerkmalen wie Authentifizierung und sichere Schlüsselgenerierung

Bild 5. Feinstaubsensoren von DEUS POLLUTRACK sammeln Daten von stationären und mobilen IoT-Geräten. Die Daten werden über 4G- und 5G-Mobilfunknetze an das Backend übermittelt. (Quelle: DEUS POLLUTRACK)

Bild 6. Ein cloudbasiertes Dashboard zeigt die Luftschadstoffwerte (hier für Hamburg) an. Die lokalen Behörden nutzen solche Daten, um Entscheidungen über die Verkehrsführung zu treffen. (Quelle: DEUS POLLUTRACK)

und -austausch. Es wird jedoch auch klargestellt, dass eine End-to-end-Verschlüsselung nicht zum Standard gehört und, falls sie als notwendig erachtet wird, mit dem Netzbetreiber besprochen werden muss.

Bereitstellung von stadtweiten IoT-Lösungen

Bedenken hinsichtlich der Sicherheit in LPWAN-Netzen beeinflussten auch die Wahl der Technologie der DEUS POLLUTRACK Smart City für ihre IoT-Plattform [16]. Das DEUS-Team entwickelt seit mehr als einem Jahrzehnt IoT-Sensornetzwerke zur Überwachung von Feinstaub in Städten. Die Technologie wird in mehr als 15 europäischen Städten eingesetzt und ermöglicht es den Verantwortlichen in den Kommunen, fundierte Entscheidungen zur Luftverschmutzung zu treffen. Die patentierten optischen Partikelzähler (OPC) sind in der Lage, bis hinunter zur Klassifizierung ultrafeiner Partikel (unter 0,1 µm) zu überwachen. Während größere Partikel wie PM_{10} als gefährlich für die Lunge gelten, können ultrafeine Partikel über die eingeatmete Luft in den Blutkreislauf und in andere Organe gelangen.

Die Sensortechnologie von DEUS (**Bild 5**) kombiniert stationäre und mobile Sensoren, die mit Back-End-Dashboards vernetzt sind, auf denen die erfassten Daten visualisiert werden. Städte wie Marseille und Paris verwenden 40 stationäre Sensoren, die durch 300 mobile Sensoren ergänzt werden [17]. Mobile Sensoren werden an Fahrzeugen von Partnern angebracht, etwa den Lieferwagen von DPD, die ohnehin häufig in der jeweiligen Stadt unterwegs sind. Diese Sensoren kalibrieren sich selbst anhand der bei der Vorbeifahrt an stationären Sensoren erfassten Daten, um die erforderliche Genauigkeit zu gewährleisten. All dies erfordert die Wahl eines robusten, zuverlässigen und sicheren LPWAN.

Im Gespräch erklärt der DEUS-Mitbegründer Marc Nodorft, dass in den frühen Entwicklungsphasen sowohl Sigfox als auch LoRaWAN in Betracht gezogen wurden. Sigfox bot eine Konnektivitätsinfrastruktur, die die Systembereitstellung vereinfachte, aber keine der beiden LPWANs bot den erforderlichen Datendurchsatz. LoRaWAN war damals nicht sicher genug, und ohne Infrastrukturpartner in den Städten, in denen die Technologie eingesetzt werden sollte, mussten Gateways eingesetzt werden, die über Mobilfunknetze mit dem Backend verbunden waren. Infolgedessen wurden 4G- und später 5G-Mobilfunknetze gewählt, mit denen die Probleme der Abdeckung, Zuverlässigkeit und Sicherheit auf dem erforderlichen Niveau gelöst wurden.

Nodorft berichtete uns auch, dass es zwar viele billige elektronische Lösungen für das Internet der Dinge gibt, diese aber nicht robust genug für einen langfristigen Einsatz in den Umgebungen sind, in denen die DEUS-Geräte installiert werden. Daher entschied man sich für die Entwicklung nach Industriestandards, eine Möglichkeit für alle, die eigene IoT-Produkte planen.

Ein weiterer Aspekt ist der Back-End-Betrieb, der speziell für die Anforderungen der IoT-Implementierung entwickelt wurde (**Bild 6**). In Zukunft müssen Open-Source-Dashboards für die Berichte unterstützt werden, damit die Behörden, die das System nutzen, und die Bürger auf die Daten zugreifen können, wofür ein Anbieter von Cloud-Diensten erforderlich ist. Und obwohl es eine große Auswahl gibt, wird der Anbieter als ebenso wichtig angesehen wie die technische Lösung. Es wird also nach einem Anbieter gesucht, der persönliche Unterstützung bieten kann und nicht nur einen unpersönlichen Chatbot für den Kundendienst.

Mit so viel Erfahrung in bedeutenden IoT-Implementierungen und viel Wissen über die technischen Herausforderungen fragte ich mich, welche anderen Ratschläge Nodorft denjenigen geben könnte, die IoT-Lösungen implementieren wollen. „Wir sind unserer Vision immer treu geblieben", meint er, „was uns oft dazu gezwungen hat, unseren Ansatz zu ändern." Das bedeutete, verschiedene Technologien zu erkunden, mit verschiedenen Partnern zusammenzuarbeiten und die Vertriebsstrategie auf dem Weg zum Erfolg zu ändern.

Expertenteams und Partnerschaften erforderlich

Betrachtet man die heutige IoT-Landschaft, so wird deutlich, dass es eine Fülle von Geschäftsmöglichkeiten gibt, unabhängig davon, ob man sich auf Lösungen für Verbraucher oder die Industrie konzentriert. Der Weg vom Konzept bis zur Markteinführung ist jedoch mit vielen Herausforderungen verbunden. Entwickler von eingebetteten Systemen mögen sich zwar mit der Entwicklung von Hardware und Firmware auskennen und sogar Erfahrung mit drahtlosen Technologien haben, doch das IoT und die damit verbundenen Herausforderungen in Bezug auf Sicherheit und Skalierbarkeit können eine Organisation rasch überfordern. Laut dem Bericht der Europäischen Kommission haben große Unternehmen auch bei den Geschäftsbeziehungen die Oberhand, wenn es um Dienste und Plattformen geht. Kleine Akteure und Startups dürften es als Einzelkämpfer schwer haben, in diesen asymmetrischen Beziehungen die nötige Unterstützung zu erhalten. Zweifelsohne ist Fachwissen, auch angemietet oder geliehen, unerlässlich, um über Beispielanwendungen, Demonstrations-Dashboards und Test-IoT-Dienste hinauszukommen. Und schließlich ist es von entscheidender Bedeutung, dass Sie an Ihrer Vision festhalten und gleichzeitig in allen Bereichen der Umsetzung flexibel bleiben - von der Wahl der Technologie bis zum Zielmarkt, um sie zu verwirklichen.

220053-02

Ein Beitrag von

Text: **Stuart Cording**
Redaktion: **Jens Nickel, C. J. Abate**
Übersetzung: **Rolf Gerstendorf**
Layout: **Harmen Heida**

Haben Sie Fragen oder Kommentare?

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

WEBLINKS

[1] Elektor IoT-Artikel: www.elektormagazine.de/select/internet-of-things
[2] „IoT Insights, Edition 3", Microsoft/Hypothesis, Oktober 2021: https://bit.ly/3rxMk3a
[3] „The Journey to IoT Value", Cisco, Mai 2017 (Bezahlschranke): https://bit.ly/3GzdJWS
[4] Dr. J. Lasquety-Reyes, „Smart Home - revenue forecast in Europe from 2017 to 2025", Statista, Juni 2021 (Bezahlschranke): https://bit.ly/3LlGiuG
[5] „Sector inquiry into the Consumer Internet of Things", Europäische Kommission, Januar 2022: https://bit.ly/3Lgw9iE
[6] „Final report - sector inquiry into consumer Internet of Things", Europäische Kommission, Januar 2022: https://bit.ly/3B2Htu9
[7] „The best voice assistent", ZDNet, September 2021: https://zd.net/3rxf6Rt
[8] L. Tan, „Comparison of LoRa and NBIoT in Terms of Power Consumption", KTH Royal Institute of Technology, Januar 2020: https://bit.ly/3JafsUb
[9] S. Dudek, „Low Powered and High Risk: Possible Attacks on LoRaWAN Devices", Trend Micro, Januar 2021: https://bit.ly/3rA02Tg
[10] S. Dudek, „Gauging LoRaWAN Communication Security with LoraPWN", Trend Micro, Februar 2021: https://bit.ly/3LhV0T5
[11] S. Dudek, „Protecting LoRaWAN Hardware from Attacks in the Wild", Trend Micro, März 2021: https://bit.ly/3rxquge
[12] T.C.M. Dönmez, „Security of LoRaWAN v1.1 in Backward Compatibility Scenarios", Elsevier 2018: https://bit.ly/3GtzKq0
[13] Microchip-Produktseite - ATECC608A: https://bit.ly/3B7zIms
[14] F.L. Coman et al. „Security issues in internet of things: Vulnerability analysis of LoRaWAN, sigfox and NB-IoT", IEEE, Juni 2019: https://bit.ly/3uwhUQX
[15]] „NB-IoT, LoRaWAN, Sigfox: Ein aktueller Vergleich", Deutsche Telekom AG, April 2021: https://bit.ly/3uyUydj
[16] Website DEUS-POLLUTRACK: https://bit.ly/3sHL9O5
[17] DEUS-Sensormessnetze: https://bit.ly/3Gzjbcc

Lieber doch **verkabelt**

Tipps zur Entwicklung einer 1-Gbit/s-Schnittstelle im Industrieumfeld

Dr. Heinz Zenkner (freier Mitarbeiter bei Würth Elektronik)

Drahtlose Netzwerke erfreuen sich gerade in Industrieanwendungen großer Beliebtheit. Doch in vielen Fällen bleibt eine robuste Verkabelung per Ethernet die zuverlässigere und sicherere Alternative. Wie sich eine 1-Gbit/s-Schnittstelle einfach implementieren lässt, zeigt dieser Beitrag.

Intelligente Sensoren und Zähler, die effiziente Modulations- und Codierungsverfahren mit guten Ausbreitungseigenschaften und geringen Bandbreiten nutzen, gestatten die Einrichtung industrieller, drahtloser Sensornetzwerke. Die meisten der betrachteten Anwendungsfälle sind jedoch auf Anwendungen mit geringem Datendurchsatz beschränkt. Der tatsächliche Durchsatz beträgt in diesen Anwendungsfällen häufig nicht mehr als 1 Mbit/s.

Ein drahtloses Netz hat keine festen Grenzen. So können beispielsweise selbst kleine Anpassungen der Antennenposition des Zugangspunkts die Signalstärke an den anderen Stationen erheblich beeinflussen. Wände, Decken und Böden dämpfen das Signal, und metallische Gegenstände verursachen Reflexionen. Es kann vorkommen, dass eine Station das Signal eines Zugangspunkts empfangen kann, der Zugangspunkt jedoch das Signal der Station nicht. Darüber hinaus könnte ein Zugriff von außen auf das Netz erfolgen oder die Funksignalübertragung gestört werden.

Daher ist die drahtlose Datenübertragung von Natur aus weniger zuverlässig als über ein kabelgebundenes Netz. So kann es, besonders im industriellen Umfeld zu Situationen kommen, in denen man um ein kabelgebundenes Ethernet-Netzwerk nicht herumkommt.

Kabelgebundenes Ethernet-Netzwerk

Kabelgebundene Netzwerke ähneln den drahtlosen Netzwerken in dem Sinne, dass sie durch den Austausch von Ethernet-Rahmen zwischen Endpunkten funktionieren. Um Probleme beim Aufbau eines Netzwerks zu vermeiden, sind eine Vielzahl von Regeln zu beachten. Die häufigste Ursache für Netzwerkprobleme sind Regelverletzungen. So darf man im Ethernet z.B.

nicht beliebig lange Kabel verwenden. Bei der Kaskadierung, also der Hintereinanderschaltung von Hubs, dürfen nicht beliebig viele verwendet werden und auch eine ungünstig gewählte Netzwerkstruktur kann zu Fehlern im Netzwerk führen oder aber das Netzwerk unnötig belasten. Aber auch abhängig von der Qualität der Kabel und der Leistungsfähigkeit der Hardware lassen sich oftmals die gewünschten Datenraten nicht erreichen.

Gegenwärtig sind 100Base-TX (Fast Ethernet, 100 Mbit/s), Gigabit-Ethernet (1 Gbit/s), 10-Gigabit-Ethernet (10 Gbit/s) und 100-Gigabit-Ethernet (100 Gbit/s) verfügbar geworden. Für die meisten Zwecke funktioniert Gigabit-Ethernet gut mit einem regulären Ethernet-Kabel, speziell mit den Verkabelungsstandards CAT5e und CAT6. Diese Kabeltypen folgen dem 1000BASE-T-Verdrahtungsstandard, auch IEEE 802.3ab genannt.

Die 1-GB-Ethernet-Schnittstelle arbeitet nach dem Standard 802.3ab-1999 (CL40) und benötigt vier Adernpaare / Kanäle zur Signalübertragung. Somit ergibt sich eine Symbolrate von 125 Megabaud (MBd) mit einer Bandbreite von 62,5 MHz pro Kanal (2 bit pro Symbol). Die Signalspannung bei 1000BASE-T (GB-Ethernet) beträgt durchschnittlich 750 mV differentiell, für die Grenzen gilt 820 mV > U_{Signal} > 670 mV bei einer Last von 100 W.

1-Gbit/s-Ethernet-Front-End

Ein typisches Front-End für Ethernet ist mit einer RJ45-Schnittstelle ausgestattet. Diese sind für Vollduplex-Übertragungen ausgelegt, d. h. für eine gleichzeitige Übertragung von Sende- und Empfangsdaten. Dies ist möglich, weil der Steckverbinder zwei Adernpaare umfasst, wobei immer ein Paar pro Richtung benötigt wird (Differenzspannungsprinzip). Für jeden RJ45-Anschluss wird vom IEEE-Standard eine galvanische Trennung per Übertrager gefordert. Dieser Übertrager schützt die Geräte vor Beschädigung durch Hochspannung auf der Leitung und verhindert Spannungs-Offsets, die durch Potenzialunterschiede zwischen den Geräten auftreten können. **Bild 1** zeigt die Prinzipschaltung einer Gigabit-Ethernet-Schnittstelle.

Diskrete Beschaltung der Gigabit-Ethernet-Schnittstelle

Der Ethernet-Transformer (LAN-Übertrager) ist die Schnittstelle zwischen Gerät und dem Ethernet-Kabel. Der Übertrager sorgt für die sicherheitsrelevante galvanische Trennung zwischen Gerät und Kabel und gleichzeitig für die Impedanzanpassung, einerseits zur internen Logik, andererseits an die symmetrischen Adernpaare. Weiterhin schützt der Übertrager das Gerät vor transienten Störungen, unterdrückt Gleichtaktsignale zwischen dem Transceiver-IC und dem Kabel, sowohl vom Gerät nach außen, als auch vom äußeren Kabel zur Elektronik im Gerät. Jedoch muss das Bauteil auch die Daten bis zu 1 Gbit/s breitbandig übertragen ohne das Sende- und das Empfangssignal wesentlich zu dämpfen. Um die Anpassung und die EMV-Anforderungen zu erfüllen sind zusätzliche Komponenten erforderlich.

Eine Beschaltung der Gigabit-Ethernet-Schnittstelle mit diskreten Komponenten zeigt die Schaltung in **Bild 2**. Der LAN-Übertrager sorgt für eine DC-Trennung zwischen der Elektronik und dem Netzwerkkabel. Der mittlere Abgriff der primärseitigen Wicklung zeigt den so genannten „Bob Smith"-Abschluss. Pro Adernpaar wird hier jeweils ein 75-Ω-Widerstand zu einem „Sternpunkt" zusammengeschaltet, das Ganze wird dann

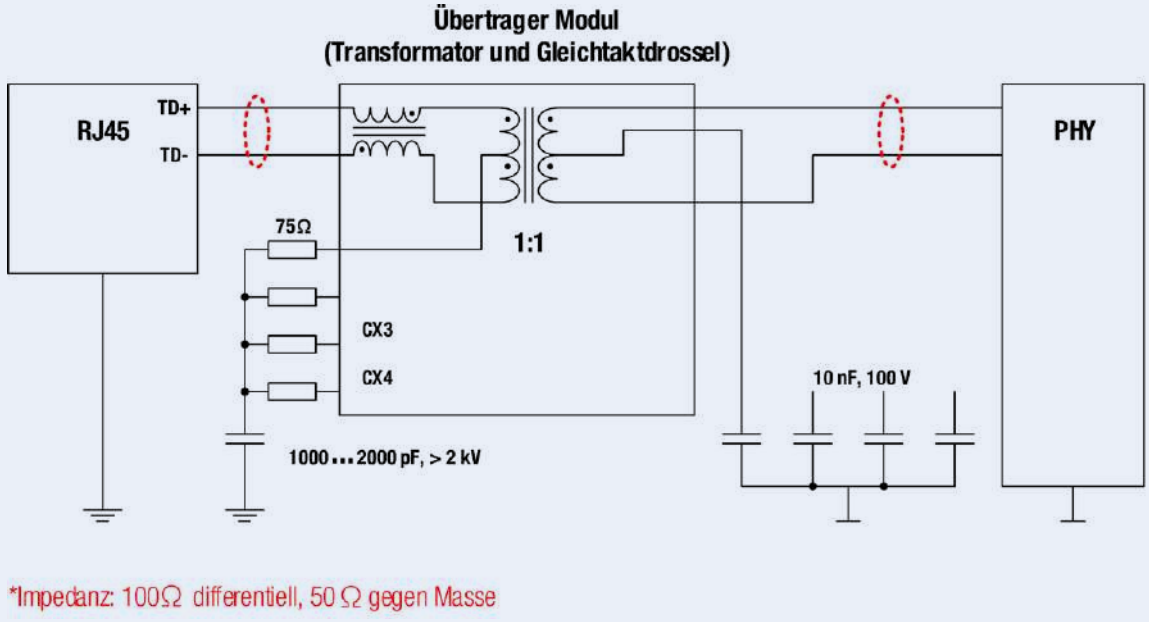


Bild 1. Prinzipschaltung einer Gigabit-Ethernet-Schnittstelle. Darstellung eines Übertragungskanals von insgesamt vier Kanälen.

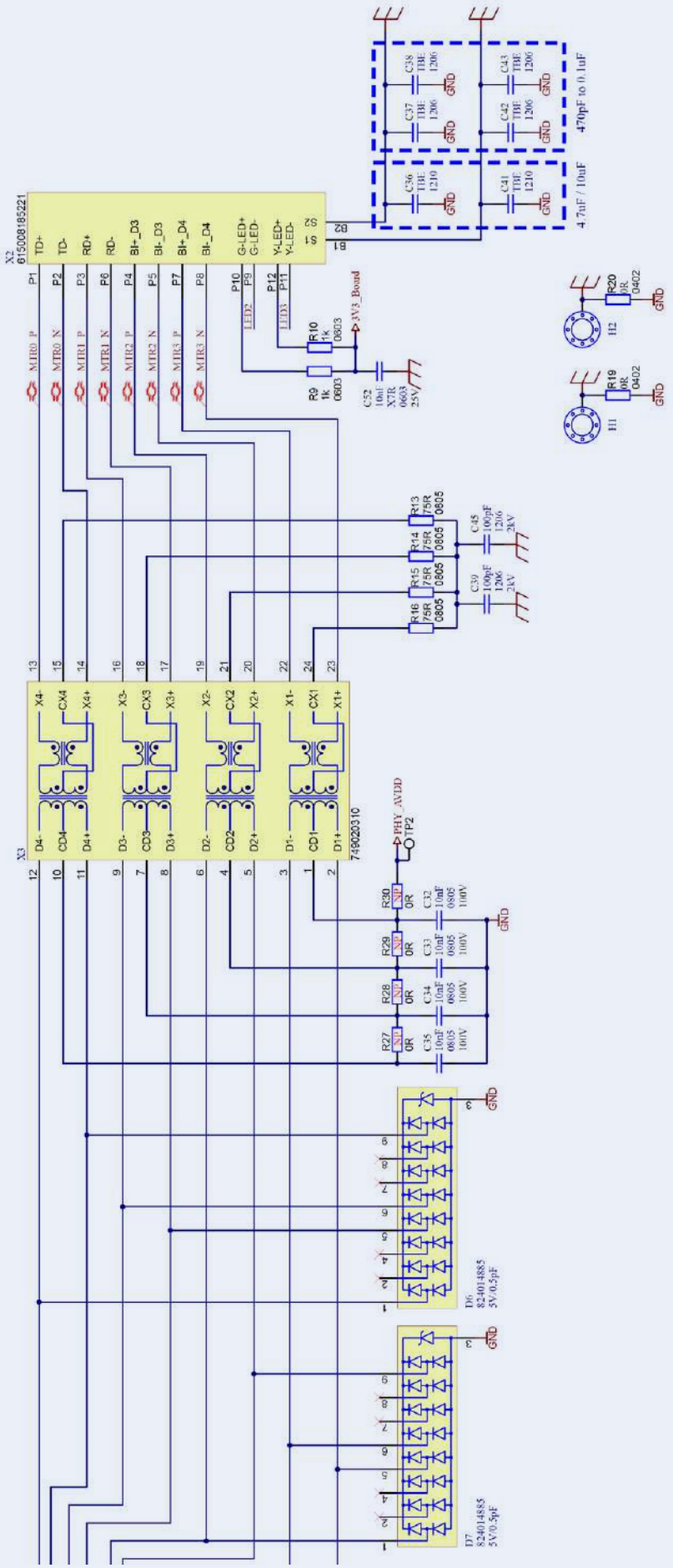


Bild 2. Diskret aufgebaute Schaltung einer Gigabit-Ethernet-Schnittstelle. Das Modul X3 umfasst die LAN-Übertrager sowie Gleichtaktdrosseln gegen unerwünschte Störungen.

galvanisch getrennt und mittels zwei parallel geschalteten 100-pF/2kV- Kondensatoren an die Gehäusemasse angeschlossen. Die zusätzlich im Modul X3 integrierten Gleichtaktdrosseln reduzieren Störungen, die über die langen Ethernet-Kabel sowohl kapazitiv, als auch induktiv eingekoppelt werden und so als Gleichtaktstörungen die Ethernet-Datenkommunikation beeinträchtigen könnten.

R9, R10 und C52 in Bild 2 dienen der Stromversorgung der typischerweise in der Anschlussbuchse integrierten LEDs. Über die Kondensatoren C36 bis C38 und C41 bis C43 kann die Schirmung der Ethernet-Buchse mit der Platinenmasse (GND) verbunden werden. Bei Blechgehäusen ist es sinnvoll diese Kondensatoren nicht zu bestücken und die Masse (GND) der Elektronik direkt über Verschraubungen an das Gehäuse anzuschließen. Bei Gehäusen aus Kunststoff sollten die Kondensatoren bestückt werden, um den Schirm des Ethernet-Kabels an die Bezugsmasse anzuschließen. Den gleichen Zweck haben die 0-Ω-Widerstände R19 und R20. Hier erfolgt jedoch keine galvanische Trennung, wie sie mit den Kondensatoren realisiert wird. Die alternativen Bestückungen wurden hier zu „experimentellen" Zwecken vorgesehen, um die Schirmqualität verschiedener Ethernet-Kabel zu vergleichen. Die Kondensatoren C32 bis C35 auf der Sekundärseite der Übertrager verbinden die Mittelabzapfungen dieser HF-technisch mit Masse (GND). Um DC-Ausgleichsströme vom PHY zu vermeiden ist eine galvanische Trennung über Kondensatoren erforderlich. Die Widerstände R27 bis R30 sind aufgrund von Anforderungen mancher PHY-Hersteller vorgesehen (Current Mode Line Driver – Option), werden aber in der Regel nicht benötigt, wenn der PHY im „Standard Voltage Mode" arbeitet. Unverzichtbar sind hingegen die TVS-Dioden Arrays D6 und D7, die die schnittstellenseitig auftretenden transienten Störungen zum PHY gegen die Schaltungsmasse (GND) begrenzen. Sekundärseitig, d.h. nach den Übertragern des X3-Moduls, treten die transienten Störungen im Gleichtakt auf, deshalb muss an jeden Anschluss der Übertrager jeweils eine TVS-Diode gegen die Bezugsmasse geschaltet werden. Die Störpegel sind auf der Sekundärseite des Übertra-

gers aber geringer, als auf der Primärseite. Wichtig für die Funktion der TVS-Dioden ist ein Anschluss der Dioden mit niedriger Impedanz, einerseits in die Signalleitungen eingeschleift und andererseits zur Masse. Das Layout aller vier Lagen der Platine vom Ethernet-Schnittstellenbereich ist in **Bild 3** gezeigt. Die Gehäuse-/Buchsenmasse zum Elektronik-GND ist in allen vier Lagen getrennt. Die Flächen der Gehäusemasse überlappen somit nicht mit anderen Lagen, um die kapazitive Kopplung so gering wie möglich zu halten. Die Masseflächen wurden im Raster ca. alle 4 mm durchkontaktiert. Die von der Ethernet-Buchse kommenden Signalleitungen sind symmetrisch, mit einer differentiellen Impedanz von 100 Ω gegen die Bezugsmasse geroutet. Die Leiterpärchen weisen eine Leiterbahnbreite von 0,154 mm auf und haben einen Abstand von 0,125 mm zueinander. Die Ethernet-Buchse ist an der Kante der Leiterplatte positioniert, damit ggf. eine Verbindung mit niedriger Impedanz zu einem Metallgehäuse gewährleistet werden kann.

Das Übertrager-Modul (X3) ist in unmittelbarer Nähe platziert, um die elektrischen Koppeleinflüsse, bzw. Beeinträchtigungen durch lange Leiterbahnen gering zu halten. Wie primärseitig, ist auch auf der Sekundärseite des Übertragermoduls bei den Leiterbahnen eine differentielle Impedanz von 100 Ω gegen die Bezugsmasse einzuhalten. Die TVS-Arrays müssen unmittelbar in den Signalpfad und gegen GND angeschlossen werden, um einen Spannungsabfall durch parasitäre Induktivitäten zu vermeiden.

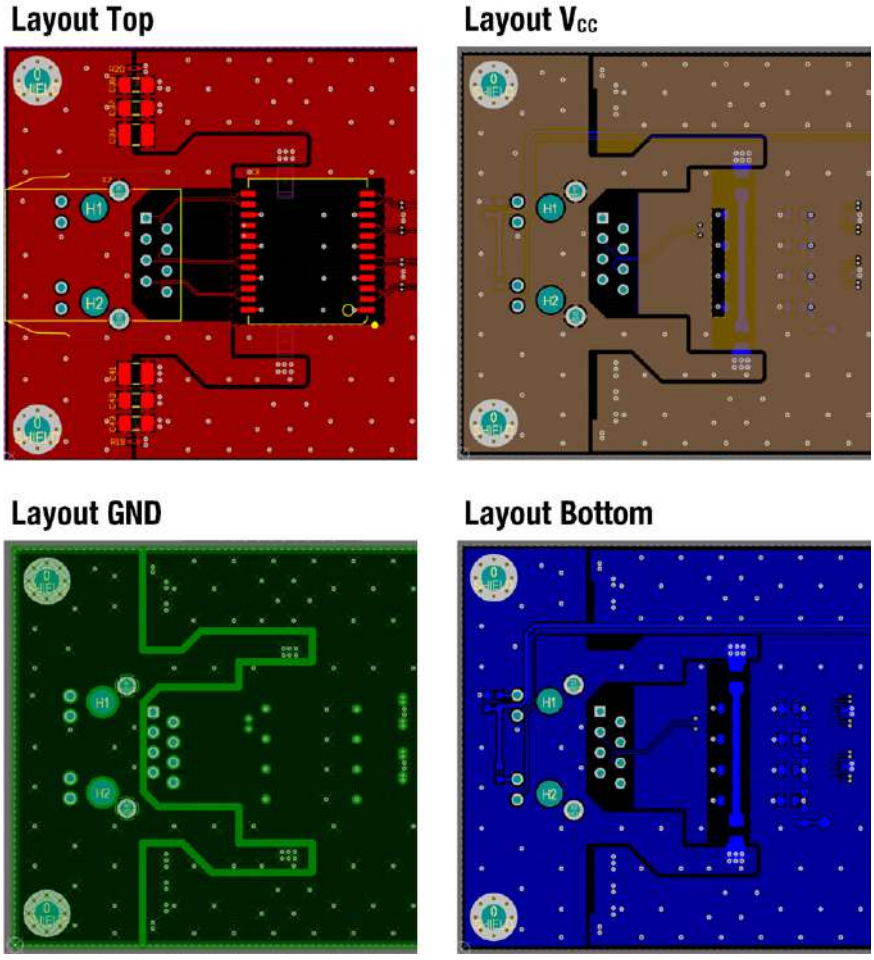


Bild 3. Layout aller vier Platinen-Lagen vom Ethernet-Schnittstellenbereich.

EMV-Konformität

Das Board hält EMV-technisch die Störfestigkeit nach Industriestandard (EN61000-6-2) und die Funkstöremissionswerte nach EN55032 Klasse B für Multimediageräte ein. Für ein erfolgreiches Design einer 1-Gbit/s-Ethernet-Schnittstelle sind zahlreiche Punkte zu beachten, diese sind ein HF-gerechtes Schaltungs- und Layout-Design, ein systemabhängiges Massekonzept und die richtige Wahl der Bauelemente. Nur wenn alle diese Punkte gemeinsam beachtet werden, kann ein zuverlässig funktionierendes Produkt entwickelt werden, dass den hohen Anforderungen gerecht wird. Weitere Informationen zu diesen Aspekten und auch für andere Schnittstellenstandards stehen in diversen App-Notes von Würth Elektronik unter [1] zur Verfügung.

220182-02

Autorenvorstellung

Dr.-Ing. **Heinz Zenkner** ist freier Mitarbeiter bei Würth Elektronik in den Bereichen Technical Marketing und Application Engineering und doziert im Bereich EMV an der technischen Akademie. Gleichzeitig ist Dr. Zenkner öffentlich bestellter und vereidigter Sachverständiger für EMV. Herr Zenkner ist langjähriger Autor verschiedener Fachzeitschriften und Bücher. Zusätzlich hat er an verschiedenen Universitäten, an der IHK und bei zahlreichen Seminaren als Dozent gearbeitet.

WEBLINK

[1] Würth Elektronik Application Guide: www.we-online.com/applicationguide/en

Objekterkennung in Echtzeit für MCUs mit mit Edge Impulse FOMO

Von Jan Jongboom, Edge Impulse

Wir Menschen verlassen uns bei vielen alltäglichen Aufgaben – von den einfachsten bis hin zu den komplexesten – in hohem Maße auf unser Sehvermögen. Mit einem Blick wissen wir, ob sich Menschen im Raum befinden, ob ein Elefant in der Nähe ist oder wie viele freie Parkplätze es gibt. Trotz der Bedeutung des Sehens können viele eingebettete Geräte Dinge nicht visuell wahrnehmen. Wäre es nicht großartig, wenn wir *allen unseren Geräten* beibringen könnten, die Welt so zu sehen wie wir?

In den letzten Jahren gab es einige erstaunliche Entwicklungen im Bereich des maschinellen Sehens, die zu Fortschritten bei Dingen wie selbstfahrenden Autos oder biometrischen Einreisekontrollen geführt haben (sehr nützlich, wenn man wie ich viel reist!). Aber diese Anwendungsfälle sind unglaublich rechenintensiv und erfordern teure Grafikprozessoren oder spezielle Beschleuniger.

Erfreulicherweise sind aber nicht alle Aufgaben für maschinellen Sehen so rechenintensiv. Die Beantwortung einer Ja/Nein-Frage wie „Sehe ich einen Elefanten?“ oder „Ist dieses Etikett richtig auf der Flasche angebracht?“ kann eingebetteten Geräten mit eingeschränkten Möglichkeiten einen enormen Mehrwert verleihen. Darüber hinaus lassen sich diese Probleme der Bildklassifizierung sogar von heutigen Mikrocontrollern lösen.

Stellen Sie sich vor, wir könnten jedes eingebettete Gerät mit noch fortschrittlicheren Bildverarbeitungsfunktionen ausstatten!

Hier kommt FOMO ins Spiel

Wir machen das wahr. Wir haben eine neuartige neuronale Netzwerkarchitektur für die Objekterkennung entwickelt, die wir „Faster Objects, More Objects“ oder FOMO nennen (**Bild 1**). Es wurde von Grund auf so konzipiert, dass es in Echtzeit auf Mikrocontrollern läuft, sodass Entwickler eingebetteter Systeme nicht befürchten müssen, etwas zu verpassen, wenn es um maschinelles Sehen geht.

Schnell, schlank und flexibel

FOMO kann auf einer 32-bit-MCU, z. B. einem Arm Cortex-M7, mit einer Bildrate von 30 Bildern pro Sekunde laufen. Und wenn Sie sich das nächste Mal für ein Gerät der Raspberry-Pi-4-Klasse entscheiden, können Sie Objekterkennung mit etwa 60 Bildern pro Sekunde durchführen. Das ist etwa 30 Mal schneller als Mobile-

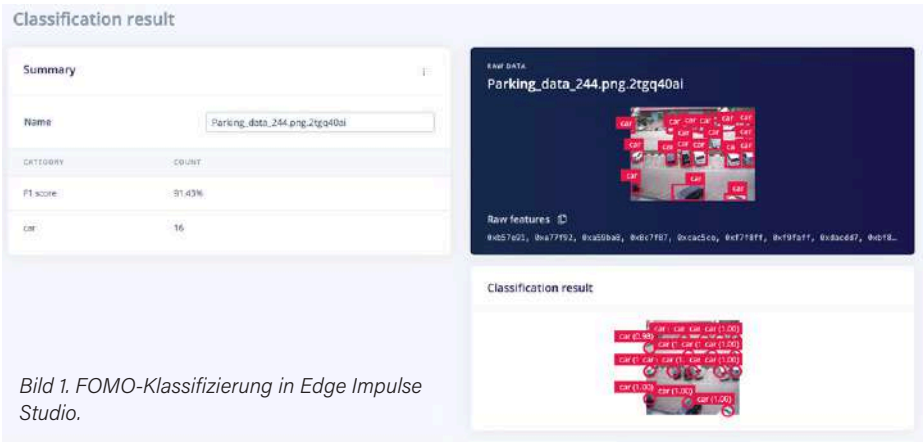


Bild 1. FOMO-Klassifizierung in Edge Impulse Studio.

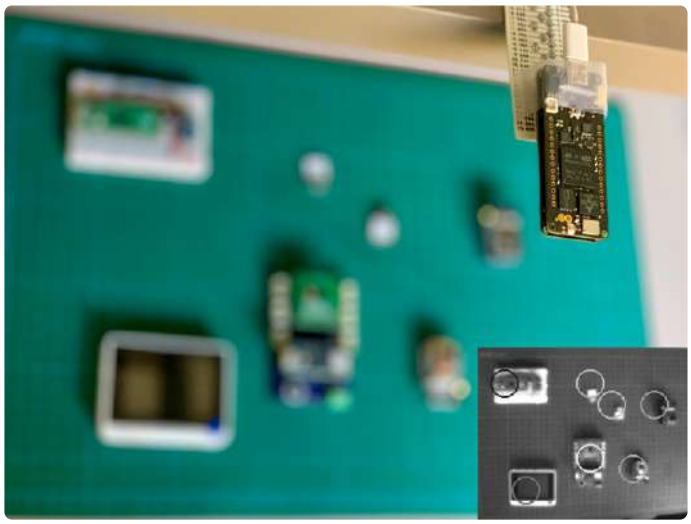


Bild 2. Objekterkennung auf einer Vielzahl von Entwicklungsplatinen, z. B. dem Arduino Portenta.

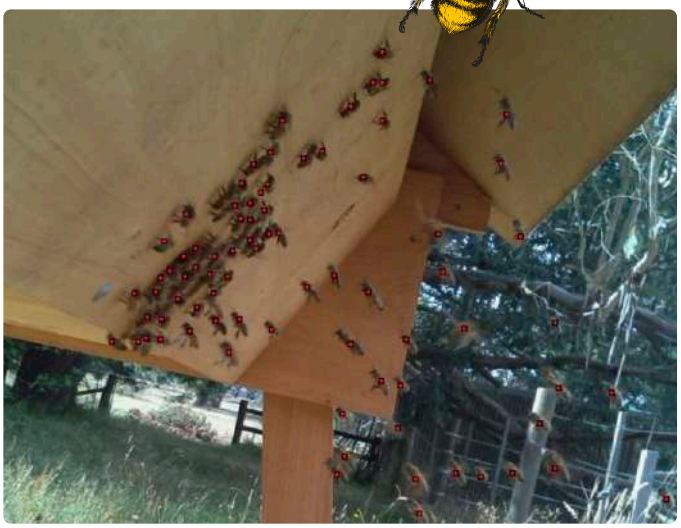


Bild 3. Eine frühere Version der FOMO-Architektur wurde zum Zählen einzelner Bienen verwendet.

Net SSD oder YOLOv5.
FOMO belegt nur etwa 100 KB RAM, was es möglich macht, die Objekterkennung in Echtzeit auf allen Systemen laufen zu lassen, von stark eingeschränkten Arm-Cortex-M4-Kernen bis hin zu leistungsfähigeren wie den Cortex-M7-Kernen auf dem Arduino Portenta H7 (**Bild 2**), dem neuen Arduino Nicla Vision (eine weitere 2-Kern-Arm-Cortex-M7/M4-CPU) oder sogar spezialisierten DSPs wie dem Himax WE-I.
FOMO kann von den kleinsten Mikrocontrollern bis hin zu vollständigen Gateways oder GPUs skaliert werden. Dank dieser hohen Flexibilität ist FOMO auch dann einsetzbar, wenn eine Fehlererkennung die Identifizierung von sehr, sehr kleinen Abweichungen innerhalb eines Bildes erfordert.
Bei einer MCU mit streng begrenzter Rechen- und Speicherkapazität ist es am besten, eine Bildgröße von etwa 96x96 Pixeln zu verwenden. Aber mit einem größeren Mikrocontroller sind 160x160 Pixel wahrscheinlich problemlos machbar. Wichtig ist, dass FOMO vollständig faltungsbasiert ist und daher mit jeder beliebigen Bildgröße funktioniert. Wenn Sie eine höhere Granularität, mehr Details oder mehr Objekte benötigen, können Sie einfach die Eingangsauflösung erhöhen.

Es sieht auch kleine Dinge

Solange die Objekte im Bild eine ähnliche Größe haben und nicht überlappen, kann diese neue Architektur sogar viele sehr kleine Objekte sehr effektiv erkennen und zählen (**Bild 3**). Das ist etwas, das MobileNet SSD und YOLOv5, obwohl sie größere und leistungsfähigere Modelle sind, nicht sehr gut können.

Nichts mehr verpassen

FOMO ist bereits erhältlich, läuft auf einer Vielzahl von Computerplattformen und ist mit Linux-Systemen, Cortex-M-Mikrocontrollern und spezialisierten DSPs kompatibel. Sie brauchen nur noch eine Kamera und Edge Impulse, und schon sind Sie bereit.
Mit FOMO können Sie nahezu jedes kamerabasierte Gerät schnell mit einer Objekterkennung ausstatten und das Risiko vermeiden, etwas zu übersehen. Eine Gefahr, mit der sich Embedded-Entwickler bisher im Bereich des maschinellen Sehens herumschlagen mussten (**Bild 4**).

Um mehr über FOMO zu erfahren und mit Ihrem eigenen Algorithmus zu experimentieren, besuchen Sie **edgeimpulse.com/fomo**.

220207-02

Bild 4. Training mit Centroids bei Bierflaschen: Oben die Quell-Label, unten das Ergebnis der Inferenz.

Über den Autor

Jan Jongboom ist Ingenieur für eingebettete Systeme und Förderer des maschinellen Lernens, immer auf der Suche nach Möglichkeiten, mehr Informationen aus der realen Welt zu gewinnen. Er hat Geräte ausgeliefert, an den neuesten Netzwerktechnologien gearbeitet, Mikrocontroller simuliert, und es gibt sogar ein Denkmal in San Francisco, das seinen Namen trägt. Derzeit ist er als Mitgründer und CTO von Edge Impulse tätig, der führenden Entwicklungsplattform für eingebettetes maschinelles Lernen mit über 80.000 Projekten.

NB-IoT – ein Überblick

Quelle: shutterstock.com

Von **Tam Hanna** (Slowakei)

Neugierig auf das schmalbandige Internet der Dinge (NB-IoT)? Vielleicht ist das etwas für Sie. Finden Sie es heraus!

Neben LoRa und Sigfox bieten sich natürlich auch die Mobilfunknetze an, IoT-Sensordaten zu übertragen. Der Umstieg von EDGE auf UMTS war hierbei von Vorteil: Die Verwendung eines schnelleren Sendesystems kann unterm Strich besser sein als die eines energiesparsameren, aber langsameren. Die immensen Bandbreiten- und Energiebedürfnisse von 4G/LTE aber sorgen dafür, dass diese alte Gleichung nicht mehr so wirklich gilt. Der Stromverbrauch der Transmitter ist wesentlich höher, dazu kommen höhere Preise der Module. Dennoch kann es sich lohnen.

Im Rahmen der allgemein und unwissenschaftlich als LTE bezeichneten Spezifikation *3GPP Release 13* spezifizierte die *GSM Association* gleich zwei Systeme für das Internet der Dinge – einerseits *Narrowband Internet of Things* (NB-IoT), andererseits auch *LTE-M*, das auch als *LTE Cat-M1* oder *eMTC* bezeichnet wird.

Während LTE-M im Prinzip nur ein „kleineres“ LTE (= 4G) mit einer Bandbreite von 1,4 MHz ist, handelt es sich bei NB-IoT um einen dedizierten Funkstandard für das Internet der Dinge. Der wichtigste Unterschied zeigt sich darin, dass LTE-M über VoLTE auch die Sprachübertragung unterstützt, während ein NB-IoT-System ausschließlich Datenmitteilungen überträgt.

NB-IoT nutzt in seinen nur 180 kHz breiten Kanälen eine Untermenge der in der LTE-Vollversion implementierten Verfahren. Im Uplink kommt eine einfache Version der *Frequency Division Multiple Access* (FDMA) zum Einsatz, für den Downlink setzt man auf *Orthogonal FDMA* (OFDMA). Das verwendete Modulationsverfahren *Quadrature Phase Shift Keying* (QPSK) benötigt aus rechentechnischer Sicht keinen großen Hardware-Aufwand.

Angemerkt sei noch, dass die Einführung von NB-IoT für den Carrier im Allgemeinen „nur“ die Kosten für die neue Hardware

Tabelle 1. Frequenzbänder.

Region	Bänder
Europea	3, 8, 20
(Ehemalige) GuS-Staaten	3, 8, 20
Nordamerika	2, 4, 5, 12, 66, 71, 26
Asien, Pazifik (APAC)	1, 3, 5, 8, 18, 20, 26, 28
Sub-Sahara-Afrika	3, 8
Naher Osten und Teile Nordamerikas	8,20
Lateinamerika	2, 3, 5, 29

verursacht. Aufgrund der extrem geringen Bandbreite passt NB-IoT komfortabel in das *Guard Band*, das die LTE-Frequenzpakete umfasst. Andererseits ist es natürlich auch erlaubt, NB-IoT im Standalone-Betrieb zu nutzen.

Performance im Blick

Der technisch interessanteste Funkstandard hilft nichts, wenn die Übertragungsleistung nicht für die vorgesehene Aufgabe ausreicht. Im Fall von NB-IoT kommt es auf die Version an, denn es gibt Unterschiede zwischen LTE Cat NB1 (Release 13) und LTE Cat NB2 (Release 14). Während die ältere Version im Upstream nur 26 kbit/s schafft, ist Cat NB2 mit 127 kbit/s im Upstream und 159 kbit/s im Downstream deutlich schneller. Zum Vergleich: Klassisches, also Nicht-HSDPA-3G kam in der Anfangszeit auf 380 kbit/s. LTE Cat M1 arbeitet derweil im Up- and Downstream mit rund 1 Mbit/s, Release 14 erweitert auf 4 Mbit/s Upstream und 7 Mbit/s Downstream.

Massiv sind die Unterschiede im Bereich der Latenz: Während LTE-M meist mit 15 ms auskommt, wird bei NB-IoT als „Arbeitsband" 1,6 s bis 10 s (!) avisiert. Der Modulhersteller Sierra Wireless, der vor allem in den USA beliebt ist, fasst die Lage folgendermaßen zusammen [1]:

„Another important fact to consider is that there are no NB-IoT use cases that LTE-M can't also support. In other words, LTE-M supports any LPWA application, whereas NB-IoT is designed for simpler static sensor type applications."

Nur in der Version 2 des NB-IoT-Standards findet sich zudem eine Unterstützung für vom Netzbetreiber angebotene Positionierung. Wenn das Modul über kein GPS verfügt oder man eine externe Antenne einsparen möchte, lassen sich auf diese Weise grundlegende Positionsdaten erhalten. Release 14 beschleunigt zudem die Zellen-Neusuche, was besonders für in Bewegung befindliche Geräte von Vorteil ist. Trotz dieser neuen Vorteile von Cat NB2 bleibt LTE-M erste Wahl für Automotive und Co., da die Zellenübergabe hier „intelligenter" gelöst ist. Die letzte Verbesserung betrifft die Sendeenergie: Nur in Release 14 sind Super-Low-Power-Sender erlaubt, die ihre Arbeit mit nur 14 dBm erledigen [2].

Wer einmal ein 4G-Modul für Verizon entworfen hat, fragt an dieser Stelle instinktiv nach den verwendeten Bändern. Das nur für Nordamerika wichtige Band 13 hat schon mehr als einem asiatischen oder europäischen Modulanbieter Probleme verursacht. Im unter [3] bereitstehenden *Deployment Guide* der GSM Association findet sich die **Tabelle 1.** Achten Sie darauf, dass Ihr auserwähltes Modul alle Bänder unterstützt, die ihr Wunsch-Carrier verwendet!

Verfügbarkeit und Verträge

Funkstandards haben logischerweise nur dann Sinn, wenn sie auch praktisch verfügbar sind. Im Fall der beiden IoT-Funkstandards empfiehlt sich die interaktive Weltkarte der GSM Association in **Bild 1** [3] (Stand September 2021). Mexiko ist dabei das einzige Land, wo es (wohl wegen der höheren Reichweite) nur CAT-M gibt; insbesondere in den Flächenstaaten Asiens, witzigerweise aber auch in Osteuropa ist „NB-IoT Only" weiter verbreitet. In den hochindustralisierten Regionen Europas, Nordamerikas, Asiens und Australien und Ozeaniens stehen beide Varianten zur Verfügung.

CAT-M-Verträge sind dabei im Allgemeinen gewöhnliche Verträge, bei denen der Volumenverbrauch und die Anzahl der SIM-Karten über die Gesamtkosten entscheiden. Der Vollständigkeit halber sei angemerkt, dass ein IoT-Provider wie *PodGroup* einer im Handel frei gekauften Prepaid-SIM in Kostenfragen oft überlegen ist.

Dass NB-IoT keinen Einschränkungen bezüglich des Duty Cycles unterliegt, bestätigt sich sich in der Beratungspraxis des Autors übrigens nicht. Spricht man mit dem Mobilfunkanbieter über IoT-Connectivity, ist und bleibt es Verhandlungssache und man bekommt nur allzu bald temporale Paket-Mengengrenzen auferlegt. Genaue Bestimmungen legen die Betreiber allerdings nur recht selten offen, weshalb das folgende Statement von T-Mobile USA [4] Seltenheitswert genießt:

„Join the first nationwide NB-IoT network to power asset tracking, connected cities, and more. Limited time offer; subject to change. Taxes and fees may be additional. Plan includes 10 single-packet transactions per hour at up to 64 Kbps, up to 12 MB. Full service payment due at activation."

Interessanterweise handelt es sich dabei um eine Einzelmeinung – von Seiten der Hutchison Holding Ltd wurde bestätigt, dass man (bei im Rahmen bleibendem) Datenverkehr auch das gesamte Volumen an einem Tag verfeuern darf. Tom Tesch, der österreichische Pressesprecher von Hutchison, meint dazu:

„Die Datenrate von NB-IoT ist – entsprechend des Standards – sehr gering und vorrangig geeignet, um einzelne Messwerte oder Stati zu übertragen. Aus diesem Grund werden für NB-IoT Geräte sehr selten mehr als 5-10 MB pro Monat benötigt. Für bandbreitenintensive Anwendungen – zum Beispiel Übertragung von Bildern oder Videos – sind 3G/4G und natürlich 5G die geeignetere Technologie. Aktuell gibt es keine Begrenzungen, wann das Volumen genutzt werden darf/ kann – es kann daher auch alles an einem Tag verbraucht werden."

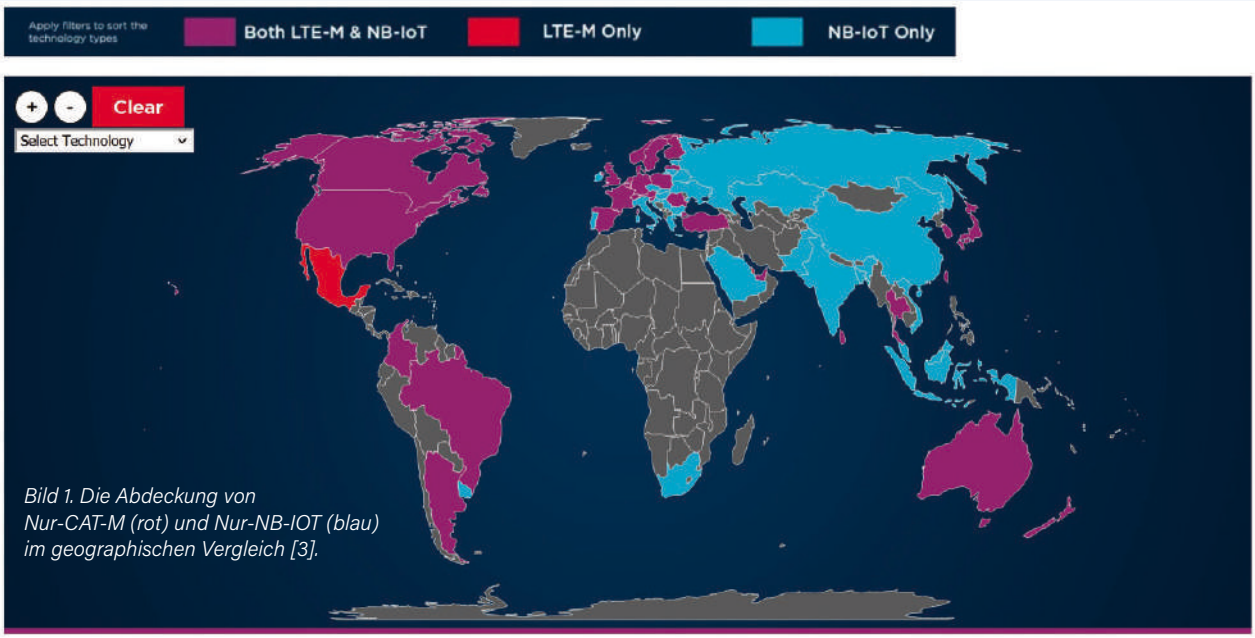


Bild 1. Die Abdeckung von Nur-CAT-M (rot) und Nur-NB-IOT (blau) im geographischen Vergleich [3].

Wie beginnen?

Nach diesem grundlegenden Überlegungen ist es an der Zeit, darüber nachzudenken, wie man NB-IoT in praktische Systeme einbinden kann. Logischerweise ist die Entwicklung eines „hauseigenen" Modems für das durchschnittliche Unternehmen nicht machbar - das „Design in" von Funkmodulen haben wir Ihnen in der Vergangenheit, beispielsweise in Elektor 5-6/2021 [5], detailliert vorgestellt. Falls Sie nicht sofort mit der Entwicklung eines eigenen Boards beginnen möchten, bietet sich - die Verfügbarkeit von Qualcomm-Chips erweist sich hierbei als Problem - die Verwendung „schlüsselfertiger" Evaluation-Boards an.

Neben dem *NBIOT-BG96-SHIELD* von Avnet, das ein BG96-Modul von Quectel integriert, gibt es mit dem *5G NB IoT click* ein Board von Mikro-Elektronika, das ein Cinterion-Modul zur Verfügung stellt. Arduino schickt mit dem *MKR NB 1500* ebenfalls eine kleine Evaluationsplatine ins Rennen - beide Boards kosten allerdings mehr als 50 Dollar. Mag ein Evaluation-Board in vielen Fällen noch mit einer SIM ausgeliefert werden, so ist das massive Rollout von auf NB-IoT basierenden Lösungen alles andere als einfach. Ursache dafür ist, dass Netzbetreiber die Technologie noch nicht für Endkunden „paketiert" haben. Dies wird von den Betreibern auch offen zugegeben, wie das Statement von Hutchison zeigt:

„NB-IoT ist ein sehr junges und neuartiges Netzwerk. Da es kaum noch Geräte am Markt gibt, handelt es sich bei der Zielgruppe vorwiegend um Geschäftskunden aus der Hardware- (und Software-)Entwicklung. Das heißt, dass unser Angebot sich aktuell daher auch ausschließlich an Geschäftskunden richtet, für die wir in Zuge einer Beratung ein individuellen Angebot erstellen."

Bei der Arbeit mit „gewöhnlichen" 2G/3G/4G-Systemen bietet die Nutzung von „virtuellen" Mobilfunkanbietern wie der PodGroup einen Ausweg. Eine diesbezügliche Anfrage wurde dahingehend beantwortet, dass sich NB-IoT derzeit - insbesondere für „globale" Lösungen, die mit einer einzelnen SIM-Karte auskommen müssen - noch nicht wirklich eignet.

Ursache dafür ist, dass erstens das NB-IoT-Rollout selbst nur vergleichsweise eingeschränkt ist und zweitens die Roamingverträge zwischen den verschiedenen Netzbetreibern im Allgemeinen (noch) nicht an den neuen NB-IoT-Funkstandard angepasst wurden. Analog zum Doppelbesteuerungsabkommen zwischen Staaten gilt auch hier, dass solche Anpassungen sehr viel Zeit in Anspruch nehmen. „Internationales" NB-IoT-Roaming steckt also noch in den Kinderschuhen!

Lohnt es sich?

Die Suche nach einem praktischen Modul, das nur NB-IoT anbietet, ist eine durchaus haarige Angelegenheit. Bei Quectel gibt es selbst bei der kleinsten BC660-Serie zwei Varianten: die eine nur mit NB-IoT, die andere mit eMTC und NB-IoT. Auch in größeren Systemen wie dem Bestseller BG95/BG96 sind beide Funkstandards vertreten. Gängige Marktpreise für die Module findet man nur bei SoS Electronic: das BC660K-GL kostet 7,63 € in Einzelstückzahlen, die Version mit LTE-M und NB-IoT ist dort nicht im Programm. Ein BG96 kostet dort 19 €.

Ergiebiger wird die Jagd bei u-blox [6]. Mit der SARA-N3-Modulfamilie steht ein ausschließlich für die NB-IoT-Protokollfamilie vorgesehenes Modul zur Verfügung, einen reinen CAT-M-Chip bieten die Schweizer indes nicht an (siehe **Bild 2**).

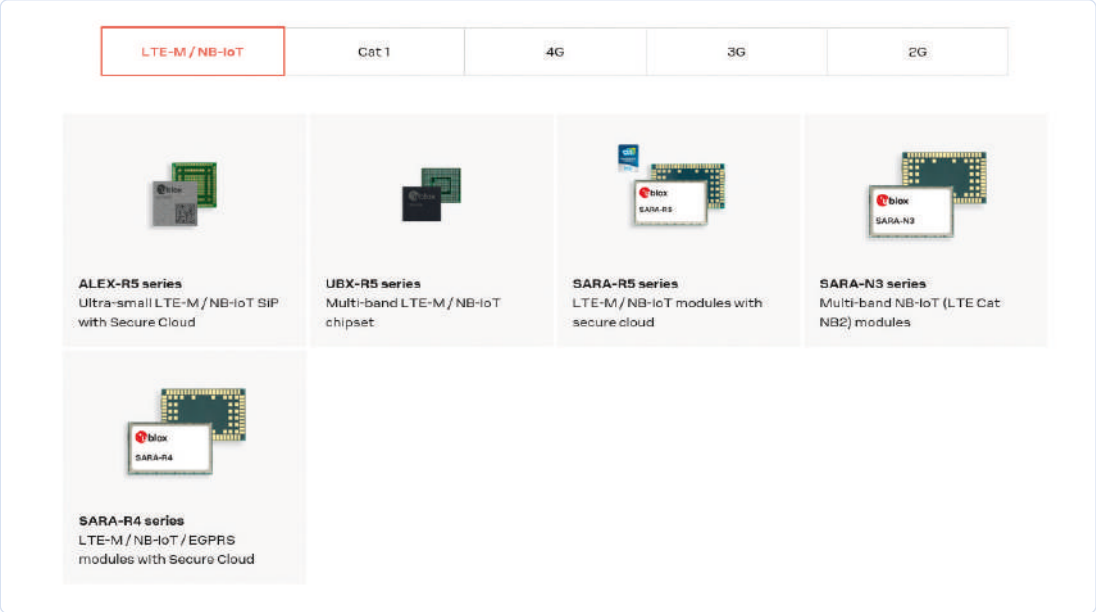


Bild 2. Die Suche nach einem reinen CAT-M-Modul ist unergiebig [6].

Bild 3. Traditionelles steierisches Windrad als Vogelscheuche (Quelle: Martin Geisler, CC BY-SA 4.0 [7]).

Bei Gemalto – seit der Übernahme durch Thales ist die Webseite verwirrender als je zuvor – gibt es mit dem EMS31 ein reines CAT-M-Modul, während der ENS22 ein im selben Formfaktor gehaltenes reines NB-IoT-Modul ist. Der tschechische Distributor Sectron ermöglicht einen Vergleich der Kosten: der EMS31 kostet 14 €, der ENS22 dagegen nur 8 €.
In den als *Hardware Interface Description* bezeichneten Datenblättern finden sich dann auch (verklausulierte) Informationen zum Stromverbrauch. Der höchstmögliche Stromverbrauch des EMS31 tritt bei der Arbeit im Band 4 auf und liegt (bei 3,8 V Versorgungsspannung) bei 239 mA. Für den ENS22 werden im Band 28 als Höchststrom 404 mA aufgeführt, wobei allerdings zu beachten ist, dass Funkmodule diese Peaks oft nur für sehr kurze Zeit benötigen.

Was springt für Sie heraus?

NB-IoT funktioniert aus technischer Sicht problemlos; hat man einen Carrier unter Vertrag genommen, so beschränkt sich der Aufwand für den Netzbetrieb (anders als bei einem hauseigenen

WEBLINKS

[1] LTE-M vs. NB-IoT: What are the Differences?: www.sierrawireless.com/iot-blog/lte-m-vs-nb-iot/
[2] Wikipedia über Schmalband-IoT: https://en.wikipedia.org/wiki/Narrowband_IoT
[3] GSMA: Weltkarte der IoT-Funkstandards: www.gsma.com/iot/deployment-map/
[4] T-Mobile über Schmalband-IoT: https://t-mo.co/3EC5Jo4
[5] Tam Hanna, Keine Angst vor dem Mobilfunkmodul, ElektorMag 5-6/2021: www.elektormagazine.de/200062-02
[6] Mobilfunkmodule von u-blox: www.u-blox.com/en/cellular-modules
[7] Klapotetz: https://bit.ly/3nOr0Fb

LoraWAN) auf einen Anruf beim Anwalt. Der vergleichsweise geringe Spitzen- und Ruhestromverbrauch der Module trägt zudem zu geringen Kosten für die Energieversorgung bei.

Ob sich die Sache am Ende wirklich lohnt, ist – analog zu Parade-Steueroasen wie Dubai oder Monaco – vor allem eine Sache der Skalierung. Wer fünf Modems im Jahr kauft, fährt mit einem „vollen“ und vielleicht um ein paar Euro teureren 4G-Modul samt dickerem Schaltregler besser: Die traurige Lebenserfahrung des Autors zeigt nämlich, dass man in der Praxis immer wieder den „anderen“ Funkstandard braucht, schon deshalb, weil am Ende doch nicht jeder Funkturm jeden Funkstandard unterstützt.

Beim Kauf von 50.000 Modems, die alle an einen Kunden gehen, sieht die Sache natürlich anders aus. Braucht das Bürgermeisteramt von *Großdorf am Klapotetz* (**Bild 3**) NB-IoT, so wird der lokale Carrier wahrscheinlich aufrüsten – und die eingesparten Dollars helfen bei der großen Stückzahl.

180021-02

Sie haben Fragen oder Kommentare?

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

Ein Beitrag von

Text: **Tam Hanna**
Redaktion: **Rolf Gerstendorf**
Layout: **Giel Dols**

PASSENDE PRODUKTE

- **E-Book: *IoT Home Hacks with ESP8266* (SKU 19159)**
  www.elektor.de/19159

Drehspul**relais**

Bemerkenswerte Bauteile

Von **David Ashton** (Australien)

In dieser Ausgabe befassen wir uns mit einem wirklich merkwürdigen Bauteil. So merkwürdig, dass außer seinen physikalischen Eigenschaften nicht viel darüber herauszufinden war. Aber trotz des Aufbaus mit einer Drehspule sind wir ziemlich sicher, dass es sich nicht um ein Messgerät handelt!

Wenn man Drehspule sagt, denken die meisten von uns sicher sofort an ein Messgerät. Auch wenn sie inzwischen etwas aus der Mode gekommen sind, haben die meisten Leute schon mal eines gesehen oder benutzt. Ich spreche jedoch von Drehspulrelais. Stellen Sie sich vor, Sie schließen einen Draht an die Nadel eines solchen Messgeräts an und verbinden einen weiteren mit dem Endanschlag. Wenn das Messgerät den vollen Skalenausschlag erreicht, wird der Kontakt hergestellt. So funktionieren im Grunde diese Relais.

Ich habe ein solches Drehspulrelais vor einigen Jahren in einer Schalttafel gefunden, und es hat einige Zeit gedauert, bis ich herausgefunden habe, was es überhaupt ist. Sie sind nicht nur wunderschön, sondern gehören auch zu den empfindlichsten elektrischen Bauteilen, die ich je gesehen habe.

Sie wurden von BBC Goerz Electro hergestellt. Obwohl BBC Goerz viele professionell aussehende Test- und Messgeräte herstellte, ist es nicht einfach, Informationen über dieses Unternehmen zu finden. Wir wissen jedoch, dass Goerz ein österreichisches Optikunternehmen war, dass BBC Brown Boveri Corporation bedeutet und ein schweizerisches Elektrounternehmen war. Die Firmen haben im Wandel der Jahrzehnte viele Veränderungen, Umfirmierungen und Fusionen durchgemacht. Um darüber zu berichten, fehlt hier der Platz, aber Sie können gerne die interessante Historie einer mehr als 120-jähigen paneuropäischen Industriegeschicht im Internet erforschen.

Trotz dieses Hintergrundwissens über den Hersteller konnte ich keine Informationen über diese Relais finden, abgesehen von jemandem, der einige auf eBay für etwa 100 $ pro Stück verkauft. Sie sind auf einer 8-poligen Röhrenfassung aufgebaut. Als ich sie zum ersten Mal zu Gesicht bekam, fragte ich mich, ob es sich um eine Art Röhre handeln könnte. Sie besitzen jedoch ein durchsichtiges Kunststoffgehäuse, das sich abschrauben lässt, so dass man den komplizierten Mechanismus im Inneren genau betrachten kann. Die Typennummer 91041-2 ist deutlich auf dem Gehäuse aufgedruckt, zusammen mit einer Seriennummer (**Bild 1**). Außerdem befindet sich an der Seite ein Anschlussplan, der das Testen der Geräte erleichterte (**Bild 2**).

Ich habe eine Prüfvorrichtung für diese Relais gebaut - praktischerweise habe ich drei passende Fassungen auf einer Halterung

Bild 1. Die Relais, hier von hinten gezeigt, mit ihrer Typennummer auf dem Kunststoffgehäuse.

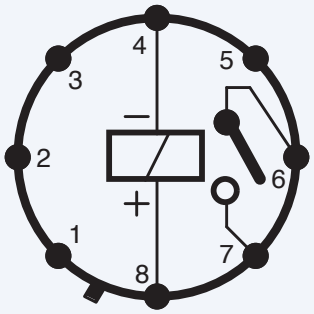


Bild 2. Dieser Anschlussplan ist auf der Seite des Relais aufgedruckt.

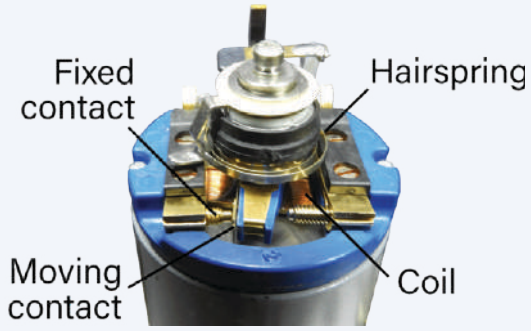


Bild 3. Auf dem Foto ist die Lage der Spiralfeder, der Spule und der Kontakte eingezeichnet.

mit vielen Löchern für die Montage anderer Bauteile. Ich habe ein 470-kΩ-Potentiometer in Reihe mit der Relaisspule und einer 12-V-Spannungsversorgung verwendet und den Kontakt zur Ansteuerung einer LED genutzt, damit ich sehen konnte, wann er sich schließen würde. Die Relais arbeiten mit einem Strom von nur 260 µA bei einer Spannung von 113 mV! Zeigen Sie mir ein anderes Relais, das so empfindlich ist!

Die Kontakte sind klein (**Bild 3**), und man könnte sie zum Betätigen eines anderen Relais verwenden, um eine sinnvolle Leistung zu schalten. Heutzutage würde man zu diesem Zweck natürlich einen Optoisolator verwenden, was mich zu der Annahme bringt, dass diese Relais ziemlich alt sind, aber man bräuchte immer noch etwas mehr Elektronik, um solche Spezifikationen zu erzielen. Wahrlich, diese Relais sind wirklich sehr eigenartig!

210557-02

Haben Sie Fragen oder Kommentare?

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

Dragino LPS8
Indoor Gateway

Schnell zum eigenen LoRaWAN-Gateway

Bild 1. Dragino LPS8 Indoor Gateway (Quelle: Dragino [5]).

Von Mathias Claußen (Elektor)

Wie man eigene Elektronik mit einem LoRaWAN verbinden kann, haben wir schon des Öfteren beschrieben. Wer sich nicht in Reichweite eines LoRaWAN-Gateways befindet oder einfach noch tiefer ins Thema einsteigen will, kann ein eigenes Gateway betreiben. Wir haben das mit dem günstigen Dragino LPS8 Indoor Gateway ausprobiert.

LoRaWAN ist ein Thema, das schon öfter in Elektor behandelt wurde. Es ist recht einfach, einen LoRaWAN-Knoten wie zum Beispiel ein Sensorboard aufzubauen. Dazu stehen LoRa-Module zur Verfügung, die man an kompakte Controllerboards anschließen kann, etwa auf Basis eines STM32 oder Raspberry Pi Pico [1][2]. Doch damit die Daten, die ein solcher Knoten über LoRa sendet, auch weiter transportiert werden, braucht es eine Gegenstelle; in diesem Fall ein LoRaWAN-Gateway, das Daten über LoRa entgegennimmt und an eine Internetplattform wie *The Things Network* weiterleitet. Man kann hierzu ein vorhandenes Gateway nutzen (viele werden von Freiwilligen betrieben) oder ein eigenes Gateway einrichten. Ich habe seit mehr als einem Jahr ein *Dragino LPS8 Indoor Gateway* im Einsatz.

Dragino LPS8

Das *Dragino LPS8 Indoor Gateway* (**Bild 1**) besitzt ein Kunststoffgehäuse. Die Elektronik im Inneren wird von einem kleinen AR9331-Prozessor von Atheros (heute Qualcomm) mit 400 MHz befeuert. Mit 64 MB RAM und 16 MB Flash ist dieses Gehirn des Gateways sicher nicht mit einem Raspberry Pi Zero 2 W zu vergleichen - doch für die Funktionen, die ein Gateway ausführen muss, ist der Prozessor mehr als ausreichend. Der Atheros AR9331 integriert neben der CPU auch WLAN nach 802.11 b/g/n und einen 10/100-Mbit-LAN-Port. Für die Datenrate, die per LoRaWAN transportiert werden soll, sind diese Geschwindigkeiten mehr als ausreichend. Auch muss das Gateway selbst keine große Rechenleistung bereitstellen, da es sich nur um das Management des integrierten LoRa-Transreceiver-Moduls kümmert und die Daten ins Internet weiterreicht. Ein Blockdiagramm ist in **Bild 2** zu sehen.

Beim LoRa-Transreceiver handelt es sich um eine Kombination aus einem LoRa-Baseband-Chip SX1308 von Semtech (Blockdiagramm in **Bild 3**) und zwei SX1257

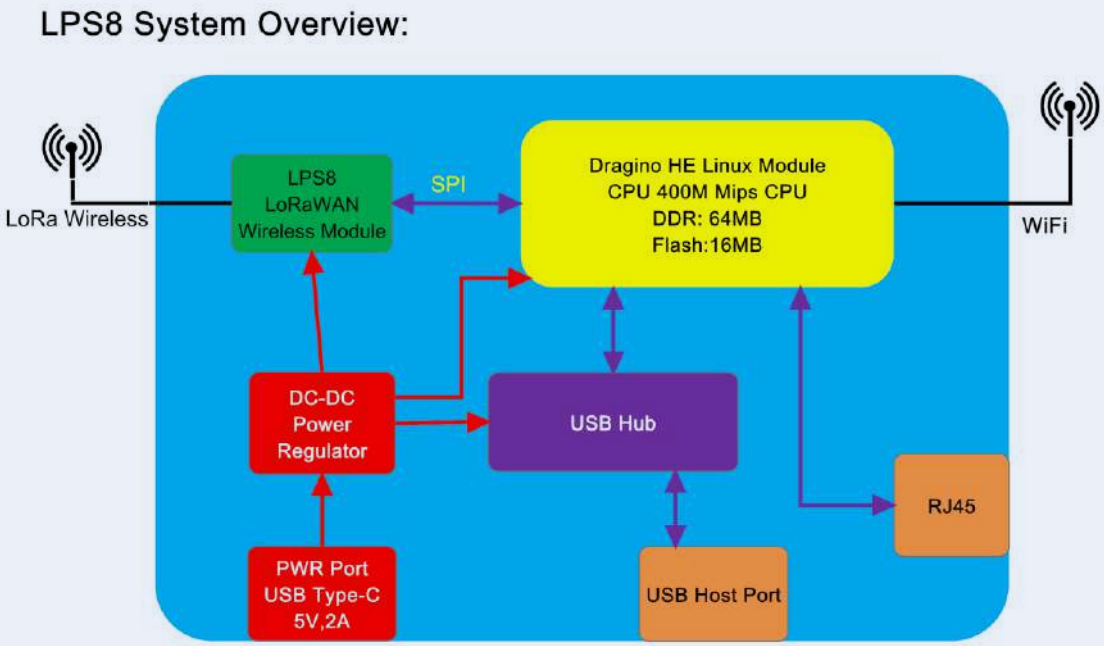


Bild 2. Blockdiagramm Dragino LPS8 (Quelle: Dragino [6]).

Front-End-Modulen (Blockdiagramm in **Bild 4**). Diese Kombination sorgt für die Umsetzung von der Funkschnittstelle zu Ethernet. Das Gateway ist sehr genügsam, was die Leistungsaufnahme angeht, 5 V und maximal 2 A (10 W) müssen von einem Netzteil bereitgestellt werden.

Wie der Name des Gateways schon verrät, ist das Gerät für den Einsatz innerhalb eines Gebäudes gedacht, die Umgebung sollte also trocken und staubarm sein. Das Gebiet, das mit dem Gateway abgedeckt werden kann, ist kleiner als eines, das ein Outdoor-Gateway abdecken kann. Letztere werden meist auf entsprechenden Sendemasten installiert.

Handbuch, Firmware und Einrichtung

Zur Einrichtung des Gateways kann das Handbuch von Dragino in aktueller Version [3] heruntergeladen werden; es wurde seit dem Release des Produktes gepflegt und spiegelt die Funktionen und Updates der aktuellen Firmware wider. Löblich - es könnte einen traurig stimmen, dass sich nicht auch andere Firmen um eine solch gute Produkt-Dokumentation kümmern.

Auch die Firmware wird gut gepflegt. Das aktuelle Release ist vom 4.11.2021 (Stand 15.12.2021) [4]. Wenn möglich, empfiehlt sich vor der Inbetriebnahme ein Firmwareupdate auf die aktuelle Version, damit Bugs und Sicherheitslücken so weit wie möglich entgegengewirkt wird.

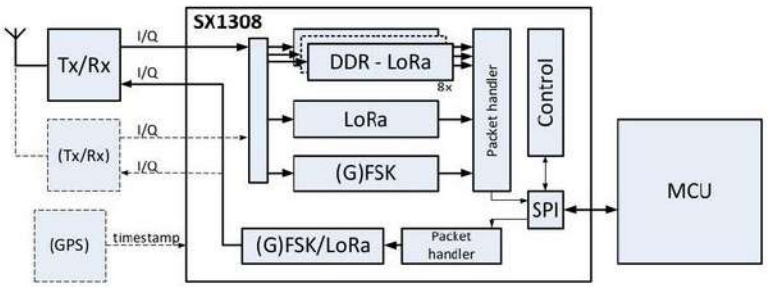


Bild 3. Blockdiagramm SX1308 (Quelle: Semtech [7]).

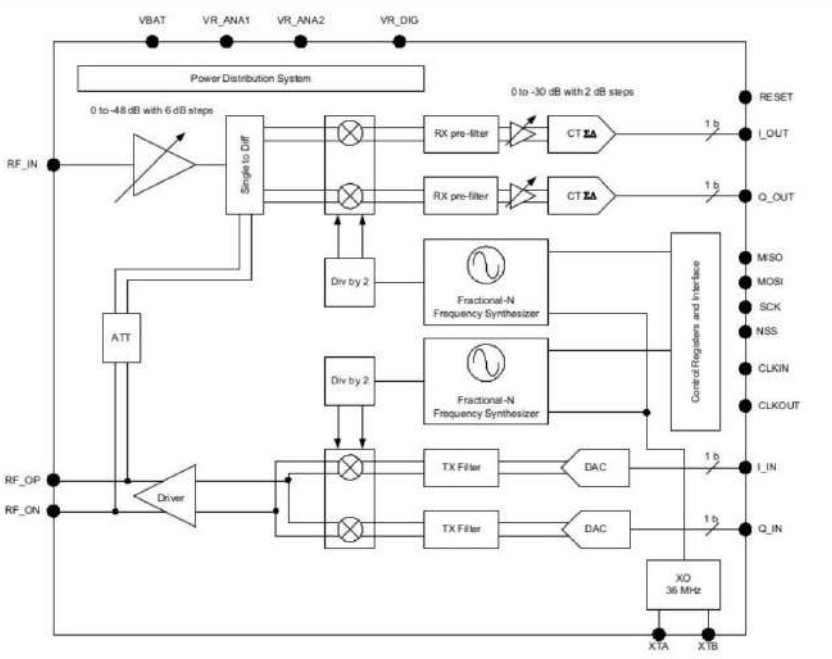


Bild 4. Blockdiagramm SX1257 (Quelle: Semtech [8]).

Das Handbuch führt einen durch die Einrichtung des Gerätes. Dabei müssen nur das Netzwerk passend konfiguriert und die Einstellungen für das passende LoRaWAN (zum Beispiel *The Things Network*) vorgenommen werden. Ab diesem Punkt ist das LoRaWAN-Gateway einsatzbereit (**Bild 5**).

Bild 5. Web-Interface des Gateways.

OpenWRT-Unterbau

Auch wenn die erste Seite des Webinterfaces das noch nicht vermuten lässt: Als Basis des D*ragino LPS8 Indoor Gateways* kommt ein OpenWRT zum Einsatz. Damit steht nicht nur ein LoRaWAN-Gateway zur Verfügung, es lassen sich auch etliche weitere Einstellungen vornehmen, die man von Routern her kennt (IP-Adressen, Weiterleitung, WLAN).

Dank des OpenWRT-Unterbaus lässt sich auch ein LTE- oder 5G-Modem an den USB-Port des Gateways anbinden, falls am Standort des Gerätes keine andere Verbindung zum Internet möglich sein sollte. Wer mag, kann auch per SSH auf die Kommandozeile des Linux zugreifen (auf eigene Gefahr!). So lassen sich per Webinterface oder Kommandozeile weitere Pakete installieren, um Funktionen hinzuzufügen.

Abschließende Worte

Das *Dragino LPS8 Indoor Gateway* ist ein unauffälliger Vertreter eines LoRaWAN-Gateways, und in diesem Falle ist dies ein Pluspunkt. Während des über einjährigen Einsatzes hat das Gerät bei mir klaglos seinen Dienst verrichtet. Für den Betrieb meiner LoRaWAN-Knoten macht das Gateway einen guten Job und auch die Abdeckung des Gebäudes (und der umliegenden) ist mehr als erfreulich. Wer also auf der Suche nach einem preiswerten LoRaWAN-Gateway ist, sollte einen Blick auf das *Dragino LPS8 Indoor Gateway* im Elektor-Shop [5] wagen.

210680-02

Sie haben Fragen oder Kommentare?

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

Ein Beitrag von

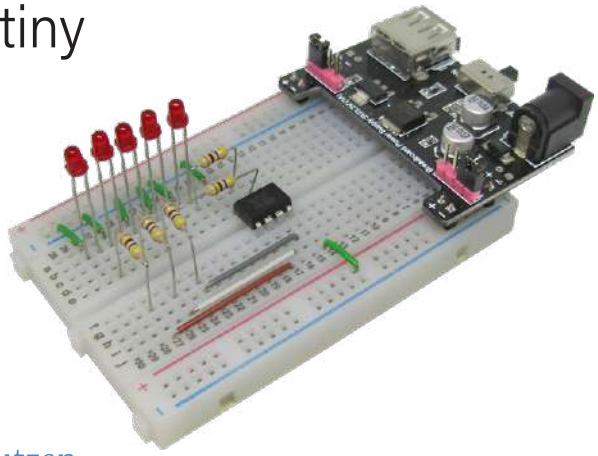
Text: **Mathias Claußen**
Redaktion: **Jens Nickel**
Layout: **Harmen Heida**

WEBLINKS

[1] Mathias Claußen, „LoRaWAN - ein einfacher Einstieg“, ElektorMag 3-4/2020 : www.elektormagazine.de/magazine/elektor-140/57101
[2] Mathias Claußen, „LoRa mit dem Raspberry Pi Pico“, ElektorMag 7-8/2021: www.elektormagazine.de/magazine/elektor-178/59750
[3] Handbuch Dragino LPS8 Indoor Gateway: www.dragino.com/downloads/index.php?dir=LoRa_Gateway/LPS8/
[4] Download Firmware Dragino LPS8 Indoor Gateway: www.dragino.com/downloads/index.php?dir=LoRa_Gateway/LPS8/Firmware/Release/
[5] Bildquelle Gateway: www.dragino.com/media/k2/galleries/148/LPS8-10.jpg
[6] User Manual: https://bit.ly/LPS8-user-manual
[7] Datenblatt Front-End SX1257: https://sforce.co/3fZmy1f
[8] Datenblatt LoRa-Transreceiver SX1308: https://sforce.co/32zxAqV

ATtiny-Mikrocontroller mit C und Assembler erforschen

Beispiel-Kapitel: I/O-Ports des ATtiny



Von Warwick A. Smith (Südafrika)

I/O-Ports steuern die Pins eines Mikrocontrollers und können sie individuell als Eingänge oder Ausgänge konfigurieren. Diese weit gefasste Aussage sollte eigentlich in keinem Einsteigerkurs und in keiner Einführung in die Mikrocontroller-Programmierung fehlen. Um aber die I/O-Fähigkeiten eines Mikrocontrollers wirklich zu verstehen und zu nutzen, muss man tiefer in die Materie eindringen. In diesem Artikel demonstriert der Elektor-Buchautor Warwick Smith die Assembler-Programmierung an dem beliebten ATtiny-Mikrocontroller. Interessiert? Dann schauen wir es uns an!

Anmerkung der Redaktion: *Dieser Artikel ist ein Auszug aus dem 376-seitigen Buch* Explore ATtiny Microcontrollers using C and Assembly Language *(W. Smith, Elektor 2021). Der Auszug wurde formatiert und leicht bearbeitet, um den redaktionellen Standards und dem Seitenlayout von Elektor Mag zu entsprechen. Da es sich um einen Auszug aus einer größeren Veröffentlichung handelt, können sich einige Begriffe in diesem Artikel auf Diskussionen an anderer Stelle im Buch beziehen. Der Autor und die Redaktion haben ihr Bestes getan, um solche Fälle auszuschließen, und sind gerne bereit, bei Fragen zu helfen. Kontaktinformationen finden Sie im Kasten* Fragen oder Kommentare?

Die Konfiguration und Steuerung der I/O-Ports erfolgt beim ATtiny13(A) und ATtiny25/45/85 über vier Register. Wenn ein Pin als Eingang konfiguriert ist, kann ein auf dem AVR laufendes Programm den Logikpegel des Pins lesen. Wenn ein Schalter an den Pin angeschlossen ist, kann der Logikpegel am Pin gelesen werden, um festzustellen, ob der Schalter offen oder geschlossen ist. Wenn ein Pin als Ausgang konfiguriert ist, kann er verwendet werden, um den Logikpegel des Pins high (auf Logikpegel 1) oder low (auf Logikpegel 0) zu schalten. Ein Ausgangspin kann verwendet werden, um eine LED anzusteuern, wie es in dem an anderer Stelle in diesem Buch beschriebenen LED-Blinkprojekt geschehen ist.

Konfigurieren von I/O-Pins als Ausgänge in Assembler

In diesem Abschnitt sehen wir, wie man mehr als einen Pin als Ausgang konfigurieren kann, indem man einen 8-poligen ATtiny mit fünf LEDs verwendet, die mit Vorwiderständen an den Pins angeschlossen sind. Der Programmcode konfiguriert die LEDs als 5-Bit-Binärzähler, der von Null an aufwärts zählt.

Bild 1 zeigt einen Schaltplan eines ATtiny13(A) oder ATtiny25/45/85 AVR-Mikrocontrollers mit fünf LEDs, die an die I/O-Pins PB0… PB4 angeschlossen sind. Der Pin PB5 des ATtiny-Mikrocontrollers wird in der Schaltung als debugWIRE-Pin für die Programmierung und das Debugging des Mikrocontrollers eingesetzt. Für

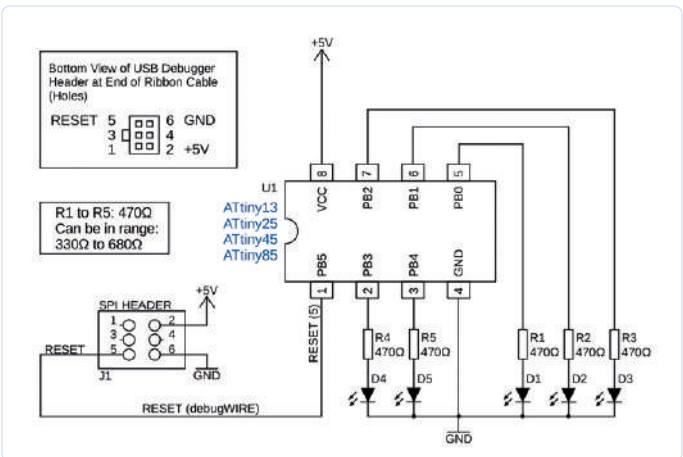


Bild 1. Schaltplan des Fünf-LED-Zählers.

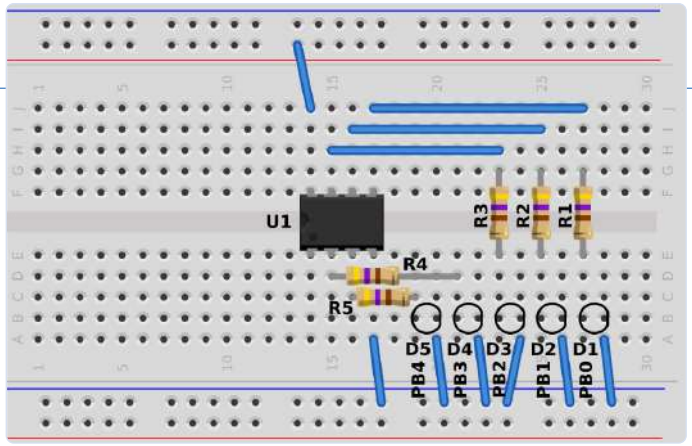


Bild 2. Breadboard-Layout des Fünf-LED-Zählers.

diese Konfiguration muss ein Programmierer/Debugger wie der Atmel-ICE oder AVR-Dragon verwendet werden.
Ein USB-Programmiergerät, das nur dem Programmieren dient, funktioniert nicht im DebugWIRE-Modus und kann kein Debugging durchführen. **Setzen Sie deshalb die DWEN-Fuse nicht bei einem reinen USB-Programmiergerät, da dieses den AVR nicht aus dem DebugWIRE-Modus zurückholen kann.** Die Idee, den debugWIRE-Modus in dieser Beispielschaltung zu verwenden, ist es, die anderen Pins des AVRs freizugeben, die normalerweise im ISP/SPI-Programmiermodus festgelegt sind.

Reine Programmiergeräte

Um das folgende Beispielprogramm auf die Schaltung in Bild 1 zu laden und den Zählwert auf den LEDs anzuzeigen, kann ein USB-Programmiergerät mit reiner Programmierfunktion verwendet werden. Verdrahten Sie die LEDs wie in Bild 1 und **Bild 2** gezeigt. Die originalen USBasp- und USBtinyISP-Designs besitzen Schutzwiderstände auf den Leitungen zum Programmiergerät, die sowohl das Programmiergerät wie auch den Ziel-AVR-Chip schützen. Der Arduino Uno, der als ArduinoISP eingesetzt wird, verfügt nicht über solche Schutzwiderstände, aber diese können der MOSI- und der SCK-Leitung des Arduino Uno zum Ziel-AVR hinzugefügt werden. Im originalen USBasp-Design werden 270-Ω-Schutzwiderstände auf den Leitungen MOSI, SCK und RESET verwendet, beim originalen USBtinyISP-Design sind 1,5-kΩ-Widerstände auf den Leitungen MOSI und RESET platziert.
Diese Schutzwiderstände verhindern einen Kurzschluss, falls einer der Ziel-AVR-Pins eine Ausgangsspannung liefert und das Programmiergerät eine andere.

Peripherie, die die Programmierung stört

Obwohl die Schaltung in Bild 1 mit den angeschlossenen LEDs und Vorwiderständen über die ISP/SPI-Schnittstelle programmiert werden kann, ist es möglich, dass an anderen Schaltungen Hardware angeschlossen ist, die die Programmierung stört. Wenn periphere Hardware an den Pins eines zu programmierenden AVRs die Programmierung stört, gibt es einige Lösungen für dieses Problem. Natürlich können diejenigen Leser, die einen debugWIRE-fähigen USB-Programmierer/Debugger wie den Atmel-ICE besitzen, den Ziel-AVR einfach in den debugWIRE-Modus versetzen und so nur einen Pin für die Programmierung verwenden. Eine andere Lösung besteht darin, den AVR auf einem anderen Breadboard über ISP/SPI zu programmieren und ihn anschließend an die Zielschaltung einzubauen.

Alternativ kann ein AVR mit mehr Pins verwendet werden, aber die freien Portpins stimmen nicht immer mit den gleichen Pins des gleichen Ports eines ATtiny im 8-Pin-PDIP-Gehäuse überein. Wenn man zum Beispiel die Portpins PB0…PB4 wie in der Schaltung in Bild 1 verwendet, stehen bei den 14-poligen ATtiny24/44/84-AVRs nicht fünf aufeinanderfolgende freie Pins zur Verfügung, da die ISP/SPI-Pins sowohl Pins von Port A als auch von Port B belegen. Beim 20-poligen ATtiny26/261/461/861 ist der gesamte Port A frei, wenn ein ISP/SPI-Programmierer angeschlossen ist, aber das bedeutet, dass die Software geändert werden muss, um Port A statt Port B zu verwenden. Glücklicherweise sind beim 20-poligen ATtiny2313/4313 die Pins PB0…PB4 frei, wenn ein ISP/SPI-Programmer angeschlossen ist.

Der AVR im debugWIRE-Modus

Damit der Mikrocontroller wie in Bild 1 über eine einzige debugWIRE-Leitung programmiert werden kann, müssen zunächst alle ISP-Header-Leitungen des USB-Programmers/Debuggers an den Mikrocontroller angeschlossen und dann die DWEN-Fuse gesetzt werden, um den AVR in den debugWIRE-Modus zu versetzen. Dann sollte sich der AVR im debugWIRE-Modus befinden. Wenn nicht, schließen Sie Ihren Programmer/Debugger wie den Atmel-ICE oder AVR-Dragon an. Überzeugen Sie sich, dass sie die DWEN-Fuse setzen können. Sobald dies geschehen ist, können alle ISP-Header-Anschlüsse aus der Schaltung entfernt werden, mit Ausnahme von RESET, +5 V (Vcc) und GND, wie in Bild 1 dargestellt.

Aufbau der 5-LED-Zählschaltung auf dem Breadboard

Wenn Sie alle Bauteile zusammen haben, können Sie die Schaltung aus Bild 1 auf einem Breadboard aufbauen. Die fünf LEDs werden auf dem Breadboard in einer Reihe platziert, wobei die Positionen den Pins PB0…PB4 entsprechen, beginnend mit PB0 (LED D1) auf der rechten und PB4 (LED D5) auf der linken Seite. Mit anderen Worten, wir wollen eine LED-Reihe, bei der PB0 mit der LED rechts verbunden ist, PB1 mit der LED links daneben, PB2 mit der LED an dritter Stelle von rechts und so weiter, wie in Bild 2 zu sehen ist. Das Bild zeigt nur die Umrisse der LEDs, damit sie nicht die Sicht auf die Drahtverbindungen und Widerstände versperren. Wenn Sie nicht über die entsprechende Hardware verfügen, können Sie die Programme auch mit einem Simulator nachvollziehen.

Assembler-Code für die 5-LED-Zählschaltung

Starten Sie ein neues Assembler-Projekt im Microchip-Studio-AVR mit dem Namen *led_count_asm*. Kopieren Sie den Code aus **Listing 1** in die Datei *main.asm* des Projekts und ersetzen Sie dabei den Skeleton-Code.
Wenn Sie die Hardware aus Bild 1 verwenden, wählen Sie im Microchip-Studio das Hardware-Tool (Ihren Debugger) wie den Atmel-ICE, mit debugWIRE als Schnittstelle. Klicken Sie dazu auf das *Hammer*symbol in der zweiten oberen Symbolleiste. Wenn Sie den Simulator verwenden, wählen Sie stattdessen *Simulator* als Werkzeug. Wenn Sie ein „Hobby"-Programmiergerät verwenden, erfahren Sie im Folgenden, wie Sie das Programm auf den Ziel-ATtiny laden können.
Das Programm *led_count_asm* konfiguriert die Pins PB0…PB4 als

Ausgänge, damit die angeschlossenen LEDs durch das Programm ein- und ausgeschaltet werden können. Das Programm zeigt eine aufsteigende Binärzahl oder Zählung auf den LEDs an, beginnend bei 0 (was bedeutet, dass alle LEDs aus sind). Wenn der Zählwert seinen Maximalwert erreicht hat (alle LEDs an), kehrt er anschließend zu Null zurück und beginnt von neuem. Jede „Aus"-LED steht für eine binäre Null oder eine logische 0, und jede „An"-LED steht für eine binäre Eins oder eine logische 1. Es ist wichtig, die LEDs wie in Bild 2 anzuordnen, damit die Zählung korrekt angezeigt wird, wobei PB0/D1 das LSB und PB4/D5 das MSB des Zählwerts darstellt.
Um das Programm zu erstellen und in den AVR in der physikalischen Hardware aus Bild 1 und Bild 2 zu laden, benutzt man beim Atmel-ICE oder AVR Dragon das Symbol *Start Without Debugging* in der oberen Symbolleiste des Microchip-Studios (oder die Tastenkombination *Ctrl* + *Alt* + *F5*), um das Programm in den AVR zu laden. Wenn die Debugger-Schnittstelle korrekt eingerichtet ist und der Chip sich im DebugWIRE-Modus befindet, wird das Programm hochgeladen und beginnt sofort zu laufen. Die aufsteigende Binärzahl wird von den LEDs angezeigt. Wenn Sie ein reines Hobby-USB-Programmiergerät verwenden, laden Sie das Programm mit der entsprechenden Funktion auf den Ziel-AVR. Ist der Code richtig eingegeben, die Schaltung richtig verdrahtet und das Programm nach der Eingabe gespeichert und kompiliert, sehen Sie den binären Zählwert auf den LEDs aufwärts zählen.
Wenn Sie nicht über die Hardware verfügen, können Sie den Zählwert dennoch mit dem Simulator im Microchip-Studio anzeigen. Mit den Icons *Start Debugging* und *Break* kann das Programm schrittweise durchlaufen werden. Wenn der Simulator gestartet ist, öffnen Sie mit *Debug Windows I/O* das I/O-Fenster im oberen Menü und klicken Sie dann auf *I/O Port (PORTB)*. Gehen Sie mit dem *Step Over*-Icon oder der *F10*-Tastaturtaste durch das Programm. Am unteren Rand des I/O-Fensters ist unter PORTB der Zählwert zu sehen, der auf den LEDs angezeigt würde, wenn sie denn angeschlossen wären. Der Zählerstand in PORTB wird jedes Mal aktualisiert, wenn die `OUT`-Anweisung in der Hauptschleife übersprungen wird.

Funktionsweise des LED-Zähl-Assemblerprogramms

Die Hälfte des Programmcodes von *led_count_asm* besteht aus der Verzögerungs-Subroutine, die auch in dem an anderer Stelle in diesem Buch besprochenen LED-Blinkprogramm verwendet wurde. Diese Subroutine wird in der Hauptschleife des Programms einmal aufgerufen und verzögert die Zählung der LEDs so weit, dass sie für das Auge sichtbar ist, ansonsten würde die Zählung viel zu schnell ablaufen. Öffnen Sie das Projekt *led_count_asm* im Microchip-Studio und den Code von *main.asm*.
Die ersten beiden Anweisungen des Programms setzen die Pins PB0...PB4 als Ausgänge für die Ansteuerung der LEDs. Dazu werden Bits im DDRB-Register gesetzt (oben in **Bild 3**). Jedes Bit in diesem Register entspricht einem Pin des Mikrocontrollers. Das Bit DDB0 entspricht zum Beispiel dem Pin PB0, DDB1 dem Pin PB1 und so weiter.
Wenn ein Bit in DDRB auf logisch 1 gesetzt wird, wird der entsprechende Pin zu einem Ausgang. Wenn ein Bit in DDRB auf logisch 0 gesetzt wird, wird der entsprechende Pin zum Eingang (was auch die Voreinstellung aller Pins beim Einschalten oder Zurücksetzen

Listing 1. led_count_asm : main.asm

```
; Set up pins PB0 to PB4 as output pins
     ldi     r16, 0b0001_1111
     out     DDRB, r16
     clr     r18              ; Clear count register
loop:
     out     PORTB, r18       ; Display count on LEDs
     rcall   delay
     inc     r18              ; Increment count
     andi    r18, 0b0001_1111 ; Clear unused bits
     rjmp    loop
; Delay subroutine
delay:
     ldi     r16, 0xff
delloop1:
     ldi     r17, 0xff
delloop2:
     dec     r17
     brb     SREG_Z, delloop2
     dec     r16
     brbc    SREG_Z, delloop1
     ret
```

DDRB : Port B Data Direction Register. Configures pin direction: 0 = Input, 1 = Output.
Address: 0x17 **Initial Value:** 0b0000_0000 **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0

PORTB : Port B Data Register. Drives output pins. Enables input pin pull-ups with logic 1.
Address: 0x18 **Initial Value:** 0b0000_0000 **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0

PINB : Port B Input Pins Register. Read input pin state. Write logic 1 to toggle output pin.
Address: 0x16 **Initial Value:** Depends on level on pin. **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

Bild 3. I/O-Port-Register des ATtiny13(A) und des ATtiny25/45/85.

ist). Am Anfang des Programms wird zunächst 0b0001_1111 in das Register R16 (unten in Bild 3) geschrieben und dann mit einem `OUT`-Befehl in das DDRB-Register übergeben. Es ist notwendig, diesen konstanten Wert zuerst in R16 zu laden, da es keinen Befehl gibt, um einen konstanten Wert direkt in ein I/O-Register wie DDRB zu schreiben. Das Schreiben von 0b0001_1111 über R16 in DDRB setzt die Bits DDB0...DDB4 und macht die Pins PB0...PB4 zu Ausgängen.
Obwohl es vier Register zur Steuerung des I/O-Ports B gibt, sind in Bild 3 nur drei dargestellt. Das vierte Register MCUCR besitzt nur ein einziges Bit, das für den Anschluss B gilt. Dieses Bit ist ein globales Pull-up-Sperrbit, das in diesem Kapitel nicht verwendet wird. Dieses Register wird im Datenblatt für den ATtiny13, ATtiny13A oder ATtiny25/45/85 im Abschnitt I/O-Ports beschrieben.
Das Register R18 in der Mitte von Bild 3 wird im Programm

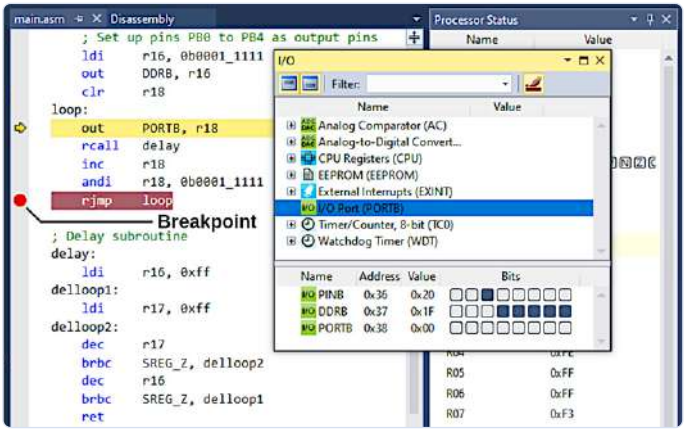


Bild 4. Einfügen eines Haltepunkts im Microchip-Studio-Debugger.

verwendet, um einen inkrementellen Zählwert zu erhalten, der auf den LEDs angezeigt wird. R18 wird vor der Hauptschleife mit `CLR` auf 0 zurückgesetzt, so dass die Zählung bei 0 beginnt.
In der Hauptschleife wird der Inhalt von R18 mit dem `OUT`-Befehl an das PORTB-Register gesendet. Auch hier entspricht jedes Bit dieses Registers einem Pin des Mikrocontrollers. Bei Pins, die mit DDRB als Ausgänge eingerichtet wurden, erscheinen die in das PORTB-Register geschriebenen Logikpegel an diesen Pins. Bei Pins, die als Eingänge definiert wurden, aktiviert eine logische 1 in PORTB einen internen Pull-up-Widerstand an diesen Pins. Eine logische 0 deaktiviert dagegen den Pull-up-Widerstand an dem zugehörigen Pin.
Da die Pins PB0...PB4 als Ausgänge konfiguriert sind, erscheint der in PORTB geschriebene Zählwert an diesen Pins als logische Pegel, die die LEDs ein- und ausschalten. Eine logische 1 im Zählwert schaltet die entsprechende LED ein und eine logische 0 schaltet die entsprechende LED aus.
Nachdem der Zählwert mit dem `OUT`-Befehl in PORTB geschrieben wurde, wird das Verzögerungsunterprogramm aufgerufen, um den Zählwert eine Zeit lang auf den LEDs zu belassen. Das Register R18 wird dann mit dem `INC`-Befehl um 1 erhöht, so dass der enthaltene Zählwert inkrementiert wird. Mit `ANDI` werden durch die Maske 0b0001_1111 die obersten drei Bits des Zählwerts in R18 gelöscht, so dass diese oberen Bits immer Null sind. Am Anfang der Schleife, wenn R18 wieder nach PORTB geschrieben wird, werden die obersten drei Bits immer mit 0 belegt, da sie mit `ANDI` gelöscht wurden. Wenn das Ende der Schleife mit der `RJMP`-Anweisung erreicht ist, beginnt die Programmausführung wieder am Anfang der Schleife, wobei der neue Zählwert in PORTB geschrieben und auf den LEDs angezeigt wird.
Bei diesem Programm sind einige Dinge zu beachten: Die I/O-Register DDRB und PORTB werden in beiden Programmen zum Einrichten und Steuern von Port B des Mikrocontrollers verwendet. Das LED-Blinkprogramm muss nur einen einzigen Pin steuern und verwendet daher die Befehle `SBI` und `CBI`, um ein einzelnes Bit in den I/O-Registern direkt zu setzen und zu löschen - keines der Arbeitsregister R0 bis R31 wird dafür benötigt. Das LED-Zählprogramm muss aber auf fünf Pins beziehungsweise LEDs gleichzeitig zugreifen und verwendet daher den `OUT`-Befehl, um mehrere Bits in einem Rutsch in die Register zu schreiben.
In einem Assemblerprogramm ist es wichtig, die Verwendung von Registern im Auge zu behalten. Zum Beispiel wird zu Beginn des Programms ein augenblicklicher Wert in R16 geladen. R16 wird in diesem Fall nur vorübergehend gebraucht und kann daher später im Programm anderweitig verwendet werden, ohne dass sein Wert vorher gespeichert werden muss. So wird das Register im Unterprogramm *delay* wieder verwendet. Da R16 und R17 in der Verzögerungs-Subroutine gebraucht werden, wurde R18 für nichts anderes als die Speicherung des Zählwerts gewählt. Dadurch wird der Zählwert niemals überschrieben. Wenn ein Programm sehr groß wird und keine freien Register für spezielle Zwecke zur Verfügung stehen, können die Werte in den verwendeten Registern mit `PUSH`- und `POP`-Befehle am Anfang und Ende eines Unterprogramms erhalten werden.

Breakpoints im Debugger verwenden

Mit dem Simulator oder der physischen Hardware und zum Beispiel dem Atmel-ICE können Sie den Debugger verwenden, um schrittweise durch das Programm zu gehen, indem Sie auf *Start Debugging* und *Break* klicken, wie wir es zuvor getan haben. Zeigen Sie die I/O-Port-Register an, indem Sie das I/O-Fenster in Microchip-Studio öffnen. Wählen Sie bei laufendem Debugger *Debug Windows I/O* und klicken Sie dann im I/O-Fenster auf *I/O Port (PORTB)*, wie in **Bild 4** gezeigt.
Wenn die Programmausführung in die Hauptschleife eintritt, ist es zweckmäßig, einen Haltepunkt (Breakpoint) an der `RJMP`-Anweisung zu setzen und die gesamte Schleife auszuführen, indem Sie auf die Schaltfläche *Continue* in der Symbolleiste klicken (Tastaturtaste F5). Ein Haltepunkt kann auf die `RJMP`-Anweisung gesetzt werden, indem man im Fenster des Microchip-Studios auf die graue Marge links neben dem Befehl klickt. Dadurch erscheint ein roter Punkt in diesem Bereich, der anzeigt, dass ein Breakpoint für diesen Befehl gesetzt wurde (siehe Bild 4). Alternativ können Sie auch auf die Anweisung klicken, die Sie mit einem Haltepunkt versehen möchten, so dass der Cursor darauf platziert wird, und dann *Debug Toggle Breakpoint* aus dem oberen Menü wählen oder die *F9*-Tastaturtaste drücken. Sobald der Haltepunkt gesetzt ist, können Sie die Schaltfläche *Continue* in der Symbolleiste oder *F5* drücken, um die gesamte Schleife zu durchlaufen und nur an deren Ende zu unterbrechen. Auf diese Weise können Sie sehen, wie sich der Zählerstand im PORTB im *I/O window* und im Register R18 im *Processor Status window* um 1 erhöht, ohne dass Sie in der Schleife von Befehl zu Befehl springen müssten.
Um den Haltepunkt von der `RJMP`-Anweisung zu entfernen, klicken Sie auf den roten Punkt links der Anweisung oder verwenden Sie das Menü oder *F9*, wenn der Cursor steht auf der `RJMP`-Anweisungszeile steht. Setzen Sie einen Haltepunkt beim Befehl `INC R18`. Drücken Sie nun *F5*, um die Schleife zu durchlaufen. Der neue Haltepunkt sorgt dafür, dass das PORTB-Register und R18 denselben Wert enthalten, wenn die Programmausführung endet. Wenn die Programmausführung bei `RJMP` stoppt, wird R18 inkrementiert, bevor der neue Wert in PORTB geschrieben wird, so dass die beiden Werte in den Debugger-Fenstern von Microchip Studio nicht synchron ansteigen.

Die Software, die der Autor zur Unterstützung des Buches erstellt hat, kann kostenlos heruntergeladen werden. Gehen Sie zu [1], scrollen Sie nach unten zu den *Downloads* und klicken Sie auf den Dateinamen *Software_Explore ATtiny Microcontrollers using C and Assembly Language*. Speichern Sie die ZIP-Archivdatei (etwa 29 kB) lokal und entpacken Sie sie.

220045-02

Ein Beitrag von

Text und Illustrationen: **Warwick Smith**
Redaktion: **Jan Buiting**
Übersetzung: **Rolf Gerstendorf**
Layout: **Giel Dols**

Haben Sie Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare zu diesem Artikel? Senden Sie eine E-Mail an den Autor unter warwsmi@axxess.co.za oder an Elektor unter redaktion@elektor.de.

WEBLINK

[1] Buchressourcen/Infoseite : www.elektor.de/20007

PASSENDE PRODUKTE

- Explore ATtiny Microcontrollers using C and Assembly Language
  Buch SKU 20007: www.elektor.de/20007
  E-Buch SKU 20008: www.elektor.de/20008

WinUI 3: Neues Grafik-Framework für Windows-Apps

Eine erste App für Elektroniker

Von **Dr. Veikko Krypczyk**

Software zur Steuerung von unterschiedlichsten Elektronik-Anwendungen läuft häufig unter Windows. Über lokal installierte Desktop-Applikationen hat man einen direkten Zugriff auf alle Systemschnittstellen des PCs. Microsoft vollzieht hier aktuell einen technologischen Umbruch: Die neue Grafikschnittstelle WinUI 3 wird das Maß der Dinge. In diesem Artikel beleuchten wir die technischen Hintergründe und zeigen anhand einer Demo für Elektroniker, wie man das Ganze programmiert.

In vielen Elektronik-Projekten kommt ein Windows-PC zum Einsatz, zur Steuerung, zum Logging von Daten und weiteren Aufgaben . Meist muss dafür Windows-Software mit einer grafischen Benutzeroberfläche individuell programmiert werden.

Lokal ausgeführte Desktop-Applikationen sind hier oft das Mittel der Wahl, denn mit ihnen hat man einen vollständigen Zugriff auf die Systemumgebung und über entsprechende Treiber auch auf die angeschlossene Peripherie.

Microsoft, als Hersteller des Betriebssystems Windows, vollzieht hier gerade einen größeren technologischen Umbruch. Im Fokus stehen die Anbindung und Gestaltung der Benutzeroberflächen. Die Entwicklungen werden unter den Projektnamen *WinUI 3* und *Windows App SDK* koordiniert. In diesem Artikel zeigen wir, was es mit dieser neuen Grafikschnittstelle auf sich hat und wo man diese einsetzen kann. Als Erstes verschaffen wir uns einen Überblick über die Möglichkeiten, Windows-Applikationen mit grafischer Benutzerschnittstelle zu programmieren. Anhand unterschiedlicher Technologien und Anwendungstypen wird dann der Sinn der WinUI 3 deutlich. Natürlich soll es nicht nur bei der Theorie bleiben: Wir experimentieren mit dem neuen Typ einer Windows-Anwendung und erstellen eine erste Applikation.

Technologien

Grundsätzlich kann man Anwendungen mit grafischer Oberfläche für das Betriebssystem Windows zum heutigen Zeitpunkt in zwei Typen einteilen. Zum einem haben wir *Desktop-Applikationen*. Diese beruhen im Kern auf der Verwendung der *Win32 API*. Für deren Entwicklung gibt es unterschiedliche Ansätze, Frameworks und Programmiersprachen. Aus dem Hause Microsoft stammen die Technologien Windows Forms (WinForms) und Windows Presentation Foundation (WPF). WinForms setzt auf die GDI-Schnittstelle von Windows, WPF basiert intern auf DirectX und war ursprünglich als Ersatz für WinForms gedacht. Beide Grafikframeworks beruhen auf dem .NET-Framework, das für Windows-Programme vorgesehen war und dessen Weiterentwicklung mit Version 4.8 endet.

Version .NET 5 ist dagegen der technologische Nachfolger von .NET Core. Dieses Framework ist nicht auf Windows beschränkt, sondern kann auch auf anderen Betriebssystemen zum Einsatz kommen. Microsoft hat überraschenderweise sowohl WinForms als auch WPF nach .NET Core transformiert. Wenn Sie heute eine App für Windows erstellen und sich dabei für WinForms oder WPF entscheiden, dann haben Sie also die Wahl zwischen dem bisherigen .NET-Framework und .NET Core. Auch eine Migration von bestehenden Applikationen ist möglich, aber wie so oft bei solchen Vorhaben nicht selten mit einer Reihe von Problemen behaftet. Auch andere Hersteller von Entwicklungswerkzeugen für Windows-Applikationen haben meist auf die Grafikschnittstelle des Betriebssystems (GDI) aufgesetzt und diese in eigenen Frameworks zur Verwendung gekapselt.

Die zweite Kategorie von Windows-Applikationen sind Apps für die *Universelle Windows Plattform (UWP).* Diese laufen in einem individuell abgeschotteten Bereich des Betriebssystems und haben auch nur einen eingeschränkten Systemzugriff. Anwender installieren diese Apps über den Store. In der Praxis hat sich dieser App-Typ jedoch nicht oder nur sehr zögerlich durchgesetzt. Das liegt unter anderem daran, dass der Systemzugriff aus diesen Apps sehr stark eingeschränkt ist. Die UWP hat jedoch den Vorteil, dass das genutzte Grafikframework WinUI 2 deutlich moderner als die Technologien von WinForms und WPF ist. Ein ansprechendes Design, neue visuelle Komponenten, der Einsatz von Materialien und die Orientierung an der Designsprache *Fluent Design* sind die herausragenden Eigenschaften. Mit anderen Worten: Apps für die UWP wirken modern, zeitgemäß und frisch in der

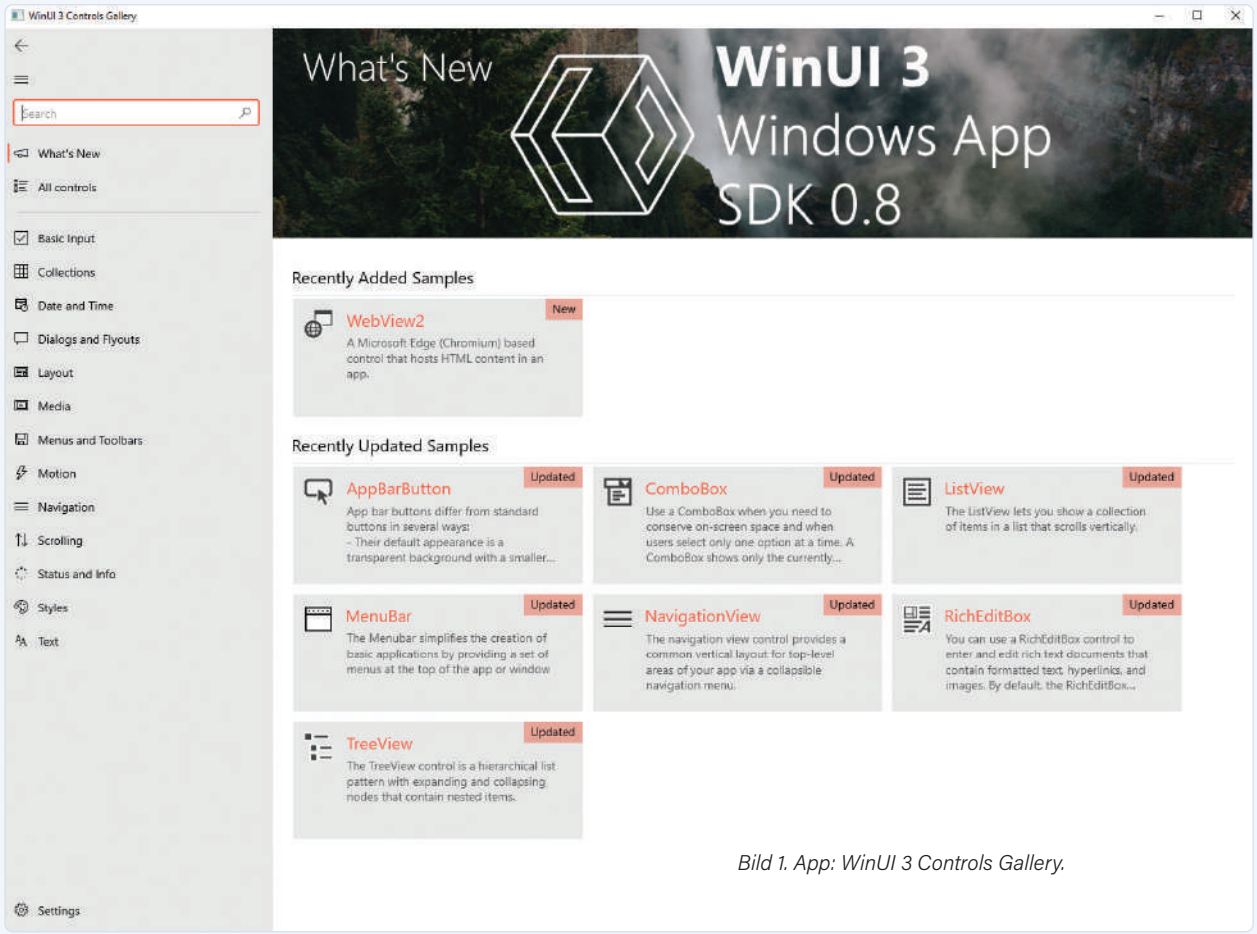


Bild 1. App: WinUI 3 Controls Gallery.

Ansicht, ihre Einsatzoptionen sind jedoch beschränkt. Um ähnliche Wirkungen mit den Technologien WinForms oder WPF zu erreichen, muss man einige Klimmzüge machen, extensiv Komponenten von Drittanbietern verwenden oder die Technologien WinForms/WPF mit der UWP „mischen". Das führt aber schnell zu komplexen App-Strukturen mit den typischen Nachteilen, einer höheren Fehleranfälligkeit und einer schlechteren Wartbarkeit.

Software für elektronische Steuerungen, Erweiterungen und so weiter sind fast ausnahmslos klassische Desktop-Anwendungen, die man auch mit anderen Werkzeugen und Frameworks erstellen kann. Für die Programmiersprache Java gibt es zum Beispiel das Grafikframework Swing, welches für Windows intern jedoch wiederum auf der Betriebssystemschnittstelle GDI basiert.

Das Grafikframework WinUI 3

Mit der Einführung der Grafikschnittstelle WinUI 3 möchte Microsoft allen Anwendungen unter Windows die Verwendung einer zeitgemäßen Grafikschnittstelle ermöglichen. WinUI 3 ist der technologische Nachfolger von WinUI 2 [1]. Sie steht jedoch für alle Arten von Windows-Anwendungen zur Verfügung und ist damit nicht auf den Einsatz in Apps für die UWP beschränkt. WinUI 3 ist Bestandteil des neuen Windows App SDK, welches auch parallel mit der Einführung von Windows 11 bereitgestellt wird. Das Windows App SDK bündelt neue Features für die Entwicklung von Windows-Anwendungen. Es zielt dabei nicht nur auf Windows 11 ab, sondern kann auch unter den aktuellen Versionen von Windows 10 verwendet werden. Das Windows App SDK befindet sich noch in der laufenden Entwicklung. Doch es steht eine erste Version zur Verfügung, welche man bereits in neu erstellten Anwendungen einsetzen kann.

WinUI 3 basiert technisch und konzeptionell auf WinUI 2. Wenn Sie schon mal eine App für die UWP entwickelt haben, dann werden Sie schnell zurechtkommen. Es wird auf die folgenden Prinzipien gesetzt:

- *Trennung von Code und Design*: Die Benutzeroberfläche wird deklarativ mit Hilfe der XML-basierten Sprache XAML in separaten Dateien erstellt.
- *Steuerelemente für die Gestaltung*: Es steht eine Palette von Steuerelementen zur Gestaltung der Oberfläche zur Verfügung. Darunter sind Basiselemente wie Buttons und Texteingabefelder, aber auch komplexere und modernere Elemente, wie ein Kalendersteuerelement, eine WebView oder ein Element zur Anzeige von Personendaten, wie wir es zum Beispiel bei einer Nutzerverwaltung benötigen. Sie möchten einen Überblick über die verfügbaren Steuerelemente bekommen? Dann wechseln Sie in den Microsoft Store. Laden Sie dort die App *WinUI 3 Controls Gallery* herunter. Bei dieser App handelt es sich um eine Vorschau der verfügbaren Steuerelemente für WinUI 3. Es wird deren Verwendung (Use Case) und die Einbindung im Quellcode (XAML) demonstriert. Ebenso findet man entsprechende Links zur Dokumentation (**Bild 1**).
- *Lose Kopplung durch DataBinding*: Die Eigenschaften und die Ereignisse der Steuerelemente werden mittels Datenbindung an den Quellcode gebunden. Auf diese Weise werden Daten zwischen dem Programmcode und den Steuerelementen der Benutzeroberfläche in beiden Richtungen ausgetauscht. Ebenso werden Ereignisse der Steuerelemente wie der Klick auf einen Button auf gleiche Art und Weise an den betreffenden Algorithmus weitergeleitet.
- *Modernes Design*: Die WinUI 3 setzt auf aktuelle Designtrends. Dazu gehören die Verwendung der Designsprache *Fluent* und

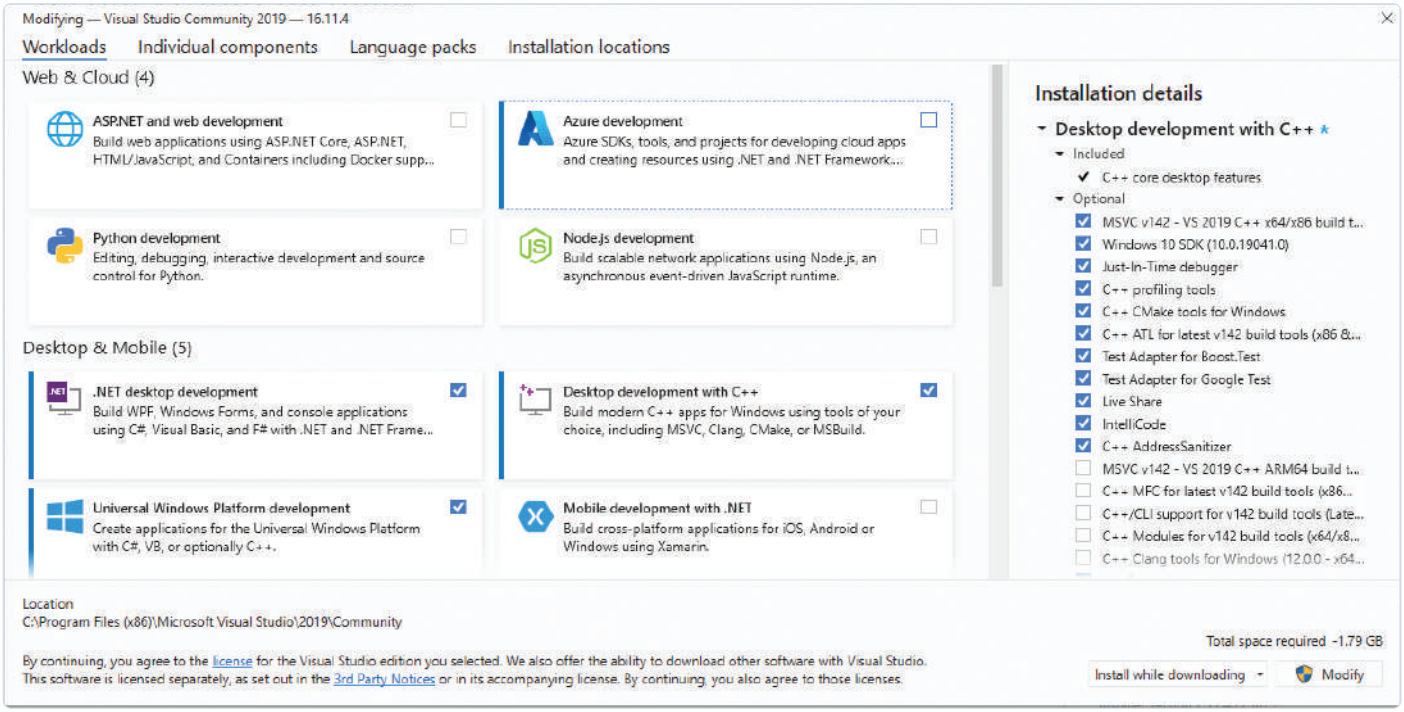


Bild 2. Auswahl der notwendigen Workloads für Visual Studio.

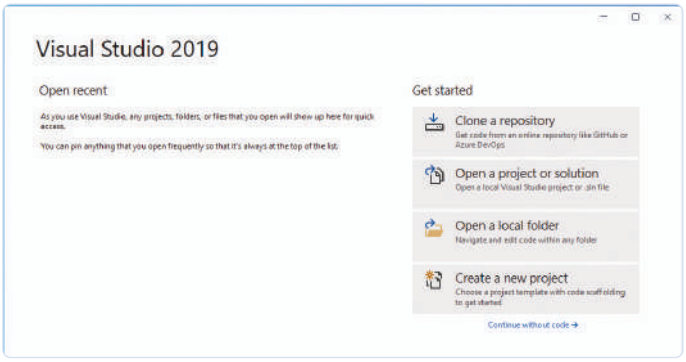


Bild 3. Erster Start von Visual Studio.

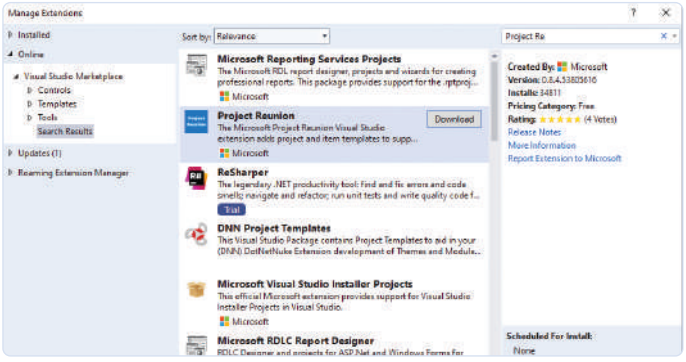


Bild 4. Installation des Windows App SDK (Projekt „Reunion").

eine Nutzung des Materials *Mica*. Die Designsprache Fluent setzt auf die folgenden Elemente: bewusster Einsatz von Geometrie und Farbe, Überlappungen von Oberflächen, Nutzung ausgewählter Materialien und der Einsatz von spezifischer Ikonografie und Typographie zur visuellen Gestaltung mittels Bildern, Symbolen und Schriftarten. Ebenso werden Bewegungen zur Unterstützung des Benutzerfeedbacks verwendet.

Sehen wir uns jetzt die Vorgehensweise zur Anwendungsentwicklung mit der WinUI 3 an. Zuvor müssen wir jedoch die Entwicklungsumgebung einrichten.

Entwicklungsumgebung und Setup

Als Entwicklungsumgebung kommt die aktuelle Version von Visual Studio 2019 [2] zum Einsatz. Es genügt die Community Edition. Wenn Sie diesen Artikel lesen, gibt es gegebenenfalls schon eine stabile Version von Visual Studio 2022, die Sie dann auch verwenden können. Zuvor empfiehlt es sich, aktuelle Updates für das Betriebssystem zu installieren. Während der Installation von Visual Studio werden Sie zur Auswahl der Installationspakete aufgefordert. Den *Visual Studio Installer* können Sie über das Startmenü auch zu einem späteren Zeitpunkt jederzeit aufrufen. Wählen Sie die folgenden Installationspakete (Workloads): *.NET desktop development*, *Desktop development with C++* und *Universal Windows Platform development* (**Bild 2**). Nach der Installation starten Sie Visual Studio und im Startbildschirm wählen Sie den Eintrag *Continue without code* -> (**Bild 3**).
Wir müssen nun zusätzlich das Template für die Entwicklung mit WinUI 3 installieren. In Visual Studio gehen Sie dazu zum Menüpunkt *Extensions* | *Manage Extensions*. Suchen Sie hier nach *Project Reunion*

(der Entwicklungsname des neuen Windows App SDK) und installieren Sie die aktuelle Version (**Bild 4**). Installieren Sie auf gleiche Weise auch die Erweiterung *Windows Template Studio*. Diese bietet erweiterte Vorlagen zum Erstellen einer neuen Anwendung. Visual Studio muss nach dem Download der Erweiterungen neu gestartet werden, dann erfolgt die Installation automatisch.

Eine App für WinUI 3

Starten wir nun mit einem neuen Projekt. Als Vorlage (**Bild 5**) wählen wir *App WinUI 3 in Desktop (Windows Template Studio)*. Im Windows Template Studio (**Bild 6**) nehmen wir Voreinstellungen für das Projekt vor:

- *Projekttyp*: Festgelegt wird die Art der Navigation, zum Beispiel mit Menüleiste oder seitlicher Navigationsleiste (Hamburger-Menü).
- *Design-Pattern*: Direkte Installation und Konfiguration des MVVM-Toolkits. Das dient dazu, die Elemente des User Interfaces (definiert in XAML) an die Programmlogik (Programmiersprache C#) zu koppeln.
- *Seiten*: Wir können dem Projekt einige Seiten hinzufügen. Dabei haben wir die Auswahl aus unterschiedlichen Vorlagen, zum Beispiel eine Seite für Programmeinstellungen.
- *Ergänzende Features*: Hier lassen sich zum Beispiel Themes auswählen oder Programmeinstellungen speichern.

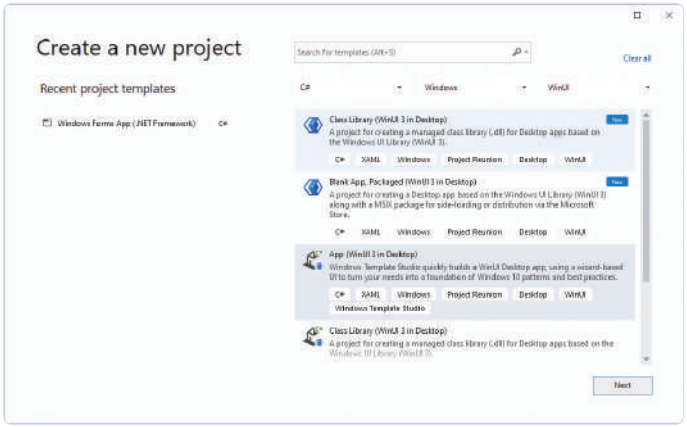


Bild 5. Projektvorlage WinUI 3 Desktop.

Mit einem Klick auf den Button *Create* erzeugen Sie die Desktop-Anwendung, welche WinUI 3 nutzt. Windows Template Studio generiert eine Projektmappe mit drei Projekten:

- *App*: Dieses enthält den Quellcode zur Desktop-Anwendung. Im Unterordner *View* finden Sie zum Beispiel die XAML-Dateien

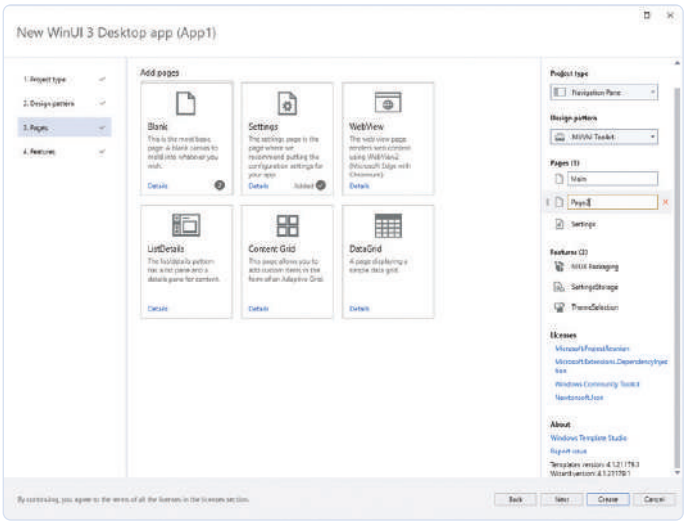


Bild 6. Windows Template Studio.

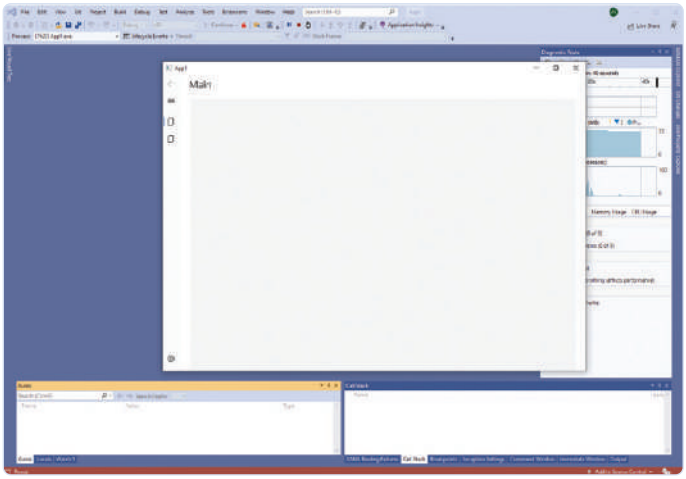


Bild 7. Eine erste Desktop-Anwendung mit WinUI 3.

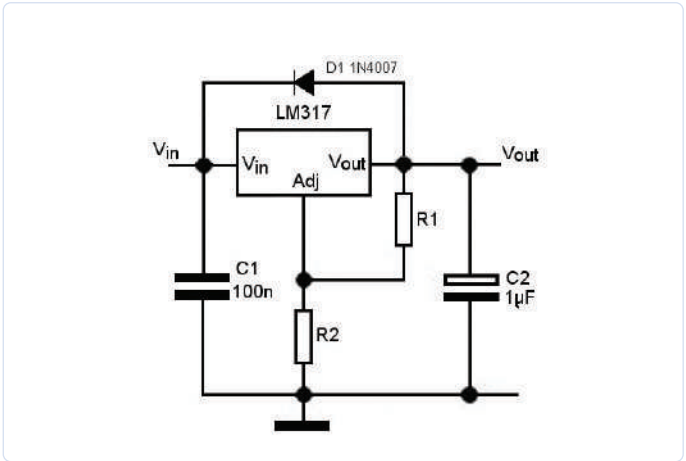


Bild 8. Schaltung mit Spannungsregler LM317.

für die Seiten, welche vom Windows Template Studio angelegt wurden. Die Programmlogik wird in Dateien des Programmordners *ViewModel* abgelegt.

- *Package*: Das Projekt ist für die Bereitstellung der Desktop-Applikation verantwortlich. Im Moment werden WinUI 3-Anwendungen über ein App-Package auf dem Zielrechner installiert. Diese Form wird bisher für Apps der UWP angewendet. Die generierten Packages können auch über den Store verteilt werden. Künftige Versionen des Windows App SDKs sollen auch eine Installation ohne App-Package ermöglichen.
- *Core*: Dieses Projekt enthält die Sammlung von Services und Klassen, welche Dienste für die App zur Verfügung stellen. Dieses Projekt ist nicht zwingend und kann auch weggelassen werden.

Starten Sie die Anwendung direkt aus Visual Studio über den grünen Pfeil in der Symbolleiste. Herzlichen Glückwunsch. Sie haben Ihre erste Anwendung mit WinUI 3 erstellt (**Bild 7**). Dabei weisen wir nochmals darauf hin: Es handelt sich um eine Desktop-Anwendung mit einem vollständigen Systemzugriff. Das ist wie gesagt wichtig für Software, die zum Beispiel externe Elektronik steuert. Eine seitliche Navigation, erste Seiten und die Möglichkeit, das Design der Anwendung anzupassen, sind bereits vorhanden. Sie haben alle Möglichkeiten für einen Zugriff auf das System, inklusive der Kommunikation mit den Systembibliotheken und Treibern. Jetzt wollen wir mit der Gestaltung des User Interfaces experimentieren.

Demo-Applikation

Am besten wird man mit einem neuen System vertraut, wenn man damit arbeitet. Also gestalten wir eine Benutzeroberfläche einer ersten Anwendung (den Quellcode zum Beispiel finden Sie auf der Webseite zum Artikel [3]). Ausgangspunkt ist die XAML-Datei der betreffenden Seite. Als Experiment wollen wir ein Rechentool für den einstellbaren Spannungsregler LM317 erstellen (**Bild 8**). Die Berechnung der Ausgangsspannung folgt der bekannten Formel $U_{out} = 1{,}25\,(1 + R2 / R1)$. Wir können diese Gleichung nach *R2* auflösen und damit den Wert für eine gewünschte Spannung errechnen. Mit Hilfe dieses Beispiels können wir die Vorgehensweise bei der Programmierung von Anwendungen mit WinUI 3 demonstrieren. Das umfasst folgende Schritte:

- Definition des User Interfaces in XAML.
- Kodieren der Programmlogik in C#.
- Bindung des User Interfaces an die Programmlogik.
- Weiterleiten der Nutzerinteraktion vom User Interface an die Programmlogik.
- Datenausgabe auf dem Formular.

Beginnen wir mit der Definition der Oberfläche. Wir benötigen zwei Textfelder für die Erfassung der Werte von *R1* und U_{Out}. Ebenso benötigen wir ein Textfeld für den Wert von *R2*. Für das Auslösen der Berechnung wird ein *Button* eingesetzt. Für die Ein- und Ausgabe verwenden wir Steuerelemente des Typs *TextBox*. Alle Elemente sollen untereinander platziert werden, daher werden diese in einen Layout-Container des Typs *<StackPanel />* eingefügt. Ohne weitergehende Konfiguration werden alle Elemente innerhalb eines *StackPanels* untereinander angeordnet. Die Steuerelemente werden über den XAML-Code konfiguriert, mit den Eigenschaften gemäß **Tabelle 1**.

Tabelle 1. Steuerelemente.

Steuer-element	Eigenschaft	Wert	Bedeutung
TextBoxR1	Width	200	Breite
	Margin	20, 20, 0, 0	Abstand nach links, oben, rechts, unten
	HorizontalAlignment	Left	Ausrichtung
	Header	R1:	Beschriftung
	Text	x:Bind ViewModel.R1	Bindung der Eigenschaft an die Variable R1 im C#-Code
TextBoxU (out)	Width	200	Breite
	Margin	20, 10	Abstand nach links und rechts, oben und unten
	HorizontalAlignment	Left	Ausrichtung
	Header	U (out):	Beschriftung
	Text	x:Bind ViewModel. UOut	Bindung der Eigenschaft an die Variable UOut im C#-Code
TextBoxR2	Width	200	Breite
	Background	LightGray	Hintergrundfarbe
	Margin	20, 10	Abstand nach links und rechts, oben und unten
	HorizontalAlignment	Left	Ausrichtung
	Header	R2:	Beschriftung
	IsReadOnly	True	Schreibschutz
	Text	x:Bind ViewModel.R2	Bindung der Eigenschaft an die Variable R2 im C#-Code
Button	Width	200	Breite
	Margin	20, 10	Abstand nach links und rechts, oben und unten
	Background	LightGreen	Hintergrundfarbe
	Command	x:Bind ViewModel. CalcCommand	Bindung an die Methode CalcCommand im C#-Code
	Content	Calc	Beschriftung
	FontWeight	Bold	Schriftstärke

Listing 1. Definition des User Interfaces.

```
<Page
    x:Class="App1.Views.MainPage"
    …>

    <Grid x:Name="ContentArea" Margin="">
        <StackPanel Background="">
            <TextBox
                Width="200"
                Margin="20,20,0,0"
                HorizontalAlignment="Left"
                Header="R1:"
                Text="" />
            <TextBox
                Width="200"
                Margin="20,10"
                HorizontalAlignment="Left"
                Header="U (out):"
                Text="" />
            <Button
                Width="200"
                Margin="20,10"
                Background="LightGreen"
                Command=""
                Content="Calc"
                FontWeight="Bold" />
            <TextBox
                Width="200"
                Margin="20,10"
                HorizontalAlignment="Left"
                Background="LightGray"
                Header="R2:"
                IsReadOnly="True"
                Text="" />
        </StackPanel>
    </Grid>
</Page>
```

Den zugehören Quellcode sehen Sie in **Listing 1**. Sie können die Codierung der Oberfläche interaktiv vornehmen. Starten Sie die Anwendung und platzieren Sie die betreffende XAML-Datei in Visual Studio und die Anwendung nebeneinander auf dem Bildschirm (**Bild 9**). Änderungen im XAML-Code werden unmittelbar bei gestarteter Anwendung – ohne Speicherung – direkt übernommen und führen zu einer Aktualisierung der Anzeige. Das Feature heißt *Hot Reload* und ist Standard beim Erstellen von grafischen Benutzeroberflächen.

Interessant ist die Bindung der Eigenschaften mit der Bezeichnung *Text* der Steuerelemente des Typs *TextBox*. Hier finden Sie im XAML-Code einen Ausdruck nach dem Muster

```
Text="{x:Bind ViewModel.R1, Mode=TwoWay,
   UpdateSourceTrigger=PropertyChanged}"
```

Das bedeutet, dass die Eigenschaft *Text* an die Variable *R1* gebunden wird. Diese wird im *ViewModel* der Seite definiert. Zugrunde liegt das MVVM-Konzept. Die Oberfläche ist die *View* und die Daten werden

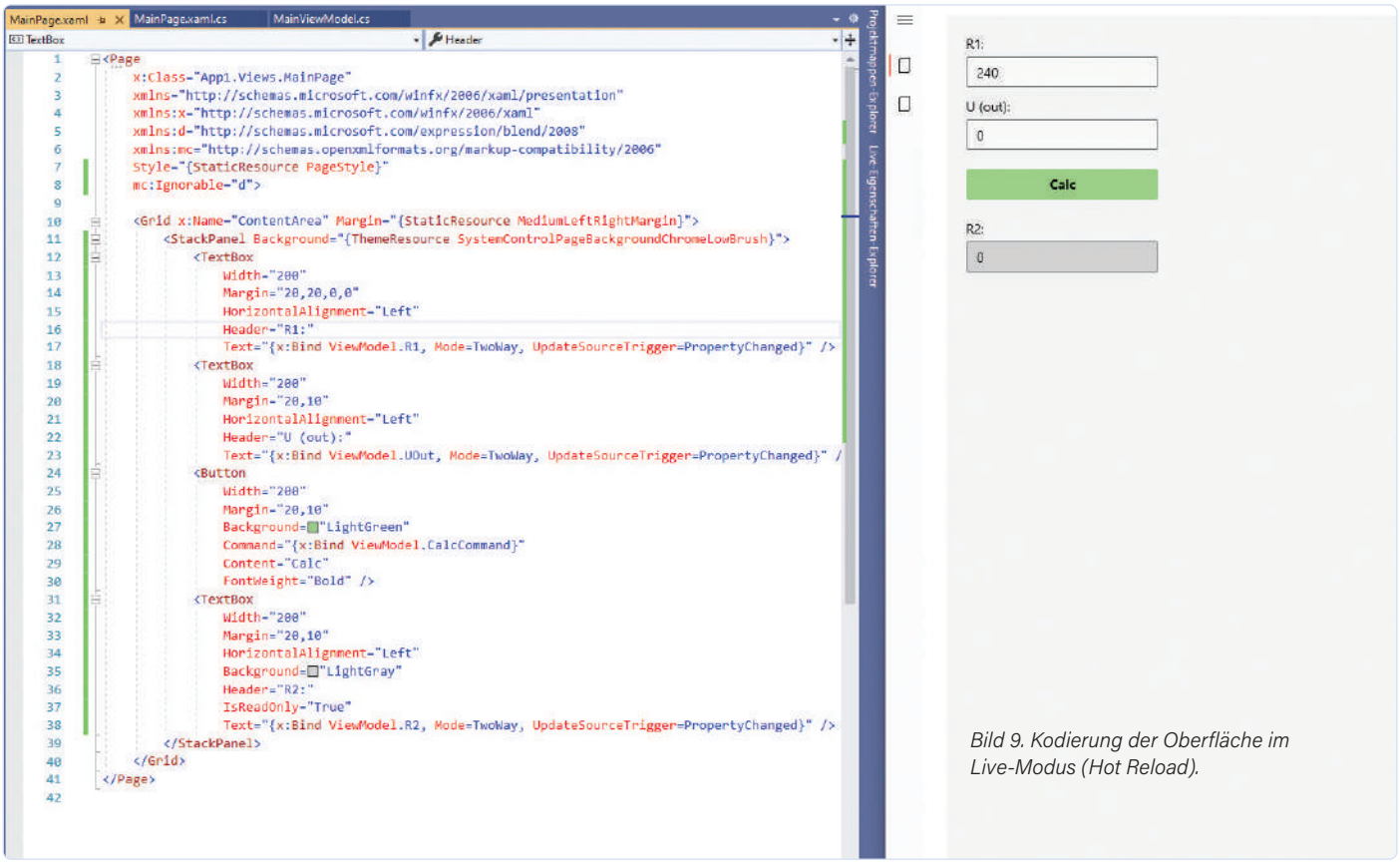


Bild 9. Kodierung der Oberfläche im Live-Modus (Hot Reload).

Listing 2. Programmlogik für den Rechner in C#.

```
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;

namespace App1.ViewModels
{
    public class MainViewModel : ObservableRecipient
    {
        public double R1 { get; set; } = 240;
        public double UOut { get; set; }

        private double r2;
        public double R2
        {
            get
            {
                return r2;
            }
            set
            {
                SetProperty(ref r2, value);
            }
        }


        public RelayCommand CalcCommand;
        public MainViewModel()
        {
            CalcCommand = new RelayCommand(calcComm
                andExecute);
        }

        private void calcCommandExecute()
        {
            R2 = (UOut / 1.25 - 1) * R1;
        }
    }
}
```

im *Model* verwaltet. Die Verbindung zwischen beiden Schichten stellt das *ViewModel* dar. Durch das MVVM-Konzept werden alle Schichten entkoppelt und lassen sich unabhängig voneinander entwickeln und warten. Informationen zum MVVM-Pattern finden Sie unter [4].

Programmlogik

Kommen wir zur Programmlogik, die in C# programmiert wird (**Listing 2**). Dazu wird zu jedem Fenster der Oberfläche (*View*) eine Programmdatei (*ViewModel*) zugeordnet. In unserem Beispiel ist es die Datei *MainViewModel*, welche der View *MainPage* zugeordnet

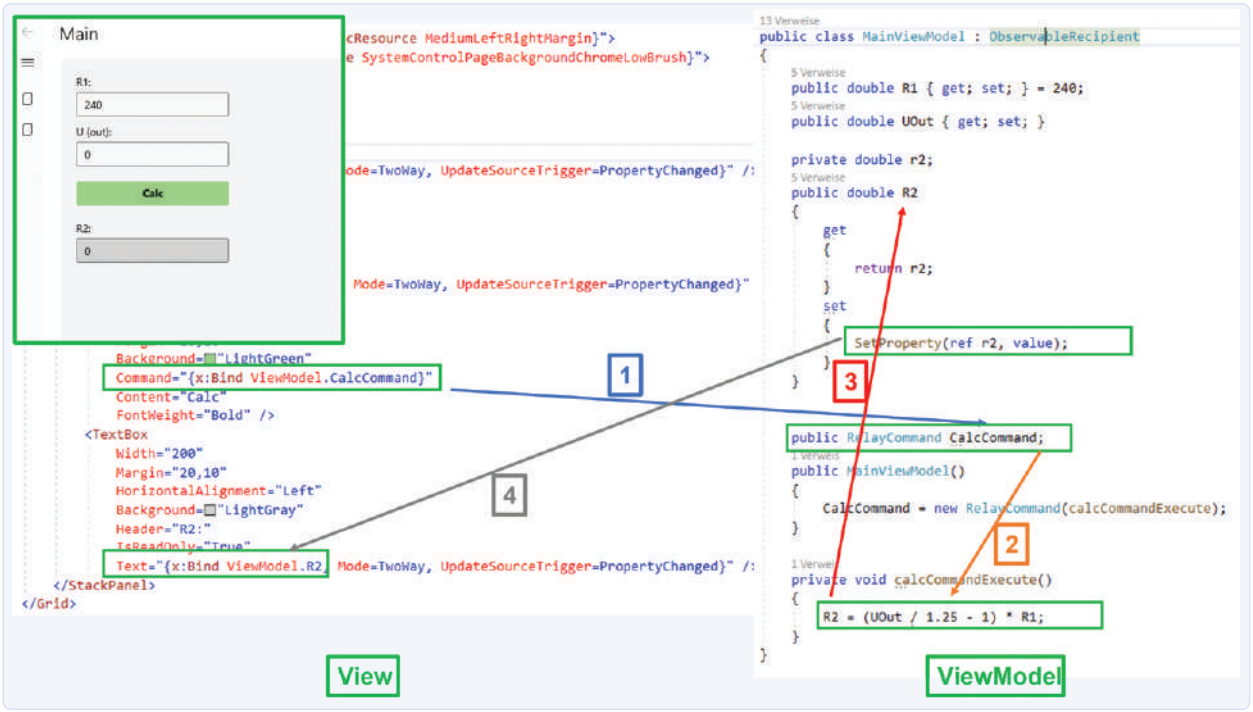


Bild 10. Zusammenhang zwischen View und ViewModel am Beispiel.

wird. Zum Programmcode sind ein paar Erläuterungen angebracht:

- **Import von Bibliotheken**: Das geschieht über die *uses*-Anweisung. In diesem Fall sind es zwei Bibliotheken für das MVVM-Pattern.
- **Definition von Eigenschaften**: Diese müssen öffentlich (*public*) sein, da wir von außen, in diesem Fall von der View, auf die Eigenschaften zugreifen.
- **Automatische Aktualisierung der Oberfläche**: Die Klasse `MainViewModel` leitet von der Basisklasse `ObservableRecipient` ab, welche durch den Projektassistenten beim Erstellen des Projektes angelegt wurde. Diese Klasse implementiert ihrerseits das so genannte `OnPropertyChanged`-Event, das dafür sorgt, dass bei Änderung eines Wertes einer Eigenschaft gebundene Elemente über die Änderung benachrichtigt werden. In unserem Fall ist die Eigenschaft `R2` von Interesse. Der Wert von `R2` wird im Programmcode errechnet. Dabei werden der so genannte *Setter* der Eigenschaft aufgerufen und über die Methode `SetProperty(…)` das eben beschriebene `OnPropertyChanged`-Ereignis ausgelöst. An die Eigenschaft `R2` (im ViewModel) ist wiederum die *Text*-Eigenschaft der *TextBox R2* gebunden. Bei Änderung von `R2` wird der dargestellte Wert in der zugehörigen TextBox automatisch aktualisiert. Das funktioniert dank Datenbindung.
- **Benutzeraktionen mittels Commands weiterleiten:** Drückt der Anwender auf den Button, dann wird ein *Command* ausgelöst. An dieses Command ist die Methode zur Berechnung gebunden. Auch hier sind Benutzeroberfläche und Programmcode nur über die Datenbindung miteinander gekoppelt.
- **Zuordnung von View und ViewModel:** Der Programmcode (Datei: *MainViewModel.cs*) wird der Oberfläche (Datei: *MainPage.xaml*) zugeordnet. Das geschieht in der so genannten Code-behind-Datei der Seite (Datei: *MainPage.xaml.cs*). Sie können sich das im Quellcode ansehen.
- **Berechnung:** Die Berechnung des Wertes von `R2` erfolgt in der Methode `calcCommandExecute(…)` nach oben genannter Formel, welche nach `R2` umgestellt wird.

Die Oberfläche ist also mittels Datenbindung „lose" an den Programmcode gebunden. Die eben beschriebenen Zusammenhänge zur Datenbindung sind in **Bild 10** am Beispiel visualisiert. Starten Sie die Anwendung und probieren Sie diese aus. Nach Eingabe von *R1* und U_{Out} erfolgt die Berechnung des Wertes für den zweiten Widerstand *R2* (**Bild 11**).

Damit haben wir das grundsätzliche Entwicklungsmodell für Desktop-Anwendungen mit dem Grafikframework WinUI 3 beschrieben. Aus heutiger Sicht wird es unter Windows zu einem neuen Standard werden und auch durch andere Entwicklungsumgebungen und -sprachen nutzbar sein. Die Vielfalt der grafischen Optionen, um moderne Anwendungen zu erstellen, ist beeindruckend.

Fazit und Ausblick

Sie haben die Möglichkeit, aus einer Vielzahl von Technologien zu wählen, um eine Anwendung für das Betriebssystem Windows zu erstellen. Der Trend – auch mit Blick auf Windows 11 – geht in Richtung des Einsatzes von WinUI 3. Damit können Desktop Anwendungen mit einem ansprechenden und modernen User Interface realisiert werden. Aus dieser Perspektive lohnt es sich, bei der Neuentwicklung einer Windows-Anwendung auf die WinUI 3 zu setzen, und für eine bestehende Anwendung eine Möglichkeit der Migration zu prüfen.

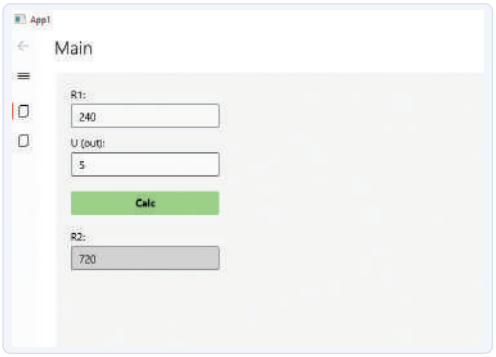


Bild 11. Die fertige Beispielanwendung.

Das Ergebnis sind Applikationen mit einer zeitgemäßen Oberfläche und einer guten User Experience. Einschränkungen beim Systemzugriff – wie bei der UWP – gibt es nicht.

210407-02

Ein Beitrag von

Text und Bilder: **Dr. Veikko Krypczyk**
Redaktion: **Jens Nickel**
Layout: **Giel Dols**

Sie haben Fragen oder Kommentare?

Haben Sie technische Fragen oder Kommentare? Bitte wenden Sie sich per E-Mail an die Elektor-Redaktion unter redaktion@elektor.de.

WEBLINKS

[1] Infos zu WinUI 3: https://docs.microsoft.com/en-us/windows/apps/winui/
[2] Visual Studio 2019: https://visualstudio.microsoft.com/de/vs/
[3] Projektseite zu diesem Beitrag: http://www.elektormagazine.de/210407-02
[4] Informationen zum MVVM-Pattern: https://de.wikipedia.org/wiki/Model_View_ViewModel

Insel-Solaranlagen

Elektrische Energie unabhängig vom Netz

Von **Dr. Thomas Scherer**

Was ist eine Insel-Solaranlage? Wo sind solche Lösungen notwendig und praktikabel? Was gibt es dabei zu beachten? Auf all diese Fragen und etwas mehr wird in diesem Artikel eingegangen.

Über Photovoltaik-Anlagen mit Netzanschluss haben wir bereits in Elektor 9-10/2021 berichtet [1]. In diesem Artikel geht es um vom öffentlichen Stromnetz isolierte, gewissermaßen autarke Solaranlagen. Damit generiert man auch dort elektrische Energie, wo ein Netzanschluss zu teuer wäre, wie etwa beim Gartenhaus eines Schrebergartens, oder gleich ganz unmöglich, wie etwa auf einem Motor- oder Segelboot. Es handelt es sich in der Regel um Systeme kleinerer Leistung, die kurzfristig Spitzenleistungen im Bereich von einigen Watt bis zu wenigen Kilowatt abgeben können. Außerdem gibt es in Zeiten der sinkenden Einspeisevergütungen neuerdings sogar vereinfachte stationäre häusliche Solaranlagen mittlerer Leistung im Bereich einiger kWp (Kilowatt peak ≈ Spitzenleistung der installierten Solarpanele), welche nichts ins öffentliche Netz einspeisen, sondern die geerntete Solarenergie mit Hilfe eines Akkus für den kompletten Selbstverbrauch nutzbar machen. Doch zunächst zu den kleineren Anlagen.

Prinzip und Details

Für eine Insel-Solaranlage braucht es minimal drei Komponenten: Zunächst die Solarzellen, dann einen Energiespeicher in Form eines Akkus und schließlich einen Laderegler, der dafür sorgt, dass der Akku nicht überladen wird. Für kleine Insel-Lösungen auf der typischen 12-V-Basis reicht das im Prinzip schon aus. Will man zusätzlich auch noch 230-V-Wechselspannung mit 50 Hz oder 60 Hz, braucht es als vierten Teil noch einen passenden Wechselrichter. **Bild 1** zeigt so eine Vier-Komponenten-Lösung. Das sieht sehr einfach aus, aber wie so oft stecken die Details voller kleiner Teufel. Im Folgenden wird daher speziell auf die einzelnen Komponenten eingegangen.

Hier ein Praxisbeispiel: Aufgrund der einfacheren Handhabung und des günstigeren Preises wegen hat sich mein guter Freund Klaus bei seinem Gartenhaus für eine 12-V-Anlage entschieden. Für die Planung und die Auslegung der Komponenten mussten zunächst zwei Fragen beantwortet werden.

Energie & Leistung

Die erste Frage ist, wieviel elektrische Energie vorgehalten = gespeichert werden soll. Da sich dies direkt auf die Kapazität des geplanten Akkus auswirkt, muss man natürlich überschlagen, welche mittlere Last von der Inselanlage versorgt werden soll. Relevant ist außerdem, wieviel Tage die Anlage bei Bewölkung „überstehen" soll. Klaus wollte in seinem Gartenhaus eine Bohrmaschine

betreiben oder sich eine Tasse Tee brauen, aber so etwas nur selten, weshalb man diesen Energieverbrauch fast vernachlässigen kann. Jedoch sollte durchgängig ein 12-V-Kühlschrank mit einer mittleren Leistungsaufnahme von etwa 20 W versorgt sein, damit immer kühles Bier zur Verfügung steht. Das Ganze müsste mindestens einen Tag ohne Sonnenschein funktionieren.

Die zweite Frage gilt der benötigten Spitzenleistung. Davon hängt nämlich die Strombelastbarkeit des eingesetzten Akkus ab und in der Folge sind davon auch der Laderegler und ein eventuell vorhandener Wechselrichter betroffen. In der Regel ist diese Frage recht einfach zu beantworten. Beim Schrebergartenhäuschen von Klaus waren das zum Beispiel 1 kW, denn er wollte einen Wasserkocher, eine normale Bohrmaschine und gelegentlich auch eine Wasserpumpe an 230 V betreiben.

Solarpanel

In 24 Stunden dürfte der Kühlschrank im Beispiel von Klaus maximal 500 Wh verbrauchen. Klaus lebt im sonnenverwöhnten Freiburg, mochte aber das Solarpanel nicht aufs Dach montieren, weil es dort von einem Kirschbaum abgeschattet worden wäre. Stattdessen wurde das Panel senkrecht an eine recht gut nach Süden orientierte Wand des Gartenhauses geschraubt. Dies reduziert die mögliche Energieausbeute um bis zu 30 % gegenüber dem optimalen Winkel zur Sonne. Man muss das Solarpanel also um fast 40 % überdimensionieren, um diesen Verlust zu kompensieren. Kein großes Problem, da Platz genug da ist und Solarpanels in den letzten Jahren preiswert geworden sind. Der Vorteil der senkrechten Montage ist, dass im Winter kein Schnee auf dem Panel liegen bleibt, durch den dann günstigeren Einstrahlwinkel die Ausbeute steigt und somit im optimalen Fall sogar an sonnigen Tagen im Winter das Bier kühl bleibt.

Nun zur Berechnung der nötigen solaren Leistung: In Freiburg kann man mit über 1.200 kWh/m² im Jahr rechnen. Für den avisierten täglichen Verbrauch sollte eine Sicherheitsreserve von 100 % ausreichen. Bei den angenommenen 500 Wh/d (Frühjahr bis Herbst) sollte man also 1 kWh/d zu erntende Energie anpeilen. Kalkuliert man mit acht Sonnenstunden pro Tag, kommt man auf eine notwendige Leistungsabgabe des Panels von etwa 125 Wp. Hinzu kommt die Kompensation wegen senkrechter Montage und man landet bei 175 Wp. Daher ist es ein 180-W-Panel (**Bild 2**) geworden – keine schlechte Wahl.

Akku(s)

Der Energiespeicher sollte bei einem Tag Reserve ≥500 Wh fassen. Bei 12 V Nennspannung wäre also ein Akku mit einer Kapazität von mindestens 40 Ah erforderlich. Da ein

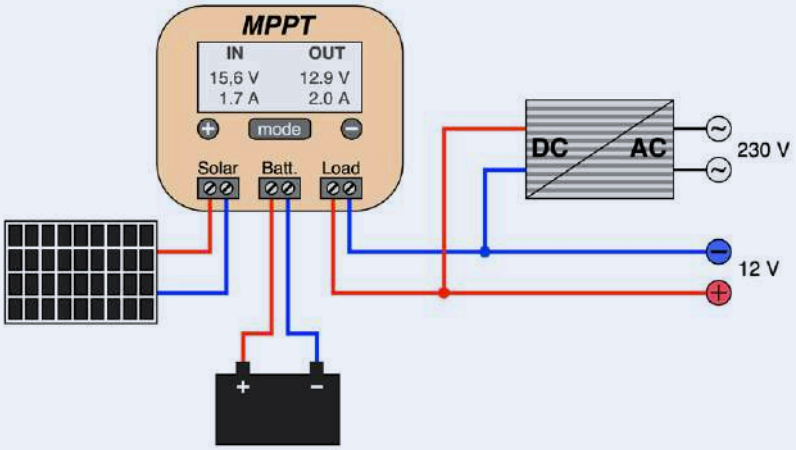


Bild 1. So werden die üblichen vier Komponenten einer solaren Inselanlage üblicherweise verschaltet. Der Inverter rechts ist nur erforderlich, wenn man für 230 V gedachte Geräte betreiben will.

Bild 2. Das 12-V-Solarpanel am Gartenhäuschen von Klaus ist senkrecht montiert und liefert 180 Wp.

Bild 3. Drei 12-V-Blei-Gel-Akkus mit je 36 Ah parallelgeschaltet als Energiespeicher im Gartenhäuschen von Klaus.

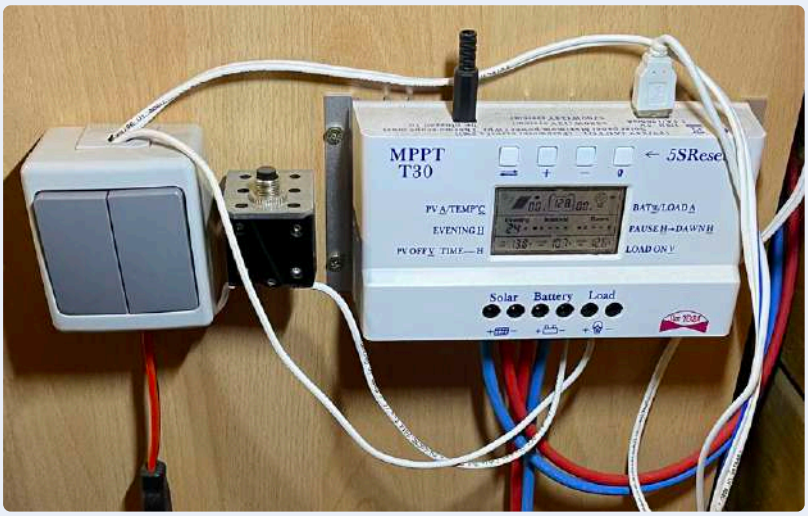


Bild 4. Von links nach rechts: Lichtschalter, elektromagnetische 30-A-Sicherung und MPPT-Laderegler.

Bild 5. Was so aussieht wie dieser Regler, hat sicher kein MPPT an Bord – selbst wenn diese vier Buchstaben aufgedruckt sein sollten (Bild: United States Department of Energy).

Inverter mit 1 kW Ausgangsleistung vorgesehen war, muss man im Hinterkopf behalten, dass dieser bei maximaler Last einen Eingangsstrom von gut 85 A benötigt. Dieser Aspekt ist wichtig bei der Akku-Auswahl. Zunächst musste der Akkutyp festgelegt werden. Ein Lithium-Akkupaket mit 40 Ah könnte aufgrund des niedrigen Innenwiderstands locker mit diesem Strom (etwa 2C = dem doppelten Strom, den der Akku 1 h lang liefern kann) fertig werden. Aber solche Akkus kosten schnell über 250 € und sind komplexer in der Handhabung. Daher wollte Klaus mit der simplen Blei-Akku-Lösung vorliebnehmen, die es für einen Bruchteil dieses Preises gibt. Naheliegend wäre eine Auto-Batterie, denn die packt auch hohe Spitzenströme. Ihre Nachteile (geringer Wirkungsgrad, geringe Zyklenbelastbarkeit und höhere Selbstentladung) führten zum Kompromiss Blei-Gel-Akku. Dieser Typ wiederum mag hohe Entladeströme nicht wirklich, weshalb sich Klaus letztlich für zwei parallelgeschaltete 12-V-Blei-Gel-Akkus mit je 36 Ah entschieden hatte. Diese Lösung bietet rechnerisch 864 Wh an gespeicherter Energie für knapp unter 150 €. Scheint die Sonne voll, liefert das gewählte Solarpanel so viel Energie, dass die Akkus an einem schönen Sommertag in Freiburg locker vollständig geladen werden. Der Energievorrat reicht gut für 1,5 Tage ohne Sonnenschein. Ich selbst hatte Bedenken wegen des hohen Stroms für den Inverter, aber Klaus wollte es zunächst darauf ankommen lassen und meinte, dass er immer noch einen Akku dazu kaufen könne, falls sie zu „weich" sein sollten. Als die Anlage installiert und die Akkus voll geladen waren, zeigte ein erster Test mit dem 1-kW-Wasserkocher, dass während des Kochens die Nennspannung an den Akkupolen auf 11,7 V einbrach. Es war problemlos möglich, einen halben Liter Wasser zum Kochen zu bringen, aber der Wirkungsgrad der Akkus (Ladeenergie/Entladeenergie) bei diesen hohen Strömen lag geschätzt höchstens noch bei 50 % und den Akkus tat diese Extremlast auch nicht gut. Folglich wurde ein dritter 36-Ah-Akku nachgeordert und dazu geschaltet (**Bild 3**). Jetzt betrug die Entladespannung bei 85 A anfangs noch 12,6 V, was akzeptabel war. Außerdem werden nun fast 1,3 kWh gespeichert, was über zwei Tage Reserve garantiert.

Bild 6. Der 1-kW-Inverter von Ective erwies sich bislang als sehr stabil und zuverlässig.

Laderegler

Schaut man sich bei eBay oder spezialisierten Händlern um, wird man von der Vielfalt an Ladereglern geradezu erschlagen. 10-A-Regler gibt es schon ab 15 €. Ein 180-Wp-Panel liefert an 12 V aber immerhin bis zu 15 A. Es muss also mindestens ein 20-A-Regler sein, den es für rund 20 € gibt. Wird der Laderegler aber so angeschlossen wie in Bild 1, sollte man besser eine 100-A-Variante wählen, die mit knapp unter 50 € zu Buche schlägt. So viel vorab.

Ein Laderegler hat die Aufgabe, den Akku mit der vom Panel kommenden Energie zu laden und die maximale Ladeschlussspannung einzuhalten, damit der angeschlossene Akku nicht überladen und somit beschädigt wird. Fast alle Laderegler steuern auch einen Lastanschluss, indem sie dafür sorgen, dass die Last bei Unterspannung abgetrennt und der Akku somit vor einer Tiefentladung geschützt wird. Da sie alle über einen Mikrocontroller verfügen, können fast alle Solar-Laderegler mit verschiedenen Akkutypen (normaler Blei-Akku, Blei-Gel-Akku und Lithium-Akku) und automatisch sowohl mit 12-V- als auch 24-V-Nennspannung umgehen. Außerdem lassen sich die Spannungen für Ladeschluss und Entladeschluss oft manuell festlegen.

Der nächste Aspekt ist die Ladetechnik, die ein Laderegler beherrscht. Dabei gilt: Alle preiswerten Exemplare arbeiten mit einer PWM-Regelung – auch wenn MPPT draufsteht. Aufdrucke sind in China nämlich sehr preiswert – sie kosten lediglich die Druckerfarbe. „Echtes MPPT" ist besser, aufwändiger und daher deutlich teurer!

Bei einer PWM-Regelung wird der Ladestrom so eingestellt, dass die Abgabespannung des Panels gerade noch über der aktuellen Akkuspannung liegt. Es wird also über einen weiten Regelbereich einfach mit dem maximal möglichen Strom geladen, der von der Beleuchtung des Panels, seiner „Größe" und dem Ladezustand des Akkus abhängt. Außer einem simplen Mikrocontroller braucht es dazu nur einen Power-MOSFET – das ist preiswert, aber nicht optimal. Die Abgabeleistung eines Panels ergibt sich aus dem Produkt von Spannung und Strom. Dabei gibt es für jedes Panel und jede Beleuchtung einen optimalen Punkt maximaler Abgabeleistung, bei dem die Spannung des Panels fast immer über der aktuellen Akkuspannung liegt. Das MPPT-Verfahren (Maximum Power Point Tracking) ermittelt diesen „sweet spot" permanent und steuert einen nachgeordneten Step-Down-Schaltregler über dessen Stromaufnahme so, dass sich immer die maximale Leistungsabgabe ergibt. Im optimalen Fall ist die Ausbeute mit MPPT um bis zu 30 % höher als bei PWM, aber das kostet eben. Die preiswertesten MPPT-Regler gibt es aktuell nicht unter 50 €. Für Markengeräte muss man mit mindestens 100 € rechnen. Der 30 A-Solarregler von Klaus in **Bild 4** ist ein preiswerter MPPT-Regler für etwa 60 €. Die Ausbeute war ihm also wichtig. Sollten Sie auch einen MPPT-Regler wollen, fallen Sie nicht auf Exemplare wie das von **Bild 5** herein. Diese Typen gibt es in allen Farben und mit unterschiedlichsten Bezeichnungen.

Inverter

Falls man selbst eine Wechselspannung von 230 V erzeugen möchte, kommt man um einen Inverter nicht herum. Von preiswerten Exemplaren mit Fantasieangaben bezüglich Leistung und nicht annähernd sinusförmigem Spannungsverlauf sollte man gleich ganz die Finger lassen. Weiter wichtig: Die Angabe der maximalen Dauerleistung am Ausgang bezieht sich auf eine ohmsche Last. Die 1 kW des Inverters von Klaus harmonieren mit seinem 1-kW-Wasserkocher mit 500 ml Fassungsvermögen 1:1. Völlig anders ist die Situation bei induktiven

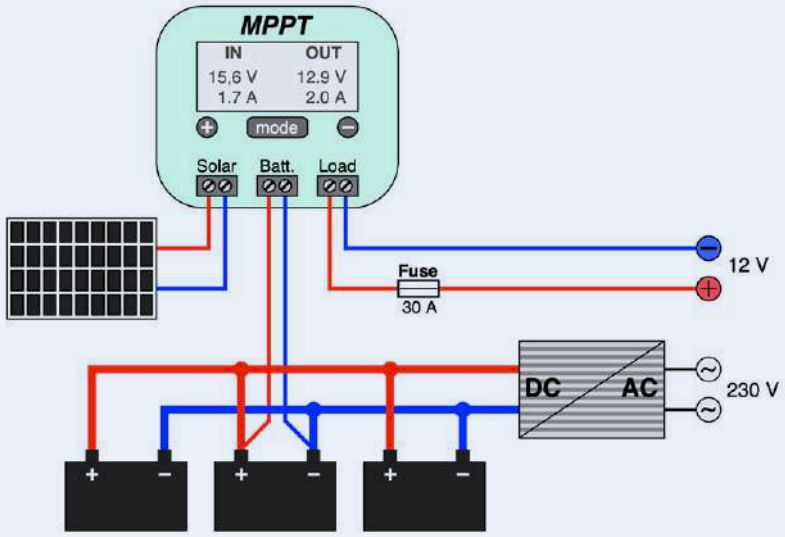


Bild 7. Bei der Lösung von Klaus ist der Inverter direkt mit den Akkus verbunden und der 12 V-Ausgang des Ladereglers extra abgesichert.

Bild 8. Die LiFePo-Akkus von Martin beim Test – vor dem Einbau in sein Boot (Bild: Martin Jepkens).

Bild 9. Das von Martin und Detlev verwendete faltbare Solarpanel liefert 120 Wp und wird bei Fahrt unter Deck verstaut (Bild: Martin Jepkens).

(häufiger) und kapazitiven Lasten (seltener). Hier muss man zusätzlich die Blindleistung berücksichtigen - die aufgenommene Leistung ist immer höher als die reine Wirkleistung. Am problematischsten sind Elektromotoren, denn sie haben zum Teil hohe Anlaufströme, welche bei zu knapp dimensionierten Invertern den integrierten Überlastschutz triggern. Eine Reserve von 100 % ist beim Betrieb von Motoren auch bei guten Invertern nicht übertrieben. Der 1-kW-Inverter von Klaus in **Bild 6** packt problemlos Bohrmaschinen und eine Wasserpumpe mit 450 W Nennleistung. Er war nicht unter 200 € zu haben.

Verkabelung

Wie in den bisherigen Bildern zu erahnen wurde die Verbindung von Solarpanel mit Laderegler und Akku mit Laderegler durch feindrähtige Litze mit 6 mm² vorgenommen (nicht abgebildet sind die 12-V-Ausgänge via Auto-Zigarettenanzünder-Buchsen). Die Akkus selbst aber wurden durch Litzen mit 16 mm² parallelgeschaltet. Die Dicke der Drähte richtet sich nach dem auftretenden Strömen – hier sollte man nicht am falschen Ort sparen. Außerdem wurde der Inverter direkt via 16-mm²-Leitungen an die Akkus geklemmt, um Verluste über den integrierten MOSFET des Ladereglers zu umgehen. Das kann man aber nur machen, wenn sich der Inverter (wie der hier eingesetzte) bei Unterspannung selbst abschaltet und so den Akku vor Tiefentladung schützt. Der Inverter wird nur eingeschaltet, wenn tatsächlich 230 V benötigt werden, denn sein Ruhestrom von einigen zig mA würde sonst unnötig Energie verbrauchen. Die resultierende Verschaltung ist in **Bild 7** zu sehen.

Andere Insel-Anlagen

Die Elektrifizierung des Gartenhäuschens von Klaus ist ein typisches Beispiel für die Installation einer solaren Insel-Anlage. Hierfür gibt es bei diversen Anbietern auch fertige Pakete aus Solarpanel, Regler und Inverter unterschiedlicher Leistung. Setzt man auf Wind- statt auf Solarenergie, gibt es auch hierfür passende Generatoren und Regler, doch das Prinzip bleibt im Wesentlichen gleich. Ich selbst hatte letztes Jahr meinen Rasenmäher-Roboter autark gemacht [1]. Hier war nur ein 50-W-Panel vorgesehen und ein einfacher PWM-Regler. Ein Inverter war überflüssig. Mittlerweile habe ich die Akkukapazität von 12 Ah auf nunmehr 30 Ah erhöht, da es dieses Jahr einige Regentage gab. Außerdem wurde der PWM-Regler dieser Tage durch einen besseren und auch teureren MPPT-Regler ersetzt, damit auch noch im Spätherbst genug Energie zum Mähen geerntet werden kann.
Es gibt aber auch noch gänzlich andere Anwendungen für Insel-Anlagen: Zwei meiner Freunde haben Boote. Während Martin mit seinem größeren Stahlboot (Verdränger) durch die niederländischen Flusslandschaften navigiert, macht Detlev mit seinem schnellen Sportboot (Gleiter) das Mittelmeer unsicher. Beiden gemeinsam ist, dass sie oft Tage von Häfen und sonstigen Anlegestellen entfernt sind und sich daher mehr Autonomie bezüglich des Kühlschranks wünschen – dabei geht es nicht nur um Bier, sondern auch um essbare Lebensmittel. Damit sie nicht in unökologischer und ineffizienter Weise häufig den Motor laufen lassen müssen, um den Bord-Akku zu laden, haben beide eine Solaranlage nachgerüstet.
Martin ist selbst ein ausgefuchster Elektroniker

Bild 10. Anleitung zum Einbau des Ladereglers in das Boot von Detlev.

und nicht unbedingt auf meine Unterstützung angewiesen. Er diskutiert seine Überlegungen aber manchmal mit mir. So beschäftigte ihn die Frage, ob die Lichtmaschine seines Bootsmotors überlastet würde, wenn er gelegentlich einen dicken, fetten 200-Ah-LiFePo-Akku dazuschaltet. Die Gefahren dieses Vorgehens werden zum Beispiel in einem YouTube-Video erläutert [2]. **Bild 8** zeigt seine Akku-Anordnung bei den von ihm vorgenommenen Kapazitätstests. Für LiFePo-Technik hat er sich vor allem aufgrund der vielen möglichen Ladezyklen entschieden. Außerdem brauchen LiFePos verglichen mit Blei-Akkus weniger Platz. Bord- und Starterbatterie sind in Martins Boot entkoppelt. Um die Lichtmaschine zu schonen, werden die Akkus über einen eigenen Laderegler geladen, wenn der Motor läuft. „Für unterwegs" beziehungsweise ohne Netzanschluss in einem Hafen ist auch noch ein faltbares 120-Wp-Solarpanel samt Solar-Regler vorgesehen (**Bild 9**).

Beim Sportboot von Detlev wäre ein fest installiertes Solarpanel schon aus Platzmangel nicht denkbar. Deshalb hat er sich für das gleiche Panel wie Martin entschieden. Beide wussten nichts von der Wahl des jeweils Anderen. Detlev ist allerdings kein Elektroniker, und er wollte zunächst seinen extra 120-A-Bordakku weiter nutzen, weil er noch recht neu war. Also musste ich für ihn rechnen und im auch mitteilen, dass die Verwendung der Buchse des Zigarettenanzünders auf seiner „Brücke" zum Anschluss des Solarpanels aus Zuverlässigkeitsgründen keine gute Idee ist. Ich hatte ihm die Verwendung von wasserdichten Neutrik-Steckverbindungen vorgeschlagen. Daher durfte ich die Stecker auch vorverdrahten und eine Anleitung (**Bild 10**) für seinen Bootsbauer erstellen, damit der das Ganze am Hafen in Istrien einbauen kann. Die Kombination aus faltbarem 120-Wp-Panel plus MPPT-Laderegler von Victron Energy hat knapp unter 500 € gekostet. Der Laderegler verfügt über Bluetooth, so dass alle Parameter und Kurven per Smartphone mit einer App überwacht werden können.

Das Haus als Halbinsel

In Zeiten abnehmender Einspeisevergütungen beginnt eine Variante stationärer Solaranlagen interessant zu werden, die möglichst alle geerntete Solarenergie dem

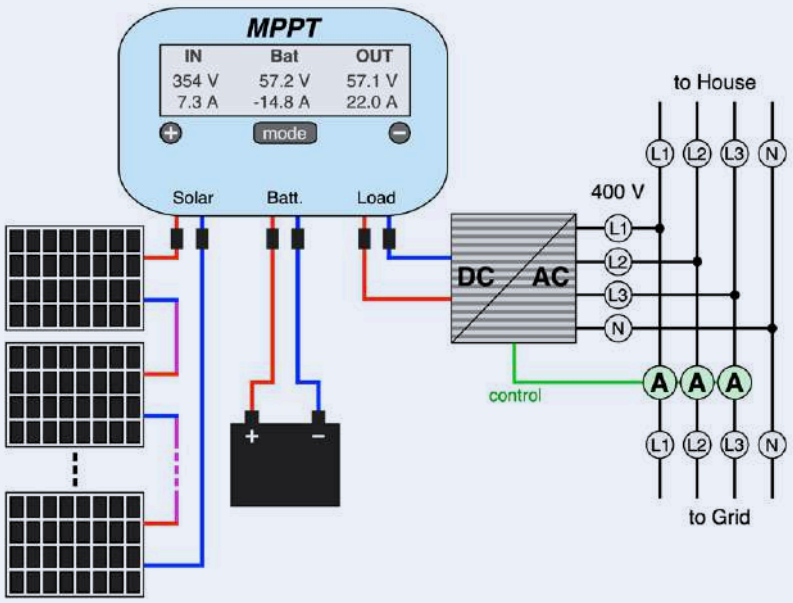


Bild 11. Solare Quasi-Inselanlage für ein Einfamilienhaus. Die Dreiphasen-Stromerfassung steuert den Wechselrichter so, dass keine elektrische Energie ins Netz abgegeben wird.

häuslichen Verbrauch zuordnet. Ein Array von zum Beispiel zehn Solarpanels liefert heute 3,75 kWp. Ein passender MPPT-Solarregler lädt einen LiFePo-Akku mit einer Energiekapazität von beispielsweise 6,5 kWh und ein dreiphasiger Wechselrichter wird von einer Stromüberwachungseinheit (die drei Amperemeter rechts unten in **Bild 11**) so gesteuert, dass unter keinen Umständen elektrische Energie ins Netz eingespeist wird. Aller „Strom" wird so selbst verbraucht. Da für das Jahr 2022 in Deutschland mit kWh-Preisen von um die 36 ¢ gerechnet wird, ist das durchaus attraktiv, denn man spart sich einen komplexeren und auch teureren einspeisefähigen Wechselrichter mit integrierter Ladeelektronik für den Akku sowie viel Bürokratie, was nicht zu verachten ist.

Bei der Anlage von Bild 11 summieren sich die Einsparungen für die vereinfachte Anlage auf 1.000 € bis 2.000 €. Es würde ein paar Jahre brauchen, bis man das mit der Einspeisevergütung wieder raus hätte. Das teuerste Element ist hier der Akku. Für einen anschlussfertigen 6,5-kW-LiFePo-Akku muss man etwas über 3.000 € kalkulieren. Bei garantierten 6.000 Ladezyklen mit einem Hub von 90 % ergeben sich etwa 36 MWh, die über den Akku geflossen sind. Das ergibt Kosten von etwa 9 ¢/kWh. Da der Akku dann noch nicht tot ist, kann man mit noch geringeren Kosten pro kWh für den Akku rechnen. Über die Laufzeit spart man mit solch einer Lösung grob geschätzt Stromkosten von mehr als 13.000 € durch den damit möglichen hohen Selbstverbrauch. Lädt man mit solch einer Anlage auch noch ein Elektroauto (mit geringer Leistung), hat sich die Anlage in wenigen Jahren amortisiert.

210644-02

Ein Beitrag von

Text und Bilder (soweit nicht anders angegeben): **Dr. Thomas Scherer**
Redaktion: **Jens Nickel**
Layout: **Harmen Heida**

Sie haben Fragen oder Kommentare?

Bei technischen Fragen können Sie sich gern an die Elektor-Redaktion wenden unter der E-Mail-Adresse redaktion@elektor.de.

PASSENDE PRODUKTE

- PeakTech Stromzange 4350 (SKU 18161)
  www.elektor.de/18161
- Pokit Meter – Portables Multimeter, Oszilloskop und Logger (SKU 19854)
  www.elektor.de/19854
- PeakTech 3445 True RMS Digital Multimeter mit Bluetooth (SKU 18774)
  www.elektor.de/18774

WEBLINKS

[1] Thomas Scherer, „Balkonkraftwerk", ElektorMag 9-10/2021: https://www.elektormagazine.de/210326-02
[2] Thomas Scherer, „Solaranlage für Mähroboter", ElektorMag 7-8/2021: https://www.elektormagazine.de/200553-02
[3] Victron Energy: „How to not blow up your Alternator when charging Lithium": https://www.youtube.com/watch?v=jgolocPgOug

Treten Sie jetzt der Elektor Community bei!

Jetzt elektor GOLD membership Mitglied werden!

- Komplettes Webarchiv ab 1970
- 6x Elektor Doppelheft (Print)
- 9x Digital (PDF) inkl. Elektor Industry
- 10% Rabatt im Online-Shop und exklusive Angebote
- Elektor Jahrgangs-DVD
- Mit Tausenden von Mitgliedern des Online-Labors gemeinsam entwickeln mit Zugang zu über 1.000 Gerber-Dateien und direktem Kontakt zu unseren Experten!
- Veröffentlichen Sie Ihr eigenes Projekt oder verkaufen Sie direkt über unseren Shop!

Auch erhältlich

Die digitale elektor GREEN membership Mitgliedschaft!

- Zugang zu unserem Webarchiv
- 10% Rabatt in unserem Online-Shop
- 6x Elektor Doppelheft (PDF)
- Exklusive Angebote
- Zugang zu über 1.000 Gerber-Dateien

www.elektor.de/mitglied

elektor design > share > earn

elektor e-zine

Your dose of electronics

Jede Woche, in der Sie den Elektor e-zine Newsletter nicht abonnieren, ist eine Woche mit großartigen Artikeln und Projekten zum Thema Elektronik, die Sie verpassen!

Also, worauf warten Sie noch? Melden Sie sich heute für unseren Elektor e-zine Newsletter unter www.elektor.de/ezine an und erhalten Sie zusätzlich ein kostenloses Raspberry Pi Projektbuch!

Was können Sie erwarten?

Redaktioneller Elektor-Newsletter
Jeden Freitag erhalten Sie die wichtigsten Artikel und Projekte der Woche. Wir zeigen MCU-basierte Projekte, IoT, Programmierung, KI und mehr!

Elektor-Newsletter mit exklusiven Angeboten
Verpassen Sie nicht unsere Shop-Angebote, jeden Dienstag und Donnerstag haben wir eine besondere Aktion für Sie.

Mailing von externen Partnern
Sie wollen über die laufenden Aktivitäten in der Branche informiert bleiben? Dann gibt Ihnen diese E-Mail die besten Einblicke. Unregelmäßig, aber immer mittwochs.

elektor
design › share › sell