Embedded Controller

Rüdiger R. Asche

Embedded Controller

Grundlagen und praktische Umsetzung für industrielle Anwendungen

Springer Vieweg

Rüdiger R. Asche
Wiesbaden, Deutschland

ISBN 978-3-658-14849-2 ISBN 978-3-658-14850-8 (eBook)
DOI 10.1007/978-3-658-14850-8

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

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist Teil von Springer Nature
Die eingetragene Gesellschaft ist Springer Fachmedien Wiesbaden GmbH
Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Strasse 46, 65189 Wiesbaden, Germany

Vorwort

Das vorliegende Buch richtet sich an alle an der Entwicklung von Software für Embedded Controller (deutsch „eingebettete Systeme“) interessierten Menschen. Sind Sie

- Einsteiger in die Embedded-Welt oder ein „Professional Maker“, der hinter die Kulissen von Arduino und Co. schauen möchte? Dann wird Ihnen dieses Buch im ersten Abschnitt im Schnelldurchgang das Handwerkszeug vermitteln, um den Einstieg in die Welt der Firmwareentwicklung für Embedded Controller zu erleichtern und das Verständnis der tiefergehenden folgenden Kapitel zu ermöglichen.
- „Umsteiger“, also bereits Entwickler, haben sich allerdings als Elektrotechniker bislang hauptsächlich mit der Hardwaresteuerung eines Mikrocontrollers beschäftigt, aber noch nicht so richtig mit Betriebssystemen und Dingen wie systemweite Integration Ihrer Hardware, beispielsweise im Rahmen der Umsetzung von Hostkommunikationsprotokollen, anfreunden können? Dann wird Ihnen dieses Buch die Informationen geben, die Sie brauchen.
- „Höhlenforscher“, also erfahrener Softwareentwickler für PC- oder Smartphoneapplikationen oder Serversysteme, der sich in die „dunklen Kellergewölbe“ der eingebetteten Systeme einarbeiten will oder muss? Dann wird Ihnen dieses Buch die Unterschiede der Entwicklungsumgebungen und des Entwicklungsprozesses zwischen Ihren bislang gewohnten Systemen und den hardwarenah zu programmierenden eingebetteten Systemen vermitteln und damit den Übergang erleichtern.
- bereits mit der Entwicklung von Firmware für Embedded Controllern vertraut? Dann kann Ihnen dieses Buch einige Tipps und Tricks aus der 20-jährigen Berufspraxis des Autors vermitteln, die Ihnen so vielleicht noch nicht begegnet sind.

Der Fokus dieses Buches liegt auf eingebetteten Systemen im Einsatz für industrielle Anwendungen, die durch folgende Charakteristiken beschrieben werden können:

- Sie sind in der Regel im unbeaufsichtigten Einsatz, beispielsweise als Messfühler an für Menschen unzugänglichen Orten oder als Alarmsysteme an selten begangenen Orten.
- Sie haben deswegen kein oder nur ein sehr rudimentäres Mensch-Maschine-Interface.

- Sie müssen während ihrer Lebenszeit 24*7*365 Stunden im stabilen Dauereinsatz ihren Dienst tun, müssen daher also auch nach unvorhergesehenen Ausnahmesituationen wie Stromausfällen oder elektromagnetischen Störungen wieder in ihren regulären Dauerbetrieb übergehen.
- Sie müssen, auch nach den eben beschriebenen Ausnahmesituationen, jederzeit zu Wartungszwecken über ein definiertes Interface erreichbar sein.
- Sie sind in der Regel nicht auf Standardhardware wie Arduino oder BeagleBoard aufgesetzt, sondern auf eigens entwickelten Platinen, die sowohl aus technischen als auch aus wirtschaftlichen Gründen auf den minimalen Ressourcenbedarf der Anwendung feinoptimiert sind.

An die Hardware solcher Systeme werden in der Regel bereits höhere Anforderungen gestellt als an Geräte, die in einer direkten Interaktion mit Menschen stehen: Es gelten höhere Hürden zur Zulassung bezüglich der elektromagnetischen Verträglichkeit; außerdem sind die Temperatur- und Umweltanforderungen höher, unter denen die Geräte arbeiten müssen. Darüber hinaus haben industrielle Systeme oft eine sehr viel längere Betriebsdauer als Consumergeräte, weswegen beim Design der zugrunde liegenden Hardware in Bezug auf Nachlieferungen und Austauschgeräte auf eine längere Verfügbarkeit der Einzelkomponenten geachtet wird.

Auf Grund der oben skizzierten Anforderungen ist auch die Software, die in industriellen Controllern zum Einsatz kommt, höheren Anforderungen ausgesetzt als die Software in Consumergeräten, was sich sowohl auf das Systemdesign als auch auf die Implementation als auch auf das Testen auswirkt. Im Gegensatz zu Consumergeräten lassen sich industrielle Controller im Feldeinsatz schwerer bis gar nicht im Fehlerfall analysieren, also muss sichergestellt werden, dass Fehler, die die Geräte nicht mehr betriebsbereit hinterlassen, im Feld ausgeschlossen sind.

Die Dauerstabilität und Robustheit der Firmware in diesen industriellen Geräten sowie die Anforderung, mit einem Minimum an Ressourcen auszukommen, ist der „rote Faden", der sich durch dieses Buch zieht. Die Vorgehensweise ist dabei top down, das heißt wir werden zunächst in eher abstrakten Betrachtungen aus der Vogelperspektive den Überblick gewinnen, uns dann recht zügig dem Boden nähern und schließlich den einen oder anderen Wurm aus dem Erdreich ziehen. Deswegen wird das vorausgesetzte Wissen zum Verständnis der diskutierten Inhalte auch mit der fortschreitenden Seitenzahl deutlich wachsen.

Zum Zeitpunkt der Drucklegung dieses Buches spielt sich ein interessanter Paradigmenwechsel am Markt der eingebetteten Systeme ab, an dem ARM®[1] Cortex®-Prozessoren einen wichtigen Anteil haben. Wie wir in Kap. 2 sehen werden, führt der Erfolg der Cortex®-basierten Prozessoren unter anderem dazu, dass Hersteller von Prozessoren immer größere Probleme haben, ihre Produkte von Konkurrenzprodukten abzusetzen. Eine momentan wichtige Waffe in dem Kampf um Märkte ist dabei die der Software: Jeder

[1] AMBA®, ARM®, Cortex® und Thumb® sind eingetragene Warenzeichen der ARM Limited. ARM7™ ist ein eingetragenes Warenzeichen der ARM Limited.

Hersteller versucht, die Entwicklung von Software für ihre Prozessoren als einfacher und schneller realisierbar zu vermarkten als bei Mitbewerbern. Dafür werden umfangreiche Pakete geschnürt (manchmal Ecosystems oder Ökosysteme genannt), in denen IDEs, Tool Sets sowie Middlewarelibraries und interaktive Konfigurationstools miteinander eine möglichst schnelle, nahtlose und unproblematische Softwareentwicklung ermöglichen sollen. Auf diesen Zug springen auch die Toolhersteller auf. Nischenprodukte ohne Bündelung in einem dieser Pakete haben immer mehr Probleme, sich auf dem Markt zu behaupten.

Da in diesem Buch Cortex®-basierte Architekturen im Allgemeinen betrachtet und spezielle Prozessoren lediglich zu Illustrationszwecken herangezogen werden, habe ich bei der Wahl der betrachteten Softwarepakete bewusst davon Abstand genommen, ein gegebenes Ökosystem heranzuziehen. Das Buch fokussiert auf folgenden Hard- und Softwarekomponenten:

- Als Hardwarebasis dienen auf Cortex® M3 und M4-basierende Prozessoren (im Folgenden kollektiv als ACP bezeichnet, siehe Abschn. 2.1), da diese am Markt den unumgänglichen Bezugspunkt bilden. Die meisten der diskutierten Konzepte sind auch unverändert für die nächstgrößeren M7 und auf späteren Releases basierende Kerne anwendbar.
- Als Boards wurden fertige Evaluation Boards von Herstellern herangezogen, die sich recht preisgünstig über den Elektronikfachhandel beschaffen lassen und zuweilen bei Fachtagungen oder Messen vergünstigt erhältlich sind: Das STM32F407 Discovery Board, das STM32F429 Evaluation Board und das Kinetis FRDM-K64F Board. Die meisten dieser Boards verfügen darüber hinaus bereits über eingebaute Debug Probes, so dass dort die Entwicklung ohne zusätzlich Hardware vorgenommen werden kann.
- Die Prozessorinitialisierungsroutinen wurden, wenn immer möglich, über die von ARM Limited definierte CMSIS-Schnittstelle (Abschn. 2.1) implementiert, um eine Vergleichbarkeit zwischen Prozessoren verschiedener Hersteller mit möglichst wenig neuem Code zu ermöglichen. Board Support Packages (Abschn. 1.3) können zwar hilfreiche Dienste bieten, ziehen aber in der Regel recht umfangreiche Anpassungen nach sich.
- Als Betriebssystem (Kap. 3) wurde FreeRTOS der Firma Real Time Engineers Limited eingesetzt, da es sich in der Open Source Domain befindet, unabhängig davon aber sehr stabil und feldtauglich ist.
- Zur Demonstration der Hostkommunikation (Abschn. 7.2) wurde die ebenfalls sehr verbreitete, auch unter entwicklungs- und autorenfreundlicher Lizenz stehende LWIP (Lightweight IP) gewählt.
- Als IDE wurde die ebenfalls frei downloadbare Version WinIdeaOpen der Firma iSystem gewählt, die unter anderem den Vorteil hat, keine Kenntnisse von Makefiles und anderen zum Teil sehr komplexen und idiosynkratischen Entwicklungskonzepten vorauszusetzen.
- WinIdeaOpen unterstützt bereits den Open Source GCC Compiler für Cortex®-Prozessoren, der automatisch mit WinIdeaOpen installiert wird.

Obwohl es absolut möglich ist, mit diesem oder einem vergleichbaren kostenneutralen Werkzeugsatz auch feldtaugliche Software für eingebettete Systeme herzustellen, sollten Sie als Leser nicht die Tatsache aus dem Auge verlieren, dass es auf dem Markt eine Vielzahl von konkurrierenden und sich ergänzenden Produkten gibt, von denen auch die kommerziell vertriebenen Pakete ihren Sinn haben. Entwickler werden sich in jedem Projekteinzelfall ihre Werkzeug- und Materialkiste selber zusammenstellen. Die Wahl der oben genannten Produkte impliziert nicht, dass es keine gleichwertigen Mitbewerberprodukte gäbe.

Um den Rahmen dieses Buches zu setzen, seien kurz die Dinge erwähnt, die Sie hier vergeblich suchen werden:

- Benutzerinterfaces haben wie bereits angesprochen in industriellen Controllern bis auf Randanwendungen eine eher untergeordnete Bedeutung. Es werden also keine Texte auf Displays ausgegeben, Tastendrücke abgefragt oder Mäuse angeschlossen. Für Leser, die sich für diese Art der interaktiven Anwendungen interessieren, sei auf die reichlich vorhandene Literatur zu diesen Themen verwiesen.
- Das nicht uninteressante Thema der Stromsparmodi wird in diesem Buch nur am Rande abgedeckt, obwohl es auch im Rahmen von industriellen Controllern Anwendungsmöglichkeiten dafür gibt.
- Da der Fokus auf dem liegt, was allen auf ACPs gemeinsam ist, werden die Spezifika – also prozessorspezifische Peripherien – nur dort tiefer diskutiert, wo sie zum Verständnis der jeweiligen Kapitel relevant sind. Es gibt auf dem Markt eine Vielzahl von Büchern, die die Peripherie von speziellen ACPs in jeder beliebigen Tiefe zum Inhalt haben. Beispielhaft sei hier Geoffrey Brown's „Discovering the STM32 Microcontroller" genannt, in dem von der Inbetriebnahme eines Discovery Boards bis hin zum Durchsteppen von Einzelanweisungen in einer IDE jeder Schritt genau bebildert beschrieben wird (http://www.cs.indiana.edu/~geobrown/book.pdf. Zugegriffen am 13.07.2016).

Bei dem Entstehen dieses Buches haben mir die Korrekturleser und -leserinnen Ralf Borchers, Robert Friedrich, Kathrin Golze, Dr. Sabine Kathke und Hans Wannenmacher unschätzbare Hilfestellung gegeben, für die ich an dieser Stelle meinen herzlichen Dank aussprechen möchte. Auch die Unterstützung durch die Firmen ARM Limited,[2] iSystem, Percepio AG, Real Time Engineers Ltd. sowie ST Microelectronics darf nicht unerwähnt bleiben.

Historisch bedingt setzt sich das Vokabular der Informationstechnologie vorrangig aus englischen Ausdrücken zusammen, was dazu führt, dass auch im alltäglichen Sprachgebrauch in den Gängen der Entwickleretagen ein zuweilen als „Denglisch" bezeichnetes Konglomerat aus deutschen und englischen Wörtern und Satzkonstruktionen vorherrscht. Buchautoren stehen dabei immer in einer Zwickmühle, denn in der geschriebenen Sprache wirken diese Mixturen noch umständlicher und künstlicher als in der gesprochenen

[2] Dieses Buch ist von der ARM Limited nicht autorisiert, gesponsort oder genehmigt.

Sprache. Die Alternative – zu versuchen, die etablierten Begriffe um jedem Preis einzudeutschen – wäre allerdings für die Praxis (um die es in diesem Buch geht) kontraproduktiv. Deswegen finden sich in diesem Buch Wortgebilde wie „gedebuggt“ oder „durchsteppen“, die bei Sprachpuristen Stirnrunzeln hervorrufen, aber dafür die Entwicklungsrealität besser widerspiegeln.

Ich bitte zu guter Letzt alle Leserinnen darum, es mir nachzusehen, wenn ich im Sprachgebrauch dieses Buches in der Regel verallgemeinernd die männliche Form „der Entwickler“ benutze, anstatt die (leider immer noch zahlenmäßig viel zu kleine) Gruppe der Entwicklerinnen einzubeziehen. Damit ist in keinster Form eine Respektlosigkeit gegenüber den Kolleginnen unter uns impliziert, sondern lediglich die Realität im Berufsbild Embedded Softwareentwicklung zu Gunsten eines kompakteren Schreibstiles abgebildet.

Abkürzungsverzeichnis

An dieser Stelle werden alphabetisch sämtliche Abkürzungen aufgelistet und erklärt, die im Buchtext vorkommen. Sie sind ausdrücklich nicht als komplette Erklärungen der verwendeten Abkürzungen gedacht, sondern als Orientierungs- und Einordnungshilfe. Die Zahl in Klammern zwischen jedem Eintrag und seiner Erklärung bezieht sich auf die Kapitel, in denen die jeweilige Abkürzung inhaltlich behandelt werden.

AAPCS	(Kap. 2), Procedure Call Standard for the ARM® Architecture. ARM Limited proprietär. ABI Spezifikation für ARM® Cortex®-basierte Prozessoren.
ABI	(Kap. 2), Application Binary Interface. Software Engineering. Schnittstellenspezifikation zwischen Programmaufrufen auf Maschinensprachebene.
ACP	(Kap. 2), ARM® Cortex® based Processor. Vom Autor eingeführte Terminologie.
ALU	(Kap. 1), Arithmetic Logic Unit. Elektrotechnik. Die Komponente in einem Prozessorkern, die nicht-Fließpunktoperationen zwischen Operanden ausführt.
API	(Kap. 1), Application Programming Interface. Software Engineering. Schnittstellenspezifikation zwischen Programmaufrufen auf Hochsprachenebene.
ARM®	(Kap. 2), Advanced RISC Machines. ARM Limited proprietär. Von ARM Limited entwickelte Prozessorarchitektur.
ARP	(Kap. 7), Address Resolution Protocol. Netzwerkkommunikation. Protokollspezifikation zur Zuordnung zwischen MAC- und IP-Adressen.
BSD	(Kap. 7), Berkeley Software Distribution. Eine Open Source Unix-Distributionsvariante. Als eine der ersten freien Distributionen dient BSD oft als Namensgeber für Technologien wie das Socket API für Zugriff auf einen TCP/IP-Netzwerkstack.
BSP	(Kap. 1), Board Support Package. Software Engineering. Vorgefertigtes Softwarepaket zur Ansteuerung einer zugehörigen Hardware. Umfasst im Wesentlichen Treiber und Prozessorinitialisierungsroutinen.
C/C++	(Kap. 1, 2 und 10), Programmiersprachen.
CAN	(Kap. 7), Controller Area Network. Buskommunikation. Vor Allem im Bereich Automotive oft anzutreffende Busarchitektur.

CCM	(Kap. 2), Core Coupled memory. Elektrotechnik. Eine Form von on-Chip realisierter RAM, der sehr eng an den Prozessorkerngebunden ist.
CMSIS	(Kap. 2), Cortex Microcontroller Software interface Standard. ARM Limited proprietär. API zum Ansprechen von Prozessorperipherie.
CPU	(Kap. 1), Central Processing Unit. Elektrotechnik. Siehe Diskussion Kap. 1.
CRC	(Kap. 7), Cyclic Redundancy Check. Software Engineering. Oft synonym mit „Checksumme“ verwendet. Mechanismus zum Erkennen von Fehlern bei einer Datenkommunikation.
CS	(Kap. 1 und 2), Chip Select. Elektrotechnik. Siehe Abschn. 1.3.1
DCF-77	(Kap. 5), Protokoll zur Uhrzeitsynchronisation über Langwelle.
DHCP	(Kap. 7), Dynamic Host Configuration Protocol. Netzwerkkommunikation. Mit DHCP (basierend auf dem BOOTP Protokoll) werden IP-Adressen in einem Netzwerk dynamisch verhandelt.
DMA	(Kap. 1 und 2), Direct Memory Access. Elektrotechnik. Technologie zur Entlastung eines Prozessors. Mit DMA können Subsysteme ohne Intervention des Hauptprozessors direkt auf Speicher zugreifen.
DNS	(Kap. 7), Dynamic Name Services. Netzwerkkommunikation. Infrastruktur zur Auflösung von Domain-Namen in IP-Adressen.
DWT	(Kap. 2 und 10), Data Watchpoint Trace. ARM Limited proprietär. Siehe Abschn. 10.1.3
EDOS	(Kap. 3), Embedded Version of Desktop Operating System. Vom Autor eingeführte Terminologie.
ETM	(Kap. 2 und 10), Embedded Trace Macrocell. ARM Limited proprietär. Siehe Abschn. 10.1.6
FBP	(Kap. 2 und 10), Flash Breakpoint Patch. ARM Limited proprietär. Siehe Abschn. 10.1.2
FIFO	(Kap. 4 und 6), First In First Out. Software Engineering. Strategie zum Zugriff auf eine Warteschlange.
FMC	(Kap. 2), Flex Memory Controller. Proprietär.
FPU	(Kap. 1 und 2), Floating Point Unit. Elektrotechnik. Subsystem zur Realisierung von Fließkommaarithmetik auf Prozessorebene.
FSM	(Kap. 4 und 7), Finite State Automaton. Informatik. Siehe Abschn. 4.1.1
FTP	(Kap. 7), File Transfer Protocol. Netzwerkkommunikation. Protokoll zum Zugriff auf das Dateisystem eines anderen Computers.
FVP	(Kap. 2), Fixed Virtual Platform. ARM Limited proprietär. Softwaresimulator für ARM® Cortex®-Prozessorkerne
GCC	(Kap. 1 und 10), GNU C Compiler.
GIT	(Kap. 1), Open Source-Versionskontrollsoftware.
GNU	(Kap. 1 und 10), Gnu’s Not Unix.
GPS	(Kap. 1 und 5), Global Positioning System. GPS Coprozessoren werden in der Regel über eine serielle Schnittstelle an einen Prozessor angebunden und kommunizieren die Positionsdaten über das NMEA-Protokoll an den Prozessor.

GUI (Kap. 1), Graphical User Interface. Software Engineering.

HAL (Kap. 1, 2, und 4), Hardware Abstraction Layer. Software Engineering. Logische Abbildung verschiedener Prozessoren oder Prozessorarchitekturen auf eine abstrakte Schnittstelle.

HMI (Kap. 1), Human-Machine Interface. Software Engineering.

HTML (Kap. 7), Hypertext Markup Language. Netzwerkkommunikation. Definition der mit HTTP übertragenen Dateninhalte.

HTTP (Kap. 7), Hypertext Transfer Protocol. Netzwerkkommunikation. Basisprotokoll des www auf ISO/OSI Schichten 5–7.

I2C (Kap. 2 und 7), Inter-Integrated Circuit. Buskommunikation. ISO/OSI Schicht 1 Protokoll zur Kommunikation zwischen Prozessoren und räumlich nah angebundenen Bausteinen.

ICMP (Kap. 7), Internet Control Message Protocol. Netzwerkkommunikation. Auf IP aufsetzendes Protokoll zum Austausch von Quality-of-Service-Daten zwischen Computern. ICMP manifestiert sich unter anderem in dem „ping“-Dienstprogramm.

IDE (Kap. 1 und 10), Integrated Development Environment. Software Engineering.

IP (Kap. 7), Internet Protocol. Netzwerkkommunikation. Basisprotokoll des Internets auf Schicht 3.

IPv4 (Kap. 7), Internet Protocol version 4. Netzwerkkommunikation. Urversion von IP. Unterstützt 32 Bit große IP-Adressen.

IPv6 (Kap. 7), Internet Protocol version 6. Netzwerkkommunikation. Erweiterung von IPv4 mit Unterstützung von bis zu 128 Bit großen IP-Adressen.

IRQ (Kap. 1, 2, 3, 4 und 6), Interrupt Request. Elektrotechnik. Ein Ereignis, das bei einem Prozessor eine Unterbrechung des momentanen Programmablaufes erzwingt. Siehe Abschn. 1.5

ISR (Kap. 1, 2, 3, 4 und 6), Interrupt Service Routine. Software Engineering. Mit einem Prozessor registrierter Programmcode zum Aufruf durch einen IRQ. Ein synonymer Begriff ist „IRQ Handler.“ Siehe Abschn. 1.5

ITM (Kap. 2 und 10), Instruction Trace Macrocell. ARM Limited proprietär. Siehe Abschn. 10.1.5

IVT (Kap. 1, 2, 3, 4, 6 und 9), Interrupt Vector Table. Software Engineering. Tabelle registrierter ISRs. Siehe Abschn. 1.5

IWDG (Kap. 8), Internal Watchdog. Proprietär. In vielen ACPs realisiertes Subsystem zur Implementation von Watchdogs (siehe Kap. 8).

JTAG (Kap. 1, 2, und 10), Joint Test Action Group. Proprietär. Wird in der Regel zur Bezeichnung von Debugschnittstellen benutzt (Abschn. 10.1.1)

LCF (Kap. 1, 2, und 9), Linker Command File. Software Engineering. Siehe Abschn. 1.4 und 9.5.1

LED (Kap. 1), Light Emitting Diode. Elektrotechnik.

LIFO (Kap. 4 und 6), Last In First Out. Software Engineering. Strategie zum Zugriff auf eine Warteschlange.

LSB (Kap. 2 und 7), Least Significant Bit. Elektrotechnik.

LTE (Kap. 7), Long Term Evolution. Netzwerkkommunikation.

LWIP (Kap. 7), Lightweight IP. Netzwerkkommunikation. Open Source-Implementation der TCP/IP-Protokollsuite. Siehe Abschn. 7.9.3

MAC (Kap. 1 und 7), Media Access Control. Elektrotechnik. Bezeichnet die die ISO/OSI Schicht 2 realiserenden Hardwarekomponenten.

MCB (Kap. 9), Module Control Block. Vom Autor eingeführte Terminologie.

MIB (Kap. 7), Managed Information Base. Netzwerkkommunikation. Logische Repräsentation einer über SNMP verwalteten Datenbank.

MII (Kap. 7), Media Independent Interface. Netzwerkkommunikation. Protokollspezifikation zur Anbindung von MAC an PHY Komponenten.

MIT (Kap. 9), Module Interface Table. Vom Autor eingeführte Terminologie.

MMU (Kap. 1 und 3), Memory Management Unit. Elektrotechnik. In einem Prozessor realisiertes Subsystem zur strukturierten Verwaltung von RAM. Bietet Hardwareunterstützung für in Betriebssystemen mögliche Implementation von virtuellen Speicherbereichen.

MPU (Kap. 2, 3 und 4), Memory Protection Unit. Elektrotechnik. In einem Prozessor realisiertes Subsystem zum Schutz von Speicherbereichen gegen unautorisierten Zugriff.

MQTT (Kap. 7), Message Queue Telemetry Transport. Netzwerkkommunikation. Open Source Message Broker Protokoll (Abschn. 7.5)

MSB (Kap. 2 und 7), Most Significant Bit. Elektrotechnik.

NAT (Kap. 7), Network Address Translation. Netzwerkkommunikation. Protokoll zur Abbildung von lokalen und globalen IP-Adressbereichen untereinander.

NMI (Kap. 1, 2 und 6), Non Maskable Interrupt. Elektrotechnik. IRQ, der nicht durch Software unterbunden werden kann.

NTP (Kap. 5), Network Time Protocol. Netzwerkkommunikation. Protokoll zur Zeitsynchronisation über UDP.

NVIC (Kap. 2, 3 und 10), Nested Vectored Interrupt Controller. ARM Limited proprietär. In jedem ACP vorhandenes Subsystem zur Arbitrierung und Verarbeitung von IRQs.

PC (Kap. 1), Personal Computer.

PHY (Kap. 2 und 7), Physical Communication Interface. Elektrotechnik. Bezeichnet die die ISO/OSI Schicht 1 realiserenden Hardwarekomponenten. Werden oft als diskrete Bauelemente an einen Prozessor angeschlossen.

PLL (Kap. 1 und 5), Phase-Locked Loop. Elektrotechnik. Siehe Abschn. 5.1.1

PPB (Kap. 2 und 10), Private Peripheral Bus. ARM Limited proprietär. Siehe Abschn. 2.2

PPP (Kap. 7), Point-to-Point Protocol. Netzwerkkommunikation. Protokoll zur Verwaltung einer logischen Verbindung über eine serielle Schnittstelle. Für den Zweck dieses Buches interessant als häufig genutztes Schicht 2-Protokoll zwischen Prozessoren und externen Modems.

RAM (Kap. 1, 2 und 4), Random Access Memory. Elektrotechnik.
RISC (Kap. 1 und 2), Reduced Instruction Set Computer. Elektrotechnik.
RMII (Kap. 7), Reduced Media Independent Interface. Netzwerkkommunikation. Leitungsreduzierte Variante des MII.
ROM (Kap. 1 und 2), Read-Only Memory. Elektrotechnik.
RS232 (Kap. 7), Kommunikationsstandard für serielle Schnittstellen. Buskommunikation.
RS485 (Kap. 7), Kommunikationsstandard für serielle Schnittstellen. Buskommunikation.
RTC (Kap. 5), Real-Time Clock. Elektrotechnik. Siehe Abschn. 5.4
RTOS (Kap. 3), Real-Time Operating System. Software Engineering. Siehe Abschn. 3.3
RWL (Kap. 6), Reader-Writer-Lock. Software Engineering. Siehe Abschn. 6.2.5
SCB (Kap. 2), System Control Block. ARM Limited proprietär. In jedem ACP vorhandenes Subsystem zum Ansprechen des Prozessorkerns.
SD (Kap. 4), Secure Digital Memory Card.
SDRAM (Kap. 1, 2 und 4), Synchronous Dynamic Random Access Memory. Elektrotechnik.
SIMD (Kap. 2), Single Instruction, Multiple Data. ARM Limited proprietär. In ARM® Cortex® M4 implementierte Untermenge des ARM® Thumb®-Befehlssatzes zur gleichzeitigen Ausführung gleichartiger Operationen in einem Befehl.
SNMP (Kap. 7), Simple Network Management Protocol. Netzwerkkommunikation. Protokoll zur Überwachung und Fernsteuerung von Computersystemen.
SoC (Kap. 1 und 2), System On Chip. Elektrotechnik. Siehe Kap. 1
SPI (Kap. 7), Serial Peripheral Interface. Buskommunikation. Busfähiges ISO/OSI Schicht 1 Protokoll zur Kommunikation zwischen Prozessoren und räumlich nah angebundenen Bausteinen.
SQL (Kap. 4), Structured Query Language. Software Engineering. Beschreibungssprache für relationale Datenbanken.
SRAM (Kap. 1, 2 und 4), Static random-access memory. Elektrotechnik.
SSL (Kap. 7), Secure Socket Layer. Netzwerkkommunikation. Protokollfamilie zur kryptographischen Erweiterung von auf IP basierenden Protokollen.
SVN (Kap. 1), Apache Subversion. Proprietär.
SWD (Kap. 2 und 10), Single Wire Debug. ARM Limited proprietär. Siehe Abschn. 10.1.1
TCB (Kap. 3), Task Control Block. Betriebssystemtechnologie. Datenstruktur zur Verwaltung von Tasks in Betriebssystemen.
TCP (Kap. 7), Transmission Control Protocol. Netzwerkkommunikation. Verbindungsorientiertes und verlässliches Basisprotokoll des Internets auf ISO/OSI Schicht 4.
TLV (Kap. 7), Tag-Length-Value. Software Engineering. Strukturierte Repräsentationsform von Datenströmen.

TPIU	(Kap. 2 und 10), Trace Port Interface Unit. ARM Limited proprietär.
TTL	(Kap. 1 und 7), Transistor-Transistor-Logic. Elektrotechnik.
UART	(Kap. 4, 6 und 7), Universal Asynchronous Receiver Transmitter. In einem Prozessor realisiertes Subsystem zur Übertragung von Daten auf einem asynchronen seriellen Interface.
UDP	(Kap. 7), User Datagram Protocol. Netzwerkkommunikation. Verbindungsloses und nicht verlässliches Basisprotokoll des Internets auf ISO/OSI Schicht 4.
UML	(Kap. 4), Unified Modeling Language. Software Engineering.
USB	(Kap. 7), Universal Serial Bus. Buskommunikation.
VPN	(Kap. 7), Virtual Private Network. Netzwerkkommunikation.
XML	(Kap. 7), Extensible Markup Language. Netzwerkkommunikation.

Inhaltsverzeichnis

1 Grundlagen 1
- 1.1 Wie wird für Embedded Controller programmiert? 3
- 1.2 Die Werkzeugkiste 4
- 1.3 Zielhardware 6
  - 1.3.1 Chip Selects 8
- 1.4 Bootsequenzen 10
- 1.5 Nebenläufigkeit und Interrupts 12
  - 1.5.1 Faults 12
- 1.6 Der Buildzyklus 14
- 1.7 Der Alltag 15
- 1.8 Low Power 18

Literatur 19

2 Die ARM® Cortex® M3- und M4-Kerne 21
- 2.1 Geschichte und Marktstellung 21
  - 2.1.1 Ökosysteme und Abstraktionslayer 23
- 2.2 Wie setzt sich ein auf ARM® Cortex®-basierender Prozessor zusammen? 25
- 2.3 Adressbereichslayout 28
- 2.4 Bootsequenz 30
- 2.5 Busse und Bus Matrizen 31
- 2.6 Interrupts im Cortex®-Kern 36
- 2.7 Code und Codeoptimierungen 39
  - 2.7.1 Funktionsaufrufe 40
  - 2.7.2 Codeinlining 43
  - 2.7.3 Praxisvergleich 44
  - 2.7.4 Entfalten von Code 46
  - 2.7.5 Weiteres Optimierungspotential 48
  - 2.7.6 Optimierungen und der Debugger 50
  - 2.7.7 Fazit 50

2.8 Endianness ... 51
2.9 Bit Banding ... 53
Literatur ... 54

3 Betriebssysteme ... 57
3.1 Das große Bild ... 57
3.2 Wann brauche ich ein Betriebssystem? Und warum? ... 60
3.3 Welche Betriebssysteme stehen mir zur Verfügung? ... 61
3.4 Wie funktioniert ein Echtzeitbetriebssystem? ... 63
3.4.1 Prioritäten ... 67
3.4.2 Context Switches ... 68
3.4.3 Speicherverwaltung ... 73
3.5 Wie wird FreeRTOS in ein bestehendes Projekt integriert? ... 75
3.6 Migration einer Standalone-Applikation auf FreeRTOS ... 76
3.6.1 Empirischer Footprint ... 81
3.7 Was müssen wir nach dem Umstieg beachten? ... 84
3.7.1 Callback-Funktionen ... 85
3.7.2 Reentrancy ... 86
Literatur ... 87

4 Systemdesign ... 89
4.1 Identifikation und Benutzung nebenläufiger Stränge ... 90
4.1.1 Zustandsautomaten ... 95
4.1.2 Prioritätenverteilung ... 100
4.1.3 Kontrollfluss in Gerätetreibern ... 102
4.2 Speicherarchitekturen ... 107
Literatur ... 109

5 Zeit ... 111
5.1 Zeit auf Prozessorebene ... 111
5.1.1 Clock Sources und Clock Trees ... 112
5.2 Echtzeitbetriebssysteme und Zeit ... 114
5.2.1 Was macht das RTOS mit „seiner“ Zeit? ... 116
5.2.2 Vom RTOS bereitgestellte Timerservices ... 121
5.3 Zeit auf Applikationsebene ... 122
5.3.1 Synchroner, in Funktionsaufrufen implementierter Timeout ... 123
5.3.2 Asynchroner applikationsgesteuerter Timeout ... 123
5.3.3 Synchroner applikationsgesteuerter Timeout ... 124
5.3.4 Timeoutintervalle konfigurieren und verwalten ... 125
5.4 Absolute Zeit ... 126
5.4.1 Zeitformate ... 127
Literatur ... 128

6 Synchronisation 129
6.1 Racekonditionen und deren Folgen 131
6.1.1 Deadlock 131
6.1.2 Starvation 132
6.1.3 Ungewünschte Takserialisierung 133
6.1.4 Convoy Effect 136
6.2 Synchronisationsobjekte und -kontrollstrukturen 137
6.2.1 Queues und von Queues abgeleitete Objekte 137
6.2.2 Mutexobjekte 139
6.2.3 Message Pumps 139
6.2.4 Die Critical Section 143
6.2.5 Zusammengesetzte Objekte 143
6.3 Synchronisation mit Interrupt Handlern und versteckten Handlungssträngen 156
6.3.1 Synchronisation zwischen ISRs und Applikationscode 156
6.3.2 Barrieren 160
6.3.3 Nebenläufigkeit mit DMA 164
Literatur 167

7 (Host-) Kommunikation 169
7.1 Physikalische Anbindung 170
7.2 Netzwerksoftware 173
7.3 Applikationsprotokoll 173
7.4 Einwahlrichtung/Infrastruktur 175
7.4.1 Grundlagen 175
7.4.2 Embedded System als Kommunikationsclient 177
7.4.3 Embedded System als Kommunikationsserver 178
7.4.4 Verbindungsdauer 180
7.5 Message Broker 181
7.6 Stabilität 183
7.6.1 Broadcaststürme und Systemauslastung 184
7.7 Sicherheit 186
7.8 Protokollempfehlungen 188
7.8.1 Sonderbetrachtung für serielle Protokolle 192
7.8.2 Portable Protokollspezifikationen 193
7.9 Praxis 194
7.9.1 Ein Beispielprotokoll 195
7.9.2 Protokollspezifikation 196
7.9.3 Implementationsstrategie 202
7.10 Wartungsinterfaces 205
Literatur 207

8 Watchdogs 209
8.1 Ausprägungen von Watchdogs 210
8.2 Einen Watchdog nachtriggern 212
8.3 Wahl des Ablaufintervalls 213
8.4 Software-Watchdogs 213
8.5 Benutzung von Watchdogs zum kontrollierten Herunterfahren 216
8.6 Grenzen und Probleme von Watchdogarchitekturen 217

9 Bootloader 219
9.1 Bootloaderarchitekturen 221
9.2 Wie wird die zu ladende Software an das Gerät kommuniziert? 225
9.3 Wo wird die Firmware zwischengespeichert und falls nötig auf Gültigkeit geprüft? 226
9.4 Wie wird die heruntergeladene Firmware in den Programmspeicher abgelegt? 227
9.5 Kochbuch zum Bootloaderdesign 228
9.5.1 Layout 229
9.5.2 Kontrollfluss des Bootloaders 237
9.5.3 Kontrollfluss der Applikationssoftware 238
9.5.4 Downloadvorgang 240
9.5.5 Zu berücksichtigende Testszenarios 240
Literatur 241

10 Praxistipps 243
10.1 Debugging 243
10.1.1 Tool Chains und Debugarchitekturen 243
10.1.2 Breakpoints und angehaltene Prozessoren 245
10.1.3 Die DWT 247
10.1.4 Silent Monitoring 251
10.1.5 Streaminganalysen 254
10.1.6 Tracing 258
10.2 Stackanalyse 258
10.3 Heapanalyse 266
10.4 C++ 269
10.4.1 Schritt1: C++ ohne C++ 269
10.4.2 Schritt 2: Einfügen von C++ Code 270
10.4.3 Kostenrechnung 273
10.4.4 Beispiele von in eingebetteten Systemen nützlichen C++ Einkapselungen 274
10.5 Fazit und Ausklang 276
Literatur 277

Anhang 1: Beispielapplikationen 279
Werkzeuge und ihre Nutzung 279
Beispiel Kapitel 2 281
Beispiele Kapitel 3 282
Beispiele Kapitel 4 284
Beispiele Kapitel 5 284
Beispiele Kapitel 6 284
Beispiel Kapitel 7 286
Beispiele Kapitel 8 286
Beispiel Kapitel 9 286
Beispiel Kapitel 10 287

Stichwortverzeichnis 289