

IoT Projekte mit Mynewt OS

Band 1: Calliope Mini als Einstieg

Programmieren mit C und CLion

Von Alfred Schilken

1. Auflage, 2017

ISBN-13: 978-1976208003

ISBN-10: 1976208009

© 2017 Alfred Schilken.

Alle Rechte vorbehalten.

Erschienen im Eigenverlag

Bergstr.7, 63486 Bruchköbel

alfred@schilken.de

Printed in Germany by Amazon Distribution GmbH, Leipzig

Folgende Bände sind in Planung:

Band 2: BBC micro:bit – Programmieren mit C und CLion

Band 3: ARM-Boards mit nRF51 & nRF52 – Programmieren mit C und CLion

Band 4: Fernwartung mit Newtmgr – LoRaWan, Iotivity, MQTT

Version 1.0, letzte Änderung: 30.10.2017, 358 Seiten / 82718 Wörter / 528752 Zeichen

Icons von <https://icons8.com>

Gesetzt in der Tex Gyre Pagella und URWClassico

Editiert mit Papyrus Autor 8.53 auf MacBook Pro.

Alle Fotos aufgenommen mit einem iPhone 6s.

Bildbearbeitung in Photoshop Elements 13.

Alle Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet und auf einem Caliope Mini 1.0 am MacBook Pro reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Autor kann für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen. Für die Mitteilung eventueller Fehler ist der Autor dankbar.

Alle Produktbezeichnungen, Firmennamen und Firmenlogos werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind wahrscheinlich eingetragene Warenzeichen.

Über das Buch:

Lesen, schreiben, rechnen, coden – das lernen Kinder mit dem Calliope Mini jetzt in der Grundschule. Dieses Buch ist für **Fortgeschrittene**. Es zeigt, wie mit einer professionellen Entwicklungsumgebung und einem leistungsfähigen Mikrocontroller-Betriebssystem komplexe IoT-Projekte entwickelt werden können.

Anhand praktischer Projekte für den Calliope Mini, erklärt es, wie das Mynewt Betriebssystem grundsätzlich aufgebaut ist und wie es in C programmiert wird. Die Entwicklungsumgebung CLion erlaubt das sehr komfortable Editieren und Testen des C-Codes. Die Programme können direkt auf dem Calliope im Einzelschritt-Modus ausgeführt werden. Dieses „Debuggen“ führt zu einem tieferen Verständnis des Programmablaufs und hilft bei der Fehlersuche. Die CLion IDE ist für Schüler und Studenten kostenlos.

Die Projekte im Detail:

Die App **mydrivertest** ermöglicht das Testen mehrerer Treiber mithilfe der Mynewt-Shell:

- Eingabe von Kommandos im Terminal-Fenster des angeschlossenen Rechners
- Ausgabe von Laufschrift oder einzelnen Zeichen auf die LED-Matrix
- Reaktion auf die Buttons
- Setzen der Farbe der RGB-LED und Erzeugung von Tönen
- Messen von Spannungswerten an der Grove-Buchse
- Testen diverser I2C-Module, wie OLED-Display, GPIO¹-Expander, ...

Die App **iBeacon** implementiert einen konfigurierbaren Beacon:

- Wahlweise iBeacon oder Eddystone-Beacon
- Major und Minor des iBeacon sind einstellbar
- Url des Eddystone ist konfigurierbar

Die App **bleadc** publiziert Messwerte über einen Bluetooth Service.

Die App **ble_uart** ist über Bluetooth fernsteuerbar. Die Bluefruit-App von Adafruit dient zum Senden von Kommandos und zeigt Messwerte als Grafik an, beispielsweise die Signale eines Herzschlagsensors.

Die App **blemib** zeigt die Möglichkeiten der Bluetooth-Central-Rolle, sie bringt ein MI-Band zum Vibrieren.

¹ Abkürzungen und Fachbegriffe sind im Glossar erklärt.

Über Mynewt und Calliope Mini:

Mynewt ist das erste Projekt der Apache Software Foundation für das Internet of Things (IoT). Unter dem Motto *"An OS to build, deploy and securely manage billions of devices"* bietet es ein vollständiges Betriebssystem für 32-Bit-Mikrocontroller.

- Es unterstützt ARM Cortex M0-M4, MIPS, RISC-V
- Es läuft ab 16 kB RAM und 256 kB Flash
- Es unterstützt Calliope Mini, BBC micro:bit, Arduino Zero und viele weitere nRF51-, nRF52-, STM- und Nucleo-Boards
- Es bietet den ersten Open-Source Bluetooth Stack
- Es implementiert diverse Protokolle für eingeschränkte Geräte
- Es erlaubt verschlüsselte Verbindungen
- Es ermöglicht die Fernwartung mit einem Management-Tool
- Es kann Firmware fehlertolerant und sicher aus der Ferne aktualisieren
- Logging- und Statistikfunktionen sind bereits eingebaut
- Seine Konfigurierbarkeit bis in kleinste Details ermöglicht maßgeschneiderte Anwendungen mit minimalem Speicherbedarf

Calliope Mini ist die deutsche Antwort auf den britischen BBC micro:bit. Die kleine Mikrocontroller-Platine ist zwar ursprünglich für Grundschüler gedacht, sie bietet aber mit ihren eingebauten Sensoren, den Grove-Buchsen und Motor-Treibern vielfältige Möglichkeiten für Maker und Elektronik-Interessierte aller Altersstufen. Das eingebaute Bluetooth erlaubt die Fernbedienung aller dieser Möglichkeiten mit dem Smartphone. Wenn ein 8-Bit-Arduino nicht mehr reicht, dann ist ein Calliope Mini eine gute Wahl.

Über den Autor:



Alfred Schilken begeisterte sich schon in den 80ern für die damals gerade aufkommenden Mikroprozessoren. Seine erste Experimentier-Platine¹ musste mit einem einzigen Kilobyte RAM und einem Kassettenrekorder als Massenspeicher auskommen – das war 1980. Es war ein Glückfall, dass seine Examensarbeit am Institut für angewandte Physik in Frankfurt ein Mikroprozessor-System zum Thema hatte.

In diesen 37 Jahren ist das Programmieren sein Hobby geblieben, allerdings arbeitet er auch seit mehr 30 Jahren als Freiberufler in Projekten bei Kunden wie der Telekom, Lufthansa Systems, Sybase, SAP

und der Deutschen Bank.

Er ist sehr früh von Assembler auf C umgestiegen, hat dann lange in C++, später in Java, Python und Swift programmiert. Mit dem Erscheinen des BBC micro:bit und dem Calliope Mini ist die Programmierung in C wieder interessant geworden.

Die Aussicht, in einem IoT-Projekt seine aufgefrischten C-Kenntnisse einsetzen zu können, motivierte ihn zum intensiven Eintauchen in die Tiefen der Embedded-Software-Entwicklung.

¹ Das war ein KIM-1 Entwicklungssystem für den 6502-Mikroprozessor von MOS-Technology

Danksagungen:

Bei dieser Gelegenheit möchte ich mich bei einigen herausragenden Personen und Organisationen bedanken. Ohne die Eigeninitiative dieser computer-begeisterten Leute wäre die Maker-Szene nicht so weit, wie sie heute ist. Mein Dank geht an:

- Massimo Banzi und David Cuartielles, die mit der Entwicklung des Arduino der Maker-Szene eine neue Richtung gegeben haben.
- **Eben Upton** für die Entwicklung des Raspberry Pi.
- Die Initiatoren des „BBC Computer Literacy Programme“, das schließlich den BBC micro:bit hervorbrachte.
- Die "**Sechs digitalen Köpfe**, die digitale Bildung an Deutschlands Schulen bringen wollen" für die Entwicklung des Calliope Mini.
- An die Mitarbeiter von **Runtime.io**, die mit der Übergabe ihres Quellcodes an die Apache Software Foundation das Mynewt Projekt ermöglicht haben.
- **Chris Wanstrath, PJ Hyett** und **Tom Preston-Werner**, die mit ihrem **GitHub**-Service der Open-Source-Gemeinde eine ideale Plattform für den Austausch ihrer Quelltexte bieten.
- An die SEGGER Microcontroller GmbH, die ihre professionellen Tools für Hobby-Anwender kostenlos zur Verfügung stellt.
- Richard Stallman, der schon 1985 die Free Software Foundation gründete. Ohne die Compiler des GNU-Projekts sind Open-Source-Projekte wie Linux, Arduino aber auch Mynewt gar nicht vorstellbar. •

Und ich danke meiner Tochter **Eva** für die Korrektur einiger Kapitel dieses Buches. Sie war schon immer gut darin, Sachen zu finden. Einige fehlende oder überflüssige Punkte, Klammern und Leerzeichen sowie ungünstige Formulieren haben es mit ihrer Hilfe nicht in diese Ausgabe geschafft.

Inhaltsverzeichnis

Vorwort.....	1
Für wen ist dieses Buch?.....	2
Was hier zu finden ist.....	3
Was wird gebraucht?.....	5
Was nicht in diesem Buch steht.....	6
Schreibstil.....	6
Quellcode.....	7
Webseiten zum Buch.....	7
Typographische Vereinbarungen.....	8
Kapitel 1 – Die Maker.....	9
Vom Mikroprozessor zum Mikrocontroller.....	9
Arduino.....	10
ARM mbed.....	11
Raspberry Pi.....	13
Internet of Things (IoT).....	13
Kapitel 2 – BBC micro:bit.....	15
Die technischen Daten.....	15
LED-Matrix und Buttons.....	15
Vergleich mit Arduino und Raspberry Pi.....	16
Kapitel 3 – Der Calliope.....	17
Die Platine.....	19
Zielgruppe: nicht nur für Grundschüler.....	20
BBC-micro:bit kompatibel – und mehr.....	21
Ein Blockschaltbild.....	22
Die RGB-LED.....	23
Die Tonausgabe – „Buzzer“.....	23
Der Motortreiber.....	24
Das Miniatur-Mikrofon.....	25
Die Grove-Buchsen.....	26

Ein externer Speicherchip (optional).....	26
Der Erweiterungsport für Maker.....	27
Kapitel 4 – Laufzeitumgebungen.....	29
Bare Metal.....	29
MikroPython.....	29
RTOS – Real-Time-Operating-System.....	30
Kapitel 5 – Entwicklungsumgebungen.....	31
Calliope mini, PXT Editor und Open Roberta Lab	31
MicroPython.....	34
Die Grove-Buchsen.....	26
Yotta – mbed offline	37
mbed-cli	44
Segger Embedded Studio – SES-IDE	44
Keil μ Vision IDE.....	46
Eclipse + GCC.....	46
Visual Studio Code.....	47
CLion + Yotta.....	47
CLion + newt.....	47
Fazit – Vergleich der Entwicklungstools.....	47
Kapitel 6 – Betriebssysteme für Calliope.....	49
ARM mbed OS.....	49
Lancaster Runtime.....	50
RIOT.....	52
Zephyr.....	53
Apache Mynewt.....	54
Fazit – Vergleich der Betriebssysteme.....	55
Kapitel 7 – Apache Mynewt.....	56
Pures C anstatt C + +	56
Projekte und Repositories.....	58
Packages.....	59
Apps.....	60

BSP – Board Support Package.....	60
HAL – Hardware Abstraction Layer.....	65
Architektur.....	67
Target = App + BSP + Profil.....	67
Bootloader.....	69
Aufteilung des Flash-Speichers.....	70
Die Flash-Map.....	71
BSP für den Single-Image-Modus.....	73
Konventionen.....	75
Treiber für Kommunikation und Sensoren.....	75
OS Tasks.....	79
OS Queues.....	80
Timer – "callout".....	81
Statistiken.....	81
Logging.....	84
Systemkonfiguration – syscfg-Parameter.....	84
Config-Variablen.....	87
Konsole.....	87
Shell.....	88
Kapitel 8 – Installation auf macOS.....	90
Installation des newt-Tools.....	90
Installation ARM Toolchain, OpenOCD und CoolTerm.....	91
Installation der „nativen“ Toolchain.....	93
CLion installieren.....	95
Kapitel 9 – Installation auf Windows 10.....	96
MinGW Shell für Windows installieren.....	96
Ausführbare Version des newt-Tools installieren.....	97
Git für Windows installieren.....	99
Installation ARM Toolchain, OpenOCD, CoolTerm.....	100
CLion installieren.....	102
Kapitel 10 – Apps bauen mit dem newt-Tool.....	103

Überblick newt – Subkommandos.....	103
Ein neues Projekt anlegen.....	104
„blinky“ bauen und auf Mac oder Linux ausführen.....	108
Debuggen eines nativen Targets.....	112
„blinky“ Bauen und auf Calliope ausführen.....	113
Kapitel 11 – CLion + newt: ein gutes Gespann.....	118
Mynewt-Projekt für CLion vorbereiten.....	118
Ein Mynewt Projekt in CLion öffnen.....	119
Projekt in CLion aktualisieren.....	122
Debuggen mit gdb und OpenOCD.....	123
Kapitel 12 – Die App mydrivertest.....	129
Ein externes Repository einbinden.....	129
Auf Buttons reagieren.....	135
Der Button-Treiber.....	137
Die Shell aktivieren.....	142
Shell-Commands.....	143
CoolTerm konfigurieren.....	144
Die Shellkommandos der App.....	146
Test des Sound-Treibers.....	149
Die GPIOs des Calliope.....	152
GPIO – den digitalen Status eines Pins einlesen.....	154
GPIO-Scanner.....	157
GPIO Ausgabelevel schalten.....	158
Der I2C Bus.....	159
I2C probe.....	160
I2C Scanner.....	160
Analog-Digital-Wandler.....	163
Zusätzliche Repositories für den ADC-Treiber.....	164
ADC testen.....	168
LED-Matrix testen.....	171
OLED-Display testen.....	173
LED-Balken testen.....	174

LED-Balken testen	174
15x7-LED-Matrix von Adafruit testen	176
Kapitel 13 – Das Seeedstudio Grove-System.....	178
Verschiedene Steckerbelegungen.....	180
Digitale GPIOs	180
Analoger Input.....	181
UART – Serielle Schnittstelle.....	181
I2C – Bus für bis zu 112 Geräte.....	182
Eigene Grove-Module löten.....	183
Grove PROTOSHIELD	183
Potenziometer am Grove-Kabel.....	184
Grove LED-Balken.....	184
LEDs direkt am Grove-Kabel	185
Kapitel 14 – Interessante Chips und Boards.....	186
I2C OLED-Display.....	187
I2C 15x7 LED-Matrix – Adafruit CharliePlex.....	189
I2C pcf8591-Board A/D und D/A-Wandler mit Sensoren.....	190
I2C MCP3008 ADC 10BIT A/D-Wandler.....	191
I2C RTC Echtzeit-Uhren	191
I2C 24FC128 128Kbit EEPROM.....	192
I2C Port-Expander mit MCP23017.....	194
Port-Expander sx1509 I2C-Breakout.....	197
Port Expander pcf8574 I2C-Breakout.....	198
Temperaturmessung mit I2C.....	199
Kapitel 15 – Projekt Lügendetektor – stresstest-App.....	200
Seedstudio Grove-Modul.....	201
ADC konfigurieren.....	201
Der Quelltext der stresstest-App.....	202
Feedback über eine Plotter-Kurve in der Arduino-IDE.....	205
Kapitel 16 – Projekt Herzschlag-Monitor.....	210
Grove Ohrclip Herzfrequenz Sensor – GRV HEART RATE3.....	210

Grove Finger-Clip Herzschlag-Sensor – GRV HEART RATE1.....	215
Pulsesensor Kickstarter Projekt von 2011.....	215
Kapitel 17 – Schaltschwelle der Digital-Eingänge bestimmen____	216
Kapitel 18 – Konfigurierbarer iBeacon_____	218
iBeacon – ein Beacon von Apple.....	218
Konfigurieren mit Systemkonfiguration in syscfg.yml.....	223
Konfigurationsvariablen in der Shell.....	225
Major und Minor in der Shell konfigurieren.....	226
Statistik-Zähler auswerten.....	233
Eigene Zähler für die Statistik.....	237
Drei Logging-Varianten.....	237
Logging auf die Konsole.....	238
Logging im Flash für die spätere Analyse.....	242
Eddystone Beacon – Googles Beacon-Variante.....	245
Hybrid-Beacon – iBeacon + Eddystone.....	246
Kapitel 19 – Die App bleadc_____	250
BLE-Grundbegriffe: Central, Peripheral, Service, Characteristic.....	250
Service und Characteristics initialisieren.....	251
Auf READ- und WRITE-Zugriffe reagieren.....	253
NOTIFY- und INDICATE-Nachrichten auslösen.....	256
Der GAP-Layer.....	258
Advertising.....	258
Auf GAP-Events reagieren.....	260
Kapitel 20 – Die App ble_uart_____	266
Ein Chat über Bluetooth.....	266
Eine Minimal-Shell über Bluetooth.....	270
Bluefruit App für iOS und Android.....	275
Kapitel 21 – MI-Band fernauslösen – blemib-App_____	278
Bluetooth-Discovery – auf Advertising-Pakete reagieren.....	280
Der GAP-Event-Handler.....	281

Verbindungskandidaten selektieren über Bluetooth-Adresse.....	284
Die Peer-Bibliothek.....	286
Scannen und Speichern der GATT-Details eines Peripherals.....	287
Schreibzugriff auf ein Characteristic des Peripherals.....	289
Logging von Connect, Discovery und write.....	291
Kapitel 22 – Das newtmgr-Tool.....	293
Kapitel 23 – Troubleshooting.....	297
Prüfen, ob das Programm mit einer Reboot-Loop crasht.....	297
Quelltext einer Assert-Adresse mit gdb analysieren.....	298
Absturz-Adresse in Map-Datei suchen.....	298
Keine Reaktion in CoolTerm trotz aktivierter Shell.....	299
Reboot stoppt nicht bei Breakpoint.....	299
Breakpoints funktionieren nicht.....	301
Selten – aber möglich: USB-Kabel defekt.....	301
Laufzeit-Daten des OS-Moduls in der Shell.....	309
Kapitel 24 – Ausblick.....	303
Weitere Projekte mit Grove-Modulen.....	303
Weitere Bluetooth-Projekte.....	303
Es muss nicht immer Bluetooth sein.....	304
Dateisystem für die Speicherung von Messreihen nutzen.....	304
Kapitel 25 – Anhang.....	306
Übergangslösung falls newt target cmake nicht funktioniert.....	306
newt selbst bauen.....	306
Laufzeit-Daten des OS-Moduls in der Shell.....	309
Speicherbedarf reduzieren.....	312
Linux als Entwicklungsrechner.....	313
Alle Tools prüfen.....	315
macOS GDB signieren.....	316
Die 5x5 LED-Matrix.....	317
newt vals – yml-Werte für bsp, app, target auflisten.....	318
Upgrade auf die nächste Mynewt-Version.....	320

Git für die Codeverwaltung benutzen.....	322
Grove LED-Balken.....	184
Bootloader aus dem Repository bauen und installieren.....	324
Target für Bootloader mit newt erzeugen.....	325
Source-Level-Debuggen in CLion.....	327
Funktionen, Symbole, Dateien finden.....	334
Änderungen vorangegangener Versionen wiederfinden	334
GDB Kommandos.....	335
Die wichtigsten GDB und OpenOCD Kommandos.....	337
Sublime Text Editor für die Suche.....	338
JLogicanalyzer benutzen.....	338
Meilensteine Hard- und Software.....	341
Bezugsquellen.....	342
Links ins Internet.....	345
Literaturverzeichnis	347
Glossar.....	349

Vorwort

Während ich dieses Vorwort schreibe, verfolge ich nebenbei die Twitter-Meldungen zu einem 24-Stunden-HACKATHON mit dem Thema SMART PUBLIC LIVE. Unter dem Hashtag #dthack17 beweisen Bilder und Kommentare die gute Stimmung der insgesamt 23 Teams.

„Rund 200 Hacker coden IoT-Anwendungen für smarte Städte“, so heißt es in der Pressemitteilung¹ der ubirch GmbH. Die Deutsche Telekom hatte mit diesem Motto zum Wettbewerb in ihre Telekom-Zentrale in Bonn eingeladen:

„Bringe Deine Ideen für die Stadt der Zukunft ein und nimm am Smart City Public Life Hackathon teil. Ob Entwickler, Designer, oder Newbie – wir suchen Dich!“

...

„Das erwartet euch: Coole IoT-Boards, Sensoren und Lötkolben, ..., Leute, die sich bestens mit den IoT-Boards auskennen ...“

Die Teilnehmer konnten zwar auch ihren Raspberry Pi oder Arduino mitbringen, die von der Telekom bereitgestellten Boards waren aber der Calliope Mini und ein NarrowBand-IoT-Modul². Zur Anbindung ans Internet stellte die ubirch GmbH ihr Backend und die Stadt Bonn ihr Open Data Portal³ zur Verfügung. Für die Programmierung konnte der visuelle Editor PXT, JavaScript oder C/C++ verwendet werden.

Die Informationsseite zum Hackathon schreibt:

„Mit C/C++ hast Du die volle Kontrolle über das Board und kannst auf alle Komponenten zugreifen.“

Das vorliegende Buch konzentriert sich auf die Programmierung in C.

Ich halte C immer noch für die beste Sprache, wenn es um Embedded Programme auf sehr eingeschränkten Mikrocontrollern geht und hoffe, mit diesem Buch ein wenig zur Verbreitung des Mynewt OS beitragen zu können.

Bruchköbel, im Oktober 2017

Alfred Schilken

¹ https://www.ubirch.de/presse/pm/170918_PM_ubirch_smart-public-life-hackathon.pdf

² Alle Informationen zur Hardware sind hier: <https://github.com/telekom/dthack17>

³ <https://opendata.bonn.de>

Für wen ist dieses Buch?

Dieses Buch ist für alle, die Programme für den Calliope Mini entwickeln wollen und die eine Alternative zu den grafischen Umgebungen PXT und Open Roberta suchen. Auch wenn die mbed-Online-IDE zu träge und die mbed-offline-Umgebung yotta nicht flexibel genug ist, der wird hier fündig. Die CLion-IDE der Softwareschmiede JetBrains beschreibe ich am praktischen Beispiel und im Tipps&Tricks Abschnitt im Anhang.

Der Schwerpunkt liegt dabei auf dem Einsatz des Mynewt Betriebssystems. Diese Open-Source-RTOS der Apache Software Foundation ist optimal vorbereitet für die Anforderungen der IoT-Anwendungen der nächsten Jahre. Gartner, ein angesehenes amerikanisches Marktforschungsunternehmen, analysiert seit 1979 den weltweiten IT-Markt und bietet Prognosen zu technischen Entwicklungen an. Gartners Prognose¹ von Ende 2015 sagt 20 Milliarden installierte IoT-Geräte für das Jahr 2020 voraus. Das sind vier mal so viele wie die für 2015 geschätzte Anzahl.

Für Soft- und Hardware-Entwickler, die auf diese boomende Technik vorbereitet sein wollen, bietet dieses Buch tiefe Einblicke in die Funktion des Mynewt RTOS. Viele seiner Komponenten werden beschrieben und in kleinen Beispielprogrammen eingesetzt. So kommen Tasks, Timer, Queues und Events in mehreren der Apps zum Einsatz. Auch „nicht-funktionale-Anforderungen“, wie Logging, Statistiken, statische und dynamische Konfiguration, Shell-Kommandos werden anhand konkreter Apps erklärt. Anhand zahlreicher Quelltext-Auszüge werden Konventionen, Fallstricke und „best practices“ vorgestellt.

Alle Code-Beispiele benutzen „pures C“, die bewährte Sprache für embedded Systeme. Auch Mynewt ist ausschließlich in C programmiert. Um den Code verstehen zu können, ist die Kenntnis einer C-ähnlichen Sprache Voraussetzung. Wer C++, Javascript oder Java lesen kann, der wird auch schnell mit dem viel einfacheren C klarkommen.

Für das Auffrischen eingerosteter C-Kenntnisse oder das Erlernen von C als neue Sprache stehen im Internet viele Quellen zur Verfügung. Empfehlenswert ist das kostenlose Onlinebuch des Rheinwerk-Verlags „C von A bis Z“,² von Jürgen Wolf. Dieses 1190 Seiten starke Buch ist 2009 bereits in der 3. Auflage erschienen und enthält alle Neuerungen des c99-Standards. Dass es auch mit sehr viel weniger Seiten geht, zeigt das 250-Seiten-Buch „The C Programming Language“ von Kernighan und Ritchie, den beiden Schöpfern der Sprache. Wichtig ist dabei, die Neuauflage aus dem Jahre 2000 zu lesen. Nur diese enthält die c99-Erweiterungen.

¹ <http://www.gartner.com/newsroom/id/3165317>

² http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/015_c_strukturen_001.htm

Was hier zu finden ist

Ich habe mich darauf konzentriert, die optimale Entwicklungsumgebung für die vielversprechende Calliope/micro:bit-Plattform zu finden und meine Arbeitstechniken dafür zu beschreiben. Dazu habe ich mehrere verschiedene Tools ausprobiert: mbed online, yotta, mbed-cli, Eclipse mit gcc-toolchain, Segger Studio, Keil und schließlich CLion.

Diese CLion IDE von JetBrains ist aus meiner Sicht das Optimum für Entwickler, die besonderen Wert auf effektives Editieren, gute Code-Navigation und Source-Level-Debugging legen. JetBrains bietet eine kostenlose 30-Tage-Testversion an. Damit können alle Programme aus diesem Buch unter macOS, Linux oder Windows kompiliert werden.

Die Quelltexte dieser Programme liegen in meinem Repository auf GitHub. Das Build-Tool von Mynewt kann dieses externe Repository sehr einfach in ein eigenes Projekt einbinden.

Ich habe schon den Boom der Mikroprozessoren und der Desktop Rechner seit den 80ern miterlebt und die nächste große Welle werden sicher die unzähligen miteinander gekoppelten Sensoren sein. Calliope und micro:bit mit ihren Sensoren oder der Raspberry Pi Zero W sind nur der Anfang.

Es mag schwierig sein, einen Treiber für einen Sensor oder für ein Anzeigeelement zu schreiben. Hierbei können unerwartete Fehlerquellen dazwischen funken, die sehr frustrierend sind. Solch ein Treiber funktioniert fast immer – und aus unerklärlichen Gründen manchmal eben nicht. Diese letzten 5 % der Fehlerbehebung können ebenso viel Zeit verschlingen wie für die 95% der funktionierenden Fälle nötig war. Wenn die benötigten Treiber aber bereits existieren, getestet sind und wie erwartet funktionieren, dann ist es kein Hexenwerk (oder sagt man heute Rocket-Science?), diese Treiber mit einem selbst geschriebenen Programmteilen zu kombinieren und damit etwas ganz Neues zu schaffen. Das ist der große Vorteil von stabilen Bibliotheken, wie sie mit den Mynewt-Packages vorliegen.

Das ist auch das Rezept, das die Arduino-Plattform so erfolgreich gemacht hat: Die komplexen Teile sind in Bibliotheken ausgelagert, der eigene Programmcode ist meist ziemlich überschaubar und einfach konstruiert. Was bei dieser Kombination herauskommt, kann aber sehr eindrucksvoll sein.

Das vorliegende Buch ist der erste Band der vier-teiligen Reihe: „IoT Projekte mit Mynewt OS“. Hier geht es um den Calliope und wie er in C mit CLion sehr komfortabel programmiert und "ge-debuggt" werden kann. Die vorgestellten Programme laufen unverändert oder mit kleinen Anpassungen der Target-Konfiguration auch auf dem micro:bit.

Diese IDE hat den besonderen Vorteil, dass Breakpoints und Single-Step auf Quelltext-Ebene funktionieren. Ohne diese Debugging-Möglichkeiten kann Softwareentwicklung schnell frustrierend werden. Kleine "Programmchen" mit 20 Quellcode-Zeilen können sicher auch mit mbed oder yotta erstellt werden. Sobald die Programme anspruchsvoller werden, würde ich nie auf gute Debugging-Features verzichten wollen, denn eine alte Entwicklerweisheit lautet: "der Teufel ist ein

Eichhörnchen". Das soll andeuten, dass die wirklich fiesen Probleme auf unerwartete Art auftreten. Und hierfür ist ein guter Debugger unverzichtbar.

Im zweiten Band der Reihe stelle ich ein Erweiterungsboard für den BBC micro:bit vor, das ihn näher an die Ausstattung des Calliope bringt. Ein Beeper, eine LED, zwei I2C-Grove-Buchsen und eine GPIO-Buchse machen den BBC micro:bit kompatibel mit dem Grove-System. Bis auf die on-board-Sensoren des Calliope gelten damit alle Aussagen des ersten Bands auch für diesen erweiterten micro:bit. Die Programme sind mit angepassten Targets leicht für den micro:bit zu konfigurieren: In den syscfg.yml-Dateien ist lediglich die Pin-Zuordnung zu anzupassen. Bastler kommen auch ohne diese Erweiterungsplatine aus. Sie löten sie entweder selbst oder nutzen ein Steckbrett für die Schaltungen.

Ein weiteres Kapitel beschäftigt sich mit der alternativen Firmware J-Link OB, die von Segger für den micro:bit kostenlos angeboten wird. Sie ist schneller und robuster als die vorinstallierte DAPLink-Firmware und bietet mehr Möglichkeiten beim Debuggen.

Der **dritte Band** macht den Schritt in die vielfältigere Welt der ARM-Boards. Calliope Mini und BBC micro:bit waren nur der Einstieg. Die immer preisgünstiger werdenden Boards mit nRF52-Mikrocontroller sind bestens geeignet für die Anforderungen des IoT: Sie sind klein, stromsparend, preiswert und haben genug Speicher für LoRaWan, Security und Fernwartbarkeit. Das praxis-erprobte Open-Source Apache Mynewt Betriebssystem und die exzellente CLion-IDE bilden eine ausgezeichnete Basis für erfolgreiche IoT-Projekte. Im dritten Band geht es um nRF51- und nRF52-Boards im Allgemeinen und um diese Platinen im Besonderen:

- Thingy52 von Nordic Semiconductor
- Arduino Primo Core
- BLE400
- Miniatur-Module aus Asien

Für das Programmieren solcher Boards ist Seggers J-Link Mini für ca. 20 € die beste Wahl. Es geht aber auch mit einem Raspberry Pi. Beide Lösungen stelle ich vor.

Der **vierte Band** konzentriert sich auf den professionellen Einsatz von nRF52-Boards. Mehr Flash und RAM erlauben die Fernwartung über Bluetooth mithilfe des newtmgr-Tools. Dieses Management-Tool wird im ersten Band nur kurz erwähnt. Der knappe Speicher des Calliope schränkt den Nutzen so stark ein, dass es keinen Spaß macht, gegen Abstürze mangels Memory anzukämpfen.

Zum Echteinsatz gehört auch die Kommunikation mit Servern in der Cloud. Die Telekom bietet dafür ihr LB-Netzwerk an. Auf Basis des Mobilfunknetzes verbindet es IoT-Geräte mit Servern – gegen Gebühr versteht sich.

Eine Alternative, die ohne Gebühren auskommt, ist der LoRaWan-Standard. Er nutzt das ISM-Frequenzband, in dem auch WLAN, Bluetooth, Funksteckdosen und vieles mehr unterwegs sind.

Mynewt bringt schon einen Treiber für einen LoRa-Chip mit. Der Preis für ein Modul liegt unter 4 € – wenn man auf die Lieferung aus China warten kann. Meine bestellten Module sind nach knapp vier Wochen angekommen.

Die manuelle Fernwartung vieler IoT-Geräte mit dem newtmgr-Tool ist viel zu aufwendig und fehleranfällig. Bei größeren Installationen wird deshalb eine maßgeschneiderte Wartungssoftware zum Einsatz kommen. Diese nutzt das newtmgr-Protokoll, um die Erreichbarkeit zu prüfen, Statistiken abzufragen, Fehlerhäufigkeiten auszuwerten, Updates zu installieren und vieles mehr. Alle diese Zugriffe sind kryptografisch abgesichert über Public-Private-Key-Verfahren, die bereits in den Bootloader eingebaut sind.

Ich hoffe, dieser Ausblick auf die drei Folgebände zum Thema IoT Projekte mit Mynewt OS haben deutlich gemacht, welches Potenzial in diesem neuen Betriebssystem der Apache Software Foundation steckt.

Die nächsten Jahre werden zeigen, wie es sich im Rennen um die Milliarden IoT-Geräte gegen die Konkurrenz von RIOT, Zephyr und mbed schlagen wird.

Was wird gebraucht?

- **Ein Calliope Mini, oder mit Einschränkungen nutzbar: ein BBC micro:bit**

Wem die 35€ für den Calliope Mini zu viel sind, der kann die meisten der Programme in diesem Buch auch auf einem BBC micro:bit¹ ausführen. Die dafür nötigen Anpassungen an die unterschiedliche Pinbelegung beschreibe ich erst im zweiten Band. Mit etwas eigener Internet-Recherche lässt sich das aber herausfinden.

- **Ein Entwicklungsrechner – am besten unter macOS. Alle Tools gibt es aber auch für Linux und Windows**

Jetbrains nennt 2 GB RAM als Mindestanforderung, 4 oder 8 sind natürlich besser. Windows wird ab Windows 7 unterstützt.

- **Eine schnelle Internetverbindung für die Installation aller Tools – oder viel Zeit**

Nach der Installation ist der Internet-Zugang nur noch nötig, wenn neue Repositories eingebunden werden sollen.

- **Mindestens die 30-Tage-Test-Version der CLion-IDE. Studenten sollten sich die kostenlose Version sichern.**

Für Nicht-Studenten kostet das Jahresabo 89€ + MwSt. Im zweiten Jahr verringert sich der Preis auf 71 € + MwSt, und ab dann kostet ein Jahr 53€ + MwSt. Wer nach einem Jahr kündigt, kann aber die Version, die zum Zeitpunkt des Kaufs aktuell war, beliebig lange weiter nutzen (perpetual fallback license²). JetBrains hat auch unter Java-Entwicklern einen guten Ruf. Ich kenne einige Kollegen die trotz der kostenlos verfügbaren Eclipse-IDE lieber die 150 Euro für die JetBrains IntelliJ IDEA investieren. Die vielen durchdachten Annehmlichkeiten sind es wirklich wert.

¹ Er ist für ca 17€ erhältlich beispielsweise bei pollin.de

² CLion ist bereits so ausgereift, dass auch mit einer nicht mehr topaktuellen Version gut gearbeitet werden kann.

- **Ausreichende Kenntnisse einer C-ähnlichen Sprache**

Erfahrung mit C nach c99-Standard ist optimal. Wer bereits mit Javascript oder Java programmiert hat, kann sich die Unterschiede schnell aneignen.

- **Keine Angst vor einer Commandline-Shell**

Einige Aktionen bei der Entwicklung mit Mynewt erfordern die Eingabe von Kommandos in einem Terminal-Fenster. Das Öffnen solch eines Terminalfensters und einige wenige Kommandos sollten bekannt sein:

pwd, cd, mkdir, ls -l, rm.

- **Grove-Module je nach Interesse**

Die on-board-Möglichkeiten des Calliope bieten schon reichlich Einsatzmöglichkeiten. Noch viel mehr wird mit dem Anschluss von Grove-Modulen an die beiden Buchsen möglich (siehe Bezugsquellen im Anhang).

- **Ein Lötkolben mit Zubehör**

Wenn Breakout-Boards oder einzelne Chips in Grove-Module verwandelt werden sollen, sind kleine Lötarbeiten nicht zu vermeiden. Wer keine Erfahrung mit dem Löten hat, sollte anfangs als Übung nur Kabel und passive Komponenten löten. Diese nehmen es nicht übel, wenn der Lötkolben ein paar Sekunden zu lange in Kontakt mit ihnen bleibt. Ein Transistor oder Sensorchip kann schnell zerstört werden, wenn er auf zu hohe Temperatur erhitzt wird.

Was nicht in diesem Buch steht

Dies ist kein Lehrbuch für C – es gibt schon genügend davon und jeder kann den für sich passenden Einstiegslevel finden. Es ist auch keine Einführung in die Elektronik der Sensoren. Hierfür gibt es ebenfalls viele Büchern und Quellen im Internet. Die meisten dieser Bücher beziehen sich auf die Arduino oder Raspberry Plattform. Hierbei ist immer zu beachten, dass die „alten“ 8-Bit-Arduinos mit 5 Volt Versorgungsspannung betrieben werden. Die neueren Arduinos mit ARM-Architektur und der Raspberry Pi arbeiten mit 3,3 Volt – wie der Calliope Mini.

Das Literaturverzeichnis im Anhang nennt einige hilfreiche Bücher, die ich bei der Arbeit an diesem Buch als hilfreich erfahren habe.

Die von Calliope und micro:bit für die Zielgruppe der 7-14 Jährigen vorgesehenen Tools PXT und Open Roberta erwähne ich nur am Rande. Es gibt inzwischen einige Bücher, die kleine Projekte mit diesen visuellen Tools vorstellen. Ich glaube, dass sie für den Einstieg und für die Zielgruppe der Schüler natürlich besser geeignet sind als eine professionelle Entwicklungsumgebung. Eine Suche nach dem Stichwort Calliope bei Amazon liefert schon im September 2017 einige Treffer – so kurz nach der Verfügbarkeit des Calliope.

Schreibstil

In Büchern dieser Art werden häufig Arbeitsschritte beschrieben, die nacheinander auszuführen sind, um ein bestimmtes Ergebnis zu erreichen. Bei solchen Anleitungen werde ich einfach schreiben, welche Aktionen ich ausführe. Da steht also nicht "man klickt erst hier und dann da" oder

"wählen sie zuerst den Gerätetyp und klicken sie dann auf den next-Button", sondern: "Ich wähle zuerst den Gerätetyp und klicke dann auf den next-Button".

Ich stelle mir dabei einen Kollegen vor, dem ich erkläre, wie ich für eine bestimmte Aufgabe meine Entwicklungsumgebung benutze. Das ist in der Softwarebranche der typische Umgangston. Seit mehr als 30 Jahren arbeite ich auf diese Art und es wäre für mich eine große Umstellung, plötzlich mit dem "Sie" formulieren zu müssen. Ich könnte dann nicht mehr so flüssig schreiben, und das Buch wäre sicher auch etwas holpriger zu lesen.

Weiterhin muss ich erwähnen, dass die Anleitungen im Text für den Mac gelten. Unter Windows bildet die MinGW-Umgebung aber eine Unix-Commandline-Shell so gut nach, dass die Arbeitsweise unter Windows kaum noch vom Mac zu unterscheiden ist. Die Installation aller Tools auf Windows beschreibe ich in einem eigenen Kapitel, für Linux ist eine Kurzanleitung im Anhang zu finden. Die CLion IDE ist für Mac, Linux und Windows verfügbar und die Bedienung ist auf allen Systemen sehr ähnlich.

Quellcode

Der Code aller in diesem Buch behandelten Apps und Treiber ist in einem Repository auf GitHub abgelegt. In *Kapitel 12 – Ein externes Repository einbinden auf Seite 129* wird beschrieben, wie der Quelltext automatisch mit dem Build-Tool heruntergeladen werden kann.

Der Quellcode des Betriebssystems einschließlich des vollständigen Bluetooth-Stacks ist nach der Installation im Verzeichnis `repos/apache-mynewt-core` zu finden. Dieser C-Code kann viele Anregungen für eigene Entdeckungsreisen im professionellen Embedded-Code geben.

Webseiten zum Buch

Das Buch enthält in den Fußnoten und im Anhang viele Links ins Internet, von denen manche leider sehr lang sind. Alle diese Links sind auch auf den Internetseiten zum Buch zu finden. Außerdem werden hier gefundene Fehlern, wichtige Änderungen im Mynewt OS und weitere Tipps erscheinen.

Bevor sie eine lange Web-Adresse im Browser eingeben, öffnen sie lieber die Webseiten zum Buch und wählen die Linkliste für den Band 1 unter: **<http://mynewt.de>**

Typographische Vereinbarungen

Bei diesen Konventionen habe ich mich an Büchern des O'Reilly-Verlags orientiert. Die Icons stammen von <https://icons8.com>. Dank diesem Hinweis darf ich sie ohne Lizenzgebühren verwenden.

Source Code Pro – Konstante Breite – fett – in grauer Box

Alle Eingaben im Terminal sind an dieser fetten Schrift auf grauem Hintergrund erkennbar. Wenn die Zeile mit einem \$ beginnt, erfolgt die Eingabe im Terminal-Fenster des Mac, Linux oder Windows-Rechners.

Beginnt die Zeile mit einer fünfstelligen Zahl, dann ist das Kommando im CoolTerm oder einem anderen Terminalprogramm einzugeben. Diese Kommandos sind für die Mynewt-Shell einer App, die auf dem Calliope läuft.

Source Code Pro – Konstante Breite – Standard – in grauer Box

Alle Ausgaben im Terminalfenster sowie Listings und Code-Schnipsel erscheinen auf grauem Hintergrund in nicht-proportionaler Schrift. Diese Schriftart eignet sich besonders gut für Quelltext und Daten, weil das große **0** und die Ziffer **0** gut unterscheidbar sind.

Source Code Pro – konstante Breite – fett – im Text

Diese fette Schrift kennzeichnet URLs, Dateinamen, Variablen, Funktionen und Typen im laufenden Text. Auch andere Namen, die typischerweise in Programmdateien auftauchen, wie Schlüsselworte und Einstellungen von Konfigurations-Dateien erscheinen so.



Dieses Icon markiert eine nützliche Konvention oder eine "best practice". Das hilft lästige Fehler zu vermeiden. Manchmal kommt es auf Kleinigkeiten an.



Stolperstein: Dieses Icon markiert eine Warnung vor typischen Fehlern oder Fallen. Beispielsweise, wenn eine gewohnte Voraussetzung im vorliegenden Fall nicht erfüllt ist.



TIPP: Dieses Icon markiert einen Hinweis, eine nützliche Zusatzinformation.



Weitere technische Details für Fortgeschrittene.



Abweichende oder zusätzliche Anleitung für Windows.

Kapitel 3 – Der Calliope

Calliope begann als Initiative von "Sechs digitalen Köpfen, die digitale Bildung an Deutschlands Schulen bringen wollen". Sie alle waren enttäuscht von der extrem langsam fortschreitenden Digitalisierung der Schulen in Deutschland. Obwohl England mit dem BBC micro:bit vorgemacht hatte, wie Schulen ins digitale Zeitalter geschubst werden können, nämlich als nicht-staatliche Initiative am Schulsystem vorbei, passierte in Deutschland nichts Vergleichbares. Während andere Länder zumindest den micro:bit für ihre Schulen importieren, zeigten sich die Kultusminister unbeeindruckt und desinteressiert.

Diese Initiativgruppe hat die Vision, "flächendeckend alle Schülerinnen und Schüler der dritten Klasse jedes Jahr mit einem mini auszustatten." Sie sieht den Calliope Mini als "den Zirkel des 21. Jahrhunderts". Hinter diesem Projekt stehen folgende Personen.

- **Gesche Joost**
ist Professorin für Designforschung an der Universität der Künste Berlin.
- **Stephan Noller**
ist Diplom-Psychologe, Digital-Unternehmer und Vater von vier Töchtern.
- **Franka Futterlieb**
ist Diplom Designerin, unterrichtet Mediengestaltung in Berlin und ist Inhaberin eines Unternehmens, das digitales Lernspielzeug entwickelt.
- **Jørn Alraun**
ist Diplom Interaktionsdesigner, Inhaber eines Unternehmens, das digitales Lernspielzeug entwickelt und Mitglied der Interaction Design Association.
- **Maxim Loick**
ist Vater, Gründer, und selbstständiger IT-Berater.
- **Klaus J. Buß**
ist kaufmännischer Leiter und begleitete mehrere Firmengründungen.

Mit ihrem technischen Hintergrund analysierten sie den BBC micro:bit und beschlossen, nicht einfach den micro:bit zu übernehmen und das Lehrmaterial an deutsche Verhältnisse anzupassen. Stattdessen entwickelten sie eine erweiterte Version mit zusätzlichen Eigenschaften, die sie für dringend erforderlich hielten "um SchülerInnen in der Grundschule mit tollen Experimenten zu versorgen". Die Basis des micro:bit wurde übernommen – also Mikrocontroller, LED-Matrix, Buttons und Sensoren – und um folgende Komponenten erweitert:

- RGB-LED
- Buzzer und Mikrofon
- Treiber für zwei Motoren mit Anschluss über Lötkontakte
- Zwei Grove-Buchsen für das SeeedStudio-System
- Erweiterungskontakte für eine optionale 26 Pin-Stiftleiste

Um diese Vision zu verwirklichen, mussten sie natürlich Unterstützer finden. Auf der einen Seite konnten sie Unternehmen als Sponsoren gewinnen, darunter Google, Cornelsen Verlag, SAP, Bosch, NXP, Nordic Semiconductor, Würth, Schleicher Electronic, Roberta, Telekom, Microsoft und einige mehr¹. Auch das BMWi (Bundesministerium für Wirtschaft und Energie) konnte für eine Förderung gewonnen werden.

Darüber hinaus stellten sie ihr Projekt auf der Crowdfunding Plattform www.startnext.com vor. Vom 19.12.2016 bis 16.01.2017 konnte das Calliope Projekt 969 Fans und 1902 Unterstützer überzeugen und mehr als 108.000€ einsammeln (Funding-Ziel war 60.000€). Der Beweis für den Erfolg des Projekts liegt jetzt in Form des Calliope Mini Boards auf meinem Schreibtisch.

Inzwischen ist Calliope eine gemeinnützige GmbH mit Sitz in Berlin und vertreibt den Calliope Mini über Amazon, EXP-Tech und Conrad auch an interessierte Maker. Von den so erwirtschafteten Einnahmen werden weitere Boards und Materialien für Schulkinder erstellt und kostenlos an Schulen verteilt. Von dem Verkaufspreis können vermutlich mehr als 5 € als „Spende“ für die Förderung der „Digitalkunde“ an deutschen Schulen angesehen werden. Das erklärt den doppelten Preis des Calliope (35€) im Vergleich zum BBC micro:bit (17€).

Kurz bevor dieses Buch in den Druck geht, lese ich eine verblüffende Meldung² im Heise-Ticker: *Calliope mini für Schulen: Geschenk oder Lobbyismus*

Die Calliope gGmbH hatte 2500 Calliope Minis an 100 Grundschulen verschenkt, ein Wert von immerhin 75.000 €. Das löste Angst und Schrecken beim *Schweriner Bildungsministerium*, bei der *Lehrergewerkschaft GEW* und beim *Verein LobbyControl* aus.

René Scheppler von der GEW wird zitiert mit der Aussage: „*Ich habe den Eindruck gewonnen, dass hier eine Einflussnahme auf schulische Bildung stattfindet.*“

Natürlich wird die schulische Bildung beeinflusst – und das ist auch dringend nötig, finde ich. Für mich ist es absurdes, politisches Theater, wenn ein Land die digitale Bildung so sträflich vernachlässigt und auch noch verhindern will, wenn andere Interessengruppen endlich die Initiative ergreifen.

Wenige Tage später berichtet *Der Tagesspiegel*, dass Bundeswirtschaftsministerin Brigitte Zypries die Produktionsstätte des Calliope in Berlin Schöneberg besuchte. In dem Artikel³ werden auch aktuelle Produktionszahlen genannt: 25.000 Calliope Minis wurden schon produziert, davon sind 16.500 deutschlandweit bereits im Einsatz. Die nächste Charge von 20.000 ist gerade in der Produktion.

Auch ein erster Nachbau in China ist bereits aufgetaucht⁴: Der sino:bit nimmt den Calliope als Grundlage und erweitert die LED-Anzeige auf 12 x 12 LEDs – gerade genug für chinesische Schriftzeichen. Wie beim Calliope ist das Ziel beim sino:bit der Einsatz in Schulen. Über zu wenig Aufmerksamkeit in den Medien kann sich der Calliope also nicht beklagen.

¹ siehe <https://calliope.cc/partner>

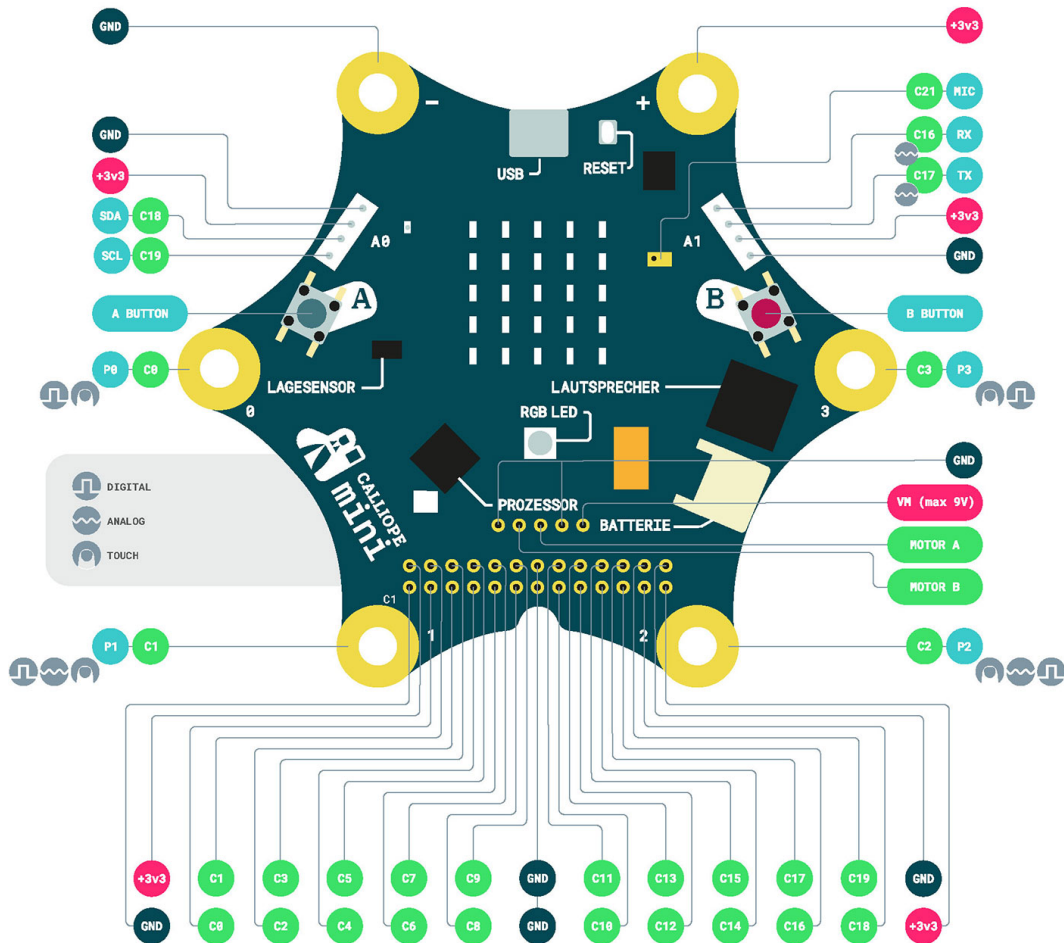
² <https://heise.de/-3867025>

³ <http://www.tagesspiegel.de/wirtschaft/calliope-kindercomputer-aus-berlin-erobert-die-schulen/20493096.html>

⁴ <https://heise.de/-3862843>

Die Platine

Die Platine gab es zuerst in einer Version 0.3. Wegen wichtigen Korrekturen wurde sie von der Version 1.0 abgelöst. Die neue Platine ist erkennbar an dem Calliope Logo, das links unterhalb der LED-Matrix liegt. Bei dem älteren Layout liegt das Logo rechts unterhalb der LEDs.



Ein-/Ausgänge bereit, während Version 1.0 zwei GPIOs benutzt, die auch analoge Signale messen können. Das ist eine deutliche Verbesserung, denn damit können alle Grove-Module mit analoger Schnittstelle angeschlossen werden.

Zielgruppe: nicht nur für Grundschüler

Allein schon die verspielte Sternform ist für Grundschüler sicher reizvoller als das schlichte Rechteck des BBC micro:bit. Dass jeder stolze Besitzer eines Calliope auf der Rückseite seinen Namen in ein weiß grundiertes Feld eintragen kann, wird vielleicht zu einem schonenderen Umgang beitragen. Da alle Komponenten als SMD flach aufgelötet sind, gibt es keine hervorstehenden Bauteile, die leicht abbrechen könnten oder Pins, an denen man sich verletzen könnte. Andere Prototyp-Boards erinnern da eher an Nagelbretter.

Für Programmieranfänger gibt es mehrere grafische Programmierumgebungen, die ich im Unterkapitel *Calliope mini*, *PXT Editor* und *Open Roberta Lab* auf Seite 31 sehr kurz anreiße. Für den Einstieg in die Logik des Programmierens sind diese Grafiktools sehr gut geeignet, aber sie stoßen an ihre Grenzen, wenn es um komplexere Aufgaben geht. Das vorliegende Buch zielt auf Maker (Bastler) und auf interessierte Schüler der Mittel- und Oberstufe ab. Denn der Calliope Mini ist mit seinen Erweiterungsmöglichkeiten auch gut für ältere Schüler geeignet. Die Motorsteuerung zusammen mit der Bluetooth-Kommunikation eröffnen beispielsweise das Experimentieren mit Schwarm-Algorithmen, die Simulation eines autonomen Staubsaugers und vieles mehr. Auch einige Grove-Module für die Messung von Feinstaub, CO₂-Gehalt und anderen Parametern der Luftqualität, Muskelspannung, Herzschlagvariabilität usw. sind erst in höheren Schulklassen und AGs sinnvoll einzusetzen.

Der Calliope wird natürlich auch den Informatik-Unterricht in der Oberstufe beeinflussen und hier ist ein robustes Real-Time-Betriebssystem wie Mynewt ein optimales Lernobjekt. Alle Bestandteile eines "ausgewachsenen" Betriebssystems sind hier am konkreten C-Code zu studieren: Scheduler, Tasks, Queues, Events, Treiber, Build-System, Quellcode-Verwaltung, ...

Hierbei hat aus meiner Sicht die Sprache C einige Vorteile gegenüber C++. Für das Verständnis ist es hilfreich, wenn möglichst wenige Details versteckt werden. Die C++-Klassen sind aber gerade dazu geschaffen, die Details in Unterklassen zu verbergen. Was für die Verwendung von Komponenten ein Vorteil ist, kann für das Verstehen der Zusammenhänge hinderlich sein. Darüber hinaus ist C++ eine sehr komplexe Sprache mit vielen Sonderregeln. Im Vergleich dazu ist C eine der kleinsten Sprachen überhaupt – jedenfalls was den Umfang der Sprachkonstrukte wie if, for, switch usw. angeht.

BBC-micro:bit kompatibel – und mehr

Die 5x5 LED-Matrix und die beiden Buttons erinnern schon rein äußerlich an den BBC micro:bit. Die Gemeinsamkeiten gehen aber noch weiter. Der Calliope verwendet denselben Mikrocontroller und weitgehend auch dieselben GPIO-Belegungen. Diese weitgehend identische Hardware hat den Vorteil, dass auch die micro:bit-Software fast unverändert übernommen werden kann. Der Bootloader, der Ladevorgang, die mbed-Toolchain, Linkerscripts, Memory-Layout, Lancaster Runtime Klassen, usw. sind mit wenig Aufwand für Calliope anzupassen. Für den Calliope mussten also die mbed-Basis, die Lancaster-Erweiterungen und die grafische PXT-Umgebung nur geringfügig erweitert werden. Alle diese Erweiterungen liegen als Open-Source auf GitHub.¹

Die Ansteuerung der LED-Matrix und Bluetooth funktionieren sogar ohne Änderung. Der Button A wird auch erkannt, nicht aber der Button B. Der micro:bit nutzt den P0.26, während Calliope den P0.16 mit dem Button B verdrahtet hat. micro:bit-Programme, die nur den Button A und die LED-Matrix verwenden, laufen also ohne Änderung. Alle anderen müssen mindestens mit den für Calliope passenden Pin-Definitionen neu übersetzt werden – und mit den zusätzlichen Treibern, wenn Calliope-spezifische Sensoren benutzt werden. Bei den Treibern für die Peripherie-Chips hört nämlich die Kompatibilität auf. Während der micro:bit für den Bewegungs- und Lage-sensor jeweils einen eigenen Chip verwendet, hat Calliope die beiden in einem einzigen Chip integriert. Die anderen Peripheriekomponenten des Calliope gibt es beim micro:bit gar nicht und natürlich auch keine Treiber dafür.

Kmponente	Calliope Mini	BBC micro:bit
Haupt Mikrocontroller	Nordic nRF51822	Nordic nRF51822
USB Mikrocontroller	NXP KL26Z	NXP KL26Z
Accelerometer	Bosch BMX055	NXP MMA8653
Magnetometer	Bosch BMX055	NXP MAG3110
Gyroskop	Bosch BMX055	–
Motortreiber	TI DRV8837	–
MEMS Mikrofon	SPW2430?	–
RGB-LED	WS2812b	–
Button A	P0.17	P0.26
Button B	P0.16	P0.16
LED-Matrix	identisch	identisch

Tabelle 2 – Vergleich der Hardware von Calliope Mini und BBC micro:bit

¹ <https://github.com/calliope-mini>

Alle diese Unterschiede spielen keine Rolle, wenn Anwendungen mit dem grafischen Tool PXT programmiert werden. Hier sind alle Anpassungen schon von den Entwicklern der Calliope Plattform erledigt worden. Für die Entwicklung mit yotta ist darauf zu achten, dass die Calliope-spezifischen Header-Dateien und Bibliotheken verwendet werden. Am aufwendigsten ist das Anpassen eines neuen Betriebssystems oder eine Bare-Metal-Programmierung, weil hier der Entwickler selbst die Unterschiede der Hardware zwischen micro:bit¹ und Calliope² beachten muss.

Ein Blockschaltbild

Das Blockschaltbild zeigt die Komponenten des Calliope. Der nRF51-Chip in der Mitte ist das Herzstück des Boards. In ihm ist das jeweilige Programm gespeichert und darin läuft auch das Programm ab. Alle anderen Blöcke dienen zur Ein- und Ausgabe. Der nRF51 steuert sie entweder direkt über GPIO-Leitungen oder den internen I2C-Bus..

Der linke Block ist ein eigenständiger Mikrocontroller, der zum Laden des Programmes in den nRF51 dient. Die darauf installierte Firmware ermöglicht auch das Debuggen über den USB-Anschluss und emuliert eine serielle Schnittstelle. Ein Terminalprogramm auf einem PC kann über diese USB-Schnittstelle Kommandos an das Programm im nRF51 schicken und seine Ausgaben anzeigen.

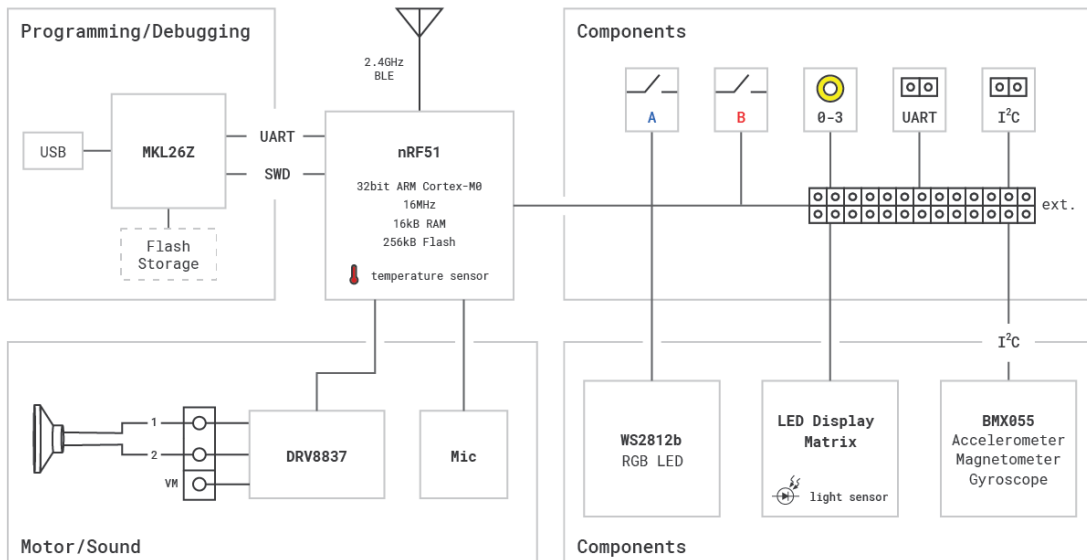


Abbildung 2 – Blockschaltbild des Calliope

CC BY SA Quelle: <https://calliope-mini.github.io/v10/>

¹ <https://github.com/micro:bit/micro:bit>

² <https://calliope-mini.github.io/v10/>

Die RGB-LED

Ein Vorteil des Calliope ist eine eingebaute RGB-Leuchtdiode, die der micro:bit nicht vorweisen kann. Ich glaube die zusätzlichen Kosten von ca. 50 Cent für das WS2812B-Bauteil lohnen sich: Allein schon rot und grün sind als Signalfarben nützlich. Die additive Farbmischung der drei Grundfarben ermöglicht auch einige Anwendungen im Kunstunterricht oder der Physik.

Die LED ist verbunden mit P0.18. Die dafür zuständige Klasse der Lancaster-Runtime ist CalliopeRGB¹. Den Treiber für Mynewt habe ich selbst entwickelt. Er ist in meinem GitHub-Repository² zu finden.

Die Ansteuerung des WS2812B-Chips ist sehr zeitkritisch. Die Farbwerte für Grün, Rot und Blau werden mit jeweils 8 Bits übertragen, wobei eine 0 als 350 nsec³ Low-Pegel und 750 nsec High-Pegel und eine 1 als 750 nsec Low und 350 nsec High – also genau umgekehrt – übertragen wird. Die Toleranzen für diese Zeitangaben sind in Datenblättern mit +/- 150 nsec angegeben. Die Meinungen, was an diesen Zeitintervallen wesentlich ist, gehen im Internet allerdings weit auseinander. Das Kritischste scheint die Länge des LOW-Pegels der 0 zu sein. Wenn diese zu lang wird, erkennt der Chip eine 1. Die Dauer des Highpegels ist unkritisch. Solange sie unter 6 µs bleibt scheinen längere Zeiten nicht zu stören.

Die Tonausgabe – „Buzzer“

Ein weiterer Pluspunkt des Calliope im Vergleich mit dem BBC micro:bit betrifft die Tonerzeugung: Der micro:bit hat keine Möglichkeit, einen Beep von sich zu geben. Auch hier bin ich wieder überzeugt, dass die zusätzlichen Kosten von ca. 50 Cent für den eingebauten Buzzer des Calliope sich lohnen. Mag sein, dass die Benutzer des micro:bit, also 11- bis 12-jährige Schüler, schon eher einen Buzzer mit Krokodilklemmen anschließen können, als die 9-10 Jährigen, für die der Calliope gedacht ist..

Die Tonausgabe nutzt dieselben Pins wie die Motorsteuerung, die als PWM-Ausgänge konfiguriert sind. Es ist also nicht möglich, Tonausgabe und Motor separat zu nutzen. Damit die Motorsteuersignale sich nicht als Pfeifen störend bemerkbar machen, sollte die PWM-Frequenz oberhalb des hörbaren Bereichs liegen, beispielsweise höher als 30 kHz.

Für die Tonausgabe wird der PWM-Duty-Cycle fest auf 50% eingestellt, um ein Rechtecksignal zu erzeugen. Die PWM-Periodenzeit bestimmt die Frequenz des erzeugten Tons. Das hört sich zwar wegen der vielen Oberwellen etwas schrill an, ist aber am einfachsten zu erzeugen. Im Physikunterricht könnte demonstriert werden, wie mit unterschiedlich großen Kondensator-

¹ <https://github.com/calliope-mini/microbit-dal/blob/master/source/drivers/CalliopeRGB.cpp>

² <https://github.com/schilken/mynewt-calliope>

³ Nanosekunden, also 0,000.000.001 Sekunden

Widerstand-Kombinationen der Ton angenehmer gemacht werden kann. (Harry Fairhead¹ empfiehlt auf seiner Website IoT Programmer einen Kondensator zwischen 1 μF und 0.1 μF mit einem Widerstand von 1k Ω .)

Die Frequenz f des mittleren C ist 281.6Hz. Nach der Formel

$$t = 1/f = 281.6 = 0,003551 \text{ sec}$$

ergibt sich eine Periodenzeit von 3551 μsec für diesen Ton.

Der in Kapitel 10 vorgestellte Sound-Treiber nutzt diese Formel, um Töne mit Frequenzen zwischen 20 und 20000 Hz aus dem Lautsprecher – oder besser dem Buzzer – ertönen zu lassen. Für meinen Mynewt-Treiber habe ich mich an der C++-Klasse CalliopeSoundMotor² orientiert.

Der Motortreiber

Es hat mich überrascht, eine Treiberstufe für zwei Motoren auf der Platine zu finden. Für die Nutzung dieser Verstärkerstufe sind Lötarbeiten für den Anschluss der Motoren und vermutlich stärkere Batterien notwendig. Diese Anwendung ist vermutlich nicht für die Grundschule geeignet. Für Maker ist es allerdings ein zusätzlicher Anreiz, sich den Calliope genauer anzuschauen. Jede eingesparte externe Komponente ist eine Fehlerquelle weniger und Fehlerquellen gibt es bei Mikrocontroller-Projekten schon genug.

Der verbaute Chip des Types DRV8837³ kann Motoren mit bis zu 1.8 A versorgen. Der Chip hat einen separaten Pin für die Motorspannung, die bis zu 11 Volt betragen darf. Das reicht auch für größere Robotfahrzeuge, die über Bluetooth ferngesteuert werden können. Auch der Betrieb von Wasserpumpen, Hebelarmen und anderen Verbrauchern ist damit möglich. Für Schüler der Sekundarstufe bedeutet das viele zusätzliche Einsatzmöglichkeiten. Die folgenden drei MCU-Pins sind mit dem Treiber-Chip verbunden:

GPIO	DRV8837	Bedeutung
P0.28	SLEEP	Schaltet den Chip auf minimalen Strombedarf
P0.29	IN1	Die vier Kombinationen schalten die beiden Ausgänge OUT1 und OUT2: 00 : stehen 01 : rückwärts 10 : vorwärts 11 : Bremsen
P0.30	IN2	

Tabelle 3 – Die GPIOs für die Motorsteuerung

¹ <http://www.iot-programmer.com/index.php/books/micro-bit-iot-in-c>

² [github → microbit-dal/blob/master/source/drivers/CalliopeSoundMotor.cpp](https://github.com/microbit-dal/blob/master/source/drivers/CalliopeSoundMotor.cpp)

³ <http://www.ti.com/product/DRV8837>

Dort gibt es auch einen Beitrag³, der die verschiedenen Anschlussarten an die fünf Motor-Kontakte beschreibt. Entweder ein Motor kann vorwärts und rückwärts gesteuert werden oder zwei Motoren unabhängig voneinander nur in einer Richtung.

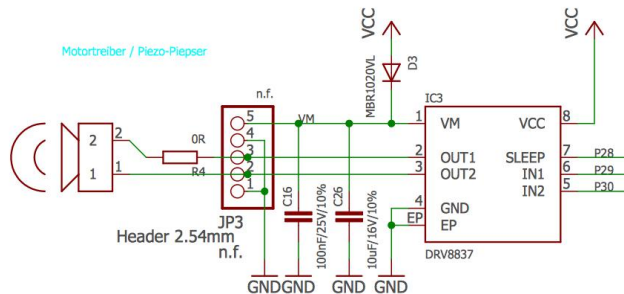
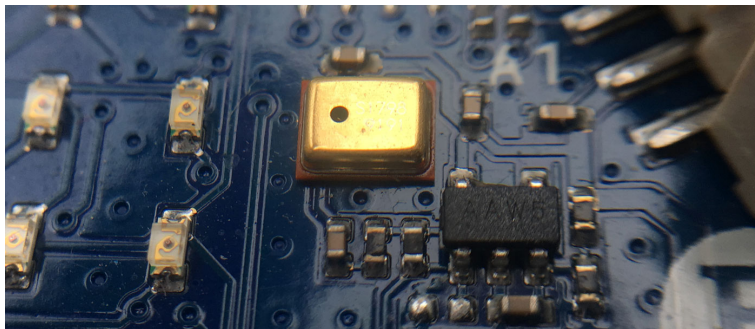


Abbildung 3 – Ausschnitt aus dem Schaltplan des Calliope.

Auf der Platine ist auch ein winziges Mikrofon vorhanden. Es kann für einfache Tonaufnahmen oder als Schallsensor für Lautstärkemessungen, Klatschschalter und dergleichen verwendet werden. Das sehr schwache Tonsignal wird mit einem Operationsverstärker auf einen Pegel verstärkt, der mit einem ADC am GPIO P0.03 „gesampelt“⁴ werden kann. Auch hier hat der BBC micro:bit nichts Vergleichbares zu bieten.



⁴ „Sampeln“ bedeutet in diesem Zusammenhang: Ein Analogsignal mit hoher Taktrate messen und speichern.

Die Grove-Buchsen

Aus meiner Sicht war es eine geniale Entscheidung, diese beiden Grove-Buchsen auf das Board zu packen. Wer versucht hat, externe Komponenten stabil mit dem micro:bit zu verbinden, der weiß das sehr zu schätzen. Es gibt zwar Erweiterungsboards, die an den micro:bit geschraubt werden, oder auch Steckverbinder, in die das micro:bit-Board gesteckt wird, aber das sind sehr plumpe Lösungen im Vergleich zu den zwei Grove-Buchsen des Calliope.

Das Grove-System ist eine Sammlung von Erweiterungsboards, die alle dieselben Steckverbinder verwenden. Es gibt über 50 verschiedene Komponenten, die von einem simplen Button bis zur Aufnahme von EKG-Daten (Elektrokardiogramm über Elektroden) reichen. Mehr dazu folgt in *Kapitel 13 – Das Seeedstudio Grove-System*.

Die linke Grove-Buchse A0 ist verdrahtet mit dem board-internen I2C Bus, der auch zur Abfrage des Accelerometers und des Kompasses dient. Dazu sind die beiden MCU-Pins P0.19 für SCL und P0.20 für SDA vorgesehen. Wenn irgend möglich sollte diese Buchse auch nur für externe I2C-Komponenten verwendet werden. Grundsätzlich können die beiden Pins umkonfiguriert werden, beispielsweise zu digitalen GPIOs oder PWM oder ADC-Input, dann sollte aber vorher der Schaltplan nach möglichen Störquellen analysiert werden. Ich würde es vermeiden und nur die zweite Grove-Buchse für solche Erweiterungen verwenden.

Die Grove-Buchse A1 war ursprünglich, also in der Version 0.3, hauptsächlich für eine serielle Schnittstelle (UART) vorgesehen und mit den Pins P0.16 für TX und P0.21 für RX verbunden. Diese Pins des nRF51 sind nur für die digitale Ein-/Ausgabe geeignet. Glücklicherweise haben die Entwickler dies bei der Version 1.0 noch geändert. Jetzt ist der Pin1 mit P0.26 und Pin2 mit P0.27 verbunden. Mit dieser Änderung kann die Buchse sowohl als UART, als auch für analoge Eingaben verwendet werden.

Ein externer Speicherchip (optional)

Den externen Speicherchip habe ich auf dem Schaltplan des Calliope entdeckt. Auch das war eine Überraschung. Die Stelle mit acht Kontakten in der Nähe der Grove-Buchse A1 ist auf meiner Calliope-Platine nicht bestückt. Hier kann ein Flash-Chip M45PE16 mit 16 MB oder ein M45PE80 mit 8 MB eingebaut werden. Auf dem Schaltplan des micro:bit ist nichts Derartiges zu finden. Anscheinend ist geplant, in diesem Flash-Chip eine Sammlung von Programmen abzuspeichern, von der mit einer Magic-Button-Sequenz ein Programm in den Arbeits-Flash-Speicher des nRF51-Mikrocontrollers geladen werden kann.

Dieser Flash-Speicher ist nur vom USB-Controller-Chip (KL26Z) über eine SPI-Schnittstelle ansprechbar. Er kann also nicht von Programmen des nRF51 für die Speicherung permanenter Daten genutzt werden. In der offiziellen Calliope Dokumentation¹ und auf dem Schaltplan ist er als Flash-Programmspeicher (optional) erwähnt.

¹ <https://calliope-mini.github.io/v10/>

Der Erweiterungsport für Maker

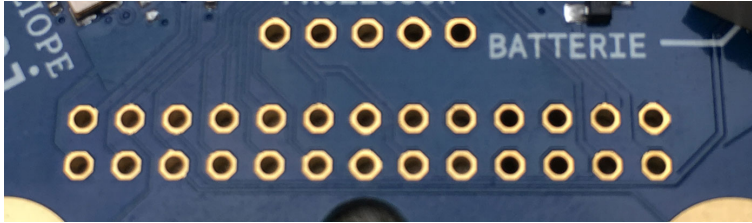


Abbildung 4 – Oben die fünf Löt-Kontakte des Motortreibers, unten die 26 Kontakte des Erweiterungsports

Weitere Möglichkeiten für Maker ergeben sich durch 26 kleine Lötkontakte auf dem Board. Diese Kontakte machen fast alle Pins des Mikrocontrollers zugänglich. Hier könnte also leicht ein Adapter für weitere Grove-Buchsen angeschlossen werden, oder eine Stiftleiste, auf die ein Breadboard-Expander gesteckt werden kann. Das ist eine verbreitete Erweiterung für Raspberry-Pi, um Schaltprototypen auszuprobieren. Die 26 Pins des Expansion-Headers sind wie folgt belegt:

Nr.	Chip-Pin	Bedeutung	Nr.	Chip-Pin	Bedeutung
1	GND		2	VCC	
3	P0.19	Grove A0 SCL	4	P0.20	Grove A0 SDA
5	P0.27	Grove A1 Pin 2	6	P0.26	Grove A1 Pin 1
7	P0.15	LED R3	8	P0.14	LED R2
9	P0.13	LED R1	10	P0.12	LED S9
11	P0.11	LED S8	12	P0.10	LED S7
13	GND		14	GND	
15	P0.09	LED S6	16	P0.08	LED S5
17	P0.07	LED S4	18	P0.06	LED S3
19	P0.05	LED S2	20	P0.04	LED S1
21	P0.22	Stern 3	22	P0.02	Stern 2
23	P0.01	Stern 1	24	P0.00	Stern 0
25	VCC		26	GND	

Tabelle 4 – Die Pin-Belegung der Erweiterungs-Kontakte

Leider sind die meisten Kontakte für die Ansteuerung der LED-Matrix reserviert. Mir ist auch unverständlich, warum vier Kontakte für GND und zwei für VCC, also die Versorgungsspannung, belegt werden.

Die drei Signal-Leitungen des SPI-BUS (P0.21 - P0.23) sind nicht auf den Erweiterungsport herausgeführt. Damit können diese MCU-Pins nicht genutzt werden. Der BBC micro:bit hat diese Pins auf seine Seitenkontakte geführt – ein großer Vorteil. Glücklicherweise ist die Zuordnung der SPI-GPIOs zu beliebigen Pins möglich. Wenn die Sternkontakte frei sind, kann MISO, MOSI und SCK beispielsweise auf P0.00, P0.01 und P0.02 gelegt werden.

Kapitel 4 – Laufzeitumgebungen

Bevor ich einige der möglichen Entwicklungsumgebungen vorstelle, will ich zuerst klären, was ich mit Laufzeitumgebung meine: Unter einer Laufzeitumgebung verstehe ich die Software, die zusätzlich zum selbst erstellten Programmcode auf dem Mikrocontroller installiert sein muss, um das Programm ausführen zu können. Dabei sind zwei Speicherarten zu unterscheiden: erstens die Größe des Binärcodes im Flashspeicher und zweitens der Bedarf an RAM-Speicher. Der RAM-Bedarf reicht von 0 kB bei "bare metal" bis zu mehreren kB von MicroPython oder den RTOS¹, also den "echten Betriebssystemen", für die es mehrere zur Auswahl gibt.

Bare Metal

"Bare metal" bedeutet, dass die Software ohne irgendeine Zwischenschicht direkt auf dem "Silizium" läuft. Kein Betriebssystem hilft beim Umgang mit Interrupts oder erleichtert die Ansteuerung von Sensoren über Treiber. Das Programm muss selbst für die Initialisierung von Stack und Vektoren sorgen. Auf die Register des Mikrocontrollers greift es direkt über die entsprechenden Memoryadressen mit Bitmanipulationsbefehlen zu. So lässt sich in einem kB Flash eines AVR ATTiny-12 sehr viel Funktionalität unterbringen. Für die vergleichsweise üppig ausgestatteten ARM-Mikrocontrollern ist das aus meiner Sicht nicht mehr zeitgemäß.

MikroPython

Das andere Extrem der Mikrocontroller-Laufzeit-Umgebungen bildet Python, das seinen Interpreter zwischen das Programm und den Chip schiebt. Es gibt schon länger eine Portierung für den BBC micro:bit und diese sollte mit geringem Aufwand auch für Calliope anpassbar sein. Eine ausführliche englischsprachige Dokumentation ist im Internet² zu finden. Die dort beschriebene Schnittstelle bietet alles, was man sich als Entwickler wünschen kann.

Einzige Ausnahme ist die Unterstützung des Bluetooth-Stacks. Der Interpreter ist zwar mit 8 kB RAM in seiner Minimalausstattung recht genügsam. Er lässt aber nicht genügend RAM übrig für das Nordic Softdevice. Ohne diese Bluetooth-Bibliothek können keine BLE-Anwendungen in Python codiert werden. Falls der Sponsor Nordic Semiconductor dem Calliope Mini die 32-kB-Version des nRF51822-Mikrocontrollers³ spendieren würde, wäre das schlagartig anders. Dann würde BLE auch unter MicroPython nutzbar sein.

Wegen der Unterschiede zum micro:bit sind kleine Anpassungen nötig. Es gibt auch bereits eine modifizierte Kopie⁴ der BBC-Version für den Calliope auf GitHub⁵. Inwieweit er schon

¹ Realtime Operating System

² <https://microbit-micropython.readthedocs.io/en/latest/>

³ Der nRF51822-QQAC hat 32 kB und kostet nur xx Cent mehr

⁴ Einen sogenannten Fork – wie Abzweigung

⁵ <https://github.com/calliope-mini/micropython>

Kapitel 5 – Entwicklungsumgebungen

Der Arduino hat gezeigt, wie wichtig eine möglichst einfache Programmierumgebung für den Erfolg einer Plattform ist. Der einfach zu bedienende Editor und der Download des erzeugten Programms direkt aufs Board waren Schritte in die richtige Richtung. Die meisten der folgenden Umgebungen bieten auch diese einfache Downloadmöglichkeit auf den Calliope. Darüber hinaus sind sie noch bedienungsfreundlicher geworden – für die Zielgruppe der Grundschüler sogar sehr spielerisch.

Calliope mini, PXT Editor und Open Roberta Lab

Die von der Calliope-Initiative empfohlenen Programmierumgebungen sind der PXT Editor von Microsoft und das Open Roberta Lab des Fraunhofer Instituts. Der Calliope Mini Editor ist nur für das erste Schnuppern gedacht. Er bietet aber in seiner sehr eingeschränkten Umgebung ein schnelles Erfolgserlebnis. Alle diese grafischen Tools kommen ohne Installation auf dem PC aus. Sie können direkt im Web-Browser bedient werden. Im Menu der Calliope Homepage <https://calliope.cc> öffnet der Menüpunkt "Editor" die Seite mit den Programmierumgebungen. Die Aufforderung *"Such Dir den Editor aus, mit dem Du dein Projekt realisieren möchtest!"* lädt zum sofortigen Start ein.

PXT ist ein Open-Source Projekt von Microsoft, das auf Basis von Makecode eine grafische Programmierumgebung für mehrere Plattformen bietet. Neben BBC micro:bit, Sparkfun, Adafruit und Minecraft gibt es auch eine auf Calliope angepasste Version.

Ich muss gestehen, dass ich den PXT-Editor anfangs unterschätzt habe und erst gar nicht ausprobieren wollte. Eher beiläufig verirrte ich mich auf dem iPhone in den PXT-Blockeditor für den micro:bit. Ich klickte mich durch das Menü mit den Blöcken für Input, Schleifen, logische Abfragen. Das meiste, was ich von einer Programmiersprache erwarte, fand ich auch hier.

Neugierig geworden öffnete ich den PXT-Editor auch im Desktop Safari und fand hier noch mehr Möglichkeiten. Im Abschnitt für Fortgeschrittene bieten die Text-Blöcke beispielsweise auch `strlen`, `strcat` und `strcmp` an. Das sind alte Bekannte aus der C-Standard-Bibliothek.

Über den letzten Punkt im Menü können auch Bibliotheken geladen werden. Hier hat mich Bluetooth natürlich am meisten interessiert. Ich stellte erstaunt fest, dass alle Services der Lancaster C++ Runtime auch hierüber verfügbar sind: LED-Service, Buttonservice, Eventservice, Accelerometer, Temperatur, Licht, Beacon. Alle diese Services können auch über einfache Blöcke erzeugt und konfiguriert werden. In vielen Fällen wird es also gar nicht nötig sein, C++-Code zu schreiben. Mit den Pin-Blöcken ist es sogar möglich, den I2C-Bus für die Abfrage von Sensoren und die Ansteuerung von Aktoren¹ zu benutzen.

Für Programmier-Anfänger finde ich den Einstieg über grafische Blöcke, die per Drag&Drop zusammengesteckt werden, ideal. Die Einbuchtungen und Nasen, die ein bisschen an Puzzle-Steine erinnern, deuten schon an, wie die verschiedenen Blocktypen zusammenpassen. So ent-

¹ Aktoren sind: Relais, Motoren, Magnetschalter

Kapitel 6 – Betriebssysteme für Calliope

Ursprünglich machte ich mich auf die Suche nach dem für mich "besten" Betriebssystem für den BBC micro:bit. Es sollte mit den Einschränkungen auf 256 kB Flash und 16 kB RAM möglichst viel Funktionalität bieten, unbedingt Bluetooth unterstützen und gute Hilfen bei der Fehlersuche bieten. Meine Odyssee führte mich der Reihe nach von ARM mbed zu Lancaster Runtime, mit kurzen Abstechern zu RIOT und Zephyr, bis ich endlich in Apache Mynewt alles fand, was ich gesucht hatte. Es kommt als letztes in dieser Aufzählung, weil danach die Suche ein Ende hatte.

ARM mbed OS

"Mbed simplifies and speeds up the creation and deployment of IoT devices based on Arm micro-controllers "

<https://developer.mbed.org>

Open-Source auf GitHub.

Apache License 2.0

Erste öffentliche Version im März 2016.

ARM mbed ist nicht nur eine Entwicklungsplattform für ARM Mikrocontroller, sondern bietet mit mbed OS auch ein RTOS an. Genauer gesagt sind es zwei: mbed OS 2 und mbed OS 5.

PRO:

- Sehr aktive Entwickler Community auf mbed.com
- Durchsuchbares Verzeichnis von Bibliotheken und Anwendungen
- Apache License 2.0
- 60 Partner, darunter alle Hersteller von ARM Mikrocontrollern
- Schneller Einstieg mit online IDE im Browser

CONTRA:

- mbed OS 5 läuft nicht mit 16 kB RAM, also nicht auf Calliope
- Nur für ARM-Mikrocontroller
- Kontrolliert von ARM Ltd.
- Weiterentwicklung von mbed OS 2 ist ungewiss
- Zukunft des offline Build-Tools yotta ist ungewiss

Kapitel 7 – Apache Mynewt

Apache Mynewt ist ein sehr junges Projekt. Im Oktober 2015 wurde es von der Firma Runtime.io der Apache Software Foundation angeboten. Die Kern-Entwickler dieser Firma hatten sich 2014 von ihrem vorherigen Arbeitgeber Silverspring Networks abgespalten und die Firma Runtime.io gegründet. Bei Silverspring hatten sie seit 2002, also mehr als zehn Jahre, Erfahrungen im Bereich embedded-C sammeln können. Die Hauptanwendungen waren "smart meter" und Steuerungen von Straßenbeleuchtungen, also massiv verteilte Systeme mit zusammen mehr als 20 Millionen vernetzten Geräten. Alle diese Geräte nutzen Mikrocontroller mit sehr eingeschränkten Ressourcen und mussten über die Jahre remote mit Updates versorgt werden. In einigen YouTube-Videos schildern Runtime-Mitarbeiter, welche Schwierigkeiten sie dabei zu überwinden hatten. Alle diese Erfahrungen flossen in die Entwicklung von Mynewt ein. Dementsprechend sind die Schwerpunkte von Mynewt:

- Debuggability: Logging, Statistiken, Debug-Scripts
- Security: secure Bootloader, Crypto-Package, Public Key, sha512, x509, AES
- Maintainability: newtmgr-Tool, newtmgr-Protokoll, Shell

Pures C anstatt C++

Lancaster und mbed nutzen hauptsächlich C++ Klassen, während Mynewt ausschließlich auf blankes, klassisches C setzt. Es war lange umstritten, ob sich C++ für den Einsatz in embedded Systemen überhaupt eignet. Der C++-Experte Scott Meyers¹ vertritt inzwischen die Meinung, dass mit etwas Vorsicht C++ auch für Mikrocontroller einsetzbar ist. C++ hat einige Vorteile. Es kapselt zusammengehörige Methoden in einer Klasse, versteckt unnötige Details und hält damit den globalen Namespace sauber. Solche Klassen vereinfachen die Anwendungen erheblich – wenn sie gut entworfen sind. Wenn eine Funktionalität fehlt, fängt entweder das Ableiten an oder die bestehende Klasse wird um eigene Methoden erweitert. Damit wird die Sache komplexer.

In C bleibt alles dem Entwickler überlassen. Er muss selbst mit geeigneten Konventionen dafür sorgen, dass keine Namenskonflikte entstehen. Das Verstecken von Details ist nicht so einfach.

Indem es pures C verwendet, lässt Mynewt dem Entwickler die volle Kontrolle. Namenskonflikte sind über sinnvolle Konventionen minimiert. Die Header-Dateien sind zwar darauf vorbereitet, auch in C++-Quelltext verwendet zu werden, ich habe aber bisher keine einzige C++-Klasse in den Beispielen gefunden.

Damit ist Mynewt in guter Gesellschaft: Der Linuxkern, das wohl am meisten verbreitete Server-Betriebssystem, ist ebenfalls nicht in C++ geschrieben, sondern in "plain old c". Es hätte sicher Gelegenheiten gegeben, den Kern auf C++ zu portieren – aber wozu?

¹ Effective C++ in an Embedded Environment

Kapitel 8 – Installation auf macOS

Zum Erzeugen (=Bauen) eines Mynewt-Programms, sind zwei Komponenten nötig: eine Toolchain für die Zielformatierung und das Commandline-Tool "newt". Die Toolchain besteht aus Compiler, Linker, Assembler, Debugger, diversen Utilities und Include-Dateien. Das newt-Tool ist ein einziges ausführbares Programm. Beide Komponenten müssen erst einmal installiert werden. Für newt gibt es zwei Möglichkeiten: Entweder es wird als fertiges Programm heruntergeladen oder aus dem Quelltext neu kompiliert.

Installation des newt-Tools

Auf einem Mac ist die Installation mit dem Homebrew Packagemanager am einfachsten. Falls Homebrew noch nicht installiert wurde, ist das mit einem Schritt erledigt:

```
$ /usr/bin/ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Listing 35 – Homebrew installieren in einem Mac-Terminal. Alles in einer Zeile eingeben!



Die Installation des Homebrew¹ Paketmanagers auf einem Mac-Rechner ist auch für viele andere Zwecke extrem nützlich. Das Tool "tree" und viele mehr sind hier zu finden.

Sobald die Installation des Paketmanagers beendet ist, reichen die drei folgenden brew-Kommandos aus. Das Erste fügt mit tap² eine neue Paketquelle zur Liste der Standard Quellen hinzu. Dann wird das Verzeichnis aller verfügbaren Pakete aktualisiert und schließlich wird das Paket mynewt-newt aus der neuen Quelle heruntergeladen und installiert.

```
$ brew tap runtimeco/homebrew-mynewt  
$ brew update  
$ brew install mynewt-newt  
==> Installing mynewt-newt from runtimeco/mynewt  
==> Downloading  
https://github.com/runtimeco/binary-releases/raw/master/mynewt-  
newt-tools_1.2.0/mynewt-newt-1.2.0.mavericks.bottle.tar.gz  
==> Downloading from  
https://raw.githubusercontent.com/runtimeco/binary-releases  
/master/mynewt-newt-tools_1.2.0/mynewt-newt-1.2.0.mavericks.
```

¹ https://brew.sh/index_de.html

² tap = englisch für Wasserhahn zeigt den Humor der Entwickler.

Kapitel 9 – Installation auf Windows 10

Von allen in diesem Buch verwendeten Programmen gibt es auch eine Version für Windows-Betriebssysteme. Beim ersten Versuch, die Toolchain zu installieren, bin ich der Anleitung in der Mynewt-Dokumentation¹ gefolgt. Ich konnte danach mit dem newt-Tool meine mydrivertest-App bauen und auf einem angeschlossenen Calliope ausführen. Allerdings wurde die msys2-Umgebung nicht von CLion akzeptiert.

Beim zweiten Versuch installierte ich anstatt der von Mynewt vorgeschlagenen msys2-Shell die umfangreichere MinGW-Shell. Damit funktionierte auch CLion. Die einzelnen Schritte dazu zeigt der folgende Abschnitt:

MinGW Shell für Windows installieren

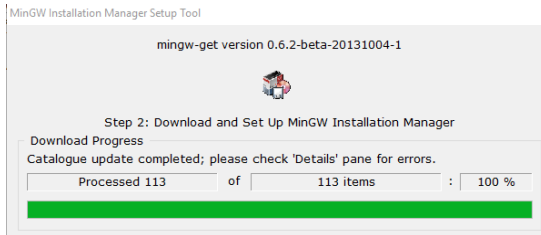
Download von: <http://www.mingw.org/>

Doppelklick auf das heruntergeladene Installationsprogramm: **mingw-get-Set-up.exe**

Klick auf den **Install**-Button

Als Installationsverzeichnis übernehme ich **c:\MinGW**

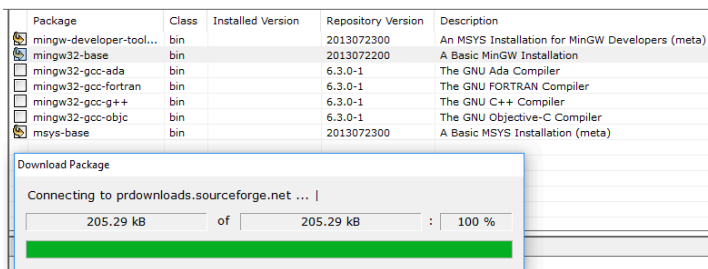
Klick auf den **Continue**-Button



Das Herunterladen der Pakete dauert einige Minuten.

Klick auf den **Continue**-Button

Im Installations-Manager-Dialog aktiviere ich die drei Packages wie in der folgenden Abbildung.



¹ http://mynewt.apache.org/latest/newt/install/newt_windows/

Kapitel 10 – Apps bauen mit dem newt-Tool

Das newt-Tool vereint die Funktionen eines Packagemanagers mit denen eines Build-Managers in einem einzigen Programm. Es stellt damit das zentrale Tool für alle Arbeitsschritte dar, die bei der Entwicklung eines Mynewt-Projekts benötigt werden. Von der Initialisierung eines Projekts bis zum Debuggen des Programmes werden alle Aktivitäten mit einem entsprechenden Subkommando ausgelöst.



Das newt-Tool ist vergleichbar mit dem maven-Tool der java-Umgebung, mit dem go-Tool für die Sprache Go, mit npm der Node.js-Umgebung, oder auch mit CocoaPods und Carthage für die Xcode-Entwicklung.

Überblick newt – Subkommandos

newt ohne weitere Parameter in der Shell ausgeführt liefert eine Übersicht aller Subkommandos:

```
$ newt
Usage:
  newt [flags]
  newt [command]
Available Commands:
  build      Build one or more targets
  clean      Delete build artifacts for one or more targets
  create-image Add image header to target binary
  debug      Open debugger session to target
  info       Show project info
  install    Install project dependencies
  load       Load built target to board
  mfg        Manufacturing flash image commands
  new        Create a new project
  pkg        Create and manage packages in the current workspace
  run        build/create-image/download/debug <target>
  size       Size of target components
  sync       Synchronize project dependencies
  target     Commands to create, delete, configure, and query targets
  test       Executes unit tests for one or more packages
  upgrade    Upgrade project dependencies
  vals       Display valid values for the specified element type(s)
  version    Display the Newt version number
```

Listing 45 – Die Liste der verfügbaren Subkommandos

Kapitel 11 – CLion + newt: ein gutes Gespann

Ich beschreibe in diesem Kapitel nur das nötigste zum:

- Installieren der CLion IDE
- Vorbereiten eines Projekts mit dem newt-Tool
- Konfigurieren des Debuggers gdb
- Starten einer Debug-Session
- Beenden der Debug-Session.

Ein ausführlicher Abschnitt im Anhang ab Seite 327 gibt weitere Tipps&Tricks zur Bedienung der CLion-IDE.

Mynewt-Projekt für CLion vorbereiten

Das vorherige Kapitel zeigte, wie das newt-Tool viele Standardaufgaben der Entwicklung vereinfacht. Die Versions- und Konfigurationsverwaltung ist darauf ausgelegt, möglichst kompakte Programme zu erzeugen. Das wird erreicht durch das Ausblenden aller nicht gebrauchten Code-Teile. Die Editier- und Navigationsmöglichkeiten einer guten IDE werden dadurch aber nicht überflüssig. CLion ist hierfür ein sehr gutes Beispiel. Allerdings braucht CLion eine vollständige Liste aller Quelltext-Dateien und Include-Verzeichnisse, um die Code-Navigation optimal zu unterstützen. Die Mynewt-Version 1.2 hat ein neues Kommando, um alle diese Projektinformationen zu exportieren.

```
$ newt target cmake <Target>
```



Das cmake-Subkommando erzeugt derzeit noch eine Datei, die Pfade im Windows-Format enthält und CLion kann diese Pfade nicht importieren. Mit dem folgenden Kommando können diese Pfade korrigiert werden:

```
$ sed -i "s/\\\\\\\\\\\\\\\\/\\\\/g" CMakeLists.txt
```

Ein großer Vorteil dieser Export-Funktion ist, dass hierdurch das Bauen direkt in der IDE ermöglicht wird. Ein Fehler, den der Compiler erst beim Build findet erscheint im unteren Bereich als Link. Ein Klick darauf führt direkt zur fehlerhaften Stelle im Quelltext.



Auch hier hat die Windows-Umgebung einige Probleme, die unter macOS nicht auftauchen. Es können zwar die C-Dateien des Mynewt-OS kompiliert werden, es gibt aber einige Dateien im Package mynewt_nordic, die zu einem Fehler führen. Die Navigation in CLion funktioniert aber trotz dieses Problems und erleichtert die Entwicklung enorm.

Kapitel 12 – Die App mydrivertest

Software-Entwicklung ohne einfache Test-Möglichkeiten für neu eingebaute Funktionen ist nicht effektiv und kann sehr frustrierend sein. Bei der Entwicklungsmethodik Test-Driven-Development (TDD) kommt die Entwicklung der Tests sogar noch vor der Entwicklung der eigentlichen Komponenten. Die hier vorgestellte App¹ ist der Testrahmen für die von mir entwickelten Treiber. Diese App habe ich parallel mit jedem zusätzlichen Treiber um die entsprechenden Testfunktionen erweitert.



Wenn ich einen neuen Treiber oder ein eigenständiges Package erstellen will, beginne ich typischerweise mit einer neuen Datei `xxx_command.c`, in der ich einen Testrahmen für das neue Package vorbereite. Dann kopiere ich ein ähnliches Package und ändere die Namen, um mit möglichst wenig Aufwand die geforderte Package-Struktur mit `pkg.yml`, `syscfg.yml`, `include` und `src` Verzeichnis zu erhalten.

Die hier benutzten Treiber sind nicht im Core-Repo enthalten, sondern in meinem öffentlichen Repo auf GitHub abgelegt. Der folgende Abschnitt zeigt, wie einfach solch ein externes Repository in ein Projekt importiert werden kann.

Ein externes Repository einbinden

Es mag überraschen, dass die Standard-Auslieferung des Mynewt-OS keinen Treiber für Buttons oder die LED-Matrix des BBC micro:bit enthält. Das zeigt einerseits, wie neu dieses Mynewt OS ist, und verweist andererseits auf die Ursprünge seiner Entwicklung: Für die Steuerung von Straßenlaternen und Smart-Metern waren keine Buttons für eine direkte Bedienung durch einen Benutzer vorgesehen.

In meinem Repository auf GitHub sind – unter anderem – Treiber für die beiden Buttons und die LED-Matrix des Calliope enthalten.

Wie im *Kapitel 8 – Installation auf macOS auf Seite 90* schon beschrieben, stehen in einem neu angelegten Projekt nur die Packages des Core-Repositories zur Verfügung. Alle diese Quelltexte liegen im Repository `apache/mynewt-core` auf GitHub.com.

In der Datei `project.yml` können weitere Repositories eingetragen werden, die genau so behandelt werden wie das Core-Repo. Um meine Packages mit den Erweiterungen für den Calliope Mini nutzbar zu machen, füge ich als ersten Schritt die dunkler markierten Zeilen in `project.yml` ein:

```
project.name: "my_project"

project.repositories:
  - apache-mynewt-core
```

¹ Manchmal bezeichne ich eine App auch als Programm oder Firmware.

Kapitel 13 – Das Seedstudio Grove-System

Seedstudio ist eine Maker-Fab¹ in China, die bereits seit 2008 allerlei Komponenten für Elektronik-Interessierte herstellt und übers Internet anbietet. Diese Firma betreibt auch eine Produktionsstraße zur Herstellung und Bestückung von gedruckten Schaltungen. Ein YouTube Video zeigt eine Besichtigung dieser Platinen-Fabrikation².

Mit dieser Anlage produziert Seedstudio eine breite Palette kleiner PCBs, die häufig nur ein Breakout von SMD-Komponenten darstellen. Das heißt, die Kontakte eines winzigen SMD-Chips sind entweder direkt oder zusammen mit weiteren Komponenten auf eine Grove-Buchse geführt. Wenn man bedenkt, dass ein Farbsensor nur 1.5 x 1.5 mm groß ist, wird verständlich, dass nicht jeder solche winzigen Komponenten löten will. Ich traue mir das jedenfalls nicht zu.

Alle diese Platinchen nutzen eine einheitliche Buchse mit vier Kontakten. Die entsprechenden Steckverbinder gibt es in zwei Ausführungen: mit und ohne Verriegelung.

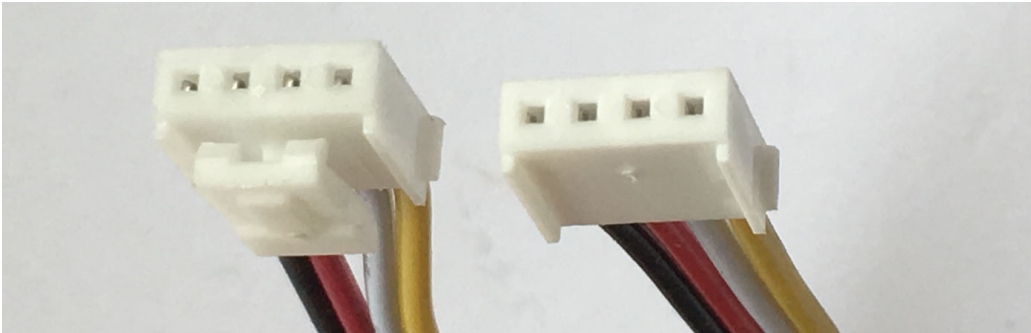


Abbildung 24 – Grove-Stecker: links mit Verriegelung, rechts ohne

Beide Varianten sind so geformt, dass sie nicht verkehrt herum in die Buchse gesteckt werden können.³ Jedem Grove-Modul liegt ein Verbindungskabel mit den einfachen Steckern bei. Die Kabel gibt es jeweils in Fünferpacks in den Größen 5cm, 20cm, 30cm und 50cm mit Verriegelungs-Steckern. Vor dem Herausziehen aus der Buchse muss ein kleiner Hebel zusammengedrückt werden, um die Verriegelung zu lösen. Zwei dieser Kontakte sind immer für Ground (GND) und Stromversorgung (VCC) reserviert. Die Farben der Leitungen sind folgendermaßen festgelegt:

- | | |
|----------------|-------|
| • Gelb | Pin 1 |
| • Weiß | Pin 2 |
| • Rot(VCC) | Pin 3 |
| • Schwarz(GND) | Pin 4 |

¹ Eine Fabrik für Maker, in diesem Fall zur Produktion von Platinen in Kleinserien

² https://youtu.be/hjK_LS3EDo4

³ Vorausgesetzt, man wendet keine rohe Gewalt an.

Kapitel 14 – Interessante Chips und Boards

Manchmal erwähnen Bücher, Zeitschriften oder Blog-Artikel interessante Schaltkreise oder auch kleine Breakoutboards, die es nicht als Grove-Modul gibt, oder die sehr viel preiswerter selbst mit einer Grove-Schnittstelle ausgerüstet werden können. Wenn es sich um ein Board mit I2C-Bus handelt, sind nur die vier Leitungen zu stecken oder zu löten. Im Grove-Programm gibt es dazu Verbindungskabel mit einem Standard-Grove-Stecker am einen Ende und einzelnen Steckbuchsen an jedem der vier Leitungsenden. Diese Kabel sind ideal geeignet, wenn das Board bereits mit vier Pins bestückt ist. Im Fünferpack kosten diese Adapterkabel bei EXP-Tech ca. 4,50€¹.

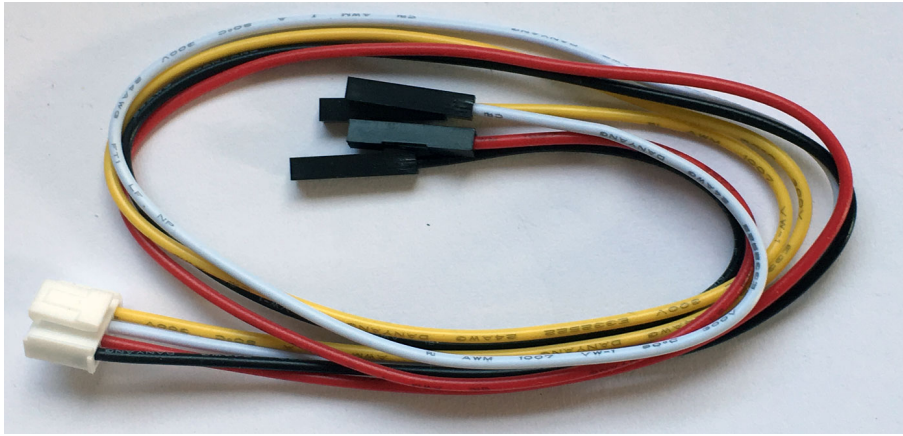


Abbildung 31 – Adapterkabel Grove-Stecker zu 4 x Female-Jumper

Oft sind die Stiftleisten noch nicht eingelötet und liegen nur lose in der Packung. Dadurch können die Pins beim Versand nicht verbogen werden und der Käufer kann selbst entscheiden, ob er Stecken oder Löten will. In solchen Fällen schneide ich ein Grove-Kabel in der Mitte auseinander und erhalte damit zwei Adapterkabel. Die vier Leitungen löte ich an die Kontakte des Boards. So kostet eine "Aufrüstung" zum Grove-Modul nur ca. 50 Cent. Einige der folgenden Boards sind auf diese Art zum Grove-Modul aufgewertet worden.

¹ Stand Oktober 2017, → Bezugsquellen im Anhang

Kapitel 15 – Projekt Lügendetektor – stresstest-App

Die bisherigen Kapitel dienten der Vorbereitung auf die nun folgenden "echten" Projekte. Alle diese Projekte haben ein praktisch anwendbares Gerät als Ergebnis. Diese Geräte beginnen mit einer minimalen Ausbaustufe, in der die Grundfunktion bereits nutzbar ist. Je nach Einsatz von weiteren Komponenten des Calliope-Boards oder dem Anschluss externer Module können die Geräte erweitert oder an andere Bedürfnisse angepasst werden. Das ist der große Vorteil eines modular aufgebauten Systems. Im Falle dieses ersten Projekts könnten diese Varianten folgendermaßen aussehen:

- **Hautwiderstand mit GSR-Modul messen und den Wert auf Konsole ausgeben**

Das ist die minimale Ausbaustufe, die hauptsächlich zum Testen des Grove-Moduls und des ADC-Treibers dient.

- **Hautwiderstand messen mit Feedback über die Tonhöhe des Buzzers**

Mit Hilfe des Sound-Treibers kann je nach gemessenem Hautwiderstand die Tonhöhe geändert werden. Je niedriger der Hautwiderstand, also je größer der Stresslevel, desto höher die Tonfrequenz.

Den Versuch mit diesem Ton-Feedback habe ich schnell abgebrochen. Der Ton ist zu grell, über längere Zeit kaum zu ertragen. Die entsprechende Stelle im Quelltext ist deshalb auskommentiert.

- **Feedback über die LED-Matrix**

Die Funktion `showInt_0_25(uint8_t value)` des LED-Matrix-Treibers zeigt Zahlenwerte im Bereich 0-25 an. Je nach übergebenem Wert leuchten entsprechend viele LEDs. Dies kann als einfaches Feedback für den Hautwiderstand genutzt werden.

- **Feedback über die Farbe der RGB-LED**

Für ein optisches Feedback bietet sich die RGB-LED an. Grün passt für hohe Widerstandswerte, rot für niedrige.

- **Feedback über das Grove-Modul „LED-Balken“**

Wenn das Grove-Modul der LED-Matrix zur Verfügung steht, kann die Anzahl der leuchtenden LEDs, also der Ausschlag des Balkens als Feedback dienen.

- **Feedback über Bluetooth und iOS-App**

Die in Kapitel 20 – auf Seite 266 – beschriebene App, kann auch den Hautwiderstand als fortlaufende Grafik in der Bluefruit App anzeigen.

- **Feedback über Steuerung der Raumbelichtung, und, und, und**

Die Möglichkeiten des Feedbacks sind fast unbegrenzt.

Dieses erste Projekt zeigt die grundsätzliche Konstruktion, die den meisten Mynewt-Projekten gemeinsam ist:

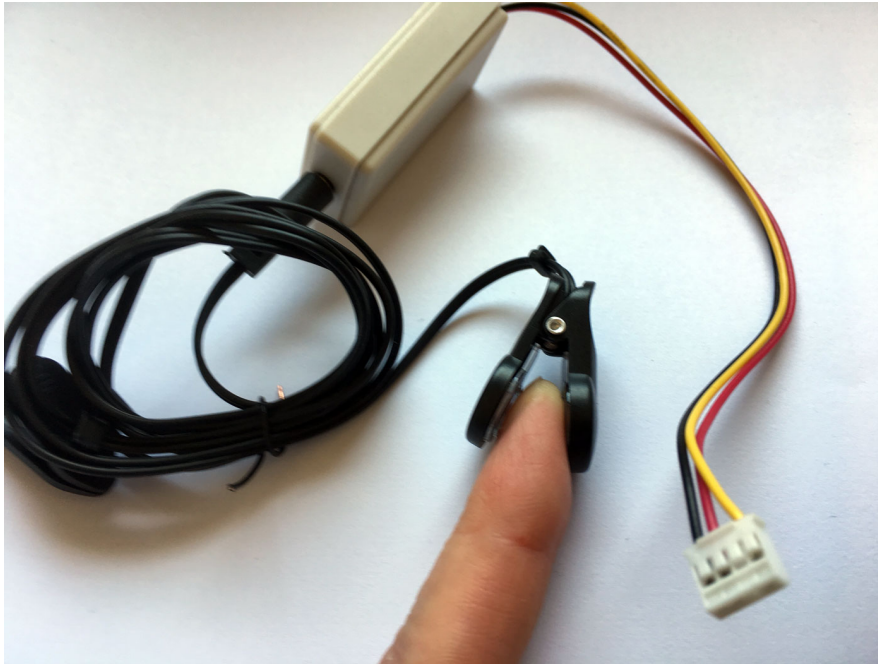
- Ein Sensor wird über die Grove-Buchse angeschlossen.
- Der zugehörige Treiber wird in `pkg.yml` und `syscfg.yml` konfiguriert

Kapitel 16 – Projekt Herzschlag-Monitor

Die Messung der Pulsrate ist eine interessante Anwendung für Sportler oder Meditierende. Die einen wollen in einem bestimmten Puls-Bereich trainieren, um die Fettverbrennung zu optimieren. Die anderen streben einen möglichst harmonischen Puls mit regelmäßiger Variation der Pause zwischen zwei Schlägen an. Inzwischen werden viele Armbanduhren angeboten, die in Zusammenarbeit mit Smartphones den Puls messen und aufzeichnen können. Geräte für das Feedback der Herzvariabilität (HRV) sind noch relativ teuer (> 100 €) und setzen oft die Verbindung mit einem Rechner oder einem Smartphone voraus.

Sseedstudio hat mehrere Module für diesen Zweck in seinem Sortiment, auch ein Kickstarter Projekt bietet einen brauchbaren Sensor auf seiner Webseite¹ an.

Grove Ohrclip Herzfrequenz Sensor – GRV HEART RATE3



Dieses Modul soll ein digitales Signal im Takt des Herzschlags liefern. Die Anleitung liefert zwar den Quelltext für ein Arduino-Programm. Sie gibt aber keinerlei Hinweise, wie der Sensor arbeitet. Weil er der Preisgünstigste der drei Grove-Module war, habe ich ihn ausgewählt für ein Beispielprogramm zum Thema Pulsmessung. Das stellte sich nach einigen Experimenten als Fehlkauf heraus. Nachdem ich eine Version mit 40 Abfragen des Digitalstatus pro Sekunde erstellt hatte, zeigte der Testlauf sehr unregelmäßige Signale. Ein Puls war darin nicht zu erkennen.

¹ <https://pulsesensor.com>

Kapitel 17 – Schaltschwelle der Digital-Eingänge bestimmen

Die grafische Anzeige von analogen und digitalen Daten mit dem Arduino-Plotter brachte mich auf die Idee, dieselbe Spannung sowohl digital als auch analog darzustellen. Auf diese Art würde ich die Schwellenspannung für den HIGH- und LOW-Pegel des Digitaleingangs direkt erkennen können.

Die Verdrahtung für dieses Experiment ist sehr einfach mit dem Grove I2C Expander zu lösen: Ich habe einfach mit einem Female-Female-Verbindungskabel den Pin1 mit dem Pin2 verbunden. Beide liegen also an derselben Eingangsspannung, die mit dem Potenziometer zwischen 0 und ca. 3.2 V eingestellt werden kann.

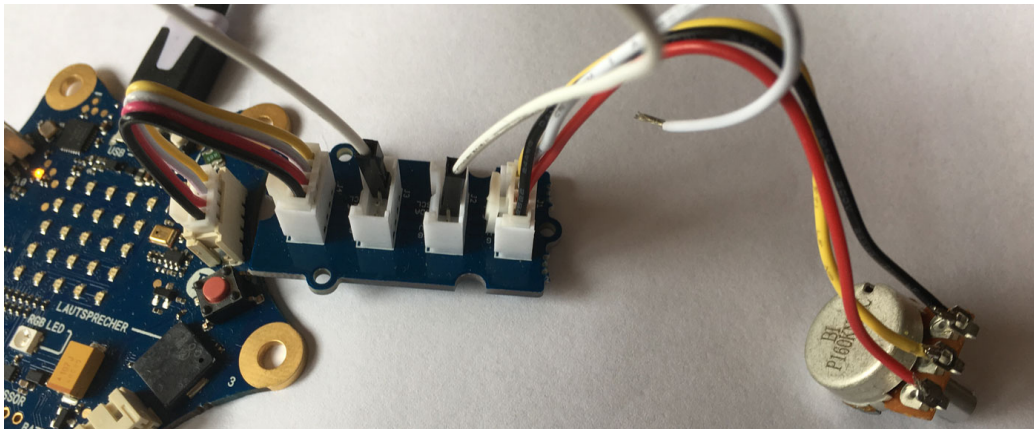


Abbildung 48 – Pin1 und Pin 2 der Grove-Buchse sind verbunden

Als Ausgangsbasis für das Programm `anadig`¹ habe ich die App `stresstest` kopiert, die Kalibrierung entfernt und die Funktion `adc_read_event()` wie im folgenden Listing erweitert:

```
int
adc_read_event(struct adc_dev *dev, void *arg, uint8_t etype,
               void *buffer, int buffer_len) {
    int rc = -1;
    int value = adc_nrf51_driver_read(buffer, buffer_len);           //1
    int digital = hal_gpio_read(GROVE_A1_P2) * 1000;                 //2

    if (value >= 0) {
        console_printf("%d %d\n", value, digital);                  //3
    } else {
        console_printf("Error while reading: %d\n", value);
    }
}
```

¹ Der vollständige Quelltext ist wie immer in <https://github.com/schilken/mynewt-calliope.git>

Kapitel 18 – Konfigurierbarer iBeacon

Die bisherigen Projekte sind auch mit einem einfacheren Mikrocontroller wie einem Arduino Uno oder nano realisierbar. Sobald Bluetooth ins Spiel kommt, müssen die AVR-Arduinos passen oder auf externe BLE-Module ausweichen. Diese Bluetooth-Module werden über eine SPI oder UART-Schnittstelle angesprochen. Diese Module sind relativ preiswert (ca. 8 € bei Versand aus Deutschland, 3 € bei Import aus China), allerdings wird der Aufwand für die Schaltung und die Programmierung höher. Der neuere ARM-Arduino GENUINO 101 nutzt wie Calliope und micro:bit einen Mikrocontroller mit ARM-Architektur. Auf diesem Board sind die Sende- und Empfangskomponenten für Bluetooth bereits integriert. Es ist allerdings teurer als der Calliope. Hier zeigt sich: Calliope und micro:bit sind eine gute Alternative zur Arduino-Plattform, wenn Bluetooth zum Einsatz kommt.

iBeacon – ein Beacon von Apple

Die einfachste Bluetooth-App ist ein iBeacon. "Beacon" ist das englische Wort für Funkfeuer oder Leuchtturm. Das "i" vor dem Beacon hat Apple traditionsgemäß für seine Variante der Beacons hinzugefügt. Apple erkannte schon sehr früh die Möglichkeiten von Bluetooth 4.0 (=BLE). Seit 2011 ist in allen iPhones ein BLE-fähiger Funkchip verbaut, aber erst 2013, zusammen mit iOS7, stellte Apple die iBeacons vor.

Ein Beacon sendet ununterbrochen Datenpakete aus, mit denen er sich selbst identifiziert. Das ist eine Grundfunktionalität des BLE-Standards, mit dem ein Peripheral¹ seine Services den Centrals² in der Nähe bekannt macht. Wenn die hierfür gesendeten Standard-Advertising-Pakete eine von Apple vorgegebene Struktur aufweisen, dann wird der Sender als iBeacon erkannt. Jedes Advertising-Paket ist maximal 31 Bytes³ groß. In einem iBeacon-Paket sind drei Anteile frei konfigurierbar. Auch die Pausenlänge zwischen zwei Paketen kann beliebig eingestellt werden. Sie kann je nach Anwendung eine Sekunde oder auch Minuten betragen. Die Lebensdauer der Batterie wird von der Länge dieser Pausenzeit bestimmt.

NimBLE, der Bluetooth-Stack von Mynewt, kann eine Advertising-Struktur passend für iBeacons mit der folgenden Funktion initialisieren:

```
int  
ble_ibeacon_set_adv_data(void *uuid128, uint16_t major, uint16_t minor);
```

Die Funktionsargumente uuid128, major und minor sind die drei konfigurierbaren Parameter eines iBeacons. Die UUID identifiziert den "Besitzer"⁴ des iBeacons. Diese 128-bit-Zahl wird typi-

¹ z.B. ein Temperatursensor


² in diesem Fall das iPhone

³ Der BLE-Standard 5.0 hat es vergrößert auf 255 Bytes

⁴ Beispielsweise Apple, McDonalds, Buchhandelskette Thalia

Kapitel 19 – Die App bleadc

Im apps-Verzeichnis des apache-mynewt-core Repos liegen die Quelltexte von mehr als 25 Programmbeispielen. Einige Namen beginnen mit dem Vorspann "ble" und das signalisiert, dass es sich hierbei um eine Bluetooth-Anwendung handelt. Nach dieser Konvention habe ich auch die Beispiel-App für dieses Kapitel benannt. Der Name beginnt mit "ble", weil Bluetooth im Spiel ist. Er endet mit "adc", weil ein ADC, ein Analog-Digital-Converter, die Spannung an einem der GPIO-Pins misst. Dieses Programm kann als Basis für viele unterschiedliche Anwendungen dienen. Je nachdem, was die gemessene Spannung bedeutet, kann eine Temperatur, ein Hautwiderstandswert, eine Muskelspannung, eine Lichtstärke oder auch der aktuelle Ladezustand des Akkus gemessen und über Bluetooth bereitgestellt werden.



Es gibt zwar acht Bluetooth-Beispiele, aber wegen des knappen Speichers laufen nicht alle auf dem Calliope. Mit einigen Tricks zum Reduzieren des Speicherbedarfs ist es aber möglich. (→ Speicherbedarf reduzieren auf Seite 312)

BLE-Grundbegriffe: Central, Peripheral, Service, Characteristic

Im vorangegangenen Kapitel nutzte der iBeacon nur das Advertising des Bluetooth-Standards. Mit den ausgesendeten Datenpaketen macht ein BLE-Peripheral auf sich aufmerksam und liefert auch schon Hinweise auf die angebotenen Services. Die Bluetooth-Special-Interest-Group¹ führt ein Verzeichnis mit knapp 40 Standard-Services². Jeder Standard-Service ist identifiziert mit einer 16-Bit-ID, also einer 4-stelligen Hexadezimalzahl. Sie beginnen mit den beiden Ziffern **18**.

Services, die nicht in dieser Liste aufgeführt sind, müssen mit einer UUID, also mit 128 Bit (32-stellige Hexadezimalzahl) identifiziert werden. Wenn ein BLE-Peripheral einen oder mehrere dieser Services anbietet, enthält das Advertising-Paket neben dem Namen des Absenders auch diese Service-IDs. Eine kleine Auswahl der Standard-IDs zeigt die folgende Tabelle:

Name	ID	Bedeutung/Nutzung
Generic Access	1800	Read-only Informationen des Geräts wie Verbindungsparameter, Gerätename, ...
Immediate Alert	1802	Auslösen eines hörbaren oder sichtbaren Signals
Battery Service	180F	Der Ladezustand der Batterie eines Geräts
Device Information	180A	Herstellerinformationen wie Seriennummer, Gerätetyp, Modell, ...
Heart Rate	180D	Herzfrequenz eines Pulsmessers

¹ Die Bluetooth SIG: <https://www.bluetooth.com>
² <https://www.bluetooth.com/specifications/gatt/services>

Kapitel 20 – Die App ble_uart

Der Bluetooth-Standard mit seiner Hierarchie von Services und Characteristics ist für einfache Anwendungen sehr umständlich. Das hat die Implementierung der drei Characteristics im vorangegangenen Kapitel gezeigt. Die langen UUIDs erschweren das noch zusätzlich. Da wünscht man sich die Einfachheit der Shell, wie sie von der mydrivertest-App geboten wurde. So wie die Shell am USB-Port funktioniert, sollte sie auch über BLE verfügbar sein. Solche COMM-Verbindungen waren mit dem alten Bluetooth-Standard 2.0 möglich und ist in Maker-Anwendungen weit verbreitet.

Bluetooth 4.0 ist allerdings nicht kompatibel mit diesem alten 2.0-Protokoll und die Bluetooth SIG¹ hat auch seltsamerweise keinen Standard-Service für diesen Zweck definiert. Glücklicherweise hat Nordic Semiconductor diese Lücke geschlossen: Schon 2011 hat sie einen UART-Service definiert, der wegen seiner Nützlichkeit weite Verbreitung gefunden hat.

Es gibt mehrere Bluetooth-Apps für iOS und Android, die nach diesem Service suchen und eine serielle Verbindung zu ihm aufbauen können. Eine empfehlenswerte App kommt wieder von Adafruit: Bluefruit. Nordic Semiconductor bietet auch mehrere Apps mit dieser Funktionalität.

Die hier vorgestellte App ble_uart liegt in zwei Ausbaustufen vor. Die erste Stufe verbindet die USB-Konsole mit dem UART-Service. Wenn beispielsweise Bluefruit eine Verbindung mit ble_uart herstellt, denn erscheinen die von Bluefruit gesendeten Eingaben im CoolTerm-Fenster. Umgekehrt zeigt Bluefruit die Antworten, die im CoolTerm eingegeben werden. Auf diese Art ist also ein bidirektionaler Chat zwischen iPhone und Mac über Bluetooth möglich.

Ein Chat über Bluetooth

NimBLE liefert mit dem Package net/nimble/host/services/bleuart das Grundgerüst für einen Service, der eine bidirektionale, serielle Kommunikation ermöglicht. Die Datei bleuart.c dieses Packages initialisiert einen Service mit den von Nordic definierten UUIDs, und konfiguriert zwei Characteristics wie folgt:

```
static const struct ble_gatt_svc_def gatt_svr_svcs[] = {
    {
        /* Service: uart */
        .type = BLE_GATT_SVC_TYPE_PRIMARY,
        .uuid = &gatt_svr_svc_uart_uuid.u,
        .characteristics = (struct ble_gatt_chr_def[]) { {
            .uuid = &gatt_svr_chr_uart_read_uuid.u,
            .val_handle = &g_bleuart_attr_read_handle,
            .access_cb = gatt_svr_chr_access_uart_write,
            .flags = BLE_GATT_CHR_F_NOTIFY,
```

¹ Die Special Interest Group für den Bluetooth Standard

Kapitel 21 – MI-Band fernauslösen – blemib-App

In den bisherigen Apps übernahm der Calliope immer die Peripheral-Rolle. Das heißt: Er bietet Services für andere BLE-Geräte an, die in der Central Rolle arbeiten. In der App dieses Kapitels übernimmt der Calliope die Central-Rolle. Normalerweise sucht ein Central in seiner Umgebung nach einem Gerät, das einen bestimmten Service per Advertising-Paket anbietet, verbindet sich mit ihm und greift auf die angebotenen Characteristics zu.

Die hier vorgestellte App blemib hält nicht nach einem Service Ausschau, sondern sucht nach einem Peripheral mit einer bestimmten Bluetooth-Adresse. Diese Adresse ist mit einem syscfg-Parameter konfiguriert. Wenn sie ein Gerät mit dieser Adresse findet, baut sie eine Verbindung auf, liest alle Services und Characteristics des Peripherals und speichert die gefundenen Details ab. Dann sucht sie in den gespeicherten Daten nach dem Standard-Service **Immediate Alert**:

```
#define BLECENT_SVC_IMMEDIATE_ALERT_UUID    0x1802
#define BLECENT_CHR_ALERT_LEVEL              0x2a06
```

Wenn dieser Service und die zugehörige Characteristic **Alert Level** in der gespeicherten ID-Liste gefunden wird, dann schreibt die App den Wert 2 in die Characteristic. Ein MI-Band¹, das es ab 12 € bei Amazon gibt, implementiert diesen Service. Wenn es in der Nähe des Calliope gefunden wird, wird dieser Schreibzugriff das MI-Band heftig vibrieren lassen. Der Wert 1 führt zu einem leichten Summen.

Die Ausgangsbasis für die vorliegende App war der Quellcode der blecent-App des Core-Repos. In peer.c ist eine Funktionssammlung implementiert, die alle Services und Characteristics eines verbundenen Peripherals scannt und speichert. Ein Peripheral mit einer aktiven Verbindung wird hier als Peer² bezeichnet. Der Prefix peer_ kennzeichnet alle Funktionen dieser Bibliothek.

Mynewt implementiert die Central- und die Peripheral-Rolle in Packages unterhalb von net/nimble im Core-Repository. Die Abhängigkeiten in pkg.yml sehen so aus:

```
pkg.name: apps/blemib
pkg.type: app
pkg.description: Simple BLE central application.
pkg.author: „Apache Mynewt <dev@mynewt.apache.org>“
pkg.homepage: „http://mynewt.apache.org/“
pkg.keywords:

pkg.deps:
  - „@apache-mynewt-core/kernel/os“
```

¹ XIAOMI Mi Band in der Kategorie Sport & Freizeit

² Etwa übersetzbar als gleichartiger Partner

Kapitel 22 – Das newtmgr-Tool

Der Newt-Manager ist ein weiteres Open-Source-Tool des Mynewt-Systems. Es kann entweder als lauffähiges Programm oder im Quelltext heruntergeladen werden. Die Installation auf dem Mac ist mit Homebrew wieder am einfachsten:

```
$ brew upgrade mynewt-newtmgr
. . .
##### 100,0%
==> Pouring mynewt-newtmgr-1.2.0.sierra.bottle.tar.gz
   /usr/local/Cellar/mynewt-newtmgr/1.2.0: 3 files, 17.3MB
```

Der Newt-Manager zeigt seine Subkommandos, wenn er mit dem Parameter **help** aufgerufen wird:

```
$ newtmgr help
Newtmgr helps you manage remote devices running the Mynewt OS

Usage:
  newtmgr [flags]
  newtmgr [command]

Available Commands:
  config      Read or write a config value on a device
  conn        Manage newtmgr connection profiles
  crash       Send a crash command to a device
  datetime    Manage datetime on a device
  echo        Send data to a device and display the
              echoed back data
  fs          Access files on a device
  help        Help about any command
  image       Manage images on a device
  log         Manage logs on a device
  mpstat      Read mempool statistics from a device
  res         Access a CoAP resource on a device
  reset       Perform a soft reset of a device
  run         Run test procedures on a device
  stat        Read statistics from a device
  taskstat    Read task statistics from a device
```

Kapitel 24 – Ausblick

Als dieses Buchprojekt die 300-Seiten-Grenze überschritt, musste ich anfangen streng auszusortieren, was in dieses Buch passte und was ich besser in eine Folgeband verschieben sollte. So entstand die Idee von der vier-teiligen Buchreihe. Einige Projekte, die auf dem micro:bit ebenso gut laufen, wie auf dem Calliope sind in den Band 2 gerutscht. Anhänge, die erst bei der Entwicklung von Miniatur-Bluetooth-Boards wichtig werden, kamen in den Band 3 und alle Details zu den Themen newtmgr, Security und LoRaWan mussten in den Band 4 wandern.

Weitere Projekte mit Grove-Modulen

Eigentlich hatte ich vor, noch weitere Projekte mit Grove-Modulen im vorliegenden Buch vorzustellen. Diese Projekte musste ich aus Platzgründen in den *Band 2: BBC micro:bit – Programmieren mit C und CLion* verschieben.

- **Projekt UV-Index anzeigen**
- **Funksteckdosen über 433-MHz-Sender steuern**
- **Temperaturmessung mit internem oder externem Sensor**
- **Lichtfarbe bestimmen mit Grove-Modul**
- **Treiber für Lichtstärkemessung mit den Onboard-LEDs**
- **Fernauslösen von Kameras mit Infrarot-Sende-LED**
- **TV und Mediaplayer mit Infrarot-Sende-LED fernsteuern**
- **Single_image für umfangreiche Apps nutzen**

Weitere Bluetooth-Projekte

- **ANCS – der Apple Notification Center Service**
Das ist eine App, die anfangs die Peripheral-Rolle annimmt und sich von einem iPhone finden lässt. Nach dem Pairing schaltet die App um in die Central-Rolle, durchsucht die Services des iPhones und verbindet sich mit dem ANCS des iPhones. Das iPhone sendet anschließend alle Apple-Notifications auch an den micro:bit oder den Calliope. So lassen sich entgangene Anrufe auf dem micro:bit signalisieren und vieles mehr.
- **Experimente mit den BLE-Services von Apple Watch und Apple TV**
- **Abfrage eines Herzschlagmonitors über BLE**
- **Weitere Experimente mit dem MI-Band**
- ...

Kapitel 25 – Anhang

Übergangslösung falls newt target cmake nicht funktioniert

Im Abschnitt *Mynewt-Projekt für CLion vorbereiten auf Seite 118* habe ich erwähnt, dass die Erzeugung der CMakeLists.txt Datei mit `newt target cmake <target>` in komplexeren Fällen scheitern kann. Für diesen Fall habe ich eine eigene Variante des newt-Tools erstellt, die zwar weniger Funktionalität bietet, aber dafür robuster ist.

Diese Version kennt die zusätzliche Option `-m` für das build-Subkommando. Vor einem Build mit dieser Option müssen mit `newt clean <target>` alle Zwischenergebnisse des vorherigen Builds gelöscht werden. Damit wird eine neue Übersetzung aller Quelltexte erzwungen¹ und das nachfolgende `newt build -m <target>` schreibt die Namen aller übersetzten C-Dateien sowie aller verwendeten Include-Verzeichnisse in die Datei CMakeLists.txt. Diese Datei enthält allerdings nicht alle Details, die CLion über das Projekt wissen sollte. Das Bauen des Projekts ist deshalb nicht möglich.

Mit wenig Aufwand kann diese modifizierte Version aus dem Go-Quellcode erzeugt werden. Eine kurze Anleitung dazu folgt im nächsten Abschnitt. Es sind nur wenige Shell-Kommandos nötig, denn das Go-Buildsystem macht die Entwicklung solcher Tools besonders einfach. Für interessierte Entwickler lohnt es sich, das go-Kommando auszuprobieren. Dabei zeigen sich auch viele Ähnlichkeiten zwischen den Build-Kommandos `go` und `newt`.



Die Option `-m` funktioniert nicht, wenn `newt` mit einem Fehler abbricht. In diesem Fall kommentiere ich die Fehlerstelle aus, generiere neu, lade in CLion die neue Filelist und editiere dann weiter.

newt selbst bauen

Go gilt als Nachfolger für die 40 Jahre alte Sprache "C". Ken Thompson, einer der drei Väter von "C", hat beim Design der Go-Sprache mitgearbeitet. Die Entwicklung begann 2007 als 20%-Nebenbei-Projekt bei Google und nahm dann aber schnell größeren Umfang an. Seit 2010 setzt Google hauptsächlich Go als Systemsprache für neue Entwicklungen ihrer Backends ein. Damit hat Go die bisherige Systemsprache C++ abgelöst – jedenfalls in den Google-Rechenzentren.

Das newt-Tool ist in dieser relativ neuen Sprache Go programmiert. Der Go-Compiler ist noch nicht so weit verbreitet, dass sie bei macOS oder Linux zur Standardauslieferung gehört. Vor dem Bauen des newt-Tool muss also die Toolchain für Go installiert werden. Dafür gibt es mehrere Möglichkeiten. Zum einen gibt es Installationspakete für macOS, Linux und Windows auf der

¹ Sonst würden hinterher nur die geänderten C-Source in der Datei CMakeLists.txt stehen.

Literaturverzeichnis

- **Adafruit nRF52 Pro Feather with mynewt, <https://learn.adafruit.com> 2017**
Auf knapp 70 Seiten bietet dieses englische PDF-Dokument einen Crash-Kurs zu Mynewt auf dem Adafruit-Board „Pro Feather“.
- **Alasdair Allan, Don Coleman, Sandeep Mistry, Make: Bluetooth, Maker Media 2015**
Das Buch bietet einen guten Einstieg in Bluetooth-LE. Einige praktische Projekte nutzen Arduino mit dem Bluetooth-Modul nRF8001 oder Raspberry Pi mit Bluetooth-Dongle. 230 Seiten, engl.
- **Burkhart Kainka, Micro:bit Praktikum, 2016**
Bis auf ein mbed-, ein MicroPython- und ein PXT-Projekt sind alle Projekte dieses Buchs im Microsoft Block-Editor programmiert. Der PXT war 2016 noch in der Beta-Phase. Neben den Quelltexten bietet das Buch auch interessante Details zur Hardware. Beispielsweise wird der micro:bit zur Messung der Kennlinie von FET-Transistoren verwendet, oder als Oszilloskop. Die Glättung von PWM-Signalen wird anschaulich erklärt. 122 Seiten
- **Burkhart Kainka, Calliope und Micro:bit in der Praxis, 2017**
Dieses Buch enthält Projekte, die auf micro:bit und Calliope laufen. Die meisten sind in PXT programmiert und als Block-Grafik und Javascript abgebildet. Es sind aber auch zwei Projekte für Open Roberta, zwei für MicroPython und drei für mbed-Plattform enthalten. 104 Seiten
- **David Brownell, OpenOCD User's Guide, <http://openocd.org>**
Das Handbuch beschreibt alle Kommandos des OpenOCD-Tools. 172 Seiten, engl.
- **Jürgen Wolf, C von A bis Z, Galileo Press 2009**
Ein ausführliches Handbuch der Sprache C. Auch als kostenloses OpenBook verfügbar: http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/015_c_strukturen_001.htm
800-1100 Seiten je nach Ausgabe, deutsch
- **Elecia White, Making Embedded Systems, O'Reilly 2011**
Das Buch liefert konkrete Empfehlungen aus der Praxis für die Software-Entwicklung von Embedded-Systemen. Es ist sehr zu empfehlen für Fortgeschrittene, die ernsthafte Embedded-Projekte realisieren wollen. 330 Seiten, engl.
- **Harry Fairhead, micro:bit IoT in C, 2016**
Dieses Buch fasst Artikel aus seinem Blog <http://www.iot-programmer.com/> zusammen. Es beginnt mit einer Kurzanleitung für die Entwicklung mit yotta und enthält Details und Code-

Glossar

Die hier gesammelten Erklärungen beziehen sich auf die spezielle Bedeutung des jeweiligen Begriffes im Kontext der Embedded-Software-Entwicklung. Häufig haben Begriffe in einem anderen Kontext auch eine andere Bedeutung. Ein gutes Beispiel ist "advertising", dass hier nur im Zusammenhang mit Bluetooth gesehen wird.

- **Advertising**

Im BLE-Protokoll vorgesehene Bekanntmachung eines BLE-Geräts. Es stehen maximal 31 Bytes für diese als Broadcast gesendeten Datenpakete zur Verfügung.

- **API – Programmier-Schnittstelle**

Im Buch verwende ich das Wort API als gleichbedeutend mit Schnittstelle.

- **ARM-Architektur**

Ein Mikroprozessor mit →RISC Befehlssatz der Firma ARM (Advanced Risc Machines).

- **Attribute**

Oberbegriff für Bluetooth Services, Characteristics und Descriptors

- **Beacon**

Bluetooth Peripheral, das nur Advertising-Pakete aussendet, ohne eine Verbindung zu erlauben.

- **BLE – Bluetooth Low Energy**

Ab Version 4.0 (2009) im Bluetooth Standard definierte besonders energiesparende Protokolle des Industriestandards. Diese BLE-Protokolle sind nicht kompatibel mit den älteren Protokollen (→Bluetooth classic).

- **Bluetooth classic**

Von einem Konsortium seit 1998 entwickelter Standard von Kommunikationsprotokollen. Ziel war der Ersatz der Kabelverbindungen von PC-Peripheriegeräten durch Funk im Nahbereich.

- **Board**

Eine Platine mit einem →Mikrocontroller und Anschlüssen für Ein-/Ausgabepins. Zwei dieser Pins (→SWD) oder ein USB-Anschluss dienen typischerweise als Schnittstelle für das Laden einer →Firmware und fürs Debuggen.

- **Bootloader**

Basiskomponente des Mynewt Betriebssystems. Siehe Abschnitt im Anhang.

- **Breakout-Board**

Kleines Platinchen mit einem aufgelötetem SMD-IC und eventuell zusätzlichen Bauteilen. Besonders nützlich ist solch eine Platine, wenn ein interessanter Chip nur in →SMD-Bauform lieferbar ist. Damit kann auch ein SMD-IC mit einfachen 2.54mm-Lötarbeiten angeschlossen werden.

- **Build – Bauen**

Der Vorgang des Kompilierens und Linkens eines Programms. Manchmal kommen noch weitere Arbeitsschritte dazu. Beispielsweise signiert Mynewt das Binärprogramm noch, oder der Binär-code wird ins textbasierte HEX-Format konvertiert.

IMU	76	Button-Treiber	81, 135ff, 140, 142, 150, 168
#if87		C-Array	157, 251, 274
#ifdef	87	Callback	89, 135ff, 140, 150, 168f, 204, 212, 214, 221, 251ff, 268ff, 274, 280, 286ff
A/D-Wandler	190f	callout	81, 139ff
Accelerator	76	Central	51, 54, 218, 250f, 253, 256ff, 263ff, 267f, 274, 278, 350
Accelerometer	79	cflags	110
Adafruit	15, 31, 132, 176, 184, 189, 192, 266, 275, 342f, 345	Characteristic	250ff, 255ff, 263ff, 270, 278f, 284, 286ff, 350, 354
ADC	25f, 60, 75, 88, 148, 153, 164ff, 191, 200ff, 207ff, 216f, 250, 252, 254ff, 264, 272ff, 320	CLion	-13f, 2ff, 7, 33, 36, 44ff, 95f, 102, 112, 118ff, 126, 128, 299, 301ff, 306, 315, 327, 329, 332f, 335, 338, 345
Advertising	220, 245ff, 258ff, 263, 278, 280f, 283f, 290, 349	cmake	33, 39, 41f, 44, 46f, 118ff, 306, 309
AES	56	CMSIS-DAP	45, 125f, 301, 329, 350
Android	245	CODING_STANDARDS	75, 107
Apache Software Foundation	-10, -8, 2, 54ff, 296, 341, 345	Commandhandler	88
Apple	12, 218f, 245, 258, 275, 316, 351	Commandline	6f, 48, 90, 335
Arduino	-10, -8, 1, 3f, 10ff, 16, 31, 34, 61f, 114, 146, 179, 183, 198, 205f, 210f, 216ff, 275, 319, 341ff, 345	compat	88, 145, 147f, 152, 157, 163, 171, 173f, 176f, 190, 193, 226, 230f, 233ff, 241ff, 264
arm-none-eabi-gdb	92	Compiler	12, 33f, 36ff, 42, 44, 46, 57, 63, 67, 75, 90f, 93f, 106f, 118, 306f, 319, 327, 351f
Assembler-Datei	115	console-API	86
Beschleunigungsmessung	76	console_printf()	87, 144, 151, 169, 215, 238, 242, 271, 313
Bibliothek	3, 11f, 21, 29ff, 36, 42, 48ff, 63, 71, 278, 284, 286, 289, 352	console_read()	86ff, 144, 270
blinky	38, 42f, 59f, 68, 104f, 107ff, 113ff, 124f, 214, 326	console_write()	86f, 215
Bluefruit	146, 200, 266, 276	CoolTerm	93, 101, 144ff, 151, 158, 193, 205f, 241, 244, 263, 266, 291, 297, 299, 315, 329
Bluetooth-Adresse	278, 280, 283ff	Cortex	-10, 12f, 15f, 60, 62, 115, 179, 341, 350, 352f
Bluetooth-Stacks	29, 258, 280	DAPLink	92, 123
bme280	75	Debug-Session	118, 122, 124, 126ff, 299, 329, 331, 333ff
BMX055	21, 36, 163	Debuggability	55f, 112
bno055	75	Discovery	258, 279ff, 286ff, 290f
Bootloader	5, 21, 34, 53, 56, 69ff, 73, 116, 131, 302, 320, 324ff, 349	DRV8837	21, 24
Bosch	18, 21, 36, 75f, 163	DS1307	191f
Breakout	178, 184, 186, 197ff, 345, 349	DS3231	192
Breakpoint	3, 54, 112, 125f, 128, 298f, 301, 327, 329ff	Eclipse	3, 5, 12, 46, 48
brew	90	Eddystone	245ff
Brian Kernighan	57	EEPROM	184, 192f
build_profile	110	Elecia White	347
Burkhard Kainka	25	elektronik-labor.de	25, 346