FRANZIS

MACH'S EINFACH

Erste Schritte:
EIGENE IoT- LÖSUNGEN MIT DEM ESP32

Mikrocontroller, Internet und PC

UDO BRANDES

Der Autor

Udo Brandes war viele Jahre Systementwickler beim Rechenzentrum der Finanzverwaltung des Landes Nordrhein-Westfalen. Nach verschiedenen Tätigkeiten bei der Stadt Wuppertal arbeitet er seit einigen Jahren als selbstständiger Programmierer und IT-Entwickler. Ansätze und Möglichkeiten der Hausautomation sind Zukunftsthemen, die den Autor faszinieren und mit deren Chancen und Risiken er sich seit Langem intensiv befasst.

FRANZIS

MACH'S EINFACH

Erste Schritte:
EIGENE IoT- LÖSUNGEN MIT DEM ESP32

Mikrocontroller, Internet und PC

UDO BRANDES

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über *http://dnb.ddb.de* abrufbar.

Lektorat: Ulrich Dorn
Satz: Nelli Ferderer (nelli@ferderer.de)
Art & design: Anna Lena Ibiş

ISBN 978-3-645-20665-7

EINLEITUNG

Das Internet und der Zugriff auf seine vielfältigen Inhalte ist von zentraler Bedeutung für das tägliche Leben. Nicht nur im wissenschaftlichen Bereich, sondern auch im privaten Sektor findet es mit Diensten wie E-Mail, Social Media, Onlineshopping und Onlinebanking eine aus dem täglichen Leben kaum mehr wegzudenkende Anwendung.

Aber dies alles ist kein statischer Zustand, sondern einer ständigen, rasanten Weiterentwicklung unterworfen. Das Internet wird zunehmend als Basis genutzt, um anhand entsprechender Technologien physische und virtuelle Gegenstände miteinander zu vernetzen und sie zusammenarbeiten zu lassen. Hierfür wurden die Begriffe »Allesnetz« und »Internet of Things« (Kurzform IoT, *Internet der Dinge*) geprägt.

Aus dem industriellen Bereich stammend, haben diese Ansätze zunehmend auch in den privaten Sektor Eingang gefunden. Schlagwörter sind hier das Smart Home, also das vernetzte und intelligente Zuhause, und die Wearables, also Computersysteme, die unmittelbar am Körper getragen werden oder in die Kleidung integriert sind und auf den Nutzer oder dessen Umwelt bezogene Daten registrieren und verarbeiten.

Häufig genügen aber die industriell gefertigten Lösungen nicht den eigenen Anforderungen, sind zu kostspielig oder vernetzen sich selbstständig mit beliebigen Servern und geben damit unerwünscht private Daten preis. Abhilfe können eigene Lösungen schaffen. Das ist nicht ungewöhnlich und auch nicht völlig neu.

Seit den Anfängen der Elektrotechnik und Elektronik realisieren Enthusiasten aus purem Interesse für die Sache selbst oder mit dem Ziel maßgeschneiderter individueller Lösungen zum Teil anspruchsvolle Projekte mit den jeweils verfügbaren Bauteilen und Möglichkeiten. Waren in früheren Zeiten Röhren, mächtige Transformatoren oder dicke Kondensatoren die wesentlichen Arbeitsmaterialien, werden heute vielfach Mikrocomputer (z. B. der Raspberry Pi) oder Mikrocontroller (z. B. der ESP32) eingesetzt.

Dabei verschwimmen die Grenzen zwischen den beiden Bauteilklassen häufig. Ein Mikrocomputer mit einem Mikroprozessor als zentraler Recheneinheit wird häufig mit einem Kleinrechner assoziiert, der, wenn auch mit deutlich abgespeckter Leistung, die wesentlichen Aufgaben eines normalen Desktopcomputers abdecken kann. Der Mikrocontroller hingegen ist eher auf die Bewältigung technischer Anforderungen zugeschnitten. Er tritt im Alltag häufig in Gestalt von sogenannten eingebetteten Systemen in Gebrauchsartikeln wie Staubsaugern, Fernbedienungen, Motorsteuerungen etc.) auf.

Für beide Bauteilklassen ist im Handel eine sehr große Anzahl verschiedener Chips erhältlich, die in eine noch größere Anzahl von Modulen und Boards verbaut sind. Für sich ist diese Hardware aber kaum sinnvoll einsetzbar. Richtig interessant wird es erst, wenn diese Bauteile mit anderen Komponenten wie z. B. Sensoren oder Schaltern verbunden werden und Informationen austauschen.

Für den Informationsaustausch ist dann wieder eine gemeinsame Arbeitsgrundlage, eine Programmiersprache, erforderlich. So kommt also der heutige Elektronikbegeisterte nicht umhin, sich mit Programmiersprachen zumindest in Ansätzen zu beschäftigen. Auch sind Grundkenntnisse zur Kommunikation über das World Wide Web von Vorteil oder sogar unabdingbar.

Ob nun ein Mikrocontroller oder ein Mikrocomputer für die anstehenden Vorhaben die beste Wahl ist, kann an dieser Stelle nicht beantwortet werden. Schwerpunkt dieses Buchs sind Mikrocontroller der ESP32-Baureihe. In den Anwendungsfällen, in denen ein Mikrocomputer die Aufgabe besser erledigen kann, werden realisierbare Lösungsvorschläge unterbreitet und ebenso ausführlich dargestellt.

Dabei wird versucht, schrittweise vom Einfachen zum Schwierigen vorzudringen und komplexere Aufgaben in kleine Teile zu zerlegen, die sich dann zu einem größeren Ganzen zusammenfügen. Sicherlich ist es nicht notwendig, die Kapitel sequenziell abzuarbeiten. Aber vielleicht stockt der Umsetzungsfluss an der einen oder anderen Stelle, sodass die Projektarbeit wieder Fahrt aufnehmen kann, wenn Informationen aus vorhergehenden Abschnitten vor Augen sind.

Neben den textlichen Ausführungen spielen Abbildungen eine tragende Rolle. Getreu dem Motto, ein Bild sagt mehr als tausend Worte, lässt sich mit Bildern – trotz des Platzes, den sie nun mal einnehmen – die zu vermittelnde Botschaft einfach besser darstellen.

Eine Sketch-Referenz, die als Anhang beigefügt ist, stellt in Teilschritten entwickelte umfangreichere Programme im Ganzen dar.

Für wen das Buch gedacht ist

Das Buch richtet sich an alle …

- die neugierig sind und sich gern mit den Möglichkeiten von Mikrocontrollern im Zusammenspiel mit dem Internet und dem Desktopcomputer beschäftigen,
- die einen Einstieg in Lösungen suchen, um Abläufe zu automatisieren und zu steuern (Smart Home, IoT),
- die mit einfachen Mitteln schnell loslegen möchten,
- die aber auch etwas weiterführende und vertiefende Anregungen, Vorschläge und praktische Anleitungen suchen,
- die auf funktionierende, aussagekräftige und ausführliche Schritt-für-Schritt-Erläuterungen Wert legen,
- die in komprimierter Form Tipps für eigene Entwicklungen suchen und
- die sich ein Stück weit unabhängig von industriellen Lösungen machen wollen.

Voraussetzungen

Im Prinzip reicht es für viele der im Buch dargestellten Beispiele aus, wenn man über ein Entwicklungsboard mit dem ESP32, einen Internetzugang und einen Desktopcomputer verfügt. Programmierkenntnisse und Erfahrungen in der Installation von Software sind nicht erforderlich, da jeder einzelne Schritt zur Realisierung eines Projekts detailliert beschrieben und mit Screenshots visuell unterlegt wird.

Hilfreich sind ein Steckbrett (engl. *Breadboard*, in der Abbildung bereits mit LEDs und Widerständen) und einige Jumperkabel, damit Entwicklungsumgebungen auf der Hardwareseite schnell und einfach aufgebaut werden können. Notwendig ist selbstverständlich auch beispielbezogen das eine oder andere Bauteil (z. B. LEDs, Sensoren etc.).

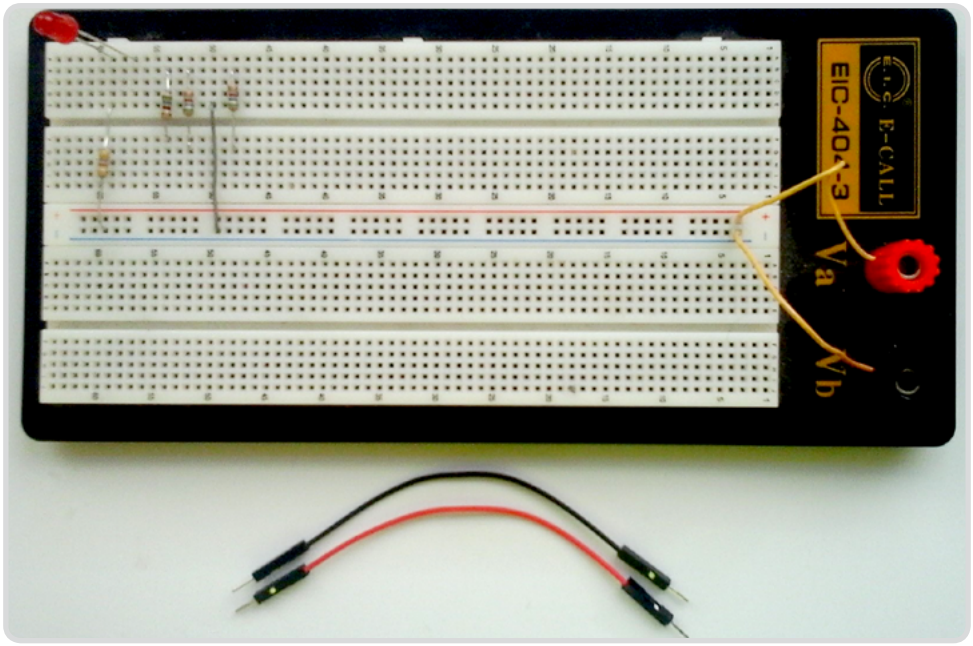

Bild E.1: Steckbrett, Verbindungskabel und diverse Bauteile.

Inhaltliche Beschreibung

Bei der Reihenfolge der Kapitel wurde versucht, von grundlegenden Informationen über die Lösung einfacher Aufgaben zu etwas anspruchsvolleren Techniken zu gelangen. Das erwartet Sie in diesem Buch:

- *Was ist der ESP32?*
  Hier werden die Entstehung und einige Grundgedanken zur Verwendung behandelt.
- *Welche integrierte Entwicklungsumgebung ist geeignet?*
  In diesem Kapitel wird das Programmierwerkzeug für die Projekte vorgestellt.
- *Programmers Focus*
  An dieser Stelle werden ein paar Hilfestellungen für die eigene Programmentwicklung gegeben.
- *Erste Schritte/LEDs ansteuern*
  Das Kapitel befasst sich mit einfachen Sketchen zur Ansteuerung von LEDs sowie dem Auslesen interner Sensoren.
- *LED-Steuerung mit PMW*
  Gegenstand ist die ESP32-Funktionalität PWM und wie sie für eigene Projekte eingesetzt werden kann.
- *Sensoren – analog meets digital*
  Hier werden einige analoge Sensoren und ihre Verwendung vorgestellt.
- *Temperatur- und Luftfeuchte messen*
  Gegenstand ist die Anwendung des DHT22-Sensors.
- *433-MHz-Signale senden und empfangen*
  Dieses Kapitel zeigt, wie 433-MHz-Funkgeräte gesteuert werden können.
- *Der ESP32 bedient I2C*
  Die Abschnitte befassen sich mit der Anbindung eines OLED-Displays über I2C.
- *Der ESP32 kommuniziert über SPI*
  Thema ist, wie Geräte über den SPI-Bus angesteuert werden können.
- *ESP32 und Bluetooth*
  Jetzt kommunizieren wir mit dem ESP32 via Bluetooth.
- *Interrupts*
  Gegenstand ist der Einsatz der nützlichen Programmfunktion Interrupt.
- *Die Blynk-App*
  Dieses Kapitel zeigt, wie der ESP32 über eine App gesteuert werden kann.

- *ESP32 als Webserver*
  Wir setzen einen Webserver auf.
- *Wetterdaten abrufen*
  Wie Wetterdaten von einem Onlinedienst abgerufen und genutzt werden können, wird an dieser Stelle erläutert.
- *E-Mails versenden*
  Thema ist der Versand von E-Mails über ein Gmail-Konto.
- *MQTT by Cayenne*
  Wir nutzen den Cloud-Dienst Cayenne, um über MQTT Daten auszutauschen.
- *ESP in den Sleep-Modus versetzen*
  Thema ist, wie der ESP in den Sleep-Modus gelangt und wieder aufgeweckt wird.
- *Den ESP über das Web updaten*
  Hier wird die webgestützte Aktualisierung von Software vorgestellt.
- *ESP32 und eigener MQTT-Broker*
  Wir setzen den MQTT-Broker Mosquitto auf und tauschen Daten mit dem ESP32 aus.
- *Die Schaltzentrale*
  Abschließend erhalten Sie einige Hinweise dazu, wie eine eigene Schaltzentrale realisiert werden kann.

BESONDERE SCHRIFTAUSZEICHNUNGEN

Verschiedene Objekte werden in einer vom üblichen Text abweichenden Schriftart dargestellt:

- `Codezeilen` und `Codezitate` werden in nicht proportionaler Schrift dargestellt.
- Namen, Dateipfade, Bildschirmzitate etc. werden *kursiv* dargestellt.

INHALT

Unsere Projekte basieren auf dem Mikrocontroller ESP32 und dem ESP32-Development-Kit 4. Es sind starke und mächtige Unterstützer unserer Vorhaben, dennoch haben sie neben positiven Eigenschaften auch ihre empfindlichen Seiten. Diese kennenzulernen ist Gegenstand der folgenden Abschnitte.

ESP32: 32-Bit-Mikrocontroller

Beim ESP32 handelt es sich um einen 32-Bit-Mikrocontroller, der 2016 von der chinesischen Firma Espressif vorgestellt wurde. Er beherbergt einen oder zwei Kerne vom Typ »Xtensa LX6«, die mit einem Systemtakt von 160 MHz bis 240 MHz arbeiten. Das nachstehende Blockdiagramm zeigt die wesentlichen Teile des Systems.

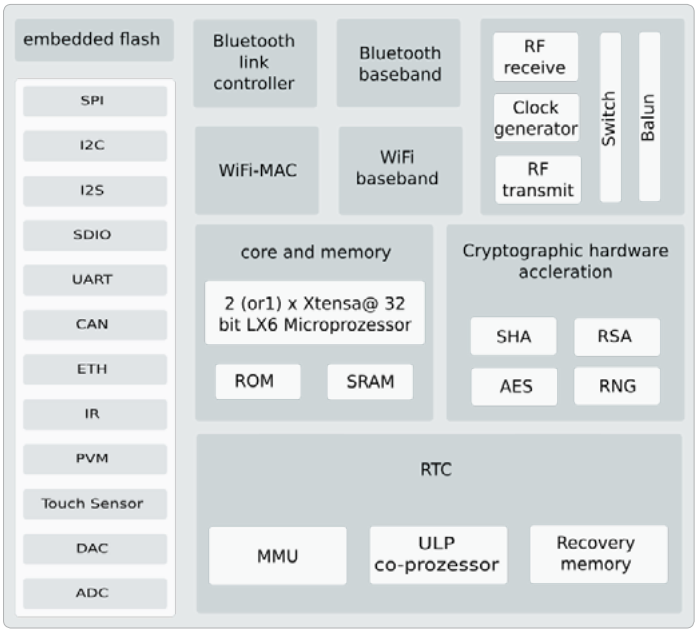

Bild 1.1: ESP32-Blockdiagramm.

Die ESP32-Mikrocontroller weisen einige Eigenschaften auf, die sie besonders für heimische Projekte interessant machen.

- **Interner Speicher**
  Der Chip bietet 520 KByte SRAM und 44 KByte ROM (für das Booten).
- **Externer Speicher**
  Es werden bis zu 4 × 16 MByte an externem Speicher unterstützt.

- **Der Systemtakt**
  Es besteht die Möglichkeit, den internen Takt (8 MHz) oder einen externen Quarztakt mit üblicherweise 160 MHz zu nutzen. Dieser übernimmt auch bei einem Zurücksetzen des Prozessors das System-Timing.
- **Transceiver**
  Für die WLAN- und die Bluetooth-Kommunikation hat der ESP32 interne 2,4-GHz-Sende- und Empfangsbausteine.
- **WLAN**
  Das WLAN arbeitet nach dem Standard IEE 802.11 b/g/n.
- **Bluetooth**
  Unterstützt wird Bluetooth 4.2 (sowohl in der herkömmlichen als auch in der Betriebsart Bluetooth Low Energy).
- **UART**
  UART (*Universal Asynchronous Receiver Transmitter*) bezeichnet eine Form der seriellen Kommunikation. Es stehen drei Bausteine zur Verfügung.
- **Impulszähler**
  Impulse bzw. Impulsflanken können mit acht Impulszählern erfasst werden. Bei Erreichen eines bestimmten Werts kann ein Interrupt ausgelöst werden.
- **Universal-Timer**
  Der ESP32 verfügt über vier 64-Bit-Universal-Timer. Sie können softwareseitig gesteuert werden und Interrupts erzeugen.
- **Watchdog-Timer**
  Es stehen drei Watchdog-Timer zur Verfügung. Dabei wird zwischen sogenannten Main-Watchdog-Timern (davon zwei im Part der Universal-Timer) und RTC-Watchdog-Timern (einer im RTC-Modul) unterschieden. Durch ein Reset des Watchdog-Timers kann ein Interrupt, ein Core-Reset oder ein CPU-Reset ausgelöst werden.
- **Interner Hall-Sensor**
  Ein interner Hall-Sensor kann für die Messung von Magnetfeldschwankungen genutzt werden.
- **Interner Temperatursensor**
  Ein interner Temperatursensor ist für einen Messbereich von 40 bis 125 Grad Celsius ausgelegt. Wie bei dem Hall-Sensor werden die analogen Messwerte von einem internen Analog-Digital-Wandler digitalisiert.
- **GPIOs**
  Es sind 34 GPIOs (universelle Ein- und Ausgänge) vorgesehen. Sie unterstützen die Kommunikation mittels analoger, digitaler und kapazitiver Daten. Die nach außen geführten Pins sind bezüglich ihrer Funktion überwiegend mehrfach belegt. Mit internen Pull-down- und Pull-up-Widerständen können definierte Zustände herbeigeführt werden.

- **Touch-Sensoren**
  Der ESP32 kann Signale von bis zu zehn unterschiedlich kapazitiven Touch-Sensoren verarbeiten. Die Eingänge sind diversen GPIO-Pins zugewiesen.
- **ADC (Analog-Digital-Wandler)**
  Es wird ein 12-Bit-A/D-Wandler mit 18 Kanälen angeboten.
- **DAC (Digital-Analog-Wandler)**
  Es können zwei unabhängige 8-Bit-Digital-Analog-Wandler genutzt werden.
- **SPI-Schnittstelle**
  Der ESP verfügt über vier SPI-Schnittstellen, die im Master- oder Slave-Modus betrieben werden können.
- **I2C-Schnittstelle**
  Es werden zwei I2C-Bus-Schnittstellen vorgehalten, die im Master- oder Slave-Modus betrieben werden können.
- **SPI-Schnittstelle**
  Es werden auch zwei SPI-Schnittstellen bereitgestellt, die ebenfalls im Master- oder Slave-Modus betrieben werden können.
- **PVM (Pulsweitenmodulation)**
  Pulsweitenmodulation dient dazu, Geräte wie Motoren, elektrische Heizungen und Ähnliches zu steuern. Der Chip verfügt über ein programmierbares Hardware-PVM und 16 Software-PVMs. Die Software-PVMs können bis zu 16 unabhängige digitale Wellenformen erzeugen. Tastverhältnis und Perioden sind frei wählbar. Software-PVMs eignen sich besonders zur LED-Ansteuerung.
- **Infrarotcontroller**
  Ein Infrarotcontroller, der bis zu acht Kanäle einer programmierbaren Infrarotfernbedienung bedienen kann, rundet das Angebot an Schnittstellen ab.

Einsetzbare Boards

Der ESP32-Chip wird auf einer Vielzahl von Boards verwendet. Der Hersteller Espressif führt selbst einige Module in seinem Programm.

- **ESP32-Pico-Kit**
  ESP32-Pico-Kit ist das kleinste Entwicklungsboard von Espressif. Es passt in ein Mini-Breadboard, ist aber mit der minimalen Anzahl diskreter Komponenten voll funktionsfähig, weil alle ESP32-Pins zugänglich sind.
- **ESP-Wrover-Kit**
  Das ESP-Wrover-Kit ist das vielseitige Entwicklungsboard von Espressif. Es hat eine umfangreiche Liste von Funktionen an Bord, z. B. LCD-Anzeige, JTAG, Kamera-Header, RGB-LEDs etc. Es können verschiedene Module aufgenommen werden (ESP32-WROVER, ESP-WROOM-32). Mehrere Stiftleisten und Steckbrücken machen die Anbindung und Konfiguration des ESP-Wrover-Kit sehr flexibel.

- **ESP32-DevKitC**
  Das ESP32-DevKitC ist eine Entwicklungsplattform für Einsteiger. Alle ESP32-Pins sind herausgeführt, gut anzuschließen und einfach zu verwenden.
- **ESP32-LyraTD-MSC**
  Das ESP32-LyraTD-MSC ist eines der Audio-Development-Boards von Espressif, das auf die Spracherkennungsanwendungen ausgerichtet ist. Es unterstützt auch Nahfeld- und Fernfeld-Weckfunktionen.

Darüber hinaus wird der ESP32 auch von anderen Herstellern und Distributoren für eigene Boards eingesetzt.

- **LoLin32-ESP32-Development-Board**
  Das LoLin32-ESP32-Development-Board ist in der Regel mit 4 MByte Flash-Speicher und einer Lithium-Batterie ausgestattet.
- **ESP32-OLED-Development-Board**
  Das ESP32-OLED-Development-Board hat bereits ein OLED-Display.
- **SparkFun ESP32 Thing**
  Die Firma SparkFun bietet eine Fülle von Modulen, Sensoren und Platinen für eigene Entwicklungen an. Das von ihr vertriebene ESP32-Board enthält bereits einige zusätzliche LEDs und Tasten.

ESP32-Development-Kit V4

Momentan zählt das ESP32-Development-Kit V4 zu den gängigsten Entwicklungsboards auf Basis des ESP32. Aus diesem Grund soll es auch für die Beispiele in diesem Buch verwendet werden. Daher können einige Kerninformationen zu diesem Modul nützlich sein. Das folgende Bild zeigt die wesentlichen Bauteile des ESP32-Development-Kit V4. Das Board selbst kann auf jedes Breadboard gesteckt werden.

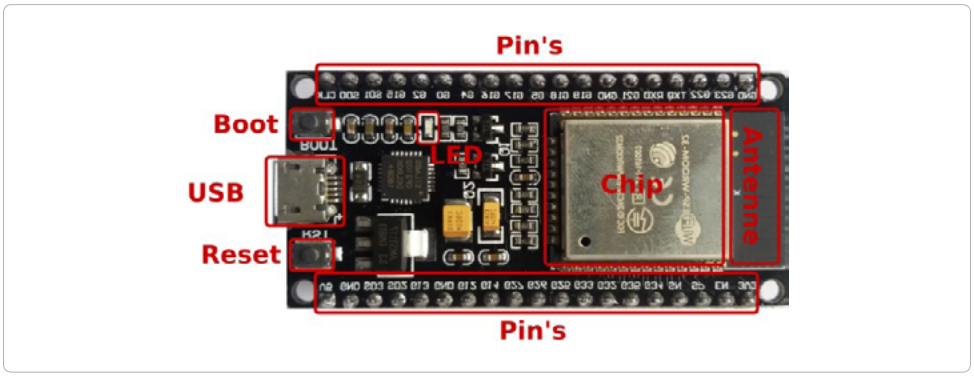

Bild 1.2: Die ESP32-Development-Kit-V4-Baugruppen.

An jeder Seite des Boards befinden sich 19 Anschlüsse für unterschiedliche Zwecke. Die Stromversorgung erfolgt über den USB-Anschluss. Da USB normalerweise mit +5 V arbeitet, wird die Spannung durch einen On-Board-Chip auf +3,3 V reduziert. Eine LED zeigt die Betriebsbereitschaft an.

Die zwei Tasten EN (enable) und BOOT dienen der Steuerung des Moduls.

- EN – Die Reset-Taste.
- BOOT – Die Taste muss für den Upload von Firmware über den USB-Port gedrückt werden.

Das Pin-out des ESP32-Delopment-Kit V4, soweit es in dem Buch von Bedeutung ist, sehen Sie in der nachstehenden Tabelle. Dabei können Pins durchaus mehrere Funktionen haben. Abweichungen zwischen dem Pin-out und der Beschriftung sollten eigentlich nicht sein. Dennoch sind Module im Handel erhältlich, bei denen die gedruckte Information auf dem Board nicht mit der erwarteten Belegung übereinstimmt. In der Regel ist nur der Aufdruck falsch, und die eigentliche Zuordnung ist zutreffend, sodass das Modul einwandfrei funktioniert. In solchen Fällen ist aber etwas mehr Vorsicht geboten.

			3,3 V		GND			
			EN		G23	GPIO23		SPI MOSI
ADC0		GPIO36	SP		G22	GPIO22	SCL	SPI RST
ADC3		GPIO39	SN		TXD	GPIO1		
ADC6		GPIO34	G34		RXD	GPIO3		
ADC7		GPIO35	G35		G21	GPIO21	SDA	SPI SDA
ADC4	Touch9	GPIO32	G32		GND			
ADC5	Touch8	GPIO33	G33		G19	GPIO19		SPI MISO
ADC18		GPIO25	G25		G18	GPIO18		SPI SCK
ADC19		GPIO26	G26		G5	GPIO5		SPI SS
ADC17	Touch7	GPIO27	G27		G17	GPIO17		
ADC16	Touch6	GPIO14	G14		G16	GPIO16		
ADC15	Touch5	GPIO12	G12		G4	GPIO4	Touch0	ADC10
			GND		G0	GPIO0	Touch1	ADC11
ADC14	Touch4	GPIO13	G13		G2	GPIO2	Touch2	ADC12
FLASH D2		GPIO9	SD2		G15	GPIO15	Touch3	ADC13
FLASH D3		GPIO10	SD3		SD1	GPIO8		FLASH D1
FLASHCMD		GPIO11	CMD		SD0	GPIO7		FLASH D0
			VIn		*CLK*	GPIO7		FLASHCLK

Das Modul arbeitet mit einer Spannung von typischerweise +3,3 V. Die GPIO-Pins sind spannungssensitiv. Es kann also zu Beschädigungen oder zu einem Totalausfall des Mikrocontrollers kommen, wenn die an einen Pin angelegte Spannung diesen Wert überschreitet. Manche Sensoren benötigen jedoch für eine einwandfreie Funktion eine Spannung von +5 Volt und geben diese auch an ihren Ausgängen weiter. Das führt gewissermaßen zu einem Dilemma.

Abhilfe können Spannungsteiler oder Pegelwandler schaffen. Im einfachsten Fall reichen hierzu zwei Widerstände.

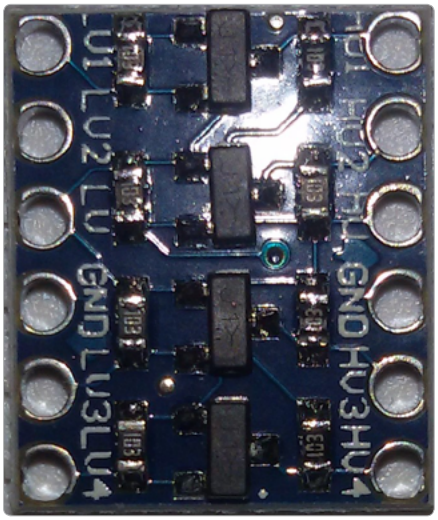

Bild 1.3: Eleganter sind integrierte Pegelwandler (Level Shifter), die auch bidirektional ausgelegt sein können.

Sie sollten also bei der Auswahl der Komponenten (Sensoren) und beim Schaltungsaufbau gesteigerte Sorgfalt walten lassen und lieber einmal mehr als einmal zu wenig prüfen, ob alles korrekt ist.

Die Pins können mit maximal 6 mA belastet werden. Eine höhere Belastung kann ebenfalls zur Zerstörung des Moduls führen. Das gleiche Malheur tritt ein, wenn die +3,3-V-Komponente zu stark belastet wird; für größere Belastungen ist der interne Spannungswandler einfach nicht ausgelegt.

Die erste Inbetriebnahme

Nun können wir das ESP32-Development-Kit V4 über ein Micro-USB-B-Kabel erstmalig mit unserem Computer verbinden. Die Betriebszustands-LED sollte einige Male deutlich rot blinken. Es liegt also Spannung an, und der On-Board-Spannungsregler arbeitet einwandfrei.

Inbetriebnahme unter Windows

Normalerweise sollte das Board von Windows selbstständig erkannt werden. Dies lässt sich einfach im Windows-Geräte-Manager (*Windows/Systemsteuerung/Hardware und Sound/Geräte-Manager*) überprüfen. Hier ist das Modul an Port COM4 angeschlossen. Die USB-zu-UART-Brücke CP210 wurde erkannt und der nötige Treiber installiert.

Bild 1.4: Der Windows-Geräte-Manager.

Inbetriebnahme unter Linux

Auch unter Linux sollte das Board problemlos erkannt werden. Überprüft wird das in einem Terminalfenster mit dem Befehl `lsusb`.

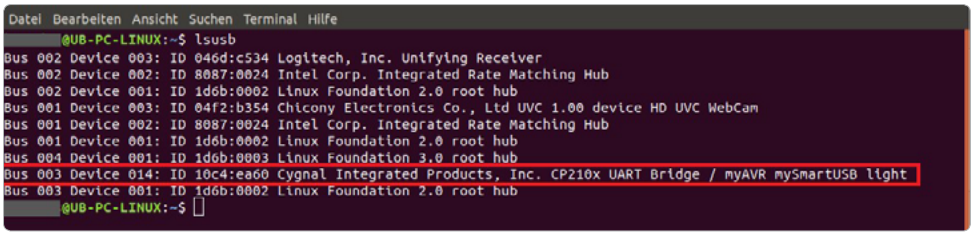

Bild 1.5: Überprüfung im Linux-Terminal mit `lsusb`.

Batterie- und Akkubetrieb

Sollte das ESP32-Development-Kit V4 nicht mit Netzstrom versorgt werden, ist auch ein Batterie- oder Akkubetrieb möglich. Für den Batteriebetrieb eignen sich sehr gut Lithium-Eisenphosphat-Zellen (LiFePo4), da ihre Spannung bereits ohne Regler passt. Ansonsten sollte ein Spannungsregler eingesetzt werden. Bei Betrieb an Akkus müssen Sie bedenken, dass sich der ESP32 bei Unterspannung nicht abschaltet. Daher ist ein zusätzlicher Tiefentladeschutz unbedingt notwendig.

Fazit: Der ESP32 bzw. das ESP32-Development-Kit-V4-Modul präsentiert sich als äußerst vielfältiger Baustein, der geradezu einlädt, ihn für eigene Projekte zu nutzen. Als Nächstes müssen wir uns damit befassen, wie wir den Mikrocontroller programmieren und so für unsere Zwecke einspannen.

2 DIE IDE-ENTWICKLUNGS-UMGEBUNG

Für unsere Programmentwicklung benötigen wir Programmierwerkzeuge, Testmöglichkeiten und Weiteres – kurz: eine integrierte IDE-Entwicklungsumgebung. In diesem Kapitel betrachten wir einige IDEs und richten eine davon für unsere weiteren Arbeiten ein.

Verfügbare Entwicklungsumgebungen

Für die Programmentwicklung kann auf einige Entwicklungsumgebungen zurückgegriffen werden:

- Arduino IDE mit ESP32-Arduino-Bibliotheken.
- ESP Easy – wird von Home-Automation-Begeisterten entwickelt.
- MicroPython.
- Espressif IoT Development Framework – das offizielle Entwicklungs-Framework für den ESP32.
- NoddeMCU LUA-Firmware – bei LUA handelt es sich um eine Skriptsprache.
- Und einiges mehr.

Jede dieser IDEs hat bestimmte Vorzüge, aber auch spezifische Nachteile. Die geläufigste IDE ist vermutlich die Arduino IDE. Sie ist recht gut zu bedienen und erfreut sich einer regen Community, die bei Problemen mit guter Aussicht auf eine weiterführende Antwort kontaktiert werden kann.

Arduino IDE installieren

Die Arduino IDE[1] lässt sich auf allen gängigen Computerplattformen implementieren (Windows, macOS, Linux). Sie ist in Java geschrieben und zunächst für die Programmierung eines Arduino entwickelt worden. Inzwischen ist die IDE so erfolgreich geworden, dass eine Vielzahl weiterer Boards integriert wurde. Sie unterstützt Programme in C bzw. C++ und erfordert eine spezielle Programmstruktur sowie differenzierte Programmierregeln. Doch dazu später.

Auf einem Rechner lassen sich mehrere Arduino IDEs installieren. Das kann hilfreich sein, um IDEs für bestimmte Boards zu separieren oder für gewisse Projekte auf unterschiedliche Programmbibliotheken zurückzugreifen.

1 *https://www.arduino.cc/*

Wir suchen die Arduino-Homepage auf und wechseln in den Download-Bereich.

Bild 2.1: Arduino-IDE-Download.

Nun trennen sich zunächst die Wege.

Linux-Installation

Unter Linux ist die Installation geringfügig aufwendiger. Das gilt leider auch für Ubuntu, da im Paketmanagement bedauerlicherweise keine aktuelle Arduino-Version unterstützt wird. Wir müssen also die entsprechende Linux-Version von der Arduino-Seite für Software herunterladen. Nach Auswahl der Installationsplattform (Linux 32 Bit/Linux 46 Bit/Linux ARM) erscheint eine Zwischenabfrage, die wissen möchte, ob ein Download mit oder ohne Spende gewünscht wird, ehe eine Auswahlbox das Download-Verfahren erfragt.

Bild 2.2: *Arduino-1.8.8-linux64.tar.xz* öffnen.

Mit *Datei speichern* wird eine Datei mit den gepackten Programmbestandteilen im Download-Ordner abgespeichert (z. B. *arduino-1.8.8-linux64.tar.xz*).

Bild 2.3: Diese Datei muss an einem Ziel Ihrer Wahl extrahiert werden – hier ist es der Download-Ordner.

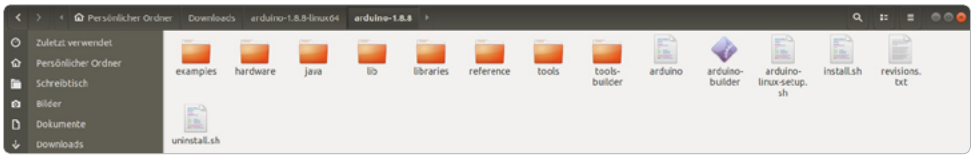

Bild 2.4: Das extrahierte Arduino-IDE-Linux-Paket.

Wir öffnen ein Terminal, wechseln in diesen Ordner und führen das Shell-Skript *install.sh* aus.

Bild 2.5: Der Arduino-IDE-Linux-`install`-Befehl.

Damit wird die Arduino IDE eingerichtet. Im Anwendungsfenster befindet sich nun ein Icon, mit dem Arduino aufgerufen werden kann. Darüber hinaus wird im Home-Verzeichnis (*Persönlicher Ordner*) des Anwenders ein neuer Ordner *Arduino* eingerichtet.

> **HINWEIS**
>
> Sollte Arduino nicht mehr erwünscht sein, können Sie das *uninstall.sh*-Skript verwenden. Verbliebene Arduino-Dateien lassen sich auch mit dem Datei-Manager löschen. Das Icon im Anwendungsfenster lässt sich durch Löschen der Datei */usrshare/applications/arduino-arduinoide.desktop* entfernen.

Windows-Installation

Bei Windows stehen Ihnen zwei Installationswege zur Verfügung. In beiden Fällen wird wie bei Linux die Spendenbereitschaft erfragt. Bei dem Weg über den Windows-Installer müssen entweder Administrationsrechte für den aktuellen Benutzer vorliegen, oder es muss zumindest die PIN für ein Administrationskonto bekannt sein. Die Installationssoftware wird in den Download-Ordner des Browsers heruntergeladen und ausgeführt. Der Installer richtet dann die erforderliche Umgebung selbstständig ein.

Die zweite Methode besteht darin, eine ZIP-Datei in den Download-Ordner herunterzuladen (z. B. *arduino-1.8.8-windows.zip*). Nach dem Entpacken sehen Sie in dem entpackten Ordner einen Unterordner (z. B. *arduino-1.8.8*), der die gesamte Arduino IDE enthält. Entweder wird dieser Ordner an der Stelle belassen, oder Sie kopieren ihn an einen Ort Ihrer Wahl, z. B. in das eigene *Home*-Verzeichnis.

In dem vorgenannten Unterordner (z. B. *arduino-1.8.8*) befindet sich die Datei *arduino.exe* für den Start der Entwicklungsumgebung. Nach dem Anlegen einer Desktopverknüpfung (Linksklick mit der Maus auf *Neu/Verknüpfung*) kann die IDE auf die bekannt einfache Weise aufgerufen werden.

Arduino IDE für ESP32 einrichten

Da Arduino vornehmlich auf Arduino-Boards ausgerichtet ist, müssen die erforderlichen Boardinformationen bzw. -bibliotheken für den ESP32 nachgeladen werden. Das erfolgt in wenigen Schritten.

1. Zunächst muss der Arduino IDE mitgeteilt werden, wo die benötigten Dateien zu finden sind. Hierzu öffnen Sie über *Start/Voreinstellungen* den Eingabebereich. Weiter unten in der Eingabemaske sehen Sie das Feld *Zusätzliche Boardverwalter-URLs*, die benötigten Boardverwalter finden sich unter *https://dl.espressif.com/dl/package_esp32_index.json*.

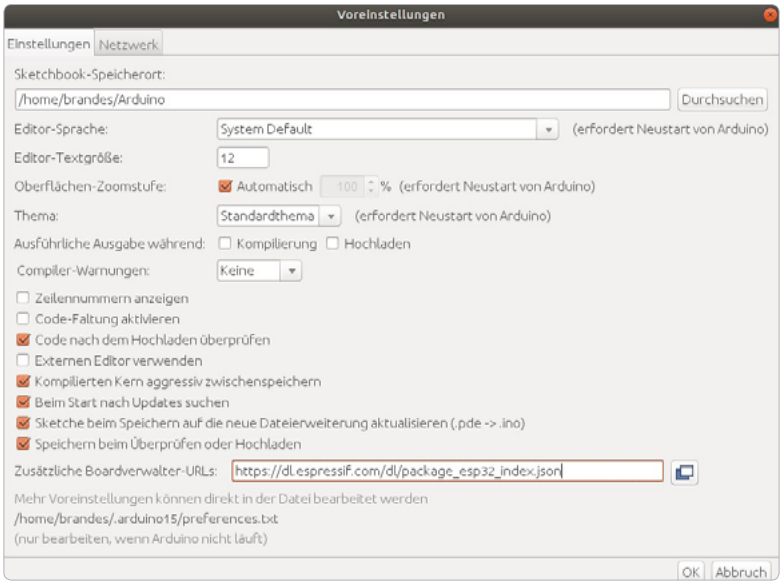

Bild 2.6: Arduino IDE einrichten – *Voreinstellungen*.

Diese Adresse tragen Sie dort ein. Falls bereits Bordverwalter vorhanden sind und noch weitere hinzugefügt werden sollen, werden die Bordverwalter-URLs durch Kommata getrennt. Mit Klick auf das Fenstersymbol rechts in der Zeile öffnet sich ein Editor, in dem die Eingaben auf benutzerfreundliche Weise getätigt werden können. Mit *OK* werden die Änderungen gespeichert und wirksam.

❷ Mit dem Wissen darüber, wo die Bibliotheken zu finden sind, kann die Arduino IDE nun das Benötigte nachladen. Hierzu wechseln Sie über *Werkzeuge/Board/Boardverwalter* in die entsprechende Eingabemaske.

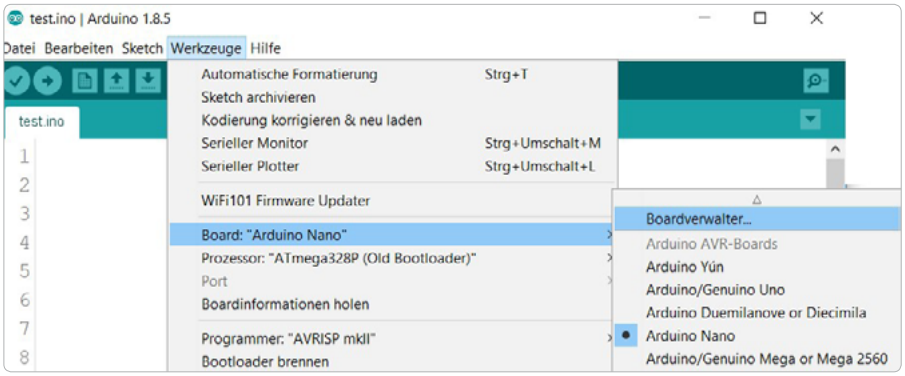

Bild 2.7: Arduino IDE einrichten und *Boardverwalter* aufrufen.

Es erscheint eine Liste aller derzeit für die geöffnete IDE erreichbaren Boardbibliotheken. Eine einfache Möglichkeit, den zutreffenden Eintrag zu finden, ist über die Eingabe von `ESP32` in das Suchfeld.

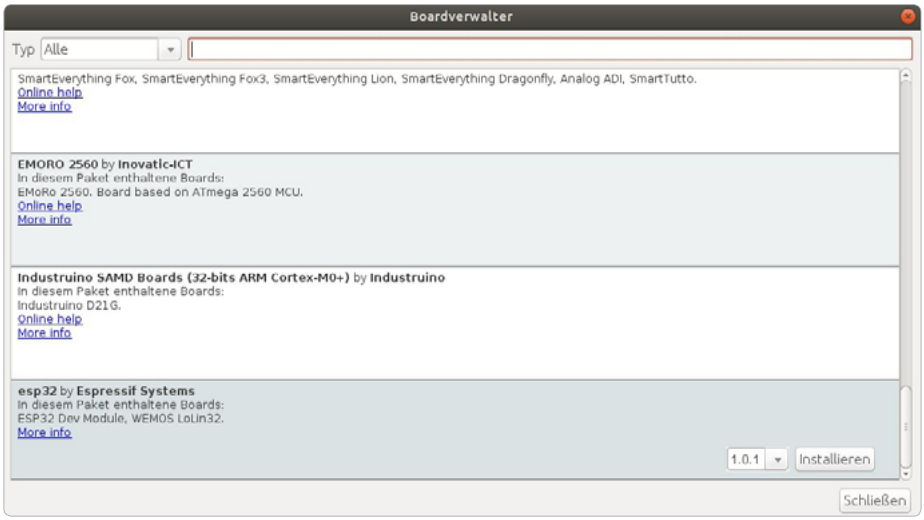

Bild 2.8: Arduino IDE einrichten und ESP32-Boardbibliotheken installieren.

3. Durch Klick auf den Button *Installieren* werden die Boardverwalter-Informationen zum ESP32 installiert. Bei Erfolg erscheint die installierte Versionsnummer neben einem Vermerk *INSTALLED* in der Überschriftszeile des Tabelleneintrags zu dem Board.

> **HINWEIS**
>
> Per Link *More info* sind weitere hilfreiche Informationen zu dem installierten Boardverwalter abrufbar. Darüber hinaus lassen sich über die Drop-down-Auswahl andere Versionen des Boardverwalters installieren. Das kann unter Umständen dann hilfreich sein, wenn nach Updates Sketche nicht mehr fehlerfrei laufen und auf eine ältere Version des Boardverwalters zurückgegriffen werden muss.

4. Ein kurzer Check zeigt, dass nun eine Reihe neuer Boards zur Auswahl zur Verfügung steht. Die Arduino IDE ist damit für unsere Zwecke grundsätzlich betriebsbereit.

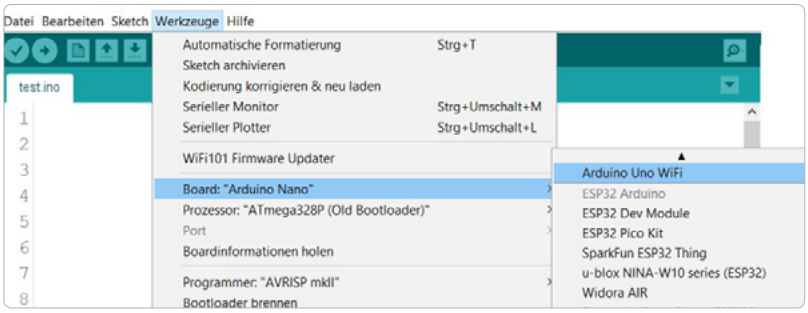

Bild 2.9: Für die Arduino IDE eingerichtete ESP32-Boards.

Wichtige Voreinstellungen

Auch wenn wir über eine funktionierende Entwicklungsumgebung verfügen, können einige Voreinstellungen im Hinblick auf eine verbesserte Entwicklungsumgebung interessant sein (*Start/Voreinstellungen*). Der Sketchbook-Speicherort enthält einen Dateipfad, unter dem nicht nur die eigenen Sketche gespeichert sind, sondern auch weitere Bibliotheken, die unter Umständen im Zuge der Projekte nachgeladen werden.

Ausführliche Ausgaben sind bei der Arduino IDE üblicherweise nicht vorgesehen. Für die Fehlersuche beim Kompilieren oder beim Hochladen kann ein Mehr an Informationen aber durchaus hilfreich sein. Das kann mit Aktivieren der entsprechenden Kontrollkästchen erreicht werden. Auch ist es gut, über Zeilennummern Codezeilen in größeren Programmen schneller zu lokalisieren – aktivieren Sie dazu das Kontrollkästchen *Zeilennummern anzeigen*.

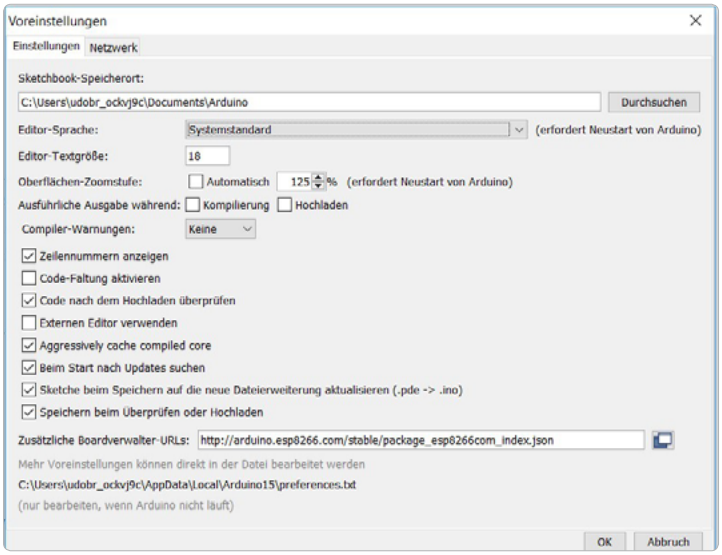

Bild 2.10: Dateipfad vom *Sketchbook-Speicherort*.

Beim Start nach Updates suchen verzögert zwar das Laden der IDE; aber so werden keine Neuerungen in den installierten Bibliotheken verpasst.

Arbeiten mit der Arduino IDE

Beim erstmaligen Start der Arduino IDE wird die Oberfläche mit einem leeren Sketch angezeigt.

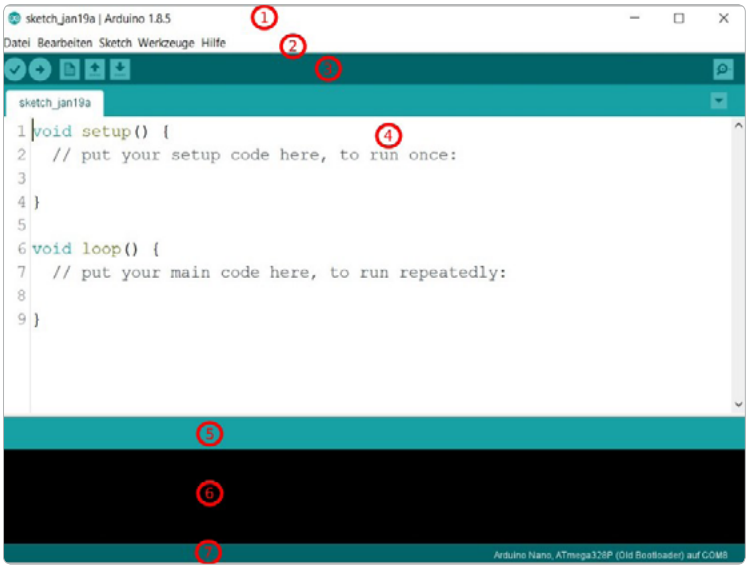

Bild 2.11: Die Arduino IDE: Titelleiste ❶, Menüleiste ❷, Speichern und Öffnen ❸, Programmier-bereich ❹, Meldungen ❺, Terminal ❻, Info ❼.

Die Titelleiste ❶ gibt Auskunft über den aktuell geladenen Sketch und die Version der Arduino IDE. Über die Menüleiste ❷ lassen sich alle Funktionen der IDE erreichen.

- In der Drop-down-Liste *Datei* finden sich die gewohnten dateibezogenen Operationen (*Neu*, *Öffnen*, *Speichern*) sowie der Pfad zu der Maske für die Voreinstellungen.
- Die Liste *Bearbeiten* beherbergt alle für die Editierung des Sourcecodes notwendigen Befehle. Die Befehle selbst lassen sich aber einfacher durch den Gebrauch von Tastaturkürzeln (Shortcuts) aufrufen.
- Bei der Zuordnung der Tasten zu den Shortcuts haben sich die IDE-Entwickler weitgehend an den gebräuchlichen Konventionen orientiert (z. B. Strg + C für Kopieren). Das gilt in ähnlicher Weise für den Drop-down-Bereich *Sketch*. Interessant sind hier darüber hinaus die Möglichkeiten *Kompilierte Binärdatei exportieren* und *Bibliothek einbinden,* doch dazu später mehr.
- Die Liste *Werkzeuge* enthält einige nützliche und sogar notwendige Einträge. An dieser Stelle soll allerdings auf eine Erörterung der Möglichkeiten verzichtet werden, da das nur in Form einer theoretischen Abhandlung möglich wäre. Wesentlich aussagekräftiger ist es, dies anhand eines praktischen Beispiels zu erkunden.
- Der Eintrag *Hilfe* bietet unter anderem Links zu Internetseiten, die bei Problemen Unterstützung bieten können.

In der nächsten Leiste ❸ befinden sich einige Icons, deren Funktionen den flüssigen Gebrauch der IDE sehr unterstützen (Sketch überprüfen, Sketch hochladen, neuer Sketch, Sketch öffnen, Sketch speichern, seriellen Monitor öffnen).

Der folgende Bereich ❹ ist der Programmierbereich. Sind mehrere Sketche in Bearbeitung, wird für jeden Sketch ein Register angelegt.

In der darunterliegenden Zeile ❺ werden Meldungen der Arduino IDE zu Bearbeitungsvorgängen des aktuellen Sketchs ausgegeben (z. B. *Speichern abgeschlossen*).

Kompilations- und Hochladevorgänge lassen sich im Terminalbereich ❻ verfolgen. Darüber hinaus werden wertvolle Hinweise zu der Programmgröße und den belegten Speicher ausgegeben. Auch finden sich hier im Fehlerfall (z. B. Übersetzungsfehler, Fehler beim Flashen) weiterführende Meldungen.

Die Fußzeile ❼ gibt Auskunft über das gewählte Board und über welchen Port es angeschlossen ist. Bei Windows beginnt der Portname in der Regel mit *COM*, bei Linux mit *ttyUSB*, gefolgt von einer Nummer. Das ist nicht ohne Bedeutung; darauf kommen wir später noch zurück.

Bibliotheken administrieren

Bibliotheken mit Programmbestandteilen sind in der Programmierung mittlerweile unverzichtbar, weil es schier unmöglich ist, für jedes einzelne Programm den gesamten Code neu zu erarbeiten. Viel einfacher und schneller ist es, auf vorhandene Bausteine zurückzugreifen. Diese Bausteine werden in Bibliotheken (Libraries) zusammengefasst. Auch die Arduino IDE bringt von Haus aus einige Bibliotheken mit, in denen themenbezogene Programmbausteine zur Verfügung gestellt werden. Dies zeigt ein kurzer Blick in das Menü *Sketch/Bibliothek einbinden* (Listeneinträge *Bridge*, *EEPROM* etc.).

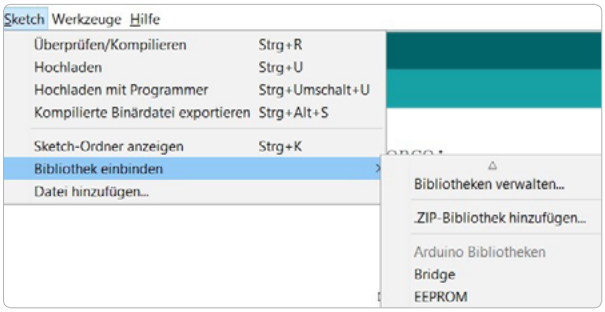

Bild 2.12: Arduino-IDE-Bibliothek einbinden.

Bedauerlicherweise sind nicht für jedes erdenkliche Projekt alle erforderlichen Bibliotheken bereits installiert. Hinzu kommt, dass die Arduino IDE für die Anwendung im Zusammenhang mit Arduino-Hardware (also ATmega-Controllern) entwickelt wurde. Mikrocontroller anderer Hersteller benötigen aber aufgrund ihrer unterschiedlichen Hardwarearchitektur möglicherweise spezielle Bibliotheken. In beiden Fällen müssen diese Bibliotheken nachträglich in die Arduino IDE eingebaut werden.

Auch für unsere Projekte wird das erforderlich sein, sodass die Schritte einer solchen Nachinstallation beispielhaft beschrieben werden sollen. Grundlage soll die Bibliothek *rc-switch* sein, die für das Handling von 433-MHz-Sendern und -Empfängern hilfreich ist. Die Arduino IDE kennt für das Einbinden von Bibliotheken zwei Möglichkeiten.

Einbinden mit dem Bibliotheksverwalter

Die etwas einfachere Möglichkeit nutzt den Bibliotheksverwalter, der über *Sketch/Bibliothek einbinden/Bibliotheken verwalten* aufgerufen wird.

Der *Bibliotheksverwalter* zeigt sowohl installierte als auch mit seiner Hilfe zu installierende Bibliotheken an. Die Liste ist äußerst umfangreich. Um sie nicht vollständig durchblättern zu müssen, kann die Darstellung auf bestimmte Typen und Themen begrenzt werden. Oder das gesuchte Objekt wird in das Suchfeld eingegeben.

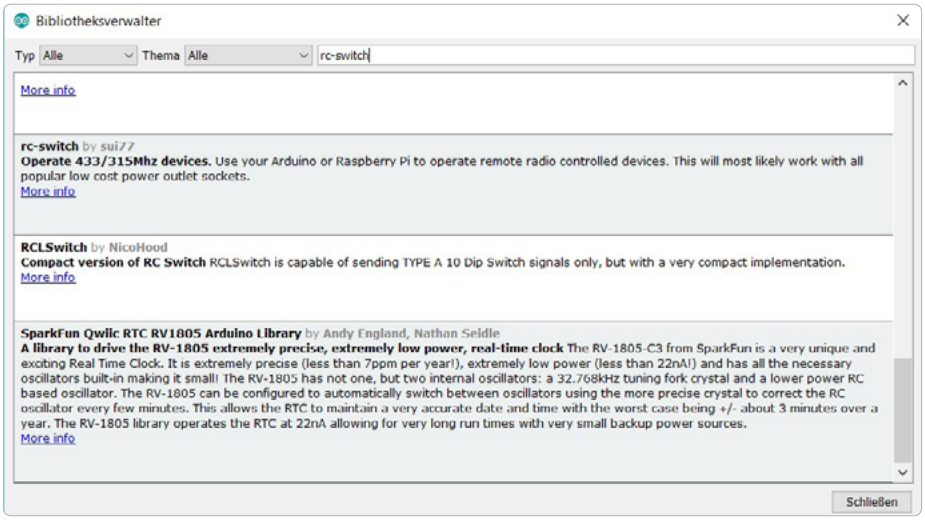

Bild 2.13: Der Arduino-IDE-*Bibliotheksverwalter*.

Wir haben hier nach *rc-switch* gesucht und trotzdem einige Treffer erhalten. Die erforderliche Bibliothek befindet sich am Listenende. Klicken Sie mit dem Mauszeiger auf den Listeneintrag.

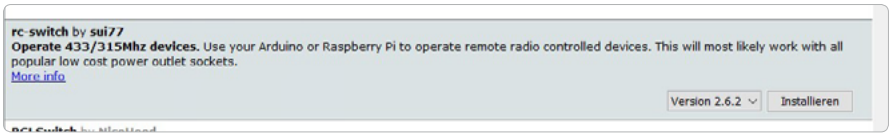

Bild 2.14: Listeneintrag im *Bibliotheksverwalter*.

Es werden nun die Drop-down-Liste *Version* (hier kann eine Versionsauswahl vorgenommen werden, sofern mehrere existieren) und ein Button *Installieren* angezeigt. Ein Klick auf diesen Button installiert die Bibliothek. Installationsziel ist der Unterordner *libraries* des Sketchbooks. Installierte Bibliotheken sind mit *Installed* gekennzeichnet. Über *More info* kann dann die Bibliothek administriert werden.

Bibliothek als ZIP-Datei einbinden

Manchmal liegen Bibliotheken auch nur als ZIP-Datei vor. Sie lassen sich dann über *Sketch/Bibliothek einbinden/ZIP-Bibliothek hinzufügen* einbinden. Aufgerufen wird ein Datei-Explorer. Durch Klick auf den *Öffnen*-Button wird die Bibliothek in den oben genannten Unterordner für Bibliotheken übertragen.

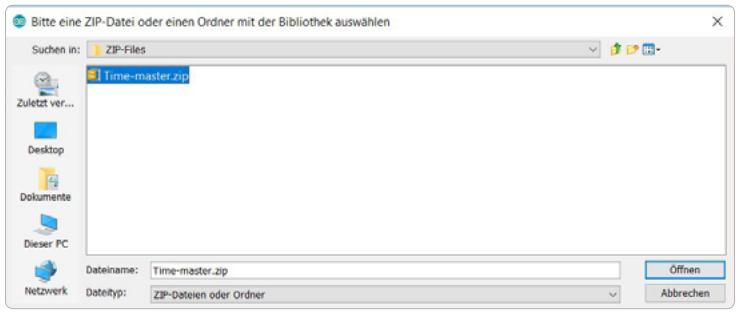

Bild 2.15: Arduino IDE – ZIP-Datei einbinden.

Bibliothek mit dem Datei-Manager einfügen

Genauso gut lassen sich die Bibliotheken auch mit dem Datei-Manager einfügen.

1. Laden Sie die Bibliothek und entpacken Sie sie. Die Bibliothek enthält einen Hauptordner, in dem zumindest eine CPP-Datei, eine H-Datei und oftmals eine TXT-Datei enthalten sind.
2. Schließen Sie das Arduino-Programm.
3. Öffnen Sie im Sketchbook den Ordner *libraries* und kopieren Sie den kompletten Hauptordner der entpackten Bibliothek hinein.
4. Anschließend öffnen Sie wieder das Arduino-Programm und prüfen unter *Sketch/Bibliothek einbinden*, ob die hinzugefügte Bibliothek dort auftaucht.

Das Werkzeug für die Entwicklung eigener Projekte ist jetzt in den wesentlichen Grundzügen eingerichtet und bekannt. Das folgende Kapitel befasst sich mit der grundlegenden Programmierung. Ungeduldige können es durchaus überspringen, sofort loslegen und zu einem späteren Zeitpunkt oder bei Bedarf darauf zurückkommen.

Die Arduino IDE und ihre Eigenheiten

Eine der augenfälligsten Eigenheiten der Arduino IDE ist, dass Programme Sketche heißen. Diese haben die Dateiendung *.ino* und werden beim Speichern in einem Ordner abgelegt, der den gleichen Namen trägt wie die Programmdatei.

Ebenfalls in diesem Ordner werden weitere individuelle Sketche (Untersketche) und C- bzw. C++-Dateien mit ihren zugehörigen H-Dateien abgelegt, die Funktionen enthalten, die vom Hauptsketch benötigt werden. In der IDE werden dann alle Dateien zur Bearbeitung angeboten.

Darüber hinaus stellt die Arduino IDE Sprachelemente zur Verfügung, denen eine direkte Entsprechung in C oder C++ fehlt (z. B. Variablentyp `String`). Wer also sein Programm portierbar halten möchte, muss spätestens dann andere Lösungen suchen.

Die Überschrift mag vielleicht etwas überdimensioniert anmuten, sie soll aber sanft darauf hinweisen, dass die Programmierung, wenn sie erfolgreich sein und Spaß machen soll, ab einem gewissen Punkt – und der ist recht bald erreicht – mehr verlangt als ein Kopieren und Einfügen von Codeschnipseln. In den nachfolgenden Abschnitten sollen Hinweise zu dem Wie, Was und Warum von Befehlen, Konventionen und der Programmerstellung schlechthin gegeben werden. Das Thema an sich könnte eigene, mächtige Bücher füllen, sodass sich hier auf das Nötigste beschränkt werden muss. Dabei ist die Reihenfolge rein willkürlich gewählt, nicht nach Wichtigkeit sortiert und beileibe nicht ansatzweise vollständig.

Programmieren in C und C++

Im Hinblick auf eine verbesserte Lesbarkeit sollte der Code hinreichend strukturiert sein. Das bedeutet, dass Codeblöcke, wie sie z. B. durch Abfragen oder Schleifen entstehen, eingerückt werden.

Ordentliche C-Codestruktur

Ebenso wichtig ist es, logisch zusammenhängende Codeabschnitte in eigene Funktionen auszulagern. Im Eifer der Programmentwicklung oder durch nachträgliche Änderungen besteht die latente Gefahr, Spaghetti-Code zu generieren: Dabei werden dann alle Ideen in eine Funktion gepackt, und das Ganze wird irgendwie zum Laufen gebracht. Treten Fehler auf oder soll nach einer Weile Neues hinzugefügt werden, fällt es schwer, die richtige Stelle für die Änderung zu finden.

Besser ist es, Aufgaben in überschaubare Teilaufgaben zu gliedern und ihnen eigene Funktionen zu spendieren. Vorteil dieser Herangehensweise ist, dass derart aufgeteilte Codeblöcke auch von anderen Programmteilen genutzt werden können.

Die Arduino IDE erlaubt es – anders als eine reine C- oder C++-IDE –, Funktionen in ihrer Reihenfolge relativ unsortiert zu deklarieren. Dennoch hat es sich etabliert, die Funktion an das Ende des Codes zu setzen.

Variablen- und Funktionsnamen

Bei der Wahl von Variablen- und Funktionsnamen hat der Entwickler weitgehend freie Hand. Hilfreich ist es jedoch, die Bezeichnungen selbsterklärend zu gestalten (z. B. nicht `p`, sondern `ledPin`). Außerdem hat sich gewissermaßen als nicht offiziell festgelegte Sprachkonvention eingebürgert, zusammengesetzte Namen im Programmteil kleinzuschreiben und das zweite und jedes weitere Wort mit einem Großbuchstaben zu beginnen (z. B. `ledPinAusgabe`). Angaben in `#defines`-Präcompilerdirektiven sind in Großbuchstaben gehalten (z. B. `LED_PIN`).

Gültigkeitsbereich von Variablen

Der Gültigkeitsbereich und die Lebensdauer von Variablen sind häufig übersehene Stolpersteine bei der Programmierung. In Abhängigkeit davon, an welcher Stelle eine Variable in einem Programm deklariert wurde, ergeben sich andere Geltungsbereiche:

- `Block` – Wird eine Variable in einem Anweisungsblock (`{}`, z. B. `if`-Block) deklariert, sind ihr Geltungsbereich und die Lebensdauer auf diesen Anweisungsblock beschränkt. Wird in diesem Anweisungsblock eine Variable dieses Namens benutzt, wird der Inhalt der im Anweisungsblock definierten Variablen verwendet, auch wenn es außerhalb dieses Blocks Variablen gleichen Namens gibt.
- `Funktion` – Wird eine Variable in einer Funktion deklariert, reichen Geltungsbereich und Lebensdauer von Funktionsblockanfang bis Funktionsblockende. Eine Ausnahme besteht dann, wenn in einem Block innerhalb der Funktion eine Variable gleichen Namens deklariert wird; innerhalb des Blocks gilt dann die entsprechende Blockvariable.
- `Datei` – Eine außerhalb von Funktionen und Anweisungsblöcken deklarierte Variable reicht für das gesamte Programm mit der Einschränkung, dass Block- oder Funktionsvariablen vorgehen.

Prägnante Kommentare

Programme können – nein, müssen – mit Kommentaren versehen werden. Das ist zugegebenermaßen überaus lästig. Wertvoll ist eine umfangreiche und aussagekräftige Kommentierung aber immer, wenn nach längerer Zeit wieder an einem Programmteil gearbeitet werden soll. Dabei gilt: lieber mehr als zu wenig. Ohne prägnante Kommentare gestaltet sich die neuerliche Einarbeitung in ein bestehendes Programm zeitraubend, frustrierend und fehleranfällig.

Kommentare werden auf zweierlei Weise eingeleitet:

- Ein doppelter Schrägstrich `//`
  Alles, was in der Zeile nach den Kommentarzeichen steht, wird vom Compiler als Kommentar betrachtet.
- Ein Schrägstrich gefolgt von einem Stern `/*`
  Hiermit wird ein Kommentarbereich begonnen, der mit der umgekehrten Zeichenfolge `*/` endet. Der Bereich kann über mehrere Programmteile gehen, die mit einem Stern `*` beginnen.

Kommentare werden in der Arduino IDE in grauen Schriftzeichen dargestellt.

JavaScript Object Notation

Die *JavaScript Object Notation* (JSON) dient als kompaktes Datenformat in einer gut lesbaren Textform für den Datenaustausch zwischen einem oder mehreren Systemen. Gerade Client-Server-Architekturen sind ein weitverbreitetes Anwendungsgebiet. JSON ist unabhängig von Programmiersprachen. Dennoch liegen JSON eine eigene Struktur und eigene Konventionen zugrunde.

Als Struktur werden Name-Wert-Paare (z. B. `"farbe": "blau"`) oder eine geordnete Liste von Werten (z. B. `"farben": ["blau", "rot", "gelb"]`) verwendet.

Der Wert, der den Eigenschaften zugewiesen wird, kann jeden gültigen JavaScript[2]-Datentyp annehmen. Dies sind:

- Boolean – `true`, `false`
- Zahl – `0`, `99.14`
- Zeichen – `"Hallo"`
- Tabelle – `[1, 2, 3, 4]`
- Objekt – `{"a": 1, "b": 2}`
- null – `0`

Dabei kann ein Tabellenelement auch durch ein Objekt repräsentiert werden. Die Notation von JSON weicht in einigen Punkten aber von der JavaScript-Syntax ab:

- In einem Objekt müssen alle Eigenschaftsnamen in doppelte Anführungszeichen gesetzt sein.
- Führende Kommata in Objekten und Arrays sind nicht erlaubt.
- Bei Zahlen sind führende Nullen verboten; einem Dezimalpunkt muss zumindest eine Ziffer folgen.
- Zeichen(-Ketten) müssen durch doppelte Anführungszeichen begrenzt sein.

Beispielhaft lässt sich das an einem Objekt `Smartphone` beschreiben. Jedes Objekt `Smartphone` lässt sich durch Eigenschaften wie Hersteller, Modell und Baujahr beschreiben. Es hat auch eine Vielzahl von Apps, die sich in einer Tabelle organisieren lassen. Jede dieser Apps hat wiederum Eigenschaften wie aktiv oder belegter Speicher.

2 *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures*

Bild 3.1: Mit dem Objekt *Smartphone* lassen sich Hersteller, Modell, Baujahr und die auf einem Smartphone installierten Apps und deren Eigenschaften beschreiben.

Das folgende Bild zeigt einen JSON-String, der diese Angaben aufnimmt. Zur besseren Darstellung wurde ein JSON-Formatter verwendet – *https://jsonformatter.org/*. Er erlaubt es, gleichzeitig selbst erstellte JSON-Strings auf Gültigkeit zu überprüfen, sie in andere Formate (z. B. XML, CSV) zu konvertieren und sie auf den lokalen Desktopcomputer herunterzuladen. In der folgenden Ansicht lässt sich die Struktur aller Eigenschaften des Objekts Smartphone bestens ablesen. Gut zu erkennen sind die geschweiften Klammern, die einzelne Objekte einschließen, und die eckigen Klammern, die die Tabelle begrenzen.

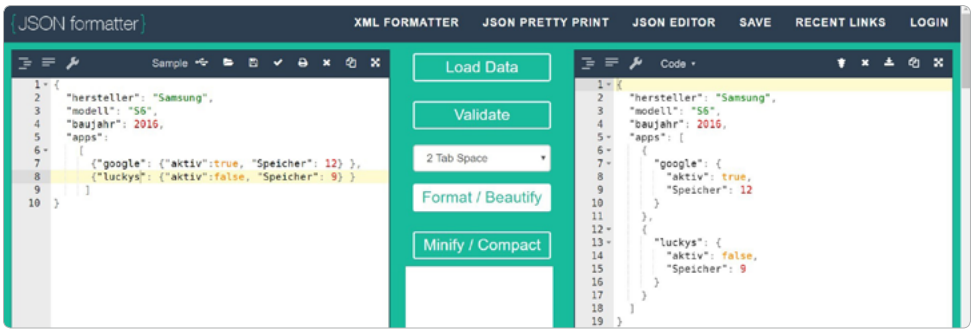

Bild 3.2: Der JSON-Formatter in Aktion.

Erste Hilfe im Problemfall

Bei Selbstbauprojekten wird es unweigerlich dazu kommen, dass Dinge gar nicht oder nicht so funktionieren, wie es erwartet wird. Dann hilft zunächst:

- Über den chinesischen Hersteller schimpfen.
- Über den Autor herziehen.
- Über sich selbst ärgern.

Danach kann man die Fehlersuche systematisch angehen.

- **Fehler in der Hardware**
  Sind alle Kontakte zuverlässig und richtig verbunden? Arbeitet jedes Bauteil einwandfrei?
- **Fehler in der Software**
  Ist der Gültigkeitsbereich von Variablen beachtet worden? Ist der Operand bei der `if`-Abfrage richtig gesetzt, insbesondere `==` bei Abfragen auf Gleichheit? Werden Schleifenzähler und -endekriterien zutreffend ausgewertet? Wird der Tabellenraum bei Arbeiten mit einer Tabelle eingehalten? Ist die Pinbelegung richtig und den Konventionen entsprechend?

Es gibt aber Fehlerquellen, die trotz intensiver Suche im Verborgenen liegen:

- **Unzureichende Stromversorgung**
  Der ESP32 legt Wert auf eine störungsfreie Stromversorgung. Manches Billignetzteil leistet das nicht. Auch kann eine zu kurze oder mangelhafte Kabelverbindung zwischen Stromquelle und ESP32 die Funktionalität beeinträchtigen.
- **Selbstverwaltungsprozesse des ESP32**
  Zuweilen benötigt der ESP32 Zeit für sogenannte interne Selbstverwaltungsprozesse, die sich, anders als das Warten auf das Ergebnis von Anfragen z. B. bei einer Wi-Fi-Verbindung, nicht von selbst erschließen.

Darüber hinaus kann ein Blick in das Datenblatt des ESP32 bei der Fehlersuche helfen. Das gibt es zwar nur in Englisch, im Allgemeinen lässt es sich dennoch recht gut lesen und verstehen.

Ein probater Weg zur Hilfestellung ist eine Anfrage in den einschlägigen Foren. Sowohl bei Espressif unter *https://bbs.espressif.com/viewforum.php?f=64* als auch bei Arduino unter *https://forum.arduino.cc/index.php?board=31.0* gibt es aufmerksame und erfahrene Nutzer, die gern weiterhelfen.

Numerisch

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

R

S

T

U

V

W

X

Z

MACH'S EINFACH

Erste Schritte:

EIGENE IoT- LÖSUNGEN MIT DEM ESP32

Mikrocontroller, Internet und PC

Das Internet wird zunehmend als Basis genutzt, um mit entsprechenden Technologien physische und virtuelle Gegenstände miteinander zu vernetzen und sie zusammenarbeiten zu lassen. Dafür wurde der Begriff „Allesnetz“ oder „Internet of Things“ – kurz „IoT“ oder „Internet der Dinge“ – geprägt.

Das Buch richtet sich an alle, die sich mit den Möglichkeiten, die Mikrocontroller im Zusammenspiel mit Internet und PC bieten, vertraut machen wollen. Es bietet praktische Anleitungen und Lösungen zur Automatisierung von Abläufen sowie vertiefende Anregungen für eigene Entwicklungen, mit denen Sie sich unabhängig von industriellen Lösungen machen können.

In diesem Sinne ist das Buch Wegbereiter und Ideengeber für eigene, gelingende Smart-Home-Lösungen unter dem Dach des Internet of Things, sei es die Ansteuerung von LEDs, das Auslesen interner Sensoren oder das Messen von Temperatur und Luftfeuchte. Erfahren Sie außerdem, wie Sie ein OLED-Display anbinden, Geräte über den SPI-Bus ansteuern oder via Bluetooth mit dem ESP32 kommunizieren.

THEMENGEBIETE:

- Der Mikrocontroller ESP32
- Das ESP32-Development-Kit V4
- Entwicklungsumgebung und IDEs
- Arduino IDE für ESP32 einrichten
- Mit der Arduino-IDE arbeiten
- Die Programmiersprachen C / C++
- JavaScript Object Notation
- Erste Schritte: LEDs ansteuern
- Der interne Temperatursensor
- Helligkeit und Farbe mit PEN steuern
- Heartbeat mit ESP32-LED-PWM
- Sensoren – analog meets digital
- Der Bodenfeuchte-Sensor YL-69
- Temperatur und Luftfeuchte messen
- 433MHz-Geräte schalten
- Der ESP32 bedient I2C
- Der ESP32 kommuniziert über SPI
- Der ESP32 und Bluetooth
- 433MHz-Funksteckdosen über Blynk steuern

ISBN 978-3-645-20661-7

FRANZIS

Besuchen Sie unsere Webseite:
www.franzis.de