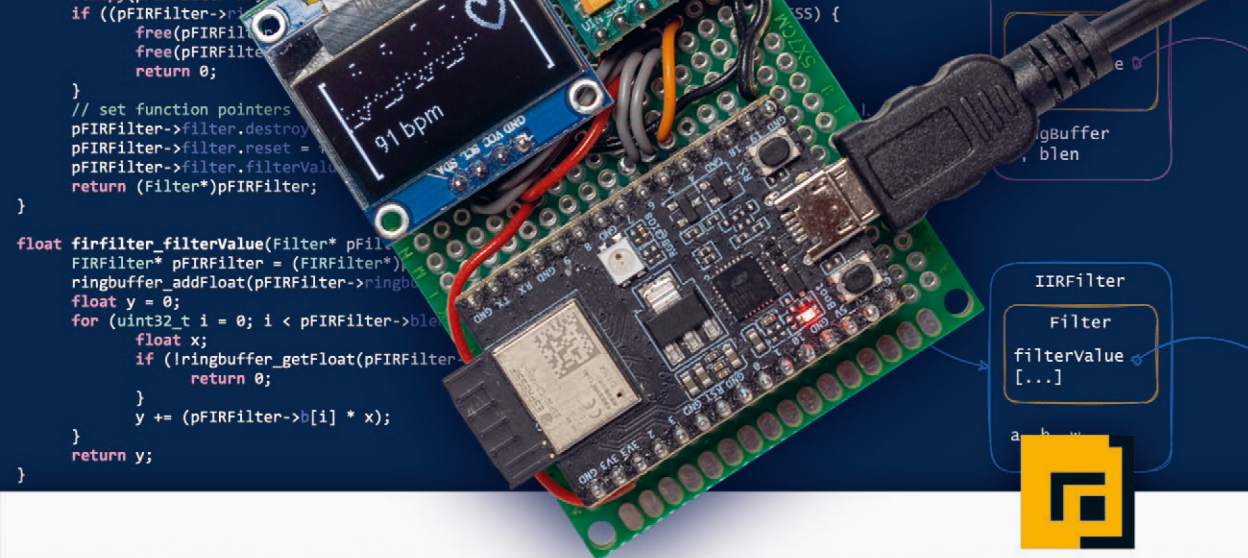
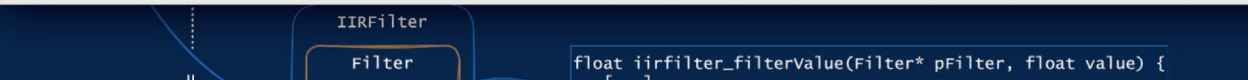


Patrick Ritschel

Embedded Systems mit RISC-V und ESP32-C3

Eine praktische Einführung in Architektur, Peripherie und eingebettete Programmierung

dpunkt.verlag



Prof.(FH) Dipl-Ing. Patrick Ritschel studierte Informatik an der TU Wien. Anschließend leitete er die Entwicklung von Smart Cards bei der Winter AG. Seit 2003 unterrichtet er embedded Systems, Programmierung und Algorithmik in C, C++ und Java, sowie Mobile Computing an der Fachhochschule Vorarlberg. Er gründete die clownfish IT GmbH, die eingebettete Anwendungen im B2B-Bereich anbietet. In seiner Freizeit zieht es ihn mit seiner Familie auf die Theaterbühne, um zu spielen, zu singen und auch Theaterstücke zu schreiben.

Patrick Ritschel

Embedded Systems mit RISC-V und ESP32-C3

Eine praktische Einführung in Architektur, Peripherie und eingebettete Programmierung

dpunkt.verlag

Patrick Ritschel
patrick@ritschel.at
www.ritschel.at

Lektorat: Gabriel Neumann
Copy-Editing: Annette Schwarz, Ditzingen
Satz: Patrick Ritschel
Herstellung: Stefanie Weidner, Frank Heidt
Umschlaggestaltung: Helmut Kraus, *www.exclam.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-86490-937-5
PDF 978-3-96910-998-4
ePub 978-3-96910-999-1
mobi 978-3-98890-000-5

1. Auflage 2023

Wieblinger Weg 17
69123 Heidelberg

Hinweis:
Dieses Buch wurde mit mineralölfreien Farben auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie. Hergestellt in Deutschland.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *hallo@dpunkt.de*.

5 4 3 2 1 0

Vorwort

»Für euch, Kinder der Wissenschaft und der Weisheit, haben wir dieses geschrieben. Erforschet das Buch und suchet euch unsere Ansicht zusammen, die wir verstreut und an mehreren Orten dargetan haben; was euch an einem Orte verborgen bleibt, das haben wir an einem anderen offengelegt, damit es fassbar werde für eure Weisheit.«

HEINRICH C. AGRIPPA VON NETTESHEIM
»DE OCCULTA PHILOSOPHIA«

Warum schreibt man heutzutage noch ein Fachbuch? – Man findet doch alle Informationen im Internet.

Diese Antwort ist grundsätzlich richtig, trifft aber nicht den Kern des Buchbegriffs:

Wenn man ein Buch liest, nimmt man es zur Hand, blättert, liest dies, sieht das. Und irgendwann auch das, was man sucht. Doch der Weg zum Gesuchten ist voller schöner Überraschungen und Informationen. Man profitiert bereits von der Suche, was bei der Internetsuche selten der Fall ist. Dort bekommt man Millionen Treffer, die nicht zu weit vom Thema weg führen.

Ein Buch ist also kein Anachronismus in einer zunehmend digitalisierten Welt, sondern durchaus zeitlos und modern. Ein E-Book erzielt, einen entsprechenden Reader vorausgesetzt, denselben Erfolg wie die Papiervariante.

Entgegen dem strukturierten Aufbau üblicher Fachbücher zu der Thematik habe ich einen beispielorientierten erzählerischen Ansatz gewählt, der entlang eines roten Fadens vom Mikroprozessor zum IoT-Ding führt. So werden viele Themen behandelt, die theoretische und praktische Bedeutung für die Entwicklung professioneller Embedded Systeme haben.

Ich möchte mich an dieser Stelle noch herzlich bei meinen geschätzten Freund:innen und Kolleg:innen bedanken, die mir mit Rat und Tat zur Seite gestanden haben. Dies sind Johannes Koch MSc, der auch Inhalte der Webseite zum Buch beigesteuert hat, Dr. Regine

Kadgien, Dr. Franz Geiger, DI Wolfgang Auer und Dr. Christoph Scheffknecht sowie die Teams des dpunkt.verlags und der Firma clownfish GmbH. Des Weiteren möchte ich meinen Lieben, Johanna, Jakob und Babsi, für ihre Geduld danken, mit der sie mich bei all meinen Ideen und Projekten unterstützen.

Ich wünsche der Leserin bzw. dem Leser viel Vergnügen beim Schmökern und Blättern, beim Löten und Messen, beim Grübeln und Programmieren! Und wenn interessante Dinge zu knapp erklärt sind, macht es doch wieder Sinn, sie nachzugoogeln ...

Inhaltsverzeichnis

I Mikrocontroller-grundlagen

1 Einleitung

»Kompliziertes kompliziert zu sagen ist einfach.
Nur Einfaches einfach zu sagen ist kompliziert.«

KARL-HEINZ KARIUS

Laut statista.com [57] wurden im Jahre 2021 weltweit 31,2 Milliarden Mikrocontroller produziert, was rund vier neuen Mikrocontrollern pro Erdenbürger entspricht. Damit sind diese Kleinstcomputer mittlerweile auch in Gegenständen des Alltags verbaut (»eingebettet«, *embedded*), in denen wir sie nicht vermuten. Oft werden Consumer-Produkte, Haushalts- und Kommunikationsgeräte, die Mikrocontroller enthalten, als »smart« bezeichnet. Vom üppig ausgestatteten Smartphone bis zur stark ressourcenbeschränkten Smart Card ist ein Mikroprozessor informationsverarbeitender Kern des Gerätes. All diese Geräte benötigen eine weitestgehend fehlerfreie Software, um mitunter ohne Update-Möglichkeit viele Jahre reibungslos zu funktionieren. Um dies zu gewährleisten, ist ein solides Grundverständnis von Aufbau und Arbeitsweise der Embedded Systeme unerlässlich.

Da die Komplexität durch wachsende Applikationsgröße und Internetanbindung steigt, wird auch die Programmentwicklung umfangreicher und komplexer. Diese IoT(»Internet of Things«)-Geräte besitzen embedded Betriebssysteme, deren Tasks miteinander kommunizieren. Auch dieses Zusammenspiel muss gut durchdacht sein, um performante Software ohne Deadlocks zu designen.

1.1 Ziel des Buchs

Ein Einstieg in die Entwicklung eingebetteter Systeme (in der Folge »Embedded Systeme« genannt) bringt verschiedene Hürden mit sich. Oft ist eine Programmiersprache bekannt, doch die technischen Details machen die Lernkurve steil und den Weg zum Ziel steinig.

Nimmt man ein Buch über die C-Programmiersprache zur Hand, enthält dieses keine Details zur Programmierung von Mikrocontrollern. Versucht man hingegen, die Datenblätter, Family Guides und Reference Manuals zu verwenden, machen die technischen Details den Einstieg schwer. Die Beispielprogramme und Application Notes auf den Webseiten der Hersteller implementieren Lösungen für sehr spezielle Probleme, was eine Umsetzung einer Lösung zu einem anders gearteten Problem schwierig gestaltet.

Insgesamt lässt sich sagen, dass eine Entwicklung oder Anpassung der Software ohne fundiertes Basiswissen aufwendiger und fehleranfälliger als mit den entsprechenden Grundlagen ist.

Ziel dieses Buches ist, der Leserin bzw. dem Leser die Grundlagen anschaulich und fundiert zu vermitteln. Aufgrund der Größe des Gebiets werden einige Bereiche nur gestreift, punktuell werden Themen aber in die Tiefe verfolgt.

1.2 Struktur des Buches

Die Struktur des Buches mag etwas ungewohnt erscheinen. Statt eines strukturierten, aufzählenden Aufbaus wurde ein beispielorientierter erzählerischer Ansatz gewählt. Anhand von Beispielen wird die embedded Welt durchwandert und erklärt.

Ein detaillierter Index am Ende des Buches dient dem Nachschlagen einzelner Themen. Der Inhalt ist drei Teilen zugeordnet, die schichtweise aufeinander aufbauen, aber jeweils für sich separat gelesen werden können, wie Abb. 1–1 zeigt.

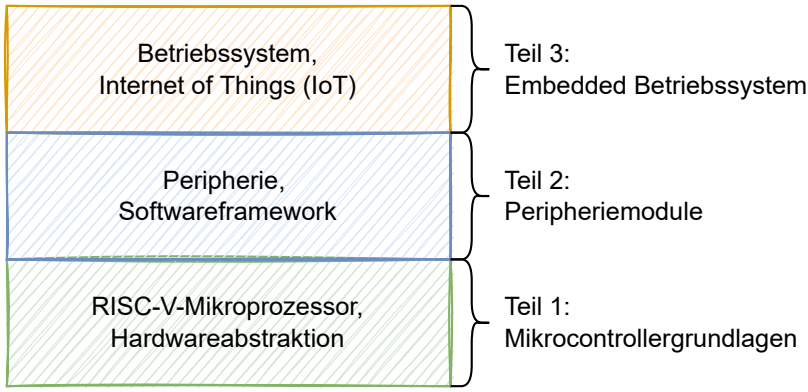


Abb. 1–1
Das Buch ist in drei Teile gegliedert.

Teil I behandelt den Aufbau eines RISC-V-Mikroprozessors, dessen Anbindung an ein Bussystem und die Grundlagen des Zugriffs auf

Peripheriemodule. Grundlagen der Assemblersprache und der hardwarenahen Programmierung in C mit Memory-Mapped I/O, Speicherverwaltung und Performanz werden vermittelt.

Teil II beinhaltet elektrotechnische Grundlagen zum Verständnis einfacher elektronischer Schaltungen. Verschiedene Peripheriemodule zur Ein-/Ausgabe, Kommunikation, Interrupt-Behandlung sowie die Verarbeitung analoger Sensordaten werden anhand der beispielhaften Implementierung eines Pulsoximeters erläutert.

Teil III bettet das Pulsoximeter in den Kontext des IoT ein, indem Daten über verschiedene Internetprotokolle verschickt werden. Die Grundlagen von embedded Betriebssystemen und deren Systemprogrammierung werden anhand des Beispiels verständlich. Eine praktische Betrachtung von Bluetooth LE und der Möglichkeiten des Stromsparens rundet das Kapitel ab.

1.3 Zielpublikum

In erster Linie ist dieses Buch für den Einsatz im einführenden und fortgeschrittenen Unterricht über Embedded Systeme geplant. Es ist von großem Vorteil für das Verständnis, wenn Grundlagen der Informatik und des Programmierens in der Programmiersprache C vorhanden sind. Alternativ sind Grundlagen in einer anderen Programmiersprache sehr anzuraten. Die einzelnen Teile bieten sich an, in separaten Lehrveranstaltungen zu Rechnerarchitektur/-organisation (Teil I), Embedded Programmierung (Teil II), Betriebssystemen (Teil III) und Kommunikationssystemen/IoT (Teil III) Eingang zu finden.

In zweiter Linie richtet sich das Buch an interessierte Leser:innen mit informatischem Background. Entsprechendes Interesse vorausgesetzt profitiert diese Leserschaft vom Inhalt. Die Lesereihenfolge ist beliebig, idealerweise sequenziell vom Anfang zum Ende.

Auch Personen, die bereits im Embedded Systems-Umfeld aktiv sind, sind von diesem Buch angesprochen. Die technischen Details zu Architektur und Programmierung, die hier zusammengetragen sind, vertiefen das vorhandene Wissen. Dieser Leserschaft ist angeraten, das Buch von vorne nach hinten zu lesen und bekannte Teile zu überfliegen. Beim Überspringen könnten wertvolle Details ausgelassen werden.

Für alle Leser:innen dient das Buch als Nachschlagewerk zu den vielfältigen Technologien, die in Embedded Systemen zum Einsatz kommen.

1.4 Gebrauchsanweisung

Es ist möglich, das Buch nur zu lesen. Um ideal zu profitieren, ist anzuraten, die entsprechende embedded Hardware anzuschaffen und die Beispiele im Buch nachzuvollziehen. In diesem Sinne handelt es sich nicht um ein Lese-, sondern ein Arbeitsbuch.

Embedded Hardware Als embedded Hardware findet ein ESP32-C3-Mikrocontroller (siehe Abschnitt 2.2.1), basierend auf einem modernen RISC-V-Prozessor, Anwendung. Die Beispiele erfordern teilweise zusätzliche Komponenten wie eine Steckplatine (siehe Abschnitt 5.3.4) und Bauteile, die auch mit einem kleinen Budget zu beschaffen sind. Quellen zur Materialbeschaffung finden Sie auf der Webseite zum Buch (siehe Anhang A).

Beispielprogramme Die Herausforderungen, die sich bei den ersten Implementierungen im embedded Umfeld ergeben, folgen typischerweise bestimmten Mustern, die sich hauptsächlich aus der Interaktion des Systems mit seiner Umgebung ergeben. Solche Muster werden im Buch zum besseren Verständnis durchgehend in Beispielprogrammen verwendet. Um diese besser nachzuvollziehen, ist der Code der Beispiele online zugänglich (siehe Anhang A) abgelegt. Dies ist vor allem beim großen Pulsoximeterbeispiel wichtig, da im Buch wesentliche Teile, aber nicht die gesamten Sourcen abgedruckt sind.

Übungsbeispiele Jeder Teil verfügt über theoretische und praktische Übungen. Diese dienen dem Sammeln von Erfahrungen mit der embedded Plattform und dem Üben anhand typischer Problemstellungen. Ein Vergleich mit den bereitgestellten Lösungen hilft bei der Beurteilung der eigenen Ausarbeitung.

Wichtig ist dabei zu beachten, dass eine Musterlösung nicht die einzig sinnvolle Lösung darstellt. Es gibt immer viele Wege zum Ziel, die jeweils ihre eigene Begründung haben. Deshalb werden auf der Webseite zum Buch (siehe A) Kommentare zu den Musterlösungen und alternative Ansätze gesammelt und bereitgestellt.

1.4.1 Konventionen

Im Bereich der Informatik werden oft Fachbegriffe verwendet, die eine sperrige oder ungewohnte deutsche Übersetzung haben. Aus diesem Grund werden die Begriffe bei der ersten Verwendung in »französischen« Anführungszeichen eingeführt und dann direkt verwendet. Eine Vermischung wie »Embedded Systeme« statt »eingebettete

Systeme« oder »embedded systems« ist eine unweigerliche Folge, die dem Autor bitte verziehen wird.

Die abgedruckten Sourcen in der Programmiersprache C wurden der Übersichtlichkeit halber auf den Platzbedarf hin optimiert. Umbrüche langer Zeilen werden mit ↪ dargestellt. Der Programmierstil ist modern und teils ungewöhnlich. So wird ein `int *pX` als `int* pX` dargestellt, um zu zeigen, dass der Pointer zum Typ gehört und nicht zum Namen. Die Benennung von Variablen und Funktionen ist modern in Camel-Case. Wer das unangebracht findet, möge bitte über diese Spitzfindigkeiten hinwegsehen und die eigenen Konventionen weiter verwenden.

Werden im Fließtext Variablen verwendet, werden diese in einer `Terminal-Schriftart` dargestellt. Zur einfachen Darstellung, dass es sich um eine Funktion handelt, wird diese mit runden Klammen, aber ohne Parameterliste, angegeben, beispielsweise `doIt()`.

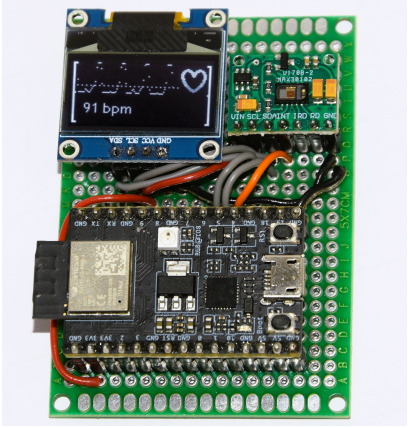


Abb. 1–2
Die Implementierung des Pulsoximeters `poxi` *bildet die Basis der Teile II und III.*

Weiterführendes

Einführende oder weiterführende Themen werden in separaten Kästen untergebracht. Bei Interesse können sie gelesen werden; sie sind allerdings nicht im Fließtext eingebettet.

Die Größenverhältnisse von Kleinsystemen bieten sich für eine kurze Betrachtung an. Die kleinsten Mikrocontroller sind mit Gehäuse kleiner als 2 mm x 2 mm x 0,5 mm. Der in diesem Buch verwendete ESP32-C3-Mikrocontroller hat in etwa die Leistungsfähigkeit eines Intel-Pentium-PC-Prozessors und passt mit seiner Größe 5 mm x 5 mm x 0,85 mm etwa 6½ Mal auf dessen Siliziumfläche - inklusive RAM, Wi-Fi und Bluetooth-Hardware.

Die weiteren Themen sind nicht minder spannend!

2 Hallo, Welt!

»Das Durchschnittliche gibt der Welt ihren Bestand,
das Außergewöhnliche ihren Wert.«

OSCAR WILDE

Seit Kernighan und Ritchie in ihrem Buch *The C Programming Language* [36] gleich zu Beginn ein Programm schrieben, das

```
hello, world
```

auf dem Bildschirm ausgab, verwenden viele Programmierbücher diesen Ansatz. Die Begründung, »der einzige Weg, eine Programmiersprache zu lernen, ist, Programme in ihr zu schreiben«, ist ja durchaus plausibel.
Da dieses Buch unter anderem den Anspruch erhebt, die Programmierung von Embedded Systemen als Fertigkeit zu erlernen, wird dieser beispielgetriebene Ansatz auch hier verfolgt. In diesem Kapitel wird ein erstes C-Programm erstellt, auf eine embedded Plattform mit einem RISC-V-basierten Mikrocontroller aufgespielt und dort gestartet. Der Debugger dient dann zur schrittweisen Programmausführung.

Auf diese Weise werden die einzelnen Komponenten des Entwicklungsflusses verständlich. Begleitend zu dieser praktischen »Implementierung« werden die Entwicklungsumgebung und das »Embedded System«, nämlich der Rechner, auf dem das Programm ausgeführt wird, erläutert. Dieses Embedded System unterscheidet sich doch stark von einem klassischen PC mit Monitor, Tastatur, Maus und grafischer Benutzeroberfläche.

PC, Personal Computer: Heimcomputer oder Arbeitsplatzrechner wie Desktop, Notebook oder Tablet

Embedded System

Unter einem *embedded system* (zu Deutsch: *eingebettetes System*) versteht man ein Computersystem, bestehend aus Hard- und Software, das in einen technischen Kontext eingebettet ist. In diesem Kontext verrichtet es Arbeiten wie Überwachung, Steuerung, Regelung und die weitere Datenverarbeitung. In modernen Systemen nimmt die Kommunikation eine wachsende Rolle ein, was sich in den technischen Modeschlagworten IoT und IIoT (*[Industrial] Internet of Things*: Sensoren und andere Geräte, die [im industriellen Umfeld] vernetzt sind) niederschlägt.

Embedded Systeme treten in ihren Applikationen oft so weit in den Hintergrund, dass sie für Anwender unsichtbar sind oder nicht mehr als Computer wahrgenommen werden. Beispiele sind moderne Haushaltsmaschinen, Unterhaltungsgeräte wie Uhren etc., aber auch Geräte der Kommunikationsinfrastruktur, Industrie und Fahrzeuge vom Automotive-Bereich bis zur Raumfahrt. Aufgrund dieser Unsichtbarkeit, Durchdringung und Allgegenwärtigkeit von Computersystemen stößt man im Umfeld auf die Begriffe *invisible*, *pervasive* und *ubiquitous* Computing.

Da solche Systeme oft mobil sind, keinen Anschluss an das Stromnetz haben, auch extremen Umweltbedingungen ausgesetzt sind und technisch in großen Stückzahlen produziert werden, liegt der Fokus auf kleinen, stromsparenden, robusten und günstigen Komponenten. Dadurch nicht vergleichbar mit üppig ausgestatteten PCs sprechen wir von *ressourcenbeschränkten Systemen*. Diese Beschränkung von Ressourcen wie Arbeitsspeicher, Akkukapazität, Bandbreite und Latenz der Kommunikation, Ein-/Ausgabemöglichkeiten, Antwortzeit und Echtzeitfähigkeit, Kosten etc. wirkt sich direkt auf die gesamte Hard- und Softwareentwicklung aus.

Die Hardware besteht neben Gehäuse und mechanischen Komponenten aus einer Elektronik, die diverse Schnittstellen bereitstellt. Neben LEDs, Displays, Tastern, Joysticks, Segmentanzeigen, Leistungselektronik, Sensoren für Temperatur, Druck, Helligkeit, Beschleunigung und vielem mehr enthält die Hardware als steuernde Komponente einen Mikroprozessor bzw. Mikrocontroller.

2.1 Wahl der Programmiersprache

Für die Programmierung von Embedded Systemen werden verschiedene Programmiersprachen angepriesen und in der Praxis auch verwendet. Sowohl von Skriptsprachen als auch von Sprachen mit einer virtuellen Maschine wird in diesem Buch aus mehreren Gründen Abstand genommen:

Ein wesentliches Ziel dieses Buches ist es, ein grundlagenbasiertes Verständnis des Systems zu vermitteln, weshalb auf die gesamte

Hardware direkt, also ohne interpretierende Zwischenschicht, zugegriffen wird. Python, Lua, Java, C# usw. fallen dadurch weg.

Das wohl stärkste Kriterium bei der Auswahl einer Programmiersprache für Embedded Systeme ist aufgrund der beschränkten Ressourcen die Performanz in Bezug auf Ausführungsgeschwindigkeit und Speichernutzung. Um den Zugang zu sämtlichen Ressourcen zu ermöglichen, spielt die Hardwarenähe ebenso eine große Rolle, was die Sprache *C* mit ihrem Pointer-Konzept in den Fokus rückt.

Viele Konstrukte und Paradigmen moderner Sprachen wie Objektorientierung und funktionale Programmierung spielen in den hardwarenahen Schichten eine untergeordnete Rolle. Bei der Bewältigung von Aufgaben mit hoher Komplexität, wie sie in höheren Schichten üblich sind, sind sie aber hilfreich, weshalb Sprachen mit derartigen Konzepten hier breiten Einsatz finden. Die Sprache C++ kann deshalb als durchgängige Sprache eingesetzt werden, wird aber gerade wegen der Fülle an Funktionalität und damit einhergehender Komplexität und Beherrschungsschwierigkeiten oft gemieden.

Meist fällt die Wahl als Sprache der »unteren Schichten« auf die Programmiersprache C, was sich auch dadurch ausdrückt, dass die Mikrocontrollerhersteller vorrangig C-Code in ihren Entwicklungskits, Application Notes und Treiberbibliotheken bereitstellen.

Der TIOBE Index beurteilt monatlich die Popularität von Programmiersprachen.

Aus diesen Gründen wurde auch für die Praxisbeispiele in diesem Buch die Programmiersprache C gewählt. Die Einfachheit und Klarheit der Sprache dürfen ebenfalls nicht unterschätzt werden. Die Programmiersprache C gehört laut TIOBE Index [60] zu den populärsten Programmiersprachen überhaupt. Zuletzt war C 2019 »Programming Language of the Year«.

C wurde mit dem Ziel entwickelt, eine Hochsprachenabstraktion zur Assemblersprache zu bieten. Als Resultat ist C-Code verhältnismäßig leicht in Assembler zu übersetzen, was den Aufwand der Compiler-Portierung auf eine neue Prozessorplattform gering hält. Der freie GNU C Compiler (gcc) ist auf allen gängigen Plattformen und Betriebssystemen verfügbar, und C-Programme sind damit leicht auf diese portierbar. Die Performanz des übersetzten C-Codes ergibt sich auch daraus, dass Mikroprozessorarchitekturen wie RISC-V, ARM, MIPS und weitere gemeinsam mit einem (und damit für einen) C-Compiler entwickelt werden.

Eine interessante, weil auf Performanz und Sicherheit hin entwickelte Sprache stellt Go dar. Aufgrund der derzeit geringen Verbreitung wird diese objektorientierte Sprache in diesem Buch aber nicht eingesetzt. Eine weitere Sprache mit diesem Fokus ist Rust, auf dessen Grundlage Google das embedded Betriebssystem KataOS entwickelt.

Eine weitere hardwarenahe Programmiersprache, die aber zunehmend durch Hochsprachen ersetzt wird, ist Assembler. Moderne optimierende Compiler generieren Code, der an Effizienz oft handgeschriebenes Assembly übertrifft, und das bei schnellerem Entwicklungstempo und stärkerer Sicherheit der Hochsprachen. Im Rahmen dieses Buches wird RISC-V Assembler gestreift, um die RISC-V ISA (Instruction Set Architecture) zu verstehen. Ebenso wird das Disassembly beim Debuggen verwendet, um schwer zu findende Fehler zu lokalisieren.

2.2 Benötigte Komponenten für die Applikationsentwicklung

Damit eine Applikation entwickelt werden kann, werden verschiedene Komponenten benötigt, wie sie auch in Abb. 2–1 ersichtlich sind. Nach der folgenden Übersicht wird in diesem Abschnitt detailliert auf die einzelnen Teile eingegangen.

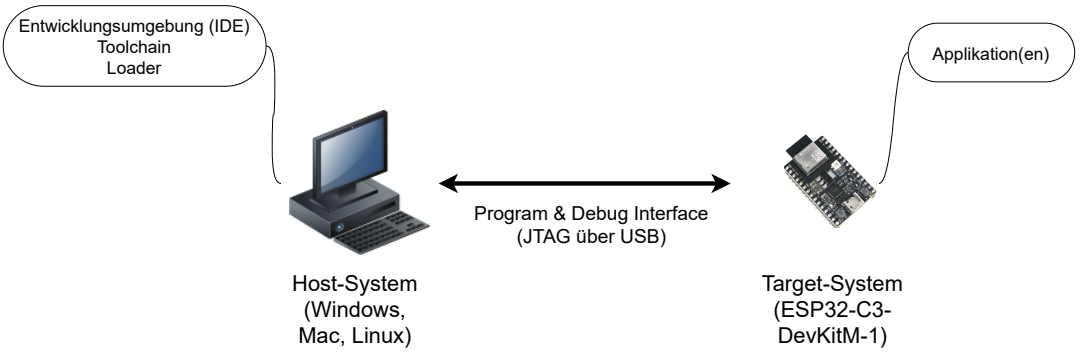


Abb. 2–1
Komponenten der Cross-Platform-Entwicklung

Target-System Einerseits wird das »Target-System«, also das Zielsystem, benötigt. Auf diesem wird die erstellte Software ausgeführt. Es ist auch möglich, per *ISD* (»In-System Debugging«) die Software auf der Zielplattform zu debuggen, also schrittweise auszuführen, Variablen einzusehen und vieles mehr. Für die weitere Arbeit mit diesem Buch empfiehlt sich das in Abschnitt 2.2.1 vorgeschlagene preiswerte RISC-V-Entwicklerboard.

Host-System Wenn sich, wie in unserem Fall, das »Host-System«, auf dem die Software entwickelt wird, vom Target-System unterscheidet, spricht man von »Cross-Platform-Entwicklung«. Das

Host-System ist typischerweise ein Windows-PC, auf dem die integrierte Entwicklungsumgebung (siehe Abschnitt 2.2.2) läuft. Da ein Apple Mac sich hardwaretechnisch nicht wesentlich von anderen PCs unterscheidet, ist diese Computerklasse unter dem Begriff »PC« in diesem Buch mit eingeschlossen. Mithilfe einer »Toolchain«, also einer Sammlung von Softwarewerkzeugen, wird der Sourcecode in eine Applikation übersetzt und anschließend mit einem Loader auf das Zielsystem übertragen.

Die in den Beispielen eingesetzte Software läuft auf PCs mit den Betriebssystemen Windows, Linux und macOS. Hinweise zur Installation und Benutzung sind in Anhang A.1 zu finden.

Program & Debug Interface Die Kommunikation zwischen Host- und Target-System wird über das Program & Debug Interface gewährleistet. In der Praxis kommen auch verschiedene Interfaces für beide Zwecke zum Einsatz, also beispielsweise serielle Kommunikation (RS-232) zum Programmieren und ein JTAG (Join Test Action Group) Interface zum Debuggen. Diese Interfaces sind üblicherweise kabelgebunden. Bei vielen Development Boards werden diese Schnittstellen zugleich mit der Stromversorgung über USB angeboten.

2.2.1 Development Board

Die von vielen Herstellern angebotenen »Development Boards« (Entwicklungsboards) sind mit einem Mikrocontroller, verschiedener Peripherie (LEDs, Taster, Displays, Sensoren und Aktoren verschiedener Art) sowie meist einem Program & Debug Interface ausgestattet.

Espressif ESP32-C3-DevKitM-1

Das Entwicklungsboard ESP32-C3-DevKitM-1 von Espressif (siehe [14]) wird für die Beispiele in diesem Buch verwendet. Grundsätzlich kann auch ein anderes Board genutzt werden. Die Beispiele müssen in diesem Fall durch die Leserin bzw. den Leser angepasst werden.

Abb. 2–2 zeigt das Board mit den Komponenten:

ESP32-C3-MINI-1 Dieses Modul beinhaltet den RISC-V-Mikrocontroller ESP32-C3Fx4 (siehe Kapitel 3), Flash-Speicher, einen Quarz und die Antenne für Wi-Fi und Bluetooth.

Micro-USB Port Dieser Anschluss dient der Programmierung und der seriellen Datenausgabe, wofür auch die »USB-to-UART Bridge«

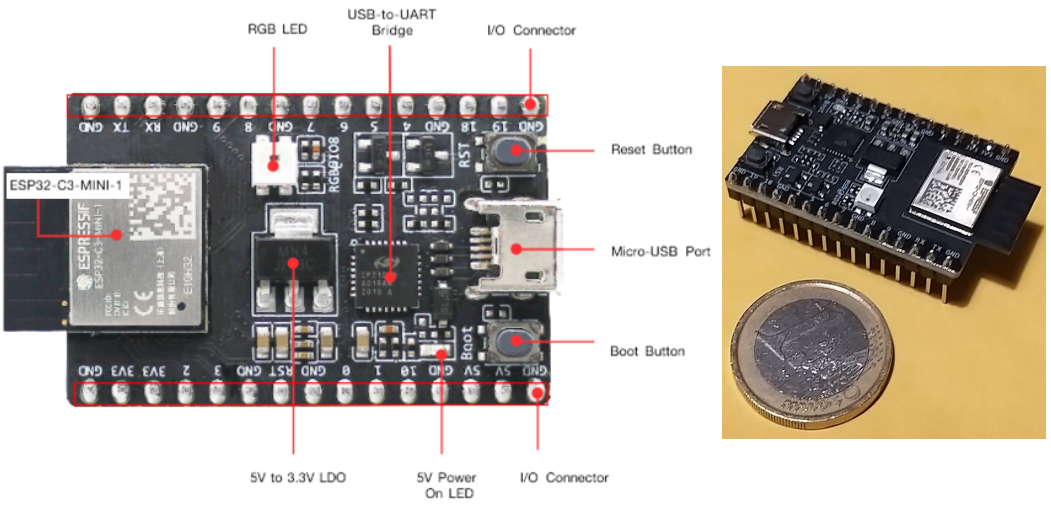


Abb. 2–2
Espressif ESP32-C3-DevKitM-1 Development Board

verwendet wird. Alternativ besteht die Möglichkeit, zwei Widerstände umzulöten, um JTAG Debugging auf der USB-Schnittstelle anzubieten. In Anhang A.1 wird gezeigt, wie auf die eingebaute JTAG-Schnittstelle zugegriffen werden kann, alternativ auch ohne Notwendigkeit des Lötens.

5V Power On LED Diese LED leuchtet, wenn das Board per USB mit einer Spannung von 5V versorgt wird. Der »5V to 3.3V LDO« ist ein Spannungswandler, der die benötigte 3,3-V-Spannung für das Board aus der USB-Spannung generiert.

Taster Per »Reset Button« wird das System neu gestartet, wobei die Daten im RAM verloren gehen. Wird der »Boot Button« während des Neustarts gedrückt, wird der serielle Upload-Modus des Bootloaders gestartet. Andernfalls wird die Applikation gestartet. Nach dem Reset kann der Taster als Eingabemöglichkeit für die Applikation verwendet werden.

RGB LED Diese mehrfarbige LED kann in der Applikation beliebig als Benutzerschnittstelle verwendet werden.

I/O Connector Diese beiden Stiftleisten bilden die Ein- und Ausgänge des Mikrocontrollers ab. Die Beispiele des Teils II Peripheriemodule verwenden diese Steckverbindung extensiv.

Die Ausstattung dieses Boards ist im Vergleich mit Development Boards anderer Hersteller nicht üppig. Diese Beschränkung kann aber auch von Vorteil sein: Wenn man eigene Hardware für ein embedded-Gerät aufbauen möchte und eventuell eine kleine Serie produzieren will, wird die zusätzliche Peripherie nicht verwendet und verteuert

nur das Produkt. Die Vorgehensweise in diesem Fall ist, dass man eine kleine Platine mit benötigter Peripherie fertigt und das Development Board darauf ansteckt. Bei einer größeren Serie kommt dann eine Platine zum Einsatz, auf die das ESP32-C3-MINI-1-Modul und andere Komponenten des Entwicklerboards direkt aufgelötet werden.

Geliefert wird das Board in einer Konfiguration, die das Programmieren und die Ausgabe von Statusmeldungen über USB erlaubt. Ein Debuggen per JTAG ist nicht direkt möglich. Eine Spezialität des eingesetzten Mikrocontrollers ist aber ein integriertes JTAG-over-USB-Modul. Es muss also im Grunde nur ein USB-Kabel an die richtigen Pins der Stiftleisten gehängt werden, um das Debugging zu ermöglichen.

Weitere Informationen zur Verwendung des Boards, zur Installation der Software sowie zur Vorgehensweise beim Debugging sind in Anhang A.1 hinterlegt.

Blockschaltbild

Um die einzelnen Komponenten und deren Zusammenspiel zu zeigen, wird statt eines Fotos wie in Abb. 2–2 üblicherweise ein schematischer Aufbau wie in Abb. 2–3 gezeigt. Bei dieser Darstellungsform, dem »Blockschaltbild«, werden Funktionsblöcke mit Rechtecken und deren Verbindungen (Signale, Leitungen) mit Pfeilen dargestellt. Elektrische und zeitliche Zusammenhänge spielen dabei eine untergeordnete Rolle, sodass das Zusammenspiel der Komponenten im Vordergrund steht und intuitiv erfasst werden kann. Die Pfeile geben dabei die logische Richtung des (Signal-)Flusses an.

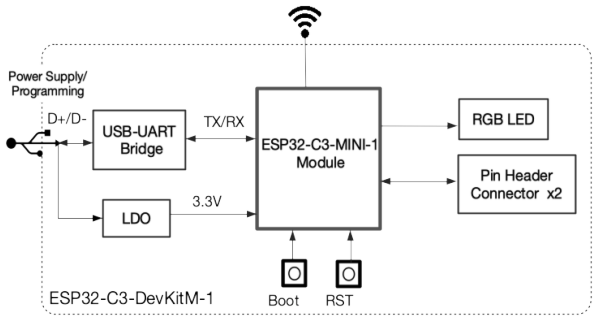


Abb. 2–3
Schematischer Aufbau des Espressif ESP32-C3-DevKitM-1

Weitere Symbole, wie in der Abbildung das USB-Symbol links und das Antennensymbol oben, können zur weiteren Veranschaulichung verwendet werden. Im schematischen Aufbau ist so beispielsweise ersichtlich, dass

- die beiden Taster unten ihren Status an das ESP32-Modul senden,
- die RGB LED vom Modul gesteuert wird,
- das Modul über die Leitungen TX und RX per USB-UART Bridge an USB bidirektional angeschlossen ist, also Daten senden und empfangen kann,
- zwei Stecker (»x2«) angeschlossen sind, über die Daten ein- und ausgegeben werden können,
- der LDO aus der USB-Spannung die 3,3V für das Modul erzeugt und
- eine Antenne angeschlossen ist.

Derartige Blockschaltbilder werden in der Praxis vielfach verwendet. So finden sie auch in den Datenblättern und Reference Manuals der verschiedenen Hersteller Verwendung.

2.2.2 Software für die Entwicklung

Auf dem Host-System, also dem PC, wird verschiedene Software für die Programmentwicklung benötigt. Grundsätzlich stellt sich die Wahl, ob kostenfreie oder zahlungspflichtige Software verwendet werden soll. In diesem Buch wird, wie in vielen Projekten der Wirtschaft, robuste und ausgereifte freie Software verwendet, um den Einstieg in die Entwicklung zu erleichtern.

In der Praxis kann es auch aus mehreren Gründen nötig werden, zahlungspflichtige Software einzusetzen. Teilweise sind Bibliotheken oder Support nicht frei verfügbar, teilweise möchten sich die Entwickler auch gegen die Verwendung fehlerhafter Tools absichern: Nur wenn der Hersteller bekannt ist und für die Software haftet, kann im Schadensfall ein Regress erfolgreich abgewickelt werden.

Ein grundsätzlicher, wesentlicher Aspekt bei Entwicklung und Vertrieb von Software ist die Haftung im Fehlerfall. Mögen Fehler in kaufmännischer Software finanzielle Schäden bewirken, die auch finanziell abgegolten werden können, besteht bei Geräten mit Einfluss auf die Umwelt das Problem, dass Sach-, Umwelt- oder auch Personenschäden auftreten können. Da diese teils strafrechtlichen Konsequenzen nicht durch Versicherungen abgedeckt werden können, muss hier zu besonderer Vorsicht und Einhaltung der bestehenden Richtlinien geraten werden.

Development Toolchain

Unter einer »Development Toolchain« wird eine Sammlung von Werkzeugen verstanden, die bei der Implementierung von Software einge-

setzt wird. Unter der Implementierung wird der Teil der Softwareentwicklung verstanden, der sich mit der Programmierung, Ausführung und der Fehlerlokalisierung beschäftigt. Essenzielle begleitende und organisatorische Maßnahmen wie Analyse, Design, Spezifikation, Testen, Dokumentation, Zertifizierung usw. sind nicht Teil der Implementierung.

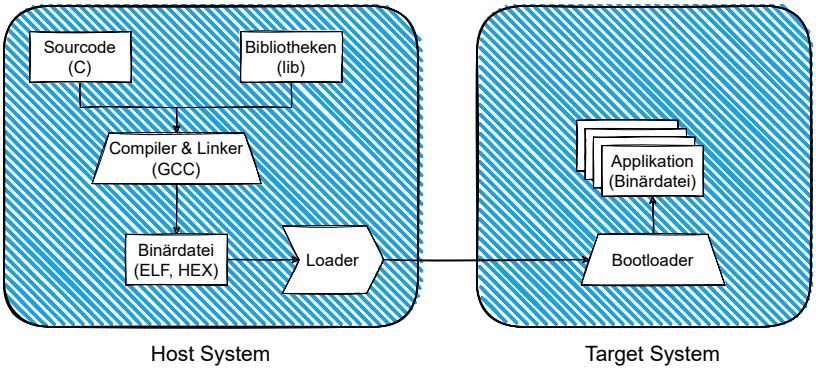


Abb. 2–4
Cross-Platform Development Toolchain

Abb. 2–4 zeigt die wesentlichen Werkzeuge, die beim Kompilieren und Aufspielen der Applikation zum Einsatz kommen. Es befinden sich noch wesentlich mehr Werkzeuge in der Toolchain, die aber in der Abbildung nicht gezeigt sind.

Die auf dem Host-System befindlichen Quellcodedateien (in den Sprachen C, C++ geschrieben) werden vom Compiler in die Assemblersprache des Target-Systems übersetzt. Dabei handelt es sich um eine Sprache, die die Befehle (»Instructions«) der Maschinensprache in menschenlesbarer Form (sogenannte »Mnemonics«) abbildet. Ein Assembler übernimmt dann die Transformation der Mnemonics in Instructions und erzeugt Objektdateien. Die entstandenen Assemblerdateien sind temporär und werden wieder gelöscht. Der Linker nimmt diese und weitere Objektdateien (z.B. aus Bibliotheken) und ordnet sie hintereinander im Speicher an. Referenzen (»Links«) von einer Datei in eine andere werden dabei aufgelöst.

Das Ergebnis dieses sogenannten »Build-Prozesses« ist die Applikation als Binärdatei, typischerweise im ELF(*Executable and Linking*)-Format mit Debug-Informationen oder im Intel HEX(*Hexadecimal Object File*)-Format ohne weitere Informationen. Da diese Applikation auf dem Host-System erzeugt und gespeichert wird, aber auf dem Target-System ausgeführt, also eine Systemgrenze überquert wird, spricht man hier von »Cross-Platform Development«.

Eine wichtige begriffliche Unterscheidung liegt zwischen »statisch« (zur Compile-Zeit) und »dynamisch« (zur Laufzeit). Die Laufzeit eines Programms beginnt mit dem Laden des Programms in

den Speicher. Somit sind bereits das Laden, das Anlegen der globalen Variablen im RAM, die Ausführung, die Reservierung von Speicherplatz für lokale Variablen auf dem Stack usw. dynamisch. Das Schreiben des C-Codes, das Kompilieren und Linken (»Binden«), das Schreiben des Programmspeichers usw. sind hingegen statisch.

Um die Systemgrenze physisch zu überwinden, wird die Applikation über einen Loader auf das Target-System übertragen. Am Zielsystem ist hierfür eine minimale Startsoftware untergebracht, der sogenannte Bootloader, der die Applikation per serieller Verbindung übernimmt und permanent speichert. Der Bootloader lässt sich in ausgelieferten Produkten deaktivieren, sodass eine nachträgliche Änderung der Software nicht mehr möglich ist.

Es gibt auch Systeme ohne Bootloader, die auf anderen Wegen (z.B. per JTAG) programmiert werden.

Für die Beispiele in diesem Buch wird eine Anpassung der GNU Toolchain für RISC-V in Verbindung mit weiteren Tools unter dem Namen ESP-IDF (*Espressif IoT Development Framework*) verwendet. Informationen zur Installation finden Sie in Anhang A.1.

IDE, Integrierte Entwicklungsumgebung

Ursprünglich erfolgte die Bedienung der Toolchain über eingegebene Kommandos in der Konsole. Mit einem separaten Editor wurde der Quellcode geändert, mit einem Terminalprogramm wurden die Programmausgaben angezeigt. Moderner ist die Verwendung einer IDE (*Integrated Development Environment*), die Editoren und Toolchain unter einer Oberfläche miteinander vereint und automatisiert verknüpft.

Für den ESP32-C3 werden die weit verbreiteten IDEs *Eclipse* und *Visual Studio Code* mit entsprechenden Erweiterungen (*Plug-ins*) unterstützt. Für die Beispiele in diesem Buch wird Eclipse aufgrund der weiten Verbreitung im embedded Umfeld verwendet.

Abb. 2–5 zeigt die ursprünglich für Java entwickelte, leicht erweiterbare IDE Eclipse während des Debuggings. Das Mittelfenster zeigt den Quellcode in automatischer Einfärbung, während die Programmausführung gerade in Zeile 128 angehalten ist. Rechts ist das Variablenfenster, das die aktuellen Werte der lokalen Variablen anzeigt, zu sehen. Die tatsächliche Ausführung findet dabei direkt auf dem angeschlossenen Mikrocontroller statt (ISD, *In-System Debugging*).

Weitere Informationen zur Installation und Verwendung der IDE Eclipse und auch der Alternative Visual Studio Code finden Sie in Anhang A.1.