LEHRBUCH

Klaus Dembowski

Raspberry Pi – Das technische Handbuch

Konfiguration, Hardware, Applikationserstellung

2. Auflage

Springer Vieweg

Raspberry Pi – Das technische Handbuch

Klaus Dembowski

Raspberry Pi – Das technische Handbuch

Konfiguration, Hardware, Applikationserstellung

2., erweiterte und überarbeitete Auflage

Springer Vieweg

Klaus Dembowski
Hamburg, Deutschland

ISBN 978-3-658-08710-4 ISBN 978-3-658-08711-1 (eBook)
DOI 10.1007/978-3-658-08711-1

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

Springer Vieweg
© Springer Fachmedien Wiesbaden 2013, 2015
Die 1. Auflage ist unter dem Titel „Raspberry Pi – Das Handbuch“ erschienen.
Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.
Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften. Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen.

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Fachmedien Wiesbaden ist Teil der Fachverlagsgruppe Springer Science+Business Media
(www.springer.com)

Vorwort zur 2. Auflage

Der Raspberry Pi begeistert. Selten hat eine Schaltung so viel Aufmerksamkeit und Enthusiasmus erregt und das nicht nur bei Systementwicklern, sondern auch bei Hobby-Programmierern und Elektronik-Bastlern weltweit, so dass mittlerweile über drei Millionen Stück davon verkauft worden sind. Ausgehend von dem ursprünglichen Modell B gibt es mittlerweile fünf verschiedene Raspberry Pi-Typen, die in diesem Buch entsprechend behandelt werden.

In der Vergangenheit gab es bereits ähnliche Mini-Computer, und es werden auch weiterhin neue entwickelt. Gleichwohl ist der Raspberry Pi für das, was er zu leisten vermag, sowie funktionstechnisch und preislich gesehen ein äußerst interessantes Produkt. Für ca. 25 € kann man weder die dazugehörige Platine herstellen noch die darauf verwendeten Bauelemente einzeln kaufen und erst recht nicht das System bestücken und testen.

Neben dem preislichen Aspekt spielt seine universelle Verwendbarkeit eine große Rolle, die durch den Einsatz von Linux als Betriebssystem sichergestellt wird, wodurch kaum Berührungsängste bei Computeranwendern entstehen. Im Gegensatz zu leistungsfähigeren Systemen, die ebenfalls mit einem ARM-Prozessor arbeiten, sind für die Applikationserstellung mit dem Raspberry Pi keine Profikenntnisse weder bei der Programmierung noch beim Schaltungsaufbau notwendig.

Generell eignet sich der Raspberry Pi aufgrund des Open-Source-Betriebssystems Linux und der Schnittstellenvielfalt hervorragend für die Systementwicklung. Obwohl es die ursprüngliche Intention der Raspberry Pi-Entwickler an der Universität in Cambridge war, den Studenten sowie allen Interessierten ein kostengünstiges System an die Hand zu geben, um ihnen das Programmieren von Computern näher zu bringen, kann der Raspberry Pi durchaus in Profianwendungen (Embedded Systems) bestehen. Hierfür gibt es bereits einige Industrielösungen, beispielsweise für die Hausautomation.

Außerdem kann der Raspberry Pi auch ohne spezielle elektronische Schaltungen und Programmierarbeiten für typische Computeranwendungen wie Office, Internet oder hochauflösendes Video und als Plattform für den Aufbau von Mediacentern oder Drucker- und NAS-Server eingesetzt werden. Zu beachten ist dabei, dass der Raspberry Pi weder ein PC, noch ein Notebook und auch kein Smartphone ist, was mitunter vor lauter Begeisterung etwas aus dem Blickfeld gerät, etwa, wenn der Raspberry Pi den heimischen PC ersetzen oder als Player und Recorder für HDTV oder auch als NAS-Server im Windows-Netzwerk fungieren soll. Es ist

wichtig, dass die limitierten Ressourcen des Raspberry Pi stets mit beachtet werden.

In diesem Handbuch geht es darum, den Raspberry Pi zu verstehen und ihn dementsprechend optimal einsetzen zu können. Dabei wird darauf Wert gelegt, dass das Zusammenspiel der Software mit der Hardware deutlich wird.

Der Schwerpunkt des Buches liegt auf der Hardware mit den Schnittstellen (LAN, WLAN, GPIO, seriell SPI, I^2C), die für die Kommunikation und den Datenaustausch mit anderen Einheiten prädestiniert sind. Es geht weniger darum, Linux oder eine bestimmte Programmiersprache zu erlernen oder um den Einsatz spezieller Peripherie. Dies sind gewissermaßen alles nur »Hilfsmittel«, um auf dieser Basis die Verwirklichung eigener Applikationen mit dem Raspberry Pi zu ermöglichen.

Geesthacht, im Januar 2015

Klaus Dembowski

Hinweis für die E-Book-Ausgabe: Die beiden Umschlagseiten mit der Schnellübersicht der Raspberry Pi-Befehle und die Signalbelegungen des GPIO-Ports stehen auf der Internetseite zum Buch unter http://www.springer.com/978-3-658-03166-4

Inhaltsverzeichnis

Vorwort zur 2. Auflage V

Inhaltsverzeichnis VII

Einführung 1

Wie es begann 2
Arduino 5
Mikrocontroller und Mini-Computer 7

1 Schnellstart 13

1.1 Anschließen 16
1.1.1 Netzteil 16
1.1.2 Tastatur und Maus 18
1.1.3 Monitor 19
1.2 Einschalten und booten 21
1.3 Grundlegende Konfigurierung 27
1.3.1 Information – About raspi-config 28
1.3.2 Kapazitätsbeschränkung aufheben – Expand Filesystem 28
1.3.3 Password ändern – Change User Password 28
1.3.4 Desktop automatisch starten – Enable Boot to Desktop/Scratch 29
1.3.5 Spracheinstellungen – Internationalisation Options 29
1.3.6 Nationale Zeichensätze – Set Locale 29
1.3.7 Gebiet und Zeitzone – Set Timezone 30
1.3.8 Tastatureinstellungen – Set Keyboard Layout 31
1.3.9 Kamera aktivieren – Enable Camera 32
1.3.10 Registrieren – Add to Rastrack 32
1.3.11 Übertakten – Overclock 32
1.3.12 Erweiterte Optionen – Advanced Options 33
1.3.13 Monitorabstimmung – Change Overscan 34
1.3.14 Computername – Hostname 34
1.3.15 Speicheraufteilung – Set Memory Split 34
1.3.16 Secure Shell aktivieren – SSH Enable 35
1.3.17 Schnittstellen – SPI, I²C, Serial 36
1.3.18 Tonausgang bestimmen – Audio 37
1.3.19 Aktualisierung – Update 37
1.4 New Out Of Box Software – NOOBS 38

2 Software ... 41

2.1 Dateisystem und erste Software-Installation ... 41
2.2 Verzeichnisstruktur ... 45
2.3 Linux-Orientierung und Befehle ... 48
2.4 Zugriffsrechte ... 53
2.5 Verwaltung und Paketmanager ... 55
2.6 Firmware ... 58
2.6.1 Bootvorgang – Firmware und Kernel ... 58
2.7 Aktualisierung – Updates ... 59

3 Hardware ... 63

3.1 ARM-Prozessor BCM2835 ... 63
3.2 ARM-Architektur ... 64
3.2.1 Cores und Typen ... 66
3.3 Speichereinheiten ... 71
3.3.1 SD Cards ... 73
3.4 Grafikeinheit ... 77
3.4.1 HDMI und DVI ... 78
3.4.2 Composite Video ... 82
3.5 Audio ... 84
3.6 Ethernet und USB ... 84
3.6.1 LAN9512 ... 86
3.6.2 PHY und MAC ... 86
3.6.3 Netzwerkverbindung ... 89
3.6.4 TAP- und USB-Controller ... 89
3.6.5 Polyfuses ... 92
3.7 Spannungsversorgung und Taktung ... 92
3.7.1 Taktung ... 95
3.8 Reset-Schaltung ... 95
3.9 DSI- und CSI-Schaltung ... 97
3.9.1 Pi Camera ... 100
3.10 Compute Module ... 101
3.10.1 Spannungsregler ... 104
3.10.2 Embedded Multimedia Card beschreiben ... 105
3.11 Die +Modelle ... 108
3.12 General Purpose Input Output ... 111
3.12.1 Weitere GPIO-Ports der Revision 2 ... 117
3.12.2 GPIO-Port der +Modelle ... 118
3.12.3 GPIO-Port des Compute Module ... 120

4 Konfigurierung und Optimierung 123

4.1 Betriebssysteme 123
4.2 Systeminstallation 125
4.3 Audio aktivieren und einsetzen 127
4.4 Videoplayer und Lizenzen 132
4.5 Mediacenter 133
4.6 Externe Laufwerke einsetzen 137
4.7 Drucken 141
4.8 Netzwerkverbindungen und -Einstellungen 144
4.8.1 Dynamic Host Configuration Protocol – DHCP 146
4.8.2 Übersicht und Analyse 146
4.8.3 Netzwerkadressen 147
4.8.4 Konfigurationsdatei anpassen 150
4.8.5 Adressenumsetzung – Domain Name Service 151
4.8.6 Verbindungsverfahren 152
4.8.7 Secure Shell – SSH 153
4.8.8 Virtual Network Computing – VNC 155
4.8.9 File Transfer Protocol – FTP 157
4.9 WLAN 160
4.9.1 Standards und Kompatibilität 161
4.9.2 Topologien 162
4.9.3 Raspberry Pi für das WLAN konfigurieren 165

5 Programmierung 171

5.1 Hardware-nahe Programmierung 173
5.1.1 Assembler 173
5.1.2 Turbo Pascal 176
5.2 Skriptsprachen 177
5.3 Java 178
5.4 Microsofts .NET 179
5.4.1 Mono 179
5.5 Standard Tools auf dem Desktop 180
5.6 Programmieren mit Python 182
5.7 Programmieren mit C 188

6 Schnittstellen-Kommunikation 193

6.1 OnBoard-LED ansteuern 193
6.1.1 Trigger 194
6.1.2 Heartbeat 194

6.1.3 Mit Brightness schalten ... 195
6.1.4 Python-Programm ... 196
6.2 Einsatz des GPIO-Ports ... 197
6.2.1 Software ... 199
6.2.2 Kernel GPIO-Unterstützung ... 201
6.2.3 GPIO mit Python ... 203
6.3 Serielle Schnittstelle ... 205
6.3.1 Verbindungen ... 206
6.3.2 RS232-Treiber ... 208
6.3.3 Kommunikationsbeispiel ... 209
6.4 Serial Peripheral Interface – SPI ... 213
6.4.1 Chip-Kommunikation ... 215
6.4.2 Linux-Treiber und Anwendung ... 217
6.4.2.1 Blacklist ... 217
6.4.2.2 Systemzugriff mit Input/Output Controls ... 218
6.4.2.3 Python-Bibliothek ... 219
6.4.3 A/D- und D/A-Wandlerschaltung ... 220
6.5 Inter Integrated Bus – I²C ... 228
6.5.1 Betriebsarten ... 228
6.5.2 Bus-Kommunikation ... 230
6.5.3 Adressen ... 233
6.5.4 Programmierung ... 235
6.5.5 Applikation ... 238
6.5.5.1 Real Time Clock ... 242
6.5.5.2 A/D- und D/A-Wandler ... 248
6.5.5.3 LCD-Ansteuerung ... 253
6.5.5.4 Temperatursensor ... 254
6.5.5.5 I/O-Port ... 256

7 Low Power-Betrieb ... 261

7.1 Optimierung des Stromverbrauchs ... 261
7.1.1 Betriebssystem ... 263
7.1.2 Embedded System ... 264
7.1.3 Einstellung der Taktfrequenzen ... 266
7.1.4 Spannungseinstellung ... 269
7.1.5 Austausch des Linearreglers ... 270
7.2 Mobile Spannungsversorgungen ... 274
7.2.1 Spannungsversorgungoptionen ... 274
7.2.1.1 Betrieb mit 3,3 V ... 275

7.2.2 Batterien und Akkus 276
7.2.3 Wirkungsgrad und Laufzeit 277
7.2.4 Power Packs 278

Stichwortverzeichnis 281

Einführung

Der Raspberry Pi ist ein kleines preiswertes System, welches aus einer Platine von ca. 9 x 6 cm besteht, auf der sich alle notwendigen Komponenten befinden, um es als Computer verwenden zu können. Es ist lediglich noch eine SD-Karte als Speichermedium einzusetzen und ein Netzteil für die Spannungsversorgung anzuschließen. Je nach Einsatzzweck kann dies bereits die minimale Ausstattungsform dieses Single Board-Computers sein, wenn das System zuvor entsprechend für eine bestimmte Aufgabe programmiert worden ist, was gemeinhin unter einem *Embedded System* verstanden wird.

Der Raspberry Pi ist ein kleines Computerboard mit einem bis dato ungewöhnlich großem Leitungsumfang zu einem kleinen Preis.

Außerdem lassen sich an das Board ein Monitor und als Eingabegeräte eine USB-Tastatur sowie eine USB-Maus anschließen. Eine Netzwerk- und eine Audioverbindung komplettieren den Raspberry Pi hardwaretechnisch gesehen zu einem üblichen und bedienbaren Computer. Als Betriebssystem wird hierfür eine spezielle Linux-Distribution eingesetzt, die sich im Gegensatz zu anderen Systemen – insbesondere zu Windows – als recht genügsam darstellt, wenn es darum geht, mit relativ begrenzten Hardware-Ressourcen in puncto Prozessorleistung, Speichergröße und Grafiksystem umgehen zu können, die bei einem derartigen Single Board-Computer systemimmanent sind.

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1_1, © Springer Fachmedien Wiesbaden 2015

Wie es begann

Der Ursprung des Single Board-Computers reicht in das Jahr 2006 an die Universität Cambridge zurück. Die Intention war dabei, den Studenten sowie allen Interessierten ein kostengünstiges System an die Hand geben zu können, um ihnen hiermit das Programmieren von Computern näher zu bringen.

Fast jeder kennt den Umgang mit Personal Computern (PC) und in erster Linie mit dem Betriebssystem Windows. Im Laufe der Jahre ist es jedoch immer schwieriger und unübersichtlicher geworden, einen PC zu durchschauen und programmieren zu können, was insbesondere für die Hardware und die Programmierung derselben für eigene Anwendungen gilt. Selbst unter dem Gesichtspunkt, dass PCs für das, was sie leistungs- und funktionstechnisch bieten, immer günstiger geworden sind, kosten sie verhältnismäßig viel Geld.

In früheren Zeiten war es für viele PC-Benutzer eigentlich selbstverständlich, dass auch eigene Programme und Hardware-Erweiterungen für den Computer umgesetzt wurden, weshalb der PC damals noch als ein Gerät für Spezialisten wie Informatiker oder Elektroniker angesehen wurde. Diesen Status hat er schon lange nicht mehr. Dass er sich prinzipiell intuitiv von jedermann für alle möglichen Aufgaben einsetzen lässt, ist letztendlich der für den Anwender immer einfacher erscheinenden Software geschuldet.

Für die Programmiererseite führt diese Anwenderfreundlichkeit vermehrt zu aufwendigeren Prozessen, was nicht nur das entsprechende Know-How, sondern auch eine Vielzahl spezieller Tools, Entwicklungsumgebungen und Programmiersprachen erfordert. Nicht selten entsteht beim Windows-Betriebssystem und den entsprechenden Programmen der Eindruck, dass hier geradezu eine Unmenge an überflüssigem Programmcode »mitgeschleppt« wird, der selten bis nie aufgerufen wird. Dies liegt teilweise an der durchgängigen Kompatibilität des Personal Computers mit Windows, so dass selbst jahrzehntealte Programme möglichst noch mit der neuesten Windows-Version funktionieren, was zu recht hohen Leistungs- und Speicheranforderungen führt. Aus diesen Gründen erscheint ein üblicher PC als Lern-, Experimentier- und Programmiersystem für Einsteiger ungeeignet.

An der Universität in Cambridge wurde Anfang der achtziger Jahre ein Heimcomputer mit 6502-CPU (2 MHz) entwickelt, der in britischen Schulen als Lehrmodell fungierte und in einer Sendereihe von der BBC unterstützt wurde. Hergestellt wurde dieser Heimcomputer, der als *BBC Micro* bekannt ist, von der englischen Firma Acorn, die später als ARM (siehe Kapitel 3.2) von sich reden machte. Einer der Entwickler war Jack Lang, der auch an der Raspberry Pi-Entwicklung beteiligt war, was verdeutlicht, dass hier ebenfalls der Gedanke für die Schaffung eines »Lehrsystems« Pate gestanden hat.

Der Computer BBC Micro ist ursprünglich ebenfalls für Lehrzwecke von der Universität Cambridge konzipiert worden. © en.wikipedia.org

Im Gegensatz zum *BBC Micro* sollte der Raspberry Pi jedoch wesentlich günstiger sein, was mit einem Preis von ca. 25 € (Modell A) sicher auch gelungen ist. Außerdem ist der Raspberry Pi von vornherein dafür vorgesehen, auch Büroanwendungen und Spiele ausführen zu können. Aktuelle Features wie die Möglichkeit eines Internet-Zuganges und die Videowiedergabe in High Definition-Qualität lassen erkennen, dass es sich dabei um einen »richtigen« Computer im Kleinformat handelt.

Die Entwicklung basierte maßgeblich auf den Vorstellungen von David Braben, der bereits als Student in Cambridge 1984 das bekannte Weltraumspiel »Elite« entwickelt und das Entwicklerstudio *Frontier Developments* gegründet hatte. Zusammen mit zwei weiteren Informatikern aus Cambridge (Alan Mycroft, Rob Mullins) und Eben Upton, einem Cambridge-Absolventen, der mittlerweile Technical Director bei der Firma Broadcom war, sowie dem Unternehmer Pete Lomas (Norcott Technologies, Electronic Design), ebenfalls ein Cambridge-Absolvent, wurde die *Raspberry Pi Foundation* als gemeinnützige Stiftung gegründet, um das Projekt zu finanzieren und Entwickler von Open-Source-Software mit auf den Plan zu rufen.

Die Foundation wollte bei der Namensgebung für den Single Board-Computer die Tradition weiterverfolgen Computer nach Obstsorten zu benennen, wie Acorn (Eichel) oder Apricot (Aprikose), beide mit Cambridge/ARM-Tradition. Die bekannteste Frucht ist sicherlich Apple, deren Produkte, wie iPad oder iPhone, interessanterweise ebenfalls mit ARM-Prozessoren (Kapitel 3.2) arbeiten. Nach längeren Diskussionen einigte man sich dann auf Raspberry (Himbeere) mit dem Zusatz Pi, der nicht für Pie, also Torte oder Kuchen, stehen soll, sondern für die Kennzeichnung der Hauptprogrammiersprache des Systems, Python, gedacht ist.

Entgegen der ursprünglichen Idee wurde der Vertrieb der Raspberry Pi-Boards nicht selbst von der Foundation übernommen, sondern an die beiden weltweit agierenden Distributoren RS-Components und Farnell übertragen. Nach dem Verkaufsstart am 29.2.12 konnten die Distributoren innerhalb einer Stunde über 10.000 Boards verkaufen. Der Andrang war so groß, dass der Verkauf ausgesetzt werden musste und Interessenten sich erst für ein einziges Exemplar pro Käufer zu registrieren hatten, bevor es dann Wochen bis Monate später wieder erhältlich war. Innerhalb eines Jahres (2012) haben die beiden Firmen dann über 800.000 Stück davon verkauft. Mittlerweile sind es über drei Millionen Stück.

Der Raspberry Pi erlaubt die Verbindung mit verschiedener Peripherie, was vielfältige Einsatzmöglichkeiten gestattet.

Der Raspberry Pi hat weltweit einen »Hype« ausgelöst, was nicht nur am günstigen Preis liegt, sondern wohl eher am Gesamtkonzept, bei dem nicht der maximale Profit das Streben ist, sondern die Tatsache, dass hier maximale Leistung bei minimalem Preis mit dem Open-Source-Gedanken verbunden wird. Somit kann sich

auch jeder interessierte (junge) Mensch – mit schmalem Geldbeutel – Programmieren und Experimentieren auf aktueller Hardware leisten und regen Austausch mit anderen Entwicklern in der Open-Source-Gemeinde betreiben. Es ist tatsächlich beachtlich, wie viele Anwendungen und Applikationen für den Raspberry Pi in kürzester Zeit entstanden sind, die – dank Open Source – kein Geld kosten und laufend weiterentwickelt werden.

Arduino

Das erste bekannte Open-Source-Projekt, welches Software mit Hardware kombiniert, ist als Arduino bekannt. Hiermit wird ebenfalls versucht, auch Laien einen einfachen, relativ kostengünstigen Zugang zur Programmierung und Hardware-Entwicklung zu ermöglichen. Initiiert wurde Arduino im Jahre 2005 von Massimo Banzi und David Cuartielles am *Interaction Design Institute Iverea* in Italien, ursprünglich gedacht für die Steuerung von interaktiven Objekten und Installationen in Kunstausstellungen und Museen. Dabei spielt die Interaktion mit der Umwelt und/oder den Betrachtern eine wichtige Rolle. Für die Kommunikation mit der realen – der analogen – Umwelt ist deshalb von vornherein der Einsatz von Sensoren und Aktoren vorgesehen, die die analogen Signale umsetzen und mit einem Mikrocontroller kommunizieren, was aktuell als *Physical Computing* bezeichnet wird.

Das Uno-Board arbeitet mit einem AVR-Mikrocontroller vom Typ ATmega328 und bildet aktuell die am häufigsten verwendete Arduino-Platine.

Arduino ist im Gegensatz zum Raspberry Pi nicht als Computer mit eigenem Betriebssystem ausgelegt, welches alle möglichen Anwendungen (Office, Mediacen-

ter, Web-Server) ausführen kann. Demnach kann der Raspberry Pi auch ohne eigenes Programmieren »nur« als Mini-PC eingesetzt werden. Bei Arduino wird demgegenüber stets ein Mikrocontroller für eigene Projekte programmiert.

Das Arduino-Konzept, welches aus quelloffener Hard- und Software besteht, wurde rasch zu einer populären Plattform, die es auf einfache Art und Weise ermöglicht, eigene Hardware-Projekte realisieren zu können, was durch eine sehr aktive Community unterstützt wird und in der Zwischenzeit zu einer geradezu unüberschaubaren Anzahl von Projekten und Applikationen geführt hat, die teilweise für den Raspberry Pi adaptiert werden. Als Basis einer Entwicklung dient ein meist preisgünstiges Arduino-Board, wovon es mittlerweile eine ganze Reihe recht unterschiedlicher Modelle gibt. Ursprünglich wird auf einem Arduino-Board ein AVR-Mikrocontroller der Firma Atmel mit 8 Bit und einem typischen Takt von 16 MHz eingesetzt.

Die Arduino-Entwicklungsumgebung ist einfach aufgebaut, was die Erstellung eigener Programme sehr erleichtert. Hier ist das typische LED-Blink-Programm in der Bearbeitung.

Die Arduino-Entwicklungsumgebung, die mit Windows-, Linux- und Mac OS-Computern eingesetzt werden kann, basiert auf einem Java-Dialekt (Processing) und wirkt sehr übersichtlich. Die Programme werden in einem vereinfachten C-

Dialekt (Wiring) erstellt, die bei Arduino als Sketches bezeichnet werden. Ein wesentlicher Bestandteil sind dabei die zur Verfügung gestellten Core Libraries, die eine Vielzahl von unterschiedlichen Boards unterstützen und die Komplexität der Software gegenüber dem Programmierer verbergen.

Mikrocontroller und Mini-Computer

Die Arduino-Plattform und der Raspberry Pi haben wesentlich zum Boom der elektronischen *Maker Scene* beigetragen. Deshalb verwundert es auch nicht, dass momentan von allen möglichen Firmen und Herstellern verschiedenste Mikrocontroller- und Computer-Boards vorgestellt werden, die diese Zielgruppe adressieren. Selbst Firmen wie Microsoft und Intel sind mit eigenen Lösungen vertreten.

Von Microsoft wird die .NET Gadgeteer-Plattform propagiert, die wie der Raspberry Pi auf einem ARM-Prozessor basiert. Die Gadgeteer-Boards können mit verschiedenen Erweiterungs-Boards (Sockets) verbunden werden, was dem Konzept von Arduino entspricht, wo die Erweiterungsplatinen als *Shields* bezeichnet und direkt auf das Controller-Board gesteckt werden, während die *Sockets* über Flachbandkabel mit dem Gadgeteer-Board verbunden werden. Die Gadgeteer-Programme werden in C# mithilfe des Microsoft Visual Studios erstellt und laufen auf der Hardware unter dem .NET Micro Framework.

Die Peripherie wird an ein Gadgeteer-Board über einzelne Flachbandkabel angeschlossen. Der Prozessor befindet sich auf der Platinenrückseite.

Die Firma Intel hat ein offizielles Arduino-Board, welches demnach von der Arduino-Organisation unterstützt wird, unter der Bezeichnung *Galileo* vorgestellt. Eingesetzt wird hier ein *Intel Quark SoC X1000*, ein bisher für eine Arduino-Platine

ungewöhnlicher Prozessor, dessen Software dennoch in der Arduino-IDE entwickelt werden kann. In der Galileo-Firmware arbeitet ein Minimal-Linux (Yocto), welches die Board-Hardware entsprechend unterstützt. Der X1000-Chip enthält eine 32-Bit x86-CPU mit Gleitkommaeinheit, die zum mathematischen Coprozessor 80387 kompatibel ist. Die Basis des SoC (System on Chip) bildet die Technik eines 486-Prozessors ohne Sprungvorhersage und mit einer fünfstufigen 80486-Pipeline. Diese Technik geht zurück auf die Intel-Entwicklungen aus dem Jahr 1987. Der Befehlssatz umfasst den eines Pentium-Prozessors ohne die Codeerweiterungen wie MMX, SSE und SIMD. Als Energiesparfunktionen (ACPI) sind einige Modi des Atom-Prozessors implementiert.

Das Edison-Modul auf einer Trägerplatine, die auch Arduino Shields unterstützt.

Das zweite System von Intel wird als Edison bezeichnet. Das Modul ist nicht größer als eine SD-Karte und enthält in einem Scheckkarten-großen Modul drei CPU-Kerne (Dual Core Atom plus Quark SoC), 1 GByte SDRAM sowie WLAN und Bluetooth 4.0. Als Schnittstellen stehen an den 70 Kontakten USB 2.0, I²C, SPI, UART sowie 40 GPIO-Leitungen zur Verfügung. Wie bei Galileo wird auch hier ein Yocto-Linux eingesetzt, über Adapterplatinen lässt sich eine Kompatibilität zu Arduino herstellen. Die beiden Intel-Lösungen zielen auf ein breites Spektrum von leistungsfähigen Embedded Designs ab, wie auf Applikationen für das Internet-Of-Things (IoT) und Wearable-Electronic. Eine direkte Konkurrenz für den Raspberry Pi sind diese Boards nicht, denn sie verfügen über keine Grafikfähigkeiten.

Der Raspberry Pi ist hingegen ein Mini-Computer, der sich äußerst vielseitig einsetzen lässt, was auch andere Systeme versprechen, die bereits vor oder fast zeitgleich mit dem Raspberry Pi erschienen sind, wie das Cubieboard oder Cubietruck und das Pandaboard, die ebenfalls unter Open-Source firmieren und über eigene aktive Communities verfügen. Diese Boards, von denen es bereits mehrere Varian-

ten gibt, arbeiten wie der Raspberry Pi ebenfalls mit Prozessoren auf ARM-Basis wie Cortex A7 (Dual Core) und Cortex A8, die mit einem Grafikchip (Mali 400) und weiteren Peripherieeinheiten zu SoCs wie Allwinner A10, Allwinner A20 und OMAP 4430/60 kombiniert und mit mindestens 1 GHz getaktet werden, also höher als der Raspberry Pi. Sie verfügen außerdem über bis zu 2 GByte an SDRAM-Speicher. In puncto Audio/Video und Netzwerk sowie Schnittstellen bieten diese Boards mindestens eine vergleichbare, wenn nicht sogar bessere Ausstattung mit Gigabit-LAN, SATA, WLAN und Bluetooth, was im Detail von der jeweiligen Board-Version abhängt.

Näher in der Preisregion des Raspberry Pi bewegen sich *Beaglebone Black* sowie das *Banana Board*, welches einem Raspberry Pi Board fast zum Verwechseln ähnlich sieht. Das Beaglebone Black-System ist mit einer schwarzen Platine aufgebaut und hat einen Vorgänger, den *Beaglebone* aus dem Jahr 2008 mit weißer Platine, der demgegenüber weniger gut ausgestattet und fast doppelt so teuer ist. Initiiert wurde die Beagleboard-Open-Source-Community vom internationalen Distributor Digi-Key aus den USA.

Der Beaglebone Black bietet sich als teurere, dafür aber leistungsfähigere Alternative zum Raspberry Pi an.

Beim Beaglebone Black werden eine ARM-CPU vom Typ Cortex A8 mit einem Takt von 1 GHz und eine Grafik-Engine vom Typ SGX30 von der Firma Texas

Instruments zum SoC Sitara AM3359 kombiniert. Als SDRAM-Speicher sind 512 MByte DDR3-SDRAM vorhanden, und 2 GByte Flash-Speicher stehen für den Programmcode zur Verfügung, wobei hier bereits werksseitig eine Linux-Version (Ångström Linux mit Gnome-Desktop) sowie eine Java-basierte Entwicklungsumgebung auf dem integrierten Embedded MultiMedia Card-Flash-Speicher (eMMC) installiert sind. Im Gegensatz zum Raspberry Pi wird das Betriebssystem hier also nicht per SD-Karte integriert, so dass der vorhandene SD-Kartensteckplatz für Speichererweiterungen einsetzbar ist. Über eine SD Card ist es auch möglich, ein anderes Betriebssystem per Image im eMMC-Speicher zu speichern, etwa Ubuntu oder Android, was vom Raspberry Pi beides nicht unterstützt wird.

Anfand des Jahres 2014 erschien der Banana Pi, für den sehr massiv geworben wird. Trotz der Bezeichnung stammt er nicht von der *Raspberry Pi Foundation*, sondern aus China und soll als System für die dortige Nutzung in Bildungseinrichtungen vorgesehen sein. Er verwendet ebenfalls eine ARM-CPU, und zwar einen Dual Cortex A7, der mit einer Mali 400-Grafik in einem Allwinner A20 SoC mit 1 GHz zusammengefasst wird. Der Allwinner A20 wird ebenfalls bei den Cubietruck-Boards (s.o.) eingesetzt und kommuniziert mit 1 GByte DDR3-SDRAM.

Die Ausführung des Boards nebst den Anschlüssen für die Spannungsversorgung, Audio, Video, USB, SD Card-Steckplatz, CSI/DSI, HDMI, Ethernet und sogar der 26-polige GPIO-Pfostenstecker sind ganz offensichtlich als Kopie vom Raspberry Pi Modell B angefertigt worden. Das Netzwerk erlaubt allerdings Gigibit-Ethernet, statt zwei gibt es drei USB-Anschlüsse, außerdem einen SATA-Port, einen CAN-Bus-Controller, einen Mikrofoneingang und einen Infrarotempfänger.

Das Board ist geringfügig größer, so dass die Banana Pi-Platine nicht in die üblichen Raspberry Pi-Gehäuse passt. Die GPIO-Leiste vom Banana Pi ist zwar elektrisch kompatibel, sie ist jedoch etwas näher an der gelben Videobuchse positioniert, so dass sich einige Erweiterungsplatinen nicht aufstecken lassen.

Der Banana Pi ist eine Kopie aus China. Der Allwinner SoC und die Speicherchips befinden sich auf der Platinenrückseite.

Die Software ist keineswegs kompatibel, weil der Allwinner A20 anders verschaltet ist und abweichende Hardware verwendet wird, so dass auch speziell angepasste Betriebssysteme für den Banana Pi notwendig sind, die (momentan noch) keine Hardware-basierte Videobeschleunigung bieten. Ein paar Monate nachdem die *Raspberry Pi Foundation* das Modell B+ (siehe Kapitel 3.11) vorgestellt hat, gibt es auch vom Banana Pi bereits eine Pro-Version, die ebenfalls die wichtigen Änderungen wie die 40-polige GPIO-Leiste und den Micro SD-Kartenslot entsprechend übernimmt.

1 Schnellstart

Den Verkauf der Raspberry Pi-Boards hat die *Raspberry Pi Foundation* den beiden direkten Vertriebspartnern Distributoren RS-Components und Farnell (Element 14) übertragen. Mittlerweile sind Raspberry Pi Boards auch bei anderen Firmen wie etwa bei Amazon, Reichelt oder Pollin erhältlich, wobei die Preise keineswegs identisch sind und die Kosten für Transport und Verpackung ebenfalls unterschiedlich ausfallen.

Abbildung 1.1: Das Raspberry Pi-Modell B

Der Verkauf startete mit dem Modell B, gefolgt vom Modell A Anfang des Jahres 2013. Beide Board-Typen unterscheiden sich lediglich dadurch, dass das Modell B über einen SDRAM-Speicher von 512 MByte verfügt, das Modell A hingegen lediglich 256 MByte, nur einen einzigen USB-Port und keinen Ethernet-Anschluss besitzt. Die erste Version des Modell B, die nicht mehr verkauft wird, besaß auch nur einen SDRAM-Speicher von 256 MByte.

Im Jahre 2014 folgten die +Modelle (siehe Kapitel 3.11), die als wesentliche Neuerung über einen um 14 Kontakte erweiterten GPIO-Connector, der nunmehr 40-polig ist, verfügen und als Speichermedium statt einer SD Card eine Micro SD Card verwenden. Außerdem ist ein Raspberry Pi Compute Module (siehe Kapitel 3.10) plus einer hierzu passenden Trägerkarte (I/O Board) als *Compute Module Development Kit* erhältlich, was explizit auf die Entwicklung eigener Embedded-Lösungen abzielt, die den Raspberry Pi als Controller einsetzen.

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1_2, © Springer Fachmedien Wiesbaden 2015

Abbildung 1.2: Das Compute Module Development Kit wird mit dem Raspberry Pi-Modul und einer dazu passenden Trägerplatine sowie einigem Zubehör geliefert.

Die neueren Raspberry Pi-Boards brechen nicht mit der Kompatibilität, die mit den Ursprungsmodellen gesetzt wurde, die die folgenden Eigenschaften aufweisen:

Kurzdaten der ursprünglichen Raspberry Pi-Boards:

- System on Chip (SoC) BCM2835 der Firma Broadcom mit integrierter CPU, GPU, DSP und SDRAM.
- CPU: *Central Processing Unit* ARM1176JZF-S, getaktet mit 700 MHz.
- GPU: *Graphic Processing Unit* Broadcom Video Core IV, Open GL ES2.0, 1080p30, H264/MPEG-4 AVC High Profile Decoder (Blu-Ray-Qualität).
- DSP: Integrierter *Digitaler System Prozessor*, der systemintern für Grafik, Video und Audio genutzt wird und (noch) nicht separat (per API) programmierbar ist.
- Speicher von 512 MByte SDRAM beim Modell B Revison 2 sowie 256 MByte SDRAM beim Modell B Revision 1 und beim Modell A Revision 1.
- Grafik/Video-Ausgänge: Composite RCA und HDMI.
- Audio-Ausgang: 3,5 mm Stereo-Klinkenbuchse und per HDMI-Connector.
- Tastatur- und Mausanschluss über die vorhandenen USB 2.0-Ports. Das Modell B verfügt über zwei, das Modell A nur über einen USB-Anschluss.

- Spannungsversorgung (5 V) mit externem Netzteil über microUSB-Anschluss. Leistung mindestens 3,5 W (700 mA für Modell B) und 2,5 W (500 mA für Modell A).
- SD-Karten-Slot für SD- (max. 2 GByte) und SDHC-Cards (max. 32 GByte) als Programm- und Datenspeichermedium (für Betriebssystem).
- Ethernet-Anschluss mit RJ45-Connector für 10 und 100 MBit/s (autodetect). Ist nicht beim Modell A vorhanden.
- WLAN-Adapter (Wi-Fi) und weitere optionale Peripherie wird über die USB-Ports angeschlossen, was den Einsatz eines separaten USB-Hubs mit eigenem Netzteil nahelegt.

Neben dem Raspberry Pi-Board empfiehlt es sich, auch gleich eine passende SD Card mitzukaufen, die bereits ein installiertes Betriebssystem enthält, was typischerweise die spezielle Linux-Version Raspbian ist. Das NOOBS-Paket, welches einen Installer bietet, um wahlweise Arch Linux, Pidora Linux und Raspbian sowie RISC OS und die beiden Mediacenterlösungen OpenELEC und RaspBMC auszuwählen und daraufhin einsetzen zu können, wird ebenfalls auf fertig konfektionierten SD Cards angeboten. Im Kapitel 1.4 wird hierauf näher eingegangen. Alle Systeme sind auch kostenlos als Download, beispielsweise von der Internetseite der *Raspberry Pi Foundation*, erhältlich.

Abbildung 1.3: Diese SD-Karten sind bereits mit einem oder gleich mehreren Systemen vorinstalliert. Die +Modelle benötigen eine Micro SD Card, die in einem SD Card-Adapter untergebracht sein kann.

Insbesondere beim Einstieg in die Raspberry Pi-Thematik kann es einige Stolpersteine geben, so dass man bestimmte Dinge, wie eine nicht korrekt funktionierende SD Card, am besten gleich von vornherein ausschließen sollte. Im Kapitel 4 wird noch genau auf die verschiedenen Betriebssysteme und deren Installation eingegangen. Zunächst soll der Raspberry Pi nur booten, womit bewiesen ist, dass das System grundsätzlich funktioniert.

Abbildung 1.4: Die SD-Karte wird in den Kartenslot auf der Platinenrückseite eingesetzt, wobei auf die abgeschrägte Kante zu achten ist, die den Pin 1 markiert. Ohne Gewalt lässt sich die Karte gar nicht falsch herum einsetzen.

1.1 Anschließen

Nach dem Einsetzen der SD-Karte (Abbildung 1.4) sind die folgenden Peripherie-Einheiten notwendig und mit dem Raspberry Pi-Board zu verbinden:

- Das USB-Netzteil
- Das USB-Spannungskabel für die Verbindung des USB-Netzteils mit der Micro-USB-Buchse des Raspberry Pi-Boards
- Die Tastatur mit dem USB-Anschluss
- Das HDMI-Kabel für die Verbindung mit einem geeigneten Monitor

1.1.1 Netzteil

Im Prinzip kann der Raspberry Pi für die Spannungsversorgung über ein entsprechendes USB-Kabel mit dem USB-Anschluss eines PCs oder eines USB-Hubs verbunden werden, was als erster Funktionstest sogar als eine ratsame Methode anzusehen ist. Für den üblichen Betrieb wird jedoch eher eine autarke Versorgung mit einem passenden Netzteil sinnvoll sein. Hierfür hat die *Raspberry Pi Foundation* einen gebräuchlichen Typ vorgesehen, wie er bei vielen Kleingeräten zum Einsatz kommt. Demnach handelt es um ein relativ kompaktes Steckernetzteil mit einem gewöhnlichen USB-Anschluss (Buchse USB-Typ A) und einem Verbindungskabel, welches mit einem USB-Typ-A Stecker auf der einen und mit einem USB-Micro-B-Stecker auf der anderen Seite versehen ist.

Das Netzteil sollte mindestens 1 A (1000 mA, 5 W) liefern können, damit noch etwas Reserve für später anzuschließende Peripherie gegeben ist. Wichtig ist, dass es

sich um ein Netzteil mit einer geregelten Ausgangsspannung von 5 V handelt und dass es über einen Micro-USB-Anschluss verfügt, der dementsprechend mit der Power-Buchse (S1, siehe Abbildung 1.1) des Raspberry Pi-Board verbunden wird. Oftmals befindet sich das Verbindungskabel nicht im Lieferumfang des Netzteils und muss möglicherweise dazugekauft werden.

Abbildung 1.5: Passende Netzteile mit festem oder steckbarem USB-Kabel

Mitunter wird auch ein »Ladegerät« empfohlen, wie es für Smartphones eingesetzt wird. Bei den aktuellen Modellen, die mit einem Lithium-Akku arbeiten, entspricht das vermeintliche Ladegerät allerdings einem üblichen Netzteil, denn die eigentliche Ladeschaltung befindet sich innerhalb des Smartphones. Grundsätzlich handelt es sich bei einem Ladegerät und bei einem Netzteil um zwei unterschiedliche Gerätetypen, so dass nicht das Ladegerät eines x-beliebigen Mobiltelefons verwendet werden kann. Ein »richtiges« Ladegerät liefert demgegenüber nicht eine konstante Spannung von 5 V, wie sie der Raspberry Pi benötigt, sondern startet typischerweise mit einem konstanten Ladestrom, der immer geringer wird, bevor am Schluss eine konstante Spannung (von vielleicht 5 V) für die Restaufladung des Akkus sorgt.

Aus Sicherheitsgründen sollten Spannungsverbindungen nur bei ausgeschaltetem Board vorgenommen werden, auch wenn der USB den Geräteanschluss bei aktivem Gerät erlaubt und HDMI dies ebenfalls tolerieren sollte. Gleichwohl ist das Board mit den quasi freiliegenden Anschlüssen, insbesondere wenn es sich noch nicht in einem Gehäuse befindet, nicht gegen den unabsichtlichen Kontakt mit daneben liegenden Kabeln oder Werkzeugen geschützt.

In die Steckdose wird das Netzteil erst dann eingesteckt, wenn alle Verbindungen soweit hergestellt sind. Weil weder der Raspberry Pi noch die üblichen USB-Netzteile über einen Einschalter verfügen, empfiehlt sich der Einsatz einer schalt-

baren Steckdosenleiste, die später auch das Netzteil für einen USB-Hub aufnehmen kann. Das Raspberry Pi-System kann somit mit einem einzigen Schalter komplett ein- und ausgeschaltet werden, ohne dass laufend die einzelnen Netzteile aus der Dose entfernt werden müssen. Es sei darauf hingewiesen, dass es keine praktikable Alternative ist, wenn stattdessen das Kabel für die Spannungsversorgung aus der Micro-USB-Buchse ständig eingesteckt und herausgezogen wird. Dieses Verfahren sollte schon gar nicht mit anderen Netzteilen, die später vielleicht noch dazukommen, praktiziert werden, weil Stecker durchaus verkanten können oder man unglücklicherweise mit dem spannungsführenden Kabel an eine Stelle der Elektronik gerät, die dies nicht verträgt.

1.1.2 Tastatur und Maus

Als Eingabegerät ist eine gewöhnliche Tastatur mit einem USB-Anschluss vorgesehen, die mit einer USB-Buchse des Raspberry Pi-Boards zu verbinden ist. Das Modell A verfügt nur über eine einzige USB-Buchse, so dass für eine Maus im Bedarfsfall ein USB-Hub mit eigenem Netzteil notwendig wird.

Abbildung 1.6: Die Eingabegeräte mit USB-Anschluss sollten einfach ausgeführt sein.

Eine Maus wird zunächst jedoch nicht benötigt, so dass sie für die erste Inbetriebnahme auch nicht anzuschließen ist. Beide Eingabegeräte sollten einer einfachen Standardausführung genügen, denn eine Tastatur mit beleuchteten Tasten oder eine Gaming-Maus verbrauchen unnötig viel Strom, der vom Raspberry Pi möglicherweise nicht per USB geliefert werden kann.

1.1.3 Monitor

Jeder aktuelle Computermonitor verfügt über einen DVI-Anschluss (siehe Kapitel 3.4.1), der vom Prinzip her auch HDMI (High Definition Multimedia Interface) unterstützt, was bedeutet, dass es durchaus ältere Monitore gibt, die zwar DVI, nicht jedoch HDMI verarbeiten können.

HDMI setzt für die digitale Bild- und Tonübertragung über HDMI einen Kopierschutz ein, der als *High bandwith Digital Content Protection* (HDCP) bezeichnet wird und auch vom Raspberry Pi unterstützt wird, etwa um Blu-ray-Videos abspielen zu können. Wenn für einen Monitor lediglich DVI (ohne HDMI und/oder HDCP) spezifiziert wird, nützt auch kein DVI-HDMI-Adapter oder -Kabel, er wird mit dem Raspberry Pi nicht funktionieren.

Abbildung 1.7: Für den Raspberry Pi wird ein Standard HDMI-Kabel (Typ A) benötigt.

Falls ein Adapter oder auch ein Adapterkabel zwischen HDMI des Raspberry Pi und DVI bei einem Monitor eingesetzt werden soll, ist insbesondere darauf zu achten, wo sich der Buchsen- und wo sich der Steckkontakt bei den Anschlüssen befindet, was sich bei HDMI nicht so einfach wie bei anderen Anschlüssen feststellen lässt. Wenn man davon ausgeht, dass sich auf dem Raspberry Pi-Board eine Buchse befindet, führt diese in der Mitte einen Steg mit den Kontakten, so dass dementsprechend ein DVI-Kabel mit zwei Typ A-Steckern (Abbildung 1.7) benötigt wird.

Beim beabsichtigtem Einsatz eines Adapters, wie er beispielsweise in der Abbildung 1.8 zu erkennen ist, muss darauf geachtet werden, dass der Monitor hinten genügend Platz für den Adapter mit dem aufgesteckten HDMI-Kabel bietet, also beides hintereinander einsteckbar ist. Fall dies nicht gegeben sein sollte, ist ein

Adapter für die andere Seite (am Raspberry Pi) oder ein Adapterkabel – DVI-Stecker auf HDMI-Stecker – notwendig.

Abbildung 1.8: Dieser Monitor verfügt über zwei Anschlüsse, wobei die DVI-Buchse mit einem Adapter (rechts) auf eine HDMI-Buchse umgesetzt werden kann.

Computermonitore haben oftmals zwei (oder mehr) verschiedene Anschlüsse, so dass es praktisch erscheint, einen für den üblichen PC und den anderen für den Raspberry Pi zu nehmen. Es ist dann möglich, zwischen den beiden Anschlüssen, VGA für den PC und DVI/HDMI für den Raspberry Pi, per Monitormenü umzuschalten, was in der Praxis – in Abhängigkeit vom jeweiligen Monitormodell – mehr oder weniger gut funktioniert, so dass u.U. ein Nichtfunktionieren des Raspberry Pi diagnostiziert wird, weil wider Erwarten kein Bild erscheint. Die Status-Leuchtdioden (PWR, ACT, siehe Kapitel 1.2) auf dem Raspberry Pi-Board können diesen Zustand zwar grundsätzlich kenntlich machen, doch das bringt bei der Fehlersuche zunächst keinen Erfolg. Die LEDs leuchten bzw. blinken zwar, der Monitor bleibt jedoch schwarz.

Abbildung 1.9: Im Monitormenü muss auf DVI umgeschaltet werden.

Wenn das Umschalten am Monitor nichts bewirkt, sollte allein das DVI/HDMI-Kabel angeschlossen sein und sowohl der Raspberry Pi als auch der Monitor aus- und wieder eingeschaltet werden. Dabei zuerst den Monitor und dann den Raspberry Pi einschalten. Falls dennoch kein Bild erscheinen sollte, ist die Verbindung oder der Monitor am DVI/HDMI-Anschluss nicht in Ordnung, wenn der Monitor nicht grundsätzlich defekt ist und mit dem VGA-Anschluss einwandfrei funktioniert. Es sei erwähnt, dass die automatische Signalerkennung bei einem Monitor

mitunter auch nicht optimal funktioniert, so dass der gewünschte Eingang explizit per Monitormenü (Abbildung 1.9) zu selektieren ist.

Abbildung 1.10: Praktisch, aber verhältnismäßig teuer, der VGA- auf HDMI-Adapter Pi-View für den Raspberry Pi.

In der Regel wird der DVI/HDMI-Anschluss eines Monitors für den PC eingesetzt, weil dies eine bessere Bildqualität als VGA sicherstellt, was insbesondere für Spiele von Bedeutung ist. Deshalb kann auch der VGA-Eingang für den Raspberry Pi geeignet sein, zumindest dann, wenn er ohne eine grafische Oberfläche – im Textmodus – betrieben wird. Aufgrund der völlig unterschiedlichen Elektronik (VGA = analog, DVI = digital) ist es jedoch nicht möglich, den Raspberry Pi direkt über VGA zu betreiben. Gleichwohl gibt es hierfür eine Adapterlösung (Pi-View), die eine aufwendige Konverterelektronik enthält.

1.2 Einschalten und booten

Wenn die Verbindungen mit dem Netzteil, der USB-Tastatur und dem Monitor über HDMI hergestellt sind, wird zunächst der Monitor und dann die Steckerleiste mit dem USB-Netzteil eingeschaltet, woraufhin die PWR-LED leuchtet und kurz danach die ACT blinkt, was signalisiert, dass der Bootvorgang von der SD Card stattfindet. Dabei sollte auch das Bild auf dem Monitor erscheinen. Falls dies nicht passiert, sind die zuvor erläuterten Schritte noch einmal zu überprüfen.

Abbildung 1.11: Diese Minimalkonfiguration sollte immer funktionieren.

An dieser Stelle ist es wichtig, dass tatsächlich nur diese drei Verbindungen (siehe Abbildung 1.11) und keine weiteren hergestellt worden sind, auch kein Audio, kein Ethernet oder sonstiges. Es scheint keine Seltenheit zu sein, dass vor lauter Euphorie gleich am Anfang alle möglichen Dinge angeschlossen werden, wie bereits eine Ethernet-Verbindung, ein *Wi-Fi Dongle* für den Aufbau einer WLAN-Verbindung oder ein USB-Hub oder eine Erweiterungsplatine, was beim ersten Start nur für unübersichtliche Verhältnisse sorgt und deshalb vermieden werden sollte.

Abbildung 1.12: Die Spannung liegt an (PWR-LED) und das System bootet (ACT-LED).

Auf dem Monitor erscheint als erstes das Logo – die Himbeere –, gefolgt von einem für Linux typischen Boot-Prozess, der die Ausführung der einzelnen Initialisierungsschritte ausweist. Was dies im Einzelnen bedeutet, ist im Moment (siehe hierfür Kapitel 4) nicht von Bedeutung, wichtig ist nur, dass der Vorgang ohne Fehler ausgeführt wird, woraufhin beim ersten Boot ein Dialog (Abbildung 1.13) erscheint, der es ermöglicht, einige Systemeinstellungen vorzunehmen. Dazu gehören Einstellungen für die Sprache, die Zeitzone, das Tastaturlayout und einige weitere Dinge. Eine Beschreibung der einzelnen Optionen, die sich in Abhängigkeit von der jeweiligen Version durchaus voneinander unterscheiden und verschiedene Bezeichnungen tragen können, ist im Kapitel 1.3 angegeben.

Abbildung 1.13: Ein Ausschnitt aus dem Bootvorgang

Das Konfigurationsprogramm (raspi-config) kann grundsätzlich mit der folgenden Befehlszeile aufgerufen werden, so dass im Grunde genommen jederzeit entspre-

chende Anpassungen an den Bootoptionen und der Systemkonfiguration durchgeführt werden können:

```
pi@raspberrypi ~ $ sudo raspi-config
```

Falls noch keine deutsche Tastaturbelegung festgelegt wurde, ist statt des Minus-Zeichen »-« die Taste mit dem Zeichen »ß« (? \) einzugeben.

Das Konfigurationsprogramm lässt sich ausschließlich mit der Tastatur bedienen, weil es zu diesem Zeitpunkt (und generell im Textmodus) keine Mausunterstützung gibt. Für die Selektierung werden die Pfeiltasten und für das Ausführen der jeweiligen Option die Eingabetaste eingesetzt. Mithilfe der Leertaste lassen sich bei einigen Optionen (z.B. change locale) mehrere Optionen markieren, was durch das Zeichen »*« kenntlich gemacht wird. Über die Tab-Taste wird zu den verschiedenen Auslöse-Buttons (Ok, Cancel, Select, Finish) geschaltet.

Das Konfigurationsprogramm bietet im Prinzip lediglich eine Bedienoberfläche und ruft je nach ausgewählter Option ein weiteres Programm auf, welches die eigentliche Funktion übernimmt und was je nach Komplexität eine geraume Zeit (bis zu mehrere Minuten) in Anspruch nimmt.

Abbildung 1.14: Raspi-config erscheint automatisch beim ersten Bootvorgang.

Die an dieser Stelle wichtigste Festlegung betrifft die Länder- und Tastatureinstellung (configure_keyboard auch unter Internationalisation Options), denn in der Voreinstellung wird davon ausgegangen, dass eine englisch/amerikanische Tastatur zum Einsatz kommt. Falls zunächst keine Konfigurierung erfolgen soll, wird dieser Vorgang einfach durch `Finish` beendet. Es erscheint der *Raspberry Login* (Abbildung 1.15), so dass hier eine Anmeldung verlangt wird. Der zunächst einzi-

ge Benutzer lautet *pi* und das dazugehörige Password *raspberry*, wobei beim englisch/amerikanischen Layout stattdessen rasberrz einzugeben ist:

```
pi@raspberrypi ~ $ Raspberrypi login: pi
pi@raspberrypi ~ $ Password: raspberrz
```

Jede ausgegebene Kommandozeile beginnt mit `pi@raspberrypi ~ $`, was als Command Prompt zu verstehen ist, und `raspberrypi` ist der so genannte Hostname, der im Bedarfsfall in der Datei `/etc/hostname` geändert werden kann.

Abbildung 1.15: Einloggen und Starten des Desktops

Auf dieser Kommandozeilenebene ist die Linux-übliche Bedienung und Ausführung von Kommandos und Programmen gegeben, mit den Besonderheiten der Debian-Distribution. Nach dem Start des grafischen Desktops ist dort eine *Debian Reference* zu finden, die das System und seine Bedienung hinreichend erläutert.

Der Desktop wird als LXDE bezeichnet, was für *Lightweight X11 Desktop Environment* steht und eine freie Desktop-Umgebung für Linux und andere verwandte (UNIX) Betriebssysteme darstellt. Bekanntere Umgebungen sind KDE oder Gnome, die allerdings höhere Anforderungen an die Hardware stellen. LXDE ist explizit als »schlankes«, energiesparendes und schnelles System ausgelegt, bei dem die einzelnen Komponenten nur geringe Abhängigkeiten voneinander haben, so dass einzelne Module leicht anpassbar sind. Der Desktop wird gestartet mit:

```
pi@raspberrypi ~ $ startx
```

Daraufhin erscheint der Raspberry Pi-Desktop mit der Himbeere als Mittelpunkt, der sich am besten mit einer Maus bedienen lässt, so dass jetzt der richtige Zeitpunkt gekommen ist, eine Maus an die zweite USB-Buchse anzuschließen. Falls es sich um eine gewöhnliche Standard-USB-Maus (Abbildung 1.6) ohne Adapter und Sonderfunktionen handelt, wird sie üblicherweise sofort erkannt und kann unmit-

telbar verwendet werden. Die Programme auf dem Desktop sind typischerweise die folgenden:

- **Scratch**: Eine grafisch orientierte Programmiersprache für Kinder.
- **IDLE**: Eine Entwicklungsumgebung (IDE: Integrated Development Environment) für die Programmiersprache Python.
- **Debian Reference:** Eine recht ausführliche Beschreibung der Debian Linux Distribution.
- **Python Games:** Verschiedene einfache Spielprogramme (Tetris, Sokoban etc.), die an die Klassiker der achtziger Jahre anknüpfen.
- **Midori**: Ein einfacher Web-Browser, der auch Private Browsing unterstützt.
- **IDLE3:** Eine neuere Version (3) der Python-Entwicklungsumgebung.
- **WiFi Config:** Konfigurationsprogramm zum Einrichten eines WLAN-Zugangs, was die entsprechende Hardware (Wi-Fi Dongle, Router) erfordert.
- **LXTerminal:** Das Lightweight-Terminal-Programm für die direkte Ausführung von Linux-Kommandos.

Abbildung 1.16: Der LXDE-Desktop nach dem ersten Start

Der Desktop wird durch die Selektierung von Logout (rotes Symbol in der rechten Ecke der Taskleiste) beendet, wodurch man wieder zur die Kommandozeilenebene gelangt. Weil der Raspberry Pi über keinen Ein/Aus-Schalter verfügt, erfolgt das Ausschalten über eine schaltbare Steckdosenleiste, was meist nicht mit späteren Nachwirkungen (bootet nicht mehr, Daten und oder Programme sind beschädigt) verbunden ist, solange dies von der Kommandozeilenebene aus erfolgt und nicht gerade explizit Programme ausgeführt werden. Besser ist es jedoch, wenn das System vor dem Ausschalten ordnungsgemäß herunter gefahren wird, was beispielsweise mit dem Kommando `halt` ausgelöst wird.

```
pi@raspberrypi ~ $ sudo halt
```

Danach schaltet sich der Raspberry Pi ab und es leuchtet nur noch die Power-LED, so dass jetzt auch die Spannungsversorgung abgeschaltet werden kann.

Bei einigen Befehlen wird `sudo` vorangestellt, was man sich als »super user do« vorstellen kann, also dass der betreffende Befehl quasi von einem *Superuser* ausgeführt wird. Ein grundlegendes Prinzip von Linux ist es, dass es als Multiuser-System ausgelegt ist und demnach verschiedene Benutzer kennt, denen unterschiedliche Rechte zugewiesen werden können. Dabei existiert mindestens ein Superuser (Administrator), der als *root* fungiert und dem sämtliche Rechte am System zustehen, d.h., er kann standardmäßig sämtliche Befehle und Konfigurationsarbeiten ausführen, was einem »gewöhnlichen« Benutzer verwehrt bleibt.

Da ein Superuser demnach auch (unabsichtlich) nicht gewollte Systemänderungen vornehmen kann, ist es bei Linux nicht üblich, dass jeder Benutzer über Root-Rechte verfügt. Statt entweder nur als User oder als Root zu arbeiten, wird es mithilfe von *sudo* ermöglicht, dass bestimmte Anwender Kommandos ausführen können, die Root-Rechte erfordern. Es ist demnach keine vollständige Root-Funktionalität für das System erforderlich. Bei der Distribution, wie sie hier beim Raspberry Pi eingesetzt wird, gehört der Anwender *Pi* automatisch zur sudo-Gruppe, der durch `sudo -s` zum Benutzer »root« befördert werden kann.

1.3 Grundlegende Konfigurierung

Im vorherigen Abschnitt ist das Konfigurationsprogramm `raspi-config` (Abbildung 1.14) erwähnt worden, welches einige wichtige Einstellungsoptionen, wie Länder- und Tastatureinstellungen und einige weniger wichtige (z.B. Boot Behaviour) bietet, die in diesem Abschnitt erläutert werden.

Das Erscheinungsbild von raspi-config hat sich im Laufe der Zeit etwas geändert, weil Optionen wie *Enable Camera* oder *Add to Rastrack* hinzugekommen und einige Einstellungen zusammengefasst, umbenannt oder auch in einer anderen Reihenfolge angeordnet worden sind, wie es ein Vergleich der neueren Version (Abbil-

dung 1.17) mit einer der ersten (Abbildung 1.14) zeigt. Grundsätzliche Unterschiede gibt es zwischen den verschiedenen Versionen aber nicht. Es sind stets Funktionen hinzugekommen und bisher keine entfernt worden.

Abbildung 1.17: Eine raspi-config-Version neueren Datums

1.3.1 Information – About raspi-config

Unter *Info* oder *About raspi-config* (Information about this Tool) wird lediglich ein Text ausgegeben, der kurz die Funktion des Programms beschreibt. Eine Versionsnummer oder sonstige Angaben zum Programm gibt es hier nicht.

1.3.2 Kapazitätsbeschränkung aufheben – Expand Filesystem

Standardmäßig wird das Root-Dateisystem auf maximal 2 GByte beschränkt, demnach kann die durch eine SD-Karte zur Verfügung gestellte größere Kapazität nicht ausgenutzt werden. Durch Selektierung der *Option Expand Filesystem*, die auch unter *expand_rootfs* (Expand root Partition) firmiert, wird diese Beschränkung aufgehoben, so dass nach einem Neustart der gesamte Speicherplatz der Karte zur Verfügung steht.

1.3.3 Password ändern – Change User Password

Mit dieser Option, die auch als *change_pass* zu finden ist, kann das Password für den Benutzeruser *Pi*, welches von Hause aus *raspberry* lautet, geändert werden. Die Eingabezeilen erscheinen dabei unterhalb des Menübildschirmes.

Grundsätzlich ist bei Linux die Groß- und Kleinschreibung zu beachten. Sonderzeichen sollten zumindest in der Experimentierphase nicht verwendet werden, weil dies möglicherweise mit einer abweichenden Tastaturbelegung (englisch/amerikanisch) zu Problemen führen kann. Nachdem das neue Password, an das keine besonderen Komplexitätsanforderungen gestellt werden, eingegeben worden ist, ist noch eine Bestätigung notwendig, womit dieser Vorgang beendet ist.

1.3.4 Desktop automatisch starten – Enable Boot to Desktop/Scratch

In diesem Menüpunkt (auch Boot Behaviour) ist festzulegen, ob beim Boot automatisch der Konsolenmodus oder der Desktop (LXDE) oder direkt die Scratch-Programmierumgebung gestartet werden soll. Bei letzterem wird standardmäßig keine Anmeldung verlangt, was deshalb lediglich für die Programmierung im Lehr- und Schulungsumfeld sinnvoll sein kann.

Abbildung 1.18: Festlegung der Bootoption

1.3.5 Spracheinstellungen – Internationalisation Options

Dies ist eine der wichtigen Optionen, denn hier werden die Sprache (Change Locale), die Zeitzone (Change Timezone) und das Tastaturlayout (Change Keyboard Layout) bestimmt. Bei den älteren raspi-config-Versionen werden die drei Punkte als separate Einträge geführt, während sie bei den neueren unter den *Internationalisation Options* zusammengefasst werden, die ansonsten aber die gleiche Funktionalität aufweisen wie sie im Folgenden erläutert werden.

1.3.6 Nationale Zeichensätze – Set Locale

Diese Option (change locale, set locale) ist für das Laden von nationalen Zeichensätzen zuständig, also für die Ländereinstellungen, die hier nicht für die Tastatur (siehe Configure Keyboard), sondern für Anwendungen und Menütexte mit dem LXDE-Desktop von Bedeutung sind. Dies wird auch als Standortfestlegung bezeichnet. Dabei sind mehrere Zeichensätze mit der Leertaste selektierbar. Im Bedarfsfall kann dann später zwischen ihnen umgeschaltet werden.

Diese Umschaltung ist sofort auf der zweiten Menüseite der Change locale-Option möglich oder unter dem LXDE-Desktop mit einem optionalen Displaymanager (Slim). Der passende Zeichensatz für Deutschland ist: *de_DE.UTF-8 UTF-8*. Nach der entsprechenden Konfigurierung und dem Start des LXDE-Desktops erscheint dieser dann automatisch mit deutschen Bezeichnungen, was auch für einige Punkte in raspi-config gilt.

Abbildung 1.19: Auswahl des deutschen Zeichensatzes

1.3.7 Gebiet und Zeitzone – Set Timezone

Auf der ersten Seite dieser Option (Change Timezone, Set Timezone) wird das geografische Gebiet (Europa) und auf der zweiten die Zeitzone (Berlin) ausgewählt. Die Möglichkeit für das Stellen der Uhr oder des Datums ist an dieser Stelle hingegen nicht zu finden, weil der Raspberry Pi keine Echtzeituhr besitzt. Diese müsste schließlich auch im ausgeschalteten Zustand weiterlaufen, wofür jedoch kein Akku oder eine Batterie vorhanden ist. Im Kapitel 6.5.5 ist eine Applikation gezeigt, die eine Echtzeituhr beinhaltet.

Datum und Uhrzeit lassen sich anzeigen mit:

```
pi@raspberrypi ~ $ date
```

Dabei hat diese Angabe zunächst einmal nichts mit den aktuellen Zeitdaten zu tun. Gleichwohl sind Datum und Zeit wichtige Kriterien, auf die sich Installationen sowie alle möglichen Dateien (Logdaten) und Programmfunktionen beziehen. Ein manuelles Stellen ist möglich, etwa mit:

```
pi@raspberrypi ~ $ sudo date -s "11 march 2013 18:45:00"
```

Datum und Uhrzeit nach jedem Start des Raspberry Pi manuell setzen zu müssen, ist umständlich und ungenau. Die Lösung hierfür ist eine Internet-Verbindung (siehe auch Kapitel 4.8), die die Daten nach jedem Boot automatisch von einem Zeitserver (z.B. NTP) bezieht. Eine spezielle Konfigurierung ist hierfür nicht notwendig.

1.3.8 Tastatureinstellungen – Set Keyboard Layout

Wie erwähnt, wird vom Betriebssystem zunächst davon ausgegangen, dass eine Tastatur mit englisch/amerikanischer Belegung verwendet wird, so dass eine Reihe von Tasten in ihrer Funktion nicht mit der jeweiligen Beschriftung übereinstimmen. Nach der Selektierung von *Configure Keyboard* vergeht eine kurze Zeit, bis eine recht lange Liste mit verschiedenen Tastaturmodellen erscheint. Hier stehen ca. 100 verschiedene Modelle zur Auswahl, wobei man mit dem Typ *Generische PC-Tastatur mit 105 Tasten* (*Intl*), meist nichts (grob) Falsches ausgewählt hat.

Abbildung 1.20: Auswahl der Tastatur

Danach wird nach der Tastaturbelegung gefragt, die einfach mit *Deutsch* ausgewählt wird, woraufhin noch einige Sonderbelegungen festgelegt werden können. Alternativ wird einfach *Der Standard für die Tastaturbelegung* und nachfolgend *Keine Compose-Taste* gewählt. Die letzte Möglichkeit an dieser Stelle ist die Festlegung, ob die Tastenkombination Strg+Alt+Entf zum Beenden des X-Servers und damit zum Herunterfahren des LXDE-Desktops führen soll, was letztendlich Geschmackssache ist, gleichwohl eine vertraute Funktion darstellt und deshalb mit *Ja* selektiert werden kann, womit die Tastatur-Konfigurierung beendet ist.

1.3.9 Kamera aktivieren – Enable Camera

Diese Option ist für die Aktivierung der Pi Camera (siehe Kapitel 3.9.1) zuständig, die an das *Camera Serial Interface* (CSI) angeschlossen ist. Wenn diese Enable Camera-Option in raspi-config vorhanden ist, steht auch die passende Software für die Kamera unter */opt/vc/bin* zur Verfügung.

1.3.10 Registrieren – Add to Rastrack

Rastrack bietet eine Google-Maps basierte Weltkarte mit den Standorten von Raspberry Pi-Nutzern, die ihr System über diese Option oder auch direkt über die Rastrack-Internetseite registriert haben. Hierzu ist ein Benutzername und eine E-Mail-Adresse einzutragen. Es sind auch einige zusätzliche Angaben wie das Kaufdatum und ein Twitter-Name möglich. Rastrack soll dazu dienen, den Kontakt zwischen Raspberry-Pi-Nutzern herzustellen. Die jeweilige Position wird automatisch durch Google-Maps ermittelt und kann bei Bedarf manuell noch genauer angegeben werden.

1.3.11 Übertakten – Overclock

Das Übertakten eines Prozessors (Overclocking) ist gerade aus dem PC-Bereich eine bekannte, wenn auch nicht ungefährliche Methode, um eine schnellere Funktionsausführung zu ermöglichen. Deshalb mag es etwas verwunderlich sein, dass sich eine derartige Tuning-Maßnahme bei den grundlegenden Einstellungen befindet.

Abbildung 1.21: Optionen für das Übertakten

Der Standardtakt beträgt 700 MHz, der sich hier per Menüeinstellung in vier Stufen bis auf 1 GHz erhöhen lässt. Ausgehend von diesem Takt werden einzelne Komponenten (Core, SDRAM) des Broadcom BCM2835 ebenfalls mit einem unterschiedlich erhöhten Takt betrieben. Die einzelnen Frequenzen werden Chip-intern durch Taktvervielfachung alle aus dem externen Takt (Kapitel 3.7.1) von 19,2 MHz gewonnen.

Generell bedeutet ein höherer Takt einen höheren Stromverbrauch, was mit einer stärkeren Erwärmung des Chips einhergeht, der jedoch erst ab ca. 85 °C Schaden nehmen kann. Außerdem muss die interne Versorgungsspannung bei einer Takterhöhung ebenfalls angehoben werden, die vom CPU-Treiber in Abhängigkeit von der Taktfrequenz dynamisch angepasst wird.

Mit den vom Konfigurationsprogramm vorgeschlagenen Übertaktungsoptionen geht man demnach kein besonderes Risiko ein. Wenn sich die Taktfrequenz dennoch als zu hoch erweisen sollte, was sich durch ein instabiles Verhalten des Raspberry Pi bemerkbar macht, ist beim erneuten Boot lediglich die Shift-Taste gedrückt zu halten, wodurch die Normaleinstellung wieder aktiviert wird. Zu beachten ist beim Übertakten und dem damit erhöhten Stromverbrauch, dass das angeschlossene USB-Netzteil den zusätzlichen Strom auch liefern kann und die 5 V-Spannung nicht zusammenbricht.

Benchmarks haben gezeigt, dass insbesondere die SDRAM-Speicherzugriffe von einem erhöhten Takt profitieren und dass Integer- und Floating-Point-Berechnungen bei 1 GHz fast doppelt so schnell wie bei 700 MHz ausgeführt werden, so dass es – je nach Anwendung – Sinn machen kann, sich (später) etwas intensiver mit dem Overclocking zu beschäftigen.

1.3.12 Erweiterte Optionen – Advanced Options

Unter diesen Punkt sind gleich eine ganze Reihe von Optionen zu finden, die zuvor als separate Einträge in raspi-config zu finden waren. Hinzugefügt wurde an dieser Stelle die Möglichkeit, jetzt auch die Schnittstellen (SPI, I^2C, Serial) aktivieren zu können, was durch das automatische Laden der entsprechenden Kernel-Module durchgeführt wird. Außerdem ist die Konfigurierung des Audioausgangs, der sich zwischen der Übertragung über HDMI oder über die Buchse des analogen Ausgangs umschalten lässt, vorgesehen.

Abbildung 1.22: Unter den Advanced Options sind verschiedene Einstellungs- oder auch nur Aktivierungsmöglichkeiten (A4, A5, A6) zu finden.

1.3.13 Monitorabstimmung – Change Overscan

Overscan stellt eine Abstimmungsfunktion für den Monitor dar, damit die Raspberry Pi-Darstellung auf dem jeweiligen Monitor möglichst optimal erfolgt und das Bild einerseits nicht abgeschnitten und andererseits nicht zu klein wird.

Üblicherweise ist diese Option nur für die Videowiedergabe bzw. bei der Ausgabe über Composite-Video (nicht bei HDMI) von Bedeutung. Bei Overscan passt nicht das gesamte Bild auf den Monitor (es ist zu groß), während bei Underscan nicht die volle Bildschirmgröße ausgenutzt wird (es ist zu klein), so dass ein schwarzer Rahmen um das Bild entsteht.

Die Overscan-Funktion ist direkt im Konfigurationsprogramm verfügbar, während weitere Funktionen für die Video-Anpassung in der Datei /boot/config.txt zugänglich sind. Der Bildbereich kann mit der Overscan-Funktion auch vergrößert werden, indem negative Werte eingegeben werden, bis der Rahmen nicht mehr sichtbar ist.

1.3.14 Computername – Hostname

Der Name des Raspberry Pi, unter dem er in einem LAN erscheint, lässt sich unter der Option *Hostname* angeben. Hierfür sind nur die Standard ASCII-Zeichen erlaubt und die Groß- und Kleinschreibung ist relevant. Der Hostname ist üblicherweise in der Datei *hostname* abgelegt und kann dort auch weiterhin manuell etwa mit `sudo nano /etc/hostname` editiert werden.

1.3.15 Speicheraufteilung – Set Memory Split

Der Raspberry Pi verfügt über 256 MByte oder über 512 MByte an SDRAM-Speicher, der automatisch zwischen dem Prozessor (CPU) und der Grafikeinheit (GPU) aufgeteilt wird. Standardmäßig werden 192 MByte für die CPU und 64 MByte für die CPU reserviert, und zwar möglicherweise auch dann, wenn nicht

nur 256 MByte eingebaut sind, sondern sogar 512 MByte. In diesem Fall liegt die Hälfte des Speichers brach, was dann an einem veralteten Boot-Loader liegt, der nur die ältere 256 MByte-Version (das erste Modell B) kennt. Deshalb ist eine Aktualisierung notwendig, mit der sich auch noch einige weitere Probleme aus der Welt schaffen lassen, was durch ein Firmware-Update (siehe Kapitel 2.6.1) zu bewerkstelligen ist.

Falls der Raspberry Pi für grafikintensive Anwendungen (CAD, Video, Blu-Ray-Wiedergabe) eingesetzt wird, empfiehlt es sich, dem Video Core mehr Speicher zuzuteilen, während der Raspberry Pi als Server nur relativ wenig Grafikspeicher benötigt und deshalb eher von mehr CPU-Speicher profitiert.

Abbildung 1.23: Aufteilung des internen Speichers zwischen CPU und GPU, die hier in der Summe lediglich maximal 256 MByte ergeben.

Entgegen der Darstellung in der Abbildung 1.23 wird der gewünschte Wert für den GPU-Speicher bei den neueren raspi-config-Versionen manuell in einer Zeile eingetragen und vom jeweils vorhandenen SDRAM-Speicher abgezogen. Dabei sind auch unsinnige Angaben möglich, die zwar hier angezeigt, nicht jedoch aktiviert werden. In solch einem Fall werden stattdessen die Defaultwerte eingestellt.

1.3.16 Secure Shell aktivieren – SSH Enable

SSH steht für *Secure Shell* und ist eine Standardapplikation für den sicheren Datenaustausch zwischen zwei Computern, in diesem Fall, um etwa mit einem PC auf den Raspberry Pi zugreifen zu können oder auch umgekehrt. SSH benötigt stets einen Server und als Gegenpart einen Client.

Mit der hier gegebenen SSH-Option kann der SSH-Server auf dem Raspberry Pi aktiviert (Enable) oder deaktiviert (Disable) werden. Um die Funktion nutzen zu können, ist eine Netzwerkverbindung zwischen den beiden SSH-Teilnehmern notwendig, der Ethernet- oder der (optionale) WLAN-Adapter muss über eine zugeteilte IP-Adresse verfügen. Wie die Netzwerkfunktionen eingestellt werden können, ist im Kapitel 4.8 genauer erläutert. Danach lässt sich der Raspberry Pi bequem vom PC aus über diese SSH-Verbindung bedienen und auch konfigurieren. Alle üblichen Kommandozeilen-Programme lassen sich in einem per SSH »ferngesteuerten« Terminalfenster wie gewohnt ausführen.

```
pi@raspberrypi: ~
login as: pi
pi@192.168.0.50's password:
Linux raspberrypi 3.2.27+ #160 PREEMPT Mon Sep 17 23:18:42 BST 2012 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Mar 13 18:08:00 2013 from 192.168.0.11
pi@raspberrypi ~ $ free
             total       used       free     shared    buffers     cached
Mem:        189104      82292     106812          0      12844      50456
-/+ buffers/cache:      18992     170112
Swap:       102396          0     102396
pi@raspberrypi ~ $
```

Abbildung 1.24: Der Raspberry Pi funktioniert im Terminalfenster per SSH genauso wie lokal. Hier wird zum Test der Befehl für die Ermittlung der Speicherbelegung angewendet.

1.3.17 Schnittstellen – SPI, I²C, Serial

Über diese drei Menüpunkte lassen sich drei Schnittstellen (SPI, I²C, Serial) aktivieren, so dass die entsprechenden Kernel-Module automatisch beim Boot geladen werden. Für SPI und I²C sind die notwendigen Treiber generell zwar vorhanden; sie sind jedoch durch eine Blacklist gesperrt (siehe auch Kapitel 6.4.2.1), die auch an dieser Stelle entsperrt werden kann. Die serielle Schnittstelle unterstützt lediglich eine TXD- und eine RXD-Leitung auf TTL-Pegel, es ist also keine RS232-Implementierung, so dass hiermit keine Gerätekopplung möglich ist. Ohne zusätzliche Peripherie eignet sich diese serielle Schnittstelle lediglich für die Chip-zu-Chip-Kommunikation.

Die Serial-Option der Advanced Options erlaubt die Aktivierung der Shell/Message-Funktion, damit sich hierüber Linux-Nachrichten austauschen lassen, so dass hiermit auch eine Remote-Anmeldung (per SSH-Terminal) möglich ist.

1.3.18 Tonausgang bestimmen – Audio

Die Ausgabe von Audiosignalen erfolgt entweder mit einer direkten HDMI-Verbindung oder analog über die Audiobuchse. Standardmäßig wird hierfür HDMI verwendet, wie es auch im Automatikmodus erkannt und aktiviert wird. Falls eine der beiden Ausgabequellen erzwungen werden soll, lässt sich dies über *Audio* bestimmen. Wann dies der Fall ist, hängt von der jeweils verwendeten Software ab.

Abbildung 1.25: Normalerweise ist der Auto-Modus die passende Einstellung.

1.3.19 Aktualisierung – Update

Die Update-Funktion (try to upgrade raspi-config) ist natürlich nur dann möglich, wenn eine Netzwerkverbindung mit einer Verbindung zum Internet vorhanden ist. Unmittelbar und ohne eine weitere Abfrage wird dann nach Betätigung dieser Option ein Update des Konfigurationsprogramms ausgeführt, wobei die bisher hiermit getätigten Einstellungen weiterhin gelten. Es handelt sich hier also lediglich um ein Update von raspi-config und nicht das Systems oder der Firmware, was im Kapitel 2.6.1 erläutert ist.

```
pi@raspberrypi: ~
Ign http://mirrordirector.raspbian.org wheezy/main Translation-de
Ign http://mirrordirector.raspbian.org wheezy/main Translation-en
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-de_DE
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-de
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-en
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-de_DE
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-de
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en
Es wurden 7.499 kB in 58 s geholt (128 kB/s)
Paketlisten werden gelesen... Fertig
Paketlisten werden gelesen... Fertig
Abhängigkeitsbaum wird aufgebaut
Statusinformationen werden eingelesen... Fertig
Die folgenden Pakete werden aktualisiert (Upgrade):
  raspi-config
1 aktualisiert, 0 neu installiert, 0 zu entfernen und 254 nicht aktua
Es müssen 6.654 B an Archiven heruntergeladen werden.
Nach dieser Operation werden 0 B Plattenplatz zusätzlich benutzt.
Hole:1 http://archive.raspberrypi.org/debian/ wheezy/main raspi-confi
Es wurden 6.654 B in 0 s geholt (29,3 kB/s)
(Lese Datenbank ... 57768 Dateien und Verzeichnisse sind derzeit inst
Vorbereitung zum Ersetzen von raspi-config 20120918 (durch .../raspi-
Ersatz für raspi-config wird entpackt ...
raspi-config (20121028) wird eingerichtet ...
Sleeping 5 seconds before reloading raspi-config
```

Abbildung 1.26: Ein Update von raspi-config wird durchgeführt.

1.4 New Out Of Box Software – NOOBS

Seit Mitte des Jahres 2013 stellt die *Raspberry Pi Foundation* ein Paket mit sechs verschiedenen Betriebssystemen zur Verfügung, welches einen Installer (NOOBS) bietet, um wahlweise und ohne großen Aufwand eines dieser Systeme installieren zu können. In NOOBS sind die Linux-Betriebssysteme Arch Linux, Pidora Linux und Raspbian enthalten sowie RISC OS und die beiden Mediacenterlösungen OpenELEC und RaspBMC.

Die Systeme sind als gepackte TAR-Archive angelegt, die ca. 80 MByte (Arch Linux) bis hin zu 600 MByte (Raspbian) an Speicherplatz in Anspruch nehmen. Diese werden während der Installation automatisch ausgepackt und in einer separaten Partition installiert. Das komplette Paket (1,5 GByte als ZIP-Datei, v1.3.9) lässt sich aus dem Internet bei der *Raspberry Pi Foundation* beziehen und ist nach dem Entpacken auf eine SD Card zu kopieren, von der aus dann die NOOB-Installation stattfindet. Demnach ist hierfür keine Internetverbindung notwendig. Für die Verarbeitung von Images empfiehlt sich das Programm *Win32 Disk Imager*, auf das im Kapitel 4.2 noch etwas näher eingegangen wird.

Abbildung 1.27: Auswahl des gewünschten Betriebssystems

Die SD Card sollte mindestens über eine Kapazität von 4 GByte verfügen. Empfehlenswert sind allerdings 8 GByte, damit noch etwas Platz für die nachfolgend zu installierende Software und die eigenen Dateien zur Verfügung steht. NOOBS wird auch mit bereits vorhandenem Installer und den genannten OS-Paketen auf einer SD Card verkauft, wobei die neueren Karten eine Kapazität von 8 GByte aufweisen. Die SD Card ist (zunächst) allein FAT32-formatiert und enthält eine einzige Partition, was bei der Installation des Betriebssystems automatisch für die neue (Linux-) Partition angepasst wird, so dass die FAT32-Partition verkleinert (auf ca. 1,43 GByte) und das System auf eine eigene Partition (ext4) installiert wird.

Nach dem ersten Boot mit der SD Card erscheinen kurz die Meldungen *Formating - Resizing* und *please wait while NOOBS initialises*, woraufhin die sechs Systeme zur Auswahl (Abbildung 1.27) sowie zwei weitere Optionen (Boot to Scratch, Add Data Partition) angeboten werden. In dieser Programmschleife lässt sich das gewünschte System selektieren sowie unterhalb des Auswahlfensters die gewünschte Sprache und das Tastatur-Layout. Nach der Betätigung von *Install* und der Nachfrage, ob das Image geschrieben werden soll, weil dadurch alle Daten auf der SD-Karte verloren gehen, werden die Dateien hierfür entpackt und das System wird so in den freien Bereich der SD Card installiert, damit es nachfolgend automatisch startet.

Die Warnung, dass beim Schreiben des Image auf die SD-Karte alle Daten gelöscht werden, ist etwas missverständlich. Es gehen eben nicht alle Daten auf der SD-Karte verloren, sondern nur die, die sich auf der Linux-Partition befinden. Es handelt sich also um die Partition des bereits installierten Systems, wobei diese Partiti-

on beim erstmaligen Aufspielen noch gar nicht existiert, also während des Installationsvorganges erst angelegt wird.

In das NOOBS-Auswahlmenü gelangt man erneut, wenn die Shift-Taste während des Bootens nach der Meldung mehrmals schnell hintereinander betätigt wird, woraufhin ein neues System installiert und daraufhin ausprobiert werden kann. Diese Funktionalität wird als *Recovery Mode* bezeichnet.

Abbildung 1.28: Nach der Warnung wird Raspbian installiert, woraufhin ein Recovery Mode für die erneute Auswahl eines Betriebssystems zur Verfügung steht.

Vor dem Erscheinen des Auswahlmenüs (Abbildung 1.27) erscheint möglicherweise eine Fehlermeldung (Error resizing existing FAT partition) mit dem Hinweis darauf, dass das auf der SD-Karte befindliche Tool (usr/sbin/parted) für die Größenänderung der Partition nicht korrekt funktioniert, so dass stattdessen spezielle Filesystem-Tools wie *e2fsprogs* verwendet werden sollten. Eine Installation ist dann nicht möglich, und als verfügbarer Speicherplatz werden unter Disk *Space 0 Bytes Available* ausgewiesen. In den meisten Fällen ist dieser Fehler auf eine nicht korrekt funktionierende SD-Karte zurückzuführen, was sich durch eine Neuformatierung (keine Schnellformatierung) im FAT32-Format mit dem anschließenden erneuten Kopieren der NOOBS-Daten auf die SD-Karte beheben lässt. Dies ist einfach mit der Format-Funktion von Windows durchführbar. Ein spezielles Formatierungs-Tool wird für Windows nicht benötigt.

Nach der Verwendung von NOOBS ist es bei Raspbian nicht notwendig, mithilfe von raspi-config, welches automatisch nach dem ersten Boot mit dem Betriebssystem erscheint, das Root-Dateisystem zu erweitern oder länderspezifische Einstellungen vorzunehmen, weil dies bereits zuvor automatisch erledigt bzw. festgelegt worden ist.

2 Software

In diesem Kapitel werden einige grundlegende Dinge für den Umgang mit Software für den Raspberry Pi erläutert. Als Festspeicher für die Software wird die eingesetzte SD-Karte (2 - 32 GByte) verwendet. Darauf befinden sich verschiedene Partitionen, die während der Installation des Betriebssystems angelegt worden sind. Das Standardbetriebssystem für den Raspberry Pi ist aus *Debian Wheezy* entstanden, bei dem es sich um eine speziell angepasste Version für den Raspberry Pi – den Broadcom BCM2835 – handelt und welches als *Raspbian* bezeichnet wird. Es lässt sich verhältnismäßig einfach konfigurieren und einsetzen, wie es im vorherigen Kapitel gezeigt wurde. Es unterstützt auch die *Floating Point Unit* des BCM 2835 standardmäßig, was bei anderen Systemen, etwa bei Debian Squezze, nicht der Fall ist. Raspbian stellt gewissermaßen das Allround-System für den Raspberry Pi ohne Spezialisierung auf bestimmte Applikationsschwerpunkte dar. Näheres zu den verschiedenen Betriebssystemen, die sich für den Raspberry Pi eignen und für welche Anwendung und Benutzertypen sie explizit vorgesehen sind, ist im Kapitel 4 erläutert.

2.1 Dateisystem und erste Software-Installation

Um sich einen ersten (groben) Überblick über das Dateisystem und die Daten auf der SD-Karte zu verschaffen, empfiehlt sich die Anwendung des Befehls *df*, was für *Disk Free* steht und mehrere Optionen kennt. Welche dies im Einzelnen sind, lässt sich durch das Voranstellen des Kürzels *man*, was für Manual, also Anleitung steht, herausfinden, was ganz allgemein für die üblichen Linux-Befehle gilt.

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1_3, © Springer Fachmedien Wiesbaden 2015

```
pi@raspberrypi ~ $ man df
DF(1)                          User Commands                          DF(1)

NAME
       df - report file system disk space usage

SYNOPSIS
       df [OPTION]... [FILE]...

DESCRIPTION
       This  manual  page  documents  the  GNU version of df.  df displays the
       amount of disk space available on the file system containing each  file
       name  argument.   If  no file name is given, the space available on all
       currently mounted file systems is shown.  Disk space  is  shown  in  1K
       blocks  by  default, unless the environment variable POSIXLY_CORRECT is
       set, in which case 512-byte blocks are used.

       If an argument is the absolute file name of a disk device node contain
       ing  a  mounted  file system, df shows the space available on that file
       system rather than on the file system containing the device node (which
       is  always  the  root file system).  This version of df cannot show the
       space available on unmounted file systems, because  on  most  kinds  of
       systems  doing  so requires very nonportable intimate knowledge of file
       system structures.

OPTIONS
       Show information about the file system on which each FILE  resides,  or
       all file systems by default.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              include dummy file systems

       -B, --block-size=SIZE
              scale sizes by SIZE before printing them.   E.g.,  `-BM'  prints
              sizes in units of 1,048,576 bytes.  See SIZE format below.

       --total
              produce a grand total

       -h, --human-readable
              print sizes in human readable format (e.g., 1K 234M 2G)

       -H, --si
              likewise, but use powers of 1000 not 1024

       -i, --inodes
              list inode information instead of block usage
```

Abbildung 2.1: Zu jedem gewöhnlichen Linux-Befehl gibt es eine Erläuterung oder Hilfedatei, indem dem jeweiligen Befehl »man« vorangestellt wird.

Mit der Option *-h* wird beim df-Befehl eine einfache Ausgabe (human readable) erreicht, wie es in der Abbildung 2.2 zu erkennen ist. Die Überschrift der ausgegebenen Tabelle kennzeichnet die gesamte Größe einer Partition (in Gigabyte), *BENUTZT* gibt an, wie viel davon in Gebrauch sind, *VERF.* zeigt, wie viel Speicher

noch zur Verfügung steht, gefolgt von einer Prozentangabe für die aktuelle Verwendung (*VERW %*) und an welcher Stelle die Partitionen oder Verzeichnisse *eingehängt* sind. Der Begriff »eingehängt« soll besagen, an welcher Stelle des Dateisystems die Verzeichnisse oder Partitionen lokalisiert sind, es handelt sich dabei gegewissermaßen um eine Pfadangabe, auch wenn es sich dabei nicht um Pfade oder Verzeichnisse handelt wie sie von DOS oder Windows her bekannt sind. Die Laufwerke werden unter Linux stets »gemountet«, d.h., sie werden in den Dateibaum unter Linux eingehängt (mount /dev/sdd1) und können dementsprechend auch wieder aus dem Dateibaum herausgenommen werden (umount /dev/sdd1).

```
pi@raspberrypi ~ $ df -h
Dateisystem     Größe Benutzt Verf. Verw% Eingehängt auf
rootfs           3,6G    1,3G  2,2G   39% /
/dev/root        3,6G    1,3G  2,2G   39% /
devtmpfs          93M       0   93M    0% /dev
tmpfs             19M    232K   19M    2% /run
tmpfs            5,0M       0  5,0M    0% /run/lock
tmpfs             37M       0   37M    0% /run/shm
/dev/mmcblk0p1    56M     36M   21M   63% /boot
```

Abbildung 2.2: Die Belegung der 4 GByte-SD Card

Unter Linux werden alle Geräte wie Festplatten oder Schnittstellen als Dateien bzw. Verzeichnisse geführt. Die hierfür geltenden Bezeichnungen sind mehr oder weniger standardisiert. Es gibt hier generell kein Laufwerk A: (Diskettenlaufwerk) oder ein Laufwerk C: (Festplatte), sondern spezielle Bezeichnungen, wie beispielsweise */dev/fd0* für ein Diskettenlaufwerk oder /dev/hda für eine Festplatte. Beides ist beim Raspberry Pi nicht vorhanden, sondern nur die SD Card, die als */dev/root* und */dev/mmcblk0p1* auftaucht, wobei letzteres der Boot-Partition entspricht. Die Bezeichnung »mmcblk0p1« ist während der Installation des Betriebssystems angelegt worden und könnte auch eine einfachere Bezeichnung wie eben sdd1 oder sda tragen. Falls (später) weitere Laufwerke hinzugefügt werden, was beispielsweise ein USB-Stick sein kann, wird das zugrunde gelegte Bezeichnungsschema weitergeführt. Das Laufwerk erhält eine Bezeichnung wie *sdd2*, *sdb* oder eben *mmcblk0p2* und kann über das Mounten (mount /dev/sdd2) in den Linux-Dateibaum eingehängt werden.

Nur der Boot-Teil mit 56 MByte (vgl. Abbildung 2.2 und 2.3), wovon 36 MByte belegt sind, lässt sich auch unter einem anderen Betriebssystem wie Windows anzeigen. Einige Dateien (z.B. config.txt) können dann auch mit einem Editor gelesen und angepasst werden. Die Boot-Partition ist im üblichen FAT32-Format (File Allocation Table) angelegt, während der Rest von 3,64 GByte das – für Windows nicht lesbare – Linux-Format (ext4) verwendet.

Abbildung 2.3: Unter Windows kann lediglich die Bootpartition der für Linux vorgesehenen SD-Karte angezeigt werden.

Neben der Boot-Partition gibt es demnach die Linux-Partition */dev/root* (Abbildung 2.2), dort sind mehrere tmp-Pfade »eingehängt«, wobei die mit »run« bezeichneten während der Laufzeit erzeugt werden; dies sind also keine physikalisch vorhandenen, sondern sie werden automatisch im SDRAM-Speicher erzeugt und verwaltet und entsprechend in den Linux-Dateibaum eingehängt. Alle weiteren Partitionen werden grundsätzlich unterhalb des Root-Verzeichnisses eingehängt, wodurch sich die Baumstruktur ergibt, die mit dem Befehl *tree* angezeigt werden kann. Standardmäßig ist *tree* bei Raspbian nicht installiert, was mit der folgenden Zeile bei vorhandener Internet-Verbindung nachgeholt werden kann:

```
pi@raspberrypi ~ $ sudo apt-get install tree
```

Hier wird bereits ein neues Programm auf dem System installiert, was mithilfe des *Advanced Packaging Tool* (APT) erfolgt. Dies ist ein Paketmanager, der das Werkzeug schlechthin für das Hinzufügen, Aktualisieren und Entfernen von Software darstellt. Grundsätzlich sollte sich die Softwaredatenbank (*Repository* siehe Kapitel 2.5) stets auf aktuellem Stand befinden, damit Software-Installationen korrekt funktionieren, weshalb die folgende Zeile stets vor jedem Install-Vorgang auszuführen ist:

```
pi@raspberrypi ~ $ sudo apt-get update
```

Beide Zeilen bzw. Befehle werden zusammengefasst mit:

```
pi@raspberrypi ~ $ sudo apt-get update && apt-get install tree
```

Im Laufe des Buches wird dieses Tool noch des Öfteren eingesetzt, es wird auch noch etwas genauer darauf eingegangen (siehe Kapitel 2.5), während im Folgenden einige grundlegende Dinge zu Linux erläutert werden.

2.2 Verzeichnisstruktur

Wie jedes Betriebssystem kennt auch Linux Dateien und Verzeichnisse, wobei hier Ähnlichkeiten zu DOS nicht von ungefähr kommen, denn DOS stellt gewissermaßen ein stark reduziertes UNIX dar – und Linux ist eine UNIX-Variante.

DOS kennt allerdings weder Multitasking noch einen Multiuser-Betrieb und bringt standardmäßig auch keinerlei Netzwerkunterstützung mit. Außerdem unterliegt Linux nicht der 8.3-Bezeichnungskonvention (d.h. die Bezeichnung einer Datei darf hier nur aus maximal 8 Buchstaben plus einer dreistelligen Endung wie etwa *.doc* oder *.com* bestehen). Es wird generell zwischen der Groß- und Kleinschreibung unterschieden, was DOS- und auch Windows-Anwender als Linux-Einsteiger immer wieder zu fehlerhaften Angaben verleitet.

Spezielle Endungen (Extensions) für ausführbare Dateien gibt es bei Linux nicht, diese werden hier durch einen Stern (*) gekennzeichnet. Ein weiterer wichtiger Unterschied zu DOS ist die Angabe von Pfaden bzw. Unterverzeichnissen. Um unter DOS zu einem Unterverzeichnis *Briefe* im Verzeichnis Texte zu wechseln, wäre *cd \texte\briefe* einzugeben, während dies unter Linux mit *cd /texte/briefe* zu erledigen ist. Statt des Backslash »\« wird also der Schrägstrich »/« verwendet. Dies sollte man keineswegs verwechseln, da es schwerwiegende Konsequenzen haben kann, etwa bei der verzeichnisweisen Umbenennung von Dateien oder der Veränderung von Dateiattributen.

Die folgende Tabelle zeigt die bei Linux vorhandenen Verzeichnisse, wie sie auch bei Raspbian (Raspbian) vorhanden sind, mit einer kurzen Erläuterung zu den wesentlichen Inhalten. In Abhängigkeit von der jeweiligen Distribution kann es noch weitere Verzeichnisse geben, oder, es sind möglicherweise auch einige nicht vorhanden, oder einige sind auch leer, was für das grundlegende Verständnis aber keine Rolle spielt.

Tabelle 2.1: Verzeichnisse unter Linux und was sie im Wesentlichen beinhalten

Verzeichnis	Bedeutung	Funktion/Inhalt
/..	Verzeichnis	Das Hauptverzeichnis, welches alle anderen als Unterverzeichnisse enthält.
/bin	Binary	Enthält die für alle Benutzer zugänglichen Befehle sowie Binärfiles für den Desktop.
/boot	Boot	Bootsektorinformationen und die Kernel-Dateien. Im Grunde genommen sind hier alle Daten zu finden, die für den Systemstart benötigt werden.
/dev	Devices	Hier sind die Gerätedateien (Treiber) der Linux-Distribution abgelegt. Sie erlauben den direkten Zugriff auf bestimmte Hard- und Software-Funktionen (z.B. Sound, HDMI).
/etc	Ecetera	Enthält Informations- und Konfigurationsdateien, wie z.B. für die Hardware- und Benutzerverwaltung und die verschlüsselten Passwörter.
/home	Home Directories	Dieses Verzeichnis enthält für jeden im System eingerichteten Benutzer ein eigenes Unterverzeichnis. Standardmäßig existiert hier zunächst nur der User pi.
/lib	Library	Hier sind die verschiedenen Bibliotheken zu finden, die erst dann (zur Laufzeit) geladen werden, wenn sie von einer Anwendung benötigt werden.
/lost+found	Lost and Found	Dient als »Papierkorb« für Dateien, die noch im System vorhanden sind, aber deren Verzeichniseinträge gelöscht worden sind. Der direkte Zugriff ist automatisch nur dem User *root* gestattet.
/media	Media	Ein spezielles Verzeichnis für das Mounten von Wechseldatenträgern wie USB-Sticks oder CD/DVD-Laufwerken.
/mnt	Mount	Dies ist das allgemeine Verzeichnis für die Aufnahme von Dateisystemen, die im System »gemountet« werden.
/opt	Optional	Hier wird zusätzliche Software gespeichert, die nicht direkter Bestandteil des Betriebssystems ist.
/proc	Processes	Enthält Informationen über die Konfiguration (z.B. CPU-Typ, USB-Devices) und laufende Prozesse, die von Programmen ausgewertet werden können.

Verzeichnis	Bedeutung	Funktion/Inhalt
/root	Root	Das Verzeichnis des Benutzers *root*.
/run	Run	In diesem Verzeichnis werden zur Laufzeit benötigte Daten abgelegt, die in */tmp* nicht sicher genug sind, weil sie dort versehentlich gelöscht werden könnten.
/sbin	System Binary	Programme für die System- und Benutzerverwaltung sowie diverse Systemdienste, die für die Administration (root) vorgesehen sind.
/selinux	Security Enhanced	Optionale spezielle Sicherheitsprogramme. Typischerweise ist dieses Verzeichnis leer.
/srv	Services	Hier befinden sich Daten, die von Diensten des Systems stammen. Falls keine Server-Dienste wie ein Web- oder FTP-Server genutzt werden, ist das Verzeichnis leer.
/tmp	Temporary	Ein temporäres Verzeichnis für die vorübergehende Ablage von Daten.
/usr	User	In diesem Verzeichnis ist der größte Anteil der installierten Software zu finden. Die hier lokalisierten Unterverzeichnisse wie */usr/bin* oder */usr/lib* entsprechen in ihren Funktionen in großen Teilen denjenigen Verzeichnissen, die auch direkt zugänglich sind, und ergänzen sich gewissermaßen.
/var	Variables	Dieses Verzeichnis enthält Konfigurationsdateien und auch veränderbare Anteile des Betriebssystems.

Wie erwähnt, stellt das Kommando *tree* den Verzeichnisbaum dar, der in Abhängigkeit von dem Verzeichnis, von dem er gerade aufgerufen wird, mehr oder weniger ausführlich erscheint, wobei es auch hierfür wieder einige Optionen (siehe *man tree*) gibt. Beispielsweise werden durch die Option *-d* nur die Verzeichnisse, nicht jedoch die darin befindlichen Daten angezeigt.

Die Textausgabe ist bei vielen Befehlen oftmals länger als eine Bildschirmseite, so dass der Text möglicherweise seitenweise »vorbeirauscht«, was dementsprechend auch für die Tree-Ausgabe gilt, Damit eine Ausgabe seitenweise erfolgt, kann dem jeweiligen Befehl das Zeichen »|« (Strg Alt + <) nachgestellt werden, woraufhin die Anzeige nach jeder Seite anhält (More) und erst durch eine Tastenbetätigung die nächste erscheint.

```
pi@raspberrypi / $ tree -d | more
.
├── bin
├── boot
├── dev
│   ├── block
│   ├── bus
│   │   └── usb
│   │       └── 001
│   ├── char
│   ├── disk
│   │   ├── by-id
│   │   └── by-uuid
│   ├── fd -> /proc/self/fd
│   ├── input
│   │   ├── by-id
│   │   └── by-path
│   ├── mapper
│   ├── net
│   ├── pts
│   ├── raw
│   ├── shm -> /run/shm
│   └── snd
│       └── by-path
--More--
```

Abbildung 2.4: Seitenweise Anzeige des Verzeichnisbaumes mit tree

2.3 Linux-Orientierung und Befehle

Das beim Raspberry Pi standardmäßig verwendete Betriebssystem ist Raspbian, welches einer angepassten Debian-Linux-Version (von Squezze Version 6) entspricht. Die verschiedenen Linux-Distributionen, wie beispielsweise Ubuntu, welches ebenfalls auf Debian basiert, oder Red-Hat oder auch SuSE haben alle einen gemeinsamen Linux-Kern, der in Abhängigkeit von der jeweiligen Distribution über unterschiedliche Programme, Werkzeuge und besondere Ausstattungsmerkmale verfügt.

Deshalb sind die meisten der grundlegenden Befehle auf der Kommandozeile (Terminalmodus) bei allen Linux-Versionen gleich und gehorchen dem zuvor erläuterten Partitions- und Verzeichnisgefüge, wodurch der Raspberry Pi für Linux-Kenner eine vertraute Umgebung bietet. Für alle anderen Benutzer werden in diesem Kapitel einige nützliche Linux-spezifische Dinge zusammengefasst dargestellt.

- **Kommandozeilen-Historie:** Die eingegebenen Kommandozeilen werden in einer Command History-Liste geführt, so dass sie ganz einfach über die Betätigung der Pfeil-Auf-Taste » ↑« erneut zur Verfügung stehen. Die Liste reicht mit ca. 100 Einträgen über verschiedene Sessions hinweg und ist als Datei

bash_history unter */home/pi* zu finden. Mit der Pfeil-Ab-Taste » ↓« bewegt man sich in der Liste wieder in Vorwärtsrichtung.

- **Kommandozeilen-Komplettierung:** Kommandos sowie auch Pfade und Dateinamen müssen nicht stets ausgeschrieben werden, sondern es reicht die Eingabe der Anfangsbuchstaben und die Betätigung der TAB-Taste, woraufhin eine automatische Vervollständigung vorgeschlagen wird.
- **Hilfe:** Zu fast jedem Linux-Befehl existiert eine Hilfedatei, die dadurch aufgerufen wird, indem dem Kommando das Kürzel »man« vorangestellt wird (z.B. *man df*).
- **Programme oder Ausführung beenden:** Je nach Befehl bzw. Programm gibt es hierfür verschiedene Tastenkombinationen, fast immer funktioniert die Kombination Strg+C.
- **Terminal/Konsole wechseln:** Linux unterstützt gleichzeitig mehrere Terminalsitzungen, zwischen denen mithilfe Strg+Alt+F1 (erstes Terminal) bis Strg+Alt+F6 (sechstes Terminal) umgeschaltet werden kann. Das Ausloggen aus dem jeweiligen Terminal kann über Strg+D erfolgen.
- **Starten und Beenden des Desktops:** Mit *startx* wird der Desktop (LXDE) gestartet und beendet durch die Selektierung von Logout (rotes Symbol in der rechten Ecke der Taskleiste). Falls dies während der Konfiguration mit raspi-config festgelegt wurde, wird der Desktop außerdem über die Tastenkombination Strg+Alt+Entf beendet.
- **Ausführungsrechte:** Einige Befehle lassen sich nur mit Root-Rechten ausführen, die der gewöhnliche Benutzer nicht ausführen kann. Um nur für die Ausführung bestimmter Befehle Superuser-Rechte zu erlangen, wird dem jeweiligen Befehl *sudo* vorangestellt, was als »super user do« zu verstehen ist.
- **Beenden mit Reboot:** Um Linux zu beenden und unmittelbar danach einen Neuboot auszuführen, ist *sudo reboot* einzugeben oder die Tastenkombination Strg+Alt+Entf auszuführen.
- **Herunterfahren:** Eingabe von *sudo halt*.

Viele Arbeiten lassen sich unter Linux im Terminalmodus, also ohne grafische Oberfläche, durchführen, was demgegenüber meist schneller vonstattengeht. Wirklich wichtig für diese grundlegenden Linux-Operationen sind eigentlich nur recht wenige Befehle, die mit einigen Beispielen in der Tabelle 2.2 angegeben sind.

Tabelle 2.2: Wichtige Befehle und Tastenkombinationen für die Bedienung von Linux auf Konsolenebene

Befehl/Tasten	Funktion	Einige Optionen und Beispiele

cat	Dateiinhalte anzeigen	`cat /proc/cpuinfo`
		Informationen über den Prozessor anzeigen
		`cat /etc/hosts`
		Anzeige der Netzwerk-Hosts

cd /verzeichnis	Wechseln in das Verzeichnis	`cd ..` geht ein Verzeichnis zurück (hoch) `cd ~` geht in das Hauptverzeichnis zu beachten ist der Abstand zwischen cd und dem folgenden Zeichen!
chgrp	Ändern der Gruppenzugehörigkeit von Dateien oder Verzeichnissen	`chgrp donald /home/kd/*` alle Dateien im Verzeichnis /home/kd werden der Gruppe *donald* zugeordnet.
chmod	Zugriffsrechte für Dateien und Verzeichnisse ändern	`chmod g+x test` Ausführungsrecht (x) der Datei *test* der Gruppe (g) zuweisen (+)
chown	Eigentümer von Dateien ändern	`chown gruppe1 *.doc` alle doc-Dateien werden der *gruppe1* als zugehörig deklariert
clear	Bildschirm löschen	`clear`
cp	Kopieren von Dateien	`cp datei1 datei2` die Datei1 wird in die Datei2 kopiert -b, falls die Zieldatei bereits besteht, wird eine Sicherheitskopie erstellt.
df	Zeigt den freien Speicherplatz an.	-h, einfach lesbar in k-, M- oder GByte -k, Anzeige in kByte -T, zeigt auch den Typ (Filesystem) an
free	Zeigt die RAM-Größe der CPU an.	-h, einfach lesbar in k-, M- oder GByte -m, Anzeige in MByte
ls	Inhalt des aktuellen Verzeichnisses anzeigen	-a, versteckte Dateien mit anzeigen -l, erweiterte Anzeige, mit Zugriffsrechten

Befehl/Tasten	Funktion	Einige Optionen und Beispiele
kill	Beenden von hängenden Prozessen	`kill 1222` den Prozess mit der PID (Prozess-ID) 1222 beenden (vgl. Abbildung 2.6).
mkdir *verzeichnis*	Anlegen eines Verzeichnisses	`mkdir Texte` Anlegen des Verzeichnisses *Texte*
mount	Einhängen (mounten) von Dateisystemen, die Voreinstellungen sind in der Datei */etc/.fstab* zu finden	`mount /mnt/cdrom` mounten einer CD im Verzeichnis */mnt/cd*
rm datei	Löschen von Dateien	Das Löschen erfolgt ohne vorherige Sicherheitsabfrage – und es gibt keine Undelete-Funktion!
rmdir *verzeichnis*	Löschen eines Verzeichnisses	`rmdir Texte` Löschen des Verzeichnisses Texte, es muss aber leer sein
pwd	Anzeige des aktuellen Pfads	(falls der Überblick über das aktuelle Verzeichnis verloren gegangen sein sollte)
tail	Anzeige der letzten Zeilen einer Datei	`-f` Kontinuierliche Anzeige, bis die Ausgabe über Strg+C beendet wird. Hilfreich zur Analyse von Log-Dateien.
top	Zeigt die aktuellen CPU-Prozesse an	`-n zahl` Zahl = Anzahl der Wiederholungen (vgl. Abbildung 2.6)
tree	Anzeige des Verzeichnisbaumes	`-d` Es werden nur die Verzeichnisse und nicht die Dateien angezeigt.
umount	Unmounten (abhängen) von Dateisystemen, die Voreinstellungen sind in der Datei */etc/.fstab* zu finden.	`umount /mnt/CD-ROM` Unmounten der CD im Verzeichnis */mnt/cdrom* (Das »n« ist im Befehl tatsächlich nicht vorhanden.)

```
pi@raspberrypi ~ $ top -n1
top - 16:30:15 up 43 min,  3 users,  load average: 0,02, 0,05, 0,06
Tasks:  78 total,   1 running,  77 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2,1 us,  1,4 sy,  0,0 ni, 96,3 id,  0,1 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:    189104 total,   120236 used,    68868 free,    13340 buffers
KiB Swap:   102396 total,        0 used,   102396 free,    59172 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 2233 pi        20   0  4644 1320  964 R  16,3  0,7   0:00.07 top
    1 root      20   0  2136  732  624 S   0,0  0,4   0:01.79 init
    2 root      20   0     0    0    0 S   0,0  0,0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0,0  0,0   0:00.00 ksoftirqd/0
    4 root      20   0     0    0    0 S   0,0  0,0   0:00.02 kworker/0:0
    5 root      20   0     0    0    0 S   0,0  0,0   0:00.36 kworker/u:0
    6 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 khelper
    7 root      20   0     0    0    0 S   0,0  0,0   0:00.00 kdevtmpfs
    8 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 netns
    9 root      20   0     0    0    0 S   0,0  0,0   0:00.02 sync_supers
   10 root      20   0     0    0    0 S   0,0  0,0   0:00.00 bdi-default
   11 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 kblockd
   12 root      20   0     0    0    0 S   0,0  0,0   0:00.55 khubd
   13 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 rpciod
   15 root      20   0     0    0    0 S   0,0  0,0   0:00.00 khungtaskd
   16 root      20   0     0    0    0 S   0,0  0,0   0:00.00 kswapd0
   17 root      20   0     0    0    0 S   0,0  0,0   0:00.00 fsnotify_mark
   18 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 nfsiod
   19 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 crypto
   26 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 kthrotld
   27 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 VCHIQ-0
   28 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 VCHIQr-0
   29 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 dwc_otg
   30 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 DWC Notificatio
   31 root      20   0     0    0    0 S   0,0  0,0   0:00.00 kworker/u:1
   32 root      20   0     0    0    0 S   0,0  0,0   0:06.45 mmcqd/0
   33 root      20   0     0    0    0 S   0,0  0,0   0:00.02 jbd2/mmcblk0p2-
   34 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 ext4-dio-unwrit
  138 root      20   0  2872 1284  752 S   0,0  0,7   0:00.47 udevd
  310 root      20   0     0    0    0 S   0,0  0,0   0:00.01 flush-179:0
  608 root      20   0  2868  964  428 S   0,0  0,5   0:00.00 udevd
  609 root      20   0  2868  904  372 S   0,0  0,5   0:00.00 udevd
 1376 root      20   0  1736  512  420 S   0,0  0,3   0:01.47 ifplugd
 1435 root      20   0  1736  504  420 S   0,0  0,3   0:00.30 ifplugd
pi@raspberrypi ~ $
```

Abbildung 2.5: Anzeige der aktiven Prozesse mit »top«. Mit dem Befehl »kill« können hängende Prozesse (ausgewiesen durch PID) beendet werden.

Ein empfehlenswertes Programm für alle möglichen Aufgaben im Text-Modus stellt der *Midnight Commander* (Aufruf mit mc) dar. Dies ist ein typischer Dateimanager für das Anlegen, Kopieren, Löschen und Umbenennen von Dateien und Verzeichnissen sowie für das Editieren von Dateiattributen.

Jemand, der noch aus DOS-Zeiten den bekannten Norton Commander oder einen Windows-Ableger davon (Windows-, Total-Commander) kennt, kommt auch sofort mit dem Midnight Commander zurecht, mit dem sich im Übrigen recht kom-

fortabel Dateiattribute und Besitzrechte ändern lassen. Falls der Midnight Commander mit einem Terminal (LXTerminal oder über SSH) vom Desktop ausgeführt wird, lässt er sich auch mit der Maus bedienen. Von »Hause« aus ist der Midnight Commander nicht installiert, was sich bei aktiver Internet-Verbindung mit der folgenden Zeile jedoch ganz einfach durchführen lässt:

```
pi@raspberrypi ~ $ sudo apt-get install mc
```

Abbildung 2.6: Der Midnight Commander ist ein sehr praktischer Dateimanager mit einer Vielzahl von nützlichen Funktionen.

2.4 Zugriffsrechte

Zur Anzeige eines Verzeichnisinhaltes kann der Befehl *ls* (vgl. auch Tabelle 2.2) verwendet werden, der auch einige Optionen kennt. Nach der Eingabe von *ls -l* erhält man eine erweiterte Anzeige (hier unter */home/pi*), beispielsweise in der folgenden Form:

drwxr-xr-x	2 pi	pi	4096	Sep	18	14:34	Desktop
drwxr-xr-x	3 pi	root	4096	Sep	18	14:45	Documents
drwxrw-xr-x	2 pi	pi	4096	Jul	20	20:12	python_games
drwxr-xr-x	2 pi	root	4096	Sep	18	14:44	Scratch

Anhand dieser Darstellung kann eine ganze Reihe von Eigenschaften abgelesen werden.

Erstes Zeichen (Typ):

d = Verzeichnis (directory)

- = Datei (data)

l = Verknüpfung (link)

Der Beispielanzeige nach handelt es sich jeweils um Verzeichnisse. Nach der Typangabe folgt unmittelbar die sogenannte *Rechtemaske*, die jeweils drei Stellen umfasst und die Rechte des Eigentümers, der Gruppe und aller weiteren Anwender (Andere) kennzeichnet:

Rechtemaske:

r = Leseerlaubnis

w = Schreiberlaubnis

x = Ausführungsrechte bei einer Datei bzw. das Recht zum Verzeichniswechsel bei einem Verzeichnis.

Eine Maske der Form -rwx -r-- --- bedeutet demnach, dass der Eigentümer die Datei (-) lesen, beschreiben und ausführen (rwx) kann, die Gruppe diese Datei nur lesen kann (r--) und alle anderen Benutzer mit dieser Datei gar nichts anfangen (---) können.

Ein Beispiel für eine Rechtemaske bei einer Datei:

Typ	**Eigentümer**			**Gruppe**			**Andere**		
-	r	w	x	r	-	-	-	-	-

Ein Beispiel für eine Rechtemaske bei einem Verzeichnis:

Typ	**Eigentümer**			**Gruppe**			**Andere**		
d	r	w	x	r	-	x	r	-	-

Mit Verzeichnissen und Links funktioniert dies analog zu einer Datei, lediglich der Typ ist dann ein anderer (d oder l). Ein »x« kennzeichnet in diesem Fall, dass die jeweiligen Benutzer (Eigentümer, Gruppe, Andere) in dieses Verzeichnis wechseln können. Die auf die Rechtmaske folgende Zahl gibt die Anzahl der Verzeichniseinträge an, unter denen die Datei ansprechbar ist, d.h., falls die Zahl größer Eins ist, existieren Verknüpfungen, die auf diese Datei verweisen.

Danach folgen der Eigentümer- und der Gruppenname sowie die Größe der Datei in Byte. Die letzten Einträge kennzeichnen das Datum und die Uhrzeit der letzten Änderung und die Bezeichnung des Datei- oder eben auch des Verzeichnisnamens.

Jede Datei gehört demnach einem bestimmten Benutzer. Was andere mit dieser Datei anfangen können, wird eindeutig durch die Zugriffsrechte definiert. Für die Veränderung der Zugriffsrechte (oder auch der Attribute im DOS/Windows-Sinne) einer Datei gibt es den Befehl *chmod* (change mode), für die Veränderung des Besitzers den Befehl *chown* (change owner) und die der Gruppenzugehörigkeit *chgrp* (change group). Weil sich die Rechtezuordnung auch komfortabel mit dem Midnight Commander oder einen anderen Tool vornehmen lässt, wird hierauf nicht weiter eingegangen.

Die Veränderungen können jedoch nur vom Eigentümer oder vom Benutzer root durchgeführt werden. Sie sollten stets mit Bedacht bei definitiv bekannten Dateien durchgeführt werden, denn die Rechteveränderung kann im Nachhinein durchaus zu Systemfehlern führen (keine Datenaktualisierung mehr möglich o.ä.), wenn man etwa aus Versehen alle Daten in einem Verzeichnis dem Zugriff des alleinigen Benutzers *root* entzieht oder schreibgeschützte Dateien danach nicht mehr geschützt sind.

Bei dieser Rechtzuordnung bleibt der so genannte Superuser (sudo) unberücksichtigt, denn er ist gewissermaßen nachträglich in Linux eingeführt worden, was keinen direkten Einfluss auf die Rechtezuordnung im Dateisystem zur Folge hat. Mit *sudo* können lediglich ganz bestimmte Befehle für ausgewählte Benutzer (z.B. pi), mit Root-Rechten ausgeführt werden.

2.5 Verwaltung und Paketmanager

Für das Betriebssystem Linux gibt es Programme und Tools für alle erdenklichen Aufgaben und Anwendungen. Einiges an Software wird bereits automatisch während der Installation mit installiert, wobei einige Programme möglicherweise nicht benötigt werden und stattdessen andere Programme wünschenswert sind.

Dabei gibt es – je nach betreffender Software – einige Abhängigkeiten zu berücksichtigen, d.h., ein bestimmtes Programm setzt das Vorhandensein eines anderen Programms, einer Bibliothek oder eines bestimmten Projektes voraus, was das Software-Management in früheren Linux-Zeiten doch recht beschwerlich machte, zumal sich der Anwender im Fehlerfall auch noch mit dem Compiler und dem Linker beschäftigen musste.

Diese Zeiten sind erfreulicherweise vorbei, denn alle aktuellen Linux-Distributionen verfügen über einen so genannten *Paketmanager*, der das Herunterladen von Programmen aus dem Internet und deren Installation, nebst Beachtung der jeweiligen Abhängigkeiten, automatisiert, so dass sich das Software-Management als sehr einfach und schnell darstellt, wie es in dieser Form mit Windows nicht denkbar ist.

Der Paketmanager der Debian-Distributionen wird als *Advanced Packaging Tool* (apt) bezeichnet und für die Installation von Software mit *apt-get* gestartet, wie es bereits in den vorherigen Kapiteln bei der Installation von *tree* und dem *Midnight Commander* praktiziert worden ist. Apt-get bedient sich dabei eines vordefinierten Pools oder *Repository*. Der Link dorthin, wobei es mehrere davon – auch lokale – geben kann, wird in einer Liste (sources.list) geführt, die unter *etc/apt* zu finden ist. Neben der Installation von Paketen können mit dem apt-get-Manager auch Pakete gelöscht und aktualisiert werden.

```
pi@raspberrypi ~ $ cat /etc/apt/sources.list
deb http://mirrordirector.raspbian.org/raspbian/ wheezy main contrib non-free rpi
pi@raspberrypi ~ $
```

Abbildung 2.7: Die Sources-Liste besteht nur aus einem Eintrag.

Die Suche nach bestimmten Paketen ist zwar ebenfalls möglich, allerdings muss der jeweilige Name hierfür bekannt sein, was oftmals nicht der Fall ist. Für die Suche wird nicht apt-get, sondern apt-file eingesetzt, welches ebenfalls auf einer Datenbank basiert und zunächst zu installieren ist:

```
pi@raspberrypi ~ $ sudo apt-get install apt-file
```

In der folgenden Tabelle sind die apt-get- und apt-file-Funktionen mit Beispielen als Überblick angegeben.

Tabelle 2.3: Umgang mit Software-(Paketen) mithilfe von apt-get und apt-file

Pakete/Software	**Erläuterung/ Beispiel**
Installieren: **sudo apt-get install mc**	Der Midnight Commander mc wird installiert.
Deinstallieren: **sudo apt-get purge mc**	Der Midnight Commander mc wird deinstalliert.
Softwaredatenbank aktualisieren: **sudo apt-get update**	Paketliste vom zentralen Server herunterladen und lokale apt-get-Liste aktualisieren. Dieser Befehl sollte vor jeder Programminstallation ausgeführt werden.
Programme aktualisieren: **sudo apt-get upgrade**	Aktualisierung der bereits mit apt-get installierten Programme.
Löschen temporärer Dateien: **sudo apt-get autoclean**	Die während eines Aktualisierungsvorgangs angelegten und nicht mehr notwendigen Dateien werden gelöscht.
Löschen überflüssiger Dateien: **sudo apt-get autoremove**	Dateien, die aufgrund geänderter Abhängigkeiten nicht mehr benötigt werden, werden gelöscht.

Pakete/Software	Erläuterung/ Beispiel
Datenbank aktualisieren: **sudo apt-file update**	Paket- und Abhängigkeitsliste für apt-file aktualisieren.
Pakete nach Namen suchen: **sudo apt-file -l search gpio**	Suchen nach Paketen, die gpio betreffen.

Bei der Suche mit apt-file ist zu beachten, dass der Suchbegriff keineswegs in den Namen der gefundenen Dateien vorkommen muss, sondern vielmehr haben diese Dateien »irgendetwas« mit dem Suchbegriff zu tun, der in einer dazugehörigen Bibliothek oder vielleicht auch nur als Aufruf in einer der gefundenen Daten vorkommt. Demnach ist apt-file zwar nützlich, erspart es einem jedoch nicht, sich genauer mit den Programmen und Daten beschäftigen zu müssen.

```
pi@raspberrypi ~ $ sudo apt-file -l search gpio
fso-deviced
libdc1394-22-doc
linux-doc-3.2
linux-headers-3.2.0-4-common
linux-headers-3.2.0-4-rpi
openwince-include
oss4-dkms
python-rpi.gpio
python3-rpi.gpio
raspberrypi-bootloader
smp-utils
pi@raspberrypi ~ $
```

Abbildung 2.8: Die Suche nach GPIO hat mehrere Dateien zutage gefördert.

Grundsätzlich ist es möglich, verschiedene Linux-Programme auf dem Raspberry Pi zu installieren, wobei jedoch die begrenzten Ressourcen des Systems zu beachten sind. Das macht sich bereits beim Browsen im Internet mit dem Standard-Browser Midori bemerkbar, denn der Aufbau der Seiten geht vielfach sehr gemächlich voran, zumal eine ganze Reihe von Seiten nicht korrekt angezeigt werden können und er auch kein HTML5 oder Flash unterstützt. Gebräuchliche Browser wie Chrome oder Firefox benötigen weit mehr Speicher als Midori, so dass es keinen Sinn macht, diese Programme auf dem Raspberry Pi zu installieren, solange es hierfür keine speziell angepassten Versionen gibt.

Die *Raspberry Pi Foundation* hat Ende des Jahres 2012 einen *Pi Store* etabliert, der spezielle Linux-Programme für den Raspberry Pi offeriert. Um Programme hierüber beziehen und installieren zu können, ist ein *Pi Store Client* notwendig, der nach der Installation (sudo apt-get install pistore) als Link auf dem LXDE-Desktop erscheint.

2.6 Firmware

Zur Software ist auch die so genannte *Firmware* zu rechnen, die üblicherweise in einem Chip der Elektronik abgelegt ist, wie es etwa bei einem PC der Fall ist, wo sich die Firmware (BIOS) in einem speziellen Speicherbaustein befindet. Von der ursprünglichen Definition her handelt es sich dabei um eine »feste Software«, die vom Hersteller programmiert und in einem Speicherchip abgelegt worden ist, was in früheren PC-Zeiten standardmäßig der Fall war, denn die Software befand sich in einem ROM (Read Only Memory). Seitdem stattdessen elektrisch löschbarer Speicher (EEPROM, Flash) für das PC-BIOS eingesetzt wird, ist es üblich geworden, dass hin und wieder ein BIOS-Update notwendig wird, um auf diese Art und Weise Herstellerfehler zu beseitigen oder auch neuere Funktionen hinzuzufügen.

Aus Leistungs- und Kostengründen gibt es beim Raspberry Pi weder ein BIOS, welches beim PC neben den grundlegenden Routinen für die Ansteuerung der Hardware auch über eine Setup-Funktion (BIOS Setup) für die Hardware-Konfigurierung verfügt, noch eine Firmware im klassischen Sinne. Vielmehr befinden sich diese grundlegenden Hardware-Routinen in einer Datei auf dem einzigen Festwertspeicher, den es beim Raspberry Pi gibt, auf der SD-Karte.

2.6.1 Bootvorgang – Firmware und Kernel

Der Broadcom SoC (BCM2835) ist derart entwickelt worden, dass er die Firmware direkt von der SD-Karte lesen kann, was in mehreren Stufen erfolgt. Nach dem Einschalten des Raspberry Pi sind die eigentliche ARM-CPU sowie das SDRAM zunächst inaktiv. Die GPU lädt die erste Stufe des Bootloaders vom ROM des SoC. Genau betrachtet, wäre dieser ROM-Code die eigentliche Firmware. Die Software im ROM kann nichts anderes, als auf die SD-Karte zuzugreifen, wo die zweite Stufe des Bootloaders in Form der Datei bootcode.bin (siehe auch Abbildung 2.3) von der FAT-Partition der SD-Karte geladen wird. Diese aktiviert das SDRAM und lädt die dritte Stufe (loader.bin) von der SD-Karte in das RAM, womit der Bootloadervorgang komplett ist, so dass jetzt die Firmware (start.elf) in das RAM geladen werden kann. Hierfür gibt es verschiedene Standardkonfigurationen, die mit den Dateien arm128_start.elf bis arm240_start.elf bestimmt werden und die SDRAM-Aufteilung zwischen CPU und GPU festlegen.

Danach folgen das Laden des Kernel Images (kernel.img) und die Konfigurationsdateien *config.txt* (GPU-Parameter, HDMI, Takt) sowie *cmdline.txt* (Kernel-Parameter, root, console) werden verarbeitet, woraufhin die GPU den ARM-Prozessor im SoC mit einem Offset-Sprung auf die Adresse 0x8000 startet. Dort befindet sich der Linux-Kernel, der nunmehr die weitere Ausführung übernimmt und das System vollständig bootet.

Der Boot-Vorgang

1. Bootloader-Schritt: Laden des SoC-ROM-Codes.
2. Bootloader-Schritt: Zugriff auf die SD-Card und die Datei ***bootcode.bin*** verarbeiten.
3. Bootloader-Schritt: Die Datei ***loader.bin*** in das SDRAM laden.
4. Die Firmware (**start.elf**) wird in das SDRAM geladen.
5. Laden des Kernel-Images (**kernel.img**)
6. Laden der Konfigurationsdateien ***config.txt*** und ***cmdline.txt***.

2.7 Aktualisierung – Updates

Wie das Betriebssystem und die Software unterliegen sowohl die Kernel-Software als auch die Firmware laufenden Änderungen, so dass es sinnvoll sein kann, eine Aktualisierung vorzunehmen. Dabei gilt auch hier, wie generell für Software-Aktualisierungen, dass mit einem Update nicht zwangsläufig merkliche Verbesserungen erzielt werden. Es kann auch passieren, dass zuvor einwandfrei funktionierende Applikationen nicht mehr wie gewohnt agieren, weil sich gewissermaßen das »Fundament« des betreffenden Programms mit dem Update geändert hat. Deshalb ist es immer eine Überlegung wert, ob überhaupt ein Update sinnvoll ist, wenn alles wie gewünscht funktioniert. Üblicherweise findet sich zu jedem Update eine kurze Erläuterung dahingehend, was sich gegenüber der vorherigen Version im Wesentlichen geändert hat. Es schadet nicht, sich hierüber zu informieren, bevor ein Update durchgeführt wird.

Zur Anzeige der momentanen *Kernel-Version* ist die folgende Zeile einzugeben:

```
pi@raspberrypi ~ $ uname -a
```

Und die Firmware-Version wird angezeigt mit:

```
pi@raspberrypi ~ $ /opt/vc/bin/vcgencmd version
```

Mithilfe des Programms *vcgencmd* (in /opt/vc/bin/) lassen sich eine Reihe von weiteren nützlichen Daten anzeigen wie etwa die Taktfrequenz, die Speicheraufteilung zwischen CPU und GPU oder auch die Chiptemperatur (vcgencmd measure_temp). Alle möglichen Befehle werden mit vcgencmd commands dargestellt.

```
pi@raspberrypi ~ $ uname -a
Linux raspberrypi 3.2.27+ #160 PREEMPT Mon Sep 17 23:18:42 BST 2012 armv6l GNU/Linux
pi@raspberrypi ~ $ /opt/vc/bin/vcgencmd version
Sep 18 2012 01:45:27
Copyright (c) 2012 Broadcom
version 337601 (release)
pi@raspberrypi ~ $
```

Abbildung 2.9: Ermittlung der Versionen von Kernel und Firmware, die hier aus dem September 2012 stammen und deshalb aktualisiert werden sollten.

Ein triftiger Grund für eine Aktualisierung ist sicherlich, wenn nur ein Speicher von 256 MByte bei einem Raspberry Pi zur Verfügung stehen, obwohl er tatsächlich über 512 MByte verfügt. Die aktuellen Kernel- und Firmware-Versionen sind im Internet bei GitHub zu finden. Dies ist ein Hosting-Dienst für unterschiedliche Entwicklungsprojekte – wie etwa für Bundestagsgesetze, Mac- und Windows-Entwicklungen und eben auch für den Raspberry Pi –, die alle das Versionsverwaltungssystem Git (engl. Schwachkopf, Blödmann) einsetzen. Git kontrolliert dabei nicht Codes, sondern die Repositories (Aufbewahrungsort) für die Codes.

Eine Möglichkeit der Aktualisierung ist das Kopieren der Dateien (start.elf, kernel.img) direkt mit einem PC auf die SD Card. Dabei ist die separate Beschaffung der Software jedoch etwas unübersichtlich und damit fehleranfällig. Statt die Dateien einzeln zu suchen und zu kopieren, erscheint es einfacher, ein komplett neues Image von der Raspberry Pi-Seite `http://www.raspberrypi.org/downloads` »aufzuspielen«, wobei dann jedoch die bis dato getätigten Änderungen und Anpassungen am System verlorengehen. Damit dies nicht passiert, könnten nur diese beiden Dateien (start.elf, kernel.img) aus einen neu beschafften Image hergenommen und direkt per PC auf die SD Card kopiert werden.

```
pi@raspberrypi ~ $ sudo wget http://goo.gl/1BOfJ -O /usr/bin/rpi-update
--2013-03-28 18:20:23--  http://goo.gl/1BOfJ
Auflösen des Hostnamen »goo.gl (goo.gl)«... 173.194.69.102, 173.194.69.113, 173.
194.69.100, ...
Verbindungsaufbau zu goo.gl (goo.gl)|173.194.69.102|:80... verbunden.
HTTP-Anforderung gesendet, warte auf Antwort... 301 Moved Permanently
Platz: https://raw.github.com/Hexxeh/rpi-update/master/rpi-update[folge]
--2013-03-28 18:20:23--  https://raw.github.com/Hexxeh/rpi-update/master/rpi-upd
ate
Auflösen des Hostnamen »raw.github.com (raw.github.com)«... 199.27.77.196
Verbindungsaufbau zu raw.github.com (raw.github.com)|199.27.77.196|:443... verbu
nden.
HTTP-Anforderung gesendet, warte auf Antwort... 200 OK
Länge: 6198 (6,1K) [text/plain]
In »»/usr/bin/rpi-update«« speichern.

100%[===================================>] 6.198       --.-K/s   in 0,001s

2013-03-28 18:20:29 (4,22 MB/s) - »»/usr/bin/rpi-update«« gespeichert [6198/6198]

pi@raspberrypi ~ $
```

Abbildung 2.10: Download von rpi-update

Es gibt noch eine automatisierte Methode, die jederzeit komfortabel direkt mit dem Raspberry Pi ausgeführt werden kann. Hierfür stellt Github (Abbildung 2.11) das Programm *rpi-update* zur Verfügung, welches die Software des Raspberry Pi entsprechend überprüft und eine automatische Aktualisierung vornehmen kann, was allerdings ein weiteres Paket (git-core) voraussetzt, welches im Bedarfsfall wie folgt zu installieren ist:

```
pi@raspberrypi ~ $ sudo apt-get install git-core
```

Mit der folgenden Zeile wird das rpi-update aus dem Internet geladen und im Verzeichnis /usr/bin gespeichert:

```
pi@raspberrypi ~ $ sudo wget http://goo.gl/1BOfJ -O /usr/bin/rpi-
update
```

Die obige Zeile ist nicht nur etwas länger, sondern muss auch exakt so eingegeben werden. Andernfalls kann es zu schwer lokalisierbaren Fehlern kommen.

Selbstverständlich sind die Groß- und Kleinschreibung zu beachten sowie die Abstände zwischen den Zeichen. Außerdem ist *1BOfJ -O* anzugeben, d.h., hier sind zwei Os und nicht etwa Nullen vorhanden. Nachfolgende Probleme bei der Ausführung des Befehls *rpi-update* liegen meist an unbemerkt falschen Angaben in dieser Zeile, denn auch wenn beispielsweise eine Null bei 1B0fJ angegeben wird, wird ein Paket ohne Fehlermeldung heruntergeladen und kann mit der folgenden Zeile auch als »ausführbar« festgelegt werden. Allerdings wird dann nach dem Aufruf von rpi-update eine Fehlermeldung (Syntax error: newline unexpected) erschienen und kein Update ausgeführt.

```
pi@raspberrypi ~ $ sudo chmod +x /usr/bin/rpi-update
```

Nach diesen Vorarbeiten sollte die Überprüfung der installierten Software sowie der Update-Vorgang stattfinden, was einige Minuten in Anspruch nimmt. Dieser Vorgang wird dann ausgelöst mit:

```
pi@raspberrypi ~ $ sudo rpi-update
```

```
pi@raspberrypi30 ~ $ sudo rpi-update
 *** Raspberry Pi firmware updater by Hexxeh, enhanced by AndrewS and Dom
 *** Performing self-update
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  7635  100  7635    0     0  33537      0 --:--:-- --:--:-- --:--:-- 46554
 *** Relaunching after update
 *** Raspberry Pi firmware updater by Hexxeh, enhanced by AndrewS and Dom
 *** Downloading specific firmware revision (this will take a few minutes)
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   168    0   168    0     0    277      0 --:--:-- --:--:-- --:--:--   375
100 22.9M  100 22.9M    0     0   900k      0  0:00:26  0:00:26 --:--:-- 1213k
 *** Updating firmware
 *** Updating kernel modules
 *** depmod 3.12.30+
 *** Updating VideoCore libraries
 *** Using HardFP libraries
 *** Updating SDK
 *** Running ldconfig
 *** Storing current firmware revision
 *** Deleting downloaded files
 *** Syncing changes to disk
 *** If no errors appeared, your firmware was successfully updated to 11bfdffaca95e630b1fed0a8992fee880
c17c26d
 *** A reboot is needed to activate the new firmware
pi@raspberrypi30 ~ $
```

Abbildung 2.11: Durchführung des Update-Vorgangs

Sicherheitshalber empfiehlt es sich vor Aktualisierungen das Datum kontrollieren bzw. zu setzen (*sudo date MMDDhhmm*), damit die neuesten Versionen übertragen

werden. Nach einem Neuboot sollte das System betreffs Firmware, Kernel und einiger weiterer Bibliotheken (siehe Abbildung 2.11) auf dem neuesten Stand sein. Um eine neue rpi-update-Version zu installieren, reicht die folgende Zeile aus.

```
pi@raspberrypi ~ $  sudo install rpi-update
```

Die eigentliche Software, wie das Betriebssystem und die installierten Pakete, ist damit jedoch nicht aktualisiert worden, wofür weitere Schritte mit apt-get notwendig sind (siehe Tabelle 2.3). Mit *sudo apt-get update* wird die Paketliste aktualisiert und nachfolgend wird die Aktualisierung der notwendigen Software mit sudo apt-get upgrade ausgelöst. Dabei kann der optionale Parameter -y mit angegeben werden, der dafür sorgt, dass Nachfragen während des Aktualisierungsvorgangs automatisch mit Ja beantwortet werden Eine kompletter Aktualisierungsvorgang kann damit wie folgt aussehen:

`date`	Kontrolle des Datums und der Uhrzeit
`sudo date MMDDhhmm`	Ggf. einstellen des Datums und der Uhrzeit
`sudo apt-get update`	Aktualisieren der Paketliste
`sudo apt-get upgrade`	Aktualisieren der Software
`sudo rpi-update`	Firmware und Kernel aktualisieren

3 Hardware

Das »Herzstück« der Raspberry Pi-Schaltung bildet der Prozessor BCM2835 der Firma Broadcom. Dies ist ein bekannter amerikanischer Hersteller von integrierten Kommunikationsschaltungen, die für xDSL, Cable Modems und drahtlose Netze (WLAN, Bluetooth 3G, LTE) eingesetzt werden, sowie von speziellen Multimediaprozessoren, die beispielsweise in Tablet-PCs und Smartphones zu finden sind.

3.1 ARM-Prozessor BCM2835

Der BCM2835 ist speziell in Zusammenarbeit mit der *Raspberry Pi Foundation* entwickelt worden und nicht einzeln erhältlich; er kommt allein beim Raspberry Pi zum Einsatz und nimmt im Produktspektrum der Firma Broadcom eine Sonderrolle ein. Gefertigt wird der BCM2835 von den Firmen Hynix und Samsung, zwei der größten Chiphersteller weltweit. Samsung fertigt Chips für zahlreiche verschiedene Firmen, wie beispielsweise auch die ARM-Chips für Apple (iPhone, iPad). Die Firma Hynix ist insbesondere als Hersteller für Speicherbauelemente (SDRAM, Flash) bekannt. Bei den ersten Raspberry Pi-Exemplaren ist auf dem BCM2835 eine Hynix-Beschriftung zu erkennen, weil der SDRAM-Speicher quasi als obere Schicht (Chip Package) des Bausteins platziert ist. Bei allen aktuellen BCM2835-Exemplaren ist hingegen eine Samsung-Beschriftung vorhanden.

Abbildung 3.1: Der BCM2835 ist als solcher nicht zu identifizieren, denn er befindet sich unter dem Speicher der Firma Samsung.

In der Abbildung 3.1 ist zu sehen, dass der BCM2835 keine erkennbaren Anschlüsse (Pins) führt, denn die einzelnen Kontakte (289 Stück) sind auf der Chipunterseite in Form von Zinnkugeln (Balls) ausgeführt. Durch das Erhitzen der Platine während der Fertigung schmelzen die Kugeln auf, so dass hiermit feste Kontakte zu den entsprechenden Punkten auf der Platine hergestellt werden. Diese Gehäuse-

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1_4, © Springer Fachmedien Wiesbaden 2015

form wird gemeinhin als BGA (Ball Grid Array) bezeichnet, wovon es zahlreiche verschiedene Varianten in verschiedene Größen mit unterschiedlicher Kontaktanzahl gibt. Der BCM2835 wird, wie andere ARM-Chips auch, in einem so genannten *Thin profile Fine Pitch Ball Grid Array* (TFBGA) hergestellt.

Abbildung 3.2: ARM-Prozessoren werden oftmals in BGA-Gehäusen gefertigt.

Neben dem eigentlichen Rechenkern (Core CPU ARM1176JZF-S) enthält der BCM2835 zahlreiche weitere Komponenten, wie etwa den Speicher (SDRAM) und die Grafikeinheit (GPU) für die Ansteuerung eines Monitors, so dass ein derartiger Chip häufig auch als *Multimedia SoC* (System on Chip) bezeichnet wird.

3.2 ARM-Architektur

Die eigentliche CPU (Central Processing Unit) wird von einem ARM11 gebildet, der mit einer Taktfrequenz von 700 MHz arbeitet. Dabei handelt es sich um eine ältere Architektur (ARMv6 aus dem Jahre 2002), was bedeutet, dass aktuelle Smartphones mit einer ARM-Cortex-Architektur demgegenüber weitaus leistungsfähiger sind. Unter der Bezeichnung ARM-Architektur gibt es zahlreiche verschiedene Mikroprozessoren und Mikrocontroller, die von unterschiedlichen Firmen wie Atmel, NXP, Texas Instruments (OMAP), Toshiba, STMicroelectronics oder auch Silicon Labs angeboten werden. Die Basis bildet dabei ein Prozessorkern, der ursprünglich von den Firma Acorn in England unter Federführung der Universität in Cambridge entwickelt wurde.

Interessanterweise ist hier über 30 Jahre später auch der *Raspberry Pi* entstanden (teilweise waren sogar die gleichen Personen daran beteiligt), denn die Intention ist bei beiden Projekten die gleiche: Eine kostengünstige und leicht zugängliche Plattform zu schaffen, die das Programmieren und Experimentieren ermöglicht,

ohne dass hierfür besondere Kenntnisse in der Informatik oder Elektronik notwendig sind, sich also auch an Einsteiger richtet.

Die Firma Acorn war Anfang bis Mitte der 80er Jahren sehr erfolgreich mit dem Computer *BBC Micro*, der als Homecomputer und für den Unterricht an englischen Schulen verwendet wurde und mit einem 6502-Mikroprozessor arbeitete. Für das Nachfolgemodell wurde ein eigener 32-Bit-Prozessor in RISC-Architektur mit der Bezeichnung *Acorn RISC Machine* – kurz ARM – entwickelt. Der erste ARM-Prozessor kam daraufhin im Modell *Acorn Archimedes* zum Einsatz. Er war letztendlich erfolgreicher als der damit betriebene Computer, weil andere Firmen Interesse an der Verwendung dieses Prozessors hatten.

Im Jahre 1990 gründete Acorn zusammen mit Apple und VLSI Technology, die auch die Prozessoren für Acorn fertigte, die Firma *Advanced Risc Machines* (ARM) mit Sitz in Cambridge. Seitdem entwickelt die Firma ARM die spezielle RISC-Architektur konsequent weiter, fertigt allerdings selbst keine Prozessoren, sondern verkauft für die ARM-Rechenkerne (IP Cores) und dazu passende Peripherie-Chips (Coprozessor, Speicher, Multimedia) entsprechende Lizenzen. Fast alle Firmen, die im Prozessorgeschäft tätig sind, sind auch ARM-Lizenznehmer; beispielsweise gehören Unternehmen wie Apple, Intel und Microsoft ebenfalls dazu.

Die Hersteller erwerben bei ARM die benötigten Cores mit den entsprechenden Tools, die auch mit IP-Cores anderer Firmen kombiniert werden können, und erzeugen durch den Vorgang der *Synthese* den eigenen ARM-basierten Controller. Dabei können verschiedene Parameter und damit Einheiten wie Caches oder Busbreiten oder auch Eigenschaften wie *Perfomance versus Leistungsaufnahme* oder die benötigte Chipfläche vom Chiphersteller variiert werden, was letztendlich zu einer umfassenden Einchip-Lösung führt. Die Firma Broadcom ist beim BCM2835 nach diesem Prinzip vorgegangen und hat neben anderen Einheiten ihren speziellen VideoCore hier mit integriert.

Ganz allgemein beruht der ARM-Erfolg in erster Linie auf der »schlanken« 32-Bit-Architektur, dem sehr guten Preis-/Leistungsverhältnis, dem effizienten Befehlssatz sowie der Tatsache, dass die Stromaufnahme gut mit der jeweiligen Taktfrequenz skaliert, was zu sehr energieeffizienten Prozessoren führt, die in zahlreichen Geräten wie Routern, Druckern, Digitalkameras oder Smartphones unter Linux, Windows CE und Android arbeiten. In den Geräten der Firma Apple wie iPhone, iPod und iPad sind ebenfalls ARM-Prozessoren eingebaut.

3.2.1 Cores und Typen

Einige der Lizenznehmer wie Freescale, NXP und Intel dürfen auch Veränderungen am ARM-Kern selbst vornehmen, was dann zu mehr oder weniger eigenen Entwicklungen wie den XScale-Prozessoren bei Intel führte. Deshalb liegt es nahe, neben den ARM-Prozessoren auch spezielle Ausführungen in Form von Mikrocontrollern oder *Application Processors* herzustellen, die bei ARM unter der Bezeichnung CORTEX geführt werden.

Tabelle 3.1: ARM-Familien und Typen im Überblick

ARM-Familie	Architektur	Cores	Merkmale
ARM1	ARMv1	ARM1	Der erste ARM-Prozessor, 4 MHz, 26-Bit Addressing
ARM2	ARMv2 ARMv2A	ARM2 ARM250	8 - 12 MHz, 32 Bit-Multiplizierer, Unterstützung für Coprozessor
ARM3	ARMv2A	ARM2a	25 MHz, 32-Bit Addressing, 4 kB-Cache
ARM6	ARMv3	ARM60 ARM600 ARM610	12 - 33 MHz, MMU-Unterstützung, Virtual Memory
ARM7	ARMv3	ARM700 ARM710 ARM710a ARM7100 ARM7500 ARM7500FE	18 - 56 MHz, 8 kB-Cache
ARM7TDMI	ARMv4T	ARM7TDMI ARM7TDMI-S ARM710T ARM720T ARM740T	18 - 60 MHz, 3-stufige Pipeline, Thumb-Befehlssatz
	ARMv5TEJ	ARM7EJ-S	5-stufige Pipeline
ARM8	ARMv4	ARM810	70 MHz

ARM-Familie	Architektur	Cores	Merkmale
ARM9TDMI	ARMv4T	ARM9TDMI ARM920T ARM922T ARM940T	180 MHz, 16 kB-Cache
	ARMv5TE	ARM946E-S ARM966E-S ARM968E-S	Kein Cache, DSP-Befehle
ARM9E	ARMv5TEJ	ARM926EJ-S	Jazelle-Befehlssatz
ARM10E	ARMv5TE	ARM1020E ARM1022E	6-stufige Pipeline, 32 kB-Cache, erweiterte DSP-Befehle
	ARMv5TEJ	ARM1026EJ-S	Thumb- und Jazelle-Befehlssatz
ARM11	ARMv6	ARM1136J-S ARM1136JF-S	400 - 660 MHz, 8-stufige Pipeline
	ARMv6T2	ARM1156T2-S ARM1156T2F-S	9-stufige Pipeline, SIMD
	ARMv6K	ARM11MP Core	Multiprozessor-Core
	ARMv6K	ARM1176JZF-S	700 - 1000 MHz, Floating Point Unit, Trustzone-Technologie
Cortex	ARMv7-A	Cortex-A5 Cortex-A8 Cortex-A9	600 - 1000 MHz, 1 - 4 Cores, SMP, 8-stufige Pipeline Thump-2-, Jazelle RCT-Befehlssatz
	ARMv7-R	Cortex-R4	500 MHz, Embedded Profile
	ARMv7-M	Cortex-M0	Nur Thumb-2-Befehlssatz, kein Cache, für 8/16-Bit-Applikationen
		Cortex-M3	Für 16/32-Bit-Applikationen
	ARMv6-M1	Cortex-M1	Für FPGA-Implementierung
	ARMv7-ME	Cortex-M4	Mikrocontroller Profile, für 32-Bit-Applikationen
	ARMv8	Cortex-A5x	64-Bit-Architektur, max. 3 GHz

Die Orientierung fällt bei den ARM-Prozessoren nicht leicht, weil zwischen der jeweiligen ARM-Familie, der dabei zugrunde gelegten Architektur sowie der Core-Implementierung unterschieden wird und der tatsächliche Chip, wie eingangs

erwähnt, von den verschiedenen Firmen mit unterschiedlichen Ausstattungsmerkmalen und Bezeichnungen angeboten wird. Die Tabelle 3.1 ist deshalb zur Orientierung angegeben.

Während die ersten Architekturen (ARM1-, ARM2-, ARM3-Familien) lediglich in den Archimedes-Computern der Firma Acorn zum Einsatz kamen, änderte sich das Geschäftskonzept mit der Umbenennung in *Advanced Risc Machines* (s.o.) grundlegend. Das erste Ergebnis war ein Prozessor (ARM610) auf der Basis des ARM6-Core, der 1993 im ersten bekannten PDA von Apple – Newton – eingesetzt wurde.

Der große Erfolg stellte sich jedoch erst zwei Jahre später mit den Nachfolgern – der ARM7-Familie – ein, die aktuell das untere Ende der ARM-Architektur markieren und nicht mehr für Neuentwicklungen empfohlen werden. Insbesondere die ARM9- als auch die ARM11-Cores sind aktuell in einer Vielzahl von Chips und damit Geräten enthalten. Beim Einsatz dieser etablierten CPU-Kerne kann der Anwender auf ein breites Spektrum an Debug- und Programmierumgebungen zurückgreifen. Es können bereits vorhandene Bibliotheken und Beispielprogramme genutzt werden.

Die Cortex-A-Cores werden als *Application Processors* bezeichnet. Sie können mit bis zu vier Cores ausgestattet sein und sind explizit für Einheiten wie Smartphones, Notebooks und Digital-TV-Geräte vorgesehen. Cortex-R- und Cortex M-Cores werden bei ARM als *Embedded Processors* geführt und avisieren Echtzeitanwendungen (Real Time), wie sie bei Netzwerk- und Automotive-Applikationen sowie in der Medizintechnik üblich sind.

Das *Embedded Profile* (Cortex-R) unterscheidet sich jedoch maßgeblich vom *Microcontroller Profile*, welches bei den Cortex-M-Cores die Basis bildet. Cortex-M ist explizit auf niedrigere Taktraten und einen effizienten Stromverbrauch bei geringen Kosten ausgelegt. Die am häufigsten verwendete Cortex-M-Architektur ist momentan Cortex-M3, die typischerweise mit 120 MHz getaktet wird und in Mikrocontrollern von Silicon Labs (EFM32, Gecko), NXP (LPC1700) oder auch STMicroelectronics (STM32-F2) zum Einsatz kommt. Die Stromaufnahme eines STM32-F2 liegt bei ca. 190 µA/MHz, was dem Fertigungsprozess (90 nm) und der niedrigen Versorgungsspannung von 1,2 V zu verdanken ist. Ein Cortex-M4 bietet gegenüber einem Cortex-M3 als wesentlichen Unterschied eine DSP-Funktionalität und wird auch mit einem Cortex-M0-Kern zu einem Dual-Core kombiniert, wie beim LPC4300 der Firma NXP.

Abbildung 3.3: Einordnung der ARM-Prozessoren im Leistungsschema

Das andere Ende der Leistungsscala markiert der Cortex-M0 (Abbildung 3.4), der in Controllern von Silicon Labs (Zero Gecko), STMicroelectronics (STM32-F0) oder auch in den Vertretern der NXP-Serien LPC 1100 und LPC 1200 eingebaut ist. Der Cortex-M0 ist zum Cortex-M3 binärkompatibel und verarbeitet eine Untermenge des Thumb-Befehlssatzes. Cortex-M0-Typen sind momentan die kleinsten Cortex-Controller mit einer Fläche von gerade einmal 2 x 2 mm und enthalten einen 8 kByte RAM- sowie einen 32 kByte-Flash-Speicher. Seine Rechenleistung ist gegenüber einem Cortex-M3 zwar geringer, dafür aber auch der Stromverbrauch, der demgegenüber nur noch halb so groß ist (ca. 85 µW/MHz).

Der im BCM2835 eingesetzte ARM-Core (ARM1176JZF-S) entspricht einem *Classic ARM Processor*, der erst durch das Hinzufügen von verschiedenen Einheiten (Video Core, SDRAM, GPIO) zum leistungsfähigen und universell einsetzbaren Raspberry Pi-Prozessor wird. Der als *Trust Zone* spezifizierte Kern (Abbildung 3.5) ist eine Ergänzung zur ARMv6-Architektur. Er erlaubt die separate Ausführung von »sicheren« und »unsicheren« Codes, was anhand eines *Secure Monitors* gesteuert wird und die Ausführung eines sicher zu betreibenden Betriebssystem-Kerns (privilegierter Mode) von der Ausführung von (möglicherweise) unsicherem Anwendungscode separieren kann. Dies ist als eine wichtige Funktion für ein sicheres und stabiles System zu werten.

Abbildung 3.4: Der Cortex-M0-Controller von NXP ist einer der kleinsten Mikrocontroller mit ARM-Architektur.

Dem Kern stehen zwei separate Cache-Speicher für Befehle und Daten zur Seite (Instruction Cache, Data Cache), wie es auch aus der x86-Architektur von Intel her bekannt ist. Über die Coprozessor-Schnittstelle lassen sich andere Prozessoren, wie etwa ein Grafikcontroller, schaltungstechnisch anbinden. Das für ARM-Prozessoren obligatorische *Debug Interface* ist für die Programmierung und das Debuggen von Programmcode (Firmware) vorgesehen.

AMBA gibt es mittlerweile in verschiedenen Versionen und ist bereits vom ARM7 her bekannt. Es stellt ein Protokoll für die On-Chip-Verbindungen und das Management der einzelnen Funktionsblöcke in einem SoC (System on Chip) dar, die über verschiedene Interfaces, wie etwa per AXI beim ARM1176, angebunden werden. Der ARM1176 verfügt insgesamt über drei AXI-Interfaces, so dass hier prinzipiell eine Vielzahl von Funktionsblöcken angeschlossen werden können.

Die als optional angegebene *Floating Point Unit* (VFP: Vector Floating Point) ist beim BCM2835 vorhanden und wird standardmäßig vom Raspbian-Betriebssystem – nicht jedoch zwangsläufig auch von anderen Systemen (siehe Kapitel 4.1) – unterstützt.

Abbildung 3.5: Das Blockschaltbild des ARM1176JZF-S

3.3 Speichereinheiten

Der interne SDRAM-Speicher des BCM2835 verfügt über eine Kapazität von 512 MByte, wobei die erste Raspberry Pi-Serie entsprechend des Modells B und das Modell A lediglich 256 MByte besitzen. Aufgrund der Integration des SDRAMs im BCM2835 – und weil keine zusätzlichen externen Speicherchips angedacht sind – gibt es extern auch keine Datenverbindungen zwischen dem Rechenkern und dem Speicher. Gleichwohl sind für den SDRAM-Betrieb einige Anschlüsse des BCM2835 wichtig, die explizit mit der Spannungsversorgung (1,8 V) und dem Massepotential verbunden sein müssen und Kondensatoren für das Abblocken von Störungen auf den Leitungen benötigen.

Abbildung 3.6: Die SDRAM-Sektion des BCM2835

3.3.1 SD Cards

Das Betriebssystem wird von einer SD-Karte (Secure Digital) geladen, die gleichermaßen als Daten- und als Programmspeicher (Festwertspeicher) fungiert. Diese muss bestimmte Voraussetzungen erfüllen, damit diese mit dem Raspberry Pi ohne Probleme eingesetzt werden kann. Deshalb werden im Internet verschiedene Kompatibilitätslisten für passende SD Cards geführt, was jedoch nur bedingt aussagekräftig ist, denn alle möglicherweise mit dem Raspberry Pi funktionierenden Typen können hier natürlich nicht verzeichnet sein.

Außerdem sind in der Liste angeführte Typen auch keine Garantie dafür, dass sich nicht dennoch Schwierigkeiten ergeben, was durchaus an einem unterschiedlichen internen Aufbau der Cards liegen kann, den der Kartenhersteller nicht mit einer (genaueren) Typenbezeichnung dokumentiert.

Wie im Kapitel 1 erläutert, empfiehlt es sich – zumindest für den Einsteiger – mit einer vorgefertigten SD Card zu beginnen, die bereits vom Hersteller mit dem Betriebssystem versehen wurde, um die Sache nicht unnötig zu verkomplizieren. Es spricht jedoch nichts dagegen, später andere SD-Karten aus dem eigenen Fundus auszuprobieren. Ein Schaden an der Elektronik kann durch den Karten-Einsatz nicht hervorgerufen werden, solange die unbeschädigte Karte richtig und ohne Gewalt in die Fassung auf der Platine eingesetzt werden kann.

Grundsätzlich entsprechen SD Cards drei Standards, die sich im Wesentlichen in ihrer maximalen Kapazität voneinander unterscheiden. Allerdings wird diesem Umstand im allgemeinen Sprachgebrauch meist nicht Rechnung getragen: Es wird allgemein von SD Cards gesprochen. Alle drei Typen besitzen die gleichen mechanischen Abmessungen (32 mm x 24 mm x 2,1 mm).

- SD Card (SD 1.0): Kapazitäten von 8 MByte bis 2 GByte
- SDHC Card (SD 2.0): High Capacity, Kapazitäten von 4 GByte bis 32 GByte
- SDXC Card (SD 3.0): eXtended Capacity, Kapazitäten von 64 GByte bis 2 TByte (momentan maximal 128 MByte)

Die SD-Karten verfügen an der Seite über einen kleiner Schieber (Lock), der für das Einschalten eines Schreibschutzes zuständig ist. Dieser ist dann ausgeschaltet, wenn der Schieber in die Richtung der Kartenkontakte weist. Prinzipiell kann die Schieberstellung mithilfe eines zusätzlichen Schalters im Card-Slot ermittelt werden, so dass die Software darauf reagieren kann.

Beim Raspberry Pi wird der Schreibschutz im Übrigen nicht ausgewertet, so dass auch dann auf die Karte geschrieben werden kann, wenn sich der Schieber in der Schreibschutzposition befindet.

Abbildung 3.7: Micro SD Card mit SD-Adapter

Neben der klassischen Bauform, die in den entsprechenden Sockel auf dem Raspberry Pi-Board passt, gibt es noch Typen gemäß *miniSD* (20 x 21,5 x 1,4 mm) und *Micro SD*, (11 mm x 15 mm x 1,0 mm) wofür entsprechende Adapter erhältlich sind, damit diese kleineren Typen ebenfalls mechanisch in einen üblichen SD-Slot passen. Das Raspberry Pi-Model B+ verfügt über einen Micro SD-Kartenslot, so dass hier ausschließlich dieser miniaturisierte SD-Kartentyp eingesetzt werden kann.

Abbildung 3.8: Kontaktbelegung und interner Aufbau einer SD Card

SDHX- und SDXC-Karten funktionieren nicht in Geräten, die lediglich für *SD Cards* spezifiziert sind. Außerdem ist eine geräteabhängige Begrenzung für die

maximal zu nutzende Kapazität gegeben, wobei es hier durchaus Unterschiede für das Lesen und das Schreiben geben kann.

Mit der SDHC-Spezifikation sind verschiedene Leistungsklassen definiert worden, die Mindestübertragungsraten für die Aufzeichnung von MPEG-Datenströmen – gewissermaßen eine minimale Schreibgeschwindigkeit – definieren, was insbesondere für Camcorder und Digitalkameras von Bedeutung ist. Die jeweilige *Class*, die gleichermaßen für SDXC-Karten gilt, ist üblicherweise auf der Karte mit angegeben, oder die Schreibdatenrate wird direkt ausgewiesen.

- Class 2: 2 MByte/s Mindestschreibdatenrate
- Class 4: 4 MByte/s Mindestschreibdatenrate
- Class 6: 6 MByte/s Mindestschreibdatenrate
- Class 10: 10 MByte/s Mindestschreibdatenrate

Der Raspberry Pi kann maximal eine SDHC-Karte einsetzen, so dass der maximale Speicher 32 GByte betragen kann. Mit den Klassen 2 und 4 gibt es dabei eher selten Probleme, während es mit den höherklassigen durchaus zu Kompatibilitätsproblemen kommen kann. Diese sind jedoch für den Einsatz des Raspberry Pi als Mediacenter (XBMC, Kapitel 4.5) zu bevorzugen, weil mit ihnen eine ruckelfreie Videowiedergabe möglich ist.

Der eigentliche Card-Controller befindet sich in der jeweiligen SD-Karte selbst, so dass die hierfür notwendige Elektronik recht einfach ist, wie es der Abbildung 3.9 entnommen werden kann. Die Spannungsversorgung (Vdd) beträgt 3,3 V. Es sind zwei Masseleitungen (Vss1, Vss2) sowie die Masse des Blechslots selbst (MTG1, MTG2) vorhanden.

Die Daten werden über die Leitungen DAT0, DAT1, DAT2 und DAT 3 transportiert, wobei das DAT3-Signal auch als Kartenerkennung (CD: Card Detect) fungieren kann. Die Datenübertragung wird vom BCM2835 taktgesteuert (CLK). Mithilfe des CMD-Signals wird die Transferrichtung bestimmt.

Abbildung 3.9: Das SD-Karteninterface

3.4 Grafikeinheit

Die Grafikeinheit des BCM2835 mit der genaueren Bezeichnung *VideoCore IV* kann sowohl Signale für HDMI (High Definition Multimedia Interface) als auch für *Composite Video* ausgeben, wofür auf der Platine zwei entsprechende Buchsen vorhanden sind. *Composite Video* ist lediglich bei älteren PC-Monitoren zu finden. Es ist jedoch damit möglich, einen Fernseher anzusteuern, wenn das Signal über einen Adapter auf einen SCART-Anschluss des Fernsehgerätes geführt werden kann. Mitunter ist auch ein direkter, passender Video-In-Eingang am Gerät vorhanden. Die Audioverbindung ist separat herzustellen, wofür üblicherweise Cinch-Verbindungen (RCA Jack) eingesetzt werden.

Abbildung 3.10: Die Architektur der Graphics Processing Unit

Als Grafikspeicher wird automatisch ein Teil des BCM2835-Arbeitsspeichers eingesetzt. Die Grafikeinheit (GPU) ist in der Lage, Videos bis hin zu Blu-Ray-Qualität mit einer Datenkompression entsprechend H.264 mit 40 MBit/s über HDMI auszugeben und unterstützt die Standards *OpenGL-ES2.0* für 3D-Funktionen, *OpenVG* für die Verarbeitung von Vektorgrafiken sowie *OpenMAX* für den Zugriff auf Multimedia-CODECs, was in der Summe eine verhältnismäßig gute Grafik-Leistung

und 3D-Darstellung erlaubt. Die jeweiligen Applikationen, die, wie der Kernel Driver, unter Open Source firmieren, setzen auf diese drei Grafikstandards auf. Zur unteren Schicht hin (siehe Abbildung 3.10) fungiert EGL als *Native Graphic Plattform Interface*.

Im Gegensatz zu den CPU-Interna sind genaue Daten der GPU zunächst nicht veröffentlicht worden, was zu massiver Kritik in der *Open Source Community* geführt hat. Broadcom reagierte Anfang des Jahres 2014 darauf und gab den Driver Stack (mit OpenGL ES 1.1/2.0 und EGL) plus einer Dokumentation dazu frei, so dass jetzt auch eigene Grafik- und Video-Entwicklungen für unterschiedliche Systeme mit dieser speziellen GPU möglich sind.

3.4.1 HDMI und DVI

Grundsätzlich führt eine HDMI-Verbindung sowohl Video- als auch Audiosignale und ist kompatibel zur DVI-Schnittstelle, so dass im Bedarfsfall auch einfache Adapter, etwa für die Umsetzung von HMDI auf den DVI-Anschluss eines Monitors, eingesetzt werden können. Das *Digital Visual Interface* (DVI) gibt es in zwei unterschiedlichen Ausführungen: Einmal mit 24 Kontakten und einmal mit 29 Kontakten, wobei die letzte Ausführung auch analoge VGA-Signale führt, so dass mit einem geeigneten Kabel auch weiterhin ein Röhrenmonitor betrieben werden kann.

Abbildung 3.11: Die beiden gebräuchlichen DVI-Anschlüsse

Die digitale HDMI-Datenübertragung beruht auf mehreren TDMS-Kanälen (Transition Minimized Differential Signaling Protocol), wozu jeweils zwei Leitungen gehören auf denen eine *differentielle Datenübertragung* stattfindet. Es findet also keine Datenübertragung in Bezug auf die Schaltungsmasse statt, so dass auch längere Kabelverbindungen von 10 m und mehr kein Problem darstellen.

Als Digitalverbindung zwischen dem Grafikchip und einem digitalem Monitor kommen spezielle Panel-Link-ICs zum Einsatz. Der Sender erzeugt aus den 24-Bit-Signalen drei serielle Signale, die differentiell (+/- 0,5 V) auf je zwei Leitungen (RGB) übertragen werden. Zwei weitere Leitungen transportieren das Taktsignal.

Im Monitor befindet sich ein Panel-Link-Empfänger, der wieder die ursprünglichen Video- und Steuersignale generiert.

Abbildung 3.12: Das Prinzip der Datenübertragung mit TDMS

Der BCM2835 beinhaltet ebenfalls eine derartige Panel-Link-Einheit. An der 19-poligen HDMI-Buchse (S3) des Raspberry Pi-Boards sind die hierfür notwendigen Signale entsprechend verschaltet, wobei verschiedene Dioden als Schutzschaltung für die einzelnen Signale eingesetzt werden, wie es der Abbildung 3.14 entnommen werden kann.

Abbildung 3.13: Die Kontakte der HDMI-Buchse

Neben den TDMS-Signalen ist ein CEC-Signal (Consumer Electronics Control) vorhanden. Es entspricht einem einadrigen Datenbus für die Vernetzung unterschiedlicher Multimediageräte (HiFi, Video, Player etc.), deren Funktionen damit zentral gesteuert werden können.

Außerdem wird der *Display Data Channel* (DDC) unterstützt, der einen Kommunikationsweg zwischen einem Grafikadapter und einem Monitor bildet, um somit eine automatische Konfiguration (Plug & Play) des Grafiksystems zu ermöglichen.

Tabelle 3.2: Die einzelnen HDMI-Signale

Pin Nr.	Signal	Raspberry Pi-Schaltung
1	TDMS Daten 2+	HDMI_TX2_P
2	TDMS Daten 2 Abschirmung	Ground
3	TDMS Daten 2-	HDMI_TX2_N
4	TDMS Daten 1+	HDMI_TX1_P
5	TDMS Daten 1 Abschirmung	Ground
6	TDMS Daten 1-	HDMI_TX1_N
7	TDMS Daten 0+	HDMI_TX0_P
8	TDMS Daten 0 Abschirmung	Ground
9	TDMS Daten 0-	HDMI_TX0_N
10	TDMS Takt +	HDMI_CLK_P
11	TDMS Takt Abschirmung	Ground
12	TDMS Takt -	HDMI_CLK_N
13	Consumer Electronics Control	HDMI_CEC_DAT
14	Reserved	Nicht angeschlossen
15	Serial Clock Line	HDMI_SCL
16	Serial Data Line	HDMI_SDA
17	Ground (DDC, CEC)	Ground
18	+ 5 V Power	+ 5V0
19	Hot Plug Detect	HDMI_HPD

DDC arbeitet in zwei Richtungen (bidirektional) und verwendet hierfür einen speziellen Bus, der im Prinzip dem I²C-Bus der Firma Philips entspricht. Der I²C-Bus (siehe auch Kapitel 6.4) ist in fast jedem CD-Player oder auch Fernseher zu finden und wird für die interne Kommunikation der einzelnen Schaltungseinheiten verwendet. Er besteht aus dem Datensignal *Serial Data Line* (SDA) und dem Taktsignal *Serial Clock Line* (SCL). Für die Detektierung eines eingesteckten HDMI-Kabels, woraufhin CEC und DCC automatisch starten können, wird eine separate Hot Plug Detect-Schaltung (HDMI_HPD) eingesetzt.

Abbildung 3.14: Die HDMI-Schaltung

3.4.2 Composite Video

Im Gegensatz zur HDMI-Schaltung (Abbildung 3.14) stellt sich die Composite Video-Schaltung als vergleichsweise einfach dar, denn hierfür wird ein BCM2835-interner *Digital Analog Converter* (DAC) verwendet, der als externe Beschaltung neben der gelben Cinch-Buchse (S4) lediglich einen Widerstand und zwei Schutzdioden benötigt.

Abbildung 3.15: Beim Modell B+ führt die Audio-Buchse auch das Composite Video-Signal.

Auf der Platine des Raspberry Pi-Model B+ existiert diese separate gelbe Buchse nicht mehr. Das Videosignal ist dort zusammen mit den Audiosignal an eine vierpolige Klinkenbuchse geführt. Die Video- (Abbildung 3.16) und auch die Audioschaltung (Abbildung 3.17) selbst sind jedoch prinzipiell gleich geblieben.

Abbildung 3.16: Die Video Out-Schaltung

3.5 Audio

Für eine Audioverbindung (Audio Out) vom Raspberry Pi mit aktiven Lautsprechern oder Audio-Verstärkern ist auf der Platine eine blaue Buchse (S6, geändert mit dem Modell B+, siehe oben) vorhanden, die einen 3,5 mm Stereo-Klinkenstecker aufnehmen kann. Demnach ist beim Einsatz eines einfachen Audiokabels, welches an beiden Enden mit einem Klinkenstecker versehen ist (wie es zu Monitoren mit integrieren Lautsprechen meist mitgeliefert wird), ein Adapter notwendig, der das Audiosignal auf die bei HiFi- und TV-Geräten üblichen Cinch-Anschlüsse umsetzt. Audioeingänge, um etwa Signale aufzunehmen (Samplen), gibt es beim Raspberry Pi standardmäßig nicht.

Die Audioschaltung auf der Raspberry Pi-Platine wird von zwei PWM-Signalen (Pulse Width Modulation) des BCM2835 gespeist, die aus einer einfachen Filterschaltung mit vorwiegender Tiefpasscharakteristik sowie zwei Dioden als Schutzschaltung (Amplitudenbegrenzung) besteht. Die Schutzschaltung entspricht prinzipiell der, wie sie beim Composite Video-Ausgang vorhanden ist.

Abbildung 3.17: Die Audio Out-Schaltung

3.6 Ethernet und USB

Als zweiter Chip ist der Typ LAN9512 der Firma SMSC auf der Platine vorhanden, der einen USB-Hub sowie einen Ethernet-Controller beinhaltet. Beim Modell A der Raspberry Pi-Platine wird der LAN9512-Chip (Ethernet, USB) nicht bestückt, so dass nur noch ein einziger USB-Port, der direkt vom BCM2835 zur Verfügung ge-

stellt wird, vorhanden ist. Es gibt hier auch keinen Netzwerkanschluss (LAN). Außerdem wird beim Modell A ein BCM2835 eingesetzt, der statt 512 MByte lediglich über 256 MByte an Speicher verfügt.

Beide Maßnahmen schlagen sich in einem günstigeren Preis für dieses Raspberry Pi-Board nieder sowie in einem niedrigeren Stromverbrauch. Dieser hängt maßgeblich davon ab, welche Aktionen gerade ausgeführt werden. Bei der Wiedergabe eines hochauflösenden Videos wird er relativ hoch sein, im nicht aktiven Modus (Sleep) relativ niedrig, so dass hierfür kein allgemein gültiger Wert angegeben werden kann, zumal die eventuell angeschlossene Peripherie wie USB-Geräte oder Leuchtdioden ebenfalls zum Stromverbrauch beitragen. Zur Orientierung beträgt der Stromverbrauch laut Datenblatt beim (üblichen) Modell B typischerweise 3,5 W und beim Modell A lediglich 2,5 W.

Dementsprechend ist das Modell A eher für Applikationen interessant, die aus einer Batterie gespeist werden sollen und bei denen die genannten Einschränkungen für die Anwendung keine ausschlaggebende Rolle spielen, also eine langsamere Datenverarbeitung aufgrund des kleineren Speichers zulässig und keine Ethernet-Netzwerkverbindung notwendig ist.

Embedded Systems, die explizit für eine bestimmte Aufgabe programmiert sind und häufig in der Mess- und Steuertechnik eingesetzt werden, sind hierfür typische Beispiele oder auch autarke, drahtlose Sensorknoten, die etwa Umweltdaten messen, diese lokal speichern und zu bestimmten Zeiten über eine drahtlose Netzwerkverbindung (WLAN, Bluetooth) an eine Basisstation übertragen.

Abbildung 3.18: Beim Modell A ist der LAN9512-Chip nicht bestückt, so dass auch die Buchsen für den Ethernet-Anschluss und den zweiten USB-Port fehlen.

3.6.1 LAN9512

Der LAN9512-Chip ist insofern etwas Besonderes, weil hier zwei recht unterschiedliche Einheiten (LAN-Controller und USB Hub) miteinander in einem Gehäuse kombiniert sind. Im Gegensatz zum Broadcom BCM2835 gehört der LAN9512 zum üblichen Programm der Firma SMSC; er kann demnach auch über die bekannten Distributoren einzeln bezogen werden. Die Firma SMSC ist insbesondere für ihre LAN-Controller bekannt, die auf unzähligen Netzwetzwerkkarten und Mainboards eingebaut sind und stellt hierfür auch entsprechende Treibersoftware (Windows, Linux) zur Verfügung.

Im LAN9512 ist kein USB-Controller enthalten, sondern lediglich ein USB-Hub mit zwei Ports (Downstream USB PHY). Stattdessen fungiert der BCM2835 auch als USB-Controller gemäß der USB 2.0-Spezifikation und leitet die Signale zum USB-Hub (Upstream USB PHY).

Abbildung 3.19: Das Prinzipschaltbild des LAN9512

Beim Raspberry Pi Modell B+ wird der Typ LAN9514 eingesetzt, der statt über zwei über vier USB-Ports verfügt, ansonsten jedoch zum LAN9512 funktionsgleich ist. Weil jedoch die Register nicht identisch sind, werden hierfür auch neue Treiber benötigt. Nach der Übernahme der Firma SMSC gehören diese Controller aktuell zur Firma Microchip.

3.6.2 PHY und MAC

Mit PHY (Physical Layer) werden üblicherweise Schaltungseinheiten bezeichnet, die die Ebene der Bitübertragung (Schicht 1) repräsentieren. Dies hat seinen Ursprung im Standard des OSI-Schichtenmodells (Open Systems Interconnect), wel-

ches insgesamt sieben unterschiedliche Schichten mit eindeutigen Funktionen definiert, um somit die Unabhängigkeit von Hardware- und Software-Einheiten in Kommunikationssystemen herstellen zu können.

Der *Physical Layer* kommuniziert ausschließlich mit der darüberliegenden Schicht 2 – dem *Data Link Layer* –, was beim Ethernet-PHY die Verbindung mit einer MAC-Einheit (Media Access Control) bedeutet. Diese Trennung von PHY und MAC ist traditionell von Netzwerkadaptern her bekannt, wo eine einzige MAC-Einheit für unterschiedlichste PHY-Einheiten verwendet werden kann, die dann entweder für Koaxial- oder für Twisted Pair-Kabel oder auch für Lichtwellenleiter (Glasfaser) ausgelegt ist.

In einer MAC-Einheit werden die Datenpakete und Adressen gebildet sowie eine grundlegende Fehlererkennung ausgeführt. Jede Netzwerkeinheit verfügt über eine eindeutige, weltweit einmalige MAC-Adresse und kann somit eindeutig identifiziert werden. Eine IP-Adresse wird beispielsweise stets auf einer MAC-Adresse abgebildet. Die *MAC Engine* befindet sich im Ethernet-Controller des LAN95-Chips, der mit der Ethernet-PHY-Einheit verbunden und für Twisted Pair-Kabel ausgelegt ist.

Die Netzwerkeinheit unterstützt den Ethernet-Standard IEEE 802.3 mit 10 (10Base-T) und mit 100 MBit/s (100Base-TX) bei automatischer Detektierung der jeweiligen Betriebsart (Auto Negotiation). Außerdem wird Full-Duplex unterstützt, was gleichzeitiges Senden und Empfangen ermöglicht. Je nach Kommunikationspartner (Switch oder PC) werden die Leitungen automatisch umgeschaltet.

Abbildung 3.20: Die Schaltung des Netzwerk-Interfaces

3.6.3 Netzwerkverbindung

Üblicherweise ist bei der Verbindung von zwei gleichartigen Netzwerkeinheiten, etwa zweier PCs, ein Netzwerkkabel notwendig, bei dem die Sende- und Empfangsleitungen überkreuz laufen, so dass die Sendeleitung des einen PC an die Empfangsleitung des zweiten (und umgekehrt) gelangt. Diese *Cross-Kabel* sind also intern anders verschaltet als die üblichen Patch-Kabel, womit ein PC an einen Switch angeschlossen wird. Eine automatische Erkennung und Umschaltung zwischen Cross- und Patch-Modus ist laut Standard erst bei Gigabit-Ethernet (1000Base-T) vorgeschrieben, weshalb dann keine speziellen Cross-Kabel mehr benötigt werden. Obwohl der LAN9512 lediglich für maximal 100Base-TX ausgelegt ist, kennt er dennoch diese praktische Automatik, so dass sich der Anwender keine besonderen Gedanken beim Netzwerkanschluss machen muss.

Bei Netzwerkkarten mit Koaxial- oder Twisted Pair-Anschluss ist generell eine galvanische Trennung zwischen der Netzwerkleitung und der Netzwerkkartenelektronik notwendig. Beim Raspberry Pi findet man die hierfür notwendigen Übertrager (Transformatoren) jedoch nicht, denn diese befinden sich in der Netzwerkbuchse. Es wird hier also keine gewöhnliche Twisted Pair-Buchse (RJ45) eingesetzt, was zu einer sehr kompakten Elektronik führt, die mithilfe dreier Leuchtdioden auch den jeweiligen Netzwerkstatus des Boards visualisiert. LINK bedeutet, dass eine Netzwerkverbindung besteht, FULL DUPLEX weist die Funktion aus, dass gleichzeitiges Senden und Empfangen erfolgt, und 100M, dass der Modus mit 100 MBit/s aktiv ist. Es gibt auch RJ45-Netzwerkbuchsen mit integrierten Übertragern (Magnetics), die außerdem noch die Leuchtdioden für die Statusanzeige (Molex, Epcos) beinhalten, was eine weitere Vereinfachung beim Aufbau von Netzwerkinterfaces mit sich bringt und beim Modell B+ umgesetzt worden ist.

An den integrierten EEPROM-Controller kann prinzipiell ein externes EEPROM als Speicher für Konfigurationsdaten (USB Deskriptoren) und die MAC-Adresse angeschlossen werden, was bei der Raspberry Pi-Schaltung jedoch nicht genutzt wird, denn die Daten sind hier Chip-intern festgelegt.

3.6.4 TAP- und USB-Controller

Für die Programmierung und den Test des Chips ist ein *TAP Controller* (Test Access Port) im LAN9512/9514 integriert. Dieser führt vier einzelne Signale (TMS, TDI, TDO, TCK), die zusammengefasst als JTAG-Interface bezeichnet werden. Die *Joint Test Action Group* (JTAG) hat diesen Standard, der unter IEEE 1149 geführt wird, ursprünglich definiert. Mithilfe dieser genau spezifizierten Schnittstelle lassen sich – je nach Chiptyp – , unterschiedliche Tests (Boundary Scan) ausführen,

um bereits während der Fertigung oder auch später beim aufgelöteten Chip verschiedene Funktionsprüfungen ausführen zu können. Der BCM2835 besitzt ebenfalls ein derartiges JTAG-Interface (Abbildung 3.20). Die hierfür notwendige 8-polige Steckerleiste wird ab der Board Revision 2 allerdings nicht mehr bestückt.

Für die USB-Schaltung (Abbildung 3.21) auf dem Raspberry Pi-Board werden nur wenige zusätzliche Bauelemente benötigt. Der USB führt generell eine Spannung von 5 V, womit hier angeschlossene Geräte mit Strom versorgt werden können. Im Gegensatz zu den üblichen PC-Realisierungen, bei denen maximal 500 mA von einem USB-Port geliefert werden können, ist dies bei den ersten Raspberry Pi-USB-Anschlüssen weitaus weniger. Mit dem Modell B+ wird der maximale Strom per Software auf 600 mA begrenzt, was sich durch eine Anpassung in der Datei config.txt (safe_mode_gpio=4) theoretisch auf bis zu 1,2 A steigern lässt, wovon jedoch abzuraten ist, weil sich die Begrenzung in der Praxis als nützliche Schutzmaßnahme darstellt.

Abbildung 3.21: Die USB-Schaltung mit LAN9512

3.6.5 Polyfuses

Damit der Strom nicht unzulässigerweise zu hoch werden kann, sind die beiden USB-Ports (Abbildung 3.21) ursprünglich durch zwei so genannte *Polyfuses* geschützt, die bei einem Strom von ca. 140 mA auslösen. Falls also ein angeschlossenes USB-Gerät mehr Strom als 140 mA benötigt, wird die Verbindung und damit die Spannungsversorgung durch die Sicherung unterbrochen. Zu jeder dieser Schutzschaltungen gehören außerdem jeweils eine Diode, damit eine externe Spannung keinen Schaden anrichten kann, und mehrere Abblockkondensatoren.

Polyfuses sind selbstrückstellende Sicherungen, die nach der Auslösung – und dem Abziehen des Stromverbrauchers oder der Beseitigung eines Kurzschlusses – nach einiger Zeit wieder einschalten. Das Element entspricht einem Kaltleiter auf Polymerbasis mit einer nichtlinearen Widerstandskennlinie, welches sich durch den fließenden Strom erwärmt und hochohmig wird. Ist das Element abgekühlt wird es niederohmig und leitet wieder. Im Gegensatz zu üblichen Feinsicherungen müssen sie nicht ersetzt werden, dafür sind sie im Schaltverhalten jedoch relativ träge.

Bei den neueren Raspberry Pi-Boards (ab Modell B Revision 1.0+) hat man auf die Bestückung der beiden 100 nF-Kondensatoren (C30, C31) sowie die der Polyfuses für den USB verzichtet, stattdessen sind bei der Übergangsserie für die Sicherungen lediglich Lötbrücken vorhanden. Bei den aktuellen Boards ist das Platinenlayout gleich entsprechend geändert worden. Deshalb ist stets besondere Vorsicht beim Anschluss von USB-Geräten, die möglicherweise zu viel Strom ziehen können, geboten. Am besten werden USB-Geräte, vielleicht mit Ausnahme der Tastatur, über einen separaten USB-Hub mit Netzteil angeschlossen, so dass für den Raspberry Pi keine Gefahr besteht.

Eine weitere Polyfuse (Abbildung 3.22) befindet sich direkt in der Spannungsversorgungsleitung zwischen dem USB-Micro-Anschluss und dem Testpunkt 1 (Abbildung 3.23), an dem die notwendige Spannung von 5 V nachgemessen werden kann.

3.7 Spannungsversorgung und Taktung

Die Spannungsversorgung für das Raspberry Pi-Board muss von einem separaten Steckernetzteil mit Micro-USB-Anschlusskabel zur Verfügung gestellt werden, welches einen Strom von mindestens 700 mA liefern sollte. Aus den hiermit generierten 5 V werden auf der Platine drei verschiedene Spannungen für die einzelnen

Bauelemente mithilfe von entsprechenden Spannungsreglern (3,3 V und 2,5 V und 1,8 V) erzeugt.

Durch eine Polyfuse (s.o.) wird in der Raspberry Pi-Schaltung verhindert, dass ein Strom von mehr als 1 A fließen kann. Zwei Leuchtdioden, die sich in der Platinenecke bei den drei LEDs für die Visualisierung des Netzwerkstatus befinden, signalisieren, dass die Versorgungsspannung von 3,3 V vorhanden (rote LED) und der Prozessor aktiv ist (grüne Status-LED).

Abbildung 3.22: Diese Polyfuse befindet sich auf der Platinenrückseite unter der Micro-USB-Buchse.

Die Spannungsversorgung für ein Raspberry Pi-Board wird in den meisten Fällen von einem separaten Steckernetzteil mit Micro-USB-Anschlusskabel zur Verfügung gestellt, welches einen Strom von mindestens 700 mA liefern sollte. Aus den hiermit generierten 5 V werden auf der Platine drei verschiedene Spannungen für die einzelnen Bauelemente mithilfe von entsprechenden Spannungsreglern (3,3 V und 2,5 V und 1,8 V) erzeugt.

- **RG2-3,3 V:** NCP1117-3v3, 1A, TO-252/DPAK-Gehäuse, ON Semiconductor
- **RG3-2,5 V:** LP2980-2v5, 50 mA, SOT 23-Gehäuse, National Semiconductor
- **RG1-1,8V:** NCP1117-1v8, 1 A, SOT 223-Gehäuse, ON Semiconductor

Abbildung 3.23: Die Schaltungen für die Spannungsversorgung

3.7.1 Taktung

Die zweite Voraussetzung für die grundlegende Funktion ist neben einer korrekten Spannungsversorgung, dass sowohl für den BCM2835 als auch für den LAN9512 der jeweils benötigte Takt zur Verfügung gestellt wird, was über zwei separate Quarzoszillatoren erfolgt, die sich auf der Platinenunterseite befinden. Der LAN-Chip (Abbildung 3.21) wird mit 25 MHz und der BCM2835 mit 19,2 MHz getaktet.

Abbildung 3.24: Die Taktschaltung für den BCM2835

3.8 Reset-Schaltung

Ein Reset-Impuls versetzt einem Mikroprozessor oder einen Mikrocontroller in seinen logischen Ausgangszustand, bei dem alle Registerinhalte gelöscht werden und eine nachfolgende Neuinitialisierung stattfindet. In vielen Applikationen übernimmt ein spezieller Chip diese Reset-Funktion, die durch einen bestimmten Zustand, ein bestimmtes Ereignis oder auch die Betätigung eines Tasters ausgelöst werden kann. Damit sind eine ganze Reihe von Effekten, wie eine zu geringe Versorgungsspannung oder eine zu hohe Temperatur oder ein nicht geschlossener Kontakt, detektierbar, die einen unsicheren Betriebszustand des Prozessors hervorrufen können, so dass er zurückgesetzt wird. Je nach Reset-Chip und -Schaltung kann der Prozessor erst dann wieder in Betrieb gehen, wenn das verursachende Problem beseitigt worden ist.

Im BCM2835 ist nur eine einfache Reset-Schaltung integriert, die durch einen Low-Impuls am Eingang RUN ausgelöst wird. Dieser Eingang wird im Normalbetrieb durch einen Widerstand (R15: 10 kΩ) konstant auf High-Pegel gehalten, woraufhin der Reset automatisch dann einmal ausgelöst wird, wenn die Betriebsspannung

»hochgefahren« ist. Mit der neueren Platinenversion (Revision 2) ist eine kleine Zusatzschaltung hinzugekommen, die auch eine manuelle Reset-Auslösung gestattet. Hierfür sind auf der Platine die beiden Kontakte an P6 vorgesehen, an die ein Taster angeschlossen werden kann. Insbesondere bei Hardware-nahen Applikationen kommt es durchaus vor, dass der Raspberry Pi hängt, so dass ein Tastendruck ihn dann wieder neu starten kann. Im Gegensatz zum Ein- und Ausschalten der Spannungsversorgung, die ebenfalls für einen Reset sorgt, verläuft der Reset per Taster etwas schneller.

Abbildung 3.25: Die Reset-Schaltung, die mit der Revision 2 hinzugekommen ist, kann optional mit einem Taster (an P6) bestückt werden.

Abbildung 3.26: Der zusätzliche Reset-Switch an P6

Mit dem Modell B+ wurde eine weitere Veränderung der Reset-Schaltung durchgeführt, indem jetzt ein spezieller Reset-Chip (APX 803 der Firma Diodes Incorporated) eingesetzt wird, der die 5 V-Versorgung überwacht und bei einem entsprechenden Pegel einen Reset-Impuls generiert, was zuverlässiger als die vorherigen Lösungen funktioniert. Auf der Modell+ Platine ist aber weiterhin die Möglichkeit für das Nachrüsten eines Reset-Switch vorgesehen, wofür unter den beiden LEDs am linken Platinenrand zwei Löcher mit der Beschriftung Run (siehe Abbildung 6.2) vorgesehen sind.

3.9 DSI- und CSI-Schaltung

Auf der Raspberry Pi-Platine sind zwei 15-polige Klemmhalterungen (Flat Flex Connector) zu erkennen, die für die Aufnahme von Folienkabel gedacht sind. Es handelt sich dabei um zwei *Serial Interfaces* (SI), wobei das eine (S2) für den Anschluss eines Display (DSI) und das andere (S5) für den Anschluss einer Kamera (CSI) gedacht ist.

Beide Interfaces führen jeweils sechs differentielle Signalpaare, mit denen die Daten seriell übertragen werden, ähnlich wie es bei PCI-Express praktiziert wird. Diese beiden Interfaces entsprechen dem Standard *Mobile Industry Processor Interface* (MIPI), der von der *MIPI Alliance* veröffentlicht und gepflegt wird. Das Besondere bei MIPI ist, dass der Stromverbrauch eine besondere Berücksichtigung erfährt, weil er für mobile Geräte nun einmal ein wichtiges Kriterium ist. Neben dem

eigentlichen Interface, welches eine maximale Datenrate von 5 GBit/s erlaubt, definiert die MIPI Alliance verschiedene Videoformate und Kommandos.

Abbildung 3.27: Zwei spezielle Schnittstellen für Display und Kamera

Am Connector des *Camera Serial Interfaces* ist für die Steuerung ein I²C-Bus-Interface (siehe Kapitel 6.5) mit der dazugehörigen Taktleitung (SCL1) und der Datenleitung (SDA1) vorhanden. Die Signale des zweiten I²C-Bus-Interface (SCL0, SDA0) befinden sich am GPIO-Connector. Dabei sind diese beiden I²C-Bus-Interfaces mit der zweiten Platinenversion gegeneinander getauscht worden, was für die Verwendung zwar keine direkte Bedeutung hat, gleichwohl muss die jeweilige Bezeichnung bei der Programmierung (Kapitel 6.5.4) beachtet werden.

Abbildung 3.28: Die beiden Interfaces laut MIPI-Standard

3.9.1 Pi Camera

Nach den beiden Raspberry Pi-Modellen A und B wurde passend zum *Camera Serial Interfaces* von der Raspberry Pi Foundation im Jahre 2013 offiziell die Pi Camera vorgestellt, die speziell für den Raspberry Pi entwickelt worden ist.

Die 5 Megapixel-Kamera kann Fotos in einer maximalen Auflösung von 2592 x 1944 Pixel und Videos mit 1080 x 720 Pixel aufnehmen. Bei einer Auflösung von 640 x 480 Bildpunkten sind bis zu 30 Bilder pro Sekunde (fps) möglich. Ein fast baugleiches Modell ist die Pi Camera NoIR, die bei ansonsten gleichen Daten kein Infrarotfilter besitzt (No Infra Red), weshalb sie für Infrarotaufnahmen oder für Spezialaufnahmen bei schwachem Licht geeignet ist.

Abbildung 3.29: Die angeschlossene Pi Camera

Die Kamera befindet sich auf einer kleinen Platine von ca. 2,5 x 2,5 cm mit dem hier fest angeschlossenen CSI-Kabel, welches mit dem CSI-Connector auf der Raspberry Pi-Platine zu verbinden ist. Hierfür ist der Klemmsteg oben an der Buchse hochzuziehen, das CSI-Kabel in die Fassung einzusetzen (Kabelbeschriftung und Kamera oben), das Kabel einzustecken und der Steg wieder herunterzudrücken. Auf dem Objektivloch der Kamera befindet sich üblicherweise eine Schutzfolie, die vor der Benutzung zu entfernen ist. Die Brennweite ist fest, so dass sich der Bildfokus nicht ändern lässt.

Um die Kamera verwenden zu können, muss sie dem Betriebssystem bekannt gemacht werden. Es ist entsprechende Software für die Aufnahme von Fotos und Videos notwendig. Am einfachsten funktioniert dies mit Raspbian, wofür das System lediglich zu aktualisieren ist (siehe Kapitel 2.7). Nach diesem Vorgang und einem Reboot steht eine neue raspi-config-Version zur Verfügung, bei der sich im Menü die Kamera aktivieren lässt (Enable Camera). Außerdem steht dann unter */opt/vc/bin* neue Software zur Verfügung, die für die Pi Camera vorgesehen ist.

Mit *raspistill* wird ein Foto aufgenommen und mit raspivid ein Video, wobei den Kommandozeilenbefehlen beim Aufruf verschiedene Parameter mitgegeben werden können, die eine Vielzahl an Optionen zur Verfügung stellen. Diese lassen sich mit der jeweils dazugehörigen Hilfe (z.B. raspistill -?) anzeigen. Während der Aufnahme leuchtet eine rote LED zur Signalisierung der Aktivität.

Mit der folgenden Zeile wird ein Foto aufgenommen, wobei das Bild standardmäßig für fünf Sekunden angezeigt wird.

```
pi@raspberrypi /opt/vc/bin $ raspistill -o
```

Um ein Foto zu speichern, wofür Root-Rechte notwendig sind, wird ein Dateiname mit dem gewünschten Bildformat als Extension angegeben:

```
pi@raspberrypi /opt/vc/bin $ sudo raspistill -o bildtest1.jpg
```

Das Aufnehmen eines Videos ist ähnlich einfach und *raspivid* verfügt über fast die gleichen Optionen wie *raspistill*. Mit der folgenden Zeile wird für 20 Sekunden (Option -t 20000) ein Video aufgenommen und in der Datei video1.h264 gespeichert:

```
pi@raspberrypi /opt/vc/bin $ sudo raspivid -t 20000 -o video1.h264
```

Ein Mikrofon beinhaltet die Kamera nicht, so dass Audio separat aufzunehmen ist. Für erste Tests und Experimente reichen die Kommandozeilenbefehle, die Linux-üblich auch automatisiert werden können, sicher aus. Die Integration der Kamera in eigenen Applikationen wird mittlerweile durch einige verfügbare Bibliotheken und Skriptlösungen, etwa für Python (picamera API) oder C++ (RaspiCam API), unterstützt.

3.10 Compute Module

Zusätzlich zu den beiden Versionen Modell A und B ist im Frühjahr 2014 von der *Raspberry Pi Foundation* eine neue Platine offiziell vorgestellt worden. Dabei handelt es sich um das *Raspberry Pi Compute Module*, welches in erster Linie für industrielle (Embedded-)Anwendungen vorgesehen ist. Es arbeitet ebenfalls mit dem

SoC BCM2835 der Firma Broadcom und weist einen RAM-Speicher von 512 MByte auf.

Der Flash-Speicher (4 GByte) ist hier in Form eines eMMC-Chips (Embedded Multimedia Card), wie er auch beim *Beaglebone Black* zum Einsatz kommt, direkt auf dem Modul eingelötet. Das Raspberry Pi Compute-Modul hat die Form eines DDR2-SO-DIMM-Speichermoduls, wie es bei Notebooks verwendet wird. Es verfügt, abgesehen von den Platinensteckkontakten, über keinerlei weitere Anschlüsse.

Abbildung 3.30: Das Raspberry Pi Compute Modul ist in Form eines Speichermoduls ausgeführt und beherbergt lediglich den SoC BCM2835 mit integriertem RAM sowie den Flash-Speicher in Form eines eMMC-Chips.

Mit dem Modul allein lässt sich zunächst nichts bewerkstelligen, denn es wird ein entsprechendes »Umfeld« mit der Spannungsversorgung und den Peripherie-Chips und den passenden Anschlüssen benötigt. Hierfür kann eine eigene individuelle Hardware entwickelt werden, die lediglich diejenigen Komponenten beinhaltet, die für die jeweilige Embedded-Applikation benötigt werden, was einen direkten Einfluss auf den Stromverbrauch hat und außerdem zu sehr kompakten Lösungen führen kann. Wie bei den bisherigen Raspberry Pi-Modellen werden wieder die drei Spannungen von 1,8 V, 2,5 V und 3,3 V für den BCM2835 und den Speicher benötigt, wie es in der Prinzipschaltung (Abbildung 3.31) zu erkennen ist.

Abbildung 3.31: Das Prinzipschaltbild des Raspberry Pi Compute Module mit den notwendigen Versorgungsspannungen.

Zum Modul gibt es auch ein offizielles »Trägerboard«, mit dem es getestet werden kann und mit dem sich Probeaufbauten durchführen lassen. Dieses I/O-Board (Abbildung 3.32) verfügt über die Konverterschaltung für die Spannungsversorgung, die auch hier wieder von einem separaten 5 V-Steckernetzteil zur Verfügung gestellt werden muss, sowie über den HDMI-Grafikconnector, zwei USB-Ports, jeweils zwei Kamera- und LCD-Anschlüsse für Flachbandkabel und zwei Steckerleisten mit GPIO-Ports. Einen Audio- und auch einen Netzwerkanschluss gibt es hier nicht. Während Audio für Embedded-Applikationen meist verzichtbar ist, kann das Fehlen eines LAN-Controllers durchaus als Einschränkung angesehen werden. Das Compute Module wird zusammen mit dem I/O-Board, zwei Adaptern für CSI und DSI, einem Steckernetzteil (5 V, 1A) sowie vier Steckleitungen als *Compute Module Development Kit* von den bekannten Distributoren verkauft.

Abbildung 3.32: Das Raspberry Pi I/O-Board mit eingesetztem Compute Modul

3.10.1 Spannungsregler

Grundsätzlich weisen Linearregler im Gegensatz zu Schaltreglern einen geringen Wirkungsgrad auf und setzen die überschüssige Leistung in Wärme um, so dass sie für Low-Power- und/oder Embedded-Applikationen eher ungeeignet sind. Dieser Tatsache wurde auch beim Raspberry Pi I/O-Board entsprechend Rechnung getragen, denn hier wird ein Schaltregler (PAM2306) der Firma *Diodes Incorporated* für die Spannungsumsetzung verwendet.

Die Schaltung basiert auf einer Applikation von *Diodes Incorporated* und kann aus einer Eingangsspannung zwei unterschiedliche Ausgangsspannungen (3,3 V, 2,8 V, 2,5 V, 1,8 V, 1,5 V, 1,2 V) generieren. Außerdem ist eine individuell einstellbare Ausgangsspannung per externer Beschaltung konfigurierbar. Beide Ausgänge sind in der Lage, bis zu 1 A liefern zu können. Die Eingangsspannung darf laut der *Raspberry Pi Foundation* im Bereich von 4,5 V bis zu 5,5 V liegen, was bedeutet, dass auch eine Versorgung durch Batterien und Akkus möglich ist. Der PAM2306 unterstützt einen Shutdown-Modus, der beim I/O-Board allerdings nicht eingesetzt wird und mit einem typischen Strom von 100 mA auch nicht als besonders effektiv zu betrachten ist.

Abbildung 3.33: Die Schaltung für die Spannungsversorgung auf dem I/O-Board

Auf dem Raspberry Pi I/O-Board werden mit dem PAM2306-Regler die beiden Ausgangsspannungen von 3,3 V und 1,8 V generiert, während für die 2,5 V ein separater Linear-Regler (Low Dropout) vom Typ AP7115 (max. 150 mA), der ebenfalls von der Firma *Diodes Incorporated* stammt, eingesetzt wird.

3.10.2 Embedded Multimedia Card beschreiben

Das I/O-Board besitzt keinen SD Card-Steckplatz, weil dies bei Embedded-Systemen als instabile mechanische Lösung betrachtet wird und sich deshalb verbietet. Stattdessen wird der Flash-Speicher mit einem eMMC-Chip auf dem Compute Module (vgl. Abbildung 3.30) realisiert, der bei Auslieferung des Boards leer ist, so dass das Betriebssystem hier zunächst installiert werden muss, was um einiges aufwendiger ist, als eine übliche SD Card mit einem Image zu beschreiben. Der eMMC-Chip ist elektrisch direkt mit dem BCM2835 verbunden. Als Modulkontakte stehen diese Signale nicht zur Verfügung.

Für die Übertragung des Betriebssystems in den eMMC-Chip wird ein Linux-Computer als Host benötigt, was durchaus ein Raspberry Pi (Modell B) sein kann. Er sollte mit einer SD Card verwendet werden, die mindestens über eine Kapazität

von 8 GByte verfügt, denn neben dem Betriebssystem für den Raspberry Pi als Host muss hier auch noch das Image selbst Platz haben, das in den eMMC-Chip geschrieben werden soll und typischerweise um die 4 GByte (Raspbian: 3.276.800 kByte) groß ist. Außerdem sind noch einige Tools für die Übertragung des Systems in den eMMC-Chip notwendig, so dass man mit einer 16 GByte-SD-Card auf der sicheren Seite ist.

Zunächst ist das Linux-Host-System auf den neuesten Stand zu bringen, wie es im Kapitel 2.7 erläutert ist. Danach sind auf dem Linux-Host die *USB Tools* sowie die USB-Bibliothek (libusb) notwendig. Hierfür wird die komplette *Tools Repository* auf das Hostsystem übertragen mit:

```
pi@raspberrypi ~ $ git clone –depth=1
https///github.com/raspberrypi/tools
```

was durch die Anzeige *Cloning into `Tools`* und durch die abschließende Meldung done signalisiert wird. Nach Beendigung des Vorgangs wird in das neu angelegte Verzeichnis gewechselt:

```
pi@raspberrypi ~ $ cd tools/usbboot
```

Die USB-Bibliothek wird in *dev* installiert mit:

```
pi@raspberrypi ~ /tools/usbboot $ sudo apt-get install-1,0-0-dev
```

Diese beiden Vorgänge müssen ohne Fehlermeldungen absolviert worden sein, so dass jetzt der Build-Vorgang für das *usbboot tool* durch die Angabe von *make* (im usbboot-Verzeichnis) ausgelöst werden kann.

Anschließend ist *rpiboot*, welches Bestandteil der *USB Tools* ist, aufzurufen, woraufhin auf die Etablierung einer USB-Verbindung gewartet wird (Anzeige: Waiting for BCM2835 …):

```
pi@raspberrypi ~ /tools/usbboot $ sudo ./rpiboot
```

Das Verzeichnis */usbboot* befindet sich unter */home/pi/tools/usbboot*.

Auf dem I/O-Board mit dem installiertem Compute Module muss sich der Jumper J4 in der Stellung *Enable* (Usb Slave Boot) befinden.

Das USB-Kabel (Typ: Micro-USB auf Standard-USB) wird mit dem I/O-Board (J15: USB slave port) und dem Host-System (Raspberry Pi) verbunden, und die Spannungsversorgung für das I/O-Board ist einzuschalten, so dass die rote LED (PWR) auf dem I/O-Board leuchtet. Die grüne LED (ACT) direkt darüber bleibt zunächst aus, bis das System auf dem Compute Module funktioniert.

Kurz danach detektiert das laufende *USB Tool* (sudo ./rpiboot) das I/O-Board und sendet einen *Boot Code*, um den Kontakt mit dem eMMC-Chip aufzubauen. Wenn dieser Vorgang abgeschlossen ist, erscheint ein neues sda-Device unter */dev*.

```
pi@raspberrypi / $ cd home/pi/tools/usbboot
pi@raspberrypi ~/tools/usbboot $ sudo ./rpiboot
Waiting for BCM2835 ...
Found serial = 0: writing file usbbootcode.bin
Successful
Waiting for BCM2835 ...
Found serial = 1: writing file msd.bin
Successful
Raspberry Pi is now a mass storage device, use lsblk to find it
pi@raspberrypi ~/tools/usbboot $
```

Abbildung 3.34: Die Verbindung zwischen dem Host und dem Compute Module, welches als USB-Speicherträger fungiert, ist korrekt hergestellt.

Zur Kontrolle, ob das neue USB Device (das I/O-Board) erkannt worden ist, empfiehlt sich die Anwendung des lsusb-Befehls oder auch die Eingabe von *lsblk*. Wichtig ist an dieser Stelle, dass tatsächlich ein /sda- oder – je nach Konfiguration bzw. Ausstattung des Host-Systems – auch ein /sdb-Device neu hinzugekommen ist.

Für die Übertragung des Image auf den eMMC-Chip wird der Befehl *dd* (Dump Device) mit den Optionen *if* (input file), *of* (output file) und *bs* (Block Size 4 MBit) wie folgt aus demjenigen Verzeichnis eingesetzt (hier /media), wo das Image abgelegt worden ist:

```
pi@raspberrypi /media $ sudo dd if=wheezy-raspbian.img of=/dev/sda
bs=4MiB
```

Das jeweils zu verwendende Image (hier wheezy-raspbian.img) lässt sich mit den üblichen Linux-Methoden, wie beispielsweise mit dem Befehl *wget*, direkt aus dem Internet beziehen oder auch per Kopie mittels PuTTY von einem PC aus mit SSH/FTP (siehe Kapitel 4.8) auf den Host-Computer übertragen.

Nachdem das Image übertragen worden ist, was mehrere Minuten dauert, wobei die ACT-LED am Raspberry Pi-Host flackert, ist das verbindende USB-Kabel zwischen Compute Module und Host-Computer abzuziehen und wieder anzustecken, damit eine Neuinitialisierung stattfindet. Danach sind zwei weitere Partitionen hinzugekommen, wobei */dev/sda* weiterhin den eMMC-Speicher repräsentiert, */dev/sda1* entspricht der ersten Partition im FAT-Format und */dev/sda2* der zweiten Partition im Linux-Format. Das bedeutet, dass die Einrichtung des Systems auf dem eMMC-Speicher beendet ist und nachfolgend von ihm gebootet werden kann, wofür der Jumper J4 wieder in die Stellung *Disable* (Usb Slave Boot) zu setzen und das USB-Kabel zum Host zu entfernen ist. Wie üblich, wird dann beim ersten Boot raspi-config automatisch gestartet.

```
pi@raspberrypi ~/tools/usbboot $ lsusb
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 04d9:1503 Holtek Semiconductor, Inc. Shortboard Lefty
Bus 001 Device 007: ID 0a5c:0001 Broadcom Corp.
pi@raspberrypi ~/tools/usbboot $ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda            8:0    1   3.7G  0 disk
├─sda1         8:1    1    56M  0 part
└─sda2         8:2    1   3.6G  0 part
mmcblk0      179:0    0  14.9G  0 disk
├─mmcblk0p1 179:1    0    56M  0 part /boot
└─mmcblk0p2 179:2    0  14.9G  0 part /
pi@raspberrypi ~/tools/usbboot $
```

Abbildung 3.35: Das USB Device 007 (Broadcom Corp.) entspricht dem Compute Module, welches als Datenträger gegenüber dem Host mit sda und den beiden Partitionen sda1 und sda2 erscheint.

Für das zu übertragende Image sollte keinesfalls ein in irgendeiner Form angepasstes Image, sondern tatsächlich ein neues (von der Raspberry Pi Foundation) verwendet werden. Auch wenn es praktisch erscheint ein Image zu übertragen, welches aus einem bereits lauffähigen System mit den entsprechenden Anpassungen und Einstellungsoptionen stammt, führt dies meist zu einem Bootproblem mit einer Fehlermeldung wie *PANIC: VFS: Unable to mount root fs on unknown-block* beim Compute Module, weil die zulässige Größe der Bootpartition überschritten wird und deshalb nicht korrekt gemounted werden kann. Üblicherweise werden die Partition per raspi-config und die Option *Expand Root Partion* auf die maximale Größe der SD-Card angepasst, so dass hier die Ursache für das spätere Bootproblem zu sehen ist, auch wenn das gewissermaßen »expandierte Image« nur von einer 4 GByte-SD-Card stammt.

3.11 Die +Modelle

Mitte des Jahres 2014 ist das *Modell Raspberry Pi Modell B+* am Markt erschienen, dem Ende des Jahres das *Modell A+* gefolgt ist. In den wesentlichen Daten entsprechen sie ihren Vorgängern, denn sie arbeiten beide mit dem BMC2835, der mit 700 MHz getaktet wird. Entsprechend verfügt das Modell B+ über einen SDRAM-Speicher von 512 MByte, das Modell A+ über 256 MByte, welches zudem keinen LAN- und nur einen USB-Anschluss besitzt. Bis auf diese drei Unterschiede sind auch die neuen Modelle wieder prinzipiell baugleich, obwohl das Model A+ über eine kleinere Platine verfügt. Die Verkleinerung – quasi auf die Länge der 40-poligen Pfostenleiste plus der zwei Befestigungslöcher – ist deswegen möglich, weil hier der Chip (LAN9514) für das LAN sowie den USB-Hub nicht notwendig ist, genauso wenig wie die LAN-Buchse und die vier USB-Buchsen. Der USB-Port wird beim Modell A+ direkt vom BMC2835 gesteuert.

Abbildung 3.36: Das Modell B+ und das Modell A+

Die neuen bzw. geänderten Eigenschaften der +Modelle gegenüber den vorherigen sind wie folgt:

- Der SD Card-Slot wird durch einen Micro SD-Slot ersetzt, so dass die hier einzusetzende Speicherkarte nicht mehr so weit herausragt und im (neuen) Gehäuse für einen Raspberry Pi fast mit dem Platinenrand abschließt. An der Handhabung der SD Card (Image aufspielen etc.) ändert sich nichts.
- Der GPIO-Port wird um 14 weitere Anschlüsse erweitert, wobei die bisherigen 26 Pins von ihrer Belegung und Funktion her gleich bleiben. Weil der GPIO-

Header auf der Platine weiter nach rechts verschoben worden ist, was auch die Audio- und Videobuchse an dieser Stelle »verdrängt« hat, lassen sich einige Erweiterungsplatinen nicht mehr aufstecken, was ebenfalls für die bekannten 26-poligen Adapter wie den Pi Cobbler und ähnliche Lösungen (mit Stecker) gilt. Deshalb gibt es hierfür eine Adapter-Platine, die die 26 GPIO-Signale um ca. 0,5 cm weiter nach links versetzt, was eher als »Huckepack-Notlösung« zu verstehen ist und sich nicht für Neuentwicklungen und eine stabile Konstruktion empfiehlt. Eine ausführliche Beschreibung der GPIO-Leitungen ist im Kapitel 3.12 zu finden.

Abbildung 3.37: Ein GPIO-Adapter für ältere Platinen, die damit auch auf das Modell B+ gesteckt werden können.

- Das Modell B+ verfügt jetzt über vier statt nur über zwei USB-Ports, wofür ein anderer USB/Ethernet-Hub (LAN9514) zum Einsatz kommt. Prinzipiell ist er ansonsten mit dem Vorgänger LAN9512 (siehe Kapitel 3.6) identisch. Dennoch wird ein neuer Treiber benötigt, damit nicht nur die zusätzlichen USB-Ports, sondern auch der LAN-Controller funktionieren. Deshalb sollte im Bedarfsfall zunächst eine Aktualisierung des Systems vorgenommen werden, wenn ein Umstieg vom Modell B zum Modell B+ geplant ist.
- Die gelbe Cinchbuchse für das Composite-Video-Signal entfällt. Dieses Videosignal wird auf einen Kontakt des Audioausgangs gelegt, so dass hierfür eine neue Buchse mit vier Kontakten (Audio links, Audio rechts, GND, VID) und dementsprechend ein passender 4-poliger Klinkenstecker (3,5 mm) notwendig ist. Die Audio/Video-Buchse befindet sich nunmehr auf der anderen Platinenseite, rechts neben dem HDMI-Anschluss.

Abbildung 3.38: Die Signalbelegung der Audio/Video-Verbindung

- Der Anschluss (Micro-USB) für das externe Netzteil befindet sich links neben dem HDMI-Connector Der bisher an dieser Stelle »herausragende« Abblockkondensator (C6) für den 3,3 V-Spannungsregler entfällt.
- Die Stromaufnahme des Raspberry Pi reduziert sich um bis zu 1 W, weil statt der LDO-Regler jetzt Schaltregler (wie beim Compute Module, siehe Abbildung 3.33) eingesetzt werden. Die Eingangsspannung (aus einem Steckernetzteil) darf im Bereich von 4,75 V bis 5,25 V liegen.
- Der Kamera- und der Display-Anschluss (CSI, DSI) sowie der HDMI-Ausgang und der LAN-Anschluss bleiben unverändert.
- Die Platine hat abgerundete Ecken, vier neu positionierte Befestigungslöcher und die Anschlussbuchsen (USB, LAN) schließen jetzt alle nahezu mit der Platine ab. Insbesondere aufgrund der zusätzlichen USB-Buchsen und der Verlagerung einiger Anschlüsse werden für die +-Modelle andere Gehäuse benötigt.

Das Modell B+ wird zum gleichen Preis wie das Vorgängermodell angeboten, welches laut der *Raspberry Pi Foundation* für unbestimmte Zeit weiterhin produziert wird, weil aufgrund der mechanischen Änderungen keine Kompatibilität gegeben ist.

3.12 General Purpose Input Output

Die 26-polige Steckerleiste auf einer Raspberry Pi-Platine führt verschiedene Signale, die zusammengefasst unter *General Purpose Input Output* (GPIO) geführt werden. GPIO ist eine allgemeine Bezeichnung, die bei vielen Mikrocontrollern oder auch Schnittstellen verwendet wird und für sich allein nichts über die einzelnen Signale und deren elektrische Daten oder über die Art und Weise der Programmierung aussagt. GPIO-Schnittstellen gibt es demnach in unzähligen Ausführungen, wobei hier die des Raspberry Pi von Bedeutung ist. Dieser GPIO-Port ist für den

Anschluss von (eigenen) Elektronikschaltungen vorgesehen, was im einfachsten Fall eine Leuchtdiode für ein Output-Signal und ein Taster für ein Input-Signal sein kann.

Die GPIO-Signale werden direkt vom BCM2835 zur Verfügung gestellt, weshalb beim Anschluss äußerste Vorsicht geboten ist, damit der Chip nicht unabsichtlich zerstört wird, denn eine Schutzschaltung gibt es hier nicht. Bereits statische Aufladung, wie sie etwa mit kunststoffhaltigen Teppichen oder Kleidungsstücken auftritt, kann den BCM2835 beim Berühren der GPIO-Kontakte mit dem (aufgeladenen) Finger zerstören. Damit wäre dann gleich das komplette Board unbrauchbar.

Das Interface arbeitet mit einer Spannung von maximal 3,3 V, so dass hier keine höheren Spannungen angelegt werden dürfen. Welcher Strom an den einzelnen Pins maximal erlaubt ist, wird nicht offiziell genannt. Weder die *Raspberry Pi Foundation* noch die Firma Broadcom haben konkrete elektrische Kenndaten hierzu veröffentlicht.

Im Vergleich mit anderen ARM11-Prozessoren kann man jedoch davon ausgehen, dass 5 mA (Source + 5 mA, Sink - 5 mA) als ein geeigneter Wert für die explizit mit GPIO bezeichneten Signale anzusehen ist, auch wenn bei einigen ARM11-Typen 8 mA und mehr spezifiziert werden. Offiziell wird zwar ein maximaler Strom von 16 mA (- 2 mA) angegeben, allerdings bezieht sich diese Angabe nicht auf einen einzelnen Pin, sondern auf eine *Funktionsgruppe*, womit wahrscheinlich die Gruppen GPIO, SPI, I²C sowie die seriellen Signale TXD und RXD gemeint sind, so dass demnach lediglich 2 mA (- 0,3 mA) für einen GPIO-Pin zulässig sind. Eindeutig sind die Angaben zum maximal erlaubten Strom jedoch nicht.

Abbildung 3.39: Die 26-polige GPIO-Kontaktleiste

In der Tabelle 3.3 sind die einzelnen Signale, wie sie am 26-poligen Anschluss vorhanden sind, angegeben. Dabei fallen die ungewöhnlichen Nummerierungen und Bezeichnungen auf, was zum einen der Doppelfunktion einiger Pins geschuldet ist und zum anderen der Anschlussbezeichnung am BCM2835. Diese ist die gewissermaßen bis an die Steckerleiste weitergeführt worden, so dass einige GPIO-Leitungen scheinbar gar nicht existieren. Sie sind jedoch nicht mit der Steckerleiste,

sondern mit anderen Peripherieeinheiten auf dem Board verbunden. Logischer erscheint demgegenüber die Bezeichnung, wie sie auch im Schaltplan (Abbildung 3.40) angewendet wird, die die GPIO-Anschlüsse kontinuierlich mit GPIO_GEN0 (Pin 11) bis GPIO_GEN6 (Pin 22) durchzählt.

In der Tabelle 3.3 sind zur Orientierung beide Bezeichnungsschemata angeführt. In der Tabelle 3.4 wird die Zuordnung noch einmal separat verdeutlicht. Demnach gibt es lediglich fünf »richtige« GPIO-Leitungen (Pins 11, 15, 16, 18, 22), die als universelle Ein- oder Ausgänge genutzt werden können und keine Doppelfunktion aufweisen.

Tabelle 3.3: Die einzelnen Signale am 26-poligen GPIO-Anschluss (Revision 2)

Pin	Signal	Raspberry Pi	Pin	Signal	Raspberry Pi
1	3,3 V	3,3 V	2	5 V	5 V
3	GPIO2 oder I²C: SDA	SDA0	4	-	(5 V)
5	GPIO3 oder I²C: SCL	SCL0	6	Masse	Ground
7	GPIO4 oder General Purpose Clock	GPIO_GCLK	8	GPIO14 oder UART TXD	TXD0
9	-	(Ground)	10	GPIO15 oder UART RXD	RXD0
11	GPIO17	GPIO_GEN0	12	GPIO18 oder PWM_CLK	GPIO_GEN1
13	GPIO27 oder PCM_DOUT	GPIO_GEN2	14	-	(Ground)
15	GPIO22	GPIO_GEN3	16	GPIO23	GPIO_GEN4
17	-	(3,3 V)	18	GPIO24	GPIO_GEN5
19	GPIO10 oder SPI: MOSI	SPI_MOSI	20	-	(Ground)
21	GPIO9 oder SPI: MISO	SPI_MISO	22	GPIO25	GPIO_GEN6
23	GPIO11 oder SPI: CLK	SPI_CLK	24	GPIO8 oder SPI: Chip Select 0	SPI_CE0_N
25	-	(Ground)	26	GPIO7 oder SPI: Chip Select 1	SPI_CE1_N

Die 26-polige Kontaktleiste ist zweireihig und im üblichen 2,54 mm-Raster ausgeführt. Die Pinnummerierung entspricht der üblichen Zählweise, d.h. in der unteren Reihe (zur Platineninnenseite hin) befinden sich stets die ungeraden Kontakte (1-25) und in der oberen (zum Platinenrand hin) die geraden. Der Pin 1 ist auf der Platine gekennzeichnet, was jedoch nur schwach zu erkennen ist, so dass die Abbildung 3.39 und die Tabellen 3.3 und 3.4 der leichteren Orientierung hilfreich sind.

Im Elektronik-Handel sind fertig konfektionierte 26-polige Flachbandkabel zu erwerben. Es gibt zudem spezielle Verbindungen (Breakout Kit, Pi Cobbler) mit Anschlussplatinen als Raspberry Pi-Zubehör, um die einzelnen Signale entsprechend herauszuführen und verdrahten zu können. Dabei ist stets darauf zu achten, dass beim Anschluss bzw. beim Zusammenbau einer Breakout Kit-Platine nicht unabsichtlich eine Verdrehung der Signalzuordnung hergestellt wird, was sich als eine der häufigsten Fehlerquellen beim GPIO-Anschluss erweist. Auf die +Modelle mit der erweiterten 40-poligen GPIO-Leiste passen diese 26-poligen Adapter bzw. Platinen mechanisch nicht.

Einige Anschlüsse haben neben der Funktion, dass sie wahlweise als Eingang oder als Ausgang geschaltet werden können (GPIO), alternative Funktionen, die durch entsprechende Treibereinträge in den Konfigurationsdateien des Betriebssystems (Linux) aktiviert werden können (Kapitel 6). Die Funktionen für den *General Purpose Clock* (Pin 7) sowie für das PWM-Ausgangssignal (Pin 12), welches auch als *PCM Clock* mit dem dazugehörigen Datensignal *PCM Data Out* (Pin 13) geführt wird, werden vergleichsweise selten eingesetzt, so dass diese Pins vorzugsweise ebenfalls als GPIO-Anschlüsse fungieren. Die PCM-Signale kommen für Audioerweiterungen wie HiFiBerry oder der Wolfson Card zum Einsatz und stehen dann nicht als GPIO-Anschlüsse zur Verfügung.

Neben den (reinen) GPIO-Signalen führt die GPIO-Leiste die Signale für insgesamt drei digitale Schnittstellen. Es gibt hier den I^2C-Bus (SDA an Pin 3, SCL an Pin 5), der entsprechend mit zwei Pull-Up-Widerständen (R1, R2 mit 1,8 kΩ) verschaltet ist (Abbildung 3.40), so dass sich bei diesen beiden Pins eine Verwendung als »gewöhnlicher« GPIO-Port ausschließt.

Außerdem werden die Signale (Pins 19, 21, 23, 24, 26) für den SPI-Bus bereitgestellt. Mithilfe der zwei dazugehörigen Chip Select-Signale (Pin 24, 26) können zwei Chips, die mit einem *Serial Peripheral Interface* (SPI) ausgestattet sind, direkt angeschlossen werden. Sowohl I^2C als auch SPI eignen sich insbesondere für die Verbindung von »intelligenten« Sensoren, die intern über eine entsprechende Sig-

nalaufbereitung verfügen und deren Messwerte direkt per Bus ausgelesen werden können.

Tabelle 3.4: Die GPIO-Signale bei der Revision 2 mit den Pinzuordnungen und ihren alternativen Funktionen (* = Belegung Revision 1)

Zählung	GPIO-Signal	Pin. Nr.	Alternative Funktion
1	GPIO2 (GPIO0)*	3	I^2C: SDA
2	GPIO3 (GPIO1)*	5	I^2C: SCL
3	GPIO4	7	General Purpose Clock
4	GPIO7	26	SPI: Chip Select 1
5	GPIO8	24	SPI: Chip Select 0
6	GPIO9	21	SPI: MISO
7	GPIO10	19	SPI: MOSI
8	GPIO11	23	SPI: CLK
9	GPIO14	8	UART: TXD
10	GPIO15	10	UART: RXD
11	GPIO17	11	keine
12	GPIO18	12	PWM/PCM Clock
13	GPIO27 (GPIO21)*	13	PCM Data Out
14	GPIO22	15	keine
15	GPIO23	16	keine
16	GPIO24	18	keine
17	GPIO25	22	keine

Abbildung 3.40: Die Schaltung für die GPIO- und Schnittstellensignale

Der Raspberry Pi kann auch über eine serielle Schnittstelle (RS232) mit geeigneter Peripherie kommunizieren. Hierfür werden lediglich die Signale TXD (Pin 8) und RXD (Pin 10) zur Verfügung gestellt, womit sich die Minimalausführung (ohne Kontrollsignale) einer RS232-Schnittstelle aufbauen lässt. Weil RS232 laut Standard mit Pegeln im Bereich von +12 V und - 12 V arbeitet, muss den beiden seriellen Signalen des Raspberry Pi-Boards noch ein geeigneter Transceiver, wie beispielsweise der gebräuchliche Typ MAX232, nachgeschaltet werden.

Das einzige Massesignal befindet sich laut den Angaben der *Raspberry Pi Foundation* am Pin 6. Es werden zudem sechs Anschlüsse mit der Bezeichnung DNT ausgewiesen, was für *Do Not Connect* steht. Demnach darf an diesen Pins, die in der Tabelle 3.3 mit dem Zeichen »-« versehen sind, nichts angeschlossen werden. Beim Vergleich mit dem Schaltplan (Abbildung 3.40) ist allerdings zu erkennen, dass an den DNT-Pins entweder Ground (Pins 9, 14, 20, 25) oder + 5 V (Pin 4) oder +3,3 V

(Pin 17) anliegen, so dass nichts dagegen spricht, diese Anschlüsse auch entsprechend einzusetzen. Diese »Unstimmigkeiten« ergeben sich, weil mit der Board-Revision 2 der Raspberry Pi-Hardware einige Pins zusätzliche Funktionen und andere Bezeichnungen erhalten haben, was in den beiden obigen Tabellen mit berücksichtigt ist.

3.12.1 Weitere GPIO-Ports der Revision 2

Neben der 26-poligen Leiste, bei der einige Pins in Abhängigkeit von der Raspberry Pi -Board-Version eine unterschiedliche Bedeutung und Funktion aufweisen, ist ab der Version 2 ein zweiter GPIO-Port vorhanden, der mit P5 auf der Platinenrückseite (Abbildung 3.41) gekennzeichnet ist. Er ist nicht mit einer Leiste herausgeführt, sondern soll für Erweiterungen durch Hardware-Hersteller reserviert sein. Gleichwohl kann der zweite GPIO-Port – nach Bestückung eines 8-poligen Pfostensteckers – genauso wie die anderen GPIO-Pins verwendet werden.

Abbildung 3.41: Die Raspberry Pi-Platine von der Rückseite aus, wo oben die 26 Kontakte des Expansion-Header und unten der zweite GPIO-Port (P5) zu erkennen ist.

Bei der Signalzuordnung an P5 (Tabelle 3.5) ist zu beachten, dass hier der Pin 1 nicht (wie bei der 26-poligen Leiste) links, sondern rechts lokalisiert ist, die Pins demnach von rechts nach links gezählt werden.

Tabelle 3.5: Die Kontaktbelegung des zweiten GPIO-Ports

Pin	Signal	Pin	Signal
2	3,3 V	1	5 V
4	GPIO 29/I²C0_SCL/PCM_FS	3	GPIO 28/I²C0_SDA/PCM_CLK
6	GPIO 31/PCM_Dout	5	GPIO 30/PCM_Din
8	Masse (GND)	7	Masse (GND)

Des Weiteren sind am Anschluss des *Camera Serial Interface* (CSI) vier weitere GPIO-Pins (GPIO 2, 3, 5, 27) zu finden, die ebenfalls für eigene Schaltungen eingesetzt

werden können, genauso wie ein durch den fehlenden Ethernet-Chip beim Modell A freigewordener Anschluss (GPIO 6).

3.12.2 GPIO-Port der +Modelle

Die 26-polige Steckerleiste wird beim Raspberry Pi-Model B+ (siehe Kapitel 3.11) erstmalig auf insgesamt 40 Anschlüsse erweitert und die P5-Verbindungen entfallen, was für einige Erweiterungskarten (z.B. Wolfson Card) ein Neudesign erfordert, weil diese die PCM- oder auch I²C-Signale am P5-Connector verwenden. Mit den 14 zusätzlichen Kontakten werden neun weitere GPIO-Signale, drei Massekontakte sowie zwei spezielle Identifizierungssignale (ID_SD und ID_SC: I2C ID EEPROM) hinzugefügt. Die bisherigen 26 Kontakte bleiben elektrisch kompatibel. Die für die Audiokarten notwendigen PCM-Signale (I²S für Chip-zu-Chip-Audiokommunikation) sowie ein zusätzlicher SPI-Port können nunmehr alternativ zu den neuen GPIO-Signalen eingesetzt werden, wie es in der Abbildung 3.42 gezeigt ist.

Abbildung 3.42: Der 40-polige GPIO-Port

Mithilfe der I²C-Identifizierungssignale können aufgesteckte Erweiterungsmodule, die über ein EEPROM mit entsprechendem Inhalt verfügen, identifiziert werden, so dass ein automatischer Setup der GPIO-Signale mit dem Laden der passenden Linux-Treiber ermöglicht wird. Dieser Mechanismus ist Bestandteil der HAT-Spezifikation, die mit dem Modell B+ vorgestellt wurde. HAT steht für *Hardware Attached on Top* und definiert aufsteckbare Erweiterungsmodule, die bestimmte Voraussetzungen erfüllen müssen, damit sie sich als HAT-kompatibel bezeichnen dürfen. Dazu gehört, dass das Board eine bestimmte Größe von 65 x 55 mm mit passenden Löchern zum Befestigen und Schlitze für die Fachbandkabel (CSI, DSI) aufweisen muss. Es passt demnach exakt auf das Modell A+.

Abbildung 3.43: Prototyplayout für eine HAT-Platine

Außerdem dürfen lediglich die Signale am 40-poligen Connector zum Einsatz kommen. Die Daten im EEPROM müssen definierte Informationen (Hersteller, Gerätetyp, GPIO-Konfiguration) in einem Device Tree-Format aufweisen. Das gleiche Prinzip zur Geräteidentifizierung mit speziellen Ressource-Daten in einem EEPROM wird seit langem bei PCI- und PCIe-Karten oder auch den USB-Einheiten eingesetzt.

Das endgültige Format, in dem die EEPROM-Daten aufgebaut sein sollen, ist zum Zeitpunkt der Bucherstellung noch nicht komplett definiert. In der Abbildung 3.44 ist deshalb eine vorläufige Version mit zwei Einträgen zur Veranschaulichung gezeigt. Ob und unter welchen Bedingungen ein Aufeinanderstapeln mehrerer HAT-Boards übereinander möglich ist, wie es mit den Shields bei der Arduino-Plattform üblich ist, ist ebenfalls noch nicht endgültig festgeschrieben.

EEPROM Structure

```
HEADER  <- EEPROM header (Required)
ATOM1   <- Vendor info atom (Required)
ATOM2   <- GPIO map atom (Required)
ATOM3   <- DT blob atom (Required for compliance with the HAT specification)
...
ATOMn
```

Vendor Info (0x0001)

Bytes	Field	
16	uuid	UUID (unique for every single board ever made)
2	pid	product ID
2	pver	product version
1	vslen	vendor string length (bytes)
1	pslen	product string length (bytes)
X	vstr	ASCII vendor string e.g. "ACME Technology Company"
Y	pstr	ASCII product string e.g. "Special Sensor Board"

EEPROM Header Structure

Bytes	Field	
4	signature	signature: 0x52, 0x2D, 0x50, 0x69 ("R-Pi" in ASCII)
1	version	EEPROM data format version (0x00 reserved, 0x01 = first version)
1	reserved	set to 0
2	numatoms	total atoms in EEPROM
4	eeplen	total length in bytes of all eeprom data (including this header)

Atom Structure

Bytes	Field	
2	type	atom type
2	count	incrementing atom count
4	dlen	length in bytes of data+CRC
N	data	N bytes, N = dlen-2
2	crc16	CRC-16-CCITT of entire atom (type, count, dlen, data)

Atom Types

```
0x0000 = invalid
0x0001 = vendor info
0x0002 = GPIO map
0x0003 = Linux device tree blob
0x0004 = manufacturer custom data
0x0005-0xfffe = reserved for future use
0xffff = invalid
```

GPIO Map (0x0002)

```
Bytes   Field
1       bank_drive  bank drive strength/slew/hysteresis, BCM2835 can only set per bank, not per IO
          Bits in byte:
          [1:0] drive       0=leave at default, 1=8mA(low*3mA), 0.15reserved
          [5:4] slew        0=leave at default, 1=slew rate limiting, 2=no slew limiting, 3=reserved
          [7:6] hysteresis  0=leave at default, 1=hysteresis disabled, 2=hysteresis enabled, 3=reserved
1       power
          [1:0] back_power  0=board does not back power Pi
                            1=board back powers and can supply up to 1.3A to the Pi
                            2=board back powers and can supply up to 2A to the Pi
                            3=reserved
                            If back_power=2 high current USB mode is automatically enabled.
          [7:2] reserved    set to 0
28      1 byte per IO pin
          Bits in each byte:
          [2:0] func_sel    GPIO function as per FSEL GPIO register field in BCM2835 datasheet
          [4:3] reserved    set to 0
          [6:5] pulltype    0=leave at default setting,  1=pullup, 2=pulldown, 3=no pull
          [  7] is_used     1=board uses this pin, 0=not connected and therefore not used
```

Abbildung 3.44: Prinzipielle Struktur der EEPROM-Daten mit zwei Beispielen

3.12.3 GPIO-Port des Compute Module

Auf dem I/O-Board für das Compute Module befinden sich zwei zweireihige Pfostenleisten mit jeweils 2 x 60 Polen. Hier sind sämtliche I/O-Anschlüsse des BCM2835, die nicht anderweitig etwa für die Display-Anschlüsse und den HDMI-Ausgang benötigt werden, angelegt. Somit ist es einfach möglich, an bestimmten Leitungen Messungen vorzunehmen und hier Peripherie zum Test anzuschließen, bevor das I/O-Board durch eine eigene Lösung ersetzt wird, die dann das Compute Module aufnimmt.

Insgesamt bietet dieser GPIO-Port 46 einzelne I/O-Ports, die zwei Bänken zugeordnet sind und die sich auf beide Pfostenleisten verteilen sowie die Versorgungsspannungen (5 V, 3.3 V, 1,8 V) und einige der speziellen Signale (CAM/2x I²C, TVDAC). Über zwei Jumper (J3) lassen sich die GPIO-Spannungen (VG0, VG1) der zwei Bänke bestimmen, die entweder 3,3, V oder 1,8 V betragen können und auf mehrere Pins der Leisten geführt werden. Dabei ist es wichtig, dass hier auf jeden Fall Jumper gesteckt sind, andernfalls kann der BCM2835 zerstört werden.

Abbildung 3.45: Die Belegung und das Layout der beiden GPIO-Leisten

4 Konfigurierung und Optimierung

Nachdem der *Schnellstart* im Kapitel 1 zu einem funktionierenden Raspberry Pi geführt hat und die wesentlichen Dinge für den Umgang mit dem Linux-Betriebssystem sowie der grundlegenden Software im Kapitel 2 behandelt wurden, geht es in diesem Kapitel um typische Konfigurationsarbeiten und um die Optimierung des Raspberry Pi.

4.1 Betriebssysteme

Für den Raspberry Pi gibt es verschiedene Betriebssysteme, wobei sich prinzipiell alle Linux-Derivate (ab Linux Kernel 2.6) und auch andere, für ARM-Prozessoren bestimmte Systeme einsetzen lassen, die jedoch die speziellen Funktionen des Broadcom BCM2835 nicht nutzen können, was nur mit den explizit für den Raspberry Pi ausgelegten Betriebssystemen möglich ist. Die typischen Linux-Desktop-Systeme lassen sich nicht auf einem Raspberry Pi installieren, genauso wenig wie Betriebssysteme für die x86-Architektur (DOS, Windows) von Intel. Die wichtigsten für den Raspberry Pi vorgesehenen Betriebssysteme sind im Folgenden kurz erläutert. Sie werden in Form eines Image zur Verfügung gestellt, welches als Download von verschiedenen Internet-Seiten bezogen werden kann. Die erste Anlaufstelle hierfür ist die Seite der *Raspberry Pi Foundation*:

`http://www.raspberrypi.org/downloads`

Die für den Raspberry Pi angepassten Betriebssysteme berücksichtigen die gegenüber anderen Plattformen (PC, Notebooks) begrenzten Ressourcen, denn es macht keinen Sinn, hier zu versuchen, die üblichen Browser oder Office-Pakete oder etwa Bildverarbeitungsprogramme mit dem Komfort eines Desktop-Systems einsetzen zu wollen.

Debian Squezze: Basiert wie Ubuntu oder auch Knoppix auf einem Debian GNU-Linux (Kernel 2.6), welches quasi den Vorläufer (ohne Spezialisierung) von Wheezy darstellt. Es richtet sich demgegenüber eher an Linux-erfahrenere Anwender. Wie alle Debian-Versionen handelt es sich um eine reine Non-Profit-Distribution, ohne dass hier wie bei Fedora (Firma Red Hat) oder SuSE (Firma Novell) eine Firma dahintersteht.

Debian Wheezy, Raspbian: Stabile, speziell für den Broadcom BCM 2835 angepasste Debian-Linux-Version, die auch die *Floating Point Unit* (Gleitkomma/Fließkommazahlenberechnung) und weitere spezielle Einheiten des Broadcom BCM2835 unterstützt. Raspbian liefert eine Vielzahl von angepassten Software-Paketen und ist zurzeit die Standard-Version für den Raspberry Pi.

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1_5, © Springer Fachmedien Wiesbaden 2015

Debian Wheezy Soft Float: Im Prinzip die gleiche Version wie Raspbian, allerdings wird hier bewusst auf die Unterstützung der hardware-basierten *Floating Point Unit* verzichtet, weil einige Anwendungen (z.B. Oracle JAVA) damit nicht lauffähig sind.

Fedora/Pidora Linux: Diese Linux-Version sollte ursprünglich die für den Raspberry Pi (Fedora 18 Remix) bevorzugte Version sein, die ebenfalls spezielle Treiber (Video Core) beinhaltet. Das Image weist eine beträchtliche Größe von ca. 3 GByte auf, gleichwohl sind nur recht wenige Applikationen dabei, die die begrenzten Ressourcen des Raspberry Pi berücksichtigen. Als Desktop ist Xfce (Version 4.10) standardmäßig vorgesehen, welches auf dem Raspberry Pi verhältnismäßig langsam arbeitet, was für das gesamte System gilt. Die Raspberry Pi Foundation entschied sich (wohl aus diesen Gründen) stattdessen für Debian (Raspbian). Fedora ist für verschiedene ARM-Architekturen (Kapitel 3.2) verfügbar und steht mittlerweile in einer für den Raspberry Pi angepassten Version als Pidora Linux in einer stabilen Version zur Verfügung.

Arch Linux ARM: Eine rudimentäre, schnelle Linux-Version für den Raspberry Pi, die in ca. 10s bootet und als *Basis-Betriebssystem für fortgeschrittene Anwender* propagiert wird. Sie unterstützt die *Floating Point Unit*, bietet jedoch standardmäßig keine grafische Oberfläche. Es wird nur eine geringe Anzahl von Software-Paketen geliefert, die sich mit dem dazugehörigen Paketmanager installieren lassen. Für Einsteiger ist diese Linux-Version nicht geeignet. Sie wird vorzugsweise für spezielle Anwendungen genutzt, die mit eigener Software ohne Anzeige arbeiten sollen.

RISC OS: Eine sehr ursprüngliche Implementierung des *Operating Systems* (OS) für einen *Reduced Instruction Set Computer*. RISC gilt als gegenteiliger Ansatz zu einem Computer nach CISC (Complex Instruction Set Computer), wofür der PC mit einem Intel-Prozessor das bekannteste Beispiel ist. Das RISC-System wurde in den achtziger Jahren in Cambridge entwickelt, für die Acorn-Computer (siehe Einführung und Kapitel 3.2) eingesetzt und wird nunmehr auch für den Raspberry Pi zur Verfügung gestellt. Im Vergleich mit den aktuellen Linux-Systemen wirkt RISC OS schlicht und etwas nostalgisch. Außerdem gibt es hierfür wenig Open-Source-Software. Anderseits arbeitet das System sehr stabil.

Es gibt noch einige weitere Linux-Systeme wie SliTaz4 oder Gentoo, die für den Raspberry Pi geeignet sind. Sie setzen allerdings noch weit mehr Linux-Kenntnisse als Arch Linux voraus, zumal Gentoo selbst zu kompilieren ist, wodurch – mit relativ viel Arbeit – ein schlankes und individuelles System hergestellt wird. SliTaz4 ist demgegenüber auf den Einsatz als schneller Web Server optimiert. Die Linux-Systeme realisieren grundsätzlich die Datei- und Verzeichnisstruktur und unterstützen die typischen Befehle, wie sie im Kapitel 2 erläutert sind. Andere Betriebssysteme wie etwa Android oder FreeBSD werden in speziellen Versionen

für den Raspberry Pi angepasst. Es werden wahrscheinlich noch weitere für besondere Einsatzgebiete folgen, wobei insbesondere die verschiedenen Mediacenter-Distributionen (siehe Kapitel 4.5), die auch Bestandteil von NOOBS (Kapitel 1.4) sind, ein große Verbreitung gefunden haben.

```
Arch Linux 3.6.11-6-ARCH+ (tty1)

alarmpi login: [ OK ] Started Netcfg networking service for profile ethernet-etho
[ OK ] Reached target Network
       Starting OpenNTP Daemon…
[ OK ] Started OpenNTP Daemon
[ OK ] Reached tasrget Multi-User.
[ OK ] Reached target Graphical Interface.

Arch Linux 3.6.11-6-ARCH+ (tty1)

alarmpi login: root
Password:

Last login: Fr Apr 12 16:27:22 BST 2013 on tty1
Last login: Fr Apr 12 16:39:51 on tty1

[root@alarmpi ~]# ls -l
total 0

[root@alarmpi ~]# df -h
Filesystem          Size      Used      Avail     Used%     Mounted on
rootfs              1.7G      458M      1.2G        29%     /
/dev/root           1.7G      458M      1.2G        29%     /
devtmpfs             83M         0       83M         0%     /dev
tmpfs               232M         0      232M         0%     /dev/shm
tmpfs               232M      268K      231M         1%     /run
tmpfs               232M         0      232M         0%     /sys/fs/cgroup
tmpfs               232M         0      232M         0%     /tmp
/dev/mmcblk0p1       90M       24M       67M        27%     /boot

[root@alarmpi ~]#
```

Abbildung 4.1: Bei Arch Linux ist root mit dem Password root (zunächst) der einzige Anwender. Hier werden die Linux-Befehle für die Inhaltsanzeige des aktuellen Verzeichnisinhaltes (ls) und Disk Free (df) ausgeführt. Dabei ist standardmäßig der englisch/amerikanische Zeichensatz aktiv.

Selbstverständlich sind eigenen Experimenten mit den verschiedenen Betriebssystemen und den Software-Paketen, die auch nicht explizit für den Raspberry Pi vorgesehen sind, kaum Grenzen gesetzt, denn eine Beschädigung der Hardware ist dadurch nicht möglich. Falls nichts mehr funktionieren sollte, wird einfach die SD-Karte mit der funktionierenden Raspbian-Version wieder eingesetzt, die momentan als das Standardbetriebssystem für den Raspberry Pi gilt.

4.2 Systeminstallation

Die Installation des Betriebssystems erfolgt auf einer SD Card (Kapitel 1.2 und 3.3.1), auf die ein Image geschrieben wird, welches das jeweilige Betriebssystem

enthält. Wie oben erwähnt, werden die verschiedenen Systeme als kostenloser Download im Internet zur Verfügung gestellt. Um die Daten des Image auf die SD Card schreiben zu können, muss ein Computer mit einem SD Card-Slot zur Verfügung stehen sowie ein *Entpacker*, weil das Image üblicherweise als komprimierte ZIP-Datei geliefert wird.

Abbildung 4.2: Entpacken des Arch Linux ZIP-Files mit 7-Zip

Außerdem wird ein spezielles Image Writer-Programm benötigt, wofür verschiedene Programme für die unterschiedlichen Computer-Betriebssysteme (Linux, Windows, MacOS) existieren, die wie ein ZIP-Packer/Entpacker (beispielsweise WinRAR, 7-Zip) ebenfalls kostenlos im Internet verfügbar sind. Ein bevorzugter *Image Writer* für Windows ist der *Win32 Disk Imager*, der einfach und funktionell aufgebaut ist. Der Einsatz des Win32DiskImage-Programms ist problemlos; es ist lediglich darauf zu achten, dass unter *Device* der richtige Kartenslot – tatsächlich die gewünschte SD Card – ausgewählt wird, falls mit dem PC mehrere SD-Karten oder vergleichbare Speichermedien wie etwa USB-Sticks verbunden sind. Eine vorherige Kontrolle über *Arbeitsplatz* oder *Computer* ist bei einem Windows-PC deshalb ratsam, denn nach dem Start des Win32DiskImage-Programms und der Selektierung von *Write* wird die SD Card unwiederbringlich überschrieben.

Abbildung 4.3: Das Image für Arch Linux wird mit dem Win32 Disk Imager auf die SD-Karte geschrieben.

Mitunter ist es nützlich, eine Kopie der SD-Karte anzulegen, etwa weil das System auf dem Raspberry Pi wunschgemäß konfiguriert und mittlerweile mit der notwendigen Software versehen ist. Eine direkte Kopie der Karte ist unter Windows aufgrund des Linux-Dateisystems nicht möglich, gleichwohl kann auch hierfür das Win32DiskImage-Programm eingesetzt werden. Die SD Card ist dementsprechend zu lesen (Read) und in ein File, wofür ein Name und ein Verzeichnis anzugeben ist, auf die Festplatte zu schreiben.

4.3 Audio aktivieren und einsetzen

Der Raspberry Pi kann ein übliches Audiosignal in Stereo ausgeben, wie etwa eine MP3- oder eine WAV-Datei und prinzipiell jeden Typ, für den es einen entsprechenden Linux-Codec gibt. Der Audioausgang ist mit einer 3,5 mm Klinkenbuchse aufgebaut, in die ein Klinkenstecker passt. Typische Ohr- oder Kopfhörer können damit für die Audioausgabe verwendet werden. Falls das Signal an eine Stereoanlage oder ein anderes HiFi-Gerät geführt werden soll, ist ein Cinch-Adapter notwendig, der das Signal vom Klinkenstecker auf zwei Cinch-Stecker (linker und rechter Kanal) umsetzt. Bei den +Modellen (Kapitel 3.1) ist eine vierpolige Buchse auf der Platine vorgesehen, die auch das Video-Signal führt (Abbildung 3.38), so dass hierfür ein anderer Adapter notwendig ist.

Am einfachsten ist es – zumindest für einen ersten Test –, wenn der verwendete Monitor über einen Audioeingang verfügt, der üblicherweise ebenfalls eine 3,5 mm Klinkenbuchse besitzt, so dass ein einfaches Audiokabel, wie es sich oftmals im Lieferumfang des Monitors befindet, verwendet werden kann.

Abbildung 4.4: Das System ist mit dem Audioanschluss und einer USB-Maus verbunden.

Das Audiosignal wird außerdem über die HDMI-Verbindung geführt, so dass in diesem Fall keine separate Audioverbindung notwendig ist. Die Audioübertragung funktioniert über HDMI aber nur dann, wenn hier kein Adapter wie etwa auf DVI eingesetzt wird. Es muss demnach eine direkte HDMI-Verbindung zwischen dem Raspberry Pi und dem Monitor bestehen.

Die Audiounterstützung ist unter Linux etwas undurchsichtig und sollte zunächst auf der Kommandozeilenebene ausprobiert werden, bevor versucht wird, Audiosignale mithilfe des LXDE-Desktop auszugeben. Standardmäßig werden mit Raspbian einige Beispiele mit unterstützenden Bibliotheken (Interlock Library mit ilclient) geliefert, die verschiedene Grafik- und Video-Tests (OpenGL, OpenMAX) sowie auch einen Audiotest erlauben. Zu finden sind sie unter */opt/vc/src/hello_pi/libs/ilclient*.

Das für Audio verantwortliche Kernel-Modul ist bei den aktuellen Raspbian-Versionen standardmäßig installiert. Falls dies nicht der Fall sein sollte, kann dies mit der folgenden Zeile erledigt werden:

```
pi@raspberrypi ~ $ sudo modprobe snd_bcm2835
```

Mit den nächsten Zeilen wird mithilfe von *make* ein Testprogramm (hello_audio.bin) kompiliert, welches einen einfachen »Heulsound« über den Audioausgang ausgibt, was als ein erster Test für eine erfolgreiche Audioausgabe nützlich ist.

```
pi@raspberrypi ~ $ cd /opt/vc/src/hello_pi/libs/ilclient
pi@raspberrypi /opt/vc/src/hello_pi/libs/ilclient $  make
pi@raspberrypi /opt/vc/src/hello_pi/libs/ilclient $  cd
../../hello_audio
pi@raspberrypi /opt/vc/src/hello_pi/hello_audio $  make
pi@raspberrypi /opt/vc/src/hello_pi/hello_audio $  ./hello_audio.bin
```

Nach der Kompilierung liegt die ausführbare Applikation im Unterverzeichnis hello_audio und kann fortan von dort aus auch direkt ausführt werden, wie es in der Abbildung 4.5 gezeigt ist.

```
pi@raspberrypi ~ $ cd /opt/vc/src/hello_pi/hello_audio
pi@raspberrypi /opt/vc/src/hello_pi/hello_audio $ ./hello_audio.bin
Outputting audio to analogue
pi@raspberrypi /opt/vc/src/hello_pi/hello_audio $
```

Abbildung 4.5: Test der Audioausgabe mit hello_audio.bin

Die Audioausgabe findet standardmäßig über HDMI statt, wenn hierfür eine durchgehende Verbindung (kein DVI-Adapter) existiert, andernfalls über den separaten Audioausgang. Um die Audioausgabe über HMDI zu erzwingen, ist die folgende Zeile notwendig, bei der einfach nur eine »1« angefügt wird:

```
pi@raspberrypi /opt/vc/src/hello_pi/hello_audio ./hello_audio.bin 1
```

Alternativ ist eine Umschaltung der Audioausgabe bei den neueren raspi-config-Versionen (Kapitel 1.3.18) im Menü möglich. Für die Kommandozeile gibt es noch weitere Optionen, wie etwa für die Lautstärke, was mit *amixer* bestimmt werden kann. Wie zu (fast) jeder Linux-Option gibt es auch zu *amixer* eine Erläuterung (man amixer). Der Aufruf von *amixer*, der zum *ALSA soundcard driver-package* gehört, hat eine Anzeige der aktuellen Einstellungen zur Folge. *amixer contents* zeigt die möglichen Optionen, *amixer controls* weist die drei möglichen Einstellungsmöglichkeiten aus, die per *numid* ausgewählt werden können, wie etwa in der folgenden Zeile, mit der der Wiedergabekanal (Route) anhand von numid=3 mit dem Wert »0« auf automatisch und die Wiedergabelautstärke per numid=1 (Volume) auf Maximum mit »400« geschaltet wird.

```
pi@raspberrypi ~ $ amixer cset numid=3 0
pi@raspberrypi ~ $ amixer cset numid=1 400
```

Die Switch-Option spielt beim Raspberry Pi keine besondere Rolle, denn es kann aufgrund der einfachen Audioschaltung nicht zwischen mehreren Kanälen umgeschaltet werden. Die Audioausgabe ist lediglich abschaltbar.

Obwohl in der Anzeige laut Abbildung 4.6 *Playback channels: Mono* angegeben ist, ist prinzipiell eine Stereoausgabe möglich, lediglich der ALSA-Kommandozeilenmixer (amixer) regelt beide Kanäle gleichzeitig, er macht demnach keinen Unterscheid zwischen linkem und rechtem Kanal (Balance), was missverständlicher Weise als »Mono« ausgewiesen wird.

```
pi@raspberrypi ~ $ amixer
Simple mixer control 'PCM',0
  Capabilities: pvolume pvolume-joined pswitch pswitch-joined penum
  Playback channels: Mono
  Limits: Playback -10239 - 400
  Mono: Playback 400 [100%] [4.00dB] [on]
pi@raspberrypi ~ $ amixer controls
numid=3,iface=MIXER,name='PCM Playback Route'
numid=2,iface=MIXER,name='PCM Playback Switch'
numid=1,iface=MIXER,name='PCM Playback Volume'
pi@raspberrypi ~ $ amixer contents
numid=3,iface=MIXER,name='PCM Playback Route'
  ; type=INTEGER,access=rw------,values=1,min=0,max=2,step=0
  : values=1
numid=2,iface=MIXER,name='PCM Playback Switch'
  ; type=BOOLEAN,access=rw------,values=1
  : values=on
numid=1,iface=MIXER,name='PCM Playback Volume'
  ; type=INTEGER,access=rw---R--,values=1,min=-10239,max=400,step=0
  : values=400
  | dBscale-min=-102.39dB,step=0.01dB,mute=1
pi@raspberrypi ~ $ amixer cset numid=3 0
numid=3,iface=MIXER,name='PCM Playback Route'
  ; type=INTEGER,access=rw------,values=1,min=0,max=2,step=0
  : values=0
pi@raspberrypi ~ $ amixer cset numid=1 400
numid=1,iface=MIXER,name='PCM Playback Volume'
  ; type=INTEGER,access=rw---R--,values=1,min=-10239,max=400,step=0
  : values=400
  | dBscale-min=-102.39dB,step=0.01dB,mute=1
pi@raspberrypi ~ $
```

Abbildung 4.6: Anzeige und konfigurieren (cset) der Audiooptionen

Die getätigten Audioeinstellungen stehen nach dem nächsten Boot wieder auf ihren vorgegebenen Werten. Die Eingaben (laut Abbildung 4.6) werden nicht automatisch gespeichert. Dies kann jedoch manuell festgelegt werden, und zwar in der

Datei *rc.local* unter */etc*, wenn hier vor der letzten Zeile (exit 0) beispielsweise die beiden Einträge (wie oben) mit

```
amixer cset numid=3 0
amixer cset numid=1 400
```

vorgenommen werden, was direkt mit einem üblichen Editor (z.B. nano) oder auch mit dem Midnight Commander per F4 durchgeführt werden kann, der sich des Standardeditors bedient.

Wie gezeigt, ist die Audiounterstützung unter Linux tatsächlich etwas gewöhnungsbedürftig und macht auf der Kommandozeile eigentlich keinen Sinn, auch wenn es hierfür verschiedene Player gibt. Player, Mixer und Soundeditoren sind meist logischer und einfacher per Desktop (LXDE) zu handhaben. Leider verfügt LXDE nicht über ein zentrales Kontrollzentrum, weshalb die Einstellungen mit unterschiedlichen Programmen durchzuführen sind, die über *Start - Einstellungen* auftauchen, wo man zwar auf Bildschirm- sowie Tastatur- und Mauseinstellungen trifft, jedoch keinerlei Audiofunktionen findet.

Abbildung 4.7: Durch einen Klick auf die Taskleiste stehen die Paneleinstellungen zur Verfügung, es kann auch ein Lautstärkeregler hinzugefügt werden.

Um im einfachsten Fall dem LXDE-Desktop einen Lautstärkeregler hinzufügen, wird in der Taskleiste (unten) die rechte Maustaste betätigt, woraufhin einige Panel-Optionen erscheinen, wovon hier die Option *Paneleinträge hinzufügen/entfernen* von Bedeutung ist. Wenn der ALSA-Soundtreiber (Advanced Linux Sound Architecture) geladen ist, findet man hier in der Liste der *Gegenwärtig geladenen Erweiterungen* auch den Punkt *Lautstärkeregler*, der zu selektieren und hinzuzufügen ist, so dass er danach unten in der LXDE-Taskleiste auftaucht. Falls kein Lautstärkeregler

erscheint, kann auch der ALSA-Treiber nicht vorhanden sein, und die zuvor beschriebenen Audiofunktionen können nicht funktioniert haben. ALSA wird wie üblich mit sudo *apt-get install alsa-utils* installiert, was spätestens jetzt nachzuholen wäre.

4.4 Videoplayer und Lizenzen

Ein bekannter Audio- und Videoplayer, der automatisch zahlreiche Formate unterstützt, ist der *VLC Media Player*, der für unterschiedliche Plattformen zur Verfügung steht und auch beim Raspberry Pi installiert werden kann.

```
pi@raspberrypi ~ $ sudo apt-get install vlc
```

Nach der Installation des Players ist er mit den bekannten Optionen unter *Unterhaltungsmedien* zu finden, so dass sich hiermit auch die Audiooptionen komfortabler nutzen lassen. Als Verzeichnis für die Mediadaten ist `/home/pi/media` vorgesehen, die sich von hier aus laden lassen.

Abbildung 4.8: Der VLC Media Player funktioniert auch mit dem Raspberry Pi.

Während verschiedene Audioformate dem Player bzw. dem Raspberry Pi keinerlei Probleme bereiten, wird das Abspielen von Videos in den üblichen Formaten vielfach nicht vernünftig funktionieren, denn hierfür ist die Leistung des Raspberry Pi

zu schwach. Lediglich der Video-Codec H.264 (MPEG-4/AVC) erfährt vom Video-Core des BCM 2835 auch eine Hardware-Beschleunigung.

Für andere Video-Formate (VC1, MPEG2) können gegen eine geringe Gebühr im *Raspberry Pi Store* direkt bei der *Raspberry Pi Foundation* entsprechende Lizenzen (`http://www.raspberrypi.com/license-keys/`) erworben werden, die jeweils an einen bestimmten Raspberry Pi zu binden sind, d.h., eine Lizenz ist jeweils nur für einen einzigen Raspberry Pi vorgesehen. Um eine Lizenz zu erhalten, ist deshalb die jeweilige Produkt/Seriennummer (Serial) an die Foundation zu übermitteln. Diese Nummer steht in cpuinfo, was mit `cat /proc/cpuinfo` angezeigt werden kann.

```
pi@raspberrypi ~ $ cat /proc/cpuinfo
Processor       : ARMv6-compatible processor rev 7 (v6l)
BogoMIPS        : 697.95
Features        : swp half thumb fastmult vfp edsp java tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xb76
CPU revision    : 7

Hardware        : BCM2708
Revision        : 000e
Serial          : 00000000ae048a77
pi@raspberrypi ~ $
```

Abbildung 4.9: Anzeige der genauen Daten der jeweils eingesetzten CPU, wobei sich die für die Lizenzierung notwendige Nummer (Serial) in der letzten Zeile befindet.

Dieses Verfahren ist deswegen erforderlich, weil für MPEG- und andere Video-Formate kostenpflichtige Lizenzen notwendig sind, was sich mit der Open Source-Philosophie des Raspberry Pi kaum anders lösen lässt, denn die Foundation kann diese Kosten nicht übernehmen.

Nachdem die Lizenznummer für den jeweiligen CODEC per Email eingetroffen ist, ist die Nummer in der Datei *config.txt* in der entsprechenden Decode-Zeile (# decode_MPG2=0×00000000) einzutragen und das Kommentarzeichen zu entfernen, was direkt mit einem PC erledigt werden kann, weil sich diese Datei in der FAT32-Partition der SD Card befindet. Je nach verwendetem Betriebssystem und Konfiguration ist der Aufbau der Datei config.txt unterschiedlich, so dass die jeweilige Beschreibung für die Eintragung der Lizenznummer zu beachten ist.

4.5 Mediacenter

Im vorherigen Kapitel sind die Audio-Optionen, die mit Raspbian und dem LXDE-Desktop verfügbar sind, erläutert. Falls nur gelegentlich Audio- und Video-Dateien auf der Basis eines universell einzusetzenden Betriebssystems wiederge-

geben werden sollen, wird diese Lösung meistens ausreichen. Eine Multimedialösung, wie man sie von anderen Systemen her kennt, ist dies allerdings nicht. Deshalb wird in diesem Kapitel die Software für den Aufbau eines Media-Centers etwas näher betrachtet.

XBMC ist eine Software, die ursprünglich als *XBox Media Center* bezeichnet wurde. Dabei handelt es sich (zunächst) nicht um ein Betriebssystem wie bei den Systemen unter Kapitel 4.1, sondern es ist eine Mediacenter-Software für die Wiedergabe von Musik, Bildern und Videos in den unterschiedlichen Formaten sowie von Audio- und Video (-Streams) aus dem Internet. XBMC ist eine freie Software, die für unterschiedliche Betriebssysteme wie Windows, Mac OS, iOS, Android oder Linux verfügbar ist und auch in einer speziellen Version für den Raspberry Pi (Raspbmc) existiert.

Abbildung 4.10: Der Raspbmc-Installer für Windows schreibt die Daten aus dem Image auf die SD Card (Volume H).

Raspbmc ist im Gegensatz zu den anderen Versionen eine komplette Linux-Distribution, auf der ausschließlich XBMC läuft. Mithilfe eines Installationsprogramms (`http://www.raspbmc.com/download/`) für das jeweils verwendete Betriebssystem (Windows, Mac OS, Linux) kann die Raspbmc-Version geladen und auf eine SD-Karte geschrieben werden (Abbildung 4.10). Diese wird daraufhin zum Boot des Raspberry Pi verwendet, wobei zunächst ein »Minimal-Linux« geladen wird, bevor die eigentliche Raspbmc-Installation mit einer Neupartitionierung der SD Card, der Formatierung und dem Laden des Kernels sowie der Bibliotheken stattfindet, was ca. 20 Minuten dauert (Meldung währenddessen: Grab a Coffee). Danach folgt ein Neustart des Raspberry Pi mit der Installation weiterer (neuerer) Raspbmc-Software aus dem Internet. Der gesamte Vorhang läuft dabei automatisch ab, ohne dass eine Angabe des Benutzers notwendig ist – eine funktionierende Internetverbindung des Raspberry Pi vorausgesetzt.

Generell unterstützt XBMC eine Vielzahl unterschiedlicher Musik- und Videoformate, die bei Bedarf auch über Add-Ons hinzugefügt werden können. Mit Rücksicht auf die Limitierungen des Raspberry Pi sollte vorzugsweise der Video-Codec H.264 (MPEG4) genutzt werden, denn dieser erfährt explizit eine entsprechende Hardware-Beschleunigung durch den Video-Core des BCM2835.

Abbildung 4.11: Vornehmen der TV-Einstellungen

XBMC bietet neben diesen Player-Funktionen unter *Musik* und *Videos*, die durch Informationen aus dem Internet automatisch ergänzt werden können, die Anzeige von Bildern (Dateilisten, Diashow). Außerdem ist eine Anzeige für das aktuelle Wetter aus der Quelle *Weather Underground* voreingestellt, die auch Prognosen und eine Satellitenwetterkarte zur Verfügung stellt.

Live TV ist eine als Add-On mitgelieferte Option, die eine Fernsehwiedergabe bietet, und zwar entweder, indem ein DVB-T-Empfänger als USB-Stick an den Raspberry Pi angeschlossen wird oder es kommt ein Client für einen TV-Server (Streaming Server) zum Einsatz, der demnach in der Netzwerkumgebung vorhanden sein muss und die Daten per LAN oder WLAN an den PVR-Client von XBMC liefern kann, was für *Live TV* als die bevorzugte Lösung gilt.

Netzwerkverbindungen (FTP, SSH, NFS u.a.) werden standardmäßig unterstützt, Wechseldatenträger (z.B. USB-Festplatten, USB-Sticks) automatisch erkannt und als Quelle im System eingebunden. Das Mediacenter lässt sich auch per Webschnittstelle fernsteuern, was am bequemsten mit dem Smartphone oder dem Tablet (Android, iOS) über WLAN möglich ist. Für Android empfiehlt sich als XBMC Remote-Lösung das Programm YATSE.

Abbildung 4.12: Android-Smartphone als XBMC-Fernbedienung

Neben Raspbmc existieren weitere Mediacenter-Lösungen, die für den Raspberry Pi vorgesehen sind, wie OpenELEC (Open Embedded Linux Entertainment Center) oder auch XBian. Die Installation erfolgt dabei auf ähnliche Art und Weise (per Image oder per Installer) wie bei Raspbmc. Die Funktionen sind ebenfalls vergleichbar. Raspbmc ist jedoch die erste Mediacenter-Lösung, bei der der Anwender ein bereits funktionierendes Paket an die Hand gegeben wird, während bei den anderen Lösungen die manuelle Erstellung einer eigenen Distribution empfohlen wird, was ein recht aufwendiger Vorgang für das Zusammenstellen und Kompilieren werden kann und dem Einsteiger deshalb eher nicht zu empfehlen ist.

Wie beim VLC-Player mit Raspbian sind für VC1-und MPEG2-Dateien, womit dann auch DVDs wiedergegeben werden können, die entsprechenden Lizenzen bei der Raspberry Pi Foundation zu erwerben, wie es im vorherigen Kapitel erläutert ist. Möglicherweise erweist es sich als praktikabel, die vorhandenen MPEG2-Videos in das MPEG4-Format und damit für den Video-Codec H.264 auszulegen, wofür separate Convertersoftware notwendig ist.

Bei aller Euphorie darf nicht außer Acht gelassen werden, dass der Raspberry Pi von der Leistung her, die sich eben bei der Video-Wiedergabe (Encoder) stark bemerkbar macht, schwächer ist als ein einfaches Netbook mit Windows oder ein aktuelles Smartphone, so dass für den Betrieb als Mediacenter die Videowiedergabe die größte Schachstelle darstellt. Für eine flüssige Wiedergabe und die Möglichkeit des Hin- und Herspulens im Video sind deshalb neben den entsprechenden

Codecs mit den dazugehörigen Lizenzen meist noch einige Tricks erforderlich. Beispielsweise ist der Speicher zwischen von CPU und GPU optimal anzupassen (Speicheraufteilung, Memory Split, siehe 1.3.15), der Takt sollte erhöht (Overclock siehe 1.3.11) und eine möglichst schnelle SD Card (Class 10) sollte ebenfalls eingesetzt werden.

4.6 Externe Laufwerke einsetzen

Im Kapitel 2 (vgl. Tabelle 2.1 und 2.2) ist erwähnt, dass Laufwerke unter Linux »gemounted« werden müssen, was demnach auch auf Datenträger zutrifft, die beim Raspberry Pi zusätzlich hinzugefügt werden. Hierfür kommen ausschließlich Laufwerke mit USB-Anschluss in Frage, die dementsprechend über einen USB-Hub anzuschließen sind, wenn weiterhin Tastatur und Maus per USB verbunden sind. USB-Sticks sind die klassischen Vertreter aktueller Laufwerke oder Wechseldatenträger, die meist problemlos am USB-Hub funktionieren und von diesem mit dem notwendigen Strom versorgt werden können, solange der USB-Hub über eine eigene Spannungsversorgung in Form eines separaten Steckernetzteils verfügt.

Für die Versorgung von externen optischen Laufwerken (CD/DVD) und Festplatten reicht die elektrische Leistung des Hubs – je nach Typ – oftmals nicht aus, auch wenn diesen Laufwerken möglicherweise Adapterkabel beiliegen, damit der Strom aus zwei USB-Ports bezogen werden kann. Deshalb sollten diese externen USB-Laufwerke ebenfalls aus eigenen Netzteilen versorgt werden.

Nach dem Anschluss eines dieser USB-Laufwerkstypen ist man es von Windows her gewohnt, dass sie nach kurzer Erkennungsphase unter *Arbeitsplatz* oder *Computer* zur Verfügung stehen, was bei Linux (traditionell) nicht der Fall ist. Gleichwohl hängt es von der jeweiligen Linux-Distribution ab, ob diese Funktionalität nicht doch gegeben ist, wie beispielsweise beim Mediacenter mit Raspbmc, wo das angeschlossene USB-Laufwerk nach dem Anschluss automatisch erkannt wird und die enthaltenen Daten verarbeitet werden können.

Da diese Funktionalität nicht allgemein für Linux-Systeme gegeben ist, werden im Folgenden die wichtigen Dinge erläutert, damit generell externe Laufwerke beim Raspberry Pi eingesetzt werden können. Zur Kontrolle, ob das Laufwerk beim Anschließen erkannt wird, wird die Log-Datei *Messages* mit »tail« und der Option -f für die kontinuierliche Ausgabe zur Anzeige gebracht, was mit Strg+C beendet werden kann. Die laufende Anzeige sollte sich beim Anschließen »bewegen«, indem vermeldet wird, dass ein neues USB-Gerät gefunden und als Laufwerk (beispielsweise als sda: sda1) identifiziert wurde.

```
pi@raspberrypi ~ $ tail -f /var/log/messages
```

```
pi@raspberrypi ~ $ tail -f /var/log/messages
Apr 19 19:21:04 raspberrypi kernel: [   16.794909] mmc0: missed completion of cmd 18 DMA (512/512 [11/[11]) - ignoring it.
Apr 19 19:21:04 raspberrypi kernel: [   16.808795] mmc0: DMA IRQ 6 ignored - results were reset
Apr 19 19:21:04 raspberrypi kernel: [   16.824158] mmc0: missed completion of cmd 18 DMA (512/512 [11/[11]) - ignoring it.
Apr 19 19:21:04 raspberrypi kernel: [   16.838083] mmc0: DMA IRQ 6 ignored - results were reset
Apr 19 19:21:04 raspberrypi kernel: [   16.861074] mmc0: missed completion of cmd 18 DMA (512/512 [11/[11]) - ignoring it.
Apr 19 19:21:04 raspberrypi kernel: [   16.875150] mmc0: DMA IRQ 6 ignored - results were reset
Apr 19 19:21:04 raspberrypi kernel: [   20.921582] smsc95xx 1-1.1:1.0: eth0: link up, 100Mbps, full-duplex, lpa 0x45E1
Apr 19 19:21:04 raspberrypi kernel: [   21.989085] bcm2835-cpufreq: switching to governor ondemand
Apr 19 19:21:04 raspberrypi kernel: [   21.989115] bcm2835-cpufreq: switching to governor ondemand
Apr 19 19:21:06 raspberrypi kernel: [   25.023894] Adding 102396k swap on /var/swap.  Priority:-1 extents:1 across:102396k SS
Apr 19 19:25:29 raspberrypi kernel: [  272.501732] usb 1-1.2.2: new high-speed USB device number 7 using dwc_otg
Apr 19 19:25:29 raspberrypi kernel: [  272.603367] usb 1-1.2.2: New USB device found, idVendor=1058, idProduct=1001
Apr 19 19:25:29 raspberrypi kernel: [  272.603398] usb 1-1.2.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
Apr 19 19:25:29 raspberrypi kernel: [  272.603414] usb 1-1.2.2: Product: External HDD
Apr 19 19:25:29 raspberrypi kernel: [  272.603428] usb 1-1.2.2: Manufacturer: Western Digital
Apr 19 19:25:29 raspberrypi kernel: [  272.603441] usb 1-1.2.2: SerialNumber: 57434153593235343837737
Apr 19 19:25:29 raspberrypi kernel: [  272.615075] scsi0 : usb-storage 1-1.2.2:1.0
Apr 19 19:25:29 raspberrypi mtp-probe: checking bus 1, device 7: "/sys/devices/platform/bcm2708_usb/usb1/1-1/1-1.2/1-1.2.2"
Apr 19 19:25:29 raspberrypi mtp-probe: bus: 1, device: 7 was not an MTP device
Apr 19 19:25:30 raspberrypi kernel: [  273.612898] scsi 0:0:0:0: Direct-Access     ND       6400AAK External 1.05 PQ: 0 ANSI: 4
Apr 19 19:25:30 raspberrypi kernel: [  273.614573] sd 0:0:0:0: [sda] 1250263728 512-byte logical blocks: (640 GB/596 GiB)
Apr 19 19:25:30 raspberrypi kernel: [  273.615674] sd 0:0:0:0: [sda] Write Protect is off
Apr 19 19:25:30 raspberrypi kernel: [  273.633289]  sda: sda1
Apr 19 19:25:30 raspberrypi kernel: [  273.639096] sd 0:0:0:0: [sda] Attached SCSI disk
```

Abbildung 4.13: Der umrandete Teil ist nach der Verbindung der USB-Festplatte mit dem Raspberry Pi in der Log-Datei erschienen.

In der Abbildung 4.13 ist der Erkennungsvorgang einer angeschlossenen USB-Festplatte der Firma Western Digital (wd) gezeigt, die als »sda1« ausgewiesen wird. Der Inhalt der Festplatte oder eines anderen Laufwerks wird üblicherweise in einem separaten Verzeichnis »gemounted«, wobei entsprechend des Typs ein möglichst wiedererkennbarer Name für das Verzeichnis zu wählen ist. Mit der folgenden Zeile wird hierfür ein neues Unterverzeichnis (wdplatte) in Media angelegt:

```
pi@raspberrypi ~ $ sudo mkdir /media/wdplatte
```

Der Name »wdpatte«, das soeben angelegte Verzeichnis, muss jetzt der Festplatte (sda1) zugeordnet werden, wobei das jeweils auf der Festplatte verwendete Dateisystem bekannt sein muss. Hier wird als Typ (mount -t) das übliche Windows-Format für größere Festplatten ntfs bestimmt.

```
pi@raspberrypi ~ $ sudo mount -t ntfs /dev/sda1 /media/wdplatte
```

Beim Mounten einer NTFS-Partition wird möglicherweise die Warnung *seems to be mounted read-only* ausgegeben, so dass zwar von der Partition gelesen, aber keine Daten geschrieben werden können. In diesem Fall fehlt dem Linux ein Teil der NTFS-Unterstützung (ntfs-3g), was mit `sudo apt-get install ntfs-config` nachinstalliert werden kann.

```
pi@raspberrypi ~ $ sudo mount -t ntfs /dev/sda1 /media/wdplatte
pi@raspberrypi ~ $ cd ~
pi@raspberrypi ~ $ cd /media/wdplatte
pi@raspberrypi /media/wdplatte $ ls -l
insgesamt 52428860
drwxrwxrwx 1 pi pi       16384 Feb  4  2010
drwxrwxrwx 1 pi pi        4096 Apr 18  2010
drwxrwxrwx 1 pi pi       16384 Nov 15  2011
drwxrwxrwx 1 pi pi        4096 Nov 15  2011
drwxrwxrwx 1 pi pi        4096 Nov 15  2011
drwxrwxrwx 1 pi pi        4096 Mär 23  2010
drwxrwxrwx 1 pi pi           0 Mär 23  2010
drwxrwxrwx 1 pi pi           0 Mär 16  2010
drwxrwxrwx 1 pi pi           0 Jan 30  2010
-rwxrwxrwx 1 pi pi 53687091200 Mär 23  2010
drwxrwxrwx 1 pi pi        4096 Jan 26  2010
drwxrwxrwx 1 pi pi        4096 Nov 15  2011
drwxrwxrwx 1 pi pi        4096 Mär 16  2010
pi@raspberrypi /media/wdplatte $ cd ..
pi@raspberrypi /media $ sudo umount /media/wdplatte
pi@raspberrypi /media $
```

Abbildung 4.14: Mounten der Festplatte, Anzeige der Verzeichnisse und wieder Unmounten der Festplatte

Damit das externe Laufwerk jedes Mal beim Booten automatisch im System eingebunden wird, kann es in die Datei *fstab* mit eingebunden werden. Vielfach ist dies jedoch gar nicht gewünscht, weil der Datenträger nicht permanent mit dem Rasp-

berry Pi verbunden wird, so dass der entsprechende Mount-Aufruf im Bedarfsfall einfach wieder aus der History-Liste (Pfeil-Auf-Taste » ↑«) zu selektieren ist.

```
pi@raspberrypi ~ $ lsusb
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 008: ID 0409:005a NEC Corp. HighSpeed Hub
Bus 001 Device 005: ID 04d9:1503 Holtek Semiconductor, Inc. Shortboard Lefty
Bus 001 Device 014: ID 1058:1001 Western Digital Technologies, Inc. External Hard Disk [Elements]
Bus 001 Device 009: ID 13fd:2040 Initio Corporation
Bus 001 Device 010: ID 0d62:a100 Darfon Electronics Corp. Optical Mouse
pi@raspberrypi ~ $ lsusb -t
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=dwc_otg/1p, 480M
    |__ Port 1: Dev 2, If 0, Class=hub, Driver=hub/3p, 480M
        |__ Port 1: Dev 3, If 0, Class=vend., Driver=smsc95xx, 480M
        |__ Port 2: Dev 8, If 0, Class=hub, Driver=hub/4p, 480M
            |__ Port 2: Dev 14, If 0, Class=stor., Driver=usb-storage, 480M
            |__ Port 3: Dev 9, If 0, Class=stor., Driver=usb-storage, 480M
            |__ Port 4: Dev 10, If 0, Class=HID, Driver=usbhid, 1.5M
        |__ Port 3: Dev 5, If 0, Class=HID, Driver=usbhid, 1.5M
        |__ Port 3: Dev 5, If 1, Class=HID, Driver=usbhid, 1.5M
pi@raspberrypi ~ $
```

Abbildung 4.15: Informationen über die angeschlossenen und erkannten USB-Geräte liefert der Befehl »lsusb«, hier in zwei unterschiedlichen Darstellungsweisen.

An dieser Stelle sei angemerkt, dass es für die Überprüfung, welche USB-Geräte angeschlossen – und erkannt – worden sind, auch noch einen speziellen Befehl *lsusb* (vgl. Abbildung 4.15) gibt, der durch die Angabe der Option -t eine Baumstruktur mit den jeweiligen USB-Gerätedaten ausgibt.

Selbstverständlich können auch USB-Sticks oder optische Laufwerke (CD/DVD), die über einen USB-Anschluss verfügen, »gemountet« werden. Die Vorgehensweise ist dabei prinzipiell die gleiche, wie es zuvor für die Festplatte beschrieben wurde, es ist sogar noch einfacher, denn üblicherweise muss das Fileformat hierfür nicht explizit angegeben werden, sondern wird meist automatisch erkannt, wie es etwa mit Raspbian der Fall ist.

Falls das Mounten bei einem gewöhnlichen USB-Stick jedoch nicht funktionieren will, ist für das Fileformat *vfat* anzugeben und für ein CD/DVD-ROM-Laufwerk *iso9660*. Falls es sich um einen Brenner handelt, ist auf der Kommandozeile keine Schreibfunktionalität gegeben, die stattdessen mit einem separaten Programm (unter einen grafischen Desktop, LXDE o.ä.) zur Verfügung gestellt wird.

Für die einfachere Orientierung empfiehlt es sich, für jeden speziellen Laufwerkstyp vor dem Mounten ein eigenes Verzeichnis anzulegen (beispielsweise `sudo mkdir /media/usbstick` oder `sudo mkdir /media/dvdlw`). Zuvor ist die jeweilige Kennzeichnung des Laufwerks (mit *tail*, siehe oben) zu ermitteln.

Mounten USB-Stick (sdb1):

```
pi@raspberrypi ~ $ sudo mount /dev/sdb1 /media/usbstick
```

Mounten CD/DVD-Laufwerk (sr0):

```
pi@raspberrypi ~ $ sudo mount /dev/sr0 /media/dvdlw
```

4.7 Drucken

Vielfach wird mit dem Raspberry Pi mithilfe von PuTTY oder TightVNC über eine Netzwerkverbindung (siehe folgendes Kapitel) kommuniziert, wobei der Host typischerweise von einem PC gebildet wird, der auch eine Druckfunktionalität zur Verfügung stellt, so dass der Raspberry Pi dann selbst keine bieten muss.

Falls der Raspberry Pi jedoch als eigenständiges System fungieren soll, sei es als Office-Computer oder Embedded System, kann eine direkte Druckerunterstützung sinnvoll sein. Außerdem kann somit auf einfache Art und Weise ein gewöhnlicher Drucker mit USB-Anschluss zu einem Netzwerkdrucker erweitert werden. Es ist ebenfalls kein weiterer Aufwand nötig, um mit dem Raspberry Pi einen Druckerserver für das eigene Netzwerk zu realisieren.

Der zu verwendende Drucker ist über den USB – über einen USB-Hub – zu verbinden und sollte nach dem Anschluss automatisch als USB-Gerät identifiziert

werden, wie es zuvor für die Laufwerke (Abbildung 4.15) erläutert ist. Unter Linux ist das *Common Unix Print System* (CUPS) als Standard zu betrachten, welches die Drucker mit den entsprechenden Diensten steuert und von der Firma Apple entwickelt wurde. CUPS liefert neben den für das Drucken notwendigen Programmen standardmäßig eine Vielzahl von Druckertreibern. CUPS wird installiert mit:

```
pi@raspberrypi ~ $ sudo apt-get install cups
```

Neben den damit automatisch mitgelieferten Standardtreibern gibt es einige (herstellerspezifische) Treiberbibliotheken, die die Nutzung speziellerer Optionen und Funktionen der einzelnen Drucker gestatten. Falls der Druckerhersteller einen spezifizierten Linux-Treiber (Debian 6) für das jeweilige Modell zur Verfügung stellt, sollte dieser sämtliche Eigenschaften des Druckers abdecken, d.h., er ist die beste Wahl und sollte funktionstechnisch über die mitunter eher rudimentäre CUPS-Unterstützung hinausgehen.

Die CUPS-Druckerkonfigurierung ist am einfachsten über einen Web-Browser möglich, denn CUPS verfügt über eine integrierten Webserver, der über den Port 631 aufgerufen wird. Hierfür kann der bei Raspbian unter LXDE standardmäßig vorhandene Browser Midori eingesetzt werden, in dem hier die folgende Adresszeile eingegeben wird:

```
http://localhost:631
```

Daraufhin erscheint die CUPS-Konfigurationsseite. Einstellungen sind jedoch ausschließlich Mitgliedern der Gruppe *lpadmin* (Line Printer Administrator) gestattet, so dass der User Pi (mit dem man sich angemeldet hat) dieser Gruppe hinzuzufügen ist, was wie folgt durchgeführt wird:

```
sudo usermod -a -G lpadmin pi
```

Diese Zeile sollte vom Linux-System kommentarlos übernommen werden, so dass nunmehr auch die Konfiguration mit CUPS durchgeführt werden kann. Hierfür ist zunächst die Seite unter *Verwaltung* aufzurufen (Abbildung 4.16).

Daraufhin ist *Drucker hinzufügen* zu wählen, woraufhin die erforderliche Authentifizierung zu erfolgen hat, indem hier der übliche Benutzername (pi) mit dem dazugehörigen Password anzugeben ist. Ein angeschlossener USB-Drucker sollte in der nun folgenden Liste auftauchen und ist dementsprechend zu selektieren. Optional ist es an dieser Stelle auch möglich, nach Netzwerkdruckern zu suchen und den »neuen« Drucker über *Freigabe* als solchen zu bestimmen.

Abbildung 4.16: Die Druckerkonfigurierung wird über Verwaltung ausgelöst.

Unter *Modell* werden typischerweise gleich mehrere (ähnliche) Druckertypen auftauchen, wobei dann die Betätigung des Buttons *Drucker hinzufügen* verschiedene konfigurierbare Optionen für den zuvor selektierten Drucker erscheinen lässt. Im einfachsten Fall wird an dieser Stelle lediglich *Standardeinstellungen festlegen* ausgewählt, womit die Konfigurierung bereits beendet ist.

Abbildung 4.17: Festlegen der Druckerstandardeinstellungen

Falls sich während der Druckerkonfigurierung mit CPUS herausstellen sollte, dass eine spezielle Treiberbibliothek für einen bestimmten Drucker notwendig ist, wird diese entsprechend angezeigt (z.B. gutenprint) und sollte daraufhin installiert

(`sudo apt-get install cups-driver-gutenprint`) werden. Von vornherein diese oder jene Treiberbibliothek auf »Verdacht hin« zu installieren, macht keinen Sinn, denn diese können recht umfangreich sein, was den zur Verfügung stehenden Speicherplatz auf der SD-Karte reduziert. Es ist auch nicht unmittelbar ersichtlich, ob der gewünschte Drucker tatsächlich unterstützt wird. Deshalb ist es zu empfehlen, CUPS und die generell die Package List (`sudo apt-get update`) aktuell zu halten, damit CUPS die aktuell notwendigen Treiber im Bedarfsfall ausweisen kann.

Abbildung 4.18: Drucken einer Testseite

Nach der Installation von CUPS sind die wichtigen Konfigurationsdaten unter */etc/cups* zu finden. Für die grundlegende Konfiguration ist dabei der Inhalt der Datei cupsd.conf von Bedeutung, die auch mit einem üblichen Editor (z.B. mit nano) angepasst werden kann. Oftmals ist diese manuelle Konfigurierung jedoch gar nicht notwendig, denn die CUPS-Konfigurationsseite (Abbildung 4.16) bietet eine Fülle von Einstellungsmöglichkeiten mit ausführlichen Erläuterungen für alle möglichen Drucker-Einsatzszenarien.

4.8 Netzwerkverbindungen und -Einstellungen

Die Raspberry Pi Typ B-Modelle verfügen über eine Twisted Pair-Buchse (Kapitel 3.6) für den Anschluss an ein Netzwerk gemäß des Ethernet-Standards, so dass mithilfe des bei Raspbian und den meisten anderen Linux-Versionen voreingestellten DHCP (fast) automatisch eine Netzwerkverbindung hergestellt werden kann.

Die in Netzwerken vorherrschende Technologie wird von einem Ethernet-Standard (IEEE 802.x) definiert, dessen Ursprung bis in die achtziger Jahre reicht

und dennoch ein hohes Maß an Kompatibilität aufweist. Verschiedene Topologien, Übertragungsverfahren und Medien (Koaxialkabel, Twisted Pair-Kabel, Lichtwellenleiter) erlauben unterschiedliche Datenraten, wobei die aktuellen und verbreiteten Ethernet-Implentierungen in LANs als 10BaseT (10 MBit/s), 100BaseTX (Fast Ethernet, 100 MBit/s) und 1000BaseT (1 GBit/s) bezeichnet werden. Das »T« steht dabei für das Medium Twisted Pair-Kabel. Innerhalb der genannten BaseT-Standards existiert Kompatibilität, die auch von den Raspberry Pi-Ethernet-Chips LAN9512 und LAN9514 unterstützt wird.

Abbildung 4.19: Aktuell verwendet man ein TP-Kabel der Kategorie 5, welches aus vier verdrillten Leitungspaaren besteht.

Ein Twisted-Pair-Kabel für Netzwerkverbindungen besteht – wie es die Bezeichnung impliziert – aus verdrillten, maximal acht einzelnen Leitungen. TP-Kabel sind in unabgeschirmter (UTP, Unshielded Twisted Pair) und abgeschirmter (STP, Shielded Twisted Pair) Ausführung erhältlich, wobei das letztere unempfindlicher gegen äußere Störungen ist. Das STP-Kabel gibt es auch noch mit einer gemeinsamen Abschirmung (S-UTP), die als Mantel (Geflecht, Metallfolie) um alle Leitungen geführt ist, und in einer anderen Auslegung, bei der jedes Aderpaar (einzeln) abgeschirmt ist. Das gebräuchlichste TP-Kabel ist das S-UTP-Kabel der Kategorie 5 (Cat5-Kabel), welches sich für alle drei Base-Standards eignet. Die Verbindungen werden generell 1:1 von Gerät (PC) zu Gerät (Hub, Switch) verlegt, was bedeutet, dass sich eine sternförmige Verkabelung der LAN-Komponenten ergibt, wie es in der Abbildung 4.20 angedeutet ist.

Abbildung 4.20: Eine typische Ethernet-Topologie

4.8.1 Dynamic Host Configuration Protocol – DHCP

Standardmäßig wird dem Ethernet-Adapter des Raspberry Pi von einem im Netzwerk befindlichen Router oder Server automatisch eine IP-Adresse per DHCP zugeteilt. Im Privatbereich entspricht der Router typischerweise dem xDSL-Modem, welches außerdem einen Switch und eine Routerfunktion besitzt, während im professionellen Bereich DHCP (wenn überhaupt) von einem Server ausgeführt wird.

Im einfachsten Fall ist im Privatbereich deshalb nur sicherzustellen, dass die DHCP-Funktion tatsächlich im Router aktiviert ist. Der Raspberry Pi wird lediglich über ein Twisted Pair-Kabel mit dem (Switch im) Router verbunden. Daraufhin werden möglicherweise gleich die drei Leuchtdioden (LNK, FDX, 100) für die Netzwerkstatusanzeige (siehe Abbildung 1.12) aktiv. Danach ist ein Neuboot des Raspberry Pi auszulösen (sudo reboot) und es erscheint am Ende des Bootvorgangs vor dem Login eine neue Zeile:

```
[ ok ] Starting OpenBSD Secure Shell server: sshd
My IP address is 192.168.0.50
```

Die per DHCP zugeteilte IP-Adresse ist demnach 192.168.0.50, so dass der Raspberry Pi für das Netzwerk damit bereits kommunikationsfähig ist. Vielfach ist jedoch kein DHCP gewünscht, etwa wenn dem Raspberry Pi eine feste IP-Adresse für den Einsatz als Druckerserver zugeteilt werden soll oder wenn im Netz kein DHCP-Server zur Verfügung steht, so dass man sich etwas näher mit der Netzwerktechnik auseinandersetzen muss.

4.8.2 Übersicht und Analyse

Bei vielen Linux-Distributionen (SuSE, Redhat) können die Netzwerkeinstellungen bequem über eine grafische Oberfläche (YAST, linuxconf) vorgenommen werden, was es in einer vergleichbaren Form für Debian und die darauf basierten Systeme wie Ubuntu oder eben Raspbian nicht gibt. Stattdessen sind die Netzwerkeinstellungen über die Werkzeuge der jeweiligen grafischen Oberfläche (KDE, Gnome, LXDE) durchführbar, wobei LXDE hierfür allerdings keine Einstellungsmöglichkeiten vorsieht. Deshalb sind die Netzwerkeinstellungen mit einem Editor manuell durchzuführen, was recht einfach zu handhaben ist, so dass hierfür keine separaten Programme oder Tools notwendig sind. Wichtig sind für die manuelle Konfigurierung die folgenden Dateien:

- **/etc/network/interfaces** Netzwerkeinstellungen (IP, Netmask, Gateway)
- **/etc/resolv.conf** Definition des Nameservers (DNS)
- **/etc/hostname** Name des Raspberry Pi im Netz (Hostname)

Der Treiber für den Netzwerkadapter, der auf der Platine mit dem Chip LAN9512/9514 realisiert ist, wird mit Raspbian und auch mit allen anderen be-

kannten Distributionen automatisch installiert, so dass hierfür keine Konfigurationsarbeiten notwendig sind.

Für eine Übersicht der vorhandenen Netzwerkinterfaces und der jeweiligen Daten wird der Befehl *ifconfig* angewendet (Abbildung 4.21). Mit ihm lassen sich nicht nur Daten anzeigen, sondern die Schnittstelle ein- und ausschalten sowie die Konfiguration anhand verschiedener Parameter durchführen. Eine Dokumentation ist dazu wie üblich mit man *ifconfig* verfügbar, so dass auf die weiteren Funktionen hier nicht weiter eingegangen werden muss.

```
pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet  Hardware Adresse b8:27:eb:04:8a:77
          inet Adresse:192.168.0.50  Bcast:192.168.0.255  Maske:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metrik:1
          RX packets:3546 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1882 errors:0 dropped:0 overruns:0 carrier:0
          Kollisionen:0 Sendewarteschlangenlänge:1000
          RX bytes:2950150 (2.8 MiB)  TX bytes:234367 (228.8 KiB)

lo        Link encap:Lokale Schleife
          inet Adresse:127.0.0.1  Maske:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metrik:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          Kollisionen:0 Sendewarteschlangenlänge:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

pi@raspberrypi ~ $
```

Abbildung 4.21: Der Befehl ifconfig zeigt die Einstellungen der vorhandenen Netzwerkadapter.

In der Abbildung 4.21 zeigt die Anwendung des Befehls *ifconfig*, dass hier zwei Netzwerkinterfaces auftauchen, wobei *eth0* dem Ethernet-Adapter mit dem Chip LAN9512 entspricht und *lo* dem bei Linux grundsätzlich vorhandenen, so genannten *Loopback Interface* mit der IP-Adresse 127.0.0.1, welches beispielsweise die lokale Kommunikation zwischen einer Server- und einer Client-Anwendung gestattet, wie es etwa bei der CUPS-Druckerkonfigurierung über die Webseite (siehe vorheriges Kapitel) praktiziert wird.

4.8.3 Netzwerkadressen

Mit der *Hardware Adresse* (Abbildung 4.21) ist bei *eth0* die MAC-Adresse (Media Access Control) gemeint, die jeweils weltweit für jeden Netzwerkadapter einmalig ist, und auf der die jeweils zugeteilte IP-Adresse (Internet Protocol) abgebildet wird. Eine offizielle IP-Adresse ist ebenfalls weltweit einmalig. Diese Adressen sind ursprünglich in den USA einmal festgelegt und zur Vergabe an länderspezifische Institutionen übertragen worden. In den Besitz von weltweit gültigen IP-Adressen kommt man, wenn diese bei einem Provider beantragt werden. In

Deutschland ist für die Registrierung unterhalb der *Top Level Domains* mit der Endung ».de« die DENIC in Frankfurt zuständig. Zahlreiche Provider bieten einen entsprechenden Service für die Domain-Registrierung an, die als reservierte Adresse bei der DENIC oder für internationale Domains (.com, .org, .edu) bei der ICANN eingetragen und verwaltet wird, was kostenpflichtig ist. Für die Benutzung in einem internen Netz (zu Hause, Firma) wird man stattdessen auf die internen oder auch privaten IP-Adressen zurückgreifen, die nichts kosten und nicht im Internet zur Anwendung kommen. Für die Umsetzung der privaten IP-Adressen im LAN auf eine offizielle IP-Adresse im Internet ist der Router zuständig, dem der jeweilige Provider eine (dynamische) IP-Adresse zuteilt.

Eine IPv4-Adresse besteht aus einem 32-Bit langen Feld, wobei jede Zahl kleiner als 256 ist und jeweils durch Punkte getrennt wird (Dotted Decimal Notation, DDN). Die IP-Adresse setzt sich aus zwei Teilen zusammen: der Netzwerk- und der Hostadresse, die mitunter auch als *Rechneradresse* bezeichnet wird. Eine IP-Adresse wird dabei stets in Dezimalzahlen zu vier Böcken, durch einen Punkt voneinander getrennt, angegeben. Ein Block wird auch als *Oktett* bezeichnet.

Die Netzwerkadresse bildet den vorderen (linken) Teil, während der hintere Teil (rechts) von der Hostadresse gebildet wird. Anhand der Netzwerkadresse wird das gesamte Netz bzw. ein Teilnetz (Subnet) angesprochen, nicht aber ein einzelner Host. Der zweite Teil ist die Hostadresse, die innerhalb der jeweiligen Institution (lokal) etwa für einen PC vergeben wird.

Durch diese Aufteilung ist es möglich, beispielsweise einer Firma eine Netzwerkadresse zuweisen zu können, die dann – je nach Klasse – eine bestimmte Anzahl von Hostadressen zur Verfügung hat. Ein Beispiel:

Netz- Hostadresse

134.28. 12.10

Der PC mit dieser IP-Adresse (134.28.12.10) befindet sich demnach im gleichen Netzwerk wie einer mit der Adresse 134.28.12.56. Ein PC mit der Adresse 134.29.60.10 befindet sich hingegen in einem anderen Netz, was einen Routing-Vorgang – eine Wegfindung – zur Folge hat, der per Router bzw. Gateway erfolgt. Wie es der Tabelle 4.1 zu entnehmen ist, erhält man in Abhängigkeit von einer Subnet-Mask den jeweiligen Klassen entsprechend eine unterschiedliche Anzahl von Netz- und Hostadressen.

Für ein LAN in einer Firma oder auch daheim macht es sicher keinen Sinn – und fördert auch nicht gerade die Übersichtlichkeit, wenn mehr als eine einzige Netzadresse verwendet wird. Demnach sollte man sich eine aussuchen und dann eine Unterteilung (z.B. 192.168.0.x) auf einzelne Hosts mit der Subnet-Mask vornehmen, etwa durch 255.255.255.0, was zu einem Netz mit 254 möglichen Hosts führt.

Tabelle 4.1: Die privaten Internetadressen

Adressbereiche	Klasse	Subnet Mask
10.0.0.0 bis 10.255.255.255	A	255.0.0.0
	B	255.255.0.0
	C	255.255.255.0
172.16.0.0 bis 172.31.255.255	B	255.255.0.0
	C	255.255.255.0
192.168.0.0 bis 192.168.255.255	C	255.255.255.0

Bei Adressen der Klasse A wird das erste Oktett (die ersten 8 Bit) für die Netzwerk-ID verwendet, bei Adressen der Klasse B das erste und das zweite Oktett (16 Bit) und bei Adressen der Klasse C die ersten drei (24 Bit), woran sich dementsprechend die Host-IDs anschließen, wie es auch in der Tabelle 4.2 gezeigt ist.

Tabelle 4.2: Die Internet-Adressklassen und die sich daraus ergebende maximale Anzahl von Netz- und Hostadressen

Klasse	Bereich	Netzadressen	Hostadressen in einem Netz
A	0-127	126	16.777.214
B	128-191	16.384	65.534
C	192-223	2.097.152	254
D	224-249	-	-
E	240-255	-	-

Die Adressen der Klasse A lassen einen großen Freiraum für die Adressenvergabe innerhalb eines Unternehmens oder Verbundes zu, da hiermit relativ große Netze aufgebaut werden können. Üblich sind heute Adressen der Klasse B und C, wobei einige für spezielle Aufgaben (Broadcast, Loopback) reserviert sind.

Eine Erweiterung des Internet Protokolls wird als IPv6 bezeichnet und verwendet statt eines Adressraumes von 32 Bit (IPv4) einen mit 128 Bit, was zu einer geradezu unvorstellbaren Anzahl (340,28 Sextillionen) von möglichen IP-Adressen führt. IPv6 oder IPnG (IP Next Generation), wie dieser Standard auch bezeichnet wird, muss zu IPv4 kompatibel sein, was dadurch erreicht wird, dass die bisherigen IPv4-Adressen mit in den unteren Bereich von IPv6 aufgenommen werden können.

4.8.4 Konfigurationsdatei anpassen

Die wichtigste Datei für die Netzwerkkonfiguration ist *interfaces* unter */etc/network*, die standardmäßig die in der folgenden Abbildung gezeigten Einträge enthält und mit einem Editor wie etwa *nano* bearbeitet (`sudo nano /etc/network/interfaces`) werden kann.

```
/etc/network/interfaces
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Abbildung 4.22: Der Inhalt der ursprünglichen Interfaces-Datei mit DHCP

Der Interfaces-Datei ist zu entnehmen, dass nach den beiden Einträgen für das Loopback-Interface die Einstellung für das eth0-Interface folgt, welches auf DHCP (iface eth0 inet dhcp) eingestellt ist. Die anderen Zeilen betreffen das WLAN-Interface, wofür die Daten hier bereits voreingestellt sind (siehe Kapitel 4.9.3).

Für eine manuelle Vergabe der IP-Adresse ist eben diese, die im gleichen Netz wie die anderen Computer des LANs liegen muss und keinesfalls bereits von einem anderen Gerät verwendet werden darf, festzulegen, außerdem die dazugehörige Netzmaske und die IP-Adresse des Gateway, welches in einem typischen Heimnetz vom (DSL-)Router gebildet wird.

In der Interfaces-Datei (/etc/network) sind für eine feste IP-Adresse die folgenden Eintragungen notwendig, wobei eine entsprechende Einrückung mit der TAB-Taste bei den Adresszeilen durchgeführt werden sollte. Beispielhaft ist hier die IP-Adresse 192.168.0.50 für den Raspberry Pi und 192.168.0.110 für den (DSL-)Router gewählt worden, der den DNS-Server »kennt«:

```
iface eth0 inet static
        address 192.168.0.50
        netmask 255.255.255.0
        gateway 192.168.0.110
```

Nach dem Abspeichern der Interfaces-Datei treten die Änderungen automatisch nach dem nächsten Boot in Kraft, oder es wird ein Restart ausgeführt mit dem Abschalten (ifdown) und Wiederanschalten (ifup) des Interface:

```
pi@raspberrypi ~ $  sudo /etc/init.d/networking restart
```

```
pi@raspberrypi ~ $  sudo ifdown eth0
pi@raspberrypi ~ $  sudo ifup eth0
```

Zum Test des Weges – der Route – und optional für die Konfigurierung der IP-Routing-Tabelle kann der Befehl *route* angewendet werden, der den verwendeten Router und weitere Optionen ausweist, wobei mit *man route* hierzu wieder eine Anleitung angezeigt werden kann.

```
  GNU nano 2.2.6

iface eth0 inet static
        address 192.168.0.50
        netmask 255.255.255.0
        gateway 192.168.0.110

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Abbildung 4.23: Angepasste Interfaces-Datei mit festen IP-Adressen für eth0

4.8.5 Adressenumsetzung – Domain Name Service

Die Gateway-Adresse wird meist auch als Adresse für den *Domain Name Service* (DNS) verwendet und ist in der Datei */etc/resolv.conf* zu finden. DNS wird für die Umsetzung von IP-Adressen in Internet-Adressen benötigt. Der Router findet meist automatisch einen entsprechenden DNS-Server im Internet bzw. im Router ist explizit eine DNS-Adresse angegeben, die dem Anwender vom jeweiligen Anbieter des Internetzugangs her bekannt ist.

Falls diese Adressenumsetzung nicht funktionieren sollte, kann eine Internetseite zwar direkt über eine IP-Adresse (173.194.113.142), nicht jedoch über den Namen (google.com) aufgerufen werden. Für Verbindungstests eignet sind generell der ping-Befehl. Im Gegensatz zum ping-Pendant bei Windows wird der ping-Befehl bei Linux ohne optionale Angaben fortlaufend ausgeführt, bis er mit Strg+C abgebrochen wird.

In der Abbildung 4.24 ist zu erkennen, dass mit dem ersten ping-Kommando das Vorhandensein eines anderen Computers im LAN (192.168.0.2) überprüft wird und mit dem zweiten der Zugriff auf Google. Demnach funktioniert hier auch die Adressenumsetzung mit dem DNS-Server, der die Bezeichnung ham02s11-in-f6.1e100.net trägt und in Hamburg von O2 betrieben wird. Falls die Adressenumsetzung mit DNS nicht funktionieren sollte, ist die zum DNS-Server dazugehörige IP-Adresse in */etc/resolv.conf* einzutragen.

```
pi@raspberrypi ~ $ ping 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.2: icmp_req=1 ttl=128 time=0.542 ms
64 bytes from 192.168.0.2: icmp_req=2 ttl=128 time=0.577 ms
64 bytes from 192.168.0.2: icmp_req=3 ttl=128 time=0.506 ms
^C
--- 192.168.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.506/0.541/0.577/0.039 ms
pi@raspberrypi ~ $ ping google.com
PING google.com (173.194.113.134) 56(84) bytes of data.
64 bytes from ham02s11-in-f6.1e100.net (173.194.113.134): icmp_req=1 ttl=58 time=33.1 ms
64 bytes from ham02s11-in-f6.1e100.net (173.194.113.134): icmp_req=2 ttl=58 time=32.8 ms
64 bytes from ham02s11-in-f6.1e100.net (173.194.113.134): icmp_req=3 ttl=58 time=33.6 ms
64 bytes from ham02s11-in-f6.1e100.net (173.194.113.134): icmp_req=4 ttl=58 time=32.7 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 32.769/33.105/33.648/0.361 ms
pi@raspberrypi ~ $
```

Abbildung 4.24: Verbindungstests mit dem ping-Befehl

Wie erwähnt wird dem Teilnehmer – dem Router – bei der Einwahl vom jeweiligen Provider eine (dynamische) IP-Adresse zuteilt, wenn keine statische erworben worden ist. Bei längerer Verbindungsdauer schaltet der Provider die Teilnehmerleitung mitunter kurzzeitig ab, damit ein erneuter Verbindungsaufbau durchgeführt wird. Bei jeder Einwahl erhält der Teilnehmer eine andere weltweit gültige IP-Adresse, was für im Internet zu erreichende Applikationen (eigener Webserver) ein Problem darstellt, weil der Name eben über eine IP-Adresse abgebildet wird. Damit ein bestimmtes Gerät trotz wechselnder IP-Adresse im Internet stets über einen eigenen Hostnamen erreicht werden kann, gibt es den DynDNS-Dienst, der von verschiedenen Firmen – auch kostenlos – angeboten wird, wie etwa MyFritz!, der jedoch explizit auf die Fritz!-Geräte der Firma AVM ausgelegt. Universell und kostenlos einsetzbar ist beispielsweise der DynDNS-Dienst der Firma Securepoint (`http://www.securepoint.de/produkte/utm-firewalls/securepoint-dyndns.html`).

4.8.6 Verbindungsverfahren

Um einen Kontakt zwischen dem Raspberry Pi und einem Computer mit Windows, Linux, Mac OS oder auch Android oder iOS aufzunehmen, gibt es – je nach Einsatzzweck – zahlreiche verschiedene Möglichkeiten. In der folgenden Tabelle sind gebräuchliche Verfahren hierfür angegeben, die mithilfe eines LAN oder WLAN funktionieren.

Tabelle 4.3: Kommunikationsmöglichkeiten zwischen Raspberry Pi und anderen Computern

Anwendung/Einsatz	Raspberry Pi	Computer
Windows-Freigaben	Server Message Block (SMB) mit Samba	Windows-Dateimanager als Client
Windows-Desktop	Client mit Remote Desktop Protocol (RDP)	Remote Desktop Service über Remoteverbindung
Webserver	Hyper Text Markup Language (http) mit Apache als Server	Internet Browser als Client
Virtual Network Computing	VNC-Server oder -Client	VNC-Client oder -Server
Remote-Zugriff mit Secure Shell	SSH-Server oder -Client	SSH- Client oder -Server
Dateizugriff mit File Transfer Protocol (FTP)	FTP-Server oder -Client	FTP- Client oder -Server

Prinzipiell ist es unerheblich, von welchem Typ der Computer ist, sei es ein gewöhnlicher PC, ein Notebook, ein Tablet oder auch ein Smartphone. Wenn es eine entsprechende Client- oder Serversoftware für das jeweils auf dem Gerät befindliche Betriebssystem gibt, ist es auch möglich, das jeweilige Protokoll mit der passenden Kommunikationsanwendung auszuführen.

Für den Raspberry Pi, der weniger leistungsfähig ist als ein PC oder selbst als ein aktuelles Smartphone, ist diese Limitierung, die nun einmal mit begrenzten Ressourcen (Taktrate, Speicher, Grafik) einhergeht, stets mit zu beachten. Es ist zu bedenken, ob es wirklich sinnvoll ist, umfangreiche Pakete wie für Samba oder Apache zu installieren, die später beim Zugriff durch (mehrere) Clients nicht schnell genug ausgeführt werden können. Deshalb wird im Folgenden lediglich auf drei verhältnismäßig einfache Verbindungsmechanismen eingegangen, die problemlos und ressourcenschonend auf dem Raspberry Pi aktiviert und ausgeführt werden können.

4.8.7 Secure Shell – SSH

Nachdem der Raspberry Pi im LAN integriert ist, erweist es sich vielfach als bequem und nützlich, wenn der Zugriff auf den Raspberry Pi mithilfe eines PC möglich ist, was auch in umgekehrter Konstellation erfolgen kann. Ein Netzwerkprotokoll, mit dem dies einfach und sicher durchführbar ist, wird als Secure Shell (SSH)

bezeichnet. SSH löst technologisch das mittlerweile als veraltet und unsicher – weil abhörbar – betrachtete Telnet-Verfahren ab.

Das SSH-Protokoll setzt einen Server- und einen Client-Dienst zwischen den beiden zu verbindenden Computersystemen voraus. Der SSH-Client ist beim Raspberry Pi automatisch aktiv, so dass beispielsweise mit der Befehlszeile *ssh compter2* auf den Computer2 zugegriffen werden kann, sofern hier ein SSH-Server aktiv ist.

Falls der Raspberry Pi als SSH-Server eingesetzt werden soll, muss dies nur entsprechend aktiviert werden, was im Kapitel 1.3.16 behandelt ist und jederzeit durch den erneuten Aufruf mit `sudo raspi-config` über *SSH Enable* durchgeführt werden kann. Dementsprechend muss auf dem PC, mit dem auf den Raspberry Pi zugegriffen werden soll, ein SSH-Client arbeiten. Das Programm PuTTY ist hierfür beispielsweise geeignet, welches kostenlos aus dem Internet bezogen werden kann.

Neben PuTTY existieren natürlich noch andere Programme, die einen Windows-Computer mit SSH-Funktionalität ausstatten können, wie Cygwin oder Winscp, wobei SSH bei Linux-Distributionen standardmäßig dazugehört. PuTTY benötigt jedoch nicht zwangsweise eine Installation, kann nach dem Kopieren direkt aufgerufen werden und ist sehr einfach zu handhaben. Im Programm ist lediglich die Adresse des Raspberry Pi anzugeben (hier 192.168.0.50). Unter der Option *Translation* empfiehlt es sich, bei *Remote Character Set* den Zeichensatz UTF-8 zu spezifizieren, damit auch die Darstellung der (meisten) Sonderzeichen funktioniert. Standardmäßig wird der Port 22 für die Kommunikation eingesetzt, was üblicherweise auch nicht geändert werden muss (Abbildung 4.25).

Nach der Betätigung des *Open*-Buttons erscheint auf dem PC ein Terminalfenster, in dem wie üblich die Anmeldung vorzunehmen ist. Daraufhin kann mit dem PC auf den Raspberry Pi (im Konsolen-/Terminal-/Textmodus) zugegriffen und gearbeitet werden. Außer der Tatsache, dass per SSH keine grafische Oberfläche (LXDE) verwendet werden kann, gibt es prinzipiell keinerlei Einschränkungen dahingehend, ob mit dem Raspberry Pi direkt per angeschlossener Tastatur oder mit einem PC per SSH kommuniziert wird. Das Abspeichern der Kommunikationsparameter (Load) funktioniert im Übrigen mit PuTTY nur dann, wenn das Programm nicht direkt aufgerufen, sondern komplett installiert worden ist, wofür unterschiedliche PuTTY-Versionen zur Verfügung stehen. Mithilfe von PuTTY ist es demnach einfach möglich, den Raspberry Pi auch ohne Monitor und Tastatur betreiben zu können. Falls es nicht ausreichen sollte, dass auf dem PC lediglich ein Textterminal zur Verfügung steht und der Raspberry Pi-Desktop inklusive Maus- und Tastaturfunktion ferngesteuert werden soll, ist ein Programm wie VNC (Virtual Network Computing) notwendig, und zwar auch wieder in einer Client- und einer Serverversion.

Abbildung 4.25: Kontaktaufnahme eines PC mit dem Raspberry Pi per SSH mithilfe von PuTTY

4.8.8 Virtual Network Computing – VNC

Wie erwähnt, erlaubt PuTTY bzw. SSH nicht die Nutzung einer grafischen Oberfläche, sondern funktioniert nur im Textmodus. Mithilfe von *Virtual Network Computing* (VNC) ist es jedoch möglich, Tastatur, Maus und Desktop von einem PC oder anderen Computer aus zu steuern. Hierfür werden wieder zwei Partner benötigt, wobei der eine als VNC-Server und der andere als VNC-Client fungiert. Um wieder die gebräuchlichste Konstellation zu wählen, wird auf dem Raspberry Pi demnach ein VNC-Server und auf dem PC ein VNC-Client eingesetzt. Wie für SSH- gibt es auch für VNC-Verbindungen verschiedene Software-Lösungen. *Tight VNC* kann beide VNC-Funktionen ausführen, ist kostenlos und für verschiedene Plattformen (Linux, Windows) verfügbar, so dass diese Software hier beispielhaft für den Aufbau einer VNC-Verbindung eingesetzt wird.

Für den Raspberry Pi ist der VNC-Server wie folgt zu installieren:

```
pi@raspberrypi ~ $ sudo apt-get install tightvncserver
```

Danach erfolgt der Aufruf mit:

```
pi@raspberrypi ~ $ tightvncserver
```

Falls dies das erste Mal passiert, wird die Vergabe eines Passworts verlangt, welches mindestens eine Länge von acht Zeichen aufweisen muss und jedes Mal beim VNC-Verbindungsaufbau anzugeben ist.

Als Client wird die Tight VNC-Windows-Software benötigt, die beispielsweise auf `http://www.tightvnc.com/download.php` zum kostenlosen Download zur Verfügung steht. Wie beim Verbindungsaufbau mit PuTTY ist die IP-Adresse des Raspberry Pi anzugeben und ein spezieller Port. Der standardisierte Basisport lautet für TightVNC 5900. Es können mehrere gleichzeitige »Bildschirme« (Sessions) aktiviert werden, so dass der erste mit Port 5901 (Abbildung 4.26), der zweite mit 5902 usw. selektiert wird.

Abbildung 4.26: Verbindung mit TightVNC aufrufen

Abbildung 4.27: Bevor die Verbindung etabliert wird, ist das Passwort, welches beim VNC-Server (Raspberry Pi) festgelegt wurde, einzugeben.

Nach den Betätigung des *Connect*-Buttons erfolgt zunächst die Abfrage des Passworts (Abbildung 4.27), woraufhin der LXDE-Desktop (Abbildung 4.28), der sich

mit PC-Tastatur und PC-Maus bedienen lässt, auf dem PC-Monitor dargestellt wird.

Abbildung 4.28: Der LXDE-Desktop per TightVNC auf einem Windows-PC

Damit Tightvncserver nicht nach jedem Neustart des Raspberry Pi erst manuell aufgerufen werden muss, sondern gleich automatisch startet, ist ein separates Shell Script (Init Script, z.B. als vncserver) notwendig, welches mit dem Programm *insserv* aktiviert wird, indem es den dazugehörigen Link in einem Runlevel-Verzeichnis (z.B. `etc/init.d/vncserver`) setzt.

4.8.9 File Transfer Protocol – FTP

Die bisher betrachteten Verfahren dienen in erster Linie dazu, einen Computer fernzusteuern. Auch wenn es mit einigen dieser Programme (WinSCP) möglich ist, Daten zwischen Computern hin und her zu kopieren, soll hier noch kurz auf das *File Transfer Protocol* (FTP) eingegangen werden, welches die Basis für diese Funktionalität bietet. Ist FTP, bzw. die sichere Implementierung davon, SFTP, auf dem Raspberry Pi (als Server) installiert, kann hierauf mit den verschiedenen FTP-Clients unterschiedlicher Betriebssysteme zugegriffen werden, was den Daten-

transfer zwischen Computern sehr vereinfacht und dies mit akzeptablen Datenraten. FTP-Programme für Linux und damit auch für den Raspberry Pi gibt es eine ganze Reihe, wobei hier stellvertretend der *Very Secure FTP Daemon* (vsftpd) eingesetzt wird. Die Installation erfolgt dabei auf die übliche Art und Weise. Wie immer empfiehlt sich zuvor die Aktualisierung der Paketdatenbank mit:

```
pi@raspberrypi ~ $ sudo apt-get update
pi@raspberrypi ~ $ sudo apt-get install vsftpd
```

Nach der Installation wird der FTP-Server automatisch gestartet, der fortan auch nach jedem Boot aktiv wird, was anhand der Bootmeldung (Starting FTP server: vsftpd) zu erkennen ist.

Abbildung 4.29: FTP-Verbindung mit WinSCP von einem Windows-PCs aus initiieren

Die Konfiguration befindet sich in der Datei `/etc/vsftpd.conf` und kann im Bedarfsfall an die eigenen Wünsche angepasst werden. Falls der übliche Raspberry Pi-Anwender (pi) über einen (Windows-)PC per FTP Zugriff auf sein Verzeichnis auf dem Raspberry Pi haben soll, sind lediglich `local_enable=YES` und `write_enable=YES` notwendig. Optionen für Verschlüsselung und Zertifikate sind ebenfalls in der Konfigurationsdatei einstellbar.

Abbildung 4.30: Problemloser Datenaustausch mithilfe von WinSCP: links das Verzeichnis des Windows-PC, rechts das vom Raspberry Pi

Prinzipiell kann vsftpd in verschiedenen Modi arbeiten, als Daemon oder Stand Alone, wobei die letztere Methode zu bevorzugen ist, was mit listen=YES festgelegt wird. Die wichtigsten Einträge der Datei vsftpd.conf sind im Folgenden gezeigt:

```
# Run standalone? vsftpd can run either from an inetd or as a standalone
# daemon started from an initscript.
listen=YES
#
# Allow anonymous FTP?
# anonymous_enable=YES
#
# Uncomment this to allow local users to log in.
local_enable=YES
#
# Uncomment this to enable any form of FTP write command.
write_enable=YES
#
# Activate directory messages - messages given to remote users when they
# go into a certain directory.
dirmessage_enable=YES
#
# Activate logging of uploads/downloads.
xferlog_enable=YES
#
```

```
# Make sure PORT transfer connections originate from port 20 (ftp-data).
connect_from_port_20=YES
#
# This option should be the name of a directory which is empty. Also,
# the directory should not be writable by the ftp user. This directory is
# used as a secure chroot() jail at times vsftpd does not require
# filesystem access.
secure_chroot_dir=/var/run/vsftpd/empty
#
# This string is the name of the PAM service vsftpd will use.
pam_service_name=vsftpd
#
# This option specifies the location of the RSA certificate to use for
# SSL encrypted connections.
rsa_cert_file=/etc/ssl/private/vsftpd.pem
```

4.9 WLAN

Unter *Wireless Local Area Networks* (WLANs) wird meist eine Implementierung laut IEEE 802.11 verstanden, die auch unter der Bezeichnung Wi-Fi geführt wird. Gegenüber den Kabel-basierten Netzwerklösungen weisen WLANs von der Stabilität und von der Leistung her einige Nachteile auf, doch sie sind auch nicht dazu gedacht, die üblichen LANs zu ersetzen. Vielmehr stellen sie eine Ergänzung oder auch Erweiterung von lokalen Netzen dar. Ein typisches WLAN-Einsatzgebiet ist deshalb überall dort, wo keine Kabel verlegt werden können oder sollen. WLANs sind insbesondere für die Verwendung daheim beliebt, wo man nicht die Wände durchbohren will, um Computer netzwerktechnisch miteinander zu koppeln.

WLANs bieten einen lokalen Funk, typischerweise mit einer Überdeckung von maximal 300 m. Dabei gibt es generell mehrere Unwägbarkeiten, welche die Nutzdatenrate gegenüber kabelbasierten Lösungen weniger vorhersagbar machen. Dazu gehört in erster Linie die Frage, wie sich die zu überbrückende Luftstrecke darstellt, denn Dinge wie Möbel oder Wände stellen sich als Hindernisse dar. Außerdem ist die Datenrate auch von der Entfernung abhängig; sie wird mit größerem Abstand geringer. Die Geräte schalten automatisch auf eine niedrigere Datenrate um.

Allgemein gilt, dass die angegebenen Datenübertragungsraten für WLANs stets als Brutto-Datenraten auf der MAC-Ebene zu verstehen sind. Was davon als Nutzdatenrate – quasi auf IP-Ebene – übrig bleibt, ist von so vielen Faktoren abhängig, dass verlässliche Vorhersagen für die erreichbare Datenrate nicht möglich sind.

Prinzipiell können alle möglichen Geräte, die über einen (integrierten) WLAN-Adapter verfügen, wie PCs, Notebooks, Tablets oder auch Smartphones in einem WLAN miteinander kommunizieren, Voraussetzung ist hierfür, dass sie alle »die gleiche Sprache beherrschen«, d.h., mindestens einen gemeinsamen WLAN-Modus unterstützen. Eine Orientierung hierfür bietet das Wi-Fi-Symbol, mit dem entsprechende kompatible Einheiten gekennzeichnet werden können. Auch wenn ein Gerät nicht explizit mit Wi-Fi gekennzeichnet ist, kann es möglicherweise dennoch mit anderen (Wi-Fi)-Netzwerkeinheiten kommunizieren, was letztendlich im jeweils implementieren WLAN-Standard (siehe Tabelle 4.4) begründet ist.

4.9.1 Standards und Kompatibilität

Der Standard IEEE 802.11 stellt im Grunde genommen die drahtlose Ethernet-Realisierung dar. Für die ersten IEEE-WLANs kommt entweder ein Modulationsverfahren mit der Bezeichnung *Frequency Hopping Spread Spectrum* (FHSS) oder *Direct Sequence Spread Spectrum* (DSSS) zum Einsatz. Letzteres ist ab IEEE 802.11b als Standard anzusehen.

Die darauffolgenden IEEE-WLAN-Standards sehen maximal 54 MBit/s (802.11a/g/h) vor, was im Wesentlichen durch ein verbessertes Modulationsverfahren (OFDM) erreicht wird. Das *Orthogonal Frequency-Division Multiplex-Verfahren*, welches auch unter Multiträger-Modulations-Verfahren (MCM, Multi Modulation Carrier) firmiert, wird quasi von allen neueren Entwicklungen (ADSL, DVB-T, LTE) eingesetzt und benötigt eine verhältnismäßig komplexe Logik für die Umsetzung vom Frequenz- in den Zeitbereich.

Tabelle 4.4: Die wichtigsten Daten der IEEE 802.11-Standards

Standard	Einführung	Frequenzband	Max. Datenrate	Funktechnik
IEEE 802.11	1997	2,4 GHz	2 MBit/s	FHSS, DSSS
IEEE 802.11a	1999	5 GHz	54 MBit/s	OFDM
IEEE 802.11b	1999	2,4 GHz	11 MBit/s	DSSS
IEEE 802.11g	2003	2,4 GHz	54 MBit/s	OFDM
IEEE 802.11h	2003	5 GHz	54 MBit/s	OFDM
IEEE 802.11n	2009	2,4 GHz	600 MBit/s	OFDM-MIMO

Systeme nach IEEE 802.11a arbeiten statt im 2,4 GHz- im 5 GHz-Bereich. Weil dieses Band in Europa von Ortungs- und Satellitenfunk sowie Radarsystemen verwendet wird, erfordert IEEE 802.11a eine europäische Anpassung, was im Standard IEEE 802.11h resultiert.

Geräte nach dem aktuellen Standard IEEE 802.11n sind für eine maximale Datenrate von bis zu 600 MBit/s vorgesehen. Dieser Standard verwendet ebenfalls OFDM und löst die Versionen IEEE 802.11a und IEEE 802.11g ab. Als wesentliche Neuerung wird bei IEEE 802.11n eine (Antennen-)Schaltungstechnik mit der Bezeichnung *Multiple Input Multiple Output* (MIMO) eingesetzt. MIMO kennzeichnet, dass das Signal auf unterschiedlichen »Wegen« empfangen und auch gesendet werden kann, wofür mehrere Antennen zum Einsatz kommen. Wenn mehrere Antennen zusammengefasst arbeiten, ist ein stärkeres Empfangssignal möglich als mit nur einer Antenne, so dass eine Funkverbindung damit verbessert werden kann. Aktuelle Einheiten nach diesem Standard werden mit einer Datenrate von 300 MBit/s bei einem Takt von 20 MHz ausgewiesen. Durch eine Verdopplung auf 40 MHz sind dann die 600 MBit/s prinzipiell möglich, was momentan aber noch nicht erreicht wird.

4.9.2 Topologien

Für den Aufbau von WLANs gibt es mehrere Möglichkeiten, die sich weniger in der jeweiligen Technologie, sondern vielmehr in der Topologie voneinander unterscheiden:

- Adhoc WLAN
- Infrastructure WLAN

Die einfachste Art ist es, wenn zwei oder mehrere Clients mit einem WLAN-Adapter (Wi-Fi) ausgestattet sind und damit direkt miteinander gekoppelt werden können, was zu einem Peer-to-Peer-Netz führt. Die mobilen Geräte wie Notebooks, Tablet-PCs oder auch Smartphones verbinden sich zu einem »vermaschten« Netz, welches keine feste Infrastruktur aufweist.

Abbildung 4.31: Ein WLAN im Adhoc-Modus entspricht einem klassischen Peer-to-Peer-Netz.

Wie es bei einem reinen Peer-to-Peer-Netz üblich ist, werden dabei Ressourcen freigegeben und entsprechend verbunden, was sich meist auf Verzeichnisse oder auch Peripherie-Einheiten wie etwa einen gemeinsam zu verwendenden Drucker bezieht. Mit dieser Methode hat man im Grunde genommen im Nu – daher rührt auch die Bezeichnung für ein derartiges WLAN »adhoc« – ein drahtloses Netzwerk realisiert, wie es sich in kleineren Büros oder auch für daheim anbietet.

Durch die Verwendung eines oder auch mehrerer *Access Points* lässt sich eine Verbindung von WLAN-Clients (Mobile Units) zu einem üblichen LAN schaffen. Auch die Kopplung von separaten LANs ist über Access Points möglich. In der Abbildung 4.32 sind typische Anwendungen hierfür gezeigt. Diese Topologien werden im sogenannten *Infrastructure Mode* betrieben, was derartigen WLANs damit ihre Bezeichnung verleiht.

Abbildung 4.32: WLAN-Realisierungen mit Access Points firmieren unter Infrastructure-Implementierungen.

Der Access Point (AP) fungiert als Bridge zwischen einem Ethernet-LAN und dem Ethernet-WLAN. Er verfügt zumeist über einen Anschluss laut 100BaseT (RJ45) und wird im Prinzip wie ein Switch im LAN integriert. Demnach sind hierfür im

Gegensatz zu einem WLAN-Adapter lokal keinerlei Treiber zu installieren. In einem DSL-Modem ist oftmals neben dem Router und dem Switch auch ein WLAN Access Point mit integriert, wie es etwa bei den bekannten FRITZ! Box-Geräten der Firma AVM der Fall ist.

4.9.3 Raspberry Pi für das WLAN konfigurieren

Der Raspberry Pi kann nachträglich mit einem WLAN-Adapter ausgestattet werden, der an eine USB-Buchse anzuschließen ist. Wie für andere Raspberry Pi-Komponenten und -Peripherie auch, werden im Internet verschiedene Kompatibilitätslisten geführt, die ausweisen, welcher WLAN-Adapter (Wi-Fi Dongle) hierfür geeignet ist.

Die WLAN-Unterstützung ist bei den meisten für den Raspberry Pi geeigneten Betriebssystemen bereits im Kernel integriert. In der Abbildung 4.33 ist ein kompatibler Wi-Fi Dongle gezeigt, der explizit für den Raspberry Pi vorgesehen ist, weshalb er unter der Bezeichnung Wi-Pi geführt wird. Er unterstützt die Standards IEEE 802.11b, IEEE 802.11g und IEEE 802.11n (vgl. Tabelle 4.4) mit einer (theoretischen) maximalen Datenrate von 150 MBit/s in den beiden erläuterten Topologien (Adhoc, Infrastrcture).

Abbildung 4.33: Dieser Wi-Fi Dongle ist für den Raspberry Pi spezifiziert.

Nach dem Anschluss des Adapters sollte zunächst – wie bei jedem »neuen« USB-Gerät – durch Anwendung des Befehls *lsusb* (vgl. Kapitel 4.6) festgestellt werden, ob das Gerät erkannt worden ist. Bei direkter Unterstützung durch den Kernel ist keine separate Treiberinstallation notwendig.

Andernfalls ist der passende Linux-Treiber zunächst aus dem Internet (von der Herstellerseite) zu beziehen und per FTP oder SD-Karte für den Raspberry Pi zugänglich zu machen. Je nach Adapter und Hersteller existiert ein bestimmtes Installationsskript (*.sh), welches aufzurufen ist, damit die Software für den Adapter installiert wird. Wie beim LAN-Adapter (siehe Kapitel 4.8.4) landen die Einträge in der Datei `/etc/network/interfaces`, deren Inhalt entsprechend angepasst werden kann. Abbildung 4.22 zeigt die vorgegebenen WLAN-Adaptereinstellungen, die unmittelbar für das Funktionieren des Adapters mit DHCP sorgen. Wie für den LAN-Adapter könnte hierfür auch eine manuelle IP-Adressenvergabe erfolgen.

Auf dem LXDE-Desktop gibt es die Applikation *WiFi Config*, mit deren Hilfe die Konfigurierung des WLANs für den Raspberry Pi bequem zu erledigen ist. Unter *Adapter* (Abbildung 4.35) sollte ein wlan0-Eintrag erschienen. Durch die Betätigung von *Scan* wird die Umgebung nach vorhandenen WLANs abgesucht.

Abbildung 4.34: Der Adapter wlan0 wurde detektiert.

Abbildung 4.35: Es wurde ein WLAN mit der Bezeichnung 4Home2Go gefunden.

Die gefundenen WLANs werden daraufhin in einem separaten Fenster (Abbildung 4.35) angezeigt. Durch einen Klick auf den jeweiligen Eintrag erscheint das Konfigurationsfenster für den WLAN-Zugang. Hier sind die jeweiligen Parameter des WLANs einzutragen, die von der Konfigurierung des *WLAN Access Point* (DSL-

Modem/Router/Switch) her bekannt sind. Üblicherweise werden die meisten Parameter automatisch erkannt (Abbildung 4.36), so dass dann nur das Passwort anzugeben ist, was hier unter PSK (Pre Shared Key) notwendig ist. Der Wi-Fi Dongle unterstützt alle aktuellen Verfahren für Verschlüsselungen und Authentifizierungen, wie *Wi-Fi Protected Access 2* (WPA) mit dem *Temporal Key Integrity Protocol* (TKIP).

Abbildung 4.36: Festlegen des WLAN-Zugangs

Die festgelegten Parameter können mit *Save* abgespeichert werden. Die Verbindung mit dem WLAN wird dann per *Connect* (Abbildung 4.37) hergestellt. Nachdem der Raspberry Pi mit dem WLAN verbunden ist, erhält er seine IP per DHCP vom *WLAN Access Point* (DSL-Modem/Router/Switch) im Netzwerk.

Es empfiehlt sich, den Raspberry Pi entweder per LAN oder per WLAN netzwerktechnisch zu koppeln, d.h. das LAN-Kabel beim Einsatz des WLAN abzuziehen, andernfalls können sich unübersichtliche Konstellationen ergeben, was bis hin zum Nichtfunktionieren des Netzwerkes führt.

Abbildung 4.37: Der Raspberry Pi ist mit dem WLAN verbunden.

Der Raspberry Pi kann nunmehr auch per WLAN in das Internet gelangen und mit dem ab Kapitel 4.8.6 erläuterten Tools Daten mit anderen Computern austauschen. Eine einfache Kontrolle, ob dies prinzipiell möglich ist, kann wieder mithilfe des Befehls *ping* (Kapitel 4.8.2, Abbildung 4.24) erfolgen. Wie bereits erwähnt, ist der Raspberry Pi in einem Windows-Netz nur dann (direkt) sichtbar, wenn er das Samba-Paket verwendet, was für die einfache Kommunikation untereinander und für den Datenaustausch mit dem Raspberry Pi überdimensioniert erscheint.

Eine Kontrolle der vorhandenen Netzwerkadapter ist wieder durch die Eingabe von `ifconfig` (vgl. Abbildung 4.21) möglich. Genauere Daten zum WLAN, wie der Name (ESSID), die unterstützten Datenraten (Bit Rates) oder auch zur Verschlüsslung (Encrytion) und Authentifizierung (Authentication) des Adapters, sind schnell mit der Eingabe von `iwlist wlan0 scan` (Abbildung 4.38) zu ermitteln.

```
pi@raspberrypi ~ $ iwlist wlan0 scan
wlan0     Scan completed :
          Cell 01 - Address: 00:1D:7E:B5:5D:85
                    Channel:11
                    Frequency:2.462 GHz (Channel 11)
                    Quality=70/70  Signal level=-35 dBm
                    Encryption key:on
                    ESSID:"4Home2Go"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6 Mb/s
                              9 Mb/s; 12 Mb/s; 18 Mb/s
                    Bit Rates:24 Mb/s; 36 Mb/s; 48 Mb/s; 54 Mb/s
                    Mode:Master
                    Extra:tsf=0000000000af0180
                    Extra: Last beacon: 2151295038ms ago
                    IE: Unknown: 000834486F6D6532476F
                    IE: Unknown: 010882848B960C121824
                    IE: Unknown: 03010B
                    IE: IEEE 802.11i/WPA2 Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (2) : CCMP TKIP
                        Authentication Suites (1) : PSK
                    IE: Unknown: 2A0100
                    IE: Unknown: 32043048606C
                    IE: Unknown: DD180050F2020101020003A4000027A4000042435E00623
```

Abbildung 4.38: Anzeige wichtiger WLAN-Daten

5 Programmierung

Der Raspberry Pi ist in erster Linie dafür entwickelt worden, um das Programmieren zu erlernen und den Kontakt zur Hardware herzustellen. Programmiersprachen gibt es in vielen unterschiedlichen Ausprägungen und für verschiedene Plattformen (Betriebssysteme). Diese haben meist ein bevorzugtes Einsatzgebiet (Internet, Datenbank, Office-Anwendungen, Hardware-Programmierung). Sie sind also für bestimmte Dinge besonders gut geeignet, für andere hingegen weniger bis gar nicht. Außerdem stellen der Zugang und der Umfang der Programmiersprache sowie die Programmierumgebung unterschiedliche Ansprüche an die Vorkenntnisse des Programmierers, so dass eine beträchtliche Zeit verstreichen kann, bis mit der jeweiligen Programmiersprache erste Ergebnisse zu verzeichnen sind.

Entsprechend der Intention der Raspberry Pi Foundation soll der Einstieg in die Programmierung für eine möglichst große Anwendergruppe einfach sein und der Schwierigkeitsgrad dementsprechend niedrig. Die Erfahrung zeigt, dass Programme im Laufe der Zeit quasi aus der Applikation heraus fast automatisch immer umfangreicher und komplexer werden, so dass das Programmierumfeld selbst eigentlich keine unnötigen Hürden aufbauen sollte.

Im Gegensatz zu Windows, welches ohne die grafische Oberfläche gar nicht denkbar ist, ist aus den bisherigen Erläuterungen zu Linux sicher erkennbar, dass hier ein Arbeiten im Textmodus (Terminalmodus) möglich ist, wenn man auf den Komfort eines Desktops (LXDE) verzichten kann, zumal bestimmte Dinge viel einfacher und schneller im Textmodus zu erledigen sind. Bereits aus diesem Grunde wirkt eine Programmiersprache für Linux im Textmodus nicht derart »erschlagend« wie es etwa bei den Visual Studio-Sprachen von Microsoft der Fall ist. Es werden dabei weit weniger Ressourcen beansprucht, was für den Raspberry Pi schließlich von Bedeutung ist.

Die klassische Programmiersprache für Linux ist C, die gemeinhin als sehr »hardware-nah« gilt, so dass es nicht verwundert, dass Betriebssystemen wie Linux und auch Windows umfangreicher C-Code zugrunde liegt. Weil es C in zahlreichen unterschiedlichen Implementierungen gibt, die im Großen und Ganzen auf einem Standard-C basieren (ANSI C), handelt es sich um eine der wenigen »Hardware-nahen« Programmiersprachen, die prinzipiell Plattform-übergreifend eingesetzt werden können. Die »Besonderheiten« der jeweiligen Hardware (Mikrocontroller, PC, DSP) werden in die Programme über Bibliotheken mit eingebunden, die nicht selten von den Chipherstellern selbst zur Verfügung gestellt werden, wie es etwa bei Mikrocontrollern und digitalen Signalprozessoren (DSP) üblich ist.

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1_6, © Springer Fachmedien Wiesbaden 2015

Position May 2013	Position May 2012	Delta in Position	Programming Language	Ratings May 2013	Delta May 2012
1	1	=	C	18.729%	+1.38%
2	2	=	Java	16.914%	+0.31%
3	4	↑	Objective-C	10.428%	+2.12%
4	3	↓	C++	9.198%	-0.63%
5	5	=	C#	6.119%	-0.70%
6	6	=	PHP	5.784%	+0.07%
7	7	=	(Visual) Basic	4.656%	-0.80%
8	8	=	Python	4.322%	+0.50%
9	9	=	Perl	2.276%	-0.53%
10	11	↑	Ruby	1.670%	+0.22%
11	10	↓	JavaScript	1.536%	-0.60%
12	12	=	Visual Basic .NET	1.131%	-0.14%
13	15	↑↑	Lisp	0.894%	-0.05%
14	18	↑↑↑↑	Transact-SQL	0.819%	+0.16%
15	17	↑↑	Pascal	0.805%	0.00%
16	24	↑↑↑↑↑↑↑↑	Bash	0.792%	+0.33%
17	14	↓↓↓	Delphi/Object Pascal	0.731%	-0.27%
18	13	↓↓↓↓↓	PL/SQL	0.708%	-0.41%
19	22	↑↑↑	Assembly	0.638%	+0.12%
20	20	=	Lua	0.632%	+0.07%

Abbildung 5.1: Beliebtheit von Programmiersprachen (TIOBE Software) © TIOBE Software

In der Abbildung 5.1 ist ein Ranking der beliebtesten Programmiersprachen angegeben, welches von der Firma TIOBE Software ermittelt wird. Die Kriterien für das Ranking ergeben sich aus verschiedenen Befragungen und Internet-Recherchen, die monatlich veröffentlicht werden. Dabei sagt die Liste nichts Direktes darüber aus, wie viele Code-Zeilen damit erstellt worden sind und auch nicht, dass C die »beste« Programmiersprache ist, die es ohnehin nicht geben kann, weil die bevor-

zugten Einsatzgebiete hierfür recht unterschiedlich sein können. Gleichwohl rangiert C mit seinen Derivaten wie C++, C# und Objective C stets auf den oberen Rängen, so dass ein (angehender) Programmierer eigentlich kaum an dieser Programmiersprache vorbeikommt.

Für Linux existiert für fast jede Programmiersprache ein passendes – kostenloses – Pendant. Die mit C oder der objektorientierten Variante C++ erstellten Programme können sehr schnell ausgeführt werden und sind auch für rechenintensive Aufgaben geeignet. Sie werden mit einem Editor erstellt und mit einem Compiler in Maschinensprache übersetzt, woraus sich auch die relativ schnelle Ausführung ergibt. Allerdings gilt C nicht als Einsteigerprogrammiersprache. Linux liefert standardmäßig einen C-Compiler (gcc) mit und verschiedene Editoren, was als Grundgerüst für die Programmierung im Textmodus bereits ausreichen kann. Im Kapitel 5.7 wird hierauf näher eingegangen.

5.1 Hardware-nahe Programmierung

Neben C sind in der Vergangenheit insbesondere Assembler und Pascal – als Turbo Pascal – für die Programmierung von Hardware eingesetzt worden. Beide Sprachen liefern Programme, die in punkto Effizienz und Geschwindigkeit geradezu als vorbildlich zu bezeichnen sind.

5.1.1 Assembler

Ein Prozessor kann lediglich Zahlen in Befehle umsetzen und sie daraufhin ausführen. Prozessoren kennen – je nach Typ – eine unterschiedliche Anzahl von so genannten *Maschinenbefehlen* (Opcodes), die intern auf ganz bestimmten Zahlen abgebildet werden und dann die gewünschte Funktion im Prozessor hervorrufen.

Neben den Zahlen, die für bestimmte Befehle stehen, rechnet ein Prozessor selbstverständlich auch mit Zahlen, die beispielsweise im Speicher abgelegt und über eine Adresse zu selektieren sind, was letztendlich auch wieder Zahlen entspricht. Als Programmierer wird man jedoch keine unübersichtlichen Zahlenfolgen eingeben wollen, wie es in den Anfängen der Prozessortechnik üblich war.

Stattdessen lässt sich ein Programm unter Verwendung von *mnemonischen Codes* oder *Mnemonics* mit einem Editor erstellen. Die nachfolgende Übersetzung in ein ablauffähiges Programm erfolgt mit einem *Assembler*, weshalb derartige Programme auch als Assembler-Programme bezeichnet werden. *Mnemonische Codes*, etwa MOV oder ADD, sind Kurzbezeichnungen für bestimmte Prozessorfunktionen, wie in diesem Fall für das Verschieben oder Addieren. Sie dienen nur der vereinfachten Programmierung, denn statt eines ADD-Befehls könnte hierfür auch der dazugehörige Opcode von beispielsweise 83h angegeben werden. Der Additions-

befehl wird demnach für den Prozessor durch den Opcode oder Maschinenbefehl von 83h dargestellt.

Wie die Zahlenfolgen durch den Prozessor zu interpretieren sind, was also ein Opcode und was eine Adresse ist, wird durch ein *Machine Instruction Format* (MIF) festgelegt, welches wie die Opcodes bzw. der korrespondierende Maschinenbefehl speziell für eine bestimmte Prozessorfamilie ausgelegt ist. Ein Assembler-Befehlssatz für Intel-Prozessoren ist demnach völlig anders als etwa für einen Mirocontroller der Firma Microchip oder für einen ARM-Prozessor.

Abbildung 5.2: MASM32 ist ein 32-Bit-Assembler für Windows

Bei der Programmierung in einer Hochsprache wie etwa C/C++ oder Visual Basic wird das Quellprogramm durch ein Programm mit der Bezeichnung *Compiler* in die Maschinensprache übersetzt. Eine vielleicht recht einfach und zudem verständlich erscheinende Anweisung in einer Hochsprache hat typischerweise eine Vielzahl von Maschinenbefehlen zur Folge, was beispielsweise mit einem Debugger oder einem Disassembler kontrolliert werden kann, der für die bessere Lesbarkeit (wieder) den Assembler-Code eines Programms darstellen kann.

Die Übersetzung von Assembler-Code in die jeweilige Maschinensprache hat im Vergleich zu Hochsprachen – die die gleiche Aufgabe wie das Assembler-Programm erfüllen sollen – grundsätzlich den effizienteren Code zur Folge, der daher sehr schnell und mit geringem Speicherbedarf ausgeführt werden kann. Nur ein Assembler, der von der Funktion her letztendlich ja auch ein Compiler ist, kann

Programme generieren, die eine sehr enge und unmittelbare Beziehung zur Hardware mithilfe eines Maschinen-Codes herstellen.

DaveSpace › ARM Assembly Language › Introduction to ARM › Movement Instructions - Windows Internet Explorer

http://www.davespace.co.uk/arm/introduction-to-arm/movement.html

Datei Bearbeiten Ansicht Favoriten Extras ?

DAVESPACE THE PERSONAL WEBSITE OF DAVID THOMAS.

SOFTWARE FOR RISC OS | ARM ASSEMBLY LANGUAGE | RANDOM PYTHON SCRIPTS

Rationale
Why Learn Assembly?
Not A Trivial Mapping

ARM Assembly Language
Instruction Sets
Registers
Program Counter
Instruction Syntax
Organisation
Movement
Arithmetic
Logical
Compare
Barrel Shifter
Operand2
Immediate Values
Branch
Conditional Execution
Multiply
Data Transfer
Addressing Modes
Multiple Register Data Transfer
The Stack
A Call Chain

Worked Examples

Introduction to ARM

Movement Instructions

```
<operation>{cond}{S} Rd,<Operand2>
```

<operation>

- `MOV` – *move*
  - Rd := Operand2
- `MVN` – *move negated*
  - Rd := 0xFFFFFFFF EOR Operand2

Examples

- `MOV r0, #42`
  - Move the constant 42 into register R0.
- `MOV r2, r3`
  - Move the contents of register R3 into register R2.
- `MVN r1, r0`
  - R1 = NOT(R0) = -43
- `MOV r0, r0`
  - A NOP (no operation) instruction.

Abbildung 5.3: Die erste Anlaufstelle für die ARM-Assembler-Programmierung ist davespave.co.uk. © davespave.co.uk

Hochsprachen sind demgegenüber für die Lösung bestimmter Probleme auf höheren Anwendungsebenen konzipiert. Gleichwohl ist es auch möglich, in Hochsprachenprogrammen Assembler-Code zu erzeugen und zu integrieren, wie es mit aktuellen C-Versionen (Visual C++) praktiziert werden kann, die über so genannte Inline-Assembler verfügen. Assemblerbefehle können dann beispielsweise inner-

halb eines C-Programms genutzt werden, um direkt auf die Hardware zuzugreifen (Prozessor, Schnittstellen) oder auch um zeitkritische Routinen auszuführen.

Für den Raspberry Pi selbst existiert kein spezieller Assembler. Stattdessen aber für die ARM-Architektur im Allgemeinen und für den im BCM2835 enthaltenen ARM1176JZ-F-Prozessorkern im Besonderen. Im gcc-Compiler kann im Übrigen auch ein ARM-Inline Assembler eingesetzt werden.

5.1.2 Turbo Pascal

Turbo Pascal der Firma Borland ist die bekannteste Implementierung der Programmiersprache Pascal für Personal Computer, die sich ab den achtziger Jahren großer Beliebtheit erfreute. Sie wurde in den neunziger Jahren in Borland Pascal umbenannt und ist in Delphi aufgegangen, welches nunmehr im Besitz der Firma Embaccadero ist. Mit Delphi begann der Übergang von DOS-Programmen zu Windows-Programmen mit der Unterstützung der entsprechenden Microsoft-Technologien (WIN32 API, .NET), die teilweise durch eigene Implementierungen ersetzt wurden.

Eine spezielle Delphi-Version für Linux wurde 2001 unter der Bezeichnung Kylix veröffentlicht, so dass ab diesem Zeitpunkt beide Plattformen unterstützt wurden. Das aktuelle Delphi XE4 verzichtet auf Linux und ist für die Erstellung von Windows-, iOS und MacOS-Applikationen vorgesehen.

Abbildung 5.4: Die Turbo Pascal-Entwicklungsumgebung

Wegweisend war damals die Turbo Pascal-Entwicklungsumgebung (IDE) unter DOS, wobei der Compiler eine bis dato unerreichte Geschwindigkeit aufwies und das gesamte Paket aus Editor, Linker und Compiler eine Größe von lediglich ca. 60 kByte. Der dabei erzeugte echte Maschinencode hatte fast die Effizienz von Assembler und ermöglichte ein recht komfortables Programmieren von Hardwarenahen-Anwendungen.

Eine Besonderheit von Turbo Pascal ist die Tatsache, dass der Code mithilfe von *Prozeduren* und *Units* sehr gut modularisiert werden kann, weil damit häufig benutzte Funktionen ausgelagert werden können und der Code in den Units nur ein einziges Mal übersetzt werden muss. Mitunter findet man im Internet – auch auf der Seite von Embaccadero – die Original Turbo Pascal-Versionen als Freeware.

Als die Firma Borland die Entwicklung von Turbo Pascal einstellte, wurde die Entwicklung von freien, kompatiblen Compilern gestartet, wobei die Versionen *GNU Pascal* und insbesondere *Free Pascal* recht verbreitet sind. Free Pascal (`http://www.freepascal.org/`) ist für zahlreiche Architekturen (Intel x86, PowerPC, ARM) und dementsprechend Plattformen wie DOS, Windows, MacOS und Linux verfügbar, so dass es auch mit dem Raspberry Pi genutzt werden kann.

5.2 Skriptsprachen

Für die Automatisierung von Verwaltungsaufgaben unter Linux können so genannte Bash-Scripts eingesetzt werden. Dies ist im Prinzip nichts anderes als eine Batch-Datei, wie man sie vielleicht von DOS oder Windows her kennt (autoexec.bat), mit der einzelne Befehle im Script der Reihe nach abgearbeitet werden. Die entsprechende Unterstützung ist bei Linux bereits automatisch enthalten. Wie ein einfaches Bash-Script aussehen kann, welches nur den Text »So isses« ausgibt, ist im Folgenden gezeigt.

`pi@raspberrypi ~ $ which bash`	Feststellen wo sich der Bash-Interpreter befindet
`pi@raspberrypi ~ $ #!/bin/bash`	Skript mit Interpreter starten
`pi@raspberrypi ~ $ STRING="So isses"`	String definieren
`pi@raspberrypi ~ $ echo $STRING`	String ausgeben

```
So isses
```

Was hier ganz simpel direkt eingegeben wurde, wird üblicherweise mit einem Editor erstellt, damit abgespeichert und daraufhin als Skript durch den Aufruf des Namens gestartet. Wichtig ist dabei stets der Verweis auf den Interpreter. Die Bash-Programmierung scheint sich immer größerer Beliebtheit zu erfreuen, denn sie belegt in der Liste den Platz 16 mit steigender Tendenz, wobei hiermit eher

keine Programme im klassischen Sinne, sondern Verwaltungsabläufe automatisiert werden.

Die Programmiersprache Perl (Platz 9) ist eine mächtige Skriptsprache, mit der sich umfangreiche Aufgaben erledigen lassen. Perl ist wie Bash-Script automatisch bei Linux und damit auch bei Raspbian für den Raspberry Pi vorinstalliert. Die erste Zeile muss auch hier eine Verknüpfung zum Perl-Interpreter enthalten (#!/usr/bin/perl).

PHP5 (Personal Home Page) ist ebenfalls eine Skriptsprache. Sie verwendet eine Syntax, die wie eine Kombination aus Perl und C wirkt. Die Programmierung wird entweder mit Funktionen oder mit einem objektorientierten Ansatz ausgeführt, wobei auch beide Konzepte gemischt werden können. PHP ist insbesondere für Webanwendungen gedacht und bietet eine Unterstützung für verschiedene Internet-Protokolle sowie Datenbanken. Wer mit dem Raspberry Pi einem Webserver (Apache) betreiben will, kommt an PHP nicht vorbei, welches separat (`sudo apt-get install php5-cli`) zu installieren ist. PHP5 ist zwar eine mächtige Skriptsprache, die Ausführung dauert jedoch – weil das Skript auch hier durch den dazugehörigen Interpreter ausgeführt werden muss – verhältnismäßig lang. Deshalb eignet sich PHP5 auch kaum für andere Zwecke als für Internet-Anwendungen.

5.3 Java

Java ist eine plattformübergreifende, objektorientierte Sprache mit einem riesigen Funktionsumfang. Die Applikationen laufen in einem *Java Runtime Environment* (JRE), also einer für das jeweilige Betriebssystem bestimmten speziellen Laufzeitumgebung. Java kann von der Firma Sun (jetzt Oracle) kostenlos bezogen werden, wozu neben der JRE hierzu passende Entwicklungswerkzeuge (Java Development Kit, JDK) verfügbar sind. Dazu gehört der Java-Compiler, der den Quelltext in einen Java-Byte-Code übersetzt, welcher daraufhin in einer virtuellen Umgebung (JRE) ausgeführt wird, was letztlich für die Plattformunabhängigkeit verantwortlich ist.

Da es Java für Linux gibt, kann es auch für den Raspberry Pi eingesetzt werden, welches wie folgt zu installieren ist. Zunächst das JDK:

```
pi@raspberrypi ~ $ sudo apt-get install openjdk-7-jdk
```

gefolgt von der Laufzeitumgebung:

```
pi@raspberrypi ~ $ sudo apt-get install openjdk-7-jre
```

Die Ressourcenanforderungen für Java sind recht hoch, so dass es für ARM-basierte Embedded Systeme und als Ableger damit auch für den Raspberry Pi eine *Java Embedded Micro Edition* (Java ME) gibt. Java arbeitet dann nicht mit seiner

Laufzeitumgebung unter einem Betriebssystem, sondern quasi direkt mit der jeweiligen Hardware. Hiermit können dann Java-basierte Embedded-Systeme entworfen werden, die für ganz bestimmte Aufgaben (Interface, Messsystem, Funkknoten, Regler) vorgesehen sind, was eine Fülle von neuen elektronischen Applikationen mit dem Raspberry Pi eröffnet. In vielen Geräten, wie beispielsweise Autoradios, Druckern oder Routern, arbeitet eine solche Java Embedded-Version.

5.4 Microsofts .NET

Microsofts Reaktion auf Java ist .NET, wo mithilfe von verschiedenen Sprachen (C#, VB.NET, ASP.NET) ein Programm erstellt werden kann. Dieses wird per Compiler in einen Byte-Code (Common Intermediate Language) umgesetzt, der dann in einer Laufzeitumgebung – dem *Microsoft .NET Framework* – abläuft. Im Gegensatz zu Java steht .NET lediglich für Windows zur Verfügung. Es ist vorgesehen, dass die Entwicklung mit den Microsoft-Programmiersprachen erfolgt. Für bestimmte Mikrocontroller und -prozessoren, wie sie in Embedded Systems sowie Smartphones und Tablets eingesetzt werden oder auch für die Gadgeteer-Plattform gibt es zwar das *.NET Micro Framework*, gleichwohl ist auch auf diesem Einsatzgebiet Java mit den entsprechenden Ergänzungen, etwa dem Android SDK, die führende Technologie.

5.4.1 Mono

Um Programme, die mit .NET erstellt worden sind, auch unter Linux ausführen zu können, ist freie Software verfügbar, die eine Implementierung des Windows .NET Frameworks darstellt. Dabei handelt es sich um *DotGnu* und die bekanntere mit der Bezeichnung *Mono*. Somit ist es prinzipiell möglich, in C unter Linux zu programmieren und die Applikation dann auch unter Windows auszuführen oder in C# unter Windows zu programmieren und die Applikation unter Linux auszuführen. Eine sicher interessante Möglichkeit, um mit Microsoft-Technologie eine gewisse Plattformunabhängigkeit zu erreichen, ob dies allerdings den Aufwand (gegenüber Java) rechtfertigt, zumal Kompatibilitätsprobleme dazugehören (die es bekanntermaßen aber auch innerhalb der Microsoft .NET Technologie gibt) hängt sehr stark von der jeweiligen Anwendung ab und wird sich in den meisten Fällen nicht lohnen.

Mono hat sich nicht wie gehofft durchgesetzt, was auch daran liegen mag, dass es auf den GNOME-Desktop (und nicht auf KDE oder LXDE oder andere) zugeschnitten ist. Erfolgreicher ist der Ableger MonoTouch, mit dem Applikationen

erstellt werden können, die sowohl unter Android als auch unter iOS ablauffähig sind.

5.5 Standard Tools auf dem Desktop

Auf dem LXDE-Desktop befinden sich bereits standardmäßig einige Tools für die Programmierung. Für die jüngsten Anwender (ab acht Jahre) ist *Scratch* vorgesehen, eine Programmierumgebung, bei der kein Quelltext zu verfassen ist, sondern Code quasi per Drag-and-Drop generiert wird. Scratch wird direkt per Klick vom Desktop aus gestartet, wobei eine lachende orange Katze das zentrale Element darstellt.

Scratch gliedert sich in vier Bildschirmabschnitte: Rechts die Stage (Bühne) mit der Katze, wo die Animation oder das Spiel abläuft. Darunter die Sprite List, welche zunächst nur die Katze enthält. Sprites bilden bei Scratch die Elemente (Charaktere), die dargestellt und auf der Bühne bewegt werden können.

Abbildung 5.5: Die Scratch-Umgebung

Das Programm wird durch das Zusammenfügen von Blöcken (Snapping) erstellt, die kurzen Instruktionen entsprechen. Ganz links ist die dazugehörige Block-Palette positioniert, die die Bewegungen beschreiben und auslösen. In der Mitte

befindet sich die Script Area, in der das Programm mithilfe der Blocks erstellt wird. Scratch ist am MIT-Forschungsinstitut im Jahre 2006 entwickelt worden und ermöglicht einen spielerischen Zugang zum Programmieren und das Erstellen einfacher Spiele wie »Katze jagt Maus« mit Geräuschen und Musik, wie sie ab den siebziger Jahren unter der Bezeichnung Arcade Games gebräuchlich waren.

Abbildung 5.6: Ein Spaßmusikprogramm mit Scratch für die Nachwuchsförderung von der Hochschule in Wismar © Hochschule Wismar

Die Programmiersprache Python gehört zu den beliebtesten überhaupt. Python ist von der Raspberry Pi Foundation als *die* Programmiersprache für den Raspberry Pi ausgesucht worden und hat auch zur Namensgebung (Pi = Py) des Systems beigetragen. Der Name *Python* selbst ist in Anlehnung an die Komikergruppe Monty Python gewählt worden, was zeigt, dass man mit Spaß dabei ist und vielleicht nicht alles so ernst (streng) zu nehmen ist.

Python ist deswegen so beliebt, weil es sehr einfach zu erlernen ist, was zunächst daran liegt, dass die Programmierbefehle durch relativ wenige einfache, verständliche (englische) Ausdrücke gebildet werden. Außerdem steht Python nicht nur für Linux, sondern auch für Windows und MacOS zur Verfügung. Diese Flexibilität

wird noch dadurch erhöht, dass mit Python auf Bibliotheken (Libraries) mit C- oder C++-Code zugegriffen werden kann.

Auf dem LXDE-Desktop sind zwei Versionen der Python Entwicklungsumgebung (Python Shell) zugänglich, IDLE und IDLE 3. Mit *Idle* wird in der Prozessortechnik ein Leerlaufprozess bezeichnet, was hiermit jedoch nicht gemeint ist, sondern die IDE (Integrated Development Environment), also die Entwicklungsumgebung.

Abbildung 5.7: Für den Einstieg in die Programmierung ist bereits gesorgt.

Neben den beiden Python Shells ist noch *Python Games* auf dem LXDE-Desktop zu finden. Dahinter verbergen sich einige simple Spiele (Arcade Games), die mit einer speziellen Python-Bibliothek (Pygame) erstellt worden sind, wobei sich einige Pygame-Module (siehe folgendes Kapitel) durchaus für andere Zwecke nutzen lassen, denn sie ermöglichen den einfachen Zugriff auf das Display und die Nutzung von Sound, Bildern, Filmen (mpeg) und bieten ein Control Timing.

5.6 Programmieren mit Python

Wie erwähnt, sind bei Raspbian zwei Python-Versionen (IDLE, IDLE3) vorinstalliert. Dabei handelt es sich um die Versionen 2.7 (IDLE) und 3.2 (IDLE3), die nur bedingt zueinander kompatibel sind, weshalb beide Versionen vorhanden sind. Python 3 existiert bereits seit dem Jahr 2008. Die Unterschiede zur Vorgängerversion sind prinzipiell – aus der Sicht des Programmierers – nicht besonders groß, allerdings sind zahlreiche Bibliotheken (Add-ons, Libraries) für die 2.x-Versionen ausgelegt, die nicht mit der aktuellen Version funktionieren, so dass viele Programme auch weiterhin mit der älteren Python-Version erstellt werden.

Da Python ebenfalls für Mac OS und Windows verfügbar ist, können Programme auch auf einer dieser Plattformen entwickelt werden. Dabei sind auftretende Inkompatibilitäten möglich, was insbesondere den unterschiedlichen Bibliotheken für 32-Bit-Windows und 64-Bit-Windows geschuldet ist.

In der Abbildung 5.8 ist zu erkennen, dass hier die Version 2 und die Version 3 aufgerufen worden sind. Wie bei anderen Programmiersprachen kann auch bei Python mithilfe des Print-Befehls eine Zeichenkette (String) ausgegeben werden, die in Anführungszeichen zu setzen ist (bei Python sind sowohl »'« als auch »"« hierfür erlaubt), was mit der Version 2 sofort funktioniert, während bei Version 3 eine Fehlermeldung (Syntax Error) auftritt.

Beim Anklicken der Programmzeile erscheint automatisch eine kontextsensitive Hilfe, die angibt, dass der Print-Befehl hier eine Klammer verlangt. Nach dieser Änderung funktioniert dann auch die Ausgabe mit der Version 3. Ähnlichen Unterschieden bei der Python-Programmierung begegnet man auch bei anderen Befehlen, wobei sich die Probleme meist leicht mithilfe der praktischen Hilfefunktion lösen lassen.

Abbildung 5.8: Bei der Ausgabe mit dem Print-Befehl gibt es einen Unterschied zwischen der Python-Version 2 und der -Version 3.

Python ist eine Interpreter-Sprache, die während der Laufzeit mithilfe des benötigten Interpreters ausgeführt wird, was direkt in den beiden Python-Shells erfolgen kann. Python kann genauso gut direkt von der Konsole aus oder in einem Terminal durch die Eingabe von *python* gestartet werden, woraufhin der Eingabe-Prompt >>> erscheint und auf Eingaben wartet, die nach Betätigung der Eingabetaste ausgeführt werden. In einer Python-Shell (Abbildung 5.8) ist die Bedienung und Kommunikation einfacher möglich, weil hier die üblichen Befehle des Betriebssystems (zum Kopieren etc.) sowie die Maus aktiv sind und die kontextsensitive Hilfe automatisch zur Verfügung steht.

In der Abbildung 5.8 ist die Eingabe der folgenden Zeile in der Shell der Version 2.7 zu erkennen:

```
print "Hier ist die Version 2.7 aktiv"
```

und in der Shell der Version 3.2

```
print "Hier ist die Version 3.2 aktiv"
```

wobei aufgrund der Inkompatibilität der beiden Versionen beim Print-Befehl ein Fehler auftritt, der durch das Hinzufügen der Klammer bei der Version 3.2 beseitigt wird:

```
print ("Hier ist die Version 3.2 aktiv")
```

Die direkte Eingabe am Python-Prompt (Interactive Mode) ist eigentlich nur für Testzwecke interessant. Deshalb ist für richtige Aufgaben ein Programm zu erstellen. Für das Schreiben eines Python-Programms, welches stets die Endung »py« aufweist, wird ein Editor benötigt, der wie alle anderen notwendigen Tools über die Menüleiste der Python Shell (File, Edit, Shell Debug, Options, Windows, Help) zur Verfügung steht.

Abbildung 5.9: Öffnen des Editors und Schreiben eines einfachen Programms

Prinzipiell kann für das Schreiben eines Python-Programms ein Editor nach Wahl eingesetzt werden. Außerdem gibt es eine Reihe von (komfortableren) Entwicklungsumgebungen (Geany, Stani´s Python Editor), über die das Python-Programm getestet und ausgeführt werden kann. Mit IDLE bzw. IDLE3 stehen jedoch alle wichtigen Funktionen zur Verfügung, so dass zunächst keine Notwendigkeit besteht, weitere Tools zu installieren. Das Programm (Abbildung 5.9) besteht lediglich aus den beiden folgenden Zeilen:

```
#! /usr/bin/env python
```

```
print "Nur ein simpler Test"
```

Nach dem Speichern des Programms (z.B. als simpletest.py), ist es durch das Voranstellen von *python* (Interpreteraufruf) ausführbar, also mit:

```
pi@raspberrypi ~ $ python simpletest.py
```

Dabei ist es von Bedeutung, in welchem Verzeichnis das Programm gespeichert ist und aus welchem Verzeichnis heraus es aufgerufen wird. Der Einfachheit halber wird das Programm beispielsweise dort gespeichert, wo sich auch andere Python-Programme befinden, also unter `/home/pi/python_games`, so dass es auch von hier aus und/oder mit der Angabe des kompletten Pfades ausgeführt werden kann.

Die erste Zeile des Programms (Abbildung 5.9) spezifiziert die Position des Python-Interpreters, weshalb es keiner besonderen Pfadangabe bedarf, sondern stets automatisch lokalisiert werden kann. Diese erste Zeile, die in jedem Python-Programm vorhanden sein sollte und auch als *shebang line* bezeichnet wird, ermöglicht es außerdem, das Python-Programm auch direkt, ohne die separate Angabe des Python-Interpreter-Aufrufs auszuführen.

Hierfür muss die Datei als ausführbar (siehe Kapitel 2.4 Zugriffsrechte) gekennzeichnet sein, d.h., das Dateiattribut, welches zunächst nur schreiben und lesen erlaubt (Eigentümer, Gruppe, Andere), ist um diese Funktionalität zu erweitern, was wie folgt zu lösen ist, woraufhin das Programm gestartet werden kann:

```
pi@raspberrypi ~ /python_games $ chmod +x simpletest.py
pi@raspberrypi ~ /python_games $ ./simpletest.py
```

Damit die Programmausführung aus dem aktuellen Verzeichnis funktioniert, ist »./« voranzustellen. Der lange Verzeichnisname kann dann weggelassen werden.

Um das Programm wie jedes andere auch, welches als Konsolenprogramm (im Terminal) zur Verfügung steht, per Name von einem beliebigen Verzeichnis aus aufzurufen, muss es ebenfalls im Verzeichnis `/user/local/bin` vorhanden sein. Es ist wie folgt dorthin zu kopieren:

```
pi@raspberrypi ~ /python_games $ sudo cp simpletest.py
/user/local/bin
```

Wenn jetzt noch die Extension .py stört, kann das Programm einfach umbenannt und die Extension weggelassen werden (wobei es sich um eine einzige Befehlszeile handelt).

```
pi@raspberrypi ~ $ sudo mv /usr/local/bin/simpletest.py
/usr/local/bin/simpletest
```

Als Beispiel für ein Python-Programm wurde hier tatsächlich nur eine ganz einfache Ausgabe mit dem Print-Befehl realisiert, wobei jedoch alle wesentlichen Schritte absolviert wurden, um das Programm letztendlich wie ein übliches Linux-Programm aussehen und funktionieren zu lassen.

Python verfügt über alle für eine Programmiersprache gebräuchlichen Befehlen und Funktionen. Arrays – wie sie von anderen Sprachen her bekannt sind – gibt es bei Python allerdings nicht. Stattdessen werden hierfür *lists* und *dictionaries* verwendet. Kommentarzeilen werden mit einem vorangestellten »#«-Zeichen als solche kenntlich gemacht und demnach nicht ausgeführt. Bei der Verwendung von Kontrollstrukturen wie bei if-, for-, oder while-Schleifen sowie bei der Definition von Funktionen (mit def) ist eine Einrückung des Codes innerhalb der Struktur notwendig, damit dieser Codeabschnitt, der mit einem Doppelpunkt »:« eingeleitet wird, vom Interpreter als solcher erkannt wird. Hierfür wird üblicherweise die TAB-Taste verwendet, wobei aber auch einzelne Leerzeichen erlaubt sind. Dabei sollte stets die gleiche Anzahl von Leerzeichen für die Strukturierung verwendet werden. Im Folgenden ist ein Beispiel angegeben, welches unter Verwendung der pygame-Bibliothek eine Uhr (Abbildung 5.10) auf dem Bildschirm darstellt.

```
# Uhr mit Python

import time,pygame
pygame.init()

theFont = pygame.font.Font(None,72)
clock = pygame.time.Clock()
screen = pygame.display.set_mode([320, 200])

pygame.display.set_caption('Pi Time')

while True:

      clock.tick(1)
      theTime=time.strftime("%H:%M:%s",time.localtime())
      timeText=theFont.render(str(theTime),
      True,(255,255,255),(0,0,0))
      screen.blit(timeText,(80,60))
      pygame.display.update()
```

Abbildung 5.10: Ausgabe des Beispielprogramms

Hilfe zur Python-Programmierung ist standardmäßig über *man python* oder auch *info python* verfügbar sowie unter */user/share/doc/python/FAQ.html*. Diese html-Datei ist über den Browser (Midori) mit der folgenden Zeile aufrufbar:

```
file:///user/share/doc/python/FAQ.html
```

Abbildung 5.11: Die lokale Datei ist ein guter Ausgangspunkt für die weitere Informationsbeschaffung. © http://www.python.org/

Diese Internetseite kann der Ausgangspunkt für die weitere Informationsbeschaffung zur Python-Programmierung sein, was insbesondere über `http://www.python.org` mit den zahlreichen Ablegern sowie der deutschen Seite unter `http://tutorial.pocoo.org/` möglich ist.

Trotz der Einfachheit von Python stehen Funktionen für die Umsetzung parallel arbeitender Programme (thread) zur Verfügung sowie spezielle Funktionen (socket API) für die Netzwerkkommunikation, was auch professionellen Ansprüchen genügt.

Mit Python ist zudem die objektorientierte Programmierung (class) leicht durchführbar, ohne dass hierfür besonders strenge Konventionen zu beachten und spezielle Anstrengungen zu unternehmen sind, wie es bei anderen Programmiersprachen der Fall ist. Die Möglichkeit, eine Mischung aus konventioneller (funktionaler) und objekt-orientierter Programmierung – auch innerhalb eines Programms – vornehmen zu können, erweist sich in der Praxis als äußerst nützlich und schnell umsetzbar. Im Folgenden ist ein Listing für ein Programm gezeigt, welches wieder die pygame-Bibliothek verwendet und einfache Kreise zeichnet,

wobei die Klasse *circle* (hier nicht angegeben) eingesetzt wird, die die entsprechenden Funktionen für das Zeichnen enthält.

```
# Kreise zeichnen
Import pygame, circle
pygame.int()
clock = pygame. time.Clock() # Clock to limit speed
WIDTH=600; HEIGHT=600
screen = pygame.display.set_mode([width, height])
BLACK=(0,0,0)
screen.fi11(BLACK)
circles=[]
for n in range(lOO):
      clock.tick(45)
      circles.append(circle.Circle(screen,WIDTH,HEIGHT))
      pygame.display.update()
clock.tick(l)
for c in circles:
      clock.tick(45)
      c.clear_circle(screen)
      pygame.di splay.update()
raw_input("Press a key")
```

5.7 Programmieren mit C

Die klassische Programmiersprache für Linux ist C, die aufgrund ihres Umfangs und ihrer Komplexität eher nicht als Einsteigerprogrammiersprache gilt, sondern vielmehr als professionelle Programmiersprache für Systemspezialisten. Mit C und der objektorientierten Variante davon C++ kann sehr systemnah und damit eng mit der Hardware, der Firmware, dem Kernel und dem Betriebssystem gearbeitet werden, was ein tieferes Verständnis über die genauen Zusammenhänge erfordert. Gleichwohl gelingt es auch dem Einsteiger, mit wenigen Code-Zeilen und unter Verwendung der zahlreichen vorhandenen Bibliotheken ein C-Programm zu realisieren, was in diesem Kapitel etwas näher betrachtet werden soll. Es ist wie das vorherige Kapitel über Python als praxisorientierte Übersicht gedacht, um die Programme für die Ansteuerung der Hardware im folgenden Kapitel nachvollziehen zu können.

Die Programme werden mit einem Editor erstellt und mit einem Compiler in Maschinensprache übersetzt, woraus sich die relativ schnelle Ausführung der C-Programme ergibt. Linux liefert standardmäßig einen C-Compiler (gcc) mit, der sowohl mit C- als auch mit C++-Code umgeben kann. Verschiedene Editoren sind ohnehin bei Linux mit enthalten, was als Grundgerüst für die Programmierung im Textmodus ausreicht. Hierfür existieren ebenfalls verschiedene Entwicklungsum-

gebungen, die mit einer benutzerfreundlichen grafischen Oberfläche arbeiten und alle notwendigen Tools integrieren. Wie für Python gilt Geany, welches mit `sudo apt-get install spe` installiert werden kann, als leistungsfähige IDE, die mehrere Templates für verschiedene Programmiersprachen mitbringt und demnach recht universell eingesetzt werden kann.

Abbildung 5.12: Einfache C-Programme können problemlos mit dem Editor nano erstellt werden.

Im einfachsten Fall wird mit dem Editor (z.B. nano, Abbildung 5.12) eine Datei mit dem folgenden Inhalt erstellt:

```
# Hallo in C
#include <stdio.h>
int main ()
{
       printf ("Hallo hier ist das C-Programm\n");
       return 0; //Rueckgabewert;
}
```

Das Programm gibt die Meldung »Hallo hier ist das C-Programm« aus, was mithilfe der printf-Anweisung erfolgt, die außerdem den Cursor mit *\n* in die folgende Zeile springen lässt. Mit *#include* wird der Inhalt der Standardbibliothek (stdio.h), die den printf-Befehl kennt, zum Programm hinzugefügt. Per *int main ()* werden die folgenden Zeilen zum Hauptprogramm definiert, welches mit geschweiften Klammern umschlossen wird. Jede Programmzeile ist mit einem Semikolon abzu-

schließen, was jedoch nicht bei Kontrollstrukturen wie den if-, for-, oder while-Schleifen durchzuführen ist. Kommentare werden üblicherweise mit vorangestelltem »//« gekennzeichnet.

Fehlerfrei ausgeführte Funktionen liefern einen Rückgabewert an die aktive Instanz zurück, der in diesem Fall mit Null (return 0) definiert ist, denn im Main-Klammerausdruck (main()) sind keine Übergabevariablen definiert, wie es typischerweise in Unterprogrammen (Funktionen) praktiziert wird. Strenggenommen, aber eigentlich optional, wird eine derartige »leere Konstruktion« mit (void) gekennzeichnet.

Damit dieses Programm ausführbar wird, ist es (mit gcc) zu kompilieren. Genaugenommen ist gcc jedoch nicht nur ein Compiler, sondern auch ein Linker. Zunächst bewirkt gcc, dass aus der Quelldatei (halloinc.c) eine Objektdatei erzeugt wird, die mit Objektcode aus den Bibliotheken »gebunden wird«, was Aufgabe eines Linkers ist. Im letzten Schritt generiert der Compiler daraus eine ausführbare Datei. Damit dies wie gewünscht funktioniert, ist lediglich die folgende Eingabe notwendig:

```
pi@raspberrypi ~ $ gcc halloinc.c
```

Damit ist das Programm entsprechend umgesetzt worden und kann mit `./a.out` gestartet werden. Falls bei gcc keine Option angegeben wird, wird die kompilierte Datei stets als a.out angelegt. Mit gcc sind eine Vielzahl von Optionen (ca. 600) möglich, die hier – bis auf eine Ausnahme – nicht weiter von Bedeutung sind. Die Ausnahme betrifft die Option -o, die dafür sorgt, dass ein Name für die kompilierte – die ausführbare – Datei angegeben werden kann und nicht die Standardnamenskonvention angewendet wird.

```
pi@raspberrypi ~ $ gcc halloinc.c -o halloinc
```

Bereits aus diesem einfachen Beispiel ist ersichtlich, dass bei einem C-Programm weitaus mehr Konventionen (stets ein Main-Modul, Code mit geschweiften Klammern umschließen, Semikolon am Ende einer Befehlszeile, Rückgabewert definieren) als bei einem vergleichbaren Python-Programm (vgl. Abbildung 5.9) zu beachten sind.

```
pi@raspberrypi ~ $ gcc halloinc.c
pi@raspberrypi ~ $ ./a.out
Hallo hier ist das C-Programm
pi@raspberrypi ~ $ gcc halloinc.c -o halloinc
pi@raspberrypi ~ $ ./halloinc
Hallo hier ist das C-Programm
pi@raspberrypi ~ $
```

Abbildung 5.13: Kompilieren der Quelldatei ohne gcc-Option liefert a.out als ausführbares Programm. Durch die Option -o kann hierfür ein eigener Name gewählt werden.

Der GNU C-Compiler (gcc) ist ein mächtiges Werkzeug, auch wenn es auf den ersten Blick mit dem simplen Kompilierungsaufruf nicht so erscheint. Gcc unterstützt auch andere Sprachen wie Fortran oder Java und stellt auf `http://gcc.gnu.org/` hilfreiche Dokumentationen zur Verfügung, bietet jedoch keine Hilfe oder ein Tutorial für die eigentliche C-Programmierung. Dies wird in erster Linie durch die zahlreichen Fachbücher zur C-Programmierung (ANSI C) geleistet, die teilweise auch als On Line-Ausgaben im Internet zu finden sind.

Im Lieferumfang von Raspbian sind einige Beispielprogramme in C zu finden, die, bis auf hello_world, speziell auf den Raspberry Pi zugeschnitten sind und eine gute Ausgangsbasis für eigene Entwicklungen bieten. Zu finden sind die Quelldateien nebst Bibliotheken und dazugehörigen Make-Files, die die Kompilierung entsprechend steuern (die Kompilierung erfolgt dann durch den Aufruf von *make* im jeweiligen Verzeichnis), unter `/opt/vc/src/hello_pi`.

```
pi@raspberrypi /opt/vc/src/hello_pi $ ls -l
insgesamt 64
drwxrwxr-x 2 root users 4096 Mär 28 19:36 hello_audio
drwxrwxr-x 2 root users 4096 Sep 18  2012 hello_dispmanx
drwxr-xr-x 2 root root  4096 Mär 28 16:47 hello_encode
drwxrwxr-x 2 root users 4096 Sep 18  2012 hello_font
drwxr-xr-x 2 root root  4096 Mär 28 16:47 hello_jpeg
drwxr-xr-x 2 root root  4096 Mär 28 16:47 hello_teapot
drwxrwxr-x 2 root users 4096 Sep 18  2012 hello_tiger
drwxrwxr-x 2 root users 4096 Sep 18  2012 hello_triangle
drwxrwxr-x 2 root users 4096 Sep 18  2012 hello_triangle2
drwxrwxr-x 2 root users 4096 Sep 18  2012 hello_video
drwxr-xr-x 2 root root  4096 Mär 28 16:47 hello_videocube
drwxrwxr-x 2 root users 4096 Sep 18  2012 hello_world
drwxrwxr-x 4 root users 4096 Sep 18  2012 libs
-rw-rw-r-- 1 root users 1125 Mär 28 16:47 Makefile.include
-rw-rw-r-- 1 root users  257 Mär 28 16:47 README
-rwxrwxr-x 1 root users  624 Mär 28 16:47 rebuild.sh
pi@raspberrypi /opt/vc/src/hello_pi $
```

Abbildung 5.14: Die Verzeichnisse mit den Beispielprogrammen in C

6 Schnittstellen-Kommunikation

Der Raspberry Pi verfügt über eine Pfostenleiste (Expansion Header), die in zwei Reihen zu jeweils 13 Pins angeordnet ist und verschiedene Signale führt, wobei einige eine Doppelfunktion aufweisen, wie es im Kapitel 3.12 erläutert ist. Hier sind mehrere GPIO-Pins, eine I²C-, zwei SPI- und eine serielle Schnittstelle vorhanden, um die Kommunikation mit externer Peripherie – der Hardware – zu ermöglichen, was im einfachsten Fall eine Leuchtdiode (Ausgabe) und ein Taster (Eingabe), die jeweils an einen GPIO-Pin angeschlossen werden, bedeutet.

Des Weiteren sind am Anschluss des *Camera Serial Interface* (CSI) vier weitere GPIO-Pins (GPIO 2, 3, 5, 27) zu finden, die ebenfalls für eigene Schaltungen eingesetzt werden können, genauso wie ein durch den fehlenden Ethernet-Chip beim Modell A freigewordener Anschluss (GPIO 6).

Abbildung 6.1: Die gebräuchlichen GPIO-Pins bei den Raspberry Pi-Modellen

Mit den +Modellen (Kapitel 3.11) ist die Pfostenleiste auf 40 Kontakte erweitert worden, wobei die bisherigen 26 Signale kompatibel bleiben. Die (unbestückte) P5-Leiste, die vier weitere I/O-Signale plus 3,3 V, 5 V und Masse führt, gibt es bei den +-Modellen nicht mehr. Genaueres zu den verschiedenen GPIO-Ports ist im Kapitel 3.12 angegeben.

6.1 OnBoard-LED ansteuern

Die Modelle A und B verfügen zwischen der Audioausgangsbuchse und den USB-Anschlüssen über fünf Leuchtdioden. Die PWR-LED (rot) leuchtet, wenn die Versorgungsspannung anliegt, während die grüne ACT-LED beispielsweise dann blinkt, wenn der Bootvorgang von der SD Card stattfindet. Die unteren drei LEDs signalisieren den Netzwerkstatus. Die grüne FDX-LED zeigt an, dass die Übertragung im Full Duplex-Modus stattfindet, die grüne LNK-LED, dass eine Netzwerkverbindung vorhanden ist, und die gelbe 100-LED, dass der Übertragungsmodus von 100 MBit/s (Fast Ethernet) genutzt wird.

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1_7, © Springer Fachmedien Wiesbaden 2015

Bei +Modellen befinden sich links unterhalb der Pfostenleiste lediglich zwei LEDs: PWR und ACT. Für die Signalisierung der Netzwerkaktivität befinden sich nunmehr zwei LEDs (grün, orange) direkt in der RJ45-Buchse.

Abbildung 6.2: Die Leuchtdioden auf verschiedenen Raspberry Pi Boards

Alle Leuchtdioden werden, bis auf die PWR-LED, vom Linux-Kernel gesteuert, wobei sich die ACT-LED (GPIO 16) auch separat ansteuern lässt, so dass dem Anwender und/oder dem Programmierer hiermit eine Status-LED für eigene Einsatzzwecke zur Verfügung steht. Die Steuerung erfolgt über Dateien, die im Verzeichnis `/sys/class/leds/led0` zu finden sind.

6.1.1 Trigger

In der Datei *Trigger* sind die Auslösebedingungen (Trigger-Bedingungen) definiert, die *none* oder *mmc0* lauten, wobei jeweils die in eckigen Klammern gesetzte Option gültig ist. Standardmäßig ist dies *[mmc0]*, was bedeutet, dass die LED beim Zugriff auf die SD-Karte aufleuchtet.

Damit die ACT-LED stattdessen per Software zu steuern ist, ist *none* in der Trigger-Datei zu aktivieren. Hierfür sind root-Rechte notwendig, was prinzipiell auch per *sudo* mit dem Editor *nano* durchgeführt werden kann:

```
pi@raspberrypi ~ $ sudo nano /sys/class/leds/led0/trigger
```

Alternativ kann dies auch direkt erfolgen:

```
pi@raspberrypi ~ $ echo none | sudo tee /sys/class/leds/led0/trigger
```

Wenn die eckigen Klammern nunmehr um *none* gesetzt sind, wird die ACT-LED nicht mehr beim Zugriff auf die SD-Karte leuchten. Nach einem Neuboot gilt allerdings wieder die ursprüngliche Einstellung.

6.1.2 Heartbeat

Zum Funktionstest der LED-Ansteuerung empfiehlt sich ein so genannter *Heartbeat*, d.h. eine Funktion, die kontinuierlich die Aktivität des Betriebssystems signalisiert, was per *ledtrig_heartbeat* und damit über die ACT-LED erfolgen kann. Hier-

für ist zunächst das passende zusätzliche Kernel-Modul notwendig, welches per *modprobe* zu installieren ist:

```
pi@raspberrypi ~ $ sudo modeprobe ledtrig_heartbeat
```

Dann wird *heartbeat* in die Trigger-Datei eingetragen und aktiviert:

```
pi@raspberrypi ~ $ echo heartbeat | sudo tee
/sys/class/leds/led0/trigger
```

Daraufhin blinkt die ACT-LED kontinuierlich in einem bestimmten Rhythmus, solange das Betriebssystem arbeitet.

```
pi@raspberrypi ~ $ echo none | sudo tee /sys/class/leds/led0/trigger
none
pi@raspberrypi ~ $ sudo modprobe ledtrig_heartbeat
pi@raspberrypi ~ $ echo heartbeat | sudo tee /sys/class/leds/led0/trigger
heartbeat
pi@raspberrypi ~ $ echo none | sudo tee /sys/class/leds/led0/trigger
none
pi@raspberrypi ~ $ echo 255 | sudo tee /sys/class/leds/led0/brightness > /dev/null
pi@raspberrypi ~ $ echo 0 | sudo tee /sys/class/leds/led0/brightness > /dev/null
pi@raspberrypi ~ $
```

Abbildung 6.3: Trigger-Bedingung auf none setzen, Kernel-Modul ledtrig_heartbeat installieren und die Trigger-Bedingung auf heartbeat setzen, was die LED zum selbsttätigen, kontinuierlichen Blinken veranlasst. Danach wieder die Trigger-Bedingung wieder auf none setzen und die LED mithilfe der Brightness-Datei manuell an- und ausschalten.

6.1.3 Mit Brightness schalten

Im Verzeichnis `/sys/class/leds/led0` befindet sich neben *trigger* die Datei *brightness*, die entweder den Wert 0 (LED aus) oder den Wert 255 (LED an) beinhaltet. Durch das Schreiben dieser Werte kann somit auf einfache Art und Weise die LED ein- oder ausgeschaltet werden.

LED an:

```
pi@raspberrypi ~ $ echo 255 | sudo tee
/sys/class/leds/led0/brightness > /dev/null
```

LED aus:

```
pi@raspberrypi ~ $ echo 0 | sudo tee /sys/class/leds/led0/brightness
> /dev/null
```

Die beiden obigen Zeilen sind einzeilig einzugeben. Unter Nutzung der Kommandozeilen-Historie (Pfeil-Auf-Taste » ↑«) lässt sich die LED dann leicht manuell schalten.

6.1.4 Python-Programm

Das zuvor gezeigte Prinzip kann natürlich auch in eigenen Programmen (Python, C) eingesetzt werden, die den gewünschten Wert in die brightness-Datei schreiben. Ausgangspunkt ist dabei stets die Einstellung *[none]* in der Datei *Trigger*. Wie dies in Python aussehen kann, zeigt das folgende Listing.

```
# Funktionen in Python

import re
dir_led = "/sys/class/leds/led0"
original_use = none

def led_claim():
      "LED-Kontrolle uebernehmen"
      global original_use
      trigger_file = dir_led+"/trigger"
      with open trigger_file, ' r') as f:
            m = re.match('\[(.*)\]', f.read())

      if m:
            original_use = m.group(1)

      else:

            original_use = none

            with open (trigger_file, 'w') as f:
            f.write('none')


def led_release():

      "LED-Kontrolle abgeben"
      global original_use
      if original_use:
            with open(dir_led+"/trigger", 'w') as f:
            f.write(original_use)


def led_on():

      "LED an"
      with open(dir_led+"/brightness", 'w') as f:   f.write('255')


def led_off():

      "LED aus"
      with open(dir_led+"/brightness", 'w') as f:   f.write('0')
```

Ein einfaches Programm, welches die oben definierten Funktionen anwendet, ist im Folgenden gezeigt:

```
# LED-Brightness in Python

import time

def main():

      led_claim()
```

```
for i in range(100):
        print("an")
        led_on()
        time.sleep(1)

        print("aus")
        led_off()
        time.sleep(1)

led_release()
return 0
```

6.2 Einsatz des GPIO-Ports

Um eigene Peripherie mit den GPIO-Anschlüssen verbinden zu können, wird eine geeignete Steckverbindung benötigt. Kabel mit angeschlossenen Steckern, wie sie in PCs für USB-Slots oder auch für Audioverbindungen zum Einsatz kommen, können hierfür durchaus verwendet werden, falls einige wenige Pins von der Pfostenleiste abgegriffen werden sollen. Für die +Modelle eignet sich als GPIO-Verbindung ein IDE-Kabel, welches über 40 Kontakte und einen Pfostenstecker verfügt. Verschiedene Hersteller bieten spezielle Adapter für den Raspberry Pi an, die die GPIO-Signale über ein Fachbankkabel auf eine Platine führen, wie beispielsweise der *Pi Cobbler* der Firma Adafruit. Die 26-polige-Version passt jedoch mechanisch nicht auf die neueren +Versionen.

Abbildung 6.4: Der Pi Cobbler wird als Bausatz geliefert.

Sowohl beim Zusammenbau als auch beim späteren Anschluss des Pi Cobbler oder einer ähnlichen Adapterlösung ist auf die Orientierung der Signale zu achten, damit nicht unabsichtlich eine Signalverdrehung hergestellt wird, was durchaus zur elektrischen Zerstörung des Raspberry Pi führen kann. Wie es im Kapitel 3.12 erwähnt ist, sind die GPIO-Pins direkt mit dem BCM2835 verbunden. Es gibt hier keine Schutzschaltung. Grundsätzlich sollte man sich stets am Pin 1 der Verbindung orientieren, der beim Verbindungskabel rot markiert ist und auf der Cobbler-

sowie auf der Raspberry Pi-Platine ebenfalls kenntlich gemacht ist. Zudem wird der Pin 1 auf den Platinen stets mit einem eckigen Lötpad, statt mit runden wie bei den anderen Pins, ausgeführt. Die auf den Cobbler einzusetzende Pfostenleiste verfügt über eine Einsparung, so dass das Kabel nur in einer Richtung eingesetzt werden kann. Die Leiste kann allerdings auch falsch herum eingelötet werden.

Für den Bastelaufbau von Schaltungen eignen sich die bekannten Steckplatinen (Breadboards), auf die die Bauteile einfach gesteckt werden können. Die elektrischen Verbindungen werden durch die Kontakte des Boards selbst oder mit kleinen Steckbrücken hergestellt.

Abbildung 6.5: Der Pi Cobbler führt alle Signale des Raspberry Pi GPIO-Ports auf ein Breadboard, womit sich schnell eine Schaltung zusammenstecken lässt.

Breadboards eignen sich nicht für finale Schaltungen, sondern lediglich für Testaufbauten. Unbestückte Lochrasterplatinen (Prototyping Boards) mit denen sich für stabilere gelötete Schaltungen realisieren und sich direkt auf den Raspberry Pi aufstecken lassen, sind ebenfalls in verschiedenen Versionen verfügbar. Außerdem sind spezielle Erweiterungsplatinen für Aufgaben in der MSR-Technik mit A/D- und D/A-Wandlern, Sensoren und Relais sowie Motorsteuerungen erhältlich. Im Gegensatz zu den Shields, wie die Erweiterungsplatinen für die Arduino-Plattform bezeichnet werden, ist das Angebot an aufsteckbaren Raspberry Pi-Boards bisher eher gering, die zudem oftmals teurer sind als ein Raspberry Pi-Board selbst.

Eine Ausnahme bildet das preisgünstige XTRINSIC-Sense Board der Firma Freescale auf dem sich drei Sensoren befinden: der MPL3115 für die Messung von Temperatur und Luftdruck zuständig, der MAG3110 als Magnetfeldsensor und der MMA8491 als Beschleunigungssensor, so dass sich hiermit recht einfach Messapplikationen wie beispielsweise Sensormessknoten oder auch Positions- und Bewegungsdetektoren realisieren lassen.

Abbildung 6.6: Das XTRINSIC-Board verfügt über drei Sensoren, die sich recht einfach für verschiedenste Messapplikationen einsetzen lassen.

Das XTRINSIC-Board ist ursprünglich die KL2-Freescale-Plattform entwickelt worden und arbeitet auch mit dem Raspberry Pi. Freescale liefert hierzu ein spezielles Raspbian-Image, welches die Sensorunterstützung anhand von drei Python-Scripts demonstriert sowie drei Web-Applikationen, die auf dem Raspberry Pi in der Konsole gestartet und auf einem üblichen Web-Browser betrachtet werden können. Der Client mit dem Web-Browser sollte sich hierfür im gleichen LAN wie der Raspberry Pi befinden.

6.2.1 Software

Für die Steuerung der GPIO-Signale sind mehrere Bibliotheken und Tools entwickelt worden, wie *RPI.GPIO* oder *wiringPi*, die dem Entwickler und dem Programmierer einen möglichst einfachen Hardware-Zugriff ermöglichen. Gleichwohl bietet der Linux Kernel selbst – auf den diese Bibliotheken aufsetzen – GPIO-Treiber in Form von C-Funktionen, die für eigene Treiber eingesetzt werden können. Damit ist eine sehr direkte Kommunikation mit der Hardware möglich, was mit den geringsten Latenzen und Verzögerungen einhergeht und insbesondere in zeitkritischen Applikationen (Timer) eine wichtige Rolle spielt.

bcm2835: input.c - Windows Internet Explorer

http://www.airspayce.com/mikem/bcm2835/input_8c-example.html

Datei Bearbeiten Ansicht Favoriten Extras ?

bcm2835 1.25

Main Page | Modules | Files | Examples

input.c

Reads the state of an RPi input pin

```
// input.c
//
// Example program for bcm2835 library
// Reads and prints the state of an input pin
//
// After installing bcm2835, you can build this
// with something like:
// gcc -o input input.c -l bcm2835
// sudo ./input
//
// Or you can test it before installing with:
// gcc -o input -I ../../src ../../src/bcm2835.c input.c
// sudo ./input
//
// Author: Mike McCauley
// Copyright (C) 2011 Mike McCauley
// $Id: RF22.h,v 1.21 2012/05/30 01:51:25 mikem Exp $

#include <bcm2835.h>
#include <stdio.h>

// Input on RPi pin GPIO 15
#define PIN RPI_GPIO_P1_15

int main(int argc, char **argv)
{
    // If you call this, it will not actually access the GPIO
    // Use for testing
//    bcm2835_set_debug(1);

    if (!bcm2835_init())
        return 1;

    // Set RPI pin P1-15 to be an input
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_INPT);
    //  with a pullup
    bcm2835_gpio_set_pud(PIN, BCM2835_GPIO_PUD_UP);

    // Blink
    while (1)
    {
        // Read some data
        uint8_t value = bcm2835_gpio_lev(PIN);
        printf("read from pin 15: %d\n", value);

        // wait a bit
        delay(500);
    }

    bcm2835_close();
    return 0;
}
```

Abbildung 6.7: Beispiel für die direkte Registerprogrammierung des BCM2835

Selbst die Funktionen des Kernels nutzen nicht alle Möglichkeiten aus, die der BCM2835-Prozessor prinzipiell für die unterschiedlichen Leitungen des GPIO-Ports vorsieht. Standardmäßig werden die GPIO-Funktionen und die Schnittstellen (SPI, I²C, Serial) unterstützt. Um auch die »alternativen« Funktionen (z.B. PCM, Clock) nutzen zu können, ist demnach eine direkte Programmierung der betreffenden fünf Register des BCM2835 notwendig, die sich über physikalische Adressen im ARM-Speicherbereich ansprechen lassen. Dabei wird der eingebaute GPIO-Treiber des Kernels für die (direkte) Programmierung ignoriert. Beispiele für diese Programmierung mithilfe einer speziellen C-Bibliothek (bcm2835.h) für den BCM2835 sind unter `http://www.airspayce.com/mikem/bcm2835/` zu finden (siehe Abbildung 6.7).

6.2.2 Kernel GPIO-Unterstützung

Ohne weitere Software installieren zu müssen, ist eine GPIO-Unterstützung bereits automatisch vorhanden, die unter `/sys/class/gpio` in Form zweier Dateien namens *export* und *unexport* zu finden ist. Wie bei der Veränderung der Triggerdatei im Kapitel 6.1.1 sind auch hier für das Schreiben Root-Rechte notwendig, so dass es am einfachsten ist, wenn man sich mit dem User *root* anmeldet (oder sudo -s ausführt).

Weil dem üblichen User *pi* bei diesen hardware-nahen Operationen die notwendigen Rechte fehlen, kommt es immer wieder vor, dass bestimmte Zugriffe auf Dateien und/oder Verzeichnisse nicht funktionieren wollen, zumal wenn sie (wie hier) gar nicht physikalisch vorhanden sind, sondern nur im Speicher existieren. Der Gefahr, dass der User root unabsichtlich Systemdateien löschen oder überschreiben kann, muss man sich dabei stets bewusst sein.

Der Inhalt des oben genannten Verzeichnisses lässt sich anzeigen mit:

```
root@raspberrypi:/home/pi# ls /sys/class/gpio
```

woraufhin die beiden oben erwähnten Dateien (export, unexport) zu erkennen sind. Um beispielsweise den Port GPIO 25 (an Pin 22) zu verwenden, wird die folgende Zeile angegeben:

```
root@raspberrypi:/home/pi# echo 25 > /sys/class/gpio/export
```

Grundsätzlich – und sicherheitshalber – sollte zumindest für die ersten Tests der GPIO-Funktion ein GPIO-Port gewählt werden, der definitiv keine alternative Funktion (etwa für SPI oder I²C) kennt. Deshalb empfehlen sich hier die Ports 22 bis 25 (vgl. Tabelle 3.4).

Durch die obige Zeile wird eine neue Datei mit der Bezeichnung gpio25 angelegt, was wieder mit dem ls-Kommando überprüft werden kann. Nach der Angabe von *unexport* auf die Nr. 25 wird diese gpio25-Datei wieder automatisch gelöscht.

```
root@raspberrypi:/home/pi# ls /sys/class/gpio/gpio25
```

Wenn das ls-Kommando explizit auf das neue Verzeichnis (gpio25) abgesetzt wird, erscheinen sieben weitere Dateien: active_low, direction, edge, power, subsystem, uevent und value, mit denen die Signaleigenschaften bestimmt werden.

Für die Signalausgabe über gpio25 ist zunächst die Pegeldefinition (active low oder active high) zu bestimmen. Mit der folgenden Zeile wird dies per active_low-File definiert. Bei Angabe einer 1 ist active_low definiert, bei einer 0 active_high, d.h., bei der späteren Angabe einer Eins für den Ausgabewert (value) hat dies ein Signal von 3,3 V zur Folge, bei einer Null dementsprechend 0 V.

```
root@raspberrypi:/home/pi# echo 0 >
/sys/class/gpio/gpio25/active_low
```

Danach ist die Signalrichtung per *direction* zu bestimmen. Mit der folgenden Zeile und der Angabe *out* wird gpio25 zum Ausgang:

```
root@raspberrypi:/home/pi# echo out >
/sys/class/gpio/gpio25/direction
```

Damit das gpio25-Signal von standardmäßig 0 V auf 3,3 V (high) schaltet, wird nun angegeben:

```
root@raspberrypi:/home/pi# echo 1 > /sys/class/gpio/gpio25/value
```

Am Pin 22 kann der Pegelwechsel mit einem Multimeter leicht nachgemessen werden, wenn die obige Zeile abwechselnd mit `echo 1 >` und `echo 0 >` ausgegeben wird.

Für das Einlesen eines Signals ist dementsprechend die Richtung (direction) zu ändern; eine Lesefunktion mit cat auf value auszuführen.

```
root@raspberrypi:/home/pi# echo out >
/sys/class/gpio/gpio25/direction
```

```
root@raspberrypi:/home/pi# cat /sys/class/gpio/gpio25/value
```

Je nach angelegtem Pegel am Pin 22 liefert *cat* als Anzeige den Wert 1 oder 0. Vom GPIO-Prinzip her, d.h. ohne separate Signalpufferung für *in* und *out*, ist es natürlich nicht möglich, dass der zuvor ausgegebene Pegel am gleichen Pin wieder eingelesen werden kann. In dem Moment, in dem die Umschaltung zwischen *out* und *in* stattfindet, ist der Out-Wert verschwunden. Der Eingang liefert so lange eine Null, bis am Pin ein High-Signal von maximal 3,3 V angelegt wird.

```
pi@raspberrypi ~ $ sudo -s
root@raspberrypi:/home/pi# ls /sys/class/gpio
export  gpiochip0  unexport
root@raspberrypi:/home/pi# echo 25 > /sys/class/gpio/export
root@raspberrypi:/home/pi# ls /sys/class/gpio
export  gpio25  gpiochip0  unexport
root@raspberrypi:
root@raspberrypi:/home/pi# ls /sys/class/gpio/gpio25
active_low  direction  edge  power  subsystem  uevent  value
root@raspberrypi:
root@raspberrypi:/home/pi# echo 0 > /sys/class/gpio/gpio25/active_low
root@raspberrypi:/home/pi# echo out > /sys/class/gpio/gpio25/direction
root@raspberrypi:/home/pi# echo 1 > /sys/class/gpio/gpio25/value
root@raspberrypi:/home/pi#
root@raspberrypi:/home/pi# echo in > /sys/class/gpio/gpio25/direction
root@raspberrypi:/home/pi# cat /sys/class/gpio/gpio25/value
0
root@raspberrypi:/home/pi#
```

Abbildung 6.8: Den GPIO-Port 25 als Ausgang und als Eingang testen

6.2.3 GPIO mit Python

Die Python-Bibliothek RPi.GPIO gibt es für Python 2 und für Python 3. Nach wie vor ist Python 2 die gebräuchlichere Version und die hierfür passende Bibliothek wird wie folgt installiert:

```
pi@raspberrypi ~ $ sudo apt-get install python-rpi.gpio
```

```
  GNU nano 2.2.6                                          Datei: porttest.py

#! /usr/bin/env python
import RPi.GPIO as GPIO # GPIO-Bibliothek
import time             # fuer time.sleep

print "Einfacher Port-Test"

# Festlegen des Bezeichnugsschemas
GPIO.setmode(GPIO.BCM)

# GPIO 25 (Pin 22) als Ausgang definieren
GPIO.setup(25, GPIO.OUT)

# GPIO 24 (Pin 18) als Eingang definieren
GPIO.setup(24, GPIO.IN)

# Endlose Programmschleife

while True:
        GPIO.output(25, True)    # Signal auf High
        time.sleep (2)           # 2 Sekunden warten
        GPIO.output(25, False)   # Signal auf Low
        time.sleep (2)           # 2 Sekunden warten
        inputvalue = GPIO.input(24)      # Portpegel lesen
        print "GPIO 24-Pegel: ", inputvalue      # und anzeigen
```

Abbildung 6.9: Einfaches Testprogramm für die GPIO-Ausgabe und -Eingabe

Im Python-Programm ist die Bibliothek zu importieren, woraufhin per *GPIO.setmode* festzulegen ist wie die folgenden Nummern für die Ports zu interpretieren sind: Entweder als BOARD, womit die Nummerierung der Pins auf der Platine gemeint ist, oder als BCM, womit die GPIO-Nummern gemeint sind, wie es auch im folgenden Programm (porttest.py) angewendet wird.

```
# GPIO mit Python
#! /usr/bin/env python     #shebang line
import RPi.GPIO as GPIO    # GPIO-Bibliothek
import time          # fuer time.sleep
print "Einfacher Port-Test"
# Festlegen des Bezeichnungsschemas
GPIO.setmode (GPIO.BCM)
# GPIO 25 (Pin 22) als Ausgang definieren
GPIO.setup(25, GPIO.OUT)
# GPIO 24 (Pin 18) als Eingang definieren
GPIO.setup(24, GPIO.IN)
# Endlose Programmschleife
while True:
      GPIO.output(25, True)      # Signal auf High
      time.sleep (2)             # 2 Sekunden warten
      GPIO.output(25, False)     # Signal auf Low
      time.sleep (2)             # 2 Sekunden warten
      inputvalue = GPIO.input(24)             # Portpegel lesen
      print "GPIO 24-Pegel: ", inputvalue     # und anzeigen
```

Mit dem GPIO.setup-Kommando werden die betreffenden Anschlüsse entweder als Ausgang oder als Eingang definiert. Das Programm gibt in einer Endlosschleife, die mit Strg+C abgebrochen werden kann, auf GPIO 25 ein High (3,3 V), gefolgt von einem Low (0 V), aus, wobei dazwischen eine Wartezeit von 2 s verstreicht. Im letzten Schritt erfolgt das Einlesen des Pegels an GPIO 24 (inputvalue) mit der Anzeige des ermittelten Pegels.

Weil der Zugriff auf die Bibliothek root-Rechte erfordert, ist die Ausführung des Programms zu initiieren mit:

```
pi@raspberrypi ~ $ sudo python porttest.py
```

```
pi@raspberrypi ~ $ sudo python porttest.py
Einfacher Port-Test
GPIO 24-Pegel:  0
GPIO 24-Pegel:  0
GPIO 24-Pegel:  1
GPIO 24-Pegel:  0
KeyboardInterrupt
pi@raspberrypi ~ $
```

Abbildung 6.10: Die Programmausführung

Optional könnte dem Programm noch die Zeile `GPIO.setwarnings(False)` hinzugefügt werden, damit beim mehrmaligen Programmaufruf oder beim Abbruch keine Warnmeldung angezeigt wird.

Ein Taster, zwischen dem 3,3 V-Anschluss (Pin 1) des Pfostensteckers und dem Anschluss GPIO 24 (Pin 18) geschaltet, kann für die einfache Funktionsüberprüfung des Eingangs eingesetzt werden. Eine gewöhnliche Leuchtdiode, deren Pluspol (Anode, der längere Anschluss) über einen Vorwiderstand von ca. 330 Ohm mit GPIO 25 (Pin 18) und deren Katode mit Masse (Pin 6) verbunden wird, dient der Funktionsüberprüfung des Ausgangs.

6.3 Serielle Schnittstelle

Für die serielle Schnittstelle lässt sich mit den aktuellen raspi-config-Versionen die Funktion für die Übertragung von Shell- und Kernel-Nachrichten aktivieren. Im Kapitel 1.3.17 ist bereits erwähnt worden, dass lediglich eine TXD- und eine RXD-Leitung auf TTL-Pegel unterstützt werden, so dass ohne zusätzliche Peripherie keine Gerätekopplung, sondern direkt lediglich eine Chip-zu-Chip-Kommunikation möglich ist.

Im PC-Bereich wird bereits seit geraumer Zeit standardmäßig keine serielle Schnittstelle mehr eingebaut, die dem Standard RS232 entspricht, was sich im Bedarfsfall jedoch über eine entsprechende Adapterkarte (PCIe oder PCI) einfach nachrüsten lässt. Diese serielle Schnittstelle ist für den asynchronen Betrieb (ohne Taktleitungen) ausgelegt und für den Anschluss wird ein 9-poliger DSUB-Verbindungsstecker eingesetzt.

Kürzel	Bezeichnung	9-polig
DCD	Data Carrier Detect	1
RXD	Receive Data	2
TXD	Transmit Data	3
DTR	Data Terminal Ready	4
GND	Ground, Masse	5
DSR	Data Set Ready	6
RTS	Request To Send	7
CTS	Clear To Send	8
RI	Ring Indicator	9

Abbildung 6.11: Signale und Anschlüsse der seriellen Schnittstelle, wie sie für PC üblich sind

Häufiger kommt statt einer Einsteckkarte ein *USB to Serial-Adapter* zum Einsatz, was den Vorteil hat, dass sich dieser auch mit einem Notebook verwenden lässt

und keine Einsteckkarte im PC eingebaut werden muss. Auf dem PC wird mit einem derartigen USB-Adapter ein virtueller COM-Port installiert, der prinzipiell wie eine physikalisch vorhandene serielle Schnittstelle zu verwenden ist, etwa mit einem Windows-Terminalprogramm. Diese Adapter enthalten einen Mikrocontroller und einen Pegelumsetzer, was typischerweise zu einer Realisierung mit einem Chip der Firma FTDI oder der Firma Prolific führt, der für die Protokoll- und Signalumsetzung von RS232 auf USB und umgekehrt zuständig ist. Diese einfachen Adapter erfüllen oftmals jedoch nicht die RS232-Spezifikation, weil die Pegel nicht eingehalten werden (statt +/-12 V nur +/- 5V oder noch weniger) oder auch weil Steuersignale (CTS, DSR) fehlen, so dass mit ihnen häufig Kommunikationsprobleme auftreten. Bei RS232-Einsteckkarten für den PC stehen in der Regel jedoch auch die Übergabeleitungen (Handshake) entsprechend der Abbildung 6.11 mit den standardisierten Pegeln zur Verfügung.

6.3.1 Verbindungen

Grundsätzlich ist bei der Verbindung von zwei Einheiten über die RS232-Schnittstelle zu beachten, von welchem Typ die Geräte sind, so dass von zwei gleichartigen Geräten (PCs) eine so genannte *Nullmodem-Schaltung* einzusetzen ist, bei der eine Überkreuzschaltung der TXD- und RXD-Leitung auszuführen ist, damit die Sendeleitung (TXD) des einen die Empfangsleitung (RXD des anderen Gerätes ist. Ein Steuergerät − wie ein PC − wird gemeinhin als *DatenEndEinrichtung* (DEE) und eine Peripherie − was klassischerweise ein Modem ist − als *DatenÜbertragungsEinrichtung* (DÜE) bezeichnet.

Die Übertragungssicherheit kann dadurch erhöht werden, dass zwischen den Geräten ein Hardware-Handshaking einführt wird. Dann werden nicht nur die Datenleitungen miteinander verbunden (überkreuz oder nicht, je nach Konfiguration), sondern auch die Leitungen RTS und CTS. Mit *Clear To Send* (CTS) meldet die Peripherie, dass sie zur Datenaufnahme bereit ist. Dies geschieht in der Regel als Reaktion auf den vorherigen Empfang des Signals *Request To Send* (RTS) von der DEE. Die Datenübertragung kann nur dann stattfinden, wenn RTS und CTS aktiv (Low) sind.

Eine weitere Verbesserung der Datenübertragung ergibt sich durch die Verwendung der Leitungen *Data Set Ready* (DSR) und *Data Terminal Ready* (DTR), die dann für den Aufbau der Verbindung verantwortlich sind. Ein Modem (DÜE) zeigt beispielsweise über DSR an, dass es seine internen Einstellungen (Initialisierung) vorgenommen hat und betriebsbereit ist. Der Computer (DEE) ist zur Datenverarbeitung bereit, wenn DTR aktiv (Low) ist. Solange dieses Signal nicht aktiv ist, findet keine Datenübertragung statt. Der Status der übrigen Leitungen spielt keine Rolle.

Falls die Software eines Computers die Unterstützung der Handshake- und der Betriebsbereitschaftsleitungen erfordert und die Peripherie diese nicht zur Verfügung stellen kann, werden oftmals lediglich die Datenleitungen (TXD, RXD) verwendet, während die Leitungen RTS-CTS und DTR-DSR miteinander verbunden werden, damit dem Computer das Vorhandensein der »fehlenden Signale« vorgetäuscht werden kann.

Über die Leitung *Data Carrier Detect* (DCD) teilt die Peripherie der DEE mit, dass sich der Signalpegel innerhalb des zulässigen Spannungsbereichs befindet und eine Übertragung daher erfolgen kann. In vielen Fällen wird diese Leitung nicht verwendet, sondern ausschließlich das Signal DSR zur Bereitschaftsanzeige. Eine Variante, wie sie beim Anschluss von Druckern und Plottern mit einem seriellen Interface gebräuchlich ist, wird so ausgeführt, dass die Handshake-Leitung CTS mit der Bereitschaftsleitung DSR des Computers und mit dem Signal DTR der Peripherie verbunden wird, so dass die Bereitschaft zur Datenaufnahme auf diese Art und Weise signalisiert werden kann.

Abbildung 6.12: Minimallösung einer seriellen Schnittstelle für den Raspberry Pi

Für den Raspberry Pi ist eine spezielle FTDI-Lösung in Form eines Adapters (TTL-232R, Abbildung 6.12) erhältlich, die mit drei Leitungen an den GPIO-Port (Pin 6: Masse, Pin 8: TXD, Pin 10 RXD) angeschlossen wird. Auf der anderen Seite steht ein USB-Anschluss zur Verfügung, der wie üblich als virtueller COM-Port behandelt wird. Der Pegel beträgt lediglich 3,3 V, so dass sich auch hiermit keine RS232-Schnittstelle realisieren lässt, sondern nur eine einfache serielle TTL-Verbindung, etwa für die serielle Übertragung von Messdaten vom Raspberry Pi zu einem Gerät mit USB-Anschluss für den ein passender Gerätetreiber von FDTI zur Verfü-

gung steht. Von diesen Adaptern sind auch andere Ausführungen mit 5 V-Pegel und Unterstützung von Übergabeleitungen erhältlich, was aber nichts am Prinzip ändert, zumal der Raspberry Pi eben nur eine Transmit- (TXD) und eine Receive- (RXD) Leitung standardmäßig für die serielle Schnittstelle unterstützt. Prinzipiell ist es natürlich möglich, dass einige der GPIO-Signale für die Steuerung der Datenübertragung zum Einsatz kommen, wenn die Peripherie dies erfordern sollte.

Bei den beiden Kommunikationspartnern müssen die gleichen Parameter für den Betrieb festgelegt werden. Die Daten sind beim Raspberry Pi standardmäßig wie folgt:

- Baudrate: 115200 Bits/s
- Datenbits: 8
- Parity: keine
- Stoppbit: 1
- Flusskontrolle: Keine

6.3.2 RS232-Treiber

Für den Aufbau einer standardisierten seriellen Schnittstelle gemäß RS232, die auch als V.24 bezeichnet wird, muss dem UART, der sich im BCM2835 befindet, ein Converter nachgeschaltet werden, weil die Mikrocontroller typischerweise mit TTL-Pegeln arbeiten, die dementsprechend auf RS232-Niveau umgesetzt werden müssen, wo ein Low einer Spannung im Bereich von 3 bis 15 V und ein High einer Spannung von -3 V bis -15 V entspricht.

Abbildung 6.13: Der MAX232 ist ein Standardbaustein für den Aufbau einer TTL-RS232-Konverterschaltung und benötigt als externe Beschaltung lediglich sechs Kondensatoren.

In vielen Applikationen steht jedoch keine Spannungsversorgung von ±12 V, sondern lediglich +5 V zur Verfügung, wie es in Mikroprozessor- oder auch allgemein bei TTL-Schaltungen der Fall ist. Es erscheint dann allein aus Kostengründen nicht sinnvoll, lediglich für die Transceiver der RS232-Schnittstelle ein eigenes Netzteil vorzusehen. Speziell aus diesem Grund hat die Firma Maxim bereits im Jahre 1985 den Baustein MAX232 entwickelt, der heute gemeinhin als Standard-Transceiver für Embedded-Applikationen gilt, die eine RS232-Schnittstelle benötigen. Der MAX232 enthält zwei Sender und zwei Empfänger sowie einen Spannungswandler nach dem Prinzip der Ladungspumpe. Extern müssen lediglich vier Kondensatoren hinzugeschaltet werden, damit aus der 5 V-Versorgungsspannung ±10 V erzeugt werden können, wodurch eine RS232-Ausgangsspannung von ca. ±9 V erreicht wird, was noch innerhalb der RS232-Spezifikation liegt.

6.3.3 Kommunikationsbeispiel

Als einfaches Beispiel für die Kommunikation mit der seriellen Schnittstelle (UART) des Raspberry Pi ist über die drei Leitungen (TXD, RXD, GND) ein Demo-Board für Energy Harvesting-Applikationen der Firma Cymbet (CBC-EVAL-09) angeschlossen. Dabei wird die Nullmodem-Schaltung eingesetzt, so dass die Sendeleitung des Raspberry Pi (TXD) die Empfangsleitung (RXD) des Boards ist und umgekehrt. Das Demo-Board kann mit einer Solarzelle, einem Thermogenerator oder auch mit einem Piezo-Element arbeiten, deren gelieferte Energie mithilfe verschiedener Schaltkreise umgesetzt und in einer Kondensatorbank gespeichert wird, so dass am Ausgang eine geregelte und stabile Spannung von 3,6 V für die Versorgung von Mikrocontrollerschaltungen und anderen Applikationen zur Verfügung steht.

Abbildung 6.14: Die Energy Harvesting-Schaltung verfügt über eine serielle Schnittstelle.

Die Basis der Schaltung bildet der so genannte *Energy Processor CBC915* von Cymbet. Dabei handelt es sich jedoch nicht um einen digitalen programmierbaren

Prozessor, sondern um einen speziellen Chip (ASIC), der über bestimmte, vom Hersteller festgelegte Funktionen verfügt. Mithilfe eines Tasters auf dem Board kann der aktuelle Status der Energie-Umsetzung bzw. der Aufladung anhand dreier LEDs visualisiert werden, was sich in ausführlicher Form auch über die serielle TTL-UART-Schnittstelle des Energy Processors abfragen lässt. Hierfür können verschiedene einfache Kommandos an den CBC915-Processor abgesetzt werden. Es gibt neun Kommandos bzw. Kommandogruppen wie *Power Available* (Command 1), *Status* (Command 2) oder auch *Find Maximum Peak Power Point* (Command 4) die entsprechende Antworten liefern und anhand des CBC915-Datanblatts zu interpretieren sind. Die Abbildung 6.15 zeigt, wie die Kommunikation mit dem erstellen Programm aussieht.

Abbildung 6.15: Kommunikation mit dem Energy Processor

In der Schaltung des Boards befindet sich für die Anpassung an die UART-Schnittstelle eines MSP430-Mikrocontrollers, der hier üblicherweise angeschlossen wird, in den beiden Leitungen (RXD, TXD) eine Widerstands/Kondensator-Kombination (10 MΩ, 1 nF) die zu überbrücken ist, damit die Kommunikation mit dem Raspberry Pi oder auch mit einem PC korrekt funktioniert.

Die Funktion für die Übertragung von Shell- und Kernel-Nachrichten über die UART-Schnittstelle, die sich beispielsweise mit raspi-config einstellen lässt, darf für eine übliche serielle Kommunikation, wie sie hier gezeigt wird, nicht aktiviert sein. Das folgende Programm kommuniziert mit dem Energy-Harvesting-Board, wobei die Grundlage hierfür von IBEX mit dem Titel *Using the UART* stammt. Es

bietet alle wesentlichen Funktionen für das Senden und Empfangen von Bytes mit einigen kommentierten Anpassungen für den verwendeten Energy Processor.

```
# UART Test mit CBC915

#include <stdio.h>                // Standard I/O
#include <unistd.h>               // UART
#include <fcntl.h>                // UART
#include <termios.h>              // UART-Konfiguration

void delay(int milliseconds);

void getCmd();

int main(void) {

      int uart0_filestream = -1;
      int rx_length = -1;
      int lword = 0;

      // UART0 öffnen

      uart0_filestream = open("/dev/ttyAMA0", O_RDWR | O_NOCTTY |
                         O_NDELAY);

      if (uart0_filestream == -1)

      {

      printf("Fehler - UART konnte nicht geöffnet werden\n");

      }

      // Konfiguration des UART für CBC915-AC

      // 9600, 8 Bit, kein Parity Bit, 1 Stop Bit (9600 8 N 1)

      // The flags defined in /usr/include/termios.h

      // Baud rate: B9600 (9600)
      // SIZE: CS8 (8 Bit)
      // CLOCAL: Ignore modem status lines
      // CREAD: Enable receiver

      struct termios options;
      tcgetattr(uart0_filestream, &options);
      options.c_cflag = B9600 | CS8; // | CLOCAL | CREAD;
      tcsetattr(uart0_filestream, TCSANOW, &options);

      while(uart0_filestream != -1) {

         // String für den Befehl: <#><Leerzeichen><Befehl><cr>

            // Beispiel: # 2<cr>
            // Mit <cr> als Carriage Return

            unsigned char tx_buffer[4];

         // Buffer der UART-Schnittstelle leeren

            tcflush(uart0_filestream, TCIOFLUSH);
            lword = 0;
            tx_buffer[0] = '#';
            tx_buffer[1] = ' ';

        // Befehl einlesen
```

```
        printf("Bitte CMD eingeben: ");
        do{ scanf("%c", &tx_buffer[2]); }while(getchar() !=
                  '\n');
    // Letztes Sende-Byte als Carriage Return setzen
        tx_buffer[3] = 13; //<CR>
    // Bytes senden
        if (uart0_filestream != -1)
        {
        int count = write(uart0_filestream, tx_buffer, 4);
        if (count < 0)
              {
                     printf("UART Sendefehler\n");
              }
        }
    // Bytes lesen
        if (uart0_filestream != -1)
        {
        // 8-Byte-weise lesen
        do {
           unsigned char rx_buffer[50];
           rx_length = read(uart0_filestream,
                  (void*)&rx_buffer[lword*8], 50 - lword*8);
        // Falls genau 8 Bytes gelesen wurden, Schleife
        // weiter durchlaufen bis die nächsten Bytes gelesen
        // wurden.
        // Ausnahme Befehl 4 und 9: Antwort ist genau 8 Byte
        if (rx_length == 8 && tx_buffer[2] != '4' &&
                  tx_buffer[2] != '9') {
           lword++;
                     }
        // Falls ungleich 8 Bytes vom Bus gelesen wurden
        // Ausgabe der Empfangenbytes und Abbruch
        // Ausnahme Befehl 4 und 9 (s.o.)
    else if (rx_length >= 1) {
    rx_length += lword * 8;
    rx_buffer[rx_length] = '\0';
    printf("%i bytes read : %s\n", rx_length, rx_buffer);
        break;
                           }
              } while (1);
        }
    }
```

```
        // UART schließen
        close(uart0_filestream);
        return 0;
}
```

6.4 Serial Peripheral Interface – SPI

Das *Serial Peripheral Interface* (SPI) ist ursprünglich eine von der Firma Motorola entwickelte synchrone Schnittstelle, die in erster Linie für die Kommunikation zwischen Mikrocontrollern und Peripheriebausteinen genutzt wird; im Prinzip für die gleichen Einsatzzwecke wie der I²C-Bus (siehe Kapitel 6.5). Dabei ist SPI jedoch nur für relativ kurze Verbindungen auf einer Leiterplatte gedacht, nicht für externe Verbindungen.

SPI ist für möglichst hohe Datenraten konzipiert, wobei durchaus 20 MByte/s möglich sind, etwa bei der Anbindung von Speicherkarten (SD). Die beteiligten Komponenten (Controller + Peripheriechip) bestimmen die jeweils maximal mögliche Datenrate, die vom Controller flexibel gehandhabt werden kann, weshalb bei den integrierten SPI-Interfaces in den Mikrocontrollern verhältnismäßig viele Einstellungsoptionen existieren.

SPI entspricht – im Gegensatz zum I²C-Bus – keinem »richtigen Bussystem«, weil jede SPI-Einheit eine eigene Leitung (/SS) benötigt und keinerlei Definitionen in Hinblick auf maximale Ausdehnung, Datenrate und damit Pull-Up oder Terminierung existieren. SPI stellt somit einen sehr »lockeren Standard« dar. Es ist eine genaue Verifizierung der jeweils zu verwendenden SPI-Einheiten notwendig, was in der Praxis immer wieder zu Kompatibilitätsproblemen führt.

Abbildung 6.16: Aufbau eines SPI-Systems mit drei unterschiedlichen Slaves

In der Standardkonfiguration zwischen einem Master und einem Slave sind zwei Steuer- und zwei Datenleitungen vorgesehen, was eine Full-Duplex-Übertragung gestattet. In der folgenden Aufzählung sind die SPI-Signale mit der entsprechenden Implementierung beim Raspberry Pi (in Klammern) angegeben.

- **/SS:** Slave Select (SPI Chip Select 0, Pin 24 und SPI Chip Select 1, Pin 26)
- **SCK:** Serial Clock (GPIO11, Pin 23)
- **MOSI:** Master Out Slave In (GPIO10, Pin 19)
- **MISO:** Master In Slave Out (GPIO21, Pin 21)

Theoretisch können beliebig viele Teilnehmer an den Bus angeschlossen werden, wobei es immer exakt nur einen einzigen Master geben darf. Für jeden Teilnehmer existiert jeweils eine eigene SS-Leitung (aktiv low) zwischen Master und Slave, mit deren Hilfe der jeweilige Slave selektiert wird. Der Raspberry Pi, d.h. der im BCM2835 integrierte SPI-Controller, bildet dabei den Master, mit dem zwei SPI Devices (Chip Select 0 und 1) unterstützt werden. Mit den +Modellen (siehe Kapitel 3.12.2) ist ein weiterer SPI-Port eingeführt worden.

Abbildung 6.17: Der Luftdrucksensor der Firma Freescale arbeitet mit SPI.

Das Spektrum von Komponenten mit SPInterface reicht vom einfachen Schieberegister über Sensoren und Wandler (A/D, D/A), über Displays bis hin zu Mikrocontrollern. Das Grundprinzip des Schieberegisters ist bei allen SPI-Komponenten anzutreffen, obgleich einige nicht alle vier Signalleitungen führen. Die Befehlscodes und Datenwerte werden seriell über die Leitungen gesendet, in ein Schieberegister eingelesen und stehen dann im Baustein zur Weiterverarbeitung zur Verfügung. Die Länge der Schieberegister ist nicht fest definiert, sondern kann von Baustein zu Baustein unterschiedlich sein.

6.4.1 Chip-Kommunikation

Der Master generiert das Taktsignal (SCK, typischerweise ab 100 kHz) und legt über die jeweilige SS-Leitung (auch als /CS=Chip Select bezeichnet) fest, mit welchem Slave die Kommunikation stattfinden soll, was mit einem Low-Pegel bestimmt wird. Im unselektierten Zustand sind die Signalleitungen im hochohmigen Zustand, wodurch der Baustein vom Bus abgekoppelt ist. Anschließend wird

1 Byte vom Master zum Slave gesendet und – je nach Bausteintyp – möglicherweise ein weiteres Byte vom Slave zum Master (MISO).

Abbildung 6.18: Die Datenübertragung zwischen dem Master und einem Slave erfolgt Full-Duplex über integrierte Schieberegister.

Einige Register benötigen eine ansteigende Flanke zum Schieben, andere eine abfallende. Um das Datensignal dem Taktsignal richtig zuordnen zu können, ist der jeweilige Datenübertragungsmodus mit den beiden Parametern Clock Polarity (CPOL) und Clock Phase (CPHA) einstellbar, was üblicherweise im Mikrocontroller durch entsprechende Register erfolgt. Insgesamt existieren vier verschiedene SPI-Betriebsarten, die in der Tabelle 6.1 angegeben sind.

Ist CPOL gleich 0, ist die Taktleitung (SCK) in Ruhe auf Low-Potenzial, also High-aktiv, bei CPOL gleich 1 ist die Taktleitung in Ruhe auf High, also Low-aktiv. Der Parameter CPHA gibt an, auf welcher Taktflanke die Daten übernommen werden sollen. Bei CPHA gleich Null werden sie bei der ersten ansteigenden Flanke übernommen, bei CPHA gleich Eins findet die Datenübernahme bei der ersten beliebigen Flanke statt. In der Abbildung 6.19 ist ein Beispiel zur Steuerung der SPI-Datenübertragung angegeben. Bei jeder Taktperiode wird ein Bit übertragen. Dementsprechend endet der Datentransfer üblicherweise nach acht Taktperioden, wobei grundsätzlich auch mehrere Bytes nacheinander übertragen werden können. Die Datenübertragung endet, sobald das /SS-Signal wieder auf High-Potential geführt wird.

Tabelle 6.1: SPI-Betriebsarten

SPI-Modus	Clock Polarity (CPOL)	Clock Phase (CPHA)
0	0	0
1	0	1
2	1	0
3	1	1

Abbildung 6.19: Datenübertragung im SPI-Modus Null. Die Daten werden auf der ersten ansteigenden Taktflanke übernommen.

6.4.2 Linux-Treiber und Anwendung

Für SPI ist ein Linux-Treiber verfügbar, der über ein eigenes Modul (spi-bcm2708) verfügt, welches im Bedarfsfall erst freizugeben ist, weil die für SPI vorgesehen Pins (siehe obige Aufzählung) standardmäßig als GPIO-Ports fungieren.

6.4.2.1 Blacklist

Das Sperren bzw. die Freigabe von Treibern erfolgt mithilfe einer *Blacklist*, damit der SPI-Treiber daraufhin aktiviert und auch automatisch geladen werden kann. Diese Blacklist wird mit dem Editor *nano* wie folgt aufgerufen:

```
pi@raspberrypi ~ $ sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

In der Liste befinden sich mindestens zwei Einträge: *blacklist spi-bcm2708* für SPI und *blacklist i2c-bcm2708* für I²C. Somit sind die passenden Treiber bereits auf dem System installiert, werden jedoch durch diese Einträge gesperrt, so dass diese gelöscht oder auch als Kommentar (mit #) gekennzeichnet werden, um diese Einstellung später leicht wieder rückgängig machen zu können. Nach einem Reboot sollte daraufhin spi_bcm2708 angezeigt werden. Wenn die Entsperrung bei dieser Gelegenheit auch gleich für I²C durchgeführt wird, dementsprechend auch das Modul i2c_bcm2708. Um festzustellen, welche Module beim Start automatisch geladen werden, wird der Befehl *lsmod* (Abbildung 6.20) angewendet. Im Kapitel 1.3.17 ist erläutert, dass das Entsperren der SPI- und der I²C-Schnittstelle auch direkt bei den neueren raspi-config-Versionen möglich ist.

```
pi@raspberrypi ~ $ lsmod
Module                  Size  Used by
snd_bcm2835            15846  0
snd_pcm                77560  1 snd_bcm2835
snd_page_alloc          5145  1 snd_pcm
snd_seq                53329  0
snd_seq_device          6438  1 snd_seq
snd_timer              19998  2 snd_pcm,snd_seq
snd                    58447  5 snd_bcm2835,snd_timer,snd_pcm,snd_seq,snd_seq_de
spidev                  5224  0
evdev                   9426  3
spi_bcm2708             4816  0
i2c_bcm2708             3759  0
pi@raspberrypi ~ $
```

Abbildung 6.20: Die Module für SPI und I²C sind geladen.

Damit sollten ein neues Verzeichnis sowie zwei neue Einheiten (spi0.0 und spi0.1) im Dateisystem auftauchen, die wie folgt sichtbar gemacht werden können:

```
pi@raspberrypi ~ $ ls /sys/bus/spi/devices
```

Diese beiden Einheiten, die als Dateien anzusehen sind, repräsentieren die zwei SPI-Schnittstellen und können im Prinzip so angesprochen werden, wie es für die GPIO-Ports im Kapitel 6.2.2 (Kernel GPIO-Unterstützung) erläutert ist. Die Einheiten müssen außerdem im Verzeichnis /dev vorhanden sein, damit sie als Gerät (device) behandelt werden können, was entsprechend zu überprüfen ist, wie es in der Abbildung 6.21 gezeigt ist.

```
pi@raspberrypi / $ ls /sys/bus/spi/devices/
spi0.0  spi0.1
pi@raspberrypi / $ ls /dev/spi*
/dev/spidev0.0  /dev/spidev0.1
pi@raspberrypi / $
```

Abbildung 6.21: Die beiden SPI-Schnittstellen im Dateisystem

Die Kommunikation mit SPI-Einheiten, wie etwa einem Sensor (Abbildung 6.17), unterliegt generell einigen Unwägbarkeiten, weil es sich dabei um einen »lockeren Standard« handelt und die Hersteller unterschiedliche Implementierungen sowie Bezeichnungen für SPI, wie etwa Microwire, verwenden. Die Kennzeichnung der Anschlüsse, die Betriebsart (Tabelle 6.1), die zulässige Taktfrequenz und die Anzahl der zu sendenden und zu empfangenden Bits können sich demnach bei SPI-Chips unterscheiden. Die Übertragung findet dabei stets in beiden Richtungen (Abbildung 6.19) statt, so dass – je nach Chiptyp – bestimmte Bits keine Bedeutung haben, weil jeweils die gleiche Anzahl von Bits für das Senden und für das Empfangen verwendet wird.

6.4.2.2 Systemzugriff mit Input/Output Controls

Der Zugriff auf die SPI-Schnittstellen kann über *Input/Output Controls* (IOCTL), eine spezielle Art von Systemaufruf, erfolgen, was sich leicht in C-Programmen

verwenden lässt. Im Folgenden ist hierfür Beispielcode gezeigt, der auf spidev0.0 zugreift und dabei verschiedene Funktionen einsetzt, die in spidev.h definiert sind. Welcher Mode, welcher Takt und wie viele Bits in welcher Reihenfolge (LSB oder MSB zuerst) dabei notwendig sind, hängt vom angeschlossenen Chip oder Sensor ab.

```
# SPI-Zugriff mit I/O Controls
#include <linux/spi/spidev.h>
// Bus Device oeffnen, File Descriptor fd zuweisen
fd = open("/dev/spi.dev0.0", O_RDWR);
int rc1, rc2, rc3, rc4; // Return Codes
int mode = SPI_CPHA | SPI_CPOL; // Mode setzen
int actual_mode;
int tx_bits = 8; //Sendebits
int rx_bits = 8; //Empfangsbit
// Schreib-Modus festlegen
rc1 = ioctl(fd, SPI_IOC_WR_MODE, &mode);
// Lese-Modus festlegen
rc2 = ioctl(fd, SPI_IOC_RD_MODE, &actual_mode);
// Bit-Anzahl fuer das Schreiben ermitteln
rc3 = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &tx_bits);
// Bit-Anzahl fuer das Lesen ermitteln
rc4 = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &rx_bits);
// Bit-Folge mit LSB in beiden Richtungen zuerst
SPI_IOC_RD_LSB-FIRST
SPI_IOC_WD_LSB-FIRST
// Einzelnes Byte (in &byte) schreiben
char byte = 0;
n = write(fd, &byte, 1); //n = gesendete Byteanzahl
// Einzelnes Byte (in &byte) lesen
n = read(fd, &byte, 1); //n = empfangene Byteanzahl
Close(fd)
```

6.4.2.3 Python-Bibliothek

Für die Erstellung von Programmen, die mit SPI-Einheiten kommunizieren sollen, sind verschiedene spezielle Bibliotheken entwickelt worden, wobei die bekannteste die py-spidev-Bibliothek ist, die für Python bestimmt ist. In der Abbildung 6.22 ist gezeigt, wie sie installiert wird. Die Funktionen der Bibliothek können in Python über die Hilfefunktion mit *import spidev* und der Eingabe von *help(spidev)* eingesehen werden.

```
pi@raspberrypi ~ $ git clone git://github.com/doceme/py-spidev
Cloning into 'py-spidev'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 2), reused 9 (delta 2)
Receiving objects: 100% (9/9), 6.16 KiB, done.
Resolving deltas: 100% (2/2), done.
pi@raspberrypi ~ $ cd py-spidev/
pi@raspberrypi ~/py-spidev $ sudo python setup.py install
running install
running build
running build_ext
building 'spidev' extension
creating build
```

Abbildung 6.22: Installation der py-spidev-Bibliothek

6.4.3 A/D- und D/A-Wandlerschaltung

Für den Anschluss an die SPI-Schnittstelle sind eine ganze Reihe von Wandlerbausteinen erhältlich. Der Aufbau der Schaltung erfolgt auf einer Lochrasterplatine, die sich auf die Raspberry Pi-Platine stecken lässt.

In der Applikation werden die beiden folgenden Typen der Firma Microchip eingesetzt, die einfach direkt mit den SPI-Leitungen verbunden und an der Spannungsversorgung von 5 V betrieben werden.

- **MCP 4802:** Zweikanaliger D/A-Wandler an CE0 (spidev0.0) mit einer Auflösung von 8 Bit. Verstärkungseinstellung (x1, x2), interne Spannungsreferenz, Shutdown-Modus (6 µA), maximaler Takt 33 MHz, typische maximale Ausgangsspannung 4,1 V, maximaler Ausgangsstrom 25 mA.
- **MCP 3208:** Achtkanaliger A/D-Wandler an CE1 (spidev0.1) mit einer Auflösung von 12 Bit. Single-Ended- oder Differenzial-Konfiguration, maximaler Takt 2 MHz, maximale Samplingrate von 100 kBit/s, maximale Eingangsspannung 5 V.

Abbildung 6.23: Die aufgebaute A/D- und D/A-Wandlerschaltung, deren Signale am Rand auf Steckpfosten geführt sind, ist auf die Raspberry Pi-Platine gesteckt.

Welcher Mode, welcher Takt und wie viele Bits in welcher Reihenfolge (LSB oder MSB zuerst) für die Übertragung notwendig sind, hängt generell vom angeschlossenen Chip oder Sensor ab. Wie in Kapitel 6.4.1 erläutert, gibt es vier gebräuchliche SPI-Modi; der hier verwendete Modus ist der Modus 0. Der Zugriff auf die SPI-Schnittstellen erfolgt über *Input/Output Controls* (IOCTL). Im Folgenden ist hierfür Beispielcode gezeigt, der auf die beiden SPI-Schnittstellen zugreift und dabei verschiedene Funktionen einsetzt, die in *spidev.h* definiert sind. Die dabei verwendete Funktion *ioctl* ist wie folgt aufgebaut:

```
int ioctl(int d, int request, &bit);
```

Als Parameter d wird ein geöffnetes Device-File übergeben, an *request* ein geräteabhängiger Code, und als dritter Parameter wird ein Pointer übergeben. Bei erfolgreicher Übertragung gibt die Funktion *ioctl* eine 0 zurück, bei einer gescheiterten Übertragung wird eine -1 zurückgegeben.

Im Programm wird die Initialisierung der SPI-Parameter ausführlich mit der Ausgabe von eventuellen Fehlermeldungen realisiert, was beim Programmablauf entsprechende Bildschirmausgaben zur Folge hat, die natürlich auch weggelassen oder zusammengefasst werden könnten, in den Demo-Programmen jedoch zur Verdeutlichung der einzelnen Schritte explizit angegeben sind. Die drei wesentlichen Teile des Programms sind die Initialisierung des SPI-Interface, die Festlegung der SPI-Übertragungsparameter und die anschließende Datenübertragung zum D/A-Wandler.

Neben dem Einsatz der I/O-Controls ist *struct spi_ioc_transfer* für das Senden und Empfangen von SPI-Daten von Bedeutung. Hier sind die Pflicht-Parameter *.tx_buf*

für die Angabe des Sendestrings und *.rx_buf* für die Angabe des Empfangsstrings mit der Länge in Byte (in *.len*) zu spezifizieren. Falls keine Werte angegeben werden, werden die Strings automatisch mit Nullen gefüllt. Außerdem können hier als optionale Parameter *.speed_hz* für das Überschreiben der spezifizierten Taktfrequenz festgelegt werden sowie die Größe eines Datenpaketes (*.bits_per_word*). Mit dem optionalen Parameter *.delay_usecs* wird für die aktuelle Übertragung die Dauer bestimmt, bis das Device automatisch deaktiviert wird. Der Parameter *.cs_change* sorgt mit der Voreinstellung (true) dafür, dass das Device deaktiviert wird, bevor die nächste Übertragung aktiv werden kann. Als Ausgabe wird der D/A-Wandler-Kanal A (Pin 8) mit der maximalen Spannung eingestellt. Im folgenden Listing ist die genaue Zuordnung der Bits zu den Funktionen des MCP4802 angegeben.

```
/* D/A-Wandler MCP4802 an CE0
/* File: DA_MCP4802.c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // Zum Verwalten von Dateien (open,close..)
#include <fcntl.h>  // Zum Lesen und Schreiben von Dateien
#include <sys/ioctl.h>          // I/O-Controls
#include <linux/spi/spidev.h>   // SPI-Kommunikation
#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
static const char *DAC = "/dev/spidev0.0"; // Interface-Zuweisung
static uint8_t Modus = 0;
static uint8_t bits = 8;   // Anzahl Bits pro Übertragungssequenz
static uint32_t speed = 500000; // Clock-Frequenz in Hz


// Abbruchfunktion bei auftretenden Fehlern
static void pabort(const char *s)
{
    perror(s);
    abort();
}
static void Ausgangsspannung(int DAWandler)
{
    int ret;
 /* Datenstring zum Initialisieren und Übergeben der Werte an den
     * DA-Wandler MCP4802
     * !A/B, x, !GA, !SDHN, D7-D0    , xxxx
     *  0  , 0,  0 ,   1  , 1111 1111, 1111
     * --------1----------, --F-- --F-- --F-;
     * hier: Kanal A (!A/B) und max. Spannung (D7-D0)
     */
    uint8_t tx[] = {
        0x1f , 0xff
```

```
     };
     /* Berechnen der Größe des Arrays für den Empfangsstring anhand
     /* der Länge des Sendestrings
    uint8_t rx[ARRAY_SIZE(tx)] = {0, };


   // Initialisieren einer SPI-Übertragung
    struct spi_ioc_transfer tr = {
        .tx_buf = (unsigned long)tx,
        .rx_buf = (unsigned long)rx,
   // Länge von rx und tx in Byte (hier: =Anzahl der Datenpakete)
        .len = ARRAY_SIZE(tx),
    };
    // Übertragung starten
    ret = ioctl(DAWandler, SPI_IOC_MESSAGE(1), &tr);
    if (ret < 1)
        pabort("SPI-Daten können nicht gesendet werden");
}
int main(int argc, char *argv[])
{
    int ret = 0;
    int DAWandler;
    DAWandler = open(DAC, O_RDWR); // Device öffnen
    if (DAWandler < 0)
        pabort("Device kann nicht geöffnet werden");


    /* Allgemeine Initialisierung der SPI-Parameter: SPI-Modus,
    /* Größe der Datenpakete und Clock-Frequenz

    // SPI-Modus, hier Modus 0
    ret = ioctl(DAWandler, SPI_IOC_WR_MODE, &Modus);
    if (ret == -1)
       pabort("SPI-Modus kann nicht gesetzt werden");
    ret = ioctl(DAWandler, SPI_IOC_RD_MODE, &Modus);
    if (ret == -1)
       pabort("SPI-Modus kann nicht gesetzt werden");


    // Größe der Datenpakete, hier 1 Byte
    ret = ioctl(DAWandler, SPI_IOC_WR_BITS_PER_WORD, &bits);
    if (ret == -1)
      pabort("Größe der Datenpakete kann nicht gesetzt werden");
    ret = ioctl(DAWandler, SPI_IOC_RD_BITS_PER_WORD, &bits);
    if (ret == -1)
      pabort("Größe der Datenpakete kann nicht gesetzt werden");
```

```
    // Max. Clock-Frequenz, hier 500 kHz
    ret = ioctl(DAWandler, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
    if (ret == -1)
      pabort("Max. Clock-Frequenz kann nicht gesetzt werden");
    ret = ioctl(DAWandler, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
    if (ret == -1)
      pabort("Max. Clock-Frequenz kann nicht gesetzt werden");
    printf("SPI-Modus: %d\n", Modus);
    printf("Bits pro Datenpaket: %d\n", bits);
    printf("Max Speed: %d Hz (%d KHz)\n", speed, speed/1000);
    Ausgangsspannung(DAWandler); // Werte an D/A-Wandler geben
    printf("\nAusgabe an Kanal A: 4,08 V\n\n");
    close(DAWandler);            // Device schließen
    return ret;
}
```

Abbildung 6.24: Der Schaltplan der A/D- und D/A-Wandlerschaltung

Das folgende Programm für die A/D-Wandlung ist analog zum obigen für die D/A-Wandlung aufgebaut, so dass die meisten Programmteile bereits bekannt sind. Der A/D-Wandler wird mit der Sequenz *07, C0, 00* als Single-Ended für die Messung auf dem Kanal 7 konfiguriert, wobei die entsprechende Bitzuordnung im Listing zu erkennen ist. Obwohl der A/D-Wandler eigentlich nur 5 Bit für die Konfigurierung benötigt, sind hier für die Übertragung drei Bytes notwendig, was dem Raspberry Pi bzw. dem implementierten SPI-Übertragungsmodus geschuldet ist.

```
/* A/D-Wandler MCP3208 an CE1
/* File: AD_MCP3208.c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>   // Zum Verwalten von Dateien (open,close..)
#include <fcntl.h>    // Zum Lesen und Schreiben von Dateien
#include <sys/ioctl.h>          // I/O-Controls
#include <linux/spi/spidev.h>   // SPI-Kommunikation
#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
static const char *ADC = "/dev/spidev0.1"; // Interface-Zuweisung
static uint8_t Modus = 0;
static uint8_t bits = 8;   // Anzahl Bits pro Übertragungssequenz
static uint32_t speed = 500000; // Clock-Frequenz in Hz
// Abbruchfunktion bei auftretenden Fehlern
static void pabort(const char *s)
{
    perror(s);
    abort();
}
static void getVoltage(int ADWandler)
 {
    int ret;
    float SpannungAD;   // Eingangsspannung speichern
   /* Datenstring zum Initialisieren des AD-Wandlers
* 0000 0, 1 (Start Bit), Single/!Diff, D2-D0, xx xxxx, xxxx  xxxx
* 0000 0,     1        ,       1     , 1 11 , 00 0000, 0000  0000
* -0-- -----------------7-------------- ,---C-- --0--,--0-- --0--
    uint8_t tx[] = { // Sende-Array/String
        0x07,0xC0,0x00
 };


/* Berechnen der Größe des Arrays für den Empfangsstring anhand
 *der Länge des Sendestrings */
    uint8_t rx[ARRAY_SIZE(tx)] = {0, }; // Empfangs-Array/String
   // Initialisieren der SPI-Übertragung
    struct spi_ioc_transfer tr = {
        .tx_buf = (unsigned long)tx,
        .rx_buf = (unsigned long)rx,
   // Länge von rx und tx in Byte (hier: =Anzahl der Datenpakete)
       .len = ARRAY_SIZE(tx),
    };
```

```
    // Übertragung starten
    ret = ioctl(ADWandler, SPI_IOC_MESSAGE(1), &tr);
    if (ret < 1)
        pabort("SPI-Daten können nicht gesendet werden");


    // Daten lesen und umsetzen
    rx[1] = rx[1]<<4;


// Bitverschiebung um 4 nach links um die Bits B11-B8 zu erhalten
  SpannungAD = rx[1]*16;  // In SpannungAD steht B11-B8,0000 0000
  SpannungAD += rx[2];    // B7-B0 addieren fuer B11-B0
  SpannungAD = (SpannungAD*5.0)/4096; // Berechnung der Spannung
    printf("\nSpannung am Kanal 7: %.2f V\n\n", SpannungAD);
            // Ausgabe der Spannung
}
int main(int argc, char *argv[])
{
    int ret = 0;
    int ADWandler;
    ADWandler = open(ADC, O_RDWR); // Device oeffnen
    if (ADWandler < 0)
        pabort("Device kann nicht geöffnet werden");


    /* Allgemeine Initialisierung der SPI-Parameter: SPI-Modus,
     * Größe der Datenpakete und Clock-Frequenz */
    // SPI-Modus, hier Modus 0
    ret = ioctl(ADWandler, SPI_IOC_WR_MODE, &Modus);
    if (ret == -1)
      pabort("SPI-Modus kann nicht gesetzt werden");
    ret = ioctl(ADWandler, SPI_IOC_RD_MODE, &Modus);
    if (ret == -1)
      pabort("SPI-Modus kann nicht gesetzt werden");
    // Größe der Datenpakete, hier 1 Byte
    ret = ioctl(ADWandler, SPI_IOC_WR_BITS_PER_WORD, &bits);
    if (ret == -1)
      pabort("Größe der Datenpakete kann nicht gesetzt werden ");
    ret = ioctl(ADWandler, SPI_IOC_RD_BITS_PER_WORD, &bits);
    if (ret == -1)
      pabort("Größe der Datenpakete kann nicht gesetzt werden ");
    // Max. Clock-Frequenz, hier 500 kHz
    ret = ioctl(ADWandler, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
    if (ret == -1)
      pabort("Max. Clock-Frequenz kann nicht gesetzt werden");
```

```
    ret = ioctl(ADWandler, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
    if (ret == -1)
      pabort("Max. Clock-Frequenz kann nicht gesetzt werden");
    printf("SPI-Modus: %d\n", Modus);
    printf("Bits pro Datenpaket: %d\n", bits);
    printf("Max Speed: %d Hz (%d KHz)\n", speed, speed/1000);
    getVoltage(ADWandler); // Werte vom AD-Wandler auslesen
    close(ADWandler);      // Device schließen
    return ret;
}
```

Die beiden Programme sind für Demozwecke gedacht und so aufgebaut, dass das Funktionsprinzip deutlich wird und sie leicht für andere SPI-Schaltkreise abgewandelt werden können. Die Konfigurierung der Chip-Parameter erfolgt direkt im Quellcode. Es gibt keine Benutzeroberfläche.

Die Kompilierung mit dem C-Compiler (gcc mit Option –o) erfolgt direkt auf der Kommandozeile, etwa mit der folgenden Zeile, um aus dem Quellcode (DA_MCP4802) das ablauffähige Programm DA_MCP4802 zu erzeugen:

```
pi@raspberrypi ~ $  gcc DA_MCP4802.c -o DA_MCP4802
```

```
pi@raspberrypi ~/SPI-Test $ gcc DA_MCP4802.c -o DA_MCP4802
pi@raspberrypi ~/SPI-Test $ sudo ./DA_MCP4802
SPI-Modus: 0
Bits pro Datenpaket: 8
Max Speed: 500000 Hz (500 KHz)

Ausgabe an Kanal A: 4,08 V

pi@raspberrypi ~/SPI-Test $ sudo ./AD_MCP3208
SPI-Modus: 0
Bits pro Datenpaket: 8
Max Speed: 500000 Hz (500 KHz)

Spannung am Kanal 7: 5.00 V

pi@raspberrypi ~/SPI-Test $ sudo ./AD_MCP3208
SPI-Modus: 0
Bits pro Datenpaket: 8
Max Speed: 500000 Hz (500 KHz)

Spannung am Kanal 7: 3.51 V

pi@raspberrypi ~/SPI-Test $
```

Abbildung 6.25: Ausgabebildschirm der beiden Testprogramme mit der Ausgabe von 4,08 V über den D/A-Wandlerkanal A und dem Einlesen zweier Analogwerte über den A/D-Wandlerkanal 7

6.5 Inter Integrated Bus – I²C

Den gebräuchlichsten seriellen Peripheriebus stellt der I²C-Bus dar, der in einer Vielzahl von Geräten, insbesondere in der Consumerelektronik (DVD-Player, Autoradios, allgemeine HiFi-Technik) verwendet wird. Der I²C-Bus (Inter Integrated Circuit Bus) ist eine Entwicklung der Firma Philips Semiconductors (jetzt NXP) aus dem Jahre 1982 für die serielle Verbindung von integrierten Schaltungen auf einer Platine bzw. innerhalb eines Gerätes. Statt zahlreicher paralleler Leitungen werden für die Kopplung der Bauelemente lediglich zwei Verbindungen und eine Masseleitung benötigt.

6.5.1 Betriebsarten

Zehn Jahre nach der Vorstellung des I²C-Bus wurde durch die Firma Philips die Version 1.0 als Standard veröffentlicht (100 kBit/s). Mit der Version 2. 0 (Fast Mode, 400 kBit/s) aus dem Jahre 1998 war bereits ein Standard etabliert, der an 50 Firmen

lizenziert war und für den über 1000 verschiedene ICs existierten. Aus lizenzrechtlichen Gründen bezeichnen einige Hersteller (z.B. Microchip) dieses System auch als *Two Wire Interface* (TWI). Die Entwicklung des Systems ist geprägt durch das Hinzufügen des Fast-Mode-Plus (1 MBit/s) mit der Version 3.0 im Jahre 2007. Es hängt von den einzelnen I²C-Chips ab, welche Betriebsart möglich ist. Diese wird stets vom Master des Systems bestimmt. Die meisten I²C-Chips unterstützen den Standard- und den Fast-Mode. Auch Chips, die explizit den High Speed-Mode unterstützen, können automatisch in eine der »langsameren« Betriebsarten schalten. Der Raspberry Pi unterstützt maximal den Fast Mode mit 400 kBit/s.

Tabelle 6.2: Betriebsarten des I²C-Bussystems

Modus	**Datenrate**
Standard Mode	100 kBit/s
Fast Mode	400 kBit/s
Fast Mode Plus	1 MBit/s
High Speed Mode	3,4 MBit/s
Ultra Fast Mode	5 MBit/s

Die neueste Entwicklung ist der Ultra-Fast-Modus, der einige Besonderheiten mit sich bringt und nicht wie die vorherigen Implementierungen abwärtskompatibel ist. Er arbeitet nur unidirektional bis maximal 5 MHz und ist für die Anbindung von solchen Bausteinen gedacht, die einerseits hohe Transferraten benötigen, andererseits keine Rückmeldung an den Master benötigen, wie etwa Display-Controller oder LED-Driver.

Der I²C-Bus wird auch auf PC-Mainboards eingesetzt. Er dient hier der Kommunikation mit Überwachungsbausteinen (Supervisory Chips oder System Health Chips), die mithilfe von Sensoren die Temperatur der CPU und des PC-Innern überwachen, einen Tachometereingang für die Kontrolle der Drehzahl des CPU-Kühlers und – je nach Typ – noch weitere Optionen besitzen. Der I²C-Bus wird bei dieser Anwendung als SMB (System Management Bus) bezeichnet und kommuniziert außerdem mit dem EEPROM, welches sich auf den Speicher-Modulen befindet und dem BIOS die jeweiligen SDRAM-Parameter für die automatische Konfigurierung mitteilt.

Neben NXP stellen insbesondere Firmen wie Analog Devices, Intel, Texas Instruments, OKI, Toshiba und Sony verschiedene Bauelemente für den I²C-Bus her. Fast jede Firma, die Umsetzerbausteine wie A/D- und D/A-Wandler oder auch »Intelligente Sensoren« (Abbildung 6.26) anbietet, führt entsprechende Chips mit I²C-Bus-Interface.

Abbildung 6.26: Der Luftdrucksensor der Firma Bosch Sensortec wird über den I²C-Bus angesteuert und ausgelesen.

6.5.2 Bus-Kommunikation

Jedes I²C-Bauelement verfügt über eine eigene Adresse und kann in Abhängigkeit von seiner Funktion als Sender (Transmitter) oder Empfänger (Receiver) arbeiten. Des Weiteren kann ein Bauelement entweder als Master, von denen mehrere am Bus möglich sind (Multimasterbus), oder als Slave arbeiten.

Der Raspberry Pi stellt üblicherweise den Master dar. Ein Sensor (Abbildung 6.26) oder auch Umsetzerchip bildet dann den Slave. Gleichwohl könnte der Raspberry Pi auch als Slave – etwa als Messknoten – mit einem Master, der von einem anderen Mikroprozessor oder Computer gebildet wird, per I²C-Bus kommunizieren. Ein Multimaster-System ist ebenfalls realisierbar.

Die räumliche Ausdehnung eines I²C-Bussystems kann nur wenige Meter betragen. Zwei Busleitungen sind für die Datenübertragung zwischen den ICs zuständig: eine Datenleitung (SDA, Serial Data- Line) und eine Taktleitung (SCL, Serial Clock Line). Innerhalb der ICs befinden sich Transistoren mit Open-Collector- oder mit Open-Drain-Ausgängen, wodurch sich eine Wired-AND-Verknüpfung aller Busteilnehmer ergibt.

Abbildung 6.27: Über die Innenschaltung der Busteilnehmer findet eine Wired AND-Verknüpfung statt.

Die beiden Datenbussignale werden über Pull Up-Widerstände mit der positiven Versorgungsspannung verbunden, was für die beiden I²C-Bus-Signale (SDA: Pin 3, SCL: Pin 5) bereits auf der Raspberry Pi-Platine durchgeführt wurde, so dass diese Pins auch dann, wenn sie nicht explizit für als I²C-Leitungen konfiguriert wurden, sondern standardmäßig als GPIO-Leitungen definiert sind, stets einen High-Pegel aufweisen, solange sie nicht extern auf Low gezogen werden.

Die Generierung des Taktsignals wird immer von einem Master-IC vorgenommen. Dabei kann jeder Master sein eigenes Taktsignal erzeugen, sobald er für die Datenübertragung verantwortlich ist. Mit den AND-Verknüpfungen über die SCL-Leitung findet eine synchronisierte Kombination der einzelnen Taktsignale statt, wodurch Teilnehmer mit unterschiedlichen Taktraten am Bus verwendet werden können und die Erlangung der Buszugriffsrechte (Arbitration) durchgeführt wird.

Findet keine Busaktivität statt, befinden sich die Signale SDA und SCL auf High-Pegel. Die Datenübertragung beginnt mit dem Übergang des Datensignals (SDA) von High nach Low, während SCL auf High verbleibt, was als *Start Condition* bezeichnet wird.

Die Pegel für High und Low sind nicht fix, sondern aufgrund der unterschiedlichen Chiptechnologien (bipolar, NMOS, CMOS), mit denen die Chips aufgebaut sein können, abhängig von der Betriebsspannung V_{DD} . Ein Low (V_{IL}) entspricht 0,3 V * V_{DD} und ein High (V_{IH}) 0,7 V * V_{DD} .

Abbildung 6.28: Die Datenübertragung beginnt mit einer Start- und endet mit einer Stoppbedingung.

Die übertragenen Daten sind nur dann gültig, wenn sich SCL auf High-Pegel befindet. Mit jedem Taktimpuls wird ein Bit übertragen. Das Ende der Datenübertragung (Stop Condition) ist durch einen Low-High-Übergang des Datensignals gekennzeichnet, wobei SCL wieder High-Pegel aufweist.

Nach der Startbedingung (Start Condition) folgt eine 7-Bit-lange Slave-Adresse, die in Abhängigkeit vom jeweiligen I²C-Bus-Bauelement in gewissen Grenzen in der Schaltung angepasst werden kann (vgl. Abbildung 6.30). Außerdem gibt es eine Erweiterung für ein 10-Bit-Adressierungschema, welches allerdings kaum verwendet wird. Es können sowohl Einheiten mit einer 7-Bit als auch mit einer 10-Bit-Adressierung am gleichen I²C-Bus eingesetzt werden.

Das MSB wird sowohl bei der Adresse als auch bei den Daten immer zuerst auf dem Bus übertragen. Mit dem darauf folgenden R/W-Bit wird die Datenübertragungsrichtung festgelegt, wobei jede Datenübertragung mit *Acknowledge* bestätigt wird. Der Master setzt hierfür die SDA-Leitung auf High und erwartet vom Slave, dass er nach der Datenübertragung das SDA-Signal auf Low zieht. Geschieht dies nicht, bricht der Master die Übertragung ab (Stop Condition). Ist ein Master hingegen der Datenempfänger, belässt der Slave das SDA-Signal auf High, setzt es selbst zurück und führt die *Stop Condition* durch.

Abbildung 6.29: Datenformat für Lese- und Schreiboperationen

Als Datentelegramm ist ebenfalls eine Kombination aus Lese- und Schreibzugriffen möglich. Nach der ersten *Stop Condition* folgt dann unmittelbar ein zweites Telegramm, welches ebenfalls dem Aufbau laut Abbildung 6.29 entspricht.

Zur Erkennung von Übertragungsfehlern existieren keine speziellen Vorkehrungen, wie etwa Verfahren mit Parität oder Prüfsumme. Dies muss allein durch die Software realisiert werden. In den meisten Fällen werden durch integrierte Tiefpassfilter lediglich hochfrequente Störungen auf der SDA- und der SDC-Leitung unterdrückt.

6.5.3 Adressen

Jedes Bauelement muss am Bus über eine jeweils eigene Adresse verfügen. Die I²C-Bus-Adresse eines Chips kann meist in einem gewissen Bereich durch nach außen geführte Anschlüsse angepasst werden, die entweder an Plus oder die Schaltungsmasse gelegt werden, womit die Adresse bestimmt wird. Die Anzahl der programmierbaren Adressbits hängt von der Anzahl der hierfür vorgesehenen Anschlüsse am IC ab (vgl. Abbildung 6.30). Hat ein IC beispielsweise vier feste und drei programmierbare Adressbits, können insgesamt acht ICs des gleichen Typs in einer Schaltung verwendet werden.

Daneben existieren auch I²C-Bus-Chips, wie der Luftdrucksensor BMP085 (Abbildung 6.26), bei denen die Adresse bei der Herstellung festgelegt worden ist und nicht geändert werden kann. Für den Fall, dass mehrere dieser Chips am gleichen I²C-Bus betrieben werden sollen, gibt es – je nach Typ und Hersteller – verschiedene Vorgehensweisen. Die gebräuchlichste Methode ist es dann, alle Chips des gleichen Typs bis auf einen abzuschalten (via XLR-Pin beim BMP085) oder auch in den Sleep-Modus zu versetzen.

Interface Definition

BYTE	BIT							
	7 (MSB)	6	5	4	3	2	1	0 (LSB)
I²C slave address	L	H	L	L	A2	A1	A0	R/$\overline{W}$
I/O data bus	P7	P6	P5	P4	P3	P2	P1	P0

Abbildung 6.30: Die I²C-Bus-Adresse kann beim Port-Expander PCF8574, der einen 8-Bit-breiten Parallel-Port (P0-P7) bereitstellt, im Adressbereich von 20-27h (Interface Definition) festgelegt werden.

Einige Adressen sind für spezielle Aufgaben vorgesehen, die in der Tabelle 6.3 angeführt sind. Durch die *General Call Address* werden gleichzeitig alle Slaves angesprochen, die, falls sie diese Funktion unterstützen, mit einem *Acknowledge* antworten. Die anderen ICs können diese Adresse einfach ignorieren. Der *General Call Address* folgt ein zweites Byte, welches die eigentliche Aktion festlegt und auch als Reset-Funktion dient.

Tabelle 6.3: Reservierte Adressen und deren Funktionen

Slave Adresse	R/W	Bezeichnung	Bedeutung/Funktion
0000 000	0	General Call Address	Verschiedene Funktionen inklusive Reset
0000 000	1	Start Byte	Beginn des Datenframe, kein Acknowledge
0000 001	X	CBUS Address	Für CBUS-ICs
0000 010	X	Reserved	Reserviert für andere Busformate
0000 011	X	Reserved	Reserviert für zukünftige Entwicklungen
0000 1XX	X	HS-Mode	High Speed Mode Master Code
1111 0XX	X	10-Bit-Mode	10-Bit Slave-Adresse
1111 1001	1	Device ID	Auslesen der Device ID (NXP, Ramtron, Analog Devices, STM, On Semic., Fujitsu)
1111 1XX	X	Reserved	Reserviert für zukünftige Entwicklungen

Das LSB (B) des zweiten Bytes hat noch eine besondere Bedeutung. Enthält dieses Bit eine Null, gelten die in der Tabelle 6.3 angegebenen Funktionen, enthält es hingegen eine Eins, stellt dies einen *Hardware General Call* dar.

Abbildung 6.31: Entspricht das erste Byte der »General Call Address« (00000000), legt ein zweites Byte die darauffolgende Aktion fest.

Master-ICs, die keine Slave-Adresse aussenden können (Master Transmitter), wie beispielsweise ein Tastatur-Scanner, verwenden beispielsweise diese Funktion. Weil dieser Master seinen Adressaten nicht kennt, verwendet er lediglich den *Hardware General Call*, gibt seine Adresse dem System bekannt und sendet seine Daten. Die weitere Verarbeitung übernimmt dann ein anderer Master im System.

6.5.4 Programmierung

Wie bei der SPI-Schnittstelle erläutert, wird für Nutzung des I²C-Bus ebenfalls die Freigabe (siehe Blacklist) des dazugehörigen Moduls (i2c_bcm2708) benötigt. Nach einem Neuboot kann mit der folgenden Eingabe die I²C-Bus-Unterstütung angezeigt werden.

```
pi@raspberrypi ~ $ ls /sys/bus/i2c
```

Die beiden Einheiten (i2c-0, i2c-1) müssen ebenfalls im Verzeichnis `/dev` vorhanden sein, damit sie als Geräte (device) behandelt werden können, was bei den Einstellungen für die SPI-Schnittstelle automatisch passiert, während dies beim I²C-Bus manuell durchzuführen ist mit:

```
pi@raspberrypi ~ $ sudo modprobe i2c-dev
```

Daraufhin steht ein neues Verzeichnis mit zwei i2c-Einträgen (Abbildung 6.32) zur Verfügung. Prinzipiell müsste *modprobe* nach jedem Boot neu aufgerufen werden. Praktischer erscheint es, wenn dieses Modul, was genauso mit dem SPI-Modul durchgeführt werden kann, in *modules* mit eintragen wird, woraufhin es beim jedem Boot automatisch mit geladen wird.

```
pi@raspberrypi ~ $ ls /sys/bus/i2c
devices  drivers  drivers_autoprobe  drivers_probe  uevent
pi@raspberrypi ~ $ ls -l /dev/i2c*
ls: Zugriff auf /dev/i2c nicht möglich: Datei oder Verzeichnis nicht gefunden
pi@raspberrypi ~ $ sudo modprobe i2c-dev
pi@raspberrypi ~ $ ls -l /dev/i2c*
crw------- 1 root root 89, 0 Jun 16 10:53 /dev/i2c-0
crw------- 1 root root 89, 1 Jun 16 10:53 /dev/i2c-1
pi@raspberrypi ~ $
```

Abbildung 6.32: Anlegen des Moduls im Verzeichnis /dev

Obwohl der Raspberry Pi lediglich eine einzige I²C-Schnittstelle am GPIO-Port besitzt, tauchen hierfür zwei Einträge auf, wie es in der Abbildung 6.32 zu erkennen ist. Dies ist dem Umstand geschuldet, dass es hier einen Unterschied zwischen der ersten und der zweiten Platinenversion gibt. Prinzipiell verfügt der Raspberry Pi tatsächlich über zwei I²C-Schnittstellen, wobei sich die zweite am *Camera Serial Interface* (CSI, Kapitel 3.9) befindet. Die Schaltung hierfür ist bei der zweiten Version geändert worden und die die beiden Interfaces sind gegeneinander getauscht, was zwar nichts an der Anschlussbelegung (SDA: Pin 3, SCL: Pin 5) am Connector ändert, jedoch die software-technische Zuordnung beeinflusst. Bei den aktuellen Platinenversionen (Revision 2) wird mit `/dev/i2c-1` die I²C-Schnittstelle am GPIO-Connector angesprochen.

```
  GNU nano 2.2.6                    Datei: /etc/modules

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
spi-bcm2708
i2c-dev

Dateiname zum Speichern: /etc/modules
^G Hilfe          M-D DOS-Format      M-A Anhängen        M-B Sicherungskopie
^C Abbrechen      M-M Mac-Format      M-P vorn Anfügen
```

Abbildung 6.33: Eintragen der I²C- und der SPI-Module in /etc/modules, damit sie nach jedem Boot automatisch zur Verfügung stehen.

Grundsätzlich kann die Kommunikation wie bei SPI mit *Input/Output Controls* (siehe oben Systemzugriff mit Input/Output Controls) erfolgen, wie es im Folgenden beispielhaft gezeigt ist.

```
# I2C-Zugriff mit I/O Controls
#include <linux/i2c-dev.h>
// Bus Device oeffnen, File Descriptor fd zuweisen
fd = open("/dev/i2c-1", O_RDWR);
int rc // Return Code
// Device mit der Adresse 30 selektieren
rc = ioctl(fd, I2C_SLAVE, 0x30);
// Einzelnes Byte (in &byte) schreiben
char byte = 0;
n = write(fd, &byte, 1); //n = gesendete Byteanzahl
// Einzelnes Byte (in &byte) lesen
n = read(fd, &byte, 1); //n = empfangene Byteanzahl
close(fd)
```

Die I²C-Programmiermöglichkeiten sind im Vergleich zu SPI umfassender und lassen sich mit einem Standard-Chip zugleich einfacher durchführen, weil keine besonderen Betriebsarten oder bausteinspezifische Einstellungen notwendig sind. Außerdem gibt es für die I²C-Schnittstelle bessere Dokumentationen und einige separate Tools. Insbesondere das *Command Line Tool* mit der Bezeichnung *i2c-tools* erweist als nützlich, welches wie folgt zu installieren ist:

```
pi@raspberrypi ~ $ sudo apt-get update && apt-get install i2c-tools
```

Wie bei den anderen Installationen auch ist auf jeden Fall zuvor die Datenbank zu aktualisieren (apt-get update), was mit der obigen, kombinierten Befehlszeile auch gleich durchgeführt wird.

Damit stehen die folgenden nützlichen Kommandos für den I²C-Test zur Verfügung:

- **i2cdetect:** Sucht die angeschlossenen I²C-Bus Devices und zeigt deren Adressen an.
- **i2cget:** Liest einen Registerwert vom Device
- **i2cset:** Setzt einen Registerwert im Device
- **i2cdump:** Zeigt die Werte eines Device

Zu diesen einzelnen Befehlen lassen sich mit *man* entsprechende Hilfedateien anzeigen, beispielsweise mit *man i2cget* zum i2cget-Aufruf.

```
pi@raspberrypi ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- 4f
50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi ~ $ i2cget -y 1 0x48
0xa9
pi@raspberrypi ~ $
```

Abbildung 6.34: Detektierung angeschlossener I²C-Bus-Chips und Lesen eines Registers vom Chip mit der Adresse 48h

Neben der Programmierung in C per I/O-Controls (i2c-dev.h) existieren natürlich noch weitere Möglichkeiten. Beispielsweise können fcntl-Routinen (fcntl.h) eingesetzt werden, die die I/O-Controls in einfachere I²C-Aufrufe wie *write* und *read* »verpacken«. Im folgenden Kapitel ist hierfür ein Beispiel gezeigt. Eine spezielle Python-Bibliothek ist für die Programmierung des I²C-Bus ebenfalls verfügbar (`https://github.com/HappyFox/Python-i2c`).

6.5.5 Applikation

Im Gegensatz zu SPI gibt es mit I²C-Bauelementen selten Probleme bei der Ansteuerung, was an dem wesentlich genauer gefassten Standard liegt. Wie erwähnt, befinden sich die notwendigen Pull-Up-Widerstände standardmäßig auf der Raspberry Pi-Platine und sind in einer eigenen Schaltung deshalb nicht notwendig. Möglicherweise macht es dennoch Sinn, welche vorzusehen, um den Signalpegel

gegebenenfalls etwas anpassen zu können, was jedoch nur dann notwendig werden kann, wenn mehrere Chips mit dem I²C-Bus verbunden sind.

Zu beachten ist stets, dass jeder am Bus verwendete Chip eine eigene Adresse besitzen muss und dass der Takt nicht zu hoch gewählt wird, der typischerweise 100 kHz (100 kBit/s, vgl. Tabelle 6.2) beträgt, wie es beim Raspberry Pi auch als Voreinstellung gilt. Der Takt sollte sich stets am langsamsten Chip, der am Bus angeschlossen ist, orientieren, denn die schnelleren können üblicherweise auch mit einem geringeren Takt umgehen. Gleichwohl ist stets die Taktangabe (bzw. die spezifizierte Datenrate, die sich hiermit ergibt) im jeweiligen Datenblatt des Chips von Bedeutung, denn es ist keineswegs sichergestellt, dass mit einem Takt, der nicht einem Standard entspricht, zuverlässig gearbeitet werden kann.

In der Abbildung 6.35 ist eine Applikation mit vier Chips gezeigt, die per I²C-Bus angesteuert werden. Gemäß der Abbildung 6.34 sind sie alle vier korrekt erkannt worden. Mit der Schaltung sind eine Vielzahl von Applikationen möglich, denn neben einem Temperatursensor und einem vierkanaligen A/D-Wandler, der außerdem noch einen D/A-Wandler beinhaltet, ist hier ein Clock/Calendar-Chip vorgesehen. Damit verfügt der Raspberry Pi über eine Echtzeituhr, wie es insbesondere für Emdedded-Anwendungen, die nicht direkt mit dem Internet kommunizieren können oder sollen, nützlich ist.

- **PCF8583 (Adresse 50h)**: Clock/Calendar
- **PCF8591 (Adresse 4Fh)**: Vierkanaliger A/D-Wandler, einkanaliger D/A-Wandler
- **PCF8574 (Adresse 20h)**: 8 Bit I/O-Expander
- **DS1621 (Adresse 48h)**: Temperatursensor

Damit die Uhr auch ohne vorhandene Betriebsspannung (3,3 V) weiterläuft, ist zur Energiespeicherung ein Goldcap-Kondensator (RTCcap, 1 F) auf der Platine vorgesehen. Jedes Mal, wenn die Spannungsversorgung für die Schaltung zur Verfügung steht, wird der Kondensator (nach)geladen.

Abbildung 6.35: Die Schaltung erweitert den Raspberry Pi um einen Temperatursensor, um eine Echtzeituhr, um digitale und analoge Ein- und Ausgänge und kann ein preisgünstiges LC-Display ansteuern.

Der 8 Bit I/O-Expander (siehe auch Abbildung 6.30) ist einer der klassischen I²C-Bus-Chips, der einen parallelen 8-Bit-Port für den Anschluss unterschiedlicher digitaler Bauelemente (LED, Taster, Relais etc.) bildet. Diese Signale sind an den Connector IO-CONN2 gelegt.

Der Port-Expander hat in der Schaltung noch eine zweite Funktion, denn am Connector DISP_CONN kann ein übliches preisgünstiges LC-Display (Abbildung 6.36) angeschlossen werden, so dass der Raspberry Pi damit über eine eigene Anzeige verfügt. Passende Displays (z.B. TC1602E-01) sind bereits ab ca. 5 € bei den bekannten Elektronik-Firmen wie Pollin oder Reichelt erhältlich. Im Grunde genommen können hier alle zum Standard-Controller HD44780 kompatiblen Displays mit zwei Zeilen à 16 Zeichen angeschlossen werden. Die Ansteuerung erfolgt dabei im 4-Bit-Modus (Nibble Mode), so dass nur die Datenleitungen DB4-DB7 des Displays verwendet werden.

Pin-Belegung

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
NAME	Vdd	Vss	V0	RS	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	A	K

Abbildung 6.36: Ein handelsübliches und preisgünstiges zweizeiliges LC-Display lässt sich mit der Schaltung ansteuern.

Das Potentiometer P1 (10 kΩ, CT) ist für die Einstellung des Kontrastes und das Potentiometer P2 (100 Ω, BL) mit dem Transistor für die Einstellung der Hintergrundbeleuchtung (Back Light) des Displays zuständig. Die Platine ist so ausgelegt, dass der PCF8574 entweder als universeller Port-Expander oder als LCD-Interface arbeitet, wofür die beiden unterschiedlichen Kontaktleisten vorhanden sind, wovon jeweils nur eine verschaltet werden darf.

In der Abbildung 6.37 ist die aufgesteckte Platine gezeigt, bei der ganz bewusst keine SMD-Bauelemente verwendet worden sind, sondern kostengünstige ICs im DIP-Gehäuse, die in passende Fassungen eingesteckt werden, was den Nachbau sehr erleichtert. Die Platine muss nicht voll bestückt werden, sondern nur jeweils mit denjenigen Chips und den dazugehörigen peripheren Bauelementen (R, C, Quarz, Potis etc.), die tatsächlich eingesetzt werden sollen.

Abbildung 6.37: Die Erweiterungsplatine in der komplett bestückten Version

Für die Inbetriebnahme ist es zunächst wichtig, dass alle bestückten Chips korrekt erkannt werden, wie es in der Abbildung 6.34 dargestellt ist. Je nach gewünschter Applikation ist die Software unterschiedlich zu gestalten. Dabei empfiehlt es sich zunächst, möglichst viele Ausgabezeilen (mit printf) vorzusehen, um die korrekte Abarbeitung der einzelnen Schritte kontrollieren zu können.

6.5.5.1 Real Time Clock

Für die Echtzeituhr vom Typ PCF8583 (Clock/Calendar) sind im Folgenden zwei Programme gezeigt: Eines zum Stellen der Uhr und das andere zum Lesen der aktuellen Uhrzeit und des Datums. Hierfür gibt es eine eigene Treiberdatei (Bibliothek) mit der Bezeichnung *rtc.c* und die passende, einzubindende Header-Datei (*rtc.h*). Beim blockweisen Lesen und Schreiben von Daten mithilfe der im i2c-Treiber (linux/i2c-dev.h) enthaltenen Funktionen tritt mitunter ein Fehler auf, weshalb die Daten stets einzeln als Byte oder Word ausgelesen und übertragen werden.

```
# Echtzeituhr-Code (rtc.c)

#include <stdio.h>
#include <stdio.h>              // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>      // i2c-Kerneltreiber
#include <fcntl.h>              // Dateizugriffssteuerung

// PCF8583-Adresse
#define pcf8583_addr 0x50
```

```
// Umwandlung von bin- in bcd-Format und umgekehrt
#define  bin2bcd(b)  (((b)%10) | (((b)/10)<<4))
#define  bcd2bin(b)  (((b)&0xf) + ((b)>>4)*10)

// Array zum Speichern des ausgelesenen Datums, der Uhrzeit
int8_t data[7];

// Struktur zur Speicherung des Datums, der Uhrzeit
typedef struct{
      uint8_t seconds;
      uint8_t minutes;
      uint8_t hours;
      uint8_t day;
      uint8_t month;
      uint8_t year;
      uint8_t weekday;
}date_time_t;

//Funktion zum Einstellen des Datums
void start_clock(int8_t sec,int8_t min,int8_t hour,int8_t
days,int8_t wday,int8_t months,int8_t year){

int device_pcf8583;  // Platzhalter fuer das geoeffnete Geraet
      const int i2c_address = pcf8583_addr; // i2C-Adresse setzen
      const char *i2cDevice = "/dev/i2c-1";
       // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"

      // i2c-Interface oeffnen
      if((device_pcf8583 = open(i2cDevice, O_RDWR)) < 0 ) {
   printf("Der i2c-Port konnte nicht geoeffnet werden! \n");
      exit(1);
      }

      // i2c-Adresse zuweisen
      if(ioctl(device_pcf8583, I2C_SLAVE, i2c_address) < 0 ){
   printf("Die Konfiguration der i2c-Adresse ist fehlgeschlagen!
            \n");
      exit(1);
      }

      //Uhr stoppen, schreiben von 0x80 in Controlregister 0x00
      i2c_smbus_write_byte_data(device_pcf8583, 0x00,0x80);

      // Setzen der Zeit
  i2c_smbus_write_byte_data(device_pcf8583, 0x01, 0x00);
  i2c_smbus_write_byte_data(device_pcf8583, 0x02, bin2bcd(sec));
  i2c_smbus_write_byte_data(device_pcf8583, 0x03, bin2bcd(min));
  i2c_smbus_write_byte_data(device_pcf8583, 0x04, bin2bcd(hour));
  i2c_smbus_write_byte_data(device_pcf8583, 0x05, (bin2bcd(days)
       | (year << 6)));

  i2c_smbus_write_byte_data(device_pcf8583, 0x06, (bin2bcd(
       months) | (wday << 5)));

      //Information ueber das Jahr im zusaetzlichen RAM speichern
      i2c_smbus_write_byte_data(device_pcf8583, 0x10, year);

      //Uhr starten
       i2c_smbus_write_byte_data(device_pcf8583,0x00,0x00);
```

```
        close(device_pcf8583);
}
// Funktion zum Auslesen der Zeit. Es wird ein Pointer auf das
// Array  mit dem ausgelesenem Datum zurueckgegeben.
int8_t* read_time(void){
    date_time_t dt;     // struct fuer das Datum
    int device_pcf8583; // Platzhalter fuer das geoeffnete Geraet
    const int i2c_address = pcf8583_addr;   // i2c-Adresse const
    char *i2cDevice = "/dev/i2c-1";
       // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"
      uint8_t temp;
      // i2c-Interface oeffnen
      if((device_pcf8583 = open(i2cDevice, O_RDWR)) < 0 ) {
      printf("Der i2c Port konnte nicht geoeffnet werden! \n");
      exit(1);
      }
      //Die i2c Adresse zuweisen
      if(ioctl(device_pcf8583, I2C_SLAVE, i2c_address) < 0 ){
    printf("Die Konfiguration der i2c Adresse ist fehlgeschlagen!
            \n");
             exit(1);
      }
// Auslesen der Daten aus den Registern 0x02 - 0x06
dt.seconds = bcd2bin(i2c_smbus_read_byte_data(device_pcf8583,0x02));
dt.minutes = bcd2bin(i2c_smbus_read_byte_data(device_pcf8583,0x03));
dt.hours = bcd2bin(i2c_smbus_read_byte_data(device_pcf8583,0x04) &
0x3F);
      temp = i2c_smbus_read_byte_data(device_pcf8583,0x05);
      dt.year = temp >> 6;
      dt.day  = bcd2bin(temp&0x3f);
      temp = i2c_smbus_read_byte_data(device_pcf8583,0x06);
      dt.month  = bcd2bin(temp & 0x1F);
      dt.weekday = temp >> 5;
    // i2C-Interface schliessen
    close(device_pcf8583);
// Zeit in ein Array uebertragen. Dadurch kann ein Pointer auf
// dieses Array zurueckgegeben und die Daten in einer anderen
// Funktion weiterverarbeitet werden.
      data[0] = dt.seconds;
      data[1] = dt.minutes;
      data[2] = dt.hours;
      data[3] = dt.day;
      data[4] = dt.weekday;
      data[5] = dt.month;
      data[6] = dt.year;
      // Rueckgabe des Pointers auf das Datenarray
      return data;
```

```
}
```

rtc.h:

```
#endif

#ifndef RTC_H
#define RTC_H

int start_clock(int8_t sec,int8_t min,int8_t hour,int8_t days,int8_t
                wday, int8_t months,int8_t years);

int read_time(void);

#endif
```

Das folgende Programm *set_time.c* setzt eine Uhrzeit und ein Datum in der Real Time Clock und startet sie anschließend. Nacheinander werden Sekunden, Minuten, Stunden, Tage, Monate, Jahre und der Wochentag abgefragt. Die Jahre sind hier nicht als absoluter Wert zu verstehen, weil der Chip nur vier Jahre zählen kann.

```
# Uhrzeit und Datum stellen (set_time.c)

#include <stdio.h>            // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include "rtc.h"               // PCF8583-Funktionen

/*Hauptprogramm*/

int main(void)

{ uint8_t time[7];             // Array für Zeitdaten
  uint16_t offset_jahr = 2014; // Startjahr

  printf("Bitte Sekunden eingeben [0-59]: ");
  scanf("%d",&time[0]);

  printf("Bitte Minuten eingeben [0-59]: ");
  scanf("%d",&time[1]);

  printf("Bitte Stunden eingeben [0-23]: ");
  scanf("%d",&time[2]);

  printf("Bitte Datum/Tag eingeben [1-31]: ");
  scanf("%d",&time[3]);

  printf("Bitte Wochentag eingeben [0-6]: ");
  scanf("%d",&time[4]);

  printf("Bitte Datum/Monat eingeben [1-12]: ");
  scanf("%d",&time[5]);

  printf("Bitte Datum/Jahr eingeben [0-3]: ");
  scanf("%d",&time[6]);

  printf("Uhrzeit/Datum -->   %d:%d:%d -- %d/%d/%d -- Wochentag:
         ",time[2],time[1],time[0],time[3],time[5],time[6]);

  switch(time[4]){

                  case 0: printf("Montag \n"); break;
                  case 1: printf("Dienstag \n"); break;
```

```
                    case 2: printf("Mittwoch \n"); break;
                    case 3: printf("Donnerstag \n"); break;
                    case 4: printf("Freitag \n"); break;
                    case 5: printf("Samstag \n"); break;
                    case 6: printf("Sonntag \n"); break;
                    default: printf("error"); break;
                 }
  printf("Die Uhr wird gestartet ... \n");
  start_clock(time[0],time[1],time[2],time[3],time[4],time[5],
              time[6]);  // Setzen der Uhrzeit und Uhr starten
  return 0;
}
```

Das Programm get_time liest die in der RTC aktuell gespeicherte Uhrzeit und das Datum und gibt sie der Konsole aus. Ein erneutes Auslesen wird per Eingabetaste (Enter) initiiert.

```
pi@raspberrypi ~/RasPi_i2c/sensnode/set_time $ sudo ./set_time
Bitte Sekunden eingeben [0-59]: 0
Bitte Minuten eingeben [0-59] 8
Bitte Stunden eingeben [0-23]: 15
Bitte Tag eingeben [1-31]: 17
Bitte Wochentag eingeben [0-6]: 2
Bitte Monat eingeben [1-12]: 12
Bitte Jahr eingeben [0-3]: 0
Uhrzeit/Datum: 15:8:0 -- 17/12/0 -- Wochentag: 2
 Die Uhr wird gestartet ...
pi@raspberrypi ~/RasPi_i2c/sensnode/set_time $
pi@raspberrypi ~/RasPi_i2c/sensnode/get_time $ sudo ./get_time
Uhrzeit/Datum: 15:11:45 -- 17/12/0 -- Dienstag
```

Abbildung 6.38: Uhr stellen und lesen

```
# Uhrzeit und Datum lesen (get_time.c)
#include <stdio.h>          // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include "i2c_lcd_drv_2.h"  // LCD-Treiber (HD44780-kompatibel)
#include "rtc.h"            // PCF8583-Funktionen
/*Hauptprogramm*/
int main(void)
{
      char str[10];             // String-Zwischenspeicher
      char time_str[25];        // String zum Umwandeln des Datums
      uint16_t offset_year = 2014;   // Jahr ab dem das Datum zählt

      uint8_t line,row;       // Variable für Displayausgabe
```

```
    int8_t* x = NULL;
    // Pointer zum Speichern der eingelesenen Zeit
    // Einlesen der Daten vom PCF8583 und Ausgabe
    x = read_time();  // Auslesen der aktuellen Zeit und
                      // Übergabe an lokalen Pointer
    printf("Uhrzeit/Datum: %d:%d:%d -- %d/%d/%d -- Wochentag:"
            ,*(x+2),*(x+1),*x,*(x+3),*(x+5),2014+*(x+6));
    switch(*(x+4)){
                case 0: printf("Montag \n"); break;
                case 1: printf("Dienstag \n"); break;
                case 2: printf("Mittwoch \n"); break;
                case 3: printf("Donnerstag \n"); break;
                case 4: printf("Freitag \n"); break;
                case 5: printf("Samstag \n"); break;
                case 6: printf("Sonntag \n"); break;
                default: printf("error"); break;
            }
    return 0;
}
```

Die Programme können wie üblich mit dem C-Compiler gcc kompiliert und gelinkt werden. Weil es hier für jedes Programm mehr als jeweils nur eine einzige Quelldatei gibt bzw. weil mehrere Dateien zu einem ausführbaren Programm zusammenzuführen sind, empfiehlt sich der Einsatz einer Make-Datei, damit keine langen Befehlszeilen einzugeben sind.

Alle für das jeweilige Projekt benötigten Dateien werden hierfür in einem eigenen Verzeichnis gespeichert, und es wird eine passende Make-Datei angelegt. Allein durch den Aufruf von make – im jeweiligen Verzeichnis – wird daraufhin der Kompilierungsvorgang komplett bis zur Generierung des ausführbaren Programms durchgeführt. Der Aufbau einer Make-Datei ist allgemein wie folgt:

```
OBJS = x.o y.o z.o      // Hier werden die Objektdateien angegeben
TARGET = programm_xyz   // Dateiname des Programms
CFLAGS = -Os            // Compilerflags z.B. hier -0s
                        // Optimierung auf Dateigröße
all: $(TARGET)          // Alle Objektdateien neu kompilieren,
                        // Aufruf mit "make all"
$(TARGET): $(OBJS)      // Objektdateien die zum Target gehören
    $(CC) $(CFLAGS) -o $@ $^ // $CC gibt den C-Compiler an
                             // $CFLAGS übergibt die oben
                             // spezifizierten Compilerflags
                             // $@ entspricht dem Namen der
                             // Zieldatei über(TARGET)
```

```
                                // $^ übergibt die Liste der
                                // Abhängigkeiten an den Compiler
clean:                          // Aufräumen (Aufruf mit "make clean")
      rm -rf $(OBJS)            // Objektfiles werden gelöscht
      rm -f $(TARGET)           // Programmdatei löschen
```

Als Beispiel folgt die Make-Datei für *get_time*:

```
OBJS = get_time.o i2c_lcd_drv_2.o rtc.o
TARGET = get_time

CFLAGS = -Os

all: $(TARGET)

$(TARGET): $(OBJS)
       $(CC) $(CFLAGS) -o $@ $^

clean:

       rm -rf $(OBJS)
       rm -f $(TARGET)
```

In der ersten Zeile findet sich die Angabe der einzelnen Objekt-Dateien und in der zweiten der Name des Programms, das daraus erzeugt werden soll. Für die anderen Programme (temp_test, adc_test, dac_test, set_time) sind an diesen Stellen die entsprechenden (passenden) Angaben notwendig. Neben den hier verwendeten Optionen gibt es zur Steuerung des Make-Vorgangs eine ganze Reihe weiterer, die beispielsweise unter `http://www.gnu.org/software/make/` erläutert sind.

6.5.5.2 A/D- und D/A-Wandler

Im Baustein PCF8591 sind ein vierkanaliger A/D- und einkanaliger D/A-Wandler mit einer Auflösung von jeweils 8 Bit integriert, die mit den beiden folgenden Programmen angesprochen werden. Die hierfür notwendigen Funktionen werden in pcf8591_demo.c implementiert und über *pcf8591_demo.h* eingebunden.

Die Funktion *float adc_conv(int8_t channel)* dient dem Einlesen eines Spannungswertes und gibt die A/D-gewandelte Spannung als float-Zahl zurück. Weil der PCF8591 über vier A/D-Wandler verfügt, erwartet diese Funktion als Argument den zu messenden Messkanal (0-3). Die zweite Funktion *void dac_out(int8_t value)* gibt einen als Argument übergebenen Wert (8-Bit) über den D/A-Wandler als analogen Spannungswert aus.

```
# pcf8591_demo.c

#include <stdlib.h>
#include <stdint.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>

// I2C-Adresse

#define PCF8591_adr 0x4F;
```

```
float adc_conv(int8_t channel){
int device_pcf8591;  // Platzhalter fuer das geoeffnete Geraet
      const int i2c_address = PCF8591_adr;  // Chipadresse
      float voltage;       // Variable fuer die Spannung
      int volt_temp;       // Variable fuer den Messwert
      float wert_lsb;      // Variable fuer Quantisierungsstufen
      const char *i2cDevice = "/dev/i2c-1";
      // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"

      wert_lsb = 3.3/256;  // Berechnung der Quantisierungsstufen
                           // mit Referenzspannung 3,3V

      // i2c-Interface oeffnen

      if((device_pcf8591 = open(i2cDevice, O_RDWR)) < 0 ) {
      printf("Der i2c-Port konnte nicht geoeffnet werden! \n");
            exit(1);

      }

      // i2c-Adresse zuweisen

      if(ioctl(device_pcf8591, I2C_SLAVE, i2c_address) < 0 ){
      printf("Die Konfiguration der i2c-Adresse ist fehlgeschlagen!
            \n");

      exit(1);

      }

      // Messung starten

      i2c_smbus_write_byte(device_pcf8591, 0x40 | channel);

      sleep(0.01);    // 10 ms warten

      // Messwert einlesen
      volt_temp = i2c_smbus_read_byte(device_pcf8591);

      // Spannung berechnen
      voltage = volt_temp * wert_lsb;

      //Interface schliessen
      close(device_pcf8591);

      return voltage;

}
void dac_out(int8_t value){

int device_pcf8591;   //Platzhalter fuer das geoeffnete Geraet
      const int i2c_address = PCF8591_adr;   // Chipadresse
      const char *i2cDevice = "/dev/i2c-1";
      // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"

      // i2c-Interface oeffnen

      if((device_pcf8591 = open(i2cDevice, O_RDWR)) < 0 ) {
      printf("Der i2c-Port konnte nicht geoeffnet werden! \n");
            exit(1);

      }

      // i2c-Adresse zuweisen
```

```
        if(ioctl(device_pcf8591, I2C_SLAVE, i2c_address) < 0 ){
         printf("Die Konfiguration der i2c-Adresse ist fehlgeschlagen!
                \n");
               exit(1);
        }
        // Ausgabe des Wertes "value" auf dem DAC
        i2c_smbus_write_byte_data(device_pcf8591, 0x40 , value);
        //Interface schliessen
        close(device_pcf8591);
}
```

pcf8581_demo.h:

```
#ifndef PCF8591_H
#define PCF8591_H
// Funktionen
float adc_conv(int channel);
// Funktion zum Auslesen der ADC-Kanaele mit der Uebergabe
// des gewuenschten Kanals (1-4)
void dac_out(int8_t value);
// Funktion zur Ausgabe eines 8-Bit-Wertes auf dem DAC
#endif
```

Das Programm *adc_in* liest über den PCF8591 die Spannungswerte von einem der vier Kanäle ein. Der einzulesende Kanal wird vom Programm abgefragt und dann sowohl auf dem LCD als auch in der Konsole ausgegeben.

Abbildung 6.39: Darstellung der vier gemessenen Kanäle auf dem LCD

 # ADC-Testprogramm (adc_test.c)

```
#include <stdio.h>          // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include "i2c_lcd_drv_2.h"  // LCD-Treiber (HD44780-kompatibel)
#include "pcf8591_demo.h"   // PCF8591-Funktionen
int main(void){
      char str[10];         // String-Zwischenspeicher
      char time_str[25];    // String zum Umwandeln
      int8_t channel = 0;
      float adc_value = 0;
```

```
while(1){
    do{
       printf("Welcher Kanal soll eingelesen werden? [0 - 3] - ");
        scanf("%d",&channel);
             if((channel < 0) || (channel > 3))
 printf("Kanalangabe fehlerhaft, bitte Kanal 0 - 3 eingeben \n");
      }
     while((channel < 0) || (channel > 3));
        // Einlesen des ADC-Wertes vom Kanal "channel"
        adc_value = adc_conv(channel+1);
        printf("Kanal Nr.%d: %1.2f V\n\n",channel,adc_value);
     // Ausgabe des ADC-Wertes auf dem LCD
     init_i2c();   // Initialisierung des PCF8574 zur
                   // LCD-Ansteuerung
          lcd_init();   // Initialisierung des LCDs
          sprintf(str,"Kanal Nr.%d: %1.2f V",channel,adc_value);
          lcd_print(str);
          i2c_close();  // Interface schliessen
        }
}
```

Mit dem folgenden Programm für die Funktionsüberprüfung des D/A-Wandlers im PCF8591 können Werte zwischen 0 und 255 in der Konsole eingegeben werden. Die Werte werden anschließend auf den DAC-Chip gegeben, der daraus in Abhängigkeit von der Referenzspannung den analogen Ausgangswert erzeugt. Danach wird das Programm in einer Schleife weitergeführt.

```
pi@raspberrypi ~/RasPi_i2c/sensnode/adc_in $ sudo ./adc_test
Welcher Kanal soll eingelesen werden? [0 - 3] - 0
Kanal Nr.0: 0.00 V

Welcher Kanal soll eingelesen werden? [0 - 3] - 1
Kanal Nr.1: 0.64 V

Welcher Kanal soll eingelesen werden? [0 - 3] - 2
Kanal Nr.2: 0.98 V

Welcher Kanal soll eingelesen werden? [0 - 3] - 3
Kanal Nr.3: 3.25 V

Welcher Kanal soll eingelesen werden? [0 - 3] - 4
Kanalangabe fehlerhaft, bitte Kanal 0 - 3 eingeben
Welcher Kanal soll eingelesen werden? [0 - 3] - ^C
pi@raspberrypi ~/RasPi_i2c/sensnode/adc_in $
pi@raspberrypi ~/RasPi_i2c/sensnode/dac_out $ sudo ./dac_test
Welcher Wert soll ausgegeben werden? [0 - 255] - 255
Wert: 255 = 3.300 V

Welcher Wert soll ausgegeben werden? [0 - 255] - 128
Wert: 128 = 1.656 V

Welcher Wert soll ausgegeben werden? [0 - 255] - 5
Wert: 5 = 0.065 V

Welcher Wert soll ausgegeben werden? [0 - 255] - ^C
pi@raspberrypi ~/RasPi_i2c/sensnode/dac_out $
```

Abbildung 6.40: Kommunikation mit A/D- und D/A-Wandler im Terminal

```
# DAC-Testprogramm (dac_test.c)

#include <stdio.h>          // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include "i2c_lcd_drv_2.h"  // LCD-Treiber (HD44780-kompatibel)
#include "pcf8591_demo.h"   // PCF8591-Funktionen

int main(void){

      char str[10];         // String-Zwischenspeicher
      uint8_t value = 0;
      float voltage = 0;

while(1){

    do{

printf("Welcher Wert soll ausgegeben werden? [0 - 255] - ");
        scanf("%d",&value);
            if((value < 0) || (value > 255))

   printf("Fehlerhafter Wert. Der Wertebereich ist 0-255 \n");

       }while((value < 0) || (value > 255));
        // Berechnung der DAC-Spannung mit Referenzspannung 3,3V
        voltage = (3.3 / 255) * value;
        dac_out(value);         // Ausgabe des Wertes auf dem DAC
        printf("Wert: %d = %1.3f V \n\n",value,voltage);
```

```
      // Ausgabe auf dem LCD
          init_i2c();
          // Initialisierung des PCF8574 zur LCD-Ansteuerung
          lcd_init();   // Initialisierung des LCDs
          sprintf(str,"Wert: %d = %1.3f V",value,voltage);
          lcd_print(str);
          i2c_close();  // Interface schliessen
       }
}
```

6.5.5.3 LCD-Ansteuerung

Das Testprogramm lcd_out.c zur Ausgabe von Zeichen auf dem LCD verwendet die bereits erwähnte LCD-Bibliothek (i2c_lcd_drv_2.h). Sie ist aus dem Jahre 2006 und stammt ursprünglich von Nico und Thomas Eichelmann. Nähere Daten hierzu sind unter `http://computerheld.de/i2clcd/doc/index.html` angegeben.

Entsprechende LCD-Routinen sind, wie bereits für den A/D-Wandler gezeigt, auch in den einzelnen Messprogrammen für die Ausgabe integriert. Das Testprogramm gibt eine beliebige Konsoleneingabe auf dem LCD aus, wobei maximal 20 Zeichen in eine Zeile des verwendeten LCDs passen.

```
# LCD-Testprogramm (lcd_out.c)
#include <stdio.h>          // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include "i2c_lcd_drv_2.h"  // LCD-Treiber (HD44780-kompatibel)
/*Hauptprogramm*/
int main(void)
{      char ausgabe[21];
      /*Endlosschleife*/
      while(1){
               printf("Was soll ausgegeben werden? : ");
               scanf("%s",ausgabe);
          init_i2c();
          // Initialisierung des PCF8574 zur Ansteuerung des LCDs
          lcd_init();   // Initialisierung des LCDs
          lcd_print(ausgabe);
          i2c_close();  // Interface schliessen
                }
        return 0;
}
```

Abbildung 6.41: Wenn ein LCD wie dieses mit dem Standard-Modell HD44780 kompatibel ist, lässt es sich mit der Schaltung einsetzen.

6.5.5.4 Temperatursensor

Der Temperatursensor DS1621 wird ebenfalls über die erwähnten I²C-Bibliotheken angesprochen. Das folgende Programm enthält die Funktion *int read_temp(void)* zum Auslesen der Temperatur des DS1621, die die Temperatur als Integer-Wert zurückgibt.

Die gemessene Temperatur entspricht allerdings nicht der Umgebungstemperatur, was daran liegt, dass die Messplatine von der CPU des Raspberry Pi von unten her aufgeheizt wird und somit näherungsweise die Temperatur der Raspberry Pi-CPU gemessen wird und nicht die eigentliche Umgebungstemperatur, was nach einiger Betriebszeit zu einer Messtemperatur von 35 °C und mehr führt.

```
# Temperatursensor DS1621 an I2C-Bus (ds1621_demo.c)

#include <stdio.h>              // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>      // i2c-Kerneltreiber
#include <fcntl.h>              // Dateizugriffssteuerung

// I2C-Adresse
#define ds1602_adr 0x48;

int8_t read_temp(void){

       const int i2c_address = ds1602_adr;     // Chipadresse
        int device_ds1621;

                        // Platzhalter fuer das geoeffnete Geraet

       int8_t temperatur;                      // Temperaturvariable
```

```
        const char *i2cDevice = "/dev/i2c-1";
        // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"
        // I2C-Interface oeffnen
        if((device_ds1621 = open(i2cDevice, O_RDWR)) < 0 ) {
                printf(" Das I2C-Interface konnte nicht geoeffnet
                            werden! \n");
                exit(1);
        }
        // I2C-Bus-Adresse zuweisen
        if(ioctl(device_ds1621,I2C_SLAVE,i2c_address) < 0 ){
                printf(" Die Zuweisung der I2C-Bus-Adresse ist
                        fehlgeschlagen! \n");
                exit(1);
        }
        i2c_smbus_write_byte_data(device_ds1621, 0xAC, 0x00);
        //DS1621 Konfiguration
        i2c_smbus_write_byte(device_ds1621, 0xEE);

        // Messung starten
        temperatur = i2c_smbus_read_byte_data(device_ds1621, 0xAA);
        // Temperatur lesen
        i2c_smbus_write_byte(device_ds1621, 0x22);
        // Messung beenden
        close(device_ds1621);           // I2C-Interface schliessen
        return temperatur;
}
```

ds1621_demo.h:

```
#ifndef DS1621_DEMO_H
#define DS1621_DEMO_H
int read_temp(void); // Funktion zum Auslesen der Temperatur
#endif
```

Das folgende Listing zeigt unter Verwendung von *ds1621_demo.h* die Temperaturerfassung mit der Ausgabe auf dem Monitor und auf dem LCD. Hierfür wird die Bibliothek *i2c_lcd_drv_2.h* eingebunden, die zahlreiche LCDs (HD44780-kompatibel) unterstützt. Nach Betätigung der Eingabetaste wird die Messung wiederholt und mit Strg+C beendet bzw. abgebrochen.

```
# Temperaturmessung (temp_test.c)
#include <stdio.h>          // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include "i2c_lcd_drv_2.h"  // LCD-Treiber (HD44780-kompatibel)
#include "ds1621_demo.h"    // DS1621-Funktionen
```

```
/*Hauptprogramm*/
int main(void)
{
      char str[10];          // String-Zwischenspeicher
      int8_t temp;           // Variable fuer Temperatur
      /*Endlosschleife*/
      while(1){
       /* Einlesen und Ausgabe der Temperatur vom DS1621*/
         temp = read_temp();          //  Auslesen der Temperatur
         printf("Temperatur auslesen? - [Enter] druecken ...");
         getchar();
         printf("Die Temperatur betraegt: %d °C\n\n", temp);
         sprintf(str, "%d", temp);    //  Umwandlung in String
          init_i2c();
          // Initialisierung des PCF8574 zur Ansteuerung des LCDs
           lcd_init();                 // Initialisierung des LCDs
           lcd_print("Temperatur: ");  // Ausgabe auf LCD
           lcd_print(str);
           lcd_print(" C");
           i2c_close();   // Schliessen der PCF8574-i2c-Verbindung
      }
       return 0;
}
```

```
pi@raspberrypi ~/RasPi_i2c/sensnode/temp_in $ sudo ./temp_test
Temperatur auslesen? - [Enter] drücken ...
Die Temperatur betraegt: 33 °C

Temperatur auslesen? - [Enter] drücken ...
Die Temperatur betraegt: 33 °C
```

Abbildung 6.42: Ausgabe der gemessenen Temperatur im Terminal

6.5.5.5 I/O-Port

Der 8-Bit-I/O-Expander PCF8574 wird für die LCD-Ansteuerung über den Anschluss CONN1 verwendet. Falls kein LCD eingesetzt werden soll, kann er als 8-Bit-I/O-Port für die universelle Ein- und Ausgabe an den CONN2-Kontakten eingesetzt werden, um unterschiedliche digitale Bauelemente wie LEDs, Taster oder Relais ansteuern und abfragen zu können.

Dies ist natürlich auch direkt mit den GPIO-Ports vom Raspberry Pi möglich, allerdings wirken sich Schaltungsfehler dort direkt auf den BCM2835 aus, der dadurch einen irreparablen Schaden nehmen kann, womit dann die komplette

Platine unbrauchbar wäre. Der PCF8574 befindet sich im DIP-Gehäuse in einer IC-Fassung, kostet ca. 2 € und ließe sich im Schadensfall daher einfach ersetzen.

```
# I/O Port Expander (pcf8574_demo.c)

#include <stdio.h>              // Standard-Bibliotheken
#include <stdint.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>      // i2c-Kerneltreiber
#include <fcntl.h>              // Dateizugriffssteuerung

// I2C Adresse
#define PCF8574_adr 0x20;

// Funktion liest den kompletten Port des PCF8574 aus und
// gibt die Zustaende als 8-Bit Integervariable zurueck

uint8_t io_in(){
      uint8_t io_register = 0;    // Variable zum Speichern des
                                  // IO-Zustands

      int device_pcf8574;
      //Platzhalter fuer das geoeffnete Geraet

      const int i2c_address = PCF8574_adr;  // Chipadresse

      const char *i2cDevice = "/dev/i2c-1";
      // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"

      // i2c-Interface oeffnen
      if((device_pcf8574 = open(i2cDevice, O_RDWR)) < 0 ) {

      printf("Der i2c Port konnte nicht geoeffnet werden! \n");

            exit(1);

      }

      // i2c-Adresse zuweisen

      if(ioctl(device_pcf8574, I2C_SLAVE, i2c_address) < 0 ){

       printf("Die Konfiguration der i2c Adresse ist

              fehlgeschlagen! \n");

            exit(1);

      }

      // Wert einlesen
      io_register = i2c_smbus_read_byte(device_pcf8574);

      //Interface schliessen
      close(device_pcf8574);

      // Rueckgabe von Wert
      return io_register;

}

// Funktion setzt den kompletten Port des PCF8574
// Die 8 Zustaende werden als 8-Bit Integerwert uebergeben

void io_out(uint8_t io_register){
```

```
        int device_pcf8574;
        //Platzhalter fuer das geoeffnete Geraet
        const int i2c_address = PCF8574_adr;    // Chipadresse
        const char *i2cDevice = "/dev/i2c-1";
         // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"
        // i2c-Interface oeffnen
        if((device_pcf8574 = open(i2cDevice, O_RDWR)) < 0 ) {
        printf("Der i2c Port konnte nicht geoeffnet werden! \n");
              exit(1);
        }
        // i2c-Adresse zuweisen
        if(ioctl(device_pcf8574, I2C_SLAVE, i2c_address) < 0 ){
        printf("Die Konfiguration der i2c Adresse ist
                fehlgeschlagen! \n");
              exit(1);
        }
        // Ausgabe des Wertes io_register auf die IO Ports
        i2c_smbus_write_byte(device_pcf8574, io_register);
        //Interface schliessen
        close(device_pcf8574);
}
// Funktion zum Setzen oder Loeschen eines bestimmten Bits
// io_bit mit set = 0 oder set = 1
void io_bit_out(uint8_t io_bit,uint8_t set){
        uint8_t io_port = 0;
        int device_pcf8574;
         // Platzhalter fuer das geoeffnete Geraet
        const int i2c_address = PCF8574_adr;    // Chipadresse
        const char *i2cDevice = "/dev/i2c-1";
         // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"
        // i2c-Interface oeffnen
        if((device_pcf8574 = open(i2cDevice, O_RDWR)) < 0 ) {
        printf("Der i2c Port konnte nicht geoeffnet werden! \n");
              exit(1);
        }
        // i2c-Adresse zuweisen
        if(ioctl(device_pcf8574, I2C_SLAVE, i2c_address) < 0 ){
        printf("Die Konfiguration der i2c Adresse ist
                fehlgeschlagen! \n");
              exit(1);
        }
```

```
        // Port einlesen
        io_port = i2c_smbus_read_byte(device_pcf8574);

        // Pin setzen oder loeschen
        if(set == 0){
                io_port &= ~(1 << io_bit);
                }
                else if(set == 1){
                io_port |= (1 << io_bit);
                }

        else{
                printf("Der angegebene Zustand ist ungueltig, nur 0
                        oder 1 moeglich\n");

                close(device_pcf8574);

        }
// Ausgabe des aktualisierten Wertes io_port auf den IO-Port
        i2c_smbus_write_byte(device_pcf8574, io_port);

        //Interface schliessen
        close(device_pcf8574);

}
// Funktion gibt den Zustand eines bestimmten Bits (io_bit)
// zurueck
uint8_t io_bit_in(uint8_t io_bit){

        uint8_t io_register = 0;
         // Variable zum Speichern des IO-Zustandes

        int device_pcf8574;
        // Platzhalter fuer das geoeffnete Geraet

        const int i2c_address = PCF8574_adr;  // Chipadresse

        const char *i2cDevice = "/dev/i2c-1";
         // Bei Raspberry Pi Rev. 1 : "/dev/i2c-0"

        // i2c-Interface oeffnen
        if((device_pcf8574 = open(i2cDevice, O_RDWR)) < 0 ) {

        printf("Der i2c Port konnte nicht geoeffnet werden! \n");

                exit(1);

        }

        // i2c-Adresse zuweisen
        if(ioctl(device_pcf8574, I2C_SLAVE, i2c_address) < 0 ){

        printf("Die Konfiguration der i2c Adresse ist
                 fehlgeschlagen! \n");

                exit(1);

        }

        // Wert einlesen
       io_register = i2c_smbus_read_byte(device_pcf8574);

       if(io_register & (1 << io_bit)){
```

```
            close(device_pcf8574);  // Interface schliessen
            return 1;               // Zustand 1 zurueckgeben
    }
      close(device_pcf8574);        // Interface schliessen
      return 0;                     // Zustand 0 zurueckgeben
}
```

7 Low Power-Betrieb

Mit dem Raspberry Pi steht ein preisgünstiger Mini-Computer zur Verfügung, der sich durch seine leicht zugänglichen Schnittstellen (GPIO, I^2C, SPI) auch für Aufgaben in der Mess-, Steuer- und Regelungstechnik eignet, wie es im vorherigen Kapitel dargestellt ist. Versorgt wird ein Raspberry Pi standardmäßig aus einem 5 V-Netzteil, welches laut Herstellervorgaben mindestens einen Strom von 500 mA liefern sollte, was darauf hindeutet, dass der Raspberry Pi im Vergleich zu typischen Mikrocontrollersystemen weitaus mehr Leistung benötigt.

Für mobile Anwendungen ist ein Steckernetzteil nicht geeignet, sondern es wird eine Batterie oder ein Akkumulator benötigt, für deren Verwendung ein Raspberry Pi nicht ausgestattet ist, so dass hierfür einige Anpassungen und/oder Erweiterungen notwendig sind. Mit verschiedenen Methoden ist es möglich den Stromverbrauch zu reduzieren, so dass sich damit auch Anwendungen umsetzen lassen, die relativ wenig Energie benötigen.

7.1 Optimierung des Stromverbrauchs

Für Elektronikprojekte, bei denen die Höhe des Stromverbrauch eine wichtige Rolle spielt, ist das Modell A, welches über keinen LAN-Chip und nur über einen (statt zwei) USB-Ports sowie über 256 MByte (statt 512 MByte) verfügt, dem Modell B vorzuziehen. Der genaue Stromverbrauch hängt davon ab, welche Applikationen bzw. Aufgaben ausgeführt werden, wobei die genannten Reduzierungen des Modell A allerdings bereits zu einem durchschnittlich 30 % niedrigeren Verbrauch gegenüber dem Modell B führen. Angeschlossene Geräte wie eine Tastatur oder ein USB-Stick oder eine spezielle Elektronik tragen ebenfalls zum Stromverbrauch bei, was bei der konkreten Anwendung mit zu berücksichtigen ist.

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1_8, © Springer Fachmedien Wiesbaden 2015

Abbildung 7.1: Das Modell A ist aufgrund des nicht bestückten LAN/USB-Chips als Plattform für stromsparende Anwendungen besser geeignet als die B-Modelle.

Weitere Faktoren und Parameter, die den Stromverbrauch des »reinen« Raspberry Pi-Systems bestimmen, sind der Typ des verwendeten Betriebssystems, die Höhe der Prozessorspannung und der Takte sowie die Effizienz der eingesetzten Spannungsregler, was demnach genauer zu untersuchen ist, denn die Leistungsangaben für die drei Modelle von 3,5 W (Modell B), 3 W (Modell B+) und 2,5 W (Modell A) sind für konkrete Leistungsberechnungen zu ungenau.

In der folgenden Tabelle sind deshalb Messwerte für den maximalen Strom angegeben, der nach dem Einschalten und während des Bootens (Boot) – wenn auch nur kurzzeitig – bei den verschiedenen Raspberry Pi-Versionen auftritt. Wenn das System (Raspbian) gebootet hat und stabil »am Login steht«, ist ein weiterer Messwert (Login) ermittelt worden. Dieser Zustand wird im Folgenden auch mit der üblicheren Bezeichnung *Idle* ausgewiesen, was allgemein für einen Leerlaufprozess steht. In diesem Fall für denjenigen Zustand, bei dem der Raspberry Pi keine (merkliche) Aktion durchführt und am Login steht. Bei genauerer Betrachtung ist der Raspberry Pi selbstverständlich aktiv, denn es laufen – im Hintergrund – verschiedene Prozesse ab. Dies ist jedoch nicht weiter von Bedeutung, weil hier vergleichende Relativmessungen durchgeführt wurden.

Der dritte Messwert ist im Stromsparmodus ermittelt worden, wenn das System per *sudo halt* abgeschaltet worden ist. Der dann noch fließende Strom wird hier als *Ruhestrom* bezeichnet und der Modus als *Standby*.

Der Aufbau war bei den getesteten Systemen stets gleich, mit angeschlossenem Monitor an HDMI und einer Tastatur aber ohne Netzwerkverbindung und ohne weitere Peripherie. Als SD-Karte wurde stets der gleiche Typ verwendet. Eine Ausnahme betrifft das Compute Module, welches keinen SD-Kartenslot besitzt und bei dem das System im eMMC-Chip (siehe Kapitel 3.10.2) abzulegen ist. Das Compute Modul wurde dabei mit dem dazugehörigen »Trägerboard« (I/O Board) des Development Kits betrieben. Ein Unterschied im Stromverbrauch beim Einsatz verschiedener SD Cards konnte im Übrigen nicht festgestellt werden. Wichtig für den Stromverbrauch ist hingegen, dass keine grafische Oberfläche verwendet wird, denn allein dadurch steigt der Stromverbrauch merklich an, kurzzeitig um bis zu 200 mA.

Tabelle 7.1: Gemessener Stromverbrauch der verschiedenen Raspberry Pi-Versionen

Max. Strom bei	**Modell A**	**Modell A mit MagI³c**	**Modell B**	**Modell B+**	**Compute Module/ Modell A+**
Boot	230 mA	170 mA	450 mA	250 mA	140 mA
Login (Idle)	136 mA	115 mA	340 mA	210 mA	95 mA
Halt Mode (Standby)	36 mA	32 mA	112 mA	57 mA	29 mA

Wie es in der Tabelle 7.1 zu erkennen ist, ist das Modell B erwartungsgemäß dasjenige, das den höchsten Stromverbrauch hat. Das Modell B+ weist demgegenüber eine Einsparung von fast 40 % auf, was vom Modell A mit ca. 60 % und dem Compute Module mit ca. 70 % weniger noch übertroffen wird. Das Modell A+ weist den gleichen Stromverbrauch wie das Compute Module auf. In der Tabelle ist außerdem das *Modell A mit MagI³c* angegeben. Dabei handelt es sich um ein mit einem Schaltregler modifiziertes A-Modell, was im Kapitel 7.1.5 erläutert ist.

7.1.1 Betriebssystem

Das verwendete Betriebssystem hat einen nicht zu vernachlässigenden Einfluss auf den Stromverbrauch. Für Low Power-Anwendungen empfiehlt es sich generell, nicht in eine grafische Oberfläche zu booten, sondern in der textbasierten (Konsolenmodus) zu arbeiten. In der Tabelle 7.2 sind die wichtigen Eigenschaften für die vier bekanntesten Betriebssysteme für den Raspberry Pi als Überblick angegeben.

Tabelle 7.2: Stromverbrauch und Bootzeit der gebräuchlichsten Systeme beim Modell A.

System	Komfort	Software-Pakete	Idle Strom in mA	Sleep Strom in mA	Bootzeit in s
Raspbian	Hoch	35000+	134,1	35,1	25
Arch Linux	Gering	4600	133,9	35,7	10
Pidora	Mittel	16000	133,3	35,0	9
RISC OS	Hoch	500	209,0	206,0	8

Raspbian ist eine an den Raspberry Pi angepasste Version von Debian-Linux und das am häufigsten verwendete System, welches sich auch für Einsteiger eignet. Arch-Linux hingegen richtet sich eher an die erfahreneren Anwender, die weniger an den zahlreichen vorinstallierten Paketen interessiert sind, sondern das System individuell anpassen möchten (und können). Der Stromverbrauch liegt bei beiden Systemen etwa auf dem gleichen Niveau, beim Booten im schwankenden Bereich von 160 mA bis 230 mA und im Idle-Mode bei 136 mA.

Die Bootzeit beträgt bei Arch-Linux jedoch lediglich ca. 10 Sekunden, während Raspbian dreimal so lange dafür benötigt. Im »ausgeschalteten« Zustand (sudo halt) verbraucht Raspbian 34 mA. Bei Arch-Linux führt *sudo halt* in einen nichtaktiven Modus, der einen Stromverbrauch von über 170 mA zur Folge hat. Mit *sudo shutdown -h now* fährt jedoch auch Arch-Linux in den niedrigen Standby-Modus, mit einem nur leicht höheren Stromverbrauch (36 mA) als Raspbian. Üblicherweise macht es keinen Unterschied, ob mit *halt* oder *shutdown* gearbeitet wird.

Bei Pidora, der angepassten Version von Fedora, beträgt der Ruhestrom ca. 35 mA, im Idle-Modus werden 133 mA verbraucht und der Bootvorgang dauert lediglich 9 s, so dass Pidora durchaus als schnellere Alternative zu Raspbian angesehen werden kann.

RISC OS wurde wie der Raspberry Pi ursprünglich in Cambridge entwickelt und zwar für die Acorn-Computer (RISC) in den achtziger Jahren. Im Vergleich mit den aktuellen Linux-Systemen wirkt RISC OS etwas schlicht und nostalgisch. Weil hier viele der elementaren Funktionen per Drag & Drop auszuführen sind, wird dementsprechend die grafische Oberfläche und eine angeschlossene Maus benötigt. Somit eignet sich dieses System, auch wegen eines Stromverbrauchs von über 200 mA im Standby-Modus, nicht für Low Power-Systeme.

7.1.2 Embedded System

Ein Embedded System muss nicht zwangsläufig mit einem Betriebssystem arbeiten, so dass auch die Möglichkeit besteht, den Raspberry Pi direkt mit einem Pro-

gramm (Assembler, C) anzusprechen. Im Internet sind hierzu einige Beispiele zu finden, beispielsweise von David Welch (Raspberry Pi Bare Metal Examples) unter `https://github.com/dwelch67/raspberrypi`. Die Stromersparnis im Vergleich zum Betrieb mit Pidora fällt nach eigenen Messungen allerdings nicht so stark aus, wie zunächst vermutet. Im Idle-Modus beträgt der Unterschied ca. 10 mA. Beim Ablauf eines exklusiv laufenden Programms im Konsolenmodus fällt der Stromverbrauch, der durch die Systemroutinen verursacht wird, nicht merklich ins Gewicht.

Für Embedded-Systems ist oftmals keine Video-Ausgabe notwendig, so dass ein Abschalten des PAL- und des HDMI-Ausgangs sinnvoll sein kann, um Strom zu sparen. Typischerweise führt dies zu einem um ca. 20 mA geringeren Verbrauch, was durch die folgende Zeile ausgelöst wird:

```
root@raspberrypi:/opt/vc/bin/#  tvservice -off
```

Nach dem nächsten Boot ist die Ausgabe automatisch wieder aktiviert.

Auch wenn die A-Modelle laut der *Raspberry Pi Foundation* explizit für batterie- und solarbetriebene Anwendungen gedacht sein sollen, ist der minimale Stromverbrauch weit höher als der von Mikrocontrollerplattformen, die lange Zeit in den verschiedenen Sleep-Modi verweilen können und nur dann aufwachen müssen, wenn die Anwendung dies erfordert. Vergleichbare Low Power Modes, wie sie etwa bei der MSP430-Familie von Texas Instruments oder auch bei ARM-Prozessoren wie den EFM32-Typen von SiliconLabs zur Verfügung stehen, lassen sich beim Raspberry Pi nicht programmieren, was zunächst an fehlenden Informationen zu den implementierten Power Modes des eingesetzten ARM-Prozessors BCM2835 liegt.

Die Firma Broadcom gibt hierzu offiziell keinerlei verwertbare Informationen preis. Die Originaldokumentation zum ARM1176JZF-S führt auch nicht wirklich weiter, weil sowohl die an den Raspberry Pi angepasste Firmware als auch der Kernel beim Registerzugang »dazwischensitzen«. Beide Software-Komponenten unterstützen mit den typischen Betriebssystemen für den Raspberry Pi keine speziellen Low Power-Modi, außer eben den Halt-Modus (sudo halt), der bei Linux zum Standard gehört. Deshalb kann nur eine direkte Registerprogrammierung von der Anwenderebene aus auf die ARM11-Register noch Aussicht auf Erfolg haben, wobei man schnell Gefahr läuft, das Betriebssystem dabei »abzuschießen«.

Ohne besonderen Aufwand ist es deshalb nur möglich den Raspberry Pi in den Halt-Modus (Standby) zu versetzen, so dass dann minimal der Ruhestrom fließt, wie er in der Tabelle 7.1 angegeben ist. Durch das Entfernen der roten Power-LED, die im Halt-Modus leuchtet, lassen sich allerdings noch weitere 2 mA einsparen. Um den Raspberry Pi wieder aus dem Standby zu aktivieren, kann dies ebenfalls

programmgesteuert erfolgen oder aber durch eine fallende Signalflanke (ca. 2 s an GND schalten) am GPIO3 (Pin 5), was somit einer Hardwareware-Steuerung entspricht.

7.1.3 Einstellung der Taktfrequenzen

Grundsätzlich hat die Höhe der Taktfrequenz (f) nach der folgenden Formel einen maßgeblichen Einfluss auf den Stromverbrauch, denn sie ist proportional zur dynamischen Leitungsaufnahme:

$$P_{dyn} = f * C_L * V_{CC}^2$$

Anhand der Formel ist auch zu erkennen, dass die Leistungsaufnahme quadratisch von der Versorgungsspannung (V_{cc}) abhängt, was deshalb noch separat zu betrachten ist. Der Kondensator C_L repräsentiert dabei die Kapazität, die sich durch die Transistoren im Chip ergibt.

Beim Raspberry Pi gibt es drei entscheidende Taktfrequenzen: Die des ARM-Prozessorkerns (CPU), die das Grafikprozessors (GPU) und die des Arbeitsspeichers (RAM), die sich aus einem Quarzoszillator von 19,2 MHz ableiten, der sich auf der Platinenunterseite befindet.

Die Veränderung der Frequenzen ist sehr einfach mit einem Editor in der Datei */boot/config.txt* (am Ende der Datei, gekennzeichnet durch *_freq*) durchführbar. Nach jeder Änderung ist die Konfigurationsdatei zu speichern und ein Neuboot auszuführen. Für den Geschwindigkeitstest des Systems wurden die zum Booten benötigte Zeit und die Laufzeit bei verschiedenen Taktfrequenzen und Spannungen eines einfachen C-Programms gemessen, welches die Berechnung von Primzahlen ausführt.

Lister - [G:\config.txt]

Datei Bearbeiten Optionen Hilfe

```
# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1

# uncomment this if your display has a black border of unused pixels visible
# and your display can output without overscan
disable_overscan=1

# uncomment the following to adjust overscan. Use positive numbers if console
# goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16

# uncomment to force a console size. By default it will be display's size minus
# overscan.
#framebuffer_width=1280
#framebuffer_height=720

# uncomment if hdmi display is not detected and composite is being output
#hdmi_force_hotplug=1

# uncomment to force a specific HDMI mode (this will force VGA)
#hdmi_group=1
#hdmi_mode=1

# uncomment to force a HDMI mode rather than DVI. This can make audio work in
# DMT (computer monitor) modes
#hdmi_drive=2

# uncomment to increase signal to HDMI, if you have interference, blanking, or
# no display
#config_hdmi_boost=4

# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
arm_freq=700

# for more options see http://elinux.org/RPi_config.txt
gpu_mem=128
start_x=0
core_freq=250
sdram_freq=400
over_voltage=0
```

Abbildung 7.2: In der Datei config.txt lassen sich manuelle Anpassungen für die Takt- und Spannungseinstellung vornehmen.

Der ARM-Prozessor arbeitet standardmäßig mit 700 MHz. Durch Über- und Untertakten sind Frequenzen im Bereich von 40 MHz bis 1000 MHz möglich. Eine Übertaktung auf über 1000 MHz ist theoretisch zwar möglich, jedoch nicht empfehlenswert, weil dadurch die Garantie verfällt, die Lebensdauer des Chips reduziert wird und das System im Regelfall auch nicht mehr stabil läuft. Die Minimalfrequenz von 40 MHz ist für die Systemstabilität ebenfalls als kritisch zu betrachten, weshalb die CPU-Frequenz etwa zwischen 200 und 900 MHz eingestellt werden sollte.

Anhand der Messwerte in der Abbildung 7.3 ist zu erkennen, dass eine Verringerung der CPU-Frequenz auf 400 MHz sinnvoll sein kann, wenn eine Einsparung von 2,3 mA gewünscht ist. Bei geringeren Frequenzen steigt die Bootzeit stark an.

Abbildung 7.3: Der Stromverbrauch der CPU im Idle-Mode steigt mit der Frequenz an, während die Bootzeit dementsprechend geringer wird.

Die GPU-Frequenz beträgt laut Voreinstellung 250 MHz und kann bis auf maximal 500 MHz. festgelegt werden. Die GPU-Frequenz *gpu_freq* (GPU Core Frequency) setzt sich aus vier verschiedenen »GPU-Teilfrequenzen« zusammen. Diese können auch einzeln konfiguriert werden, wozu die Zeile in der Konfigurationsdatei mit der Angabe *gpu_freq* auszukommentieren (#-Präfix) und durch die Angaben *core_freq*, *isp_freq*, *h264_freq* und *v3d_freq* zu ersetzen ist.

Durch eine Verringerung dieser Frequenzen ergibt sich eine Reduzierung des Stromverbrauchs. Weil die Performance der GPU für Embedded-Applikationen üblicherweise nicht von Bedeutung ist, ist hierfür lediglich die Minimalfrequenz zu ermitteln, mit der das System noch stabil funktioniert. Ein minimaler Wert von 160 MHz ergibt sich für die GPU-Frequenz als praktikable Lösung, denn eine weitere Reduzierung führt automatisch dazu, dass wieder die Default-Werte aktiviert werden. Bei einer Einstellung von 160 MHz sowie jeweils 40 MHz für die vier »GPU-Teilfrequenzen« ergibt sich eine Stromersparnis von 15,5 mA, was gegenüber einer Reduzierung der CPU- und auch der RAM-Frequenz (sdram_freq) zu einer recht beachtlichen Verringerung des Stromverbrauchs führt.

Die Einstellung *sdram_freq* beträgt üblicherweise 400 MHz. Es ist zu beobachten, dass eine Verringerung unter 300 MHz nicht zu einer Reduzierung, sondern zu einem erhöhten Stromverbrauch führt, und die Stabilität des Systems nicht mehr gewährleistet ist. Bei einer Einstellung von *sdram_freq = 300* (MHz) scheint das Optimum zu liegen, was ohne merkliche Leitungseinbußen eine Stromreduzierung von 2,5 mA zur Folge hat.

7.1.4 Spannungseinstellung

Die Spannungsversorgung für das Raspberry Pi-Board wird von einem separaten Steckernetzteil mit Micro-USB-Anschlusskabel zur Verfügung gestellt. Aus den hiermit generierten 5 V werden auf der Platine drei verschiedene Spannungen für die einzelnen Bauelemente mithilfe von entsprechenden Spannungsreglern (3,3 V und 2,5 V und 1,8 V) erzeugt, wie es im Kapitel 3.7 erläutert ist und im Kapitel 7.2 noch etwas näher behandelt wird.

Die Spannung, die dem Prozessor und der GPU intern zur Verfügung steht, wird chipintern erzeugt und beträgt zunächst 1,2 V. In der Konfigurationsdatei lassen sich mit der Option *over_voltage* Spannungen von 0,8 bis 1,4 V einstellen. Der jeweilige Wert ist eine Ganzzahl zwischen minimal -16 und maximal 8. Die Schrittweite beträgt damit 0,025 V, so dass eine recht granulare Festlegung möglich ist.

Abbildung 7.4: Der Stromverbrauch im Idle-Modus in Abhängigkeit von der Prozessorspannung.

In der Abbildung 7.4 ist der Stromverbrauch bei verschiedenen »Unterspannungen« zu erkennen. Es ist ersichtlich, dass sich der Stromverbrauch des Systems ohne negative Beeinflussung der Bootzeit auf 125 mA reduzieren lässt, was im Vergleich zu den gezeigten Taktveränderungen einer verhältnismäßig hohen Einsparung (ca. 9 mA) entspricht, die sich zudem einfach festlegen lässt. Allerdings fällt sie nicht so hoch aus, wie es die erwähnte Proportionalität zwischen Leistungsaufnahme und (siehe obige Formel) Versorgungsspannung erwarten lässt.

Durch die Kombination der erläuterten Takt- und Spannungsanpassungen bei einem Raspberry Pi Modell A und dem Betriebssystem Pidora ergibt sich eine Reduzierung des Stromes im Idle-Modus auf 109 mA. Es ist allerdings notwendig, eine gewisse Sicherheitsreserve mit einzukalkulieren, damit das System weiterhin stabil und zuverlässig ohne störende Bootverzögerungen funktioniert, was dann zu den folgenden »sicheren Werten« führt mit denen sich ohne programm- oder

hardwaretechnische Anpassungen eine Ersparnis beim Stromverbrauch von ca. 18 % ergibt.

- CPU-Frequenz: 450 MHz
- Core-Frequenz: 180 MHz
- ISP-Frequenz = H.264-Frequenz = V3D-Frequenz 45 MHz
- Interne Versorgungsspannung (Core) 1,075 V

7.1.5 Austausch des Linearreglers

Für die Spannungsregler auf den Raspberry Pi-Boards der Modelle A und B werden gewöhnliche Linearregler für die Spannungsherabsetzung verwendet, die im Gegensatz zu Schaltreglern einen geringen Wirkungsgrad aufweisen und deshalb für Low Power-Applikationen ungeeignet sind. Der Aufbau von Schaltreglerschaltungen ist demgegenüber jedoch aufwendiger, was insbesondere an der notwenigen Induktivität liegt und zudem ist hierfür ein »saubreres Layout« notwendig, um die Funktion des Reglers ohne Störungen, die sich auf andere Komponenten negativ auswirken, zu ermöglichen.

Bei den +Modellen und dem I/O-Board (Kapitel 3.10) für das Compute Module wird der Schaltregler PAM2306 für die Spannungsumsetzung von 5 V auf 3,3 V und auf 1,8 V mit zwei Induktivitäten und Widerstandsbeschaltung (vgl. Abbildung 3.33) verwendet. Die maximale Eingangsspannung darf dabei 5,5 V betragen.

Durch Messungen an den Modellen A und B lässt sich der Regler 2 (NCP1117), der die Eingangsspannung von 5 V auf 3,3 V regelt, als der »größte Energieverschwender« identifizieren, was sich auch durch seine relativ hohe Wärmeabstrahlung feststellen lässt. Hierfür einen möglichst direkten Ersatz zu finden, der sich außerdem auf der Platine ohne große mechanische Probleme und zusätzliche Komponenten unterbringen lässt, ist mit dem *Fixed Step Down Regulator Modul* (MagI^{3}c) der Firma Würth Elektronik gegeben. Das Modul befindet sich inklusive der Induktivität und einigen Kondensatoren in einem fünfbeinigen TO263-Gehäuse, so dass für den Einsatz des Schaltreglers keine weiteren Bauelemente notwendig sind.

Das MagI^{3}c-Modul arbeitet mit einer Eingangsspannung von 5 bis 18 V und einer festen Ausgangsspannung von 3,3 V bei einem maximalen Ausgangsstrom von 3 A, was für den Raspberry Pi mehr als ausreichend ist. Die Effizienz des Moduls beträgt bestenfalls 95 %. Obwohl das Datenblatt nur die Effizienz bei 1 A Ausgangsstrom bzw. 6 V Eingangsspannung ausweist, lässt sich durch Extrapolation für die Beschaltung auf dem Raspberry Pi-Board eine Effizienz prognostizieren, die sehr nah an die maximalen 95 % heranreicht.

Abbildung 7.5: Das MagI³c-Modul kann den Linearregler ohne weitere Beschaltung komplett ersetzen.

Von den sechs Anschlüssen, wobei sich PGND (Power Ground) am oberen verzinnten Gehäuserand befindet, sind lediglich VIN (Pin 1), VOUT (Pin 3) sowie GND (Pin 2 mit Brücke an PGND) an die freigelegten Kontakte auf der Platine anzulöten. Die Pins 4 (PG) und 5 (EN) sind optional und werden hier nicht benötigt. PG ist ein Power Good-Ausgang, der die korrekte Reglerfunktion nach außen signalisiert. EN dient als Enable-Pin für das Ein- und Ausschalten des Moduls durch eine externe Schaltung, wie etwa durch einen Mikrocontroller.

Nach dem Auflöten des MagI³c-Moduls sind keine weiteren Anpassungen notwendig. Es ergibt sich allein durch diese Maßnahme (ohne Takt- und Spannungsanpassungen) im Idle-Modus ein Stromverbrauch von nunmehr 115 mA, was einer Einsparung von ca. 15 % entspricht. Im Standby-Modus ergibt sich eine Reduzierung um 10 %, was aufgrund des in dieser Betriebsart ohnehin stark reduzierten Stromes nicht verwundert.

Abbildung 7.6: Links die Originalplatine, rechts die mit dem MagI³c-Modul, welches aufgrund der Pinbelegung „überkopf“ eingelötet ist.

Die Platine läuft nach der Modifizierung weiterhin völlig störungsfrei. Die Qualität der Ausgangsspannung wurde mit einem Oszilloskop sowohl nach kurzer Betriebsdauer im Idle-Modus als auch nach mehreren Stunden bei der Ausführung eines Belastungsprogramms gemessen. Ein Unterschied in der Qualität der Ausgangsspannung zwischen diesen beiden Belastungsszenarien konnte nicht festgestellt werden, was sowohl für die Originalplatine als aber auch für die modifizierte Platine gilt. Ein signifikanter Unterschied im Ripple zwischen der Originalplatine und der modifizierten Platine ließ sich ebenfalls nicht feststellen. Der MagI³c-Schaltregler zeigte sogar ein geringfügig besseres Verhalten (vgl. Effektivwerte in der Abbildung 7.7).

Abbildung 7.7: Ripple-Messungen im Vergleich

Das thermische Verhalten beider Platinen wurde mithilfe einer Wärmebildkamera analysiert. Dabei wird ersichtlich, dass der Linearregler (Punkt: RO, links in der Abbildung 7.8) eine um ca. 2,85 °C höhere Temperatur als der Schaltregler (Punkt: RM, rechts) aufweist und die Umgebung bei der modifizierten Platine ebenfalls

deutlich weniger aufgeheizt wird, was sogar noch der relativ weit entfernten CPU (Messpunkte CO, CM) zugute kommt.

Mit dem Original-Linearregler hat sich dies bei Erweiterungsplatinen, die für die Kontaktierung mit der GPIO-Kontaktleiste direkt auf die Raspberry Pi-Platine gesteckt werden, durchaus negativ bemerkbar gemacht, weil die auf der Platine befindliche Elektronik von unten aufgeheizt wird, was zum Wärmestau und instabilen Signalen führen kann, wie es im Abschnitt 6.5.5.4 über den Temperatursensor für die I²C-Erweiterungsplatine erwähnt wurde.

Abbildung 7.8: Links das Original-Raspberry Pi-Board und rechts das mit dem Schaltregler modifizierte Board. Deutlich ist zu erkennen, dass der Schaltregler weniger Abwärme produziert.

Im Gegensatz zum Originalregler, der direkt auf die Platine gelötet ist, hängt der Schaltregler bei der modifizierten Version gewissermaßen in der Luft, so dass hier kein direkter Wärmeübergang zur Platine gegeben ist, was korrekterweise bemerkt werden muss. Auffallend für eine relativ hohe Wärmeabgabe ist neben der CPU bei der modifizierten Platine (rechts oben) der Regler 1 (NCP1117-1v8), der die Spannung von 3,3 V auf 1,8 V regelt.

Der Ersatz dieses Typs durch einen Schaltregler wäre deshalb noch eine weitere Option, wie es durch den Schaltregler PAM2306 auf den +Modellen und dem I/O-Board des Compute Modules realisiert wird. Die hierfür notwendigen Induktivitäten und Kondensatoren sind auf der Platine als einzelne Bausteine vorhanden und befinden sich nicht wie beim MagI³c-Schaltregler im Modul. In Anbetracht der Tatsache, dass auch aus diesem Grunde eine neue Platine für die +Modelle entwor-

fen werden musste, ist das Ergebnis der einfachen Umrüstung mit einem MagI³c-Schaltregler durchaus bemerkenswert, wie es im Vergleich mit den anderen Raspberry Pi-Versionen in der Tabelle 7.1 zu erkennen ist.

7.2 Mobile Spannungsversorgungen

Raspberry Pi-Boards können mit einer Eingangsspannung versorgt werden, die laut der *Raspberry Pi Foundation* im Bereich von 4,75 V bis 5,25 V liegen muss.

7.2.1 Spannungsversorgungoptionen

Die Spannungsversorgung muss nicht zwingend über den Micro-USB-Anschluss aus einem Steckernetzteil erfolgen, sondern kann auch an die GPIO-Pins 2 oder 4 (5 V, GND z.B. an Pin 6) angeschlossen werden, was eine einfachere Verschaltung mit einer externen Versorgung ermöglicht.

Die 5 V aus dem Steckernetzteil oder über die GPIO-Pins, gelangen zum einen auf den 3,3 V-Regler und zum anderen direkt an die USB-Anschlüsse, den HDMI-Connector und an den Battery Sense-Pin des BCM2835.

Mit den 3,3 V-Anschlüssen an der GPIO-Pfostenleiste (Pin 1, Pin 17) ist ebenfalls eine Versorgung möglich, wenn die jeweilige Raspberry Pi-Applikation mit dieser geringeren Spannung betrieben werden kann. Dies bedeutet, dass dann keine USB-Einheiten laut Standard eingesetzt werden können, denn diese sind für 5 V spezifiziert, wobei einige den Betrieb mit 3,3 V durchaus erlauben. Bei Tastaturen und Mäusen stellt die geringere Spannung meist kein Problem dar, während einige der USB-WLAN-Adapter damit nicht funktionieren.

Grundsätzlich ist bei dieser Art der Spannungsversorgung über die Kontakte am GPIO-Pfostenstecker kein Schutz über die Sicherung auf dem Board gegeben. Bei den Modellen A und B ist es außerdem möglich die Versorgung extern über den USB herzustellen (USB Backpowering).

In der Praxis zeigt sich, dass alle Raspberry Pi Boards auch noch mit 4,5 V am Micro-USB-Spannungsanschluss und einem grafischen Desktop (LXDE) arbeiten können, wobei meist die Maus als erstes mit Verzögerungen aufgrund einer zu geringen Spannung reagiert. Beim Verzicht auf einen grafischen Desktop und einer Maus gelingt es bis auf 4 V herunterzugehen, d.h. die Tastatur reagiert bis dahin auch noch normal, wobei sie bei ca. 3,8 V nicht mehr funktioniert. Die Spannung am USB ist stets um ca. 200 mV geringer als die über den Micro-USB-Spannungsanschluss eingespeiste.

Bei einer Einspeisung über die 5 V-GPIO-Kontakte (Pin 2 oder 4 und Masse z.B. an Pin 25) muss die Minimalspannung ca. 0,2 V höher liegen als bei einer Versorgung

über den Micro-USB-Spannungsanschluss, d.h. mit minimal 4,5 V sollte das Raspberry Pi Board korrekt mit Maus und Desktop funktionieren.

7.2.1.1 Betrieb mit 3,3 V

Wie erwähnt ist es möglich ein Raspberry Pi Board mit lediglich 3,3 V über den 3,3 V-Kontakt an der GPIO-Pfostenleiste (Pin 1, Pin 17) zu versorgen, wobei dann jedoch keine USB-Geräte – auch keine Tastatur – eingesetzt werden können. Dies liegt daran, dass die USB-Spannung an der Buchse dann nur noch ca. 2,5 V beträgt.

Durch eine Veränderung am Raspberry Pi Board (Modell A und B) lässt sich dies dennoch beheben, in dem der Regler REG2 (NCP1117), der die 5 V-Eingangsspannung auf 3,3 V umsetzt entfernt und eine Brücke vom Eingang auf den Ausgang gelötet wird. Als Einspeisung sind dann natürlich 3,3 V notwendig. Demnach existiert dann keine 5 V-Versorgung mehr und USB, HMDI sowie der Battery Sense-Pin des BCM2835 arbeiten mit 3,3 V, was entsprechend positive Auswirkungen (siehe obige Formel) auf die maximale Laufzeit bei einer Versorgung mit einer Batterie oder einem Akku hat.

Abbildung 7.9: Der RG2-Spannungsregler ist entfernt und die Kontakte sind gebrückt, so dass der Raspberry Pi nur noch mit 3,3 V arbeitet.

Die Spannung von 3,3 V wird mithilfe eines Spannungsreglers erzeugt, der einem *Low Dropout Regler* (LDO) entsprechen sollte, wie beispielsweise einem MCP1825S (3,3 V) der Firma Microchip, für den eine Dropout Voltage von lediglich 210 mV angegeben wird. Er ist für einen maximalen Ausgangsstrom von 500 mA ausgelegt, so dass ein Modell A hiermit sicher und möglicherweise auch ein Modell B (ohne weitere Peripherie) versorgt werden kann. Falls ein höherer Strom (3 A) benötigt wird, ist ein Schaltregler, wie beispielsweise der Typ TPS5430 von Texas Instruments geeignet, wobei der Aufbau mit der Induktivität und den Kondensatoren nicht unkritisch ist, so dass auch hier MagI³c-Modul vorteilhaft eingesetzt werden könnte. Beim Betrieb eines Raspberry Pi mit 3,3 V und einem der angege-

benen oder auch vergleichbaren Regler ist somit der direkte Anschluss an einen Lithium-Ionen-Akku (3,7 V) möglich.

7.2.2 Batterien und Akkus

Typische AA-Batterien (Mignon 1,5 V Alkaline) liefern im günstigsten Fall eine Energie (Ladung) von 1100 mAh und die Lithium-Pendants bis zu 1300 mAh. Die Micro-Ausführungen (AAA) liegen in ihren Kapazitätswerten ca. 10-20 % unter denen der Mignon-Typen. Der Einsatz von 9 V-Blocks, die über Kapazitäten von bis zu 1200 mAh (Lithium-Typ) verfügen, ist aufgrund der zu hohen Spannung keine effektive Lösung.

Grundsätzlich wird für eine Batterie- und auch für eine Akkulösung ein weiterer Spannungsregler benötigt, der die externe Spannung in die notwendigen 5 V umsetzt. Hierfür empfiehlt sich ein Schaltregler wie etwa ein MagI³c-Modul in der 5 V-Ausführung, welches prinzipiell genauso eingesetzt werden kann, wie es im Kapitel 7.1.5 für die 3,3 V-Version erläutert wurde.

NiMH-Akkus in der AAA-Bauform liefern bis zu 2700 mAh, die Spannung beträgt allerdings nur 1,2 V pro Zelle, so dass vier Zellen eigentlich nicht ausreichen. Lithium-Ionen- und Lithium-Polymer-Zellen sind in einer Vielzahl von Ausführungen erhältlich, etwa für Digitalkameras oder Smartphones (1300 mAh für Samsung Galaxy) mit einer typischen Spannung von 3,7 V. Höhere Spannungen, etwa für Modellbau, Notebooks oder auch Elektrowerkzeuge werden durch eine Reihenschaltung einzelner Lithium-Zellen realisiert, was zu mechanisch mehr oder weniger großen Akku-Packs führt.

Abbildung 7.10: Lithium-Ionen- und Lithium-Polymer-Akkus gibt es in verschiedenen Bauformen und mit unterschiedlichen Kapazitäten.

7.2.3 Wirkungsgrad und Laufzeit

Der Wirkungsgrad (η) ist allgemein das Maß für die Energieumsetzung (E). Er gibt das Verhältnis zwischen der genutzten (P_{nutz}) und der zugeführten Leistung (P_{zu}) an.

$$\eta = \frac{E_{nutz}}{E_{zu}} = \frac{P_{nutz}}{P_{zu}}$$

Dementsprechend sollte die Eingangsspannung (U_{bat}) von einem Akku oder einer Batterie bei der Verwendung eines Linearreglers, der auch als Längsregler bezeichnet wird, möglichst nah an der für den Raspberry Pi (U_{pi}) notwendigen Versorgungsspannung liegen.

Der zulässige minimale Unterschied zwischen Eingangs- und Ausgangspannung wird durch den verwendeten Regler bestimmt, der mit einer Dropout-Spannung spezifiziert ist. Bei gewöhnlichen Linearreglern (LM78xx) beträgt diese Spannung mehrere Volt, während Low Dropout-Regler nur noch ca. 200 mV benötigen.

$$\eta = \frac{U_{pi}}{U_{bat}}$$

Demnach wäre der Wirkungsgrad bei der Versorgung eines Raspberry Pi, der mindestens 5 V benötigt, mit einem 9 V-Block nur 55 %, bei Versorgung mit 6 V (4 * 1,5 V Batterie) hingegen 83 %.

Die Laufzeit (t) eines Raspberry Pi (oder natürlich auch einer anderen Schaltung) beträgt allgemein:

$$t = \frac{Q}{I}; Laufzeit = \frac{Ladung}{Strom}$$

Die Ladung entspricht der Kapazitätsangabe des Akkus oder der Batterie und der Strom ist der (durchschnittliche) Verbrauchsstrom der Schaltung. Mit Berücksichtigung des Wirkungsgrads führt dies zu:

$$t = \frac{Q * \eta}{100 * I}$$

Mit einer beispielhaften 6 V-Versorgung (s.o.), die über eine Kapazität (Ladung) von 1000 mAh verfügt, ergibt bei einem Raspberry Pi, der einen durchschnittlichen Strom von 200 mA verbraucht:

$$t = \frac{1000mAh * 83\%}{100 * 200mA} = 4{,}15h$$

Falls der Raspberry Pi stattdessen im Halt-Modus (Standby, siehe Tabelle 7.1) mit lediglich 36 mA arbeitet, reicht die Versorgung für 27 Stunden aus. Beim Einsatz eines Schaltreglers ist der Wert für den Wirkungsgrad aus dem jeweiligen Datenblatt bei den vorgesehenen Betriebsparametern zu entnehmen.

7.2.4 Power Packs

Für die Aufladung der Akkus sind entsprechende Ladegeräte notwendig, was demnach wie beim Einsatz von Batterien, einen Wechsel der Zellen bei deren Erschöpfung erfordert. Deshalb ist neben dem Spannungsregler für 5 V noch eine Ladeschaltung für die Akkus in der Schaltung selbst wünschenswert, die sich möglicherweise auch per Solarzelle betreiben lässt, was allerdings noch einen Boost-Converter voraussetzt, der die Spannung von Solarzellen auf einen nutzbaren Pegel anhebt.

Diese Komponenten gibt es bereits zusammengefasst in fertiger Form als so genannte *Power Banks* oder *Power Packs* oder *USB Battery Packs*, die insbesondere für das Aufladen von Smartphones unterwegs gedacht sind. Im täglichen Umgang zeigt sich, dass die in einer Power Bank integrierten Akkus über die vorgesehene Solarzelle selten voll geladen werden können, so dass das Laden meist über den USB oder das mitgelieferte Netzteil durchgeführt wird. Als Ausgangsspannung sind 5 V üblich und als Ausgang ein Micro-USB-Anschluss, so dass sich hiermit direkt eine Verbindung für die Spannungsversorgung des Raspberry Pi herstellen lässt.

Abbildung 7.11: Mit Power Banks lässt sich ein Raspberry Pi auch mobil betreiben.

Die Power Banks sind mit unterschiedlichen Kapazitäten erhältlich, wobei nicht immer deutlich wird, worauf sich die Kapazitätsangabe bezieht: auf die tatsächlich zu nutzende Energie am Ausgang (5 V) der Power Bank oder aber nur auf den integrierten Akku, der typischerweise mit 3,7 V arbeitet (Lithium). Außerdem kann man davon ausgehen, dass sich die angegebenen Kapazitätsangaben in der Praxis deutlich geringer darstellen.

Die üblichen Power Banks sind für Kapazitäten von 1500 mAh bis hin zu 12600 mAh ausgelegt, was bei einem überschlagsmäßigen Stromverbrauch eines Raspberry Pi von 150 mA zu einer Laufzeit von 10 bzw. 84 Stunden führen müsste. Dies ist jedoch nicht der Fall, weil die Ausgangsspannung einer eher schwachen Power Bank beim Einschalten des Raspberry Pi so stark einbrechen kann, dass kein Betrieb mehr möglich ist. Deshalb ist die Angabe wichtig, welchen Strom die Power Bank verlässlich am Ausgang liefern kann, was man zumindest bei den teureren Modellen findet. Die Problematik, dass die Ausgangsspannung aufgrund eines hohen Einschaltstromes zusammenbricht, gibt es bei einem an eine Power Bank angeschlossen Smartphone in dieser Form nicht, weil der im Smartphone eingebaute Akku diesen Vorgang gewissermaßen »abfedert«.

Stichwortverzeichnis

486-Prozessors 8
8 Bit I/O-Expander 241, 256

A/D-Wandler 220, 239
A/D-Wandlung 224
AAA-Bauform 276
Access Point 164
Acorn 2
Acorn RISC Machine 65
Adhoc-Modus 163
Adressklassen 149
Advanced Options 33
Advanced Packaging Tool, apt 56
Advanced Risc Machines, ARM 65, 68
Allwinner 9, 11
ALSA soundcard driver-package 129
Android 10, 134, 180
ANSI C 171, 191
apt-get-Manager 56
Arcade Games 181
Arch Linux ARM 124
Arduino 5
Arduino-Entwicklungsumgebung 6
Arduino-Konzept 6
ARM1176JZ-F-Prozessorkern 176
ARM1176JZF-S 14
ARM-Architektur 64
ARM-Core ARM1176JZF-S 69
ARM-Familie 67
ARM-Prozessoren 3, 64, 123
ARM-Rechenkerne 65
ARM-Speicherbereich 201
Assembler 176
Atom-Prozessor 8
Audio- und Videoplayer 132
Audioausgabe 129
Audioeinstellungen 130
Audiokabel 128
Audiooptionen 130
Audiosignal 127
Audioverbindung 84
Ausführungsrechte 49
Auto Negotiation 87

Ball Grid Array, BGA 64
Banana Pi 10
Bash-Programmierung 177
Bash-Scripts 177
BBC Micro 3, 65
BCM2835 14, 63, 71, 84, 197, 201
Beaglebone Black 9, 102
Beispielprogramme in C 191
Beschleunigungssensor 198
Betriebsbereitschaftsleitungen 207
Betriebssysteme 123
Bibliotheken 182
BIOS 58
BIOS Setup 58
BIOS-Update 58
Blacklist 217, 235
Boost-Converter 278
Boot 236
Boot-Loader 35
Bootpartition 44
Boot-Teil 43
Breadboards 198
Broadcast 149
Broadcom 63, 265
Broadcom BCM2835 33, 41

Cache-Speicher 70
Camera Serial Interface, CSI 32, 98, 117, 193, 236
C-Compiler, gcc 173, 188, 191
CD/DVD-ROM-Laufwerk 141
Central Processing Unit, CPU 64
Chip Package 63
Chip-zu-Chip-Kommunikation 205
Cinch-Adapter 127
Clock/Calendar 239, 242
Codecs 137

K. Dembowski, *Raspberry Pi – Das technische Handbuch,*
DOI 10.1007/978-3-658-08711-1, © Springer Fachmedien Wiesbaden 2015

Command Line Tool 237
Common Unix Print System, CUPS 142
Compiler 174
Composite RCA 14
Composite Video 77, 82
Compute Module 101, 120, 270
Compute Module Development Kit 13, 103
Computermonitor 19
config.txt 267
Configure Keyboard 31
Cortex 68
Cortex-M0 70
cpuinfo 133
CUPS-Konfigurationsseite 142, 144

D/A-Wandler 220, 239
Dateisystem 41
Datum 30
DDC, Display Data Channel 79
DDR2-SO-DIMM-Speichermodul 102
Debian Squezze 41
Debian Wheezy 41
Debug Interface 70
Delphi XE4 176
Device Tree-Format 119
Device-File 221
Digital Analog Converter, DAC 82
Digital Visual Interface, DVI 78
Direct Sequence 161
Domain Name Service, DNS 151
DOS-Programme 176
Dotted Decimal Notation, DDN 148
Druckfunktionalität 141
DVI/HDMI-Anschluss 21
DVI-Adapter 129
DVI-HDMI-Adapter 19
Dynamic Host Configuration Protocol, DHCP 146
DynDNS-Dienst 152

Edison 8
EEPROM-Controller 89
Eingabegeräte 18
Embedded MultiMedia Card-Flash-Speicher, eMMC 10
Embedded Processors 68
Embedded System 1, 85, 264
Embedded-Applikation 102
Embedded-Systeme 179
Emdedded-Anwendungen 239
eMMC 10, 102, 105
Energy Harvesting 209
Energy Processor 211
Entpacker 126
Ethernet 15
Ethernet-Standard 144
Extensions 45
Externe Laufwerke 137

FAT32-Format 43
FAT-Partition 58
Festplatte 43
File Allocation Table 43
File Transfer Protocol, FTP 157
Firmware 58
Flash 58
Flash-Speicher 105
Floating Point Unit 41, 70, 123
Floating-Point 33
Free Pascal 177
Frequency Hopping 161
FRITZ! Box 165
fstab 139
Full-Duplex-Übertragung 214

Gadgeteer-Plattform 7
Galileo 7
Gateway 148
gcc 190
gcc-Compiler 176
General Purpose Input Output, GPIO 111
Github 60
GNOME-Desktop 179
GNU C-Compiler 191

GPIO 57
GPIO-Anschlüsse 197
GPIO-Connector 236
GPIO-Kontaktleiste 112
GPIO-Leitungen 231
GPIO-Pins 197
GPIO-Port 109, 117
GPIO-Schnittstellen 111
GPIO-Signale 199
GPIO-Treiber 199
GPIO-Unterstützung im Kernel 201
GPU-Frequenz 268
Grafikeinheit 77
gutenprint 144

Halt-Modus 265
Handshaking 206
Hardware Attached on Top, HAT 119
Hardware-Programmierung 171
HDMI (High Definition Multimedia Interface) 19
HDMI-Ausgang abschalten 265
HDMI-Buchse 79
HDMI-Schaltung 81
HDMI-Verbindung 78, 128
Heartbeat 194
High bandwith Digital Content Protection, HDCP 19
home directories 46
Hostadresse 148, 149

I/O-Board 103, 105, 120
I/O-Controls 238
I/O-Port 256
I²C-Bauelement 230
I²C-Bibliotheken 254
I²C-Bus 80, 114, 228, 229
I²C-Bus-Adresse 233
I²C-Bussystem 230
I²C-Programmiermöglichkeiten 237
I²C-Schnittstelle 236
Identifizierungssignale 118
Idle 26, 182
Idle-Modus 265, 272
IEEE 801.11 160
IEEE-WLAN-Standards 161
ifconfig 147
Image Writer 126
Image Writer-Programm 126
Input/Output Controls, IOCTL 218, 221, 237
Integrated Development Environment, IDE 182
Inter Integrated Circuit Bus 228
Interactive Mode 184
Interfaces-Datei 150
Internationalisation Options 29
Internet-Adressklassen 149
Internet-Of-Things, IoT 8
Interpreter-Sprache 183
IP Next Generation 149
IP-Adresse 148

Java Development Kit, JDK 178
Java Runtime Environment, JRE 178
Joint Test Action Group, JTAG 89

Kernel 46
Kernel Images 58
Kernel-Modul 33, 129, 195
Kernel-Nachrichten 205
Kernel-Software 59
Klinkenstecker 127
Kommandogruppen 210
Kommandozeilen-Historie 48
Kommandozeilen-Komplettierung 49
Kompilierung 191, 227
Kondensatorbank 209
Konfigurationsprogramm 23
Konfigurierung 27
Konsolenmodus 263
Kontrollstrukturen 186

Ladegerät 17
Laufzeit 277

LCD-Ansteuerung 253
LC-Display 241
LCD-Routinen 253
Leuchtdioden 194
Lightweight X11 Desktop Environment, LXDE 25
Linux Kernel 199
Linux-Befehle 41
Linux-Dateibaum 44
Linux-Distribution 1
Linux-Distributionen 48
Linux-Format ext4 43
Linux-Host-System 106
Linux-Programme 57
Linux-Tasten 50
Linux-Treiber 142
Lithium-Ionen-Akku 276
Lithium-Zellen 276
Lochrasterplatine 220
Log-Datei 137
Logout 27
Loopback 149
Loopback Interface 147
Low Dropout Regler, LDO 275
Luftdrucksensor 230
LXDE 25, 49
LXDE-Desktop 29, 128, 133, 156, 180
LXTerminal 26

MAC Engine 87
Machine Instruction Format, MIF 174
MacOS 181
MacOS-Applikationen 176
MagI^{3}c 263, 270, 276
make 129
Make-Datei 247
Make-Files 191
Maker Scene 7
Maschinensprache 174
MAX232 209
Media Access Control, MAC 147
Mediacenter 135
Mediacenterlösungen 15
Mediadaten 132
Micro SD 74
Micro SD Card 15
Micro SD-Slot 109
Microsoft .NET Framework 179
Microsoft-Technologien 176
microUSB-Anschluss 15
Microwire 218
Midnight Commander 52
Midnight CommanderXE 52
Midori 26, 57
Mikrocontroller 171
Mnemonics 173
mnemonischen Codes 173
Mobile Industry Processor Interface, MIPI 97
Mobile Units 163
Modell B+ 109
modprobe 195, 236
Monitor 19, 128
Monitorabstimmung 34
Monitormenü 20
Mono 179
Mounten 43, 139
Multi Modulation Carrier, MCM 161
Multimedia SoC 64
Multiple Input Multiple Output 162

nano 189, 194
Netbook 136
Netzteil 16
Netzwerkanschluss 85
Netzwerkeinheit 87
Netzwerk-Interface 88
NiMH-Akkus 276
No Infra Red, NoIR 100
NOOBS 15, 125
Norton Commander 52
Nullmodem-Schaltung 206

OFDM 161
OnBoard-LED 193
Opcodes 173

Open Source Community 78
Open-Drain 230
OpenELEC 136
OpenGL 77
Open-Source-Software 124
Optimierung 123
Orthogonal Frequency-Division Multiplex-Verfahren, OFDM 161
Overclocking 32

P5-Kontakte 117
Paketmanager 55
Panel-Link 78
Password 28
PC-Mainboards 229
Peripheriebus 228
Perl 178
Pfostenleiste, Expansion Header 193
PHP5 178
Physical Computing 5
Physical Layer 87
Pi Camera 100
Pi Cobbler 197
Pidora Linux 124
Platine 1
Polyfuses 92
Potentiometer 241
Power Packs 278
Power-LED 265
Processing 6
Programmiersprache 188
Programmiersprache C 173
Programmierung 171
Pull-Up-Widerstände 238
Pulse Width Modulation 84
PuTTY 154
PWM-Ausgangssignal 114
PWM-Signale 84
pygame-Bibliothek 186
Pygame-Module 182
Python 3, 181, 196
Python Games 26, 182
Python-Bibliothek 203, 219
Python-Entwicklungsumgebung 26
Python-Programmierung 186
Python-Shell 183
Python-Versionen 182

Quark SoC X1000 7

Raspberry Login 24
Raspberry Pi 3, 8, 15, 207, 224
Raspberry Pi Compute Module 101
Raspberry Pi Foundation 3, 15, 57, 63, 112, 116, 123, 133, 171, 265, 274
Raspberry Pi-Hardware 117
Raspberry Pi-Schaltung 63
Raspbian 41, 45, 123, 133
Raspbmc 134
Raspbmc-Installation 134
raspi-config 23, 27, 37, 107
raspistill 101
raspivid 101
Rastrack 32
RCA Jack 77
Read Only Memory, ROM 58
Real Time Clock, RTC 242, 245
Reboot 217
Rechte verändern 55
Rechtemaske 54
Recovery Mode 40
Reduced Instruction Set Computer, RISC 124
Registerprogrammierung 200
Repositories 60
Repository 56, 106
Reset-Impuls 95
Reset-Switch 97
root 27, 55
Root-Dateisystem 28
root-Rechte 204
Root-Verzeichnis 44
Router 146
rpi-update 60
RS232 206
RS232-Standard 205

RS232-Treiber 208
Ruhestrom 265
Runlevel-Verzeichnis 157

SATA-Port 10
SCART-Anschluss 77
Schaltregler PAM2306 270
Schleifen 186
Schnittstellen-Kommunikation 193
Schreibdatenrate 75
Schreibschutz 73
Scratch 26, 180
SD Card 73, 74, 134
SD Card-Steckplatz 105
SDHC Card 73
SD-Karte 1
SD-Karteninterface 76
SDRAM 58, 63
SDRAM-Parameter 229
SDRAM-Speicher 35
SDXC Card 73
Secure Shell, SSH 35, 153
Serial Peripheral Interface, SPI 213
serielle Schnittstelle 205
shebang line 185
STP 145
Sketches 7
Skriptsprachen 177
Sleep-Modi 265
Smartphone 17, 135, 136, 279
socket API 187
Sockets 7
Soft Float 124
Software 41
Spannungsregler 104, 270
Spannungsversorgung 71, 269
Speicheraufteilung 34
SPI 213
SPI-Betriebsarten 216
SPI-Schaltkreise 227
SPI-Schnittstellen 218
SPI-Übertragungsmodus 224
SSH-Server 154
SSH-Verbindung 36
Standardbibliothek 189
Standby-Modus 271
startx 49
Steckplatinen 198
Stromverbrauch 268
Subnet 148
Subnet-Mask 148
sudo 27
Superuser 27
System Management Bus, SMB 229
System on Chip, SoC 8, 58, 64, 70
Systemeinstellungen 23

Taktfrequenz 266
Taktveränderungen 269
TAP Controller 89
Tastatureinstellungen 31
Tastenkombinationen bei Linux 50
TDMS-Kanäle 78
Temperatursensor 254
Testseite drucken 144
Tight VNC 155
Transceiver 209
Transition Minimized Differential Signaling Protocol, TDMS 78
Trigger-Bedingung 194, 195
TTL-RS232-Konverterschaltung 208
Turbo Pascal 176
TV-Einstellungen 135
Twisted Pair-Anschluss 89
Twisted Pair-Buchse 144
Twitter 32
Two Wire Interface, TWI 229

UART 208
Ubuntu 10
Uhr stellen 246
Uhrzeit 30
USB Battery Packs 278
USB to Serial-Adapter 205
USB/Ethernet-Hub 110
USB-Festplatte 138

USB-Geräte 85
USB-Gerätedaten 141
USB-Hub 18, 86, 137
USB-Laufwerkstypen 137
USB-Schaltung 90
USB-Spannungsanschluss 274
USB-Stick 43

Verbindungsverfahren 152
Very Secure FTP Daemon 158
Verzeichnisbaum 47
Verzeichnisse unter Linux 46
VGA-Signale 78
Video Core IV 14
Video-Anpassung 34
Video-Codec H.264 135
VideoCore IV 77
Video-Formate 133
Video-In-Eingang 77
Virtual Network Computing, VNC 154, 155
VLC Media Player 132
VLC-Player 136

Wärmebildkamera 272
Wearable-Electronic 8
Web-Browser 26, 199
WiFi Config 26, 166
Wi-Fi Dongle 22, 165
Win32 Disk Imager 38, 126
WinSCP 157
Wired-AND 230
Wireless Local Area Networks 160
Wirkungsgrad 277
WLAN 135
WLAN Access Point 167
WLAN-Adapter 36, 162, 165
WLAN-Realisierungen 164

XBian 136
XBMC 135
XBox Media Center, XBMC 134
XScale-Prozessoren 66
X-Server 31
XTRINSIC-Sense Board 198
X-Window 53

YATSE 135
Yocto 8

Zeichensätze 29
Zeitzone 30
Zero Gecko 69
Zugriffsrechte 50