Ausarbeitungen zum Seminar

Raspberry Pi inspected

Wintersemester 2013/2014

05. Februar 2014

Fakultät II – Informatik, Wirtschafts- und
Rechtswissenschaften
Department für Informatik
Abt. Systemsoftware und verteilte Systeme

www.svs.informatik.uni-oldenburg.de

Inhaltsverzeichnis

Raspberry Pi: Von der Idee zur Realisierung

Moritz Müller

I. Einleitung

Nicht erst seit ein paar Jahren bahnt sich ein Mangel an Informatikern an. Derzeit werden in vielen Ländern Fachkräfte gesucht, da ohne diese eine geringere Wettbewerbsfähigkeit und niedrigere Gewinne zu erwarten sind. In Japan berichteten bei einer Umfrage 85 Prozent der befragten Firmen von Problemen bei der Suche nach Fachkräften [1]. Durch das verstärkte Heranführen von Kindern und Jugendlichen, insbesondere hinsichtlich der Grundlagen zur Computer und Softwareanwendung, beschäftigt sich die Jugend immer mehr mit Methoden und Möglichkeiten der Informationstechnologie. Warum werden dennoch Informatiker händeringend gesucht? Es sind nicht unbedingt wenige Personen, die sich dafür entscheiden Informatik zu studieren. Im Jahr 2013 hat sich die Rekordzahl von 50.898 Studienanfängern für den Studiengang Informatik eingeschrieben. Jedoch brechen über 50% der Studienanfänger das Studium ab [2]. Dies hängt häufig mit falschen Erwartungen an das Informatikstudium zusammen. Viele Studienanfänger scheitern an den elementaren Mathematikanteilen des Studiums und brechen dieses daraufhin ab. Es gibt viele Überlegungen, wie man den potentiellen Studienanfängern den Einstieg erleichtern könnte. Dies fängt mit dem Informatikunterricht in der Schule an und geht hin bis zu speziellen MINT-Förderungen[1]. Der Raspberry Pi wurde entwickelt, um die Jugend an die hardwarenahen Aspekte der Informatik heranzuführen und zu begeistern.

II. Raspberry Pi

Der Raspberry Pi ist ein kleiner, kreditkartengroßer Einplatinenrechner. Sein Verwendungszweck ist das Experimentieren mit Soft- und Hardware, aber auch das Realisieren und Implementieren für Projekte, deren Durchführbarkeit mit dem kleinen und kostengünstigeren Raspberry Pi durchführbar wären.

A. Idee und Intention

Die Idee des Raspberry Pi ist zu großen Teilen auf Eben Christopher Upton (*1978 in Pontypool) zurückzuführen, der als Technical Director bei der US-amerikanischen Firma Broadcom, einem Anbieter für Halbleiterlösungen im Bereich der Netzwerkanwendungen, arbeitet. Eben Upton war von September 2004 bis August 2007 im St John's College, University of Cambridge, als Dircetor of Studies in Computer Science angestellt. In dieser Position war er für die Aufnahmeinterviews der Studenten und das allgemeine Koordinieren des Grundstudiums in Computer Science zuständig [3]. Dabei fiel Upton auf, dass das Verständnis von Computern im Laufe der Jahre immer weiter abnahm. Die frühere Generation war geprägt vom Programmieren allein durch die Bedienung der damaligen Computer, wie zum Beispiel des Commodore 64. Hingegen wandelte sich dieses Computerverständnis, auch mit der Entwicklung der Computer, die mehr auf grafische Benutzeroberflächen setzten, in oberflächlicheres Wissen, zum Beispiel im Umgang mit Textverarbeitungsprogrammen. Upton vermutete, dass die Schulen maßgeblich an dieser Entwicklung beteiligt waren, indem sie keine Einblicke in die Hintergründe und Architekturen der Programme und Computer vermittelten und dadurch den Eindruck erweckten, als würde der Computer auf diese Art der Anwendung beschränkt sein. Um wieder ein größeres Bewusstsein für Programmierung zu schaffen zu können, entwickelte er zusammen mit Rob Mullins, Jack Lang und Alan Mycroft eine Plattform für Kinder, mit deren Hilfe diese das Programmieren erlernen können [4], [5].

[1]Mathematik, Informatik, Naturwissenschaften & Technik ist eine Initiative, die Talente in diesen Bereichen fördert und damit dem Mangel an Nachwuchs in MINT-Qualifikationen senken will.

Tabelle I
Modelle des BBC Mikrocomputers [8]

Modell	RAM / Clock	CPU / Clock
A	16 kB / 4 MHz	6502A / 2 MHz
B	32 kB / 4 MHz	6502A / 2 MHz
B+64	64 kB / 4 MHz	6512 / 2 MHz
B+128	128 kB / 4 MHz	6512 / 2 MHz

B. Orientierung

Bei dem Entwurf des Raspberry Pi ließ sich Eben Upton laut eigener Angaben von der BBC Microcomputer System (BBC Micro) Serie von Microcomputern der Firma Acorn inspirieren [4], die ebenfalls in Schulen Großbritanniens eingesetzt wurde, um die Technik und den Umgang mit Computern nahezubringen.

Die British Broadcasting Coporation (BBC) schrieb mit dem BBC Computer Literacy Project einen Wettbewerb aus, um einen Mikrocomputer zu präsentieren und zu bewerben, der "modelling and simulation, graphics, sound, artificial intelligence, control, and communications" [6] realisierte. Damals reichte die britische Computerfirma Acorn Computers Ltd. eine Weiterentwicklung ihres Acorn Atom Mikrocomputers den Acorn Proton ein. Dieser wurde schließlich unter Einsendungen von Sinclair Research, Dragon Data und Newbury Laboratories ausgewählt und fand als BBC Micro großen Anklang in Großbritannien. Es wurden ungefähr 80 Prozent der Schulen mit BBC Micros ausgestattet [7].

Es gab unterschiedliche Modelle des BBC Mikrocomputers, die sich neben dem stetig größer werdenden Speicher vor allem in den Erweiterungsmöglichkeiten unterschieden.

Der BBC Model B hatte im Vergleich zum Model A vor allem eine größere Anzahl an Ports, zu denen neben RGB, Econet und RS423 auch Tube zählte. Letzterer war eine Erweiterungsmöglichkeit, durch die sich ein zweiter Prozessor am BBC Micro betreiben ließ. Ein zusätzlicher Zilog Z80 erweiterte neben seiner Leistung von 2,5 MHz auch den RAM um 12 kB statischen und 15 kB dynamischen RAM [9]. Da die Serie auf eine UMA[2]-Architektur aufgebaut war und es zwei Zugänge zum Speicher gab, einen für die CPU und einen für die graphische Berechnungen, wurde der Takt des RAMs auf die doppelte Frequenz des eingebauten Prozessors gesetzt. Dadurch war der Speicher schnell genug, den beiden Eingängen nachzukommen und UMA typische Zeitverzögerungen zu vermeiden.

C. Modelle und Prototypen

Aus den ersten Prototypen aus Steckplatten entwickelten sich Prototypen auf Leiterplattenbasis, die nach und nach durch neuere Revisionen abgelöst wurden und dabei Fehler korrigierten und Verbesserungen einbauten.

1) Konzept 2006: Eben Upton verfolgte seine Idee und entwickelte im Jahr 2006 einen frühen Prototypen im Eigenbau. Dieser bestand aus einer langen Steckplatte (Stripboard) von Vero Technologies Ltd, die er mit Dual in-line packages (DIP) bestückte, einer Gehäuseform, auf deren Rückseite sich zwei Reihen von Pins zum Anschließen befinden. Dazu kamen ein auf 22,1 MHz getakteter ATMega644 Mikrocontroller von Atmel und 512 kB SRAM. Dieser Prototyp wurde ebenso testweise als Printed Circuit Board (PCB) realisiert [10].

2) USB Prototype Board: Im Mai 2011 folgte das USB Prototype Board des Raspberry Pi. Die Leiterplatte hat die Größe eines USB-Sticks, wobei sich an dem einem Ende ein USB 2.0 Anschluss befindet, der das Board mit Strom versorgt, und auf der anderen Seite ein HDMI-Ausgang, der Bild und Ton ausgibt. Bestückt wurde das Board mit dem Broadcom BCM2835 System on Chip (SoC). Dieser Chip enthält den ARM1176JZF-S ARM11 Prozessor, der mit 700 MHz getaktet ist. Zudem ist als GPU der Videocore IV Prozessor enthalten. Ebenso sind 128 MB SDRAM vorhanden. Zusätzlich ist auf dem Board eine Kamera integriert. Ein SD/MMC/SDIO Karte kann über einen integrierten Slot angeschlossen werden und stellt so den Speicherplatz des Systems [11].

3) Alpha Board: Das Alpha-Board ist ein frühes Modell des Raspberry Pi Boards, das im August 2011 entstand. Das PCB ist dabei ungefähr 20 % größer als das angestrebte Kreditkartenformat mit 85,60 mm x 53,98 mm. Für den finalen Entwurf waren ursprünglich 4 Schichten vorgesehen. Seit dem Alpha Board besitzt das Raspberry Pi Board 6 Schichten, die auch in den Nachfolgemodellen erhalten blieben. Die Schichten des Alpha Boards bestehen aus High-Density-Interconnect-Leiterplatten (HDI). Diese haben den Vorteil, dass sie feinere Leitungsstrukturen schaffen und Durchkontaktierungen weniger Platz einnehmen. Blind Via, eine Durchkontaktierung, die durch eine Schicht auf einer Zwischenschicht endet, und Buried Via, die nur die Zwischenschicht durchtrennen, bilden Debugkontaktierungen, die auf dem Alpha Board genutzt werden [12].

Das Board besitzt zwei USB 2.0 Schnittstellen und einen 10/100 Ethernet Anschluss, die über den von Smart Mixed-Signal Connectivity (SMSC) hergestellten LAN9512 Controller realisiert werden [12], [13]. Ein HDMI, ein Composite-Video Ausgang und eine 3,5 mm Klinkenbuchse sind als Ausgänge integriert. Auf dem Board sind 16 3,3 Volt General-Purpose Input/Output (GPIO), zwei 3,3 Volt Inter-Integrated Circuit (I^2C) und ein 3,3 Volt Serial Peripheral Interface (SPI) Pin vorhanden [14]. Zum Debuggen gibt es zusätzliche Schalter und LEDs auf dem Alpha Board. Der SoC ist der BCM2835 mit integrierter CPU und GPU. Der SDRAM ist nach der Package on Package (PoP) Fertigungstechnik auf den BCM2835 aufgesetzt. Mit Strom versorgt wird das Alpha Board mittels eines AC/DC Anschlusses. Der SD/MMC/SDIO Slot ist weiter vorhanden.

4) Beta Board: Das im Dezember 2011 vorgestellte Beta Board des Raspberry Pi entspricht weitestgehend dem finalen Entwurf. Die Hard- und Software der Beta Boards wurde ausführlich getestet. Im Vergleich zum Alpha Board entsprechen die Maße nun dem Kreditkartenformat. Es wurden einige der zu Testzwecken eingeführten Pins, Schalter und LEDs herausgenommen. Des Weiteren wurden die Pins und eine Großzahl der Anschlüsse versetzt oder gedreht. Der AC/DC Adapter-Anschluss ist mit einem 5 Volt Micro USB Anschluss zur Stromversorgung ersetzt worden.

Es wurden 100 Beta Boards in China produziert, von denen die ersten 10 auf Ebay versteigert wurden, um Geld für die Raspberry Pi Foundation zu sammeln [15], [16]. Von dem gesammelten Geld sollen Raspberry Pi für Schulen zur Verfügung gestellt werden [17]. Das Raspberry Pi Beta Board mit der Seriennummer #1 wurde für 3500 britischen Pfund versteigert.

5) Model B Version 1 und REV 0003: Am 10. Januar 2012 wurde verkündet, dass die Produktion des Raspberry Pi Model B gestartet wurde. Eine Charge von 10.000 Model B wurde in China gefertigt und dann nach Großbritannien geliefert [18]. Am 29. Februar 2012 wurden dann die ersten Modelle verkauft. Der Preislimit von 25 US-Dollar ist eingehalten worden. Es befinden sich insgesamt 26 Pins [19] für General Purpose IO auf dem Model B:

- Ein 5 Volt Pin.
- Sechs Do Not Connect Pins, die 5V bereitstellen oder auf Masse (0V) geschaltet sind, aber in der Version 1 des Model B nicht aktiviert sind [19].
- Ein 3 Volt Pin.
- Ein GND Pin, der auf Masse geschaltet ist (0V).
- Zwei Universal Asynchronous Receiver Transmitter (UART) Pins. UART ist dabei eine Implementierung, um serielle Schnittstellen umzusetzen. Es gibt einen Transmitter Data (TXD) Pin, der die Daten einzelner Bits nacheinander verschickt und einen Receiver Data (RXD) Pin, der den Datenfluss aufnimmt.
- Acht General Purpose Input/Output Pins.
- Fünf Serial Peripheral Interface (SPI) Pins. SPI ist ein serielles Bussystem. Entwickelt wurde es von Motorola und funktioniert nach dem Master-Slave-Prinzip. Es gibt

[2]Unified Memory Architecture ist eine Architektur, die einen Speicher für mehrere Komponenten stellt.

einen SPI_SCLK Pin, der vom Master mit allen Slaves des SPI-Verbunds verbunden ist. Diese Verbindung stellt den hauptsächlichen Datenkanal dar. Der SPI_MOSI Pin ist eine Kontaktstelle, die vom Master (Master Out) in den Slave verbunden wird (Slave In). SPI_MISO beschreibt hingegen einen Kontaktpunkt, von dem eine Leitung vom Slave (Slave Out) in den Master geht (Master In). Die Pins SPI_CE0_N und SPI:CE1_N sind mit jedem Slave verbunden und selektieren für eine Übertragung vom Master den empfangenden Slave.

- Zwei Pins, die das ebenfalls serielle Datenbussystem I^{2}C steuern. Der Serial Data Line (SDA) Pin wird mit den Slaves verbunden und ist die hauptsächliche Datenleitung. Daneben gibt es den Serial Clock Line (SCL) Pin, von dem der Takt mit dem die Bits übertragen werden, übermittelt wird. Adressiert wird, anders als bei SPI, mit dem ersten vom Master gesendeten Byte.
- Alle UART-, SPI- und I^{2}C-Pins können als GPIO umkonfiguriert werden.

Daneben gibt es noch ein Camera Serial Interface (CSI) zur Anbindung einer Kamera und ein Display Serial Interface (DSI), um ein Display anzuschließen. Über die fünf LEDs OK, PWR, FDX, LNK und 10M wird der Status des Raspberry Pi angezeigt. Der BCM2835 ist weiterhin integriert und stellt CPU und GPU. Der stacked SDRAM betrug vorerst 265 MB und wurde am 15.10.2012 auf 512 MB verdoppelt. Der Preis blieb dabei gleich [20]. Die erste Version der Model B Boards besitzt die Revisionsnummer 0002. Diese Modelle weisen Mängel auf, die in der Revisionsnummer 0003 korrigiert und im August 2012 ausgeliefert wurden [21]. Diese beinhaltet die Engineer Change Notice ECN0001.

Benutzer, die mehrere Geräte mittels HDMI zum Beispiel Fernsehgerät, Aufnahmegeräte und DVD-Player miteinander verbinden, machen sich gegebenfalls die Consumer Electronics Control (CEC) Technologie zu nutzen. Dabei gibt es in der HDMI Spezifikation seit Version 1.0 die serielle CEC Schnittstelle, die mit den verbundenen Geräten einen Bus aufbaut. Über diesen Kanal können mehrere HDMI-fähige Geräte mittels eines Switchs (AV-Receiver) miteinander kommunizieren. Dabei fungiert das Display als Hauptkontrolleinheit des Bus und Switches als Zweigstellen. Ein Anwendungsfall ist, wenn eine DVD in den DVD-Player eingelegt wird und die Wiedergabefunktion aktiviert wird. Über den CEC Bus wird der DVD-Player per One Touch Play automatisch zur aktiven Quelle des Root Displays [22]. Wurde der Raspberry Pi, ohne mit Strom versorgt zu sein, in solch einen Bus eingebunden, führte dies dazu, dass die Geräte nicht mehr miteinander kommunizieren konnten. Ursache ist eine Fehlfunktion der ESD Protection der CEC Schnittstelle am Raspberry Pi. Damit der Bus wieder funktionierte, durfte der Raspberry Pi nur eingeschaltet am Bus eingebunden sein und die Fernsehgeräte mussten auf Werkseinstellungen zurückgesetzt werden. Das Problem wurde dadurch gelöst, dass die Diode D14 entfernt wurde. In der Revision 0003 wurde hingegen die 3V-Anbindung an D14 durch eine 5V Anbindung ersetzt (siehe Abbildung 1) [23].

Zudem gab es eine Grenze von 100 mA, die pro USB-Port bezogen werden konnte. Moderne Peripheriegeräte benötigen

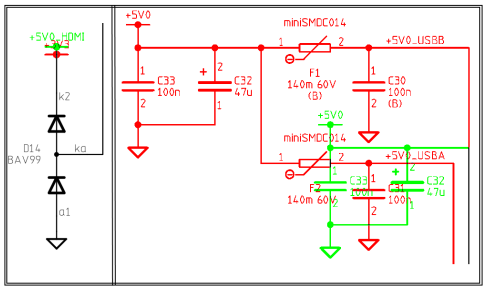


Abbildung 1. Fehler im Schaltplan bei D14 (links) und PPTC (rechts); Rot ist nur in Version 1 und Grün nur in Revision 2 enthalten [25], [26]

unter Umständen mehr Ampere. Verantwortlich dafür sind die zwei Polymeric Positive Temperature Coefficient Devices[3] (PPTC) F1 und F2. Damit diese Beschränkung aufgehoben wird, wird empfohlen entweder den PPTC zu überbrücken oder diesen gleich zu entfernen [24]. In der Revision 0003 wurden die Polyfuses F1 und F2 mit normalen Verbindungen ersetzt.

Ein weiterer Fehler war die Anbindung des Outputs des 1,8 Volt Regulators RG1 zu dem 1,8 Volt Pin (VDD18CORE) des LAN9512 Chips [27]. Diese Anbindung an den VDD18CORE „is not intended to power external nets. Equally it's not intended to ever be driven by external sources“ [28]. Aufgrund dieses Aufbaus kommt es bei einigen Raspberry Pi (Model B Version 1) zu folgenden Situationen: Der interne Regulator des LAN9512 weist eine höhere Voltzahl auf als RG1. Dadurch wird RG1 abgeschaltet und das Board nur durch den internen Regulator mit 1,8 Volt versorgt. Dies sorgt für eine unverhältnismäßig hohe Temperatur des LAN9512 Chips. Entgegengesetzt kann es auch vorkommen, dass die RG1 Voltzahl höher ist als die des internen Regulators des LAN9512. Dann läuft der Chip kalt [27].

6) Model B Revision 2: Am 5. September 2012 wurde bekannt gegeben, dass es eine Revision 2 des Model B geben wird (Revisionsnummern 0004, 0005, 0006) [21]. ECN0001 wurde dabei vollständig implementiert. Probleme mit der Diode D14 wurden durch eine Anbindung an die Stromversorgung mit 5 Volt anstatt den vorherigen 3 Volt eliminiert [29]. Auch die Polyfuses, die die USB-Ports beschränkten, wurden durch 0 Ohm Verbindungen ersetzt. Eine weitere Änderung ist, dass mit dem Trennen des SMSC 1V8 (Leitung zum VDD18CORE des LAN9512) von der Systemversorgung ein weiteres Problem des Model B Version 1 gelöst wurde. Mit dem Entfernen von vier redundanten GPIO Signalen, die zur Versionsidentifikation genutzt wurden, konnte eine neue Schnittstelle P5 hinzugefügt werden. Da diese ohne einen Header bestückt ist, muss vor der Verwendung dieser angebracht werden. Danach bietet dieser einen 5 Volt Pin, einen 3 Volt Pin, zwei GND Pins und 4 GPIO Pins. Es können damit unterschiedliche Konfigurationen genutzt werden:

- Nutzung als 4 GPIO Signale
- Nutzung als I^{2}C Kontaktstellen SDA und SCL

[3]PPTC ist ein passiver Überspannungsschutz. Kommt es zu einem Kurzschluss, erwärmt sich das Device und es wird hochohmig. Kühlt es ab, wird es wieder niederohmig.

Tabelle II
VERGLEICH VON MODEL B UND MODEL A (STAND: DEZEMBER 2013)

Eigenschaft	Model B	Model A
Preis	35 US-Dollar	25 US-Dollar
System on a Chip	Broadcom BCM2835	
CPU	ARM1176JZF-S (700 Hz)	
GPU	VideoCore IV	
RAM	512 MB	256 MB
Netzwerk	10/100 Ethernet	kein Anschluss
Höhe	17 mm	15 mm
Gewicht	40 g	31 g
USB Ports	2 x 2.0	1 x 1.0
Strombedarf [34]	700 mA	500 mA
Speicher	SD/MMC/SDIO Slot	
Format	85.0 x 56.0 mm	
Videoausgabe	HDMI, Composite Video	
Audioausgabe	HDMI, 3.5 mm Klinke, I^2C	
Andere Eingaben	GPIO, I^2C, UART, SPI	

- Nutzung als I^2S Channel. I^2S ist eine Schnittstelle zum Austausch von Stereo-Audio-Daten. Der I^2S BUS hat drei Verbindungen. Eine continuous serial clock (SCK), einen word select (WS) und eine serial data (SD) line.

Ebenso wurde die Schnittstelle P6 hinzugefügt. Diese besteht aus 2 Pins und eignet sich dazu, den Raspberry Pi zu resetten (CPU soft reset). Dazu muss der Pin 1 von P6 mit dem GND Pin in P6 verbunden werden. Um ein JTAG Debug Signal verfügbar zu machen, wurden zudem einige Pins des MIPI CSI-2 Interfaces und des P1 Headers umgeleitet. Die Label der LED D9 (10M) und D5 (OK) wurden zu 100 und ACT geändert. Es wurden drei Löcher zur Halterung in das Board gebohrt [21].

7) Model A: Am 30. November 2012 wurde verkündet, dass eine geringe Anzahl an Raspberry Pi Model A hergestellt worden sind [30]. Von diesen wurden wenige Exemplare an Freunde der Foundation weitergegeben. Weitere 12 wurden, genauso wie die 10 Beta Board im Jahr zuvor, auf Ebay versteigert. Der Gewinn aus diesen Auktionen wurde an Hilfsorganisationen gespendet [31].

Im Vergleich zum Model B des Raspberry Pi sind unterschiedliche Komponenten ersetzt oder verändert worden. Wichtigster Punkt ist hierbei das Entfernen des LAN9512 Chips. Infolgedessen ist auch der Ethernet Anschluss entfernt worden, ebenso ein USB-Port. Der verbleibende Port wird vom BCM2835 gesteuert. Das Model B besitzt 256 MB SDRAM (ursprünglich waren 128 MB geplant) [32].

Das Model A kostet 25 US-Dollar und spricht eine Interessentengruppe an, die Wert auf ein noch kleineres Format und vor allem einen geringeren Stromverbrauch legt. Durch das Entfernen des LAN9512 Chips reduziert sich der Stromverbrauch des Model A auf weniger als die Hälfte des Stromverbrauchs von Model B [33] und präferiert sich dadurch für einen Einsatz mit Batterien. Weitere Merkmale sind in der Tabelle II gelistet.

D. Produktionsstandort und Vertrieb

Der Raspberry Pi sollte neben der Förderung der Jugend auch Großbritanniens als Standort der Elektronikindustrie stärken. In Folge der Ankündigung des Model B wurde deshalb mit möglichen Herstellern in Großbritannien verhandelt. Jedoch konnten keine sinnvollen Konditionen ausgehandelt werden. Hersteller mit weniger gefüllten Lagerräumen verlangten Preise, bei denen die Produktion ein Minusgeschäft ergeben hätte [18]. Auf der anderen Seite konnten Hersteller mit profitablen Preisen meistens nur eine zu geringe Menge herstellen. Ein weiteres Problem stellten die hohen Steuern dar. Die vielen Teile mussten importiert und einzeln versteuert werden. In China ist diese Importsteuer nicht nötig. Daher wurden die ersten zehntausend Raspberry Pi Model B der Raspberry Pi Foundation in China hergestellt [18]. Im März 2012 wurde der Vertrieb und die Vermarktung des Raspberry Pi von den beiden britischen Herstellern Farnell element14 und RS Components übernommen. Diese vertrieben zudem die erste bereits produzierte Charge der Model B. Durch die Verkaufsnetze der Hersteller konnten für den Endkunden die Verschiffungskosten eingespart und die lokale Verfügbarkeit gesteigert werden.

Am 6. September 2012 wurde mit der Einführung der Rev 2 verkündet, dass der Hersteller Farnell element14 seine Produktion des Raspberry Pi in die Sony-Produktionsstätte in Pencoec in South Wales verlegt. Dadurch war es endlich möglich, einen Raspberry Pi „Made in UK“ zu verkaufen. Zudem wurden ungefähr 30 neue Arbeitsstellen geschaffen. In Wales werden 30.000 Raspberry Pi pro Monat hergestellt und getestet [35].

E. Software

Für den Raspberry Pi sind verschiedene angepasste Betriebssysteme verfügbar, zum Beispiel Fedora Remix oder Arch Linux ARM. Darüber hinaus gibt es mit OpenELEC eine primär auf multimediale Anwendungen ausgelegte Linux Distribution. OpenELEC ist eine schnell bootende Linux Distribution, die mit dem XBMC Media Center zusammen ein effektives und kostengünstiges Media Center Paket darstellt. Mit New Out Of Box Software (NOOBS) entwickelte die Raspberry Pi Foundation ein Programm, das es ermöglicht beim Start des Einplatinenrechners zwischen einer Auswahl von Betriebssystemen zu wählen. [36]

Viele Anwendungen müssen angepasst werden, damit sie die Architektur des Raspberry Pi sinnvoll nutzen. XMBC wurde beispielsweise angepasst, damit es die hardwaregestützte Grafikbeschleunigung des VideoCore IV unterstützt.

F. Probleme

Neben den o.g. Problemen, deren Lösung im Rahmen der Revisionen versucht wurde, gibt es noch weitere Ursachen, die den Raspberry Pi behinderten.

1) Fehler im Schema des BCM2835: Bei den 100 produzierten Beta Boards gab es ein Problem mit der Stromversorgung des BCM2835. Es lag ein Fehler in den Schemata des Raspberry Pi vor, so dass die SDRAM_1V8 Anbindung an den SoC nicht mit der 1,8 Volt Stromversorgung des

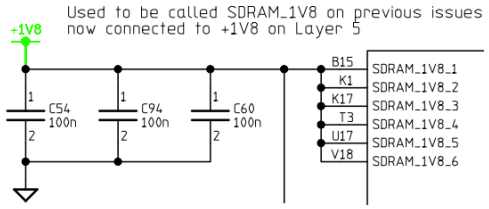


Abbildung 2. Fehler bei der Stromversorgung des Beta Board [25]

Boards verbunden war (siehe Abbildung 2). Dieser Fehler hatte zur Folge, dass der Raspberry Pi nicht startete. Glücklicherweise konnten die bereits produzierten Boards dadurch gerettet werden, dass an der naheliegenden 1,8 Volt Leitung die Beschichtung abgetragen und ein Tropfen Lötzinn aufgetragen wurde [37].

2) Falsche RJ45-Jacks verbaut: Nachdem die ersten Raspberry Pi Model B bereits verkauft worden waren und die Lieferung anstand, fiel auf, dass falsche Netzwerkanschlüsse verbaut worden waren. Die 10.000 georderten und aufgelöteten RJ45-Buchsen besaßen keine integrierten Magneten, die für den vorgesehenen Betrieb nötig waren. Die Magneten sorgen für ein besseres Signal, bessere Filterung und eine kürzere Übertragungen für Daten [38]. Daraufhin mussten 10.000 neue Netzwerkanschlüsse geordert und die alten ersetzt werden [38].

III. Schlussfolgerung

Mit dem Raspberry Pi ist es gelungen, mit einem preiswerten Produkt Kinder sowohl für Software als auch für Hardware von Computern zu begeistern. Von den bisher 1,75 Millionen verkauften Exemplaren sind über eine Million in Großbritannien hergestellt worden (Stand: Oktober 2013) [39]. IT-Riesen wie Google unterstützen die Förderung an Schulen in UK, indem sie 15.000 Raspberry Pi zur Verfügung stellen [40]. Wolfram Research stellt die mathematisch-naturwissenschaftliche Software Mathematica sowie die Programmiersprache Wolfram Language für Raspberry Pi kostenfrei zur Verfügung (regulärer Preis 150 Euro). Für den Raspberry Pi wurde Eben Upton mit der Silver Medal der Royal Academy of Engineering ausgezeichnet [4].

Literatur

[1] N. A. Zeitler. (2013, Juni) Fachkräftemangel engpassberuf informatiker. [Online]. Available: http://de.sap.info/engpassberuf-informatiker/94918

[2] BITKOM. (2013, April) Presseinfo informatik. [Online]. Available: http://www.bitkom.org/files/documents/BITKOM_Presseinfo_Informatik_2012_12_04_2013.pdf

[3] Linkedin. Linkedin upton. [Online]. Available: http://www.linkedin.com/in/ebenupton

[4] U. of Cambridge. (2013, Juli) Alumnus eben upton, co-founder of raspberry pi, has been awarded a silver medal from the royal academy of engineering. [Online]. Available: http://www.eng.cam.ac.uk/news/stories/2013/Raspberry_Pi/

[5] R. Pi-Foundation. About us. [Online]. Available: http://www.raspberrypi.org/about

[6] BBC. (1983, August) Towards computer literacy - the bbc computer literacy project 1979-1983. [Online]. Available: http://www.naec.org.uk/organisations/bbc-computer-literacy-project/towards-computer-literacy-the-bbc-computer-literacy-project-1979-1983

[7] D. D. Tibor Vasko. (1986, September) Working paper educational policies internal overview. [Online]. Available: http://webarchive.iiasa.ac.at/Admin/PUB/Documents/WP-86-052.pdf

[8] C. Why. Bbc microcomputers. [Online]. Available: http://acorn.chriswhy.co.uk/Computers/BBCMicros.html

[9] Zilog. Z80 data book. [Online]. Available: http://datasheets.chipdb.org/Zilog/Z80/Z80_DataBook.pdf

[10] E. Upton. Veroboard prototype rpi. [Online]. Available: http://www.raspberrypi.org/archives/264

[11] E. Protalinski. (2013, Mai) Raspberry pi: the $25 computer. [Online]. Available: http://www.techspot.com/news/43680-raspberry-pi-the-25-computer.html

[12] E. Upton. (2011, Juli) Alpha boards in manufacture. [Online]. Available: http://www.raspberrypi.org/archives/28

[13] S. Corporation. (2009, November) Lan9512 datasheet. [Online]. Available: http://pdf1.alldatasheet.com/datasheet-pdf/view/347287/SMSC/LAN9512.html

[14] L. Upton. (2011, Juli) Comment july 31 2011. [Online]. Available: http://www.raspberrypi.org/archives/28#comment-441

[15] ——. (2011, Dezember) We have pcbs! [Online]. Available: http://www.raspberrypi.org/archives/389

[16] ——. (2011, Dezember) Populated boards: an update on where we are. [Online]. Available: http://www.raspberrypi.org/archives/422

[17] ——. (2012, Januar) Auction update, factory video. [Online]. Available: http://www.raspberrypi.org/archives/498

[18] ——. (2012, Januar) We've started manufacture! [Online]. Available: http://www.raspberrypi.org/archives/509

[19] ——. (2011, November) Pinout for gpio connectors. [Online]. Available: http://www.raspberrypi.org/archives/384

[20] E. Upton. (2012, Oktober) Model b now ships with 512mb of ram. [Online]. Available: http://www.raspberrypi.org/archives/2180

[21] ——. (2012, September) Upcoming board revision. [Online]. Available: http://www.raspberrypi.org/archives/1929

[22] Q. Data. Designing cec into your next hdmi product. [Online]. Available: http://www.quantumdata.com/pdf/CEC_White_Paper.pdf

[23] E. Upton. (2012, August) Usb port current boost (solved). [Online]. Available: http://www.raspberrypi.org/phpBB3/viewtopic.php?f=63&t=8591&start=153

[24] G. van Loo. (2012, August) Usb port current boost (solved). [Online]. Available: http://www.raspberrypi.org/phpBB3/viewtopic.php?f=63&t=8591

[25] E. Upton. (2012, April) Model b schematics. [Online]. Available: http://www.raspberrypi.org/archives/1090

[26] ——. (2012, Oktober) Model b revision 2.0 schematics. [Online]. Available: http://www.raspberrypi.org/archives/2233

[27] D. W. Jones. (2012, August) Fixing 1.8v power rail design error. [Online]. Available: http://www.raspberrypi.org/phpBB3/viewtopic.php?f=29&t=14489

[28] selsinork. (2012, Juli) Rg1 1.8v regulator. [Online]. Available: http://www.element14.com/community/message/57246/l/re-rg1-18v-regulator#57246

[29] R. Pi-Foundation. (2012, Oktober) Raspberry pi r2.0 schematics. [Online]. Available: http://www.raspberrypi.org/wp-content/uploads/2012/10/Raspberry-Pi-R2.0-Schematics-Issue2.2_027.pdf

[30] L. Upton. (2012, September) First model a samples off the line! [Online]. Available: http://www.raspberrypi.org/archives/2615

[31] ——. (2012, Dezember) Twelve pics of christmas: We're aucauction off the first model as! [Online]. Available: http://www.raspberrypi.org/archives/2828

[32] E. Upton. (2012, März) And breathe... [Online]. Available: http://www.raspberrypi.org/archives/723

[33] RobertMM. (2013, September) Power consumption approximation of broadcom soc. [Online]. Available: http://www.raspberrypi.org/phpBB3/viewtopic.php?p=164893

[34] Raspberry-Pi-Foundation. Power. [Online]. Available: http://www.raspberrypi.org/faqs

[35] L. Upton. (2012, September) Made in the uk! [Online]. Available: http://www.raspberrypi.org/archives/1925

[36] R. Pi-Foundation. New out of box software (recommended). [Online]. Available: http://www.raspberrypi.org/downloads

[37] E. Upton. (2011, Dezember) More on the beta boards. [Online]. Available: http://www.raspberrypi.org/archives/470

[38] L. Upton. (2012, März) Manufacturing hiccup. [Online]. Available: http://www.raspberrypi.org/archives/781

[39] ——. (2013, Oktober) 1.75 million sold so far - and 1 million made in the uk. [Online]. Available: http://www.raspberrypi.org/archives/5016

[40] ——. (2013, Januar) 15,000 raspberry pis for uk schools - thanks google! [Online]. Available: http://www.raspberrypi.org/archives/3158

Raspberry Pi Hardware Overview and its Abstraction in the Linux Kernel

B.Sc. Sebastian Reichel
University of Oldenburg
Oldenburg, Germany
sre@kernel.org

***Abstract*—This paper is about the Raspberry Pi's hardware architecture. It provides an overview on the Raspberry Pi's hardware components and how they are abstracted by the Linux kernel.**

The paper also gives an introduction to Device Trees, since they are used to inform the kernel about the hardware components existence.

I. Introduction

This paper gives an overview on the Raspberry Pi's hardware components and how they are abstracted by the Linux kernel.

The paper will start with a rough overview of the Raspberry Pi's hardware components in section II. In section III follows a detailed description of the System on a Chip (SoC) used by the Raspberry Pi. The following section IV describes the Ethernet chip, that is part of the Raspberry Pi Model B. Next there is chapter V, which introduces the kernel's Device Tree concept. Afterwards a short section (VI) highlighting the differences between the different Raspberry Pi models follows. The last section (VII) contains a conclusion of the Raspberry Pi's hardware architecture and its abstraction in the Linux kernel.

II. Overview

Figure 1 gives a rough overview over the hardware components of the Raspberry Pi Model B. Model A looks the same, except, that the Ethernet related components are not assembled.

The Broadcom BCM2835 chip, which can be found in the middle of the Raspberry Pi, is a SoC. It provides most of the Raspberry Pis functionallity, but lacks support for Ethernet. For Ethernet support an USB Ethernet controller (SMSC LAN9512) is attached to the BCM2835 [22]. The Ethernet controller also contains an USB-Hub to provide two instead of a single USB port. All of the remaining interfaces (HDMI, Audio, Composite Video, GPIOs, DSI, CSI) are connected to the BCM2835 SoC.

There are three regulators on the Raspberry Pi's Printed circuit board (PCB) providing 3.3, 2.5 and 1.8V in addition to the 5V. All of these are used to power different parts of the BCM2835 SoC. Apart from the SoC the Ethernet chip needs 3.3 and 1.8V.

The next section gives are more detailed overview over the BCM2835. Each of the BCM2835's hardware components will be described in its own subsection to reduce the complexity.

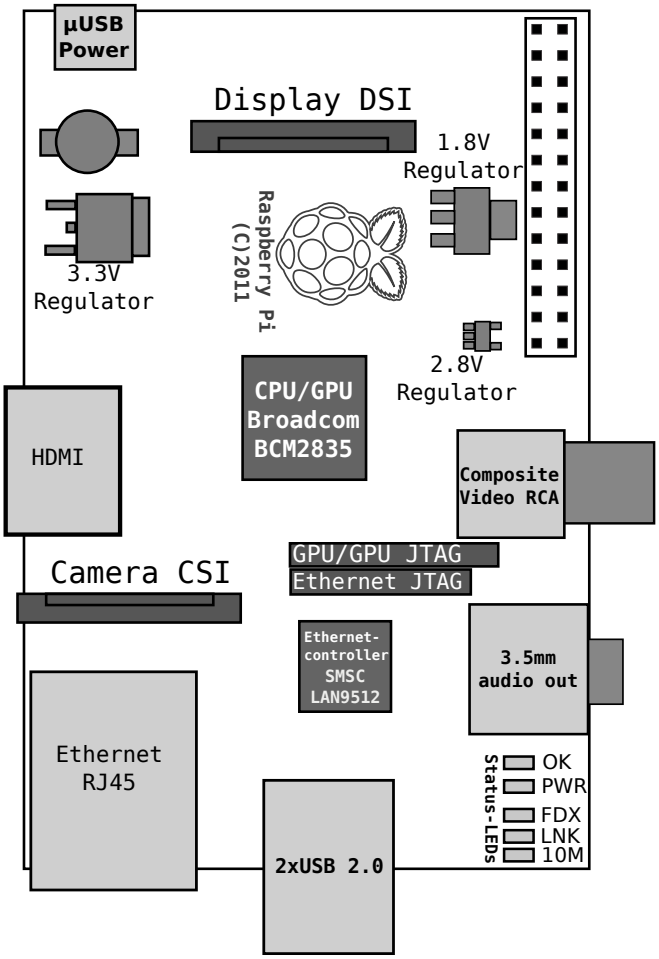


Fig. 1. Raspberry Pi Architecture

III. System on a Chip: BCM2835

The Raspberry uses a System on a Chip (SoC) called BCM2835 to provide most of its functions [22]. The SoC is a specific implementation of Broadcom's BCM2708 family [3].

It contains a Centeral Processing Unit (CPU), Memory Management Unit (MMU), Graphics Processing Unit (GPU), Camera & Display interfaces, Serial Peripheral Interfaces (SPI), Inter-Integrated Circuits (I^2C), Universal Asynchronous Receiver/Transmitter (UART), General Purpose Input/Output

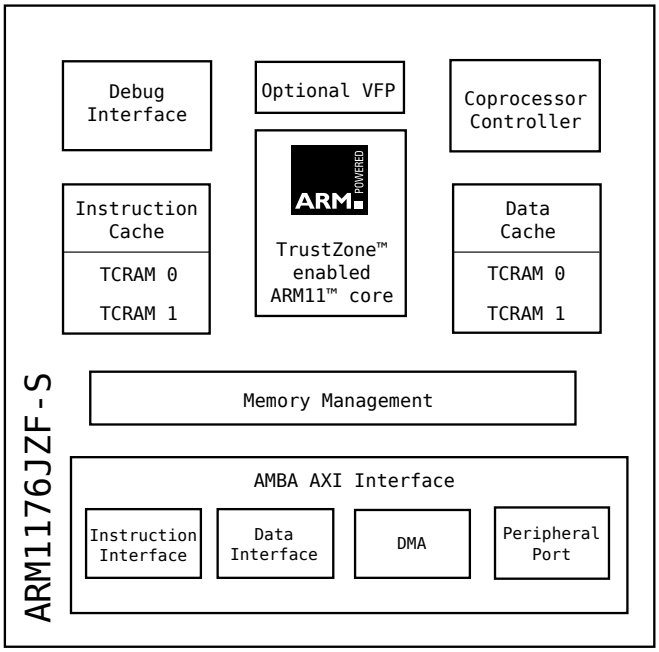


Fig. 2. ARM1176JZF-S Architecture

(GPIO), Interrupt (IRQ) Controller, Soundcard, Pulse-width modulation (PWM) Controller, Timer and Universal Serial Bus (USB) interfaces [4].

The mainline kernel currently only supports a basic subset of the BCM2835's features. For the Raspberry a custom kernel tree is used, which has support for most hardware components provided by the BCM2835 [12].

A. CPU: ARM1176JZF-S

The ARM1176JZF-S provides the actual CPU for the BCM2835. It supports the ARMv6 instruction set. The version built into the BCM2835 has additional support for the Vector Floating Point version 2 instruction set (VFPv2), which is used for floating point operations [2].

The ARM1176JZF-S contains an MMU and provides an Advanced eXtensible Interface (AXI) as system bus, which is the third generations of ARM's Advanced Microcontroller Bus Architecture (AMBA) [2].

The processor also supports the Jazelle instruction set, which can be used to run Java bytecode natively. Jazelle is implemented as a CPU mode similar to the Thumb-Mode. In this mode the ARM processor natively executes Java bytecode falling back to a software Java Virtual Machine for complex bytecode operations not implemented in hardware [8]. This instruction set is currently not supported by Linux and/or OpenJDK and is not publically documented.

Support for the ARM IP-Core and VFP is included in the mainline kernel since a long time ago and supported on all ARMv6 (and newer) based SoCs. The code lives in the following directories:

- *arch/arm* - General ARM support
- *arch/arm/vfp* - Floating Point support
- *arch/arm/mm* - Memory Management support
- *arch/arm/crypto* - ARM optimized AES & SHA1 routines

B. Random-access memory (RAM)

The Raspberry Pi's system memory is included in the BCM2835 SoC. The base address and memory size is loaded from Device Tree and differs between the Raspberry Pi models.

C. GPU: VideoCoreIV

VideoCore is a low-power mobile multimedia processor architecture from Broadcom, which has been developed by Alphamosaic Ltd originally. It features a two-dimensional DSP architecture making it flexible for decoding and encoding a wide range of multimedia codecs in software and maintaining low power usage [14]. The Raspberry Pi comes with DSP software, which provides OpenGL ES 2.0, OpenVG, Open EGL and H.264 high-profile decoding for resultions up to 1080p using 30 frames per second.

The GPU is capable of 1 Gpixel/s, 1.5 Gtexel/s or 24 GFLOPs of general purpose compute and features a bunch of texture filtering and DMA infrastructure. This means that graphics capabilities are roughly equivalent to the original Xbox's level of performance [5].

Detailed data and development tools are not available without NDA, but the CPU side of the graphic stack is available as free software. This stack consists of a kernel driver called VCHIQ, which takes care of the communication between the VideoCore and the ARM processor. The kernel driver can be found in *drivers/misc/vc04_services* and is not part of the mainline kernel. In addition a small helper driver is needed, which can be found in *rivers/char/broadcom/vc_cma/vc_cma.c*. This driver takes care of allocation a continuous memory area for the Video Core.

The other part of the stack are four libraries in the userspace, which provide OpenMAX, OpenGL ES, OpenVG and EGL functionality using the VCHIQ kernel interface */dev/vchiq*.

This driver will not be merged into the mainline kernel, since its not as open as it seems at a first glance. All of the OpenGL ES functionality is implemented directly in the (closed source) firmware running on the Video Core. The open source parts only forward the OpenGL commands to the Video Core [1].

There is currently work going on to reverse engineer and document the VideoCore IV instruction set used by the BCM2835 in the Raspberry Pi [6]. This resulted in different persons starting to work on support for the VideoCore for well known toolchains like LLVM or the GNU Toolchain. At the same time first RFC patches have been sent to the Linux kernel mailinglists, which add driver support for the BCM2835's mailbox inter-processor communication interface [15].

The Video Core is also used to actually move the image data onto the display. The Raspberry Pi Foundation's kernel contains a simple framebuffer driver for this task, which can be found in *drivers/video/bcm2708_fb.c*. This driver provides a

standard Linux framebuffer, which forwards data to the Video Core.

D. Camera Serial Interface (CSI)

The Raspberry Pi contains a lowlevel interface for cameras specified by Mobile Industry Processor Interface (MIPI) called Camera Serial Interface (CSI). This standard is used by many embedded systems and especially smartphones. The standard defines, that a CSI interface consists of an arbitrary number of data lanes, a clock lane and an additional I^2C interface for controlling the camera chip.

The CSI interface is directly connected to the Image Signal Processor inside the Video Core [22]. Normally these kind of image sensors are read via the Video4Linux 2 (V4L2) API, but the Raspberry Pi kernel is lacking the needed drivers. Instead the Raspberry Pi foundation released two tools *raspivid* and *raspistill* to acquire a video or a still image respectivly. These tools the Multi-Media Abstraction Layer (MMAL) provided by VCHIQ to communicate with the Video Core and get the image from it.

E. Display Serial Interface (DSI)

The Raspberry Pi also provides a MIPI DSI interface, which is a standard used to connect displays on many smartphones. DSI consists of a clock lane and one or more data lanes. Each lane needs two wires for the differential signaling, though.

There is currently no DSI display available from the Raspberry Pi foundation and it is believed, that the VideoCore firmware must be updated to actually support the DSI interface. Other persons have tried to create DSI based displays for the Raspberry Pi, but have failed, because the VideoCore firmware is closed source [9].

F. Universal Asynchronous Receiver/Transmitter (UART)

The BCM2835 uses an IP-Core from ARM for the serial interface, which is called ARM AMBA PrimeCell PL011 UART. The chip has independent receive and transmit FIFOs to reduce interrupts, implements flow control in hardware and supports DMA. The chip also provides capabilities for IrDA communication [20].

The UART part of the chip is supported in the mainline kernel since many years, since its also used by other SoCs. The driver can be found in *drivers/tty/serial/amba-pl011.c*. The UART interface is exposed to the userspace using a device file called "/dev/ttyAMA0". It implements the POSIX interface for serial interfaces, which is described in the manpages *termios*, *fcntl* and *stdio*.

The Raspberry Pi's UART pins are available on the pin strip next to the 1.8V regulator and use a 3.3V voltage level.

The Raspberry Pi features a second UART, which is called mini UART. This one is a low throughput UART, which is intended to be used as a console.

Both UARTs are connected to the same pins, so they cannot be used at the same time.

G. Serial Peripheral Interface (SPI)

The BCM2835 contains two SPI controllers, but only one of them is available on the pin strip. Apart from that, the controllers support three chip select lines. Unfortunately only two are available on the pin strip [22, 4].

The first SPI controller is supported by the mainline kernel since 3.10. The kernel drivers can be found in *drivers/spi/spi-bcm2835.c*. The driver could be found in *drivers/spi/spi-bcm2708.c* before it was merged into the mainline kernel and some kernel trees still contain the old one.

Normally drivers for devices connected to the SPI bus are part of the Linux kernel, but the Kernel also exposes a limited interface to the userspace. This interface is described in *Documentation/spi/spidev* and is mainly intended for for prototyping, since bugs in userspace won't bring down the whole system.

Many SPI devices already have Linux drivers in the kernel, since they are used on other embedded systems. Those drivers can be found all over the kernel depending on the SPI device type (e.g. SPI real-time clocks can be found in drivers/rtc).

H. Inter-Integrated Circuit (I^2C)

I2C is a simple bus, that is much used by computer systems for connecting simple sensors (e.g. accelerometers) and actuators (e.g. RGB led controllers). The protocol is for example used to exchange data with monitors on Video Graphics Array (VGA), Digital Visual Interface (DVI) and High-Definition Multimedia Interface (HDMI) buses.

The BCM2835 contains three I2C controllers [4]. Two of those controllers are available on the Raspberry Pi's PCB. One on the GPIO pin strip and one on the unpopulated connector next to it [22].

Both controllers act as bus master and do not support multi-master configurations. The maximal supported bus speed is 400 kHz (fast-mode). The controllers supports default 7-bit addressing an 10-bit addressing format [4]. The controllers are also supposed to support clock stretching, but the implementation has a bug making this feature almost unusable [21].

For the kernel there are two drivers again. The mainline kernel has a driver since Linux 3.10, which can be found in *drivers/i2c/busses/i2c-bcm2835.c*. Previously the driver was found in *spi-bcm2708.c*. Similar to the SPI drivers, I2C device drivers are spread all over the kernel tree.

To access the I2C bus from userspace the *i2c-dev* kernel module must be loaded, which exports I2C buses as /dev/i2c-$num with an upcounting number for $num starting at 0. The API is documented in *Documentation/i2c/dev-interface* including some example code. Alternatively one can use the tools from the *i2c-tools* package on Debian/Raspbian/... to access i2c devices.

I. Direct Memory Access (DMA)

The BCM2835 DMA controller has 16 channels in total. Only the lower 13 channels have an associated IRQ. Some arbitrary channels are used by the Raspberry Pi's firmware (1,3,6,7 in the current firmware version). Also some channels

have special functionality and are not intended for direct use [18].

There is work going on to support the BCM2835's DMA controller in the Linux kernel. Once this has been merged into the kernel other drivers can simply use the kernel's DMA engine API to request DMA operations.

J. Mass Media Controller (MMC)

The BCM2835's external Mass Media Controller (eMMC) IP-Core is a slightly updated IP-Core from Arasan and supports normal SD card mode and SPI mode [4]. The mainline kernel's implementation for the interface can be foud in *drivers/mmc/host/sdhci-bcm2835.c* and currently does not make use of the IP-Core's DMA features.

The Raspberry Pi kernel from github has an alternative variant named *sdhci-bcm2708.c*, which only works with DMA. This driver does the DMA operations manually, which is no longer accepted for new drivers in the mainline kernel. Thus DMA support can only be introduced to the MMC driver after the BCM2835 DMA driver has been merged.

K. General Purpose Input/Output (GPIO)

The BCM2835 supports 54 GPIO lines split into two banks. All GPIO pins provide at least two alternative functions within the BCM and must be multiplexed correctly. The GPIO IP-Core also takes care of the pin muxing [4].

The Linux kernel support the Raspberry Pi's GPIO pins since 3.7 with the pinctrl-bcm2835 driver in *drivers/pinctrl/pinctrl-bcm2835.c*.

The GPIO's can be accessed from inside the kernel via the GPIO API described in *Documentation/gpio/consumer.txt*. It provides functions to acquire and release GPIOs, as well as setting or reading the GPIO lanes.

The GPIOs may also be accessed from the userspace via sysfs. The GPIO sysfs API is described in *Documentation/gpio/sysfs.txt* and supports exporting GPIO lines from userspace or from kernel space supporting additional aliases. Once exported and configured one can simply read the value of */sys/class/gpio/gpiochip/$number/value* for input GPIOs or write to the same file for output GPIOs.

L. Interrupts

The BCM2835 supports two different interrupt sources. On the one hand interrupts coming from the graphical processing unit (GPU) and on the other hand interrupts coming from other ARM control peripherals [4].

The interrupt controller is supported by the Linux kernel since 3.7 in *drivers/irqchip/irq-bcm2835.c*. Inside the kernel interrupt handling can be done using the methods defined in *include/linux/interrupt.h*. Those methods normally get a callback, which is called for each received interrupt.

The kernel does not support any interrupt handling in userspace, so everything in need of interrupts must be implemented as a kernel driver.

M. Audio

The BCM2835 contains a module, which converts audio packets into an Inter-IC Sound (I^2S) stream. This chip supports 1-2 channels per direction, has two separated FIFOs for receiving and transmitting data and can be driven using either polling, an interrupt based method or DMA [4]. The I^2S connection is available on the P5 header of the 2nd generation Raspberry Pi, so that one can connect his own audio codecs to it. Apart from that the Raspberry Pi contains its own codecs for the 3.5mm jack socket and the HDMI output [22].

There is work going on to support the BCM2835's digital audio module in the mainline kernel. This driver is using the DMA capabilities of the chip and thus depends on the BCM2835 DMA driver [17]. Once supported one can configure and use the audio parts of the Raspberry Pi via the Advanced Linux Sound Architecture (ALSA).

The Kernel from the Raspberry Pi foundation has full support for both the HDMI and the audio jack socket. The code is located in *sound/soc/arm/* and relevant files starts with bcm2835- prefixes. The driver basically sends audio data to the VideoCore, which then postprocesses it in its own firmware.

N. Pulse Width Modulator (PWM)

The BCM2835 also contains a PWM controller supporting two independent channels. The PWM module is currently neither supported by the mainline kernel nor by the Raspberry Pi Foundation's kernel, but there is work in progress to get the PWM module supported in the mainline kernel [23].

The driver will expose the PWM controller using the standard PWM API described in *Documentation/pwm.txt*, which provides a kernel internal API and a sysfs API for userspace access.

O. Random Number Generator (RNG)

Also part of the BCM2835 is a random number generator, which is already supported in the mainline kernel. The driver is part of the Linux kernel since 3.10 and is located in *drivers/char/hw_random/bcm2835-rng.c*.

The hardware RNG is automatically added as source for Linux's randomness pool, but will not be used exclusively for security purposes.

P. Watchdog

Watchdog modules are used to automatically reboot the system, if the module did not receive a ping within a specified time. They are commonly implemented on embedded systems to avoid devices being broken for a longer time period.

The BCM2835 contains a simple watchdog module, which can be configured to to reboot the system if the module is not updated within a specified time between 1 and 15 seconds [19].

The Linux kernel exposes the watchdog as */dev/watchdog* and disables it by default. The watchdog can be enabled, configured and updated from the mentioned device file. There exists a daemon called watchdog, which handles the watchdog device. The Linux kernel intentionally moves the actual work

of updating the Watchdog to the userspace to detect broken userspace scenarios with working kernels.

The API for the kernel device is documented in *Documentation/watchdog/watchdog-api.txt*.

Q. Timer

The BCM2835 System Timer features a 64-bit free-running counter that runs at 1 MHz and has four separate output compare registers that can be used to schedule interrupts. However, two output compare registers are already used by the VideoCore GPU, leaving only two available for the ARM CPU to use [4].

The Linux driver is part of the mainline kernel since 3.7 and is implemented in *drivers/clocksource/bcm2835_timer.c*. The drivers registers the module as Linux clocksource, thus it can be used by the Linux kernel for any timing related work. The currently used clocksource can be seen via the sysfs file */sys/devices/system/clocksource/clocksource0/current_clocksource*. The selected clock source will be used by the Linux kernel for POSIX functions like *clock_gettime()*.

R. Universal Serial Bus (USB)

Also part of the BCM2835 is a DesignWare USB2 controller called DWC2, which supports USB Host and USB Peripheral mode.

The mainline Linux kernel supports the DWC2 controller since 3.10. Currently the driver does not support the modified version of the BCM2835, but patches have been queued for 3.14 adding that minimal support [24] for the Raspberry Pi to this driver. The driver, which only supports USB Host mode, can currently be found in *drivers/staging/dwc2*, but will be moved to *drivers/usb/dwc2* once it has been cleaned up.

Apart from that the Raspberry Pi Foundation's kernel has some additional driver quirks to reduce the number of interrupts generated by the USB controller, which are Raspberry Pi specific. There is still work going on to integrate this into the mainline driver.

S. Thermal Sensor

The BCM2835 also has a thermal sensor, which is connected to the Video Core. There is currently work in progress to get this sensor supported via the standard Linux Thermal Framework [16]. On the Raspberry Pi Foundations System one can use the tool *vcgencmd measure_temp* instead.

IV. USB Ethernet: SMSC LAN9512

The BCM2835 does not come with an Ethernet IP-Core. The Raspberry Pi still has support for 10/100 Ethernet using an USB Ethernet chip from smsc called LAN9512. This chip provides a 10/100 Ethernet media access controller (MAC) with full-duplex support and two-downstream USB ports [10].

The Ethernet MAC supports Auto-negotiation, Auto-MDIX, automatic 32-bit CRC generation and checking and TCP/UDP checksum offload support. Also supported are numerous power management wakeup features, including the common "Magic Packet". The USB side supports advanced power saving features including full power mangement for each downstream port [10].

The USB controller is supported in Linux since a long time ago and its driver can be found in *drivers/net/usb/smsc95xx.c*.

```
/dts-v1/;

/ {
    compatible = "raspberrypi,model-b",
                 "brcm,bcm2835";
    model = "Raspberry Pi Model B";

    soc {
        /* ... */

        watchdog {
            compatible = "brcm,bcm2835-pm-wdt";
            reg = <0x7e100000 0x28>;
        };

        i2c0: i2c@20205000 {
            compatible = "brcm,bcm2835-i2c";
            reg = <0x7e205000 0x1000>;
            interrupts = <2 21>;
            clocks = <&clk_i2c>;
            clock-frequency = <100000>;
            #address-cells = <1>;
            #size-cells = <0>;
        };
    };

    /* ... */
};
```

Listing 1. Shortened Device Tree source file for the Raspberry Pi [19]

V. Device Tree

Most hardware components on embedded devices cannot be detected and/or identified automatically. On common x86 based systems the information about connected components, that cannot be autodetected are stored inside the BIOS, but no such component exists on ARM devices. Therefore the kernel had board specific code for each supported ARM-based board. For example there were over 20000 lines of code for all boards using Texas Instrument's SoCs.

A technique called Device Tree has been introduced to the ARM part of the Linux kernel to remove all those board specific code from the kernel. Device Tree is a way to describe properties of hardware components and how they are connected to the remaining system in a hierarchical way. It has been chosen, since the kernel code already contained a parser for the data, which has been used by the PowerPC architecture (PowerPC's Open Firmware uses this format) [11, 7].

Converting the boardfiles and all involved drivers to support Device Tree is still an ongoing process. Nowadays kernel maintainers typically reject new ARM related drivers, which do not support Device Tree. The Raspberry Pi support has been prepared using boardcode in a custom kernel tree and work is going on to add Device Tree support to them, so that they can be included with the mainline kernel.

There are two ways to boot a Device Tree using kernel. One way is to configure the kernel for an appended Device

Tree file. In this case one can simply append the Device Tree file to the kernel. This is the recommended way if native Device Tree support is missing in the bootloader. Modern bootloaders are supposed to handle the Device Tree file similar to initramfs files: The bootloader first loads the Device Tree file into memory and provides the kernel a memory address.

To avoid big delays in the kernel boot process the Device Tree file parsed by the kernel is compiled into a binary format. The compilation is done by a tool called Device Tree Compiler (dtc), which is bundled with the kernel and available as standalone binary in most Linux distributions.

Linux's Device Tree source files for ARM are available in *arch/arm/boot/dts* and are normally prefixed by the platform's SoC. The Raspberry Pi's Device Tree source file can be found at *arch/arm/boot/dts/bcm2835-rpi-b.dts*. A simplified excerpt of the file can be seen in Listing 1.

Every Device Tree file contains a root node, which stands for the whole board. It contains an arbitrary number of subnodes, which may have subnodes again. Each node stands for one hardware component and each subnode is supposed to be reachable via its parent node.

The kernel uses the value of a node's compatible property to determine, which driver should be used. There are some other standardized properties, e.g. *reg*, *interrupts* and *clocks*, which share the same meaning across nodes. Other properties are vendor specific. These properties normally carry a vendor prefix (e.g. brcm). The Device Tree API for each driver is documented in *Documentation/devicetree/bindings* and is supposed to be usable by other, non-Linux based kernels and bootloaders.

The Device Tree also brings advantages for experienced users, who want to extend their hardware without patching and compiling the Linux kernel. Let's assume the user has a kernel with lots of hardware support available via kernel modules and want to extend his Raspberry Pi by a real-time clock (RTC). For conventional kernels this means patching the boardfile to include information about the RTC and then compiling the kernel.

For Device Tree using kernels its enough to update the Device Tree source file by adding the content of Listing 2 at the bottom and recompiling the Device Tree source file. The kernel and Debian based userlands will automatically pick up the RTC and use it appropriately.

```
&i2c0 {
    rtc: rtc@68 {
        compatible = "dallas,ds1338";
        reg = <0x68>;
    };
};
```

Listing 2. RTC for the Raspberry Pi

VI. Model Differences

Basically there are currently three different models on the market. On the one hand there is the Raspberry Pi Model A, which is lacking the SMSC LAN9512 USB Ethernet controller. Thus it is missing an Ethernet Port and has only one USB port. There has only been one revision of this model so far, which has 256MB memory.

On the other hand there is Model B, which has the additional SMSC LAN9512 chip. There exist two revisions of this Model, which differ in the amount of installed memory (Revision 1 has 256MB, Revision 2 has 512MB) and the position of pins on the board. The pin position has been changed slightly for Revision 2 to support the connection of external audio codecs to the I^2S interface.

VII. Conclusion

The Raspberry Pi is basically a breakout board for the BCM2835 system on a chip with additional Ethernet support. Despite the hype it is a more closed platform than many other things available on the market (e.g. the Pandaboard). The competition also has the advantage of higher performance, more modern ARM cores (ARMv7 instead of ARMv6) and more peripheral hardware.

The Raspberry Pi's advantage lies in its big community and its very low price. The Raspberry Pi is a nice system to introduce people to the Linux Operating System without changing their computer and as entry-level embedded system. It's useful for misc. automation-tasks, especially if they are supposed to be available via Ethernet, since the Raspberry Pi provides one of the cheapest solutions for an Ethernet-capable development board. Apart from that it's H.264 playback capability makes it a popular system for home entertainment systems running software like e.g. XBMC.

References

[1] Dave Airlie. *raspberry pi drivers are NOT useful*. Oct. 2012. URL: http://airlied.livejournal.com/76383.html.

[2] *ARM1176JZF-S TRM*. r0p7. ARM Ltd. Nov. 2009. URL: http://arm.com/products/processors/classic/arm11/arm1176.php.

[3] *BCM2708 vs. BCM2835*. URL: https://github.com/raspberrypi/linux/issues/22.

[4] *BCM2835 ARM Peripherals*. Broadcom. Feb. 2012. URL: http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf.

[5] Raspberry Pi Foundation. *FAQs*. URL: http://www.raspberrypi.org/faqs.

[6] Herman Hermitage and Scott Mansell. *VideoCore IV Programmers Manual*. URL: https://github.com/hermanhermitage/videocoreiv/wiki/VideoCore-IV-Programmers-Manual.

[7] IEEE. *IEEE-1275 Standard for Boot (OpenFirmware)*. 1994.

[8] *Jazelle Information*. URL: http://www.arm.com/products/processors/technologies/jazelle.php.

[9] Dr. Ace Jeangle. *Long awaited RaspberryPi LCD solution is here!* Aug. 2012. URL: http://www.chalk-elec.com/?p=1553.

[10] *LAN9512/LAN9512i datasheet*. 1.0. SMSC. Nov. 2009. URL: http://www.smsc.com/Downloads/SMSC/Downloads_Public/Data_Sheets/9512.pdf.

[11] Grant Likely. *Device Tree Website*. URL: http://devicetree.org/.
[12] *Linux Kernel for RPi*. URL: https://github.com/raspberrypi/linux.
[13] *Linux Kernel Mailinglist*. URL: https://lkml.org.
[14] Alphamosaic Ltd. *VideoCore®*. Feb. 2003. URL: http://web.archive.org/web/20030209213838/www.alphamosaic.com/videocore/.
[15] Craig McGeachie. *Enable BCM2835 mailbox support*. URL: http://www.spinics.net/lists/arm-kernel/msg273035.html.
[16] Craig McGeachie. *Implement BCM2835 CPU thermal sensor support*. Sept. 2013. URL: http://www.spinics.net/lists/arm-kernel/msg273037.html.
[17] Florian Meier. *ASoC: Add support for BCM2835*. Nov. 2013. URL: https://lwn.net/Articles/574650/.
[18] Florian Meier. *[PATCHv9] dmaengine: Add support for BCM2835*. Jan. 2014. URL: https://lkml.org/lkml/2014/1/2/317.
[19] *Official Linux Kernel*. URL: http://kernel.org.
[20] *PrimeCell UART (PL011) Technical Reference Manual*. r1p4. ARM Ltd. Nov. 2005. URL: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0183f/DDI0183.pdf.
[21] *Raspberry Pi I2C clock-stretching bug*. URL: http://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html.
[22] *Raspberry Pi Revision B Schematics*. 2.0. Raspberry Pi Foundation. Oct. 2012. URL: http://www.raspberrypi.org/archives/2233.
[23] Johannes Thumshirn. *[RFC] PWM: Add support for pwm-bcm2835*. Sept. 2013. URL: http://lists.infradead.org/pipermail/linux-rpi-kernel/2013-September/000701.html.
[24] Stephen Warren. *[PATCH] staging: dwc2: set up all module params*. Nov. 2013. URL: http://www.spinics.net/lists/devicetree/msg13421.html.

Raspberry Pis GPIO
Überblick, Ein- und Ausgaben sowie aufsetzende Kommunikationsschnittstellen

Simon Kurka
Carl von Ossietzky Universität Oldenburg
Department für Informatik
26111 Oldenburg
Seminar Raspberry Pi inspected

***Zusammenfassung*—Der Raspberry Pi ist ein kleiner Einplatinencomputer, der ursprünglich für die Lehre entwickelt wurde. Das Gerät bietet mehrere Anschlüsse mit Pins, die sich frei als Ein- oder Ausgang konfigurieren lassen. Die Pins können zur Einbettung in umfangreichere Systeme genutzt werden, wie z. B. in der Automatisierungs- und Steuerungstechnik. Per Software können die Pins mit einem Spannungspegel (logisch 1) oder Masse (logisch 0) belegt werden. Ebenso kann per Software von Pins gelesen werden. Der Raspberry Pi bietet außerdem verschiedene etablierte Schnittstellen als Sonderfunktionen bestimmter Pins an. Darunter finden sich SPI, UART und I²C. Diese Schnittstellen dienen alle der seriellen Kommunikation, bieten aber unterschiedliche Funktionen und nutzen dazu verschieden viele Pins. Diese Ausarbeitung vermittelt ein Grundverständnis über die GPIO-Pins des Raspberry Pi, ihrer jeweiligen Funktionen und deren Anwendung.**

***Index Terms*—Raspberry Pi, GPIO, UART, SPI, I²C**

I. Einleitung

Der Raspberry Pi, ein kleiner Einplatinencomputer, der ursprünglich für die Lehre entwickelt wurde, bietet offene Pins. Viele dieser Pins können per Software frei als Ein- bzw. Ausgang definiert werden. Bestimmte Pins bieten Sonderfunktionen für eine serielle Kommunikation an. Zunächst wird grundlegend auf die Pins und deren einfachste Nutzung eingegangen. Anschließend werden die seriellen Schnittstellen SPI, UART und I²C, die als Sonderfunktionen bestimmter Pins realisiert sind, näher behandelt. Abschließend werden die Besonderheiten der einzelnen Schnittstellen fokussiert, verbunden mit einem Ausblick bzgl. der Zukunft der seriellen Kommunikation im Allgemeinen.

II. GPIO

Neben üblichen Anschlüssen, wie Ethernet-Anschluss, USB-Ports und 3,5mm-Klinke, verfügt der Raspberry Pi über Pins. Während einige Pins der Versorgung mit 3,3V, 5V oder Masse dienen, können General-Purpose-Input-Output (GPIO)-Pins auch Informationen annehmen und ausgeben. Im folgenden Abschnitt wird auf die verschiedenen Pins eingegangen.

A. Überblick

Pins sind normalerweise in Gruppen angeordnet. Diese Gruppen bilden einen Pin-Header bzw. Anschluss. Beim Raspberry Pi wird oft nur auf die GPIO-Pins des P1-Headers eingegangen. In den folgenden Ausführungen wird auch auf die anderen Pin-Header eingegangen.

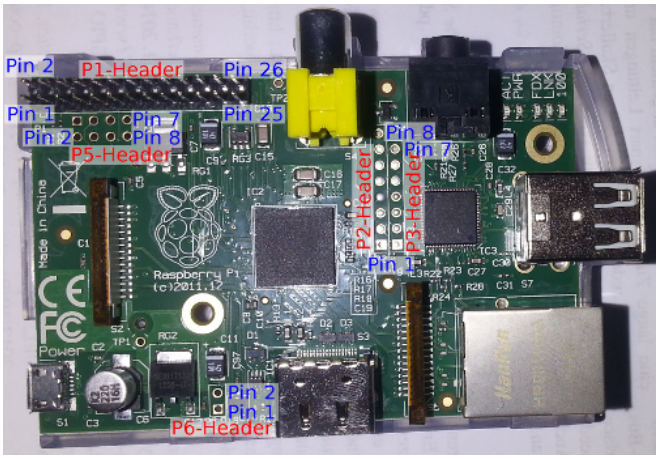


Abbildung 1. Foto des Raspberry Pi mit beschrifteten Pin-Headern. [13]

Der P1-Header ist vollständig mit 26 Pins bestückt und seit der ersten Hardwareversion des Raspberry Pi vorhanden. Neben dem P1-Header gibt es weitere Header, die nicht ab Werk mit Pins bestückt sind.

Der P2-Header hat 8 Pins. Der Header wird nur während der Produktion benutzt, es handelt sich um eine JTAG Schnittstelle zur GPU. JTAG steht für „Join Test Action Group“. Es bezeichnet einen Standard, der Verfahren zum Testen und Debuggen elektronischer Hardware beschreibt. Auf JTAG soll hier nicht weiter eingegangen werden. Für den Anwender sind Pin 1 als 3,3V Spannungsquelle sowie Pin 7 und 8 als Masse nutzbar. Pin 1 des P2- und Pin 1 des P1-Headers werden über die gleiche Leitung versorgt und unterliegen damit gemeinsam denselben Einschränkungen, auf die im Unterabschnitt II-B näher eingegangen wird. Alle anderen Pins des P2-Headers sind ab der Hardwarerevision 2.0[1] nicht mehr nutzbar.

Auch der P3-Header hat 8 Pins und dient als JTAG Schnittstelle für den LAN-Port. Diese wird nur während der Produk-

[1]Die Auslieferung der Hardwarerevision 2.0 begann schrittweise im September 2012 [13].

tion zur Qualitätssicherung verwendet. Für den Anwender ist ausschließlich Pin 7 als Masse nutzbar.

Der P4-Header wird nur selten genannt. Es handelt sich hierbei um den Ethernet-Anschluss beim Modell B des Raspberry Pi [12].

Der P5-Header wurde mit der Hardwarerevision 2.0 eingeführt, er bietet 8 Pins. Einige Pins können für spezielle Funktionen eingesetzt werden, wie z. B. einen zweiten I^2C Kanal (auf I^2C wird im Abschnitt VI näher eingegangen). Während alle anderen Pin-Header vorzugsweise von oben bestückt werden, kann der P5-Header vorzugsweise von unten bestückt werden.

Der P6-Header verfügt über 2 Pins, Pin 2 ist auf Masse gelegt. Der P6-Header wurde eingeführt, um den Raspberry Pi nach einem Herunterfahren wieder zu starten bzw. einen Reset auszulösen. Zu diesem Zweck müssen die beiden Pins des P6-Headers verbunden werden. [13]

B. Technische Angaben

Die Pins haben einen Abstand von 2,54mm. Die P1- und P5-Header sind jeweils zweireihig, die P2-, P3- und P6-Header einreihig verbaut. Die 3,3V-Pins für die Spannungsversorgung verwenden gemeinsam eine Leitung [11] und liefern daher gemeinsam einen maximalen Strom von 50mA [13]. Die 5V-Pins werden direkt von der Spannungsversorgung des Raspberry Pi gespeist [11]. Die GPIO-Pins arbeiten im Bereich von 0V bis 3,3V. Die Pins sind nicht gegen Überspannung geschützt, der Anschluss höherer Spannungen kann den Raspberry Pi zerstören [13]. Zum Schutz vor Überspannung existieren Standardschaltungen, die beim Raspberry Pi extern realisiert werden müssen.

C. Einfache Ein- und Ausgaben

Für einfache Ein- und Ausgaben stehen Bibliotheken für verschiedene Programmiersprachen zur Verfügung. Über die Shell wird deutlich, dass die GPIO-Pins direkt in das Dateisystem eingebettet werden. So lassen sich durch Beschreiben von bestimmten Dateien in `/sys/class/gpio/` die einzelnen GPIO-Pins aktivieren, als Input bzw. Output festlegen und ggf. mit Werten belegen.

Die GPIO-Pins werden durch die Nummer des GPIO-Pins referenziert. Die Nummern werden nicht durch den GPIO-Header, sondern durch den Prozessor (BCM2835) des Raspberry Pi vorgegeben [13]. Welche Nummer auf welchen Pin geschaltet ist, muss bei Bedarf geprüft werden.

Vor Verwendung muss ein Pin aktiviert werden. Auf der Shell wird dazu die Nummer des Pins in `/sys/class/gpio/export` geschrieben. Anschließend kann der Pin als Ein- bzw. Ausgang definiert werden, indem „in“ bzw. „out“ in `/sys/class/gpio/gpio<Pin Nummer>/direction` geschrieben wird. Werte lassen sich über die Datei `/sys/class/gpio/gpio<Pin Nummer>/value` lesen bzw. schreiben. Deaktivieren lässt sich ein Pin analog zum Aktivieren über die Datei `/sys/class/gpio/unexport` [13].

Eine besondere Bedeutung hat die Programmiersprache Python für den Raspberry Pi. Die Raspberry Pi Foundation hat diese als offizielle Lehrsprache für den Raspberry Pi ausgewählt, welches ursprünglich für Bildungszwecke entwickelt wurde. Python ist eine „moderne, verbreitete, vielfältig einsetzbare und für alle gängigen Plattformen verfügbare Sprache“ und damit „eine naheliegende und gute Wahl“ [8]. Die Bibliothek `RPi.GPIO` ist im Betriebssystem Raspbian standardmäßig installiert. Mit `GPIO.setmode(GPIO.BOARD)` wird die Nummernkonvention des P1-Headers verwendet; `GPIO.setmode(GPIO.BCM)` verwendet die Nummernkonvention des Prozessors. `GPIO.setup(11, GPIO.IN)` bzw. `GPIO.setup(12, GPIO.OUT)` legen den entsprechenden Pin als Ein- bzw. Ausgang fest. Mit `input_value = GPIO.input(11)` wird von Pin 11 gelesen, mit `GPIO.output(12, GPIO.HIGH)` Pin 12 auf HIGH gesetzt [13].

```
# Set up GPIO 4 and set to output
echo "4" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio4/direction

# Set up GPIO 7 and set to input
echo "7" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio7/direction

# Write output
echo "1" > /sys/class/gpio/gpio4/value

# Read from input
cat /sys/class/gpio/gpio7/value

# Clean up
echo "4" > /sys/class/gpio/unexport
echo "7" > /sys/class/gpio/unexport
```

Abbildung 2. Beispiel einer GPIO-Nutzung über die Shell. Entnommen aus [13]

```
import RPi.GPIO as GPIO

# use P1 header pin numbering convention
GPIO.setmode(GPIO.BOARD)

# Set up the GPIO channels - one input
# and one output
GPIO.setup(11, GPIO.IN)
GPIO.setup(12, GPIO.OUT)

# Input from pin 11
input_value = GPIO.input(11)

# Output to pin 12
GPIO.output(12, GPIO.HIGH)

# The same script as above but using
# BCM GPIO 00..nn numbers
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN)
GPIO.setup(18, GPIO.OUT)
input_value = GPIO.input(17)
GPIO.output(18, GPIO.HIGH)
```

Abbildung 3. Beispiel einer GPIO-Nutzung über Python. Entnommen aus [13]

In [13] sind für viele Programmiersprachen ausführliche Beispiele zu finden, wie der Anwender mit den GPIO-Pins interagieren kann.

Auf Basis dieser einfachen Ein- und Ausgaben lassen sich, mit dem Raspberry Pi als Steuereinheit, sehr komplexe Schaltungen aufbauen. Diese können durch eine Netzwerkanbindung des Raspberry Pi sogar über ein Webinterface von außen gesteuert werden [10].

III. Serielle Kommunikation

Nach Darstellung der elementaren Nutzung der Pins, werden im Folgenden mehrere serielle Schnittstellen für eine weiterreichende Kommunikation behandelt. Bevor die einzelnen Schnittstellen SPI, UART und I^2C näher betrachtet werden, wird auf Vorteile serieller Schnittstellen allgemein eingegangen.

Die seriellen Schnittstellen bieten eine einfache und einheitliche Kommunikation zwischen unterschiedlichen Komponenten. Mikrocontroller, LEDs, LCDs, Speicher, Echtzeituhren und verschiedenste Sensoren sind fast in jedem elektronischem Produkt zu finden. Mittels standardisierter Schnittstellen lassen sich verschiedenste Komponenten miteinander verbinden. Durch die vereinheitlichte Kommunikation können auch Komponenten unterschiedlicher Hersteller kommunizieren. Die Einzelkomponenten sind mehrfach einsetzbar, da nicht für jedes elektronische Endprodukt eigene Kommunikationswege entwickelt werden. Wiederverwendbare Einzelkomponenten (wie die bereits genannten) können günstig in Masse produziert, in verschiedenen Endprodukten verbaut und ohne funktionale Einbußen verwendet werden.

IV. Serial Peripheral Interface Bus (SPI)

Der Serial Peripheral Interface Bus (SPI oder auch Microwire genannt) ist ein Bus-System für serielle synchrone Datenübertragungen. Es wurde ursprünglich von Motorola etwa im Jahr 1979 für die Anbindung von Peripherie an Microcontroller entwickelt [6]. Im folgenden Abschnitt soll dieses System näher betrachtet werden.

A. Konzept

Das Serial Peripheral Interface wird für eine serielle synchrone Datenübertragung zwischen mehreren Slaves und einem Master verwendet. „Immer öfter werden serielle Bussysteme statt parallelen wegen der einfacheren Verkabelung eingesetzt.“ [17] SPI verwendet prinzipiell zwei Datenleitungen und mehrere Steuerleitungen, davon eine Taktleitung. Es ist damit vollduplexfähig und lässt unterschiedliche Taktfrequenzen zu [17]. Abbildung 4 zeigt ein Beispiel einer möglichen Verdrahtung von SPI-Elementen. Mit SPI ist es möglich einen oder mehrere Slaves anzusprechen. Sollen bestimmte Slaves ausschließlich gemeinsam angesprochen werden, so lassen sich diese durch die Verdrahtung statisch zu einer Gruppe zusammenfassen. Bestimmte Slaves können einzeln oder gemeinsam angesprochen werden, sodass der Master die Gruppen dynamisch bilden kann. Auch die Hintereinanderschaltung von Slaves ist möglich (siehe Abbildung 5), um z. B. einen größeren Funktionsblock, wie einen Speicher, zu erhalten. Die Möglichkeiten werden jeweils durch die Verdrahtung der Teilnehmer bestimmt.

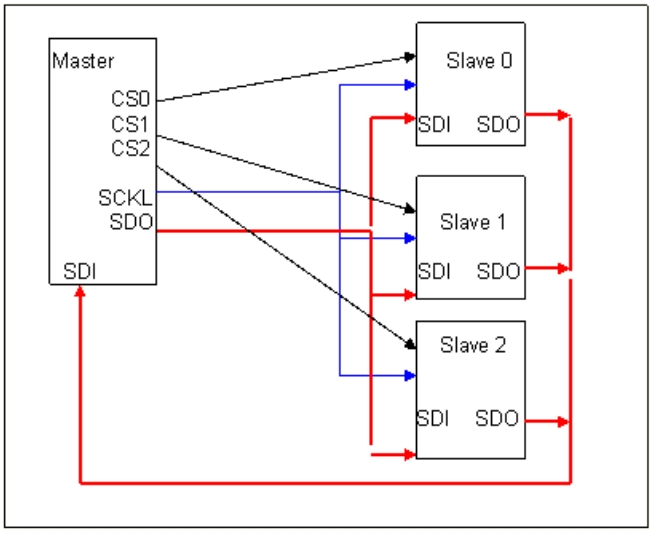


Abbildung 4. Beispiel einer SPI-Schaltung mit vielen Steuerleitungen. Entnommen aus [17]

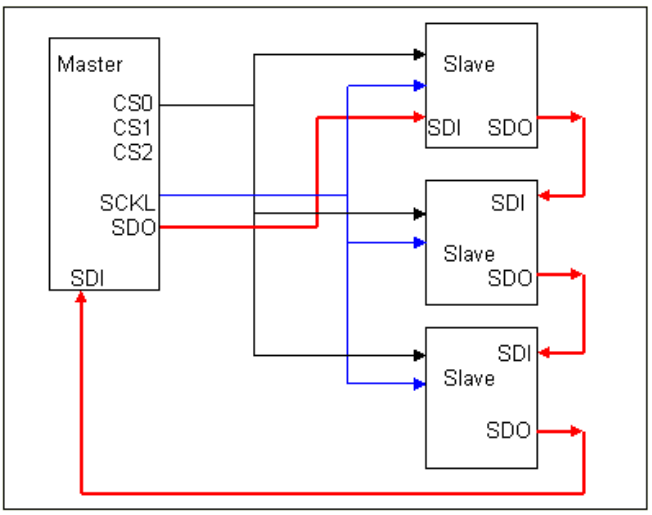


Abbildung 5. Beispiel einer Hintereinanderschaltung von SPI-Slaves. Entnommen aus [17]

B. Protokoll

Obwohl es von Motorola keine bestimmte Spezifikation bzgl. gültiger Betriebsarten gibt, haben sich vier Betriebsarten durchgesetzt, die durch Clock Polarität (CPOL) und Clock Phase (CPHA) bestimmt werden. Der Wert von CPOL gibt den Idle[2]-Zustand der Clock an. CPHA gibt an, ob die Daten bei der ersten oder zweiten Flanke übernommen werden sollen. Mit `CPOL=0` und `CPHA=0` werden die Daten also mit der ersten Flanke, die eine steigende ist, übernommen. Nicht selektierte Slaves müssen ihren Ausgang in einen hochohmigen Zustand schalten, um Konflikte zu verhindern. Üblich, aber nicht zwingend, sind byteweise Übertragungen. Da mit jedem Takt nur ein Bit übertragen werden kann, sind acht Takte nötig um ein Byte vollständig zu übertragen [17].

[2]Leerlauf- oder Normalzustand, wenn keine Aktivität auf dem Bus statt findet

Der Raspberry Pi implementiert neben dem hier dargestellten Standard-Modus zwei weitere: Im bidirektionalen Modus wird nur eine Datenleitung verwendet. Diese wird sowohl für ausgehende, als auch für eingehende Datenübertragungen verwendet. Im LoSSI Modus (Low Speed Serial Interface) werden Übertragungen von Befehlen und Daten unterschieden. Dazu wird das 8 Bit Wort um ein Bit erweitert. Ist das neunte Bit auf LOW gesetzt, ist es ein Befehl, HIGH steht für Daten [14].

C. Hardware

SPI setzt normalerweise mindestens vier Leitungen ein: eine Taktleitung (SCKL) zum Synchronisieren, zwei Datenleitungen für Slave-Data-In (SDI) und Slave-Data-Out (SDO) bzw. Master-Out-Slave-In (MOSI) und Master-In-Slave-Out (MISO) sowie eine oder mehrere Chipselect (CS) bzw. Slave-Select (SS), um die angeschlossenen Slaves vom Master aktivieren zu lassen. Abwandlungen sind dabei nicht ausgeschlossen, z. B. haben Sensoren nicht unbedingt einen Dateneingang. „Streng genommen kann man solche Bauelemente nicht zu den SPI-fähigen zählen, da sie sich sonst aber genau so verhalten[,] tut man es dennoch.“ [17]

SPI wird direkt vom Raspberry Pi unterstützt. Es muss vor Verwendung allerdings aktiviert werden, indem das Kernel-Modul „spi-bcm2708“ von der Blacklist in der Datei `/etc/modprobe.d/raspi-blacklist.conf` im Betriebssystem „Raspbian“ entfernt wird. Zu diesem Zweck ist die betreffende Zeile zu entfernen oder durch ein vorangestelltes #-Zeichen auszukommentieren. Danach arbeiten fünf GPIO-Pins des P1-Headers SPI-konform. Pin 19 dient als MOSI, Pin 21 als MISO, Pin 23 als SCLK, Pin 24 als CS0 und Pin 26 als CS1 [13].

D. Anwendung

Die Anwendungen sind vielfältig: Digital-Analog- und Analog-Digital-Wandler gibt es mit verschiedensten Auflösungen und Taktfrequenzen. Auch Speicherbausteine lassen sich über SPI anbinden. Die meisten AVR-Microcontroller von Atmel. z. B. auch der ATMEGA8, werden über SPI programmiert [3].

V. Universal Asynchronous Receiver and Transmitter Interface (UART)

Das Universal Asynchronous Receiver and Transmitter Interface (UART) bietet eine Möglichkeit zur seriellen asynchronen Kommunikation. UART ist bei elektronischen Bauelementen weit verbreitet [18]. Auch der Raspberry Pi lässt sich per UART ansprechen. So kann man bei einer Standardinstallation des Systems „Raspbian“ über UART direkt auf die Shell zugreifen. UART ist zudem die einzige Schnittstelle, die standardmäßig aktiviert ist. Aufgrund der weiten Verbreitung, der Besonderheit des Verzichts auf eine Taktleitung und der Unterstützung durch den Raspberry Pi soll UART hier näher behandelt werden.

A. Konzept

Das Konzept hinter UART ist bereits mehr als 100 Jahre alt [7]. Die Grundidee von UART, und damit der seriellen Kommunikation, stammt aus der Fernschreibertechnik. Dort wurden Daten durch Unterbrechungen in einem sonst geschlossenen Stromkreis übertragen. UART selbst bezeichnet eigentlich nur einen Interface-Baustein oder einen Funktionsblock, der einem Prozessor eine parallele Kommunikation anbietet, diese aber nach außen seriell umsetzt. Die serielle Kommunikation dient dazu, die Anzahl der benötigten physikalischen Leitungen zu reduzieren. [7] Die Besonderheit an UART, im Vergleich zu den anderen hier vorgestellten Schnittstellen, ist, dass UART die gesamte Kommunikation ohne eine eigene Taktleitung abwickelt. Näheres dazu ist im Unterabschnitt V-C zu finden.

B. Protokoll

UART verwendet einen festen Rahmen für die Datenübertragung. Dieser besteht aus Start- und Stop-Bit, fünf bis maximal neun Datenbits und einem optionalen Paritätsbit. Abbildung 6 veranschaulicht den Rahmen. Der genaue Rahmen, d. h. die Anzahl der Datenbits und die Information, ob ein Paritätsbit gesendet wird, sowie die Taktfrequenz müssen vorher festgelegt werden. Es handelt sich also explizit um ein Vorwissen, das bereits vor der Datenübertragung gegeben sein muss. Abbildung 7 zeigt typische Baud-Raten (Taktfrequenzen), welche hier der Bit-Rate entsprechen. Jede Übertragung wird unabhängig von einem festen Takt durch ein Start-Bit eröffnet und durch ein Stop-Bit beendet. Start-Bit und Stop-Bit ermöglichen eine Übertragung zu jedem beliebigen Zeitpunkt. Dies ist der Grund dafür, dass die Übertragung mittels UART als „asynchron“ bezeichnet wird. Der Beginn einer Übertragung ist nicht an einen festen, durchgehenden Takt gebunden. [7]

Abbildung 6. Beispiel eines UART Rahmens. Entnommen aus [7]

Baud = Bits/s	Baud = Bits/s	Baud = Bits/s	Baud = Bits/s
110	1.200	19.200	76.800
150	2.400	28.800	115.200
300	4.800	38.400	
600	9.600	57.600	

Abbildung 7. Typische Baud-Raten bei der seriellen Kommunikation. Entnommen aus [1]

C. Hardware

UART benutzt zwei Datenleitungen, eine zum Senden und eine zum Empfangen. UART ist damit vollduplexfähig. Auf eine Taktleitung wird verzichtet. Statt dessen verwenden die Kommunikationspartner Oszillatoren, um die Datenleitungen

in bestimmten Intervallen abzutasten bzw. mit einem neuen Wert zu belegen. Die Oszillatoren sind unterschiedlich genau, sodass es bei Übertragung längerer Bitfolgen zu Fehlern kommt. Um diesem Problem entgegenzuwirken, synchronisiert sich der Empfänger mit jedem Zeichen am Start- und Stop-Bit erneut. [7]

D. Anwendung

Die wichtigsten Anwendungen finden sich in den Schnittstellen RS232 und RS485 [19]. Allerdings definiert RS232 -3V bis -12V als LOW und +3V bis +12V als HIGH [5]. Der Raspberry Pi hat einen Arbeitsbereich von 0V bis 3,3V und könnte bei einer direkten Verbindung mit UART zerstört werden. Um dennoch eine fast direkte Verbindung zu ermöglichen, lassen sich Pegelwandler einsetzen. Logische Komponenten oder gar Prozessoren sind nicht zwingend notwendig, da die Protokolle dieser Schnittstellen identisch sind. Abbildung 8 zeigt ein Beispiel für einen Pegelwandler zwischen den Arbeitsbereichen 0V bis 3,3V und 0V bis 5V (z. B. für Microcontroller).

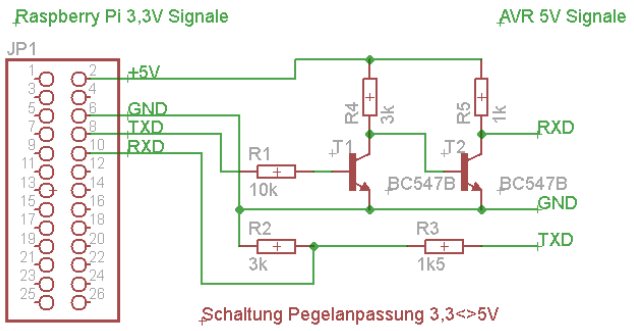


Abbildung 8. Beispiel eines Pegelwandlers. Entnommen aus [10]

Auch viele eingebettete Systeme, wie beispielsweise Router, bieten eine UART-Schnittstelle an. Über UART lassen sich die Systeme oft selbst dann noch ansprechen, wenn alle anderen Möglichkeiten, wie z. B. üblicherweise ein Webinterface, nicht mehr erreichbar sind. Dies liegt daran, dass sich das System mittels UART auf sehr einfache und direkte Weise ansprechen lässt, ohne auf komplexe Software angewiesen zu sein.

VI. Inter-Integrated Circuit Interface (I^2C)

Das Inter-Integrated Circuit Interface (I^2C) beschreibt ein von Philips Semiconductors (jetzt NXP Semiconductors) entwickeltes Bus-System, das auf zwei Leitungen basiert [16]. I^2C wird vom Raspberry Pi zweifach unterstützt. Die geringe Anzahl von zwei Leitungen sowie die zweifache Unterstützung durch den Raspberry Pi machen I^2C für diese Ausarbeitung interessant. Daher soll I^2C an dieser Stelle näher behandelt werden.

A. Konzept

I^2C wurde entwickelt, um alle Vorteile serieller Kommunikation zu nutzen. Die Besonderheit an I^2C ist, dass es nur zwei Leitungen benutzt und damit sehr einfache Schaltungen ermöglicht. Weiterhin ist jeder Teilnehmer per Software über eine Adresse ansprechbar, was ein hohes Maß an Skalierbarkeit schafft. I^2C sieht einen Master-Slave-Betrieb vor und ermöglicht sogar Multi-Master-Systeme mit eindeutiger Konflikterkennung und -behebung. [16] Ausgelegt ist I^2C für kurze Strecken bis höchstens 5,5m [2] zwischen interagierenden Schaltungen.

B. Protokoll

Das I^2C Protokoll sieht für jeden Bus-Teilnehmer (Bus beschreibt einen Übertragungsweg zwischen mehreren Teilnehmern) eine eindeutige Adresse vor. Diese werden zum Ansprechen des Kommunikationspartners verwendet. Die I^2C-Terminologie sieht grundsätzlich vier Rollen vor. Der Transmitter ist das Gerät, das Daten über den Bus sendet, der Receiver jenes, das Daten über den Bus empfängt. Außerdem gibt es Master und Slaves. Der Master beginnt und beendet eine Übertragung und gibt den Takt an. Der Slave muss von einem Master adressiert werden, um aktiv werden zu dürfen. Der Idle-Zustand beider Leitungen ist HIGH. Abbildung 9 zeigt abstrakt einen typischen Ablauf einer Übertragung mittels I^2C. In jedem Fall beginnt der Master die Übertragung und sendet die Adresse. Im Folgenden entscheidet sich durch das R/W-Bit, ob der Slave die Daten sendet und der Master bestätigt (Master-Receiver) oder der Master die Daten sendet und der Slave bestätigt (Master-Transmitter). Das erste ACK wird in jedem Fall von dem Slave gesendet, um dem Master kenntlich zu machen, dass der Slave für die Übertragung bereit ist. Jede

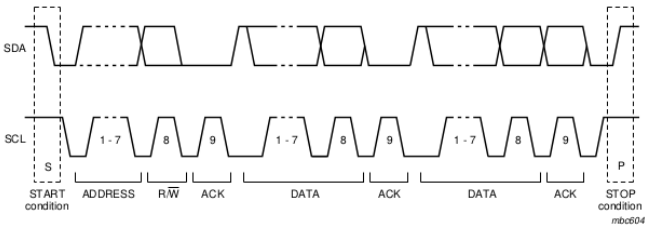


Abbildung 9. Beispiel einer vollständigen Datenübertragung. Entnommen aus [16]

Übertragung wird durch eine Start-Bedingung (fallende Flanke auf SDA, HIGH auf SCL) gestartet und durch eine Stop-Bedingung beendet. Auf die Start-Bedingung folgt die 7-bit Slave-Adresse. Darauf folgt ein Lese-/Schreibbit, wobei „0“ Schreiben und „1“ Lesen bedeuted. Der Slave muss darauf eine Bestätigung senden. Es folgen die Daten, wobei jedes Byte vom Receiver bestätigt werden muss. Beendet wird die Übertragung mit der Stop-Bedingung (steigende Flanke auf SDA, HIGH auf SCL), wobei der Master auch einen Repeated-Start senden kann, um eine neue Übertragung zu initiieren. An einem I^2C-Bus können mehrere Master angeschlossen sein. Ist dies der Fall, handelt es sich um einen Multi-Master-Bus. Da ein Master die Übertragung steuert, können mehrere Master, die den Bus belegen wollen, in Konflikt geraten. Um dies zu vermeiden, ist eine Arbitrierung, d. h. eine Ermittlung desjenigen Teilnehmers, der den Bus im Konfliktfall belegen darf, notwendig: Jeder Master hört ständig den Bus ab. Auf

diese Weise kann jeder Master feststellen, ob eine fremde Übertragung statt findet und die eigene Übertragung dementsprechend bis zum Stop-Signal einer laufenden Übertragung verzögern. Dies macht jedoch ein weiteres Problem deutlich: Warten mehrere Master auf das Stop-Signal um eine eigene Übertragung zu beginnen, so starten alle wartenden Master ihre Übertragung unmittelbar nach dem Stop-Signal und können so in Konflikt geraten.

Hier wird eine besondere Eigenschaft des I^2C-Busses genutzt: Alle Leitungen sind als logische UND-Funktion gebaut, d. h., legt ein Bus-Teilnehmer die Leitung auf LOW, so ist die Leitung LOW, selbst dann, wenn ein anderer Teilnehmer versucht die Leitung auf HIGH zu ziehen. Auf die Realisierung in der Hardware wird in Unterabschnitt VI-C eingegangen.

Master müssen weiterhin einen Konflikt erkennen, obwohl die Übertragungen gleichzeitig begonnen wurden. Dazu legen alle Master ihre Daten an und hören die Leitung gleichzeitig ab. Entspricht der Pegel der Leitung nicht dem, was der Master selbst auf die Leitung gelegt hat, so ist offensichtlich ein Konflikt vorhanden.

Beispiel zur Konflikterkennung: Master A zieht die Leitung auf HIGH, Master B auf LOW. Der Leitungspegel entspricht durch die logische UND-Funktion LOW. Master A kann dies durch Abhören der Leitung erkennen, Master B nicht. Die Übertragung von Master B wurde bisher aber auch nicht gestört, denn es lagen immer die richtigen Werte an. Zur Konfliktlösung stoppt jeder Master, der einen Konflikt erkennt, sofort die eigene Übertragung. Sie haben die Arbitrierung verloren. Außerdem müssen sich diese Master bis zum Ende der Übertragung wie ein Slave verhalten und auch die bereits gesendeten Bits als Slave interpretieren, da sie von dem Master, der die Arbitrierung gewinnt, angesprochen werden könnten.

Beim I^2C-Multi-Master-Bus ergibt sich ein weiteres Problem: Der Master gibt den Takt vor. Gibt es mehrere Master, geben auch mehrere Master den Takt vor. Dies kann während der Arbitrierung zu unendlich kurzen oder gar ausbleibenden HIGH-Phasen führen, wenn die Takte sich so ungünstig überlagern, dass die Taktleitung immer von einem Master auf LOW gehalten wird. Die Takte müssen daher synchronisiert werden. Hier nutzt man wieder die logische UND-Funktion der Leitung: Sobald ein Master einen Takt anlegt, zieht er die Taktleitung aus dem Idle-Zustand (HIGH) auf LOW. Alle anderen Master brechen ihre HIGH-Phase ab und beginnen mit der LOW-Phase. Ist die LOW-Phase eines Masters abgelaufen, legt dieser HIGH auf die Taktleitung und hört sie ab. Ist die Taktleitung weiterhin LOW, wartet der Master, bis die Leitung HIGH wird und startet erst dann die HIGH-Phase. Zu diesem Zeitpunkt starten alle Master Ihre HIGH-Phase, da jetzt auch der letzte Master die Leitung mit HIGH belegt. Damit ist der Takt synchronisiert und wird durch den Master mit der kürzesten HIGH-Phase und jenen mit der längsten LOW-Phase bestimmt. Abbildung 10 soll dieses Verhalten veranschaulichen. Die Synchronisierungsprozedur schafft eine weitere Funktion: das Clock-stretching. Ist ein angesprochener Slave noch nicht bereit, eine Antwort zu geben oder ein nächstes Byte zu empfangen, so kann dieser die Taktleitung auf LOW halten. Der aktive Master wird in einen Wartezustand

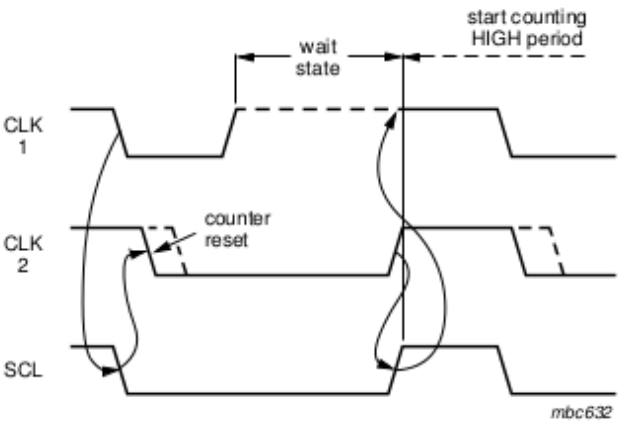


Abbildung 10. Taktsynchronisierung während der Arbitrierung. Entnommen aus [16]

gezwungen, da neue Daten erst mit dem HIGH-Pegel auf der Taktleitung gültig werden. Wie aus der Synchronisierungsprozedur bekannt, versucht der Master die Taktleitung auf HIGH zu ziehen, startet den Timer für die HIGH-Phase aber erst, wenn auf der Leitung tatsächlich HIGH anliegt. [16]

C. Hardware

I^2C benötigt zwei Leitungen: Eine Taktleitung und eine Datenleitung. Da nur eine Datenleitung zur Verfügung steht, ist ein Vollduplexbetrieb mit I^2C nicht möglich. Beide Leitungen, auch die Taktleitung, arbeiten bidirektional. Sie sind jeweils über einen Pull-Up-Widerstand angebunden. Die Verbindung zur Masse muss „offen“ (open-drain oder open-collector), also ohne Widerstand sein. Auf diese Weise wird eine logische UND-Funktion realisiert [16]. Beim Raspberry Pi sind die für I^2C vorgesehenen Pins über einen 1,8kΩ Widerstand angebunden, um einen richtigen I^2C-Betrieb zu ermöglichen. Für I^2C können Pin 3 und 5 des P1-Headers des Raspberry Pi verwendet werden. Für einen zweiten I^2C-Kanal stehen seit Hardwarerevision 2.0 die Pins 3 und 4 des P5-Headers zur Verfügung. [13]

D. Anwendung

I^2C wird in über 1000 verschiedenen ICs von über 50 Herstellern angewendet. Auch Peripherie wie der Temperatursensor LM75 von National Semiconductor lässt sich ansteuern [15]. Ebenso werden die Sensoren eines Quadcopters, der in [4] vorgestellt wird, mit I^2C angesteuert. Außerdem bildet es die Grundlage für verschiedene weiterführende Bus-Systeme wie dem System-Management-Bus (SMBus), dem Power-Management-Bus (PMBus), dem Intelligent-Platform-Management-Interface (IPMI), dem Display-Data-Channel (DDC) und der Advanced-Telecom-Computing-Architecture (ATCA). [16] Auch bietet sich I^2C für adressgebundene Eigenentwicklungen an.

VII. Ausblick

GPIO-Pins sind in ihrer Funktion so einfach und allgemein gehalten, dass sie ebenso einfach und allgemein einsetzbar

sind. Sie bilden eine Schnittstelle zur digitalen und nichtmechanischen Steuerung von elektrischem Strom. Sie befähigen dazu, weitere Systeme anzubinden, um ein bestehendes System funktional zu erweitern. Stehen nicht genug GPIO-Pins zur Verfügung, so lassen sich diese mit Erweiterungsplatinen wie dem Gertboard [9] vervielfachen. Besonders im Bereich der Automatisierungs- und Steuerungstechnik werden GPIO-Pins oft benötigt.

SPI, UART und I^2C bieten eine einfache serielle Kommunikation über die GPIO-Pins des Raspberry Pi.

UART hat seine Wurzeln in den Zeiten der Fernschreibertechnik, die nie abgelöst, sondern erweitert und verbessert wurde. Noch heute werden für Telefon und Internet elektrische Leiter zur Fernkommunikation eingesetzt. UART bietet heute in vielen Geräten *die* Möglichkeit zur direkten Kommunikation mit der Platine. Auch beim Raspberry Pi ist UART als Sonderfunktion der GPIO-Pins standardmäßig aktiviert und bietet, jenseits der üblichen Methoden wie SSH oder eine über USB angeschlossene Tastatur, Zugriff auf die Shell.

SPI hat gegenüber UART den Vorteil, mehrere Slaves anschließen zu können und keine teuren Oszillatoren zu benötigen. Heute wird SPI oft in „In-System-Programmern“ (ISP) eingesetzt um Microcontroller zu programmieren. Microcontroller sind heute und in absehbarer Zukunft fester Bestandteil in elektronischen Geräten. Oft arbeiten sie unbemerkt, wobei kaum ein elektronisches Gerät ohne Microcontroller auskommt. ISP und damit auch SPI bieten eine einfache Möglichkeit die Microcontroller, selbst in ansonsten fertig produzierten Geräten, mit den nötigen Programmen zu versehen. Diese Programme später noch einmal zu verändern ist unüblich, aber möglich. SPI hat sich etabliert, findet eine weite Verbreitung, ist einfach und günstig zu implementieren und erfüllt zuverlässig seinen Zweck.

I^2C ist die jüngste der hier vorgestellten Schnittstellen. Mit den Pull-Up-Widerständen und dem deutlich umfangreicheren Protokoll ist es zwar etwas aufwändiger zu implementieren, dennoch ist es so einfach, dass ein schnelles und umfangreiches Verständnis möglich ist. Zudem sind viele Funktionalitäten optional, zeigen aber, was mit nur zwei funktionalen Leitungen möglich ist. Als Kommunikationsmittel zwischen eingebetteten Systemen ist I^2C häufig die geeignete Schnittstelle. So wird es von vielen Microcontrollern unterstützt und wurde gleich mehrfach für spezielle Einsatzgebiete adaptiert (siehe Unterabschnitt VI-D). Es ist zudem die einzige der drei hier behandelten Schnittstellen, für die noch heute eine ordentliche Spezifikation gepflegt wird (siehe [16]).

Die serielle benötigt im Vergleich zur parallelen Kommunikation weniger Leitungen und bietet ein hohes Maß an Skalierbarkeit. Insbesondere mit Blick auf Schnittstellen wie USB (Universal *Serial* Bus) wird deutlich, dass uns die serielle Kommunikation noch lange begleiten wird. So implementiert USB weitere Funktionen wie die automatische Geräteerkennung und den Gerätewechsel im laufenden Betrieb und nutzt physikalischer Eigenschaften besser aus. [2].

LITERATUR

[1] URL: http://www.goblack.de/desy/digitalt/1_theorie/theo-schnittstellen/rs232/ (besucht am 26. 01. 2014).

[2] Eik Arnold. „Untersuchung der Implementierung und Programmierung von USB-Schnittstellen für die Übertragung von Daten und die Steuerung von Messaufbauten“. Magisterarb. Technische Universität Chemnitz, 1. Dez. 2003.

[3] Atmel. *AVR151: Setup And Use of The SPI*. URL: http://www.atmel.com/Images/doc2585.pdf (besucht am 29. 12. 2013).

[4] Andy Baker. „Quadcopter“. In: *MagPi* (Dez. 2013).

[5] *Der UART, Teil 1*. URL: http://www.physik.uni-regensburg.de/studium/edverg/elfort/C_KURS_Atmel_Programmieren%20htm/Der_UART_1.htm (besucht am 29. 12. 2013).

[6] *I^2C vs SPI*. 25. Jan. 2014. URL: http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/.

[7] Rolf Keller. *Bit für Bit*. 1983. URL: http://www.hsg-kl.de/faecher/inf/netze/material/bitfuerbit.html (besucht am 29. 12. 2013).

[8] Alexander Langer. *Python Programmierung*. 8. Apr. 2012. URL: http://raspberrycenter.de/handbuch/python-programmierung (besucht am 03. 01. 2014).

[9] *Raspberry-Pi-Erweiterung Gertboard wieder verfügbar*. 22. Jan. 2013. URL: http://www.heise.de/hardware-hacks/meldung/Raspberry-Pi-Erweiterung-Gertboard-wieder-verfuegbar-1789239.html (besucht am 26. 01. 2014).

[10] *Raspberry PI: GPIO*. URL: http://www.rn-wissen.de/index.php/Raspberry_PI:_GPIO (besucht am 02. 01. 2014).

[11] *Raspberry Pi - Revision 2.0 - Schematics*. 18. Okt. 2012. URL: http://www.raspberrypi.org/wp-content/uploads/2012/10/Raspberry-Pi-R2.0-Schematics-Issue2.2_027.pdf (besucht am 29. 12. 2013).

[12] *RPi Hardware*. URL: http://elinux.org/RPi_Hardware (besucht am 26. 01. 2014).

[13] *RPi Low-level peripherals*. 2013. URL: http://elinux.org/RPi_Low-level_peripherals (besucht am 29. 12. 2013).

[14] *RPi Low-level peripherals*. 2013. URL: http://elinux.org/RPi_SPI (besucht am 29. 12. 2013).

[15] National Semiconductor. *LM75 Digital Temperature Sensor and Thermal Watchdog with Two-Wire Interface*. 1996. URL: http://pdf.datasheetcatalog.com/datasheet/nationalsemiconductor/DS012658.PDF (besucht am 26. 01. 2014).

[16] NXP Semiconductors, Hrsg. *UM10204 - I^2C-bus specification and user manual*. 9. Okt. 2012. URL: http://www.nxp.com/documents/user_manual/UM10204.pdf.

[17] *SPI - Serial Peripheral Interface*. 2013. URL: http://www.mct.de/faq/spi.html (besucht am 29. 12. 2013).

[18] *UART*. URL: http://www.mikrocontroller.net/articles/UART (besucht am 29. 12. 2013).

[19] *UART*. URL: http://www.rn-wissen.de/index.php/UART (besucht am 29. 12. 2013).

Assemblerprogrammierung des Raspberry Pi's

Björn Hullmann

***Zusammenfassung*—Schon seit längerem gibt es das Raspberry Pi und damit auch der Assembler dafür. Viele (Hobby-) Programmierer kennen sich aber nicht mit dem Assembler aus und wissen nicht, wozu das Raspberry Pi fähig ist. In diesem Paper werden die Grundlagen erklärt, um ein eigenes Programm zu schreiben. Zudem wird noch darauf eingegangen, wie sich C und Assembler verbinden lassen und wie die spezielle Hardware (GPIO) des Raspberry Pi's angesteuert werden kann. Der Assembler des Raspberry Pi's eignet sich hervorragend für schnelle Programme und ist günstig in der Anschaffung.**

I. Einleitung

Die Informatik wird immer präsenter im Alltag der Menschen. Dies geschieht sowohl bewusst als auch unbewusst. Die meisten Haushälter besitzen heutzutage einen oder mehrere Computer, aber die wenigsten wissen, wie sie Programme schreiben können. Viele sind damit zufrieden, wenn sie darauf spielen oder schreiben können und haben Angst den Computer mit selbstgeschriebenen Programmen zu zerstören. Hier bietet sich das Raspberry Pi an. Daran können Jugendliche aber auch Erwachsene durch das Spielen und Experimentieren das Programmieren erlernen. Aber auch für Hobbybastler und Programmierer, die einen dauerhaften Controller oder Server brauchen ist das Raspberry Pi eine preiswerte Alternative.

Die meisten Programme werden in höheren Programmiersprachen geschrieben. Diese sind z.B. „C", „Java" und „Python", welche auch auf dem Raspberry Pi funktionieren. Hohe Programmiersprachen ermöglichen es, schnell und einfach große Programme zu schreiben. Der Nachteil an diesem Programm ist aber, dass beim Übersetzen in die Maschinenbefehle häufig viele unnötige Befehle generiert werden. Um dies zu vermeiden benutzt man den Assembler, weil diese Sprache eins zu eins in Maschinenbefehle übersetzt wird. Weiterhin sind laut [16] die Assemblerprogramme in der Regal wesentlich schneller als andere Programme anderer Sprachen. Ein weiterer Vorteil ist dabei, dass diese Programme weniger Speicherplatz benötigen und auch alle Funktionen der Hardware stehen einem zur Verfügung, da nicht alle immer in anderen Sprachen implementiert wurden.

Dieses Paper erläutert im Folgenden, was man braucht, um ein Assemblerprogramm für das Raspberry Pi zu schreiben. Dabei geht es als ersten auf dem Aufbau des Prozessors ein und anschließend werden die Befehle und Werte aufgelistet, die für die ersten Programme benötigt werden. Danach auf die „Calling Convention" ein und wie man C-Programme und Assembler verbindet, um schneller Programme schreiben zu können. Weiterhin wird dann noch erläutert, wie man mit dem Assembler die spezielle Hardware (GPIO) des Raspberry Pi's ansteuert. Zum Schluss werden dann noch die wichtigsten Unterschiede zwischen dem Raspberry Pi-Prozessor und den Intel Prozessoren erläutert.

II. Grundlagen

Ein Assembler ist ein Programm, welches einen Programmcode übersetzt, der sich aus Assemblerbefehlen zusammensetzt. Diese Befehle können sich stark zwischen unterschiedlichen Prozessoren unterscheiden. Prozessoren der selben Entwicklungsreihe verfügen aber über alle Befehle ihrer Vorgänger, damit die Abwärtskompatibilität erhalten bleibt. Trotzdem können neuere Prozessoren der selben Reihe zusätzliche Befehle aufweisen.[1]

Der Hauptunterschied zu höheren Sprachen, bei denen ein Befehl übersetzt aus mehreren Maschinenbefehlen besteht, ist, dass die Assemblerbefehle übersetzt genau ein Maschinenbefehle sind. Zudem können Assemblerprogramme nicht wie Programme vieler anderen Sprachen problemlos auf ein anderes System übertragen werden. Dies kommt dadurch, dass unterschiedliche Prozessoren unterschiedliche Befehle aufweisen. Deshalb ist es sehr aufwendig, ein Assemblerprogramm auf ein anderes System zu übertragen.[1]

Weiterhin muss bei der Assemblerprogrammierung darauf geachtet werden, dass es zwei Arten von Zahlen gibt. Zum einen gibt es die Zahlen, die sowohl positiv als auch negativ sein können. Diese Zahlen werden als *sigend* bezeichnet. Dahingegen gibt es auch noch Zahlen, die nur positive Werte haben. Solche Zahlen werden als *unsigend* bezeichnet. Der Unterschied zwischen den beiden Arten ist das 31. Bit. Bei *sigend* Zahlen wird es genutzt, um das Vorzeichen anzugeben. Dahingegen wird bei *unsigned* Zahlen das 31. Bit dafür verwendet, um den Zahlenbereich der positiven Zahlen zu vergrößern.

Für die Assembler Programmierung wird beim Raspberry Pi der GNU Assembler genutzt, welcher auch unter den Namen GAS bekannt ist.[9]

III. CPU

Der Prozessor vom Raspberry Pi befindet sich im Chip BCM 2835, der speziell für das Raspberry Pi hergestellt wird. Dieser Chip beinhaltet den Hauptprozessor, den Grafikprozessor und den Arbeitsspeicher.[2] Der Raspberry Pi-Prozessor basiert auf der ARMv6-Architektur und gehört der ARM11 Familie an. Dieser Prozessor hat nur einen Kern, der mit 700 MHz arbeitet. Die Bezeichnung für diesen Prozessor ist „ARM1176JZF".[4]

Für die Datenverarbeitung besitzt dieser Prozessor 16 Register je 32 Bit und ein nicht vollständig genutztes 32 Bit Statusregister. Die 16 Register können im Assembler Code mit der Bezeichnung „r0" bis „r15" angesprochen werden und haben dort verschiedene Aufgaben. Die Aufgabe der Register „r0" bis „r3", welche auch mit „a1" bis „a4" angesprochen werden können, besteht darin, die Parameter zu übergeben. Zusätzlich können sie zur Datenverarbeitung

31	30	29	28	27		24	23	19…16		9	8	7	6	5	4…0
N	Z	C	V	Q	-	J	-	GE[3:0]	-	E	A	I	F	T	M[4:0]

Tabelle I: Aufbau des Statusregisters [6], [7]

verwendet werden. Das Register „r0“ hat zudem die Aufgabe Rückgabewerte von Funktionen zu halten. Bei 64 Bit Werten wird das Register „r1“ hinzugenommen. Die Register „r4“ bis „r9“ dienen nur zur Datenverarbeitung und können auch mit der Bezeichnung „v1“ bis „v6“ angesteuert werden. Das Register „r10“ kann neben dieser Bezeichnung auch noch mit den Bezeichnungen „v7“ oder „sl“ angesprochen werden. Es dient dazu, die Grenze des Stacks zu speichern, wenn sie von einem Benutzer gesetzt wird. Weithin ist das Register „r11“ dafür zuständig, den Frame-Pointer zu speichern. Dieses Register hilft dabei auf die lokalen Variablen zu zugreifen, die auf dem Stack gespeichert sind, indem es die Anfangsadresse von den Variablen beinhaltet. Dies kommt z.B. zum Einsatz, wenn Funktionen lokale Variablen benötigen. Andere Bezeichnungen für dieses Register sind „v8“ und „fp“. Das Register „r12“ dient als temporäres Register, welches für die Datenverarbeitung eingesetzt werden kann. Das Register „r13“, auch „sp“ genannt, ist der Stackpointer, welches die Adresse des obersten Wertes auf dem Stack beinhaltet. Zudem existiert das Link-Register „r14“ (auch „lr“ bezeichnet). Dieses speichert die Rücksprungadresse. Anschließend gibt es noch das Register „r15“. Dieses ist der Programm-Counter und kann alternativ auch mit „pc“ angesprochen werden.[3], [5]

Das Statusregister ist wie in Tabelle I aufgebaut. Dabei haben die Bits folgende Bedeutung [6], [7]:

- Zustand-Flags:
  - N (Negative): Ist gesetzt, wenn das Ergebnis negativ ist.
  - Z (Zero): Ist gesetzt, wenn das Ergebnis 0 ist.
  - C (Carry): Beim Shiften übernimmt dieses Bit den Wert, welcher raus geschoben wurde. Es ist zudem gesetzt, wenn beim Vergleichen, Addieren oder Subtrahieren das Register nicht reicht und dadurch ein unsigned Overflow entsteht.
  - V (Overflow): Wenn beim Addieren oder Subtrahieren ein signed Overflow entsteht, dann wird dieses Bit gesetzt.
- Interrupt-Flags:
  - I: Wenn es gesetzt ist, sind normale Unterbrechungsanforderung (IRQ Interrupts) deaktiviert.
  - F: Wenn es gesetzt ist, sind höher priorisierte Unterbrechungsanforderung (FIQ Interrupts) deaktiviert.
  - A: Wenn es gesetzt ist, sind Imprecise Abort deaktiviert, welche Auftreten können, wenn es zu Fehler kommt, wenn auf dem virtuellen Speicher zugegriffen wird.
- Opcodemodus des Prozessors (werden später noch erklärt):
  - J (Jazelle): JVM-Bytecode wird ausgeführt.
  - T (Thumb): Thumb Bytecode wird ausgeführt.

Weiterhin steht dem Prozessor ein Co-Prozessor für Fließkommazahlen zur Verfügung, welcher über einen eigenen Registersatz verfügt und die Floating-Point-Unit (FPU) ist. Für diese FPU gibt es 32 Register mit je 32 Bit für einfache Genauigkeit und 16 Register mit 64 Bit für doppelte Genauigkeit, welche für das Arbeiten mit Fließkommazahlen zuständig sind. Dabei handelt es sich bei den 32 Bit und 64 Bit Registern um dem selben Registersatz. Zwei der 32 Bit Register entsprechen jeweils einem der 64 Bit Register. Zusätzlich hat die FPU noch drei weitere 32 Bit Register. Aus dem ersten kann nur gelesen werden und enthält die ID des Prozessors. Seine Bezeichnung im Assembler Code ist „fpsid“. Das zweite Register ist für die Exceptions zuständig. In ihm werden genauere Informationen zu den Exceptions abgespeichert und hat die Bezeichnung „fpexc“. Das letzte der drei Register erreicht man unter der Bezeichnung „fpscr“ und ist das Status und Controll Register der FPU. In ihm werden sowohl die Flags als auch die Rundungsvorschriften abgespeichert. Außerdem kann man durch das setzen von Bits im Register bestimmte Exceptions ausschalten, sodass die nicht mehr auftreten werden. Zudem wird in ihm die Art der Exceptions gespeichert.[3], [9], [8] Die 32 32 Bit-Register der Floating-Point-Unit werden mit „s0“ - „s31“ angesprochen. Wenn man sie allerdings als 64 Bit-Register ansprechen möchte, werden die Bezeichnungen „d0“ - „d15“ verwendet.[3]

IV. Befehle

Um ein Programm zu erstellen, werden einige Befehle benötigt. Dabei gibt es zwei Gruppen von Befehlen. Dies sind zum Einen die Befehle, die Anweisungen für den Assembler sind, und zum Anderen sind es die Befehle, die zur Datenverarbeitung zuständig sind. Im folgenden wird erklärt, wie Kommentare aussehen und wie man Labels und Werte definiert. Danach werden die Assembleranweisungen und in mehrere Bereichen unterschiedliche Befehlsgruppen erklärt.

- Kommentare: Wenn Kommentare in Assembler Code geschrieben werden, werden sie durch die Zeichen „/*“ eingeleitet und durch „*/“ beendet. Hierbei wird alles, was zwischen ihnen ist, als Kommentar behandelt.[9]
- Label: Um Labels zu erstellen, kann man die Zeichen „A“ - „Z“, „a“ - „z“, „1“ - „9“ und „_“ verwenden (z.B. „label_1“). An der Stelle der Definition wird noch ein „:“ dran gehängt. Hinter einem Label kann noch ein Befehl folgen. Labels sind dafür da, um Sprungmarken zu definieren, welche auch Funktionen darstellen können.[9]
- Assembleranweisungen: Die folgenden Befehle teilen dem Assembler mit, dass sie etwas bestimmtes machen sollen. Dabei sorgt der Befehl „.global“ dafür, dass das dahinter angegebene Label global ist. Dadurch existiert es auch außerhalb der Assemblerdatei und kann so von anderen Dateien aufgerufen werden. Zudem sorgt es dafür, dass globale Labels von anderen im Assemblercode benutzt werden kann.[15]
  Um jetzt Bereiche zu definieren, wird der Befehl „.section“ verwendet. Dabei schreibt man hinter diesen einen weiteren Befehl. Kommt dahinter der Befehl „.data“, wird der Datenbereich eingeleitet, wo die Variablen erstellt werden.

Will man den Codebereich einleiten, wird stattdessen der Befehl „.text“ verwendet.[15]
Mit dem Befehl „.balign“ wird dafür gesorgt, dass eine gewisse Anzahl von Bytes für das Nachfolgende reserviert wird. Wie viel wird durch die dahinter angegebenen Dezimalzahl bestimmt. Dabei wird kein Zeichen vor die Zahl geschrieben (z.B. „.balign 4“). Dem folgt in der nächsten Zeile ein Label, welchem der ganze reservierte Speicher zugewiesen wird. Zudem kommt dann noch ein weiterer Befehl dahinter, welcher angibt, um welche Art von Wert es sich handelt. Hierbei steht „.word“ für ein 32 Bit-Wert. Werden beim Reservieren mehr Bytes reserviert, als gebraucht werden, wird der Rest mit Nullen aufgefüllt. Anstatt den Befehl „.word“ kann auch „.byte“ für ein 8 Bit-Wert, „.hword“ für ein 16 Bit-Wert oder „.dword“ für ein 64 Bit-Wert angewendet werden. Diese Befehle werden gefolgt von einem Wert, mit der die Variable dann initialisiert wird (z.B. erste Zeile „.balign 4“ und nächste Zeile: „label: .word #4“).[15] Das Zugreifen auf die Variablen wird unten in einem Beispiel gezeigt.

- Werte und Adressen: Wenn man jetzt Werte definieren möchte, muss man dem Assembler sagen, um was für Werte es sich handelt. Deshalb werden Dezimalzahlen immer mit einem „#“ angeführt (z.B. „#4“ oder „#-2“). Für weitere Zahlensystem müssen andere Zeichen voran geführt werden. Hierbei steht „0x“ für Hexadezimalzahlen, „0“ für Oktalzahlen und „0b“ für Binärzahlen. Zudem kann man auch Zeichen definieren, welche mit einem „’“ beginnen und enden (z.B. „’c’“). Soll jetzt einer der Zahlenwerte eine Adresse repräsentieren, wird zusätzlich noch ein „=“ angeführt (z.B. „=0x20002020“). Um den Assembler zu sagen, dass er den Wert eines Registers als Adresse nehmen soll, werden die Zeichen „[“ und „]“ verwendet, wo das Register dazwischen geschrieben wird. Zusätzlich kann hierbei Adressrechnung gemacht werden, wodurch sich der Wert im Register nicht ändert. Dies geschieht indem man in den Klammern hinter dem Register noch ein Zahl mit einem Komma davor schreibt (z.B. „[r1, #-4]“, hier wird die Adresse um vier verringert).[8]
- Bewegungsbefehle: An Bewegungsbefehlen gibt es drei Stück, diese Befehle sind „mov“, „ldr“ und „str“.
  Mit dem Befehl „mov“ kann man Werte bewegen. So kann man zum einen zwei Register dahinter angeben (z.B. „mov r0, r1“). Dabei wird der Wert des zweiten Parameters in das andere Register kopiert. Aber als zweiter Parameter ist auch ein Wert möglich, welcher dann ins Register geschrieben wird (z.B. „mov r0, #4“). Dies können aber keine Adressen sein.[8]
  Zum Laden von Werten aus dem Speicher benötigt man den Befehl „ldr“. Mit ihm können Werte von Variablen oder Werte aus Speicheradressen in Registern geladen werden. Man benutzt den Befehl, indem man als ersten Parameter das Register angibt, indem der Wert geladen werden soll. Der zweite Parameter ist ein Register, dessen Wert er als Adresse nehmen soll (z.B. „ldr r0, [r1]“) oder die Variable. Außerdem ist es möglich, die Adresse nach dem Laden eines Wertes zu verändern. Dies geschieht, indem man als dritten Parameter noch eine Zahl angibt, welche auf die Adresse aufaddiert wird (z.B. „ldr r0, [r1], #-4“). Weiterhin kann man auch eine Adresse in ein Register laden, indem man als zweiten Parameter die Adresse angibt (z.B. „ldr r0, =0x2002020“). Hierbei wird dann die Adresse ins angegebene Register gespeichert. Neben dem „ldr“ Befehl für 32 Bit, gibt es auch noch vier weitere. Diese sind „ldrb“, welcher einen 8 Bit-Wert lädt, „ldrsb“, welcher einen 8 Bit-Wert lädt und ihn dabei als einen signed Wert behandelt, „ldr“, welcher einen 16 Bit-Wert lädt, und „ldrsh“, welcher einen 16 Bit-Wert lädt und ihn als einen signed Wert behandelt. Hierbei werden nicht belegte Bits im Register mit Ausnahme bei den Befehlen „ldrsb“ und „ldrsh“ auf Null gesetzt. Bei den anderen beiden werden die Bits nur mit Nullen gefüllt, wenn es keine negative Zahl ist, sonst wird das Register mit Einsen gefüllt.[8]
  Der Befehl „str“ ist identisch mit dem Befehl „ldr“ mit der Ausnahme, dass er anstatt Werte zu Laden, sie unter der gegebenen Speicheradresse abspeichert. Zudem gibt es keine Möglichkeit als zweiter Parameter eine Adresse anzugeben. Als Gegenstücke zu den Befehlen „ldrb“ und „ldrh“ gibt es die Befehle „strb“ und „strh“. Hierbei werden die letzten Bits eines Registers immer abgespeichert. Die Befehle „ldrsb“ und „ldrsh“ haben keine Gegenstücke, weil sie nicht gebraucht werden.[8]
- Arithmetische Befehle: Für die Addition gibt es die Befehle „add“ und „adc“. Diese Befehle verlangen drei Parameter, wobei der erste Parameter das Register ist, in dass das Ergebnis abgespeichert werden soll. Der zweite Parameter ist ein Register und gibt den ersten Wert an, welcher addiert wird. Der andere Wert, der addiert wird, wird über den dritten Parameter bestimmt. Hier kann sowohl ein Register, als auch ein Wert angegeben sein (z.B. „add r0, r1, #4“). Der Unterschied zwischen „add“ und „adc“ ist, dass bei „adc das Carry Flag mit eingerechnet wird und bei „add“ nicht.[8]
  Die Subtraktion ist identisch mit der Addition. Abgesehen davon das hier subtrahiert anstatt addiert wird. Die Befehle hierfür lauten „sub“ und für die Subtraktion mit Carry „sbc“.[8]
  Will man multiplizieren, dann verwendet man die Befehle „smull“ und „umull“. Diese Werte sind identisch mit der Ausnahme, dass „smull“ zwei signed Werte multipliziert und „umull“ zwei unsigned Werte. Diese beiden Befehle verlangen vier Register (z.B. „umull r0, r1, r2, r2“), wobei die ersten beiden Register nicht das selbe sein dürfen. Dieser Befehl multipliziert den dritten Parameter mit dem vierten und schreibt das Ergebnis in den ersten Parameter. Der entstandene Überhang wird in das Register gespeichert, welches als zweites angegeben wurde.[8]
- Schiebebefehle: Neben dem Rechnen, können auch Werte verschoben werden. Dies geschieht mit den Befehlen „lsl“, „lsr“ und „asr“. „asr“ ist für das arithmetische Verschieben nach rechts und „lsr“ bzw. „lsl“ für das logische verschieben nach rechts bzw. links. Dabei ist der erste Parameter das Zielregister und der zweite Parameter, das Register, dessen Wert verschoben wird. Der dritte Parameter ist ein Wert von 1 - 31 bzw. bei „lsl“ 0 - 31, der angibt, um wie viele Stellen verschoben wird (z.B. „lsr r0, r1, #4“). Diese Befehle stehen

nur den Registern „r0“ bis „r7“ zur Verfügung.[8]

- Logische Befehle: Als logische Befehle stehen die Befehle „and“, „orr“, „eor“ und „bic“ zur Verfügung. Bei diesen Befehlen wird Bitweise die logische Operation zwischen dem zweiten und dritten Parameter durchgeführt und das Ergebnis in dem als erstes angegebenen Register gespeichert. Der zweite Parameter ist ein Register und der dritte kann ein Register oder ein Wert sein (z.B. „bic r0, r1, #10“). Hierbei steht der Befehl „and“ für „und“, „orr“ für „oder“, „eor“ für „exklusives oder“ und „bic“ für „nicht und“.[8]
- Stack-Befehle: Um auf dem Stack zuzugreifen, gibt es die Befehle „push“ und „pop“. Dabei ist beim Stack zu beachten, dass die oberen Werte auf dem Stack, die niedrigeren Adressen haben. Um etwas auf dem Stack zu schreiben, verwendet man den Befehl „push“. Dabei wird der Wert des angegeben Registers auf dem Stack geschrieben. Man kann auch gleichzeitig mehrere Parameter auf dem Stack schreiben, indem man sie, mit einem Komma getrennt, oder mit Bindestrich dazwischen in geschweifte Klammer schreibt. Beim Bindestrich werden auch alle sich dazwischen befindliche Register auf dem Stack geschrieben. Dabei werden die Register in der Reihenfolge „r0 “ bis „r15“ abgelegt (z.B. „push {r0, r2-r4}“).[8]
  Um Werte vom Stack zu holen, benutzt man „pop“. Dieser Befehl hat die gleiche Syntax wie „push“ und speichert den obersten Wert vom Stack ins Register. Bei mehreren übergebenen Register wird der obersten Wert des Stacks in dem höchsten Register gespeichert.[8]
- Sprungbefehle: Für Sprünge gibt es vier grundlegende Befehle, die es in verschiedenen Variationen gibt. Diese Befehle sind „b“, „bl“, „bx“ und „blx“. Die Befehle „b“ und „bl“ verlangen ein Label als Parameter. Hingegen verlangt der Befehl „bx“ ein Register mit einer Adresse drin. Beim Befehl „blx“ kann sowohl ein Label als auch ein Register angegeben werden. Beim Befehl „b“ wird nur zum angegebenen Label gesprungen. Dahingegen speichert der Befehl „bl“ dazu noch die Rücksprungadresse (also die Adresse der nächsten Anweisung) in das Register „lr“. Es ist auch möglich, beim Springen den Modus zu wechseln. Dies geschieht über den Befehl „bx“. Durch diesen Befehl wird zu der angegebenen Adresse in dem übergebenen Register gesprungen. Hierbei ist zu beachten, dass wenn das nullte Bit in der Adresse gesetzt ist, dass es in den Thumb-Modus springt und dann bei der Adresse minus Eins weiter macht. Ansonsten wechselt es in dem normalen Modus. Was der Thumb-Modus ist, wird weiter unten erklärt. Da alle Befehle 4 Byte bzw. in Thumb 2 Byte groß sind, liegt der Anfang eines Befehls immer bei einer geraden Adresse. Dadurch kann man noch zu allen Befehlen springen. Der letzte Befehl ist „blx“. Dieser speichert die Rücksprungadresse in „lr“ und wechselt gleichzeitig den Modus wie „bx“, wenn ihm ein Register übergeben wird. Wird ein Label übergeben, wechselt der Befehl immer in den Thumb-Modus. Sowohl „bl“ als auch „blx“ speichern den vorherigen Modus in die Rücksprungadresse beim nullten Bit mit ab.[8]
  Diese Befehle gibt es zudem in Versionen, die nur unter bestimmten Bedingungen ausgeführt werden. Hierbei wird der Befehl um eine Endung erweitert. Will man z.B. nur springen wenn das Zero-Flag gesetzt ist, wird die Endung „eq“ verwendet (z.B. bei „bl“: „bleq label“). Im folgenden werden einige dieser Endungen aufgelistet und unter welchen Bedingungen sie springen:[8]
  - „eq“: Gleichheit (Z-Flag ist gesetzt).
  - „ne“: Ungleichheit (Z-Flag ist nicht gesetzt).
  - „vs“: Overflow (V-Flag ist gesetzt).
  - „vc“: Kein Overflow (V-Flag ist nicht gesetzt).
  - „mi“: Negativ (N-Flag ist gesetzt).
  - „pl“: Positiv oder Zero (N-Flag ist nicht gesetzt).
  - „hs“: Größer oder gleich (C-Flag ist gesetzt).
  - „hi“: Größer (C-Flag ist gesetzt und Z-Flag nicht).
  - „ls“: Kleiner oder gleich (C-Flag ist nicht gesetzt, aber Z-Flag).
  - „lo“: Kleiner (C-Flag ist nicht gesetzt).
- Vergleichsbefehle: Um zwei Werte miteinander zu Vergleichen, wird der Befehl „cmp“ verwendet. Dabei ist der erste Parameter ein Register und der zweite ein Register oder ein Wert. Hierbei wird der zweite auch Parameter vom ersten abgezogen, aber das Ergebnis wird nicht gespeichert. Stattdessen werden hier nur die Flags gesetzt.[8]

Wie eben schon erwähnt, gibt es mehre Modi. Zum einen ist es der normale ARM-Modus. Aber es gibt auch noch den Thumb-Modus und den Jazelle-Modus. Der Thumb-Modus wurde eingeführt, um Speicher zu sparen. Dies geschieht dadurch, dass die Befehle nur 16 Bit anstatt 32 Bit lang sind. Der Vorteil davon ist, dass weniger Speicher für das Programm benötigt werden. Aber der Nachteil ist dabei, dass im Thumb-Modus nicht alle Befehle einem zur Verfügung stehen. Diese Befehle müssen dann Mithilfe von anderen ersetzt werden, weshalb man mehrere Befehle für die gleiche Anweisung braucht. Weiterhin kommt es bei einem Sprung immer zu einer Störung des Programmflusses. Dies ist beim Thumb-Modus vom Nachteil, weil viele bedingte Befehle, die hier nicht aufgeführt wurden, es in diesem Modus nicht gibt, und deshalb mit Sprüngen ersetzt werden müssen. Deshalb wird dieser Modus häufig dazu verwendet, um Speicher zu sparen. Aber in zeitkritischen Abschnitten (z.B. Interrupts) wird dann zum normalen Modus gewechselt.[8]

Der Jazelle-Modus wurde eingeführt, um beim Prozessor bei Java einer höhere Performance zu erreichen. Dies geschieht dadurch, dass in diesem Modus der Java Bytecode direkt verarbeitet werden kann.[8]

A. Assembler Programm

Im Folgenden werden einige Dinge anhand eines Beispiels erklärt [9]:

```
.section .data
.balign 4
return: .word #0

.section .text
func1:
        bx lr

.global main
main:
    ldr r4, address_of_return
```

```
    str lr, [r4]
    bl func1
    ldr lr, [r4]
    bx lr
address_of_return: .word return
```

In diesem Beispiel wurden durch „.section“ zwei Bereiche definiert. Dies ist zum einen der Datenbereich durch „.data“ und zum anderen der Code Bereich durch „.text“. Im Datenbereich wird mit „.balign 4“ dem Assembler mitgeteilt, dass ein vier Byte großer Bereich benötigt wird. Dieser Bereich wird dann dem Label „return“ zugewiesen, welcher ihn mit einem 32 Bit-Wert nämlich Null belegt. Im Codebereich wird unten erst einmal eine Möglichkeit erstellt, auf die Daten zuzugreifen. Dies wird benötigt, da es sonst nicht möglich ist, aus einem Bereich einen anderen Bereich zu nutzen. Dadurch wird im Label „address_of_return“ die Adresse von „return“ gespeichert. Am Anfang wurde die Funktion „func1“ definiert, was über ein Label geschehen ist. Dennoch kann man es auch als normales Sprungziel nutzen. Diese Funktion springt gleich zu der Adresse, welche in „lr“ steht. Danach wird noch die Funktion „main“ erstellt, wobei mit „.global“ erreicht wird, dass sie auch von außerhalb aufrufbar ist. Dies wird benötigt, da sonst das Programm nicht ausführbar ist.[9]

Dieses Programm lädt, als erstes den Wert unter der Adresse „address_of_return“ in „r1“. Dieser Wert ist die Adresse von „return“, welche dann in „r4“ gespeichert ist. Als nächstes wird dann der Wert von „lr“ unter der Adresse von „return“ abgelegt. Dadurch wurde die Rücksprungadresse gesichert, da mit „bl func1“ wird eine neue Rücksprungadresse in „lr“ geschrieben und die Funktion „func1“ ausgeführt wird. Diese Funktion springt sofort zurück und danach wird die alte Rücksprungadresse aus „return“ wieder geholt, weil die Adresse von „return“ immer noch in „r4“ abgespeichert ist. Danach wird mit „bx lr“ das Programm beendet, weil in „lr“ jetzt die Adresse ist, von der aus das Programm aufgerufen wurde.[9]

V. Calling Convention

Wenn z.B. eine Library erstellt werden soll, wird das Wissen benötigt, wie die Übergabe von Parametern geregelt ist. Wird es nicht geregelt, gibt es eine Großzahl an Möglichkeiten die Parameter zu übergeben. Die erste Person nutzt nur den Stack, der nächste nur „r1“ und wiederum ein anderer „r2“ und „r3“. Dadurch kommt es dazu, dass man von jeder Funktion die Parameterübergabe wissen oder nachschauen muss. Dieses Problem wurde durch ein Abkommen zur Parameterübergabe gelöst.

Beim Raspberry Pi ist es so geregelt, dass die ersten vier Register („r0“ - „r3“) für die Parameterübergabe genutzt werden. Hierbei wird der erste Wert ins Register „r0“ geschrieben und der vierte Wert ins Register „r3“. Bei einem 64 Bit Wert wird der Wert in zwei Register gespeichert. Dabei werden die hinteren Bits in das erste Register geschrieben und die vorderen Bits ins nächste. Werden mehr als vier Parameter übergeben, dann werden die weiteren Parameter auf den Stack übertragen. Hierbei wird der hinterste Parameter als erstes auf dem Stack geschrieben und bekommt damit die höhere Adresse. Folglich wird Parameter fünf immer als letztes auf dem Stack geschrieben, womit der Wert des Parameters die niedrigsten Speicheradresse bekommt. Wird ein größeres Objekt übergeben (z.B. ein Array), wird nicht jeder Wert des Objektes übergeben, sondern es wird nur ein Zeiger auf das Objekt (also die Anfangsadresse des Objektes) übergeben.

Das gleiche Prinzip wird auch bei der Rückgabe eines Wertes angewendet. Ist es auch hier ein Objekt, wird nur der Zeiger darauf zurückgegeben, welcher in „r0“ gespeichert wird. Bei einem 32 Bit Wert wird stattdessen der Wert direkt in „r0“ zurückgegeben, während bei einem 64 Bit Wert die niedrigeren Bits in „r0“ gespeichert werden und die höheren in „r0“.

Beim schreiben einer Funktion, muss man dafür sorgen, dass die Register „r4“ - „r11“ und das Register „lr“ gesichert werden, wenn diese in der Funktion zum Einsatz kommen. Dies geschieht, indem die Register am Anfang einer Funktion auf dem Stack geschrieben werden und vor dem Beenden der Funktion wieder in die Register geladen werden.[13]

VI. C und Assembler

Um schneller Programme schreiben zu können, kann man sowohl in C als auch im Assembler programmieren. Dies hat den Vorteil, dass man in C schneller Programme schreiben kann, aber trotzdem an den gewählten Stellen die Vorteile vom Assembler besitzt kann. Im Folgenden wird der gegenseitige Aufruf anhand eines Beispiels erklärt.

Assemblerprogramm:

```
.section .text
.global main
main:
        mov r0, #7
        mov r1, #5
        push lr
        bl cfunc
        pop lr
        bx lr

.global asm
asm:
        push {r1, lr}
        sub r0, r0, #1
        bl cfunc
        pop {r1, lr}
        add r0, r0, r1
        bx lr

.global cfunc
```

C-Programm:

```
int asm(int a, int b);

int cfunc(int c, int d) {
        if (c > d) {
                return asm(c, d);
        }
        return c;
}
```

Um im Assembler eine C-Methode nutzen zu können, muss man wissen, welche Parameterwerte die Funktion verlangt und welchen Rückgabewert sie hat. Um jetzt auf die Funktion zugreifen zu können, muss im Assemblercode im Codeabschnitt die Funktion definiert werden. Dies geschieht mit

dem Assemblerbefehl „.global cfunc“. Nun kann man vom Assemblercode aus die C-Funktion „cfunc“ ausführen, wie es bei einer Assemblerfunktion funktioniert. Im C-Code definiert man nur den Rumpf der Assembler Methode. Hier wird es mit „int asm(int a, int b)“ bewerkstelligt und dabei wird in C festgelegt, welche Parameter übergeben werden und welche Typen sie und der Rückgabewert haben werden.[8], [15]

Die beiden Funktionen machen nur wenig. Die C-Funktion überprüft, ob der erste Parameter größer als der zweite ist. Wenn dem so ist, ruft sie die Assemblerfunktion auf, sonst gibt sie den ersten Parameter zurück. Diese Assemblerfunktion dekrementiert den ersten Parameter und ruft dann wieder die C-Funktion auf. Danach addiert die Funktion den Rückgabewert mit dem zweiten Parameter und gibt es zurück.

VII. Die spezielle Hardware

Für eigene Schaltkreise ist die spezielle Hardware des Raspberry Pi's interessant. Diese besteht aus den 26 Pins, auch General purpose input/output (GPIO) genannt, die an der Außenseite des Raspberry Pi's angebracht sind. Diese Pins sind frei programmierbar.[10] Wie die Pins genutzt werden, wird im Folgenden anhand der zweiten Version des Raspberry Pi's gezeigt. Aber auch die Anordnung der Pins kann sich zwischen den Versionen unterscheiden. Bei Version 2 haben zwei Pins eine Spannung von 3,3 V und zwei die Spannung von 5 V. Zusätzlich gibt es fünf Pins, die geerdet sind. Die anderen 17 Pins stehen zur Programmierung zur Verfügung. Die Anordnung der Pins bei der zweiten Version vom Raspberry Pi ist in der Tabelle II dargestellt.[11]

Für die Benutzung dieser Pins braucht man lediglich nur die zuständigen Speicheradressen, welche ab der Adresse „0x20200000“ losgehen. Jede Speicheradresse besteht aus 4 Bytes, wobei jeder Pin jeweils 3 Bits zugeordnet ist. Diese Bits bestimmen, welche Funktion der Pin übernimmt. Dabei steht der Wert „000“ für Input und „001“ für Output. Das 30. und 31. Bit jeder Speicheradresse ist reserviert. Dabei beinhaltet die Adresse „0x20200000“ die Pins 0-9 und „0x20200004“ die Pins 10-19. Dies geht bis zur Adresse „0x20200014“ so weiter, welche die Pins 50-53 enthält. Hierbei sind nicht alle auf der Hardwareebene zur Verfügung. Ein Beispiel wäre, dass die Speicheradresse „0x2020004“ mit dem Wert „0000 0000 0000 0000 0001 0000 0000 1000“ belegt wäre. Hier würden die Pins 10, 12, 13, 15, 16, 17, 18 und 19 auf Input eingestellt sein und die Pins 11 und 14 auf Output.[11]

Im folgenden wird der Output-Funktion näher erläutert. Wird der Pin auf Output gestellt, wird noch kein Signal gesendet. Dies geschieht erst, wenn man in der Speicheradresse „0x20200028“ für die Pins 0 - 31 bzw. „0x2020002C“ für die Pins 32 - 53 bei dem jeweiligen Bit eine Eins setzt. Schreibt man jetzt z.B. beim nullten und beim 16. Bit eine Eins, sendet der Pin 0 und Pin 16 ein Signal.[11] Schreibt man aber an der selben stelle eine Null statt der Eins rein, wird das Signal nicht beendet. Um es abzuschalten werden die Adresse „0x2020001C“ bzw. „0x20200020“ benötigt. Das Ausschalten eines Signals funktioniert wie das Einschalten, indem man z.B. an der nullten Stelle eine Eins schreibt.[12]

```
.section .text
.global main

main:
        ldr r0, =0x20200000
        mov r1 ,#1
        lsl r1 ,#18
        str r1, [r0,#4]

        lsr r1, #2
        str r1, [r0,#40]

        loop:
        b loop
```

Dies war ein kleines Beispiel, wie man ein dauerhaftes Signal beim Pin 16 sendet. Zuerst wird hier die Startadresse der Pins geladen. Danach wird eine Eins in „r1“ geschrieben und auf die 18. Stelle verschoben, womit dann der Wert „001“ im Abschnitt für Pin 16 stehen würde. Danach wird in der Adresse „0x20200004“ der Wert von „r1“ geschrieben, womit der Pin auf Output eingestellt wird. Um das Signal zu senden wird die Eins in r1 um zwei nach rechts verschoben, womit sie auf die 16. Stelle im Register ist. Dies wird dann in der Adresse „0x20200028“ gespeichert und ein Signal wird bei Pin 16 gesendet. Danach wird nur noch eine Endlosschleife ausgeführt, die dafür sorgt, dass das Programm nicht beendet wird.[11]

Das Einlesen eines Wertes ist fast gleich. Der Unterschied besteht lediglich darin, dass ein Wert beim Raspberry Pi unter der Adresse „0x20200034“ für Pin 0 bis 31 bzw. „0x20200034“ für 32 bis 53 ausgelesen wird. Das Raspberry Pi speichert automatisch eine Eins an der Stelle, die für den jeweiligen Pin steht, wenn ein Signal reingeht. Beispiel: Kommt bei Pin 22 ein Signal rein, dann wird das 22. Bit des Wertes unter der Adresse „0x20200034“ auf Eins gesetzt. Dies könnte man dann auslesen lassen.[10]

VIII. Raspberry Pi vs. Intel Prozessoren

Der Prozessor vom Raspberry Pi und die Intel Prozessoren haben einige Grundlegende Unterschiede. Das Raspberry Pi verwendet ein ARM-Prozessor, welche auf einer RISC-Architektur basiert. Die Intel Prozessoren verwenden eine CISC-Architektur. Dies bedeutet, dass die ARM-Prozessoren deutlich weniger Befehle haben als Intel Prozessoren, weil sie auf möglichst einfache Befehle ausgelegt sind. Dagegen ist die CISC-Architektur darauf ausgelegt, möglichst viele und auch komplexe Befehle auf Hardwareebene zu haben, um dadurch eine höhere Performance zu erreichen. Der Nachteil ist dabei aber, dass schnelle Befehle (z.B. das Kopieren des Wertes von einem ins andere Register) einen genauso langen Takt haben, wie die langsameren, komplexeren Befehle, weil die Taktfrequenz dabei unverändert bleibt. Da der ARM Prozessor keine komplexen Befehle hat, besitzt er im Vergleich einen höheren Befehlsdurchsatz und auch bei Interrupts kann schneller reagiert werden. Weiterhin kann dadurch eine kleinere und somit günstigere Makrozelle verwendet werden, weil in ihr nicht die Steuerung komplexer Befehle gespeichert ist, sondern nur einfache. Allerdings brauchen die RISC-Architekturen mehrere Befehle, um die komplexen Befehle einer CISC-Architektur nachzubilden.[8]

5V	5V	Ground	Pin 14	Pin 15	Pin 18	Ground	Pin 23	Pin 24	Ground	Pin 25	Pin 8	Pin 7
3.3V	Pin 0	Pin 1	Pin 4	Ground	Pin 17	Pin 27	Pin 22	3.3V	Pin 10	Pin 9	Pin 22	Ground

Tabelle II: GPIO Anordnung beim Raspberry Version 2[11]

Des weiteren kann der ARM Prozessor bei Interrupts auf Schattenregister zurückgreifen, wodurch eine schnellere Abarbeitung erfolgen kann. Im Vergleich dazu müssen bei Intel Prozessoren die Register erst auf dem Stack gesichert werden, bevor sie die Interrupts bearbeitet werden können.[8] Die Vergleichsprozessoren von Intel wären ihre x86-Reihe, welche wie der Arm-Prozessor 32 Bit Architekturen sind. Diese haben aber nur acht Register plus den Programm Counter, wobei der ARM Prozessor 16 Register zur normalen Verarbeitung zur Verfügung steht. Ähnlich ist es bei der FPU. Bei Intels x86-Reihe stehen der FPU acht Register der Größe 80 Bit zur Verfügung, wobei die Werte beim Abspeichern auf 64 Bit reduziert werden [14]. Hingegen hat der ARM Prozessor für die FPU 16 Register je 64 Bit.[9]

IX. Zusammenfassung

Zusammenfassend lässt sich sagen, dass mit dem Assembler auf dem Raspberry Pi schnelle Programm geschrieben werden können, die auch wegen dem Thumb-Modus deutlich weniger Speicherplatz verbrauchen. Trotz der RISC-Architektur kann er durch den Einsatz mehrere Befehle die Befehle nachstellen, die eine CISC-Architektur zusätzlich besitzt. Zudem stehen ihm im Vergleich zu Intel's x86-Reihe mehr Register zur Verfügung. Das Raspberry Pi besitzt außerdem einen Co-Prozessor für die FPU, um auch Fließkommazahlen zu verarbeiten. Seine Register im Hauptprozssor sind zwar nur 32 Bit lang, trotzdem kann es 64 Bit zahlen bearbeiten. Das Interessante am Raspberry Pi ist seine GPIO (die 26 Pins), welche sich leicht ansteuern lassen. Dadurch ist es eine gute Option, um eigene elektronische Schaltkreise zu steuern. Des weiteren ist es möglich, für das schnellere schreiben von Programme, C und Assembler zu verbinden.

Folglich lässt sich sagen, dass das Raspberry Pi eine günstige und sehr gute alternative für Personen ist, die mit ihm Assembler programmieren möchten oder einen Controller für Schaltkreise brauchen.

Literatur

[1] Sebastian Stein, Grundlagen Assembler :: assembler.hpfsc.de, Internet: http://assembler.hpfsc.de/ Zugriff 2013-11-18.

[2] Assembler Handbuch :: Assembler Programmierung :: raspberrycenter.de, Internet: http://raspberrycenter.de/handbuch/assembler-programmierung Zugriff 2013-11-18.

[3] Microsoft Developer Network :: ARM Registers :: , Internet http://msdn.microsoft.com/en-us/library/ms253983%28v=vs.80%29.aspx Zugriff 2013-11-21

[4] Broadcom :: BCM2835:: broadcom.com, Internet: http://www.broadcom.com/products/BCM2835 Zugriff 2013-11-21.

[5] PPS Pascal Programming For Schools :: ARM Assembler :: pp4s.co.uk, Internet: http://www.pp4s.co.uk/main/tu-trans-asm-arm.html Zugriff 2013-11-21

[6] ARMwiki - HeyRick! :: The Status Register :: heyrick.co.uk, Internet: http://www.heyrick.co.uk/armwiki/The_Status_register Zugriff 2013-11-28

[7] Low Level :: ARM :: lowlevel.eu, Internet: http://www.lowlevel.eu/wiki/ARM Zugriff 2013-11-28

[8] ARM :: The Architecture for the Digital World :: infocenter.arm.com, Internet: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0204j/Bcdcfejg.html Zugriff 2013-12-19

[9] Think in Geek :: ARM assembler in Raspberry Pi :: thinkingeek.com, Internet: http://thinkingeek.com/2013/01/09/arm-assembler-raspberry-pi-chapter-1/ Zugriff 2013-12-19

[10] Raspberry Pi :: raspberrypi.org, Internet: http://www.raspberrypi.org/ Zugriff 2014-01-01

[11] Interrupt Handler :: Raspberry Pi Operating System Development: Working with GPIO Pins :: http://interrupthandler.com, Internet: http://interrupthandler.com/raspberry-pi-os-dev-notes-lesson-1/ Zugriff 2014-01-01

[12] University of Cambridge :: Raspberry Pi :: http://www.cl.cam.ac.uk, Internet: http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok02.html Zugriff 2014-01-01

[13] ARM :: Procedure Call Standard for the ARM® Architecture :: 30.11.2012 :: infocenter.arm.com, Internet: http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf Zugriff 2014-01-01

[14] x86-Assembler-Programmierung :: Die FPU verwenden :: freebsd.org, Internet: https://www.freebsd.org/doc/de/books/developers-handbook/x86-fpu.html Zugriff 2014-01-01

[15] Using as :: sourceware.org, Internet: https://sourceware.org/binutils/docs/as/ Zugriff 2014-01-21

[16] Rohde, Joachim / Roming, Marcus: Assembler. Grundlagen der Programmierung, Österreich: mitp, 2006

Running the Raspberry Pi with 3.3V

Jan Steffen Becker
Carl von Ossietzky University of Oldenburg
mail: jan.steffen.becker@uni-oldenburg.de

***Abstract*—The raspberry Pi is a small computer that is attractive for mobile applications because of its size. These applications often lack an infinite power source and therefore the device is required to consume as less power as possible. In this paper an efficient 3.3V power supply for the Raspberry Pi is designed and evaluated.**

***Index Terms*—Raspberry Pi, power supply, linear voltage regulator, switching mode regulator**

I. Introduction

The Raspberry Pi is a very small, cheap and easy to use general purpose computer. Its size makes it attractive for portable devices that can be easily carried and set up. A possible application is a portable measuring station for water quality. These applications also require a portable power supply for the Raspberry Pi.

The usual way of supplying power to the Pi is via the USB port. But other more portable forms might be possible. So one can think of alternative ways, e.g.

- power supply by battery (e.g. 12V car battery and a set of 1.5V batteries)
- power supply by solar energy

Possible ways to reduce the Pi's power consumption are of interest. In this paper an alternative power supply is designed and evaluated. It supplies the Raspberry Pi with 3.3V via its GPIO module. Instead the manufacturer designed the Pi to be powered with 5V via a USB port.

This paper is structured as follows. In Section II the basics of DC power supplies are explained. This is followed in Section III by an explanation of an external power supply design that replaces the USB port.

This power supply and the power consumption of the Pi in its standard configuration is evaluated afterwards in Section V. The results are listed in Section VI. This leads to some simple calculations how long the Raspberry Pi can run using batteries.

II. Power supply basics

The Raspberry Pi is designed to get power via a USB mini port. USB is specified to work with a voltage of 5V so the Pi gets 5V input from the USB. But most of the Pi's components require an input of 3.3V or less. To reduce the the input voltage of 5V (or any voltage higher than 3.3V) to 3.3V, an electrical circuit, called *voltage regulator* or *power converter*, is used.

There are two types of power converters: *Linear power converters*, sometimes called *Low Dropout converters* (LDO in short), and switching mode power converters [1]. These two kinds of converters work in different ways and each has its own pros and cons. In the following, both are explained and compared.

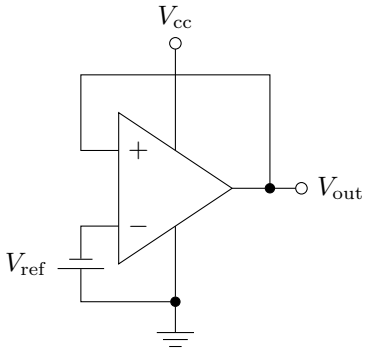


Figure 1. Linear regulator functional block circuit diagram

A. Linear power converters

linear power converters represent the more simple and somewhat older technology. In an abstracted view they consist of a reference voltage source and an amplifier that is used to pull the differential between output and ground to the reference voltage or a multiple of it. This is shown in Figure 1.

Ideally in the circuit the equation holds:

$$V_{\mathrm{out}} = A_{\mathrm{vol}}(V_{\mathrm{ref}} - V_{\mathrm{out}}) \tag{1}$$

V_{out} is the output voltage and A_{vol} the factor of the amplifier. This is equivalent to

$$V_{\mathrm{out}} = V_{\mathrm{ref}} \frac{A_{\mathrm{vol}}}{1 + A_{\mathrm{vol}}} \tag{2}$$

Setting A_{vol} to an infinite value reduces this to

$$V_{\mathrm{out}} = V_{\mathrm{ref}} \tag{3}$$

Even if this is a very simplified view of a linear regulator's functionality, it gives an idea how it works.

Linear regulators are very simple and therefore cheap components. They produce low noise and are accurate. But the big disadvantage of this circuit is its efficiency. At the amplifier, the input voltage is divided in two parts—the reference voltage and the rest. In a regular voltage regulator the input and output currents are approximately equal. The power from the source that is not needed is wasted to heat. Therefore LDOs often require a heat sink. If one calculates the efficiency under that

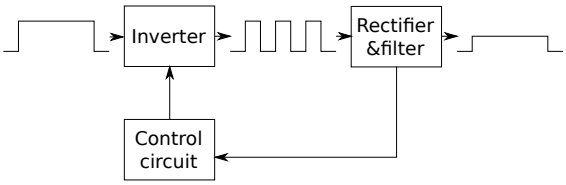


Figure 2. Basic functionality of a switching mode power converter

assumption, it results in (cf. [2])

$$q_{\text{eff}} = \frac{p_{\text{in}}}{p_{\text{out}}} = \frac{V_{\text{in}} \cdot I_{\text{in}}}{V_{\text{out}} \cdot I_{\text{out}}} = \frac{V_{\text{in}}}{V_{\text{out}}} \tag{4}$$

So for high differences between the input and the output voltage, the efficiency becomes very low.

B. Switching mode converters

As mentioned above, switching mode power converters are based on a totally different technology. Compared to linear regulators switching mode converters are more efficient for high input–output voltage differences. The functionality can described as follows:

A switching mode converter consists of three components as shown in Figure 2. The first component is a high frequency inverter that converts the DC input voltage to a high frequency rectangular wave. This wave is rectified and filtered back to a DC output of lower voltage. A control circuit compares the resulting output voltage with a reference voltage source and adjusts the on-time of the inverter output, so that the filtered output is at the intended voltage level.

The advantage of this design is that the voltage drop is not achieved by wasting power. The efficiency is much higher (more than 90%) even at high voltage differences. Moreover the circuit is more complex and less cheap than a linear regulator. Because of the internal conversion to a rectangular wave some noise remains at the output.

The efficiency can not be calculated easily as with linear regulators. It highly depends on the regulator itself and can be read from the component's data sheet.

III. Designing an external power supply

The Raspberry Pi is intended to be powered via a USB port. As one can see from the Raspberry Pi's board schematic [3], the mini-USB port for power supply on the board is connected to the input pin of a linear power converter. This regulator produces the 3.3V output for other components on the board. The intention for an external 3.3V power supply is to replace or circumvent this regulator and to supply directly 3.3V to the Pi.

A. Dave Akerman's way

On his blog, Dave Akerman [4] gives detailed instructions how to replace the on-board regulator with a more efficient one to improve the efficiency.

He proposes to remove the on board regulator chip totally together with a stabilizing capacitor on the board. Because he uses the Raspberry Pi in a weather balloon he might intend to reduce the weight as much as possible.

He suggests to replace the on-board regulator (an LM1117) with an MCP1825S. This is a 3.3V-LDO regulator, too, but it has a minimum dropout voltage that is less than 0.2V. That means that the chip just needs an input voltage of about 3.5V to work. The idea is that batteries that supply together less than 5V can be used as a power source.

Dave Akerman gives instructions to connect the MCP1825S output pin to both the pins on the board that are connected to the input and the output pin of the on-board regulator. The MCP1825S ground pin is to connect the ground pin on the board and the batteries. The input pin is just connected to the battery.

In this way, the 5V and 3.3V rails on the Pi's board are connected together. As a result all components on the board are supplied with 3.3V. Even if no component on the board requires 5V to provide its functionality the 5V-line can not be left open. Namely, the power sensing logic of the processor is connected to this line.

Dave Akerman does not use any stabilizing capacitors as they are highly recommended in linear regulator data sheets. This seems to work anyway.

The design proposed by Dave Akerman has some disadvantages. Because of the soldering on the board the manufacturer's warranty for the device is lost. If one is not familiar with SMD soldering there is a risk to damage the Pi. Additionally the modifications can not be undone. The device can not be driven with 5V anymore.

B. The used power supply

Because of the facts mentioned above another way is chosen in this work. As a power converter chip a TSR 1-2433 is selected. As suggested by Dave Akerman, the 3.3V and 5V rails on the Raspberry Pi are connected. But instead of removing the old converter chip, the Pi is not modified.

1) The TSR 1-2433 switching mode converter: The heart of the designed power supply is a TSR 1-2433 switching mode power converter. Even if it is not cheap (about 9 EUR when ordered from Reichelt [5]) it has a lot of nice features for this application. There are no additional components required. Most switching mode converters require at least two filtering capacitors and one inductor. So it is very easy to use. It has a wide input range from 4.75 to 32V. With an additional capacitor at the input even 36V are possible. It is at full load $\pm 2\%$ accurate and has less than 50mV noise. It is between 78% at maximum input voltage and 91% at minimum input voltage efficient.

2) The GPIO module: Here the Raspberry Pi's GPIO module is used to supply the 3.3V power. The idea is to build a socket that can be connected to any applicable power source (e.g. batteries). The Raspberry Pi's GPIO module is simply plugged into it.

The Raspberry Pi's GPIO module consists of two rows with 13 pins each. The use of the pins is shown in figure 3. As can be seen from the board schematics, pins 2 and 3 (the 5V output

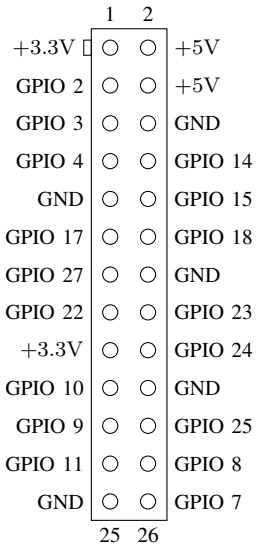


Figure 3. The Raspberry Pi's GPIO module pins [6]

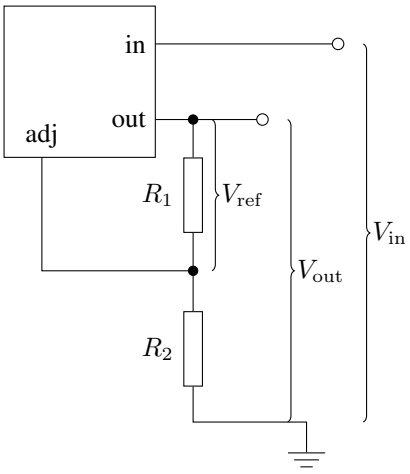


Figure 4. Adjusting a linear power supply. R_1 and R_2 are used to divide the output voltage. The regulator adjusts the voltage at R_1 to its reference voltage.

pins) are directly connected to the input pin of the on-board LDO regulator and pins 1 and 17 are directly connected to the regulator's output pin. Pins 6, 9, 14, 20 and 25 are connected to ground. So they can be used to supply power to the Pi. It was chosen to use pins 1 and 2 as power pins and pin 6 as ground pin.

To connect any DC voltage source via the TSR 1-2433 to the Raspberry Pi, pin 1 (the input pin) and pin 2 (the ground pin) of the regulator are simply connected to the power source. Pin 2 is also connected to pin 6 of the Pi's GPIO module and pin 3 (the regulator's output pin) is connected to pins 1 and 2 of the GPIO module.

3) Why the old regulator can be left on the board: The next question to answer is if one can leave the on-board regulator at its place without risking damage or power loss. A voltage regulator is designed to have a voltage higher than its minimum input voltage at its input. It is not intended to supply the output voltage to both the input and the output pins. But this is what happens with the external power supply designed here. How will the regulator behave? The answer is in the data sheet of the regulator. To achieve an output voltage higher than the reference voltage (these are 1.25V) two resistors are used as shown in Figure 4. The first resistor is between the output and the adjust pin (this is also the ground pin) of the voltage regulator. The second resistor is placed between the adjust pin and the ground. This divides the output voltage in two parts. The regulator adjusts the output voltage, so that the reference voltage is at resistor 1. So the following equation holds:

$$\frac{V_{\text{out}}}{V_{\text{ref}}} = \frac{R_1 + R_2}{R_1} \tag{5}$$

For the fixed[1] versions of the LM-1117 ICs used for the Pi, these resistors are integrated in the circuit. If one supplies power to both the input and the output pins, the current floating through the circuit is limited at least by resistor 2. According to the LM-1117 data sheet, in normal operation the quiescent current (that is the current at the ground pin) for the 3.3V fixed version is less than 10mA. Some simple calculations give that R_2 is about 125Ω. This means that when supplying 3.3V the current is less than 16mA, assuming the internal adjust pin is short cut to the output. This is quite low and a vulnerable loss of energy.

IV. The 3.3V-problem

When the Raspberry Pi is powered with 3.3V in one of the explained ways (replacing the on-board regulator or using the GPIO module) all outputs of the Raspberry Pi are 3.3V or less, too. However, some peripherals for the Raspberry Pi require 5V power. 5V output are required in the USB and HDMI specifications. Even if there seem to exist some USB devices that work with 3.3V, too [4], other require 5V. So which power supply to choose depends on the application, actually

- on the primary input voltage and
- if 5V output is required.

If 5V output is required, the input voltage has to be at least 5V, too. If the input voltage is about 5V, the standard setup of the Pi can be used or the power source can be wired directly to the GPIO module. If the input is above 5V it has to be stepped down anyway. Therefore a switching mode converter with 5V output is applicable.

If 5V output is not required and the input is below the minimum input of the switching mode regulator, a linear

[1]Most regulators are produced in fixed and adjustable versions. Fixed versions are adjusted to an output voltage (e.g. 3.3V) by the manufacturer. In adjustable versions the application designer uses resistors as in Figure 4 for a custom output voltage.

regulator with very low dropout can be used. This is mentioned by Akerman as described above.

Otherwise—if 5V output is not needed and the input is above 4.75V—the design shown in this paper is applicable.

V. The test setup

The following section describes the experiments that were executed with the Raspberry Pi Model A. Goal of these experiments is to measure the power consumption of the Pi with and without the external power supply.

During each run of the experiment the device's power consumption is measured in four different situations.

1) When the CPU is idle, that means that no task with intense calculations or file system use is executed.
2) When the CPU is permanently loaded with calculations.
3) When a task that accesses permanently the SD card, is executed.
4) After the Raspberry Pi has shut down.

First it was planned to measure the power consumption during the Raspberry Pi's boot process, too. But it was found that the consumption is not constant in that phase, so that with the available equipment no measurement was possible.

The experiment is executed with the TSR 1-2433 as an external power converter with both 5V (to compare with the use of the on board LDO) and 12V input (simulating for example a car battery). Additionally it is performed with the on-board LDO instead of the switching converter chip. In this way one can compare the use of a switching mode power supply with the manufacturer intended design. The results are discussed in section VI.

A. The software layout

First the software running on the Raspberry Pi during the tests should be explained. The Pi runs the current out-of-the-box version of the Raspbian operating system. Because the Pi runs without any peripherals (the 5V-output-problem was discussed above) there are only limited ways of communication with the device. There are no means of input other than switching on the power. During the test, the Pi can give feedback through the OK LED on the board. Normally, this LED shows the SD card activity but it can be controlled software, too. This way one can use it as a general purpose binary output. Here it signals which phase of the above experiment is currently executed.

Because there is no input to the Pi, a CRON job is used, which executes a shell script every time after the Pi has booted. This script controls the signal LED using a simple additional tool, and executes two benchmark programs that put a constant load to the system. The schedule of the script is as follows:

1) Flash the OK LED ten times with the pattern *long long short short short*. This pattern can be easily distinguished from the irregular flashing of the LED during the boot process. The process that controls the LED is in a waiting state in most of the time, so one can assume the CPU to be idle. The first value is measured here.
2) Switch off the OK LED and execute the *DhryStone* benchmark (see Section V-A2). This puts calculation load to the CPU. The second value can be measured.
3) Flash the LED once for a second and switch it on. Then execute the *fsdisk-w* benchmark. This benchmark simply writes random data to the SD card. This simulates file system use. The third value is measured in this phase.
4) Shut down the Raspberry Pi. After shutdown the system flashes the OK LED ten times. When this has been visualized the last value is measured.

1) Controlling the OK LED: The OK LED on the Raspberry Pi's board can be controlled very simple in software via an easy to use file system based interface [7]. The file `/sys/class/leds/led0/brightness` contains a decimal number that controls the LED. Storing 0 in that file switches the LED off. Storing a number > 0, i.e. 1, switches the LED on.

The file `/sys/class/leds/led0/trigger` indicates how the system uses the LED. By default, the LED indicates the SD card activity. Storing `None` in that file deactivates that.

Because root permissions are required to access these files, a helper program, with the suid-bit set, does that job. It has been written in C for this purpose and is documented in the annex.

2) The benchmark suite: To put load on the system, two benchmarks from the popular benchmark suite UnixBench [8] are used. This benchmark suite is available as source code under the GPL license and can be compiled and executed on most Unix-like platforms. The current version 5.1.3 compiles on the Raspbian system without problems.

The used benchmarks are

- *Dhrystone:* This benchmark was originally proposed by Reinhold P. Weicker in [9]. It was designed to measure the performance of standard programming statements in compiled code. It does not use IO functions so it is used in the experiments to power up the CPU.
- *fsdisc-w:* This benchmark is intended to measure disc writing performance [8]. It writes blocks of random data to the file system. Therefore, it is used here to evaluate the power consumption when writing to the SD card.

B. The physical layout

There are two different experiments performed. The second one at two different input voltages, once with 5V and once with 12V input.

The physical layout of the experiments is very simple. An adjustable DC power source is used to obtain an input voltage of 5V or 12V. For the second experiment the output socket of the power source is connected to the input pin of the switching mode converter. Between the source's ground socket and the converter's ground pin an ammeter is placed. It is used to measure the current drawn from the power source. A voltmeter between the converters input and ground pins is used to adjust the input voltage. This is shown in Figure 5.

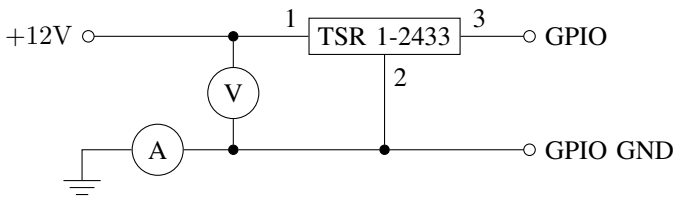


Figure 5. Layout of the second experiment.

Task	5V LDO	5V SMPS	12V SMPS
IDLE	119	97	47
DHRY	181	150	74
SD	173	145	69
OFF	33	26	13

Table I
TEST RESULTS. THE MEASURED CURRENT IN mA VS. THE INPUT VOLTAGE AND TEST SETUP.

For the first experiment the converter chip is omitted and the power source directly connected to the 5V pin of the Pi's GPIO module. The 3.3V output pin is left open.

Each experiment is started by raising the power source until the voltmeter shows the desired voltage. The Raspberry Pi starts working as soon as the voltage raises above the converters minimum input level. At any time the power consumption can be calculated from the current shown by the ammeter. One uses the simple formula

$$p_{\text{in}} = V_{\text{in}} \cdot I_{\text{in}} \tag{6}$$

VI. RESULTS

First, it should be calculated how much the power consumption can be reduced using the external power supply.

The on board linear regulator has an efficiency of about

$$q_{\text{eff}} = \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{3.3\text{V}}{5.0\text{V}} \approx 0.66 \tag{7}$$

The used switching mode converter has an efficiency between 91% and 78%. So it shall be possible to save between

$$1 - \frac{0.66}{0.78} \approx 15\% \quad \text{and} \quad 1 - \frac{0.66}{0.91} \approx 27\% \tag{8}$$

of power. Some online references say that the Raspberry Pi Model A consumes about 500mA via USB [10]. These are $500\text{mA} \cdot 5.0\text{V} = 2.5\text{W}$.

A. Test results

The results of the experiments are given in Table I. They are visualized in Figures 6, 7 and 8.

The first observation is that the current measured with the on-board LDO in use (180 mA max.) is less than the value found in other sources (between 200 and 500mA [4], [10]). The next observation is that when the Pi is idle it consumes just half of the power needed under load. As seen from Figure 8, when using 5V input up to 20% of power was saved compared to the linear regulator use. When using 12V input still 5% were saved. But one has to take into account that a 12V power source has to be stepped down to 5V before the on board LDO can be used. So in practice there is some greater effort from using the switching mode converter directly.

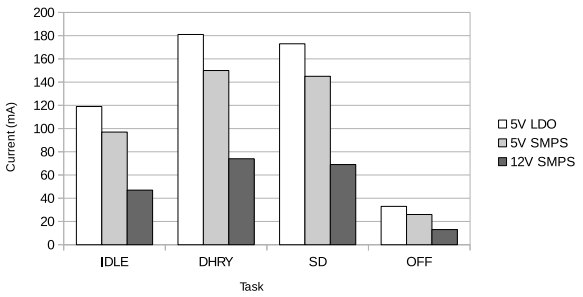


Figure 6. Test results. The measured current as in Table I.

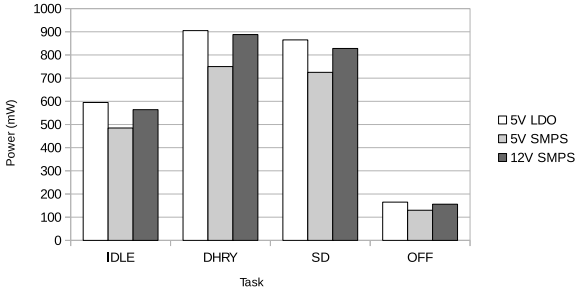


Figure 7. Test results. The measured power consumption calculated from Table I.

The power enhancements calculated above are not achieved in this experiment. However, it can be seen that using an external switching mode converter reduces the power consumption rapidly.

B. Runtime calculations

In the following, some simple runtime calculations based on the results from the last section are made.

Here is the best case assumed. That means that

- no peripherals are connected to the Pi and

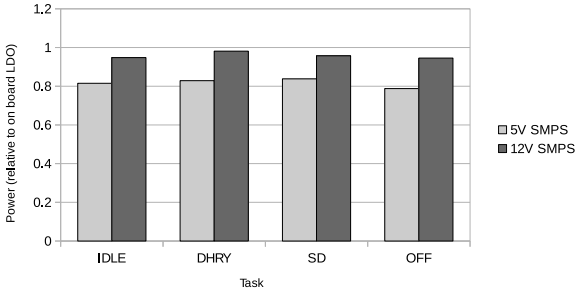


Figure 8. Test results. The measured power consumption relative to the consumption measured with the on-board regulator.

- the Pi is idle most of the time.

First let's assume that 4 simple AA batteries are used to power the Raspberry Pi. One AA battery (Alcaline) has an energy of about 3000mAh and a voltage of 3.5V. So four batteries have 3000mAh at 6V. This are 18Wh. With a power consumption of 0.5W this will last for 36h.

Dave Akerman calculates (assuming 200 mA consumption) about 15h using 4 AA batteries and a linear regulator. For 6 AA batteries he calculates about 28h using a switching mode regulator [4].

For a car battery (12V) with 70Ah capacity assuming a constant current of 50mA leads to 1400h (about 8 weeks) runtime.

However, these are very optimistic assumptions. In practice, the Raspberry Pi runs rarely without peripherals. The devices attached to the Pi consume energy, too. E.g. the Raspberry Pi Camera Board requires about 250mA [10]. So the real runtime strongly depends on the application itself.

There are some battery packs (battery powered USB chargers) that were tested for the use with the Raspberry Pi [11]. They provide between 700mAh and 12000mAh at stable 5V via USB. They shall last for up to 16h of use [11].

VII. Conclusion

In this work a power supply for the Raspberry Pi was designed. The power supply runs the Pi with 3.3V instead of the 5V the Pi is designed for. To reduce the power consumption the linear voltage regulator on the Raspberry Pi's board is replaced with a more efficient switching mode power converter. The 3.3V input is supplied to the Pi via its GPIO pins that are directly connected to the power lines.

Experiments showed that the consumed power can be reduced about 20% this way.

References

[1] *Linear & Switching Voltage Regulator Handbook*, revision 4 ed., ON Semiconductor, February 2002, aviable online at http://www.onsemi.com/pub_link/Collateral/HB206-D.PDF.

[2] B. S. Lee, *Understanding the Terms and Definitions of LDO Voltage Regulators*, Texas Instruments, October 1999, online at http://www.ti.com/lit/an/slva079/slva079.pdf.

[3] http://www.raspberrypi.org/wp-content/uploads/2012/04/Raspberry-Pi-Schematics-R1.0.pdf.

[4] D. Akerman, "Running the raspberry pi on batteries," online at http://www.daveakerman.com/?page_id=1294 last accesed on January 4, 2014, 5.00 PM.

[5] "Reichelt Electronik website," http://www.reichelt.de/index.html?ACTION=3;ARTICLE=116849;SEARCH=TSR%201-2433 last accessed on January 21, 2014, 1.15 PM.

[6] "RPi low-level peripherals," http://elinux.org/RPi_Low-level_peripherals.

[7] "Can we control the on-board leds," July 2012, raspberry Pi forum http://www.raspberrypi.org/phpBB3/viewtopic.php?f=31&t=12530, last accessed on Dec 30, 2013, 6.30PM.

[8] K. Lucas *et al.*, "byte-unixbench: A unix benchmark suite," January 2011, Version 5.1.13. Available online at http://code.google.com/p/byte-unixbench/, last accessed on Dec 30, 2013, 5.30PM.

[9] R. P. Weicker, "Dhrystone: A synthetic systems programming benchmark," *Communications of the ACM*, vol. 27, no. 10, October 1984.

[10] "Raspberry pi faqs," http://www.raspberrypi.org/faqs last accessed on January 9, 2014, 1.30 PM.

[11] "RPi verified peripherals," http://elinux.org/RPi_VerifiedPeripherals last accessed on January 9, 2014, 1.30 PM.

Appendix
Short documentation for the LED helper command

During the experiments a small utility program is used to flash the Raspberry Pi's OK-LED. In the following, it is briefly explained, how to use this program.

The utility can be invoked from the command line via

`setled` *options*

where *options* is one or more of the following:

`-f`, `--fork`: Fork to the background. The command immediately returns and the flashing of the LED is done in the background. The child's process ID is printed to standard out. When not specified, the command blocks.

`-t` *trigger*, `--trigger` *trigger*: Set the LED's trigger to *trigger*. See [7] for explanation.

`-p` *pattern*, `--pattern` *pattern*: Flash the LED according to *pattern*. *pattern* is a string of `0` and `1`, where `1` means "switch the LED on" and `0` means "switch the LED off".

`-l`, `--loop`[=*count*]: Repeat the pattern *count* times. If *count* is omitted or -1, the pattern is repeated forever or until the process receives a SIGTERM signal.

`-d` *delay*, `--delay` *delay*: Specifies the time, for that every step in the pattern is executed, in milliseconds. It defaults to 1000.

`-0`, `--off`: When terminating, switch the LED off.

`-1`, `--on`: When terminating, switch the LED on.

The order in which options are given on the command line, does not matter.

Raspberry Pi
Router & Access Point

Julian Cekici – Carl von Ossietzky Universität Oldenburg

***Abstract*—This paper is about setting up a router or access point using a Raspberry Pi. First some definitions are given followed by a deeper content, what to consider when setting up a router or access point. Furthermore the advantage of a Pi router compared to a commercially available will be explained.**

***Index Terms*—Raspberry Pi, Router, Access Point**

I. Overview

"The Raspberry Pi is a credit-card sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects, and for many of the things that your desktop PC does, like spreadsheets, word-processing and games. It also plays high-definition video. We want to see it being used by kids all over the world to learn programming."[19] From the citation of the official Raspberry Pi page shows that it is just a small computer that will inspire kids to learn how to program.

A. Router

The following definition will be translated and essentially taken over by [15]. In large networks data blocks must be forwarded to a receiver, which often has to be done through several other computer networks. Routers decide on the basis of the destination address to which other router or recipient a data block is to pass. In addition, routers got further tasks to do.

If in different Networks different data blocks are used, adress field and command field can be reversed. Otherwise if data blocks need to be reduced (fragmented), so the data blocks must be implemented accordingly. These tasks are fulfilled by routers, which have to implement data blocks from one network to another. Router thus perform various tasks, such as

- Adjustment of the data block format
- Adress translation
- Fragmenting data blocks into plurality of smaller data blocks
- Reassemble (put together) of fragmented data blocks
- Error detection and treatment
- Reporting of special events at the sender or management station
- Forwarding data blocks in the correct network (routing)
- Flow Control

As in some protocols, only short addresses are used, which are valid only on a connecting line (link), an address conversion is needed in each router that assigns each block in each network has a new address. For this, the router must perform corresponding tables from which it can remove the assignment of addresses and connections.

Since various networks can only transfer data blocks of different length, under certain circumstances has a fragmentation of long blocks to be done, or they need to be put back together again. This is partly a considerable expense, which will be avoided in modern protocols (IPv6). Packets are sent only with a fixed length, so that each packet can be transmitted without fragmentation. If errors are detected in the data, the router can still forward the data blocks still or repair it themselve.

Special events that occur in routers like interruption of a connection, overflow of buffers or routing tables a router must be able to respond to. It can send an error report or log the failure internally.

B. Access Point (AP)

The following definition will be translated and essentially taken over by [2]. An AP is an interface between wireless and wired networks. If a group of connected wireless stations with exactly one access point will be connected to wired network it is called a Basic Service Set (BSS). If more than one AP exists, then it is called an Extended Service Set (ESS). The area covered by an access point is called a cell.

II. Technical accessory

There are a few things requiered to be able to build an own router or access point. This list is mainly taken from [22].

- Ethernet cable
- Raspberry Pi model B
- WiFi adapter
- SD Card with minimum 4GB
- Micro USB cable and a power supply for the Pi

III. Preparations

A. Raspian OS

Basically it is possible to install Raspberry Pi with any operating system. In terms of comfort it is advisable to use the Raspbian OS. This is a Debian derivative and has been developed specifically for the Raspberry Pi. It can be downloaded from the source of the following quotation.

"Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled

software bundled in a nice format for easy installation on your Raspberry Pi.

The initial build of over 35,000 Raspbian packages, optimized for best performance on the Raspberry Pi, was completed in June of 2012. However, Raspbian is still under active development with an emphasis on improving the stability and performance of as many Debian packages as possible." [17]

B. Install the operation system onto SD card

Even if the Pi came with an SD card and the OS is already installed, make sure that it is updated to the latest version (in order to get the latest improvements and bug fixes). All data on the SD will be wiped off after installing the OS.

C. Boot the Pi and configure

The tool Raspi-Config automatically starts, when the Raspberry Pi runs the first time it is booting.

"Some menu entries modify the file /boot/config.txt. This file, out of the box, contains a number of commented out configuration entries; raspi-config adds entries at the end of this file. You can see what raspi-config has done to the file by viewing it on the Pi using Leafpad.[...]"[21]

As described in the aforementioned source, various changes in the config file can be done at the first start or later on in the terminal.

D. Set up Ethernet and WiFi

In order to enable the Raspberry to connect to the Internet, the ethernet cable must be connected to the Pi first. After a reboot, this should be available. Now, if the WiFi dongle is connected to the Raspberry, the first network device (wlan0) should be recognized. [24]

In order to verify whether wlan0 is active, it is able to check this in the terminal. The $ifconfig$ command does exactly this.

"If no arguments are given, ifconfig displays the status of the currently active interfaces. If a single interface argument is given, it displays the status of the given interface only; if a single argument is given, it displays the status of all interfaces, even those that are down. Otherwise, it configures an interface." [5]

IV. Install Software

This section describes which software to install and what it will be exactly used for.

A. Dynamic Host Configuration Protocol (DHCP)

"The Dynamic Host Configuration Protocol (DHCP) enables any of the computers on your local area network (LAN) to be given a network configuration automatically as soon as the boot process on the machine gets underway. Most routers are capable of this function. This function can also be carried out by a server computer, although almost any of your computers can fulfil the role. The other computers which a re configured to take advantage of this service are called DHCP Clients, and need to have their networking setup configured to use DHCP." [10] In order to download the software to set up a DHCP server best follow the instructions in the aforementioned source. Furthermore there are instructions on how to do further adjustments.

B. Set up static IP

The following definition will be translated and essentially taken over by [14].

The Raspberry Pi gets automatically assigned an IP in the noraml case in the internal network using the DHCP server. However there is the case that the router always assigns a different IP address to the Pi, so it does not have to look up after each restart.

In the above source is shown how to assign the same IP address all the time and as well which inputs are necessary for it.

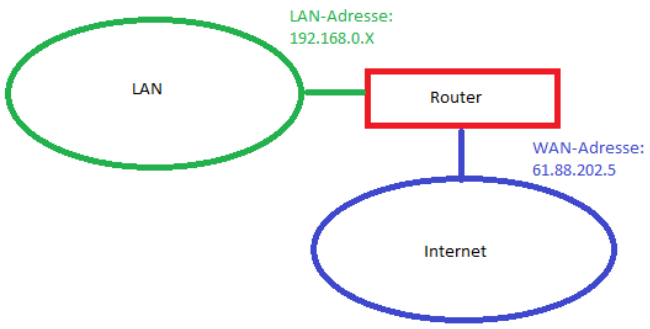
C. Hostapd

"hostapd is an $IEEE802.11AP$ and $IEEE802.1X/WPA/WPA2/EAP/RADIUS$ Authenticator." [7] "In simple words, hostapd allows you to create software wifi access points allowing decent amount of configuration options."[6] In order to install and configure hostapd there is a manual inside the source above.

The access point sofware needs to be updated to a WiFi adapter supporting version before it can be run. Using a premade hostapd from adafruit [1] makes this thing much easierer.

D. Network Address Translation

"Network Address Translation (NAT) is the process where a network device, usually a firewall, assigns a public address to a computer (or group of computers) inside a private network. The main use of NAT is to limit the number of public IP addresses an organization or company must use, for both economy and security purposes."[20]



The graph shown above makes clear how a router exactly communicates between the LAN (Local Area Network) and the WAN(Wide Area Network). In the same local network it is rather uncommon to use a NAT. In generell the number of clients communicating to each other is managed by the

size of the network. In this example the smallest network (C-Class) is mentioned. This begins at 192.168.0.0 and ends at 192.168.255.255 and can accomodate up to 2^{16} IP addresses. Furthermore there is a Class B network with 2^{20} adresses beginning with 172.16.0.0 and ending with 172.31.255.255 and an A-class network with up to 2^{24} addresses. This network adress begins with 10.0.0.0 and ends at 10.255.255.255.[13]

In the above example a client connects from a LAN to the internet. The router translates the local adress to a global one. In this case 192.168.0.X is translated to 61.88.202.5, therefore "X" stands for a number within 0-255 range. When a client sends a request to the WAN, the router adds an entry in a table. After receiving an answer, it can forward it to the appropriate client. In this case the entry in the table is removed.

To increase security, a router allows only response packets access to the LAN. A NAT is referred especially in product-related descriptions as a security feature. Thus, the mechanism is meant to prevent direct accessing stations from outside. From outside initiated connections are discarded or access is not granted to the local network. Hacker who cyclically scan all TCP-Ports for open ports do not get response from the router.

In generell NAT acts as a rudimentary firewall blocking all unauthorized external access. This is a deliberate protective function against unrequested and insecure traffic. A NAT does not replace a packet filter and certainly not a full firewall. This prevents only data connections not initiated from the local network.[20]

E. Firewall

"A firewall is a security device that can be a software program or a dedicated network appliance. The main purpose of a firewall is to separate a secure area from a less secure area and to control communications between the two. Firewalls can perform a variety of other functions, but are chiefly responsible for controlling inbound and outbound communications on anything from a single machine to an entire network."[18]

"Firewalls are vital to network management. Without this control over computer and network access, large networks could not store sensitive data intended for selective retrieval. Firewalls are also very important for home broadband users [...]."[18]

1) Software Firewall: "Software firewalls, also sometimes called personal firewalls, are designed to run on a single computer. These are most commonly used on home or small office computers that have broadband access, which tend to be left on all the time. A software firewall prevents unwanted access to the computer over a network connection by identifying and preventing communication over risky ports. Computers communicate over many different recognized ports, and the firewall will tend to permit these without prompting or alerting the user[...].

A software firewall also allows certain programs on the user's computer to access the Internet, often by express permission of the user [...].

Some software firewalls also allow configuration of trusted zones. These permit unlimited communication over a wide variety of ports. This type of access may be necessary when a user starts a VPN client to reach a corporate intranet. "[18]

2) Hardware Firewall: "Hardware firewalls are more complex. They also have software components, but run either on a specially engineered network appliance or on an optimized server dedicated to the task of running the firewall. The operating system underlying a hardware firewall is as basic as possible and very difficult to attack. Since no other software runs on these machines, and configuration takes a little more thought than clicking on an "allow" prompt, they are difficult to compromise and tend to be extremely secure.

A hardware firewall is placed between a network, such as a corporation, and a less secure area, such as the Internet. Firewalls also can separate more secure networks from less secure networks, such as one corporate location within a larger corporate structure. Versions of hardware firewalls are available to home users who want stronger protection from potential Internet attacks. There are many different default configurations for these devices - some allow no communications from the outside and must be configured, using rules[...]."[18]

3) Raspberry Pi Firewall: The Linux firewall $iptables$ is already installed in the Raspbian OS. In the Raspbian basic configuration of iptables every access to any port for any IP address is allowed. This is not desirable from view of security, because services should only be available for certain network areas. In general all traffic should be filtered through a firewall.

To configure the firewall to prevent foreign access and allow SSH connections (even only in local network), best following this [11].

V. Connect & Test

To test the connection to the Raspberry Pi router it is only required to set up hostapd, the firewall is rather optional. At this point another computer or smartphone is needed to connect to the above established SSID. For this purpose the test computer has to open the wireless menu. Then try to connect to the Raspberry AP.

In order to understand the process at the Pi console and to debug if necessary, it is possible to monitor the connection with the $tail$ command on the Raspberry command line.

This command outputs the last part of a file. "Print the last 10 lines of each FILE to standard output. With more than one FILE [...]." [16] To output the data while the file grows the $-f$ option after $tail$ is necessary. This needs to target to a file. To target the syslog on the Pi, the complete command is:

```
"tail -f /var/log/syslog"
```

[3]

Alternativly it is possible to look up the network settings of the test computer whether a connection to the router or the Internet has been established.

VI. Finishing up

After the connection was made and tested it is advisable to start a service (daemon) that starts our Raspberry automatically as an AP on every boot.

"A daemon is a type of program on Unix-like operating systems that runs unobtrusively in the background, rather than under the direct control of a user, waiting to be activated by the occurance of a specific event or condition.

Unix-like systems typically run numerous daemons, mainly to accommodate requests for services from other computers on a network, but also to respond to other programs and to hardware activity."[4]

In this case the event is the boot process and the background service is the start of hostapd.

VII. TOR NETWORK

A. Inception

"Tor was originally designed, implemented, and deployed as a third-generation onion routing project of the U.S. Naval Research Laboratory. It was originally developed with the U.S. Navy in mind, for the primary purpose of protecting government communications. Today, it is used every day for a wide variety of purposes by normal people, the military, journalists, law enforcement officers, activists, and many others."[8]

B. Overview

"Tor is a network of virtual tunnels that allows people and groups to improve their privacy and security on the Internet. It also enables software developers to create new communication tools with built-in privacy features. Tor provides the foundation for a range of applications that allow organizations and individuals to share information over public networks without compromising their privacy.

Individuals use Tor to keep websites from tracking them and their family members, or to connect to news sites, instant messaging services, or the like when these are blocked by their local Internet providers. Tor's hidden services let users publish web sites and other services without needing to reveal the location of the site. Individuals also use Tor for socially sensitive communication: chat rooms and web forums for rape and abuse survivors, or people with illnesses. [...]

The variety of people who use Tor is actually part of what makes it so secure. Tor hides you among the other users on the network, so the more populous and diverse the user base for Tor is, the more your anonymity will be protected."[12]

C. Why using TOR?

"Using Tor protects you against a common form of Internet surveillance known as "traffic analysis." Traffic analysis can be used to infer who is talking to whom over a public network. Knowing the source and destination of your Internet traffic allows others to track your behavior and interests. This can impact your checkbook if, for example, an e-commerce site uses price discrimination based on your country or institution of origin. It can even threaten your job and physical safety by revealing who and where you are." [23]

D. Set up TOR on Raspberry Pi

To use the Raspberry Pi with the TOR-network, a functioning Raspberry AP is required first. Once this has been set up, the next steps can be found here[9].

First the TOR-package must be downloaded. After that, just a few modifications are required in the config-File. At least little changes in the firewall $(iptable)$ have to be done.

VIII. CONCLUSION

To set up a Raspberry Pi access point or router is very simple and does not take very long even at the first time. It is much cheaper than to buy a standard router from the merchant and you are able to do much more with it. On one hand you have full access to the device and can adjust it completely to your own requirements. On the other hand you must put much more work into it to achieve an added value compared to commercial routers.

The complete configuration of the firewall is for experienced users very advantageous as they can protect the LAN very well against foreign access. Nowadays you dont know about every backdoor a programmer left to access his device or software, so building your own firewall is much safer, than trusting anyone else (if you are experienced).

A layman, who does not exactly know what to do, will maybe neglect to adjust the firewall properly and does not close the default open ports. This results in enormous reduction in security.

The very easy access to the TOR-network is a very strong advantage in privacy. The downside, however, is that the connection is likely to be slower than usual.

In my opinion the advantages outweigh to set up an own Raspberry Pi router or access point. There is an additional expense, but the benefits are greater than this. The additional security features depending on your desires can easily be integrated.

REFERENCES

[1] URL: http://www.adafruit.com/downloads/adafruit_hostapd.zip.

[2] *Aufbau von WLANs*. URL: http://einstein.informatik.uni-oldenburg.de/lehre/semester/seminar/02ss/wlan/WLANnode11.html.

[3] *Connect and Test*. URL: http://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point/connect.

[4] *Daemon Definition*. URL: http://www.linfo.org/daemon.html.

[5] *Description*. URL: http://www.computerhope.com/unix/uifconfi.htm.

[6] *HOSTAPD*. URL: http://nims11.wordpress.com/2012/04/27/hostapd-the-linux-way-to-create-virtual-wifi-access-point/.

[7] *hostapd Linux documentation page*. URL: http://wireless.kernel.org/en/users/Documentation/hostapd.

[8] *Inception*. URL: https://www.torproject.org/about/overview.html.en.

[9] *Install Tor*. URL: http://learn.adafruit.com/onion-pi/install-tor.
[10] *Introduction*. URL: http://www.myguitars.mine.nu/images/rpi_raspbianwheezy_dhcp_server.pdf.
[11] *Linux-Firewall iptables unter Raspbian konfigurieren*. URL: http://www.forum-raspberrypi.de/Thread-tutorial-linux-firewall-iptables-unter-raspbian-konfigurieren.
[12] *Overview*. URL: https://www.torproject.org/about/overview.html.en.
[13] *Private networks*. URL: http://en.wikipedia.org/wiki/IPv4.
[14] *Raspberry Pi: Statische/Feste IP-Adresse vergeben*. URL: http://jankarres.de/2013/09/raspberry-pi-statischefeste-ip-adresse-vergeben/.
[15] *Router*. URL: http://einstein.informatik.uni-oldenburg.de/rechnernetze/router.htm.
[16] *tail(1) - Linux man page*. URL: http://linux.die.net/man/1/tail.
[17] *Welcome to Raspbian*. URL: http://www.raspbian.org/.
[18] *What is a Firewall*. URL: http://whatismyipaddress.com/firewall.
[19] *What is a Raspberry Pi*. URL: http://www.raspberrypi.org/faqs#introWhatIs.
[20] *What is Network Address Translation*. URL: http://whatismyipaddress.com/nat.
[21] *What raspi-config does*. URL: http://elinux.org/RPi_raspi-config.
[22] *What you'll need*. URL: http://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point/what-youll-need.
[23] *Why we need Tor*. URL: https://www.torproject.org/about/overview.html.en.
[24] *WLAN-Konfiguration unter Raspbian*. URL: https://www.datenreise.de/raspberry-pi-wlan-einrichten-edimax/.

Inspection of the Raspberry Pi Camera Module

Marc-Hendric Lühr
Email: marcluehr@googlemail.com
Carl von Ossietzky Universität Oldenburg
Seminar Raspberry Pi inspected (WS 2013/14)

***Abstract*—The *Raspberry Pi* is getting more and more popular under interested home tinkerer. This is not only due to the low price, but also to the amount of possible applications for it. The *Raspberry Pi Foundation* released a camera for the *Raspberry Pi* which brings up more options. Unlike most other cameras, the connection is not made over a *USB* connection but to an interface of the graphical processor. This provides faster transmission rates and embedded image encoding. This article describes the *GPU*, the camera interface and the Linux integration. The integration of the image processing framework *OpenCV* into a custom application is a part of this article because of certain differences between the camera on the *Raspberry Pi* and other cameras. This work also presents an example of using the camera with *OpenCV* which is a simple laser scanner. Since most common laser range finders are expensive, a less exact setting can be constructed using the *Raspberry Pi* with its camera. In contrast to other laser range finders, this one is less expensive, but only one dimensional. In addition this article presents several first thoughts for other applications that could be a good starting point for own projects.**

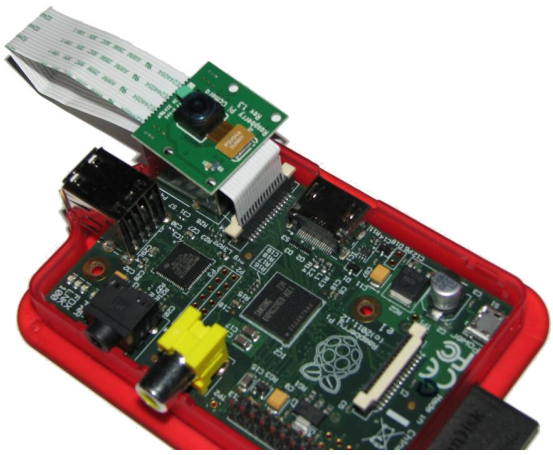


Fig. 1. A camera connected to the Raspberry Pi

I. Outline

The *Raspberry Pi* is a mobile but a powerful computation platform. The computation power allows to run effective applications using complex filtering and processing methods for images and videos. The small size allows the system to be used for unsupervised observation tasks but also other applications like image recognition. Therefore a camera is required as an image source. For the *Raspberry Pi* is a special camera available which can be connected to a plug on the *Raspberry Pi*. This plug is an interface to the *Graphical Processing Unit (GPU)*. In comparison to *USB* cameras the bandwidth is higher and images can be encoded directly by the *GPU*. Saving this step on computation time on the *Central Processor Unit (CPU)* leaves more time to other tasks.

The amount of applications for cameras is huge but increases by combining the camera with other interfaces, devices and other enhancements. An important step to enhance the mobility is using batteries as power supply. Also the connectivity can benefit from the interfaces provided by the *Raspberry Pi*. Parts of a distributed system can be connected by using the wired networking, a wireless *USB* dongle or a *WiFi SD-Card*. If it is not required to transmit images or videos but messages or measurement values, I2C or SPI can be suitable for the application. These interfaces are available through the *General Purpose Input/Output (GPIO)*. Also buttons, lamps, relais and other peripherals can be connected to the *GPIO*.

The task of the project group "MIPSwarm" [1] is to build a robot driving with two wheels. In robotics an often used sensor is a laser range finder but common laser range finders are too expensive for the project budget. So the group tried to build an ultra low cost laser range finder with a simple laser pointer and a camera using triangulation. Unfortunately the developing time on the given platform took too long that the development was cancelled. But using the *Raspberry Pi* for this applications brings up more computation power and flexibility for operating systems and drivers. The applications behaviour should be like a sensor, so the measurement values needs to be read by another device. I2C or SPI will be suitable since only text data will be exchanged. Also the laser pointer needs to be switched on and off. Therefore a GPIO pin will be used.

For this application an image processing environment is required. A popular framework is the *OpenCV* library. There are implementations in *C/C++*, *Java*, *Python* and many more. Since a resource efficient operating system is required to leave maximal computation time over for processing images, this work concentrates on an operating system with a minimized *Linux Kernel*. The *Archlinux* distribution is used as operating system for the *Raspberry Pi*. The main advantage is the simple software building system and the good community support for ARM based systems. There is already a special kernel for ARM available but it is also possible to build an own kernel which can be rebuild automatically to update the system [2].

After the outline an overview of related work on this topic is shown. This work consists of three parts followed by

a conclusion. The first part describes the camera hardware with interfaces and transmission protocols. The *GPU* is also described since the camera is connected to it. The second part shows the basic techniques to work with the camera and *OpenCV*. Therefore the source code of the example application provided by broadcom will be inspected and extended as an interface to *OpenCV*. Afterwards the example application of a triangulating laser range finder is shown. The expected result is a working laser range finder but the results may not be perfect accurate. Therefore the example will be extended by a further work section to show improvements of the experimental setting. Later on other example ideas for applications will be shown followed by a section that explains the advantages of an implementation of these examples on the *Raspberry Pi*. At last a conclusion finishes this work.

A. Related work

The first occurrence of a camera for Raspberry Pi was found in an article in the *MagPi* magazine [3]. The camera for the *Raspberyy Pi* is available since July 2013, so there are no scientific articles about this topic. Most of the knowledge about the camera, the connection to the Raspberry Pi and the *OpenCV* integration can be found in internet forums, blogs and different data sheets [4], [5], [6], [7].

The article in the *MagPi* magazine describes how to connect the camera to the Raspberry Pi and to capture still images or videos. The two applications for these features are *raspistill* and *raspivid* which are part of the *Raspberry Pi Userland* applications [8].

There is a blog article with a guide to adapt these applications to use them as image source for *OpenCV* [4]. This is the basic guideline to connect the camera to *OpenCV*.

An interesting application for a camera system comes from a laser range finder used in *Naetos* vacuum cleaner robot. *Naeto* supported a research for laser range finder for less than $30 [9]. The principle of this laser range finder is inspected on an internet blog [10]. This laser range finder does not have any dynamic optic. It has only a static mounted infrared laser pointer and an infrared image sensor. This article also provides the basics for the construction of the example application.

II. Camera hardware and interfaces

The camera can be connected directly to the *GPU* so all applications require to work with it. Therefore this section outlines the *GPU* and its interfaces. Afterwards the camera is described.

A. The VideoCore 4 GPU

The *VideoCore* is a multimedia processing architecture which is part of the System-on-Chip ARM on the *Raspberry Pi*. It has an embedded *Digial Signal Processor (DSP)* which is able to encode and decode video streams up to 1080p with 30 frames per second. The coding is done by a specialized processor which performs common used mathematical operations and algorithms. The processor is completely software programmable. This design decision has many advantages in comparison to a hardware accelerated design. The architecture is flexible to be able to add new features by adapting the software on the processor. The scalability is also higher, the software can be optimized and the clock speed can be decreased. It is also more efficient because the whole architecture is used while not all accelerators are used at the same time in hardware accelerated architectures. In comparison to hardware accelerated design the power consumption of the *VideoCore* is lower due to the clock speed [7]. A closer look into the processor would be interesting but there are no details or data sheets published for copyright reasons. The basic interface for the camera is the *Advenced Image Sensor Pipeline (ISP)* which can get up to 220 megapixel per second from an external camera. A protocol need to be defined above the *ISP* to adapt the camera properties to the pipeline architecture. For the *Raspberry Pi* this step is done by the *Camera Serial Interface*.

B. Camera Serial Interface 2 (CSI2)

"The Camera Serial Interface 2 specification defines an interface between a peripheral device (camera) and a host processor" [11]. It provides an industrial standard which needs to be implemented by the camera and the signal processor. The pixel data are exchanged by a fix amount of data lines plus a clock line between the camera and the signal processor. The *Camera Control Interface (CCI)* is used to exchange configuration and control commands or requests. The *CCI* is similar to the fast *I2C* interface standard. The receiver is the master to control and configure the camera. The camera is a slave. It receives commands from the master and sends answers to requests from the master.

Figure 2 shows the layers of the interface protocol. The image data and the control commands are transmitted separately. The camera on transmitter side converts the pixel data to a byte stream to send the image through the *physical layer (PHY)* to the *Raspberry Pi* on receiver side. On the receiver side the *GPU* of the *Raspberry Pi* decodes the byte stream to process the image. The pixel format can be changed to increase the efficiency depending on the amount of parallel data lanes and the further application. By default, the pixel data is not encoded. The raw values of each of the three colours are represented by six bits. The camera has ten parallel data lanes, so one pixel can be transmitted in two transmission cycles. Also many other image and colour formats are supported by the camera firmware but the protocol allows to define own pixel data formats. The next sections shows a compatible camera. These camera must implement *CSI2* and must fit into the connector on the *Raspberry Pi* board.

C. Compatible Cameras

Currently the *OmiVision OV5647* is the only existing compatible camera. There are two versions, the first one is a normal camera and the other one is the same model but with a removed infrared filter. The advantage of the second camera is the ability to capture images or videos in darkness with infrared light. It also allows to observe the photosynthesis of plants and other infrared light emitting objects. The *Raspberry*

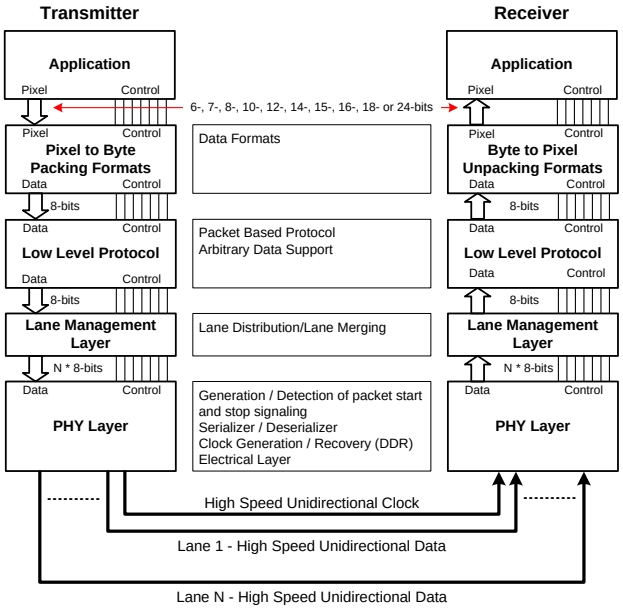


Fig. 2. Layer architecture of CSI2 [11]

determine the image properties and the pixel format. So the serial data can be decoded without signals like *VSYNC* or *HREF*. This saves programming effort to handle the interrupts for *VSYNC* and *HREF* but the frame rate defined by *CSI2* is still reachable.

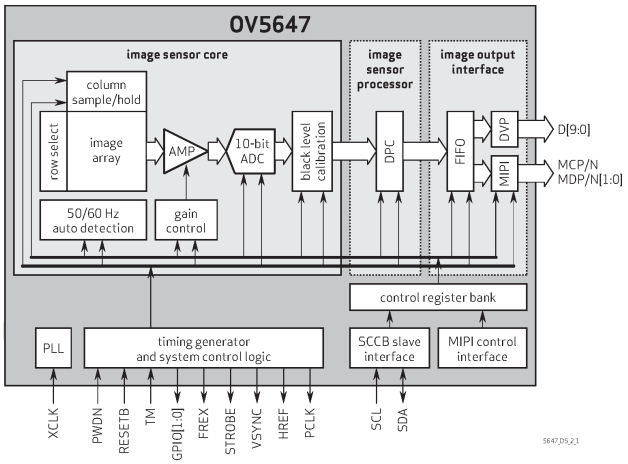


Fig. 3. Block diagram of the OV5647 pin conntectors [6, p. 19]

Pi Foundation developed an adapter for this camera from a thin sensitive foil cable to a more stable cable fitting into the camera connector on the *Raspberry Pi*. The camera can capture images up to five mega pixel and videos up to *1080p*. The image sensor has automated black calibration and the integrated image processor performs lens correction, white balance, and defect pixel cancellation [6, p. 47-54].

The block diagram in Figure 3 shows the intern parts of the image creation and the external interfaces. The image sensor core contains an image array which has micro lenses in front of each pixel. The luminance of each colour is measured by a light sensitive component. The voltage through the component gets amplified to use the full range of the analog-to-digial converter. The next step is the black level calibration. Then, the image gets processed and transmitted over an external interface.

The camera needs an external clock signal which is generated on the adapter board. Commands and configurations can be sent and received over an *SCCB interface* which is basically an *I2C* slave interface. In *CSI2* this interface is used as *CCI*. There are ten parallel pin connectors to transmit ten bits of the image data per clock step. To determine if one row of the image is completely transmitted and later the complete image. The receiver evaluated these output pins from the timing generator. *VSYNC* is triggered for each complete image and *HREF* for a complete transmitted row. The *PCLK* is the clock signal for the serial image data. Using these connectors is the recommended way to access the image data but the *CSI2* only uses a small amount of these pins. As shown in Figure 2, only the clock *PCLK* and the data pins are connected to the *Raspberry Pi*. The other pins are connected to the adapter board or left unconnected. Using the *CCI* the receiver can

Apart from the special camera of the *Raspberry Pi*, every Linux compatible camera with *USB* 2.0 connector can be used. Unfortunately such a camera is slower than the special *Raspberry Pi* camera because the *GPU* cannot be used to encode the images. The camera is sold ready to use and just needs to be plugged into the port in the right direction.

III. Creating Camera Applications

This section describes the software to capture images and videos. The camera is not represented as a block device in *Linux*. The *Raspberry Pi* firmware tools provide two applications to capture images and videos with the camera. Still photos can be taken with *raspistill*. The other application is *raspivid* to record a video or send it to the standard output. Both applications are open source software and the source code is available [8]. The further work in this paper is based on this source code. The first goal is to access the camera with *OpenCV*. The *OpenCV* manual [12, p. 3] advises to access the camera as a block device but this requires changes of the graphic kernel driver which is too extensive for this work.

A. Inspecting the Raspistill Application

The *raspistill* and *raspivid* applications are built for command-line use. A guide to capture photos and videos can be found by reading the help page. Each of both applications are written in one *C* file with a simple main function. Only the *raspistill* application is inspected in this work. The *raspivid* is similar but the *raspistill* is easier to read and to adapt.

The *raspistill.c* contains the whole source code to capture still pictures and to write them back to a specified file. A closer look into the source code gives a better impression about accessing the camera from an own application. At first, a quick overview over the functions and includes can

be taken. Some functions are used to parse command-line parameters. Other functions are used to set image properties and several helper functions. The functions to create and destroy components as well as the connect port function are described later. The first include is the *bcm_host.h* file. *BCM* seems to be an abbreviation for *Broadcom*; the manufacturer of the *System-on-Chip (SoC)*. The header provides access to core functions of the system. The *vcos* interface provides access to the *VideoCore Operating System*. The *bcm_host* and the *vcos* interface need to be initialized before using the *VideoCore API* [13]. The *VideoCore API* is still at a low programming level. Therefore, the *MultiMedia Abstraction Layer (MMAL)* is used between the *VideoCode API* and the user application as another abstraction level. The next section described this abstraction layer.

B. Multimedia Abstraction Layer (MMAL)

The *MMAL* abstracts the low level access to components of the *VideoCore*. The framework is organized in components like the components of the *VideoCore*, so it is easy to extend. Examples for these components are the converter, the decoder, the encoder, the renderer, and the camera itself. *MMAL* manages the connection between the *CCI* and the configuration of the camera settings. The data stream of the components can be connected to their ports. For example, the output of the camera can be connected to the input of the image encoder. Three components are used for the *raspistill* application. The first one is the camera. The second one is an image encoder to encode the raw image data to a specified file using a special hardware unit, so the format conversion must not be done by the processor. The last component is a preview component. Because it is also used in the *raspivid* application, the component creation and destruction is in another .c file. The preview opens a window on the screen which shows the video steam from the camera. Now all relevant parts of the *raspistill* applications are explained. Removing unnecessary functions and cleaning up the code is the next part to create own applications.

C. Beginning an own project

The base of an own project is the *raspistill.c* file. First, all command-line parsing belongings can be removed. Also the preview can be removed from the main function. Now, the custom code needs to be inserted into the existing code. The *encoder_buffer_callback* is called when an image is completely encoded. At this point the application writes the image to a file. The image can be passed to *OpenCV* instead of the saving the image to a file. Therefore the captured image needs to be converted to a matrix. Another option is to create a virtual block device on the output of an application and configure *OpenCV* to use the device. The block device needs to be emulated by sending the images as a binary stream. But this causes more programming effort and less performance, so this work uses the first method.

D. OpenCV image processing

OpenCV is a free image processing library written in *C/C++*. It contains functions and algorithms to process and analyse images. *OpenCV* has "a strong focus on real-time applications" [14]. The functions are optimized and runtime efficient but for the *Raspberry Pi* processor some of them might take too much computation time for real-time image processing. The computation effort can be reduced by capturing only necessary image data. A grayscale image is good enough for acceptable results for most purposes. Therefore the source image is coloured and can be converted in different ways to a grayscale image. Some conversion can be made by selecting the image format. But depending on the application other parts of the image data can be reduced or left out. If only one colour is relevant for object recognition, only one colour channel can be used from an *RGB* image. The data source for *OpenCV* can be an image or video from file or a camera. For using a camera, a block device representation by the operation system is required. However the *Raspberry Pi* camera is not represented as one.

IV. An Example: The *Raspberry Pi* Laser Range Finder

Laser range finders can use different methods to determine the distance to an object. Common methods are phase shift measurement or light runtime measurement. Beside the measurement methods laser range finders use a dynamic optic like moving lenses and mirrors to extend the measurable area. These methods are quite complicated and time sensitive. Using these methods causes high costs for laser range finders. A simpler and less expensive method is using laser triangulation with a camera and a laser pointer. The optic is static and the measurement area can only be extended by rotating the whole laser range finder. The laser pointer can be switched on and off over a *General Purpose Input/Output (GPIO)* port. Since the *GPIO* port of the *Raspberry Pi* does not have enough power to supply the laser pointer, a transistor circuit is required to toggle the laser pointer. The principle of triangulation will be described in the next section. Afterwards the physical setting of the experiment will be shown.

A. Triangulation

This section describes the theoretical background for the triangulation. This method uses triangle calculation. The calculation uses some variables which names angles, edges and other parts of the triangle in Figure 4. These variables are:

- a, b and c are the sides of the triangle. c is known from the physical setting of the mount. a and b depends on the distance to the object.
- d is the distance from the central view point of the camera to the reflection of the laser beam.
- e is the distance from the camera to the focus point. The distance is known from the camera characteristic or can be determined by sample measurements.
- h is the distance to the object and also the height of the triangle.

- α is the fix angle between laser pointer and camera
- β is a corresponding angle in the camera between d and the extension of b

The goal for the calculation is to determine h.

A laser emitter and a linear laser detector are placed in a solid distance and angle. Figure 4 shows the principle with a laser pointer adjusted to a fix angle (α) while the angle from the camera is 90°. In the following calculation the pixel distance is represented by d. With these values the angle in the small triangle in the camera can be calculated as follows:

$$\beta = tan(\frac{e}{d}) \tag{1}$$

Angle β is a corresponding angle in the triangle to the object. The height of the triangle can be calculated by sine rules.

$$\frac{a}{sin(\alpha)} = \frac{b}{sin(\beta)} = \frac{c}{sin(\gamma)} \tag{2}$$

Since α, β, γ (from angle sum) and c are known, every side of the triangle can be calculated. Side a can be calculated as follows.

$$a = \frac{sin(\alpha) \cdot c}{sin(\gamma)} \tag{3}$$

The distance to the object can be calculated using the trigonometric cosine function.

$$h = cos(\beta) \cdot a \tag{4}$$

Combining equation (1), (2), (3) and (4) lead to the following equation:

$$h = cos(\beta) \cdot \frac{sin(\alpha) \cdot c}{sin(180 - \alpha - tan(\frac{e}{d}))} \tag{5}$$

The accurateness of this calculation depends on the accurateness of the mount where the laser pointer and the camera are placed on. Since this accurateness cannot be reached in this work, the triangulation is supported by sample measurements. An object is set to specific distances and the resulting values are saved as calibration values. The values will be used backward in equation (5) to correct the variance from angle β. These calibration results in a linear correction of the measured height h as gain and offset.

$$h_{cal} = cal_{gain} \cdot h + cal_{offset} \tag{6}$$

B. Physical setting

The camera is connected to the camera port on the *Raspberry Pi* and the laser pointer is connected to a simple transistor circuit. A standard *NPN* Transistor is used to control the laser pointer. The base of the transistor is connected through an external pull-down resistor to a *GPIO* pin. The resistor is used to protect the *Raspberry Pi* from using too much power while the internal pull-down resistor is not configured properly.

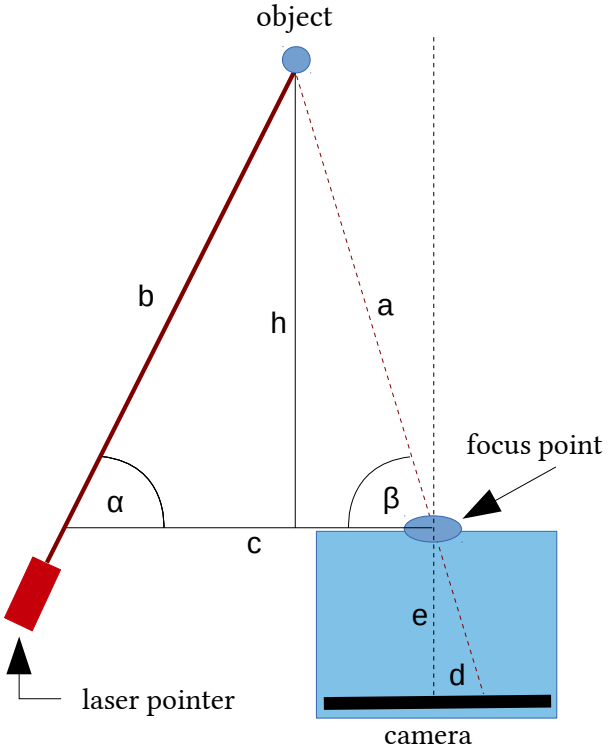


Fig. 4. Principle of laser triangulation

GPIO pin 17 is used in the experimental setting because it does not overlap with another relevant alternative function.

Figure 5 shows the provisional mount for the laser pointer and the camera. Like in Figure 4, a simple laser pointer is set in a fix angle to the central view point of the camera. The camera captures the reflection of the laser beam and evaluates the position.

Fig. 5. Provisional mount for the camera and the laser pointer

C. Implementation

The Figure 6 shows the control flow of the application computing the distance h and controlling the laser pointer. The implementation is inserted into the project created in section III-C. The major task is to locate the laser dot reflection in the image by analysing it. The colour of the laser beam is read. Therefore the grayscale image is created by subtracting the maximum of the green and the blue channel from the red channel. Other colours which are mixed from different colours, like white, yellow, or magenta, are reduced in this step. The laser pointer is switched on after receiving the first picture. After the second picture, the laser pointer will be switched off and the point with the maximal difference between these two pictures is the pixel value used in equation (5) as d. The last step is to apply the calibration equation (6) on the calculated distance. The output is sent to the standard output *stdout*.

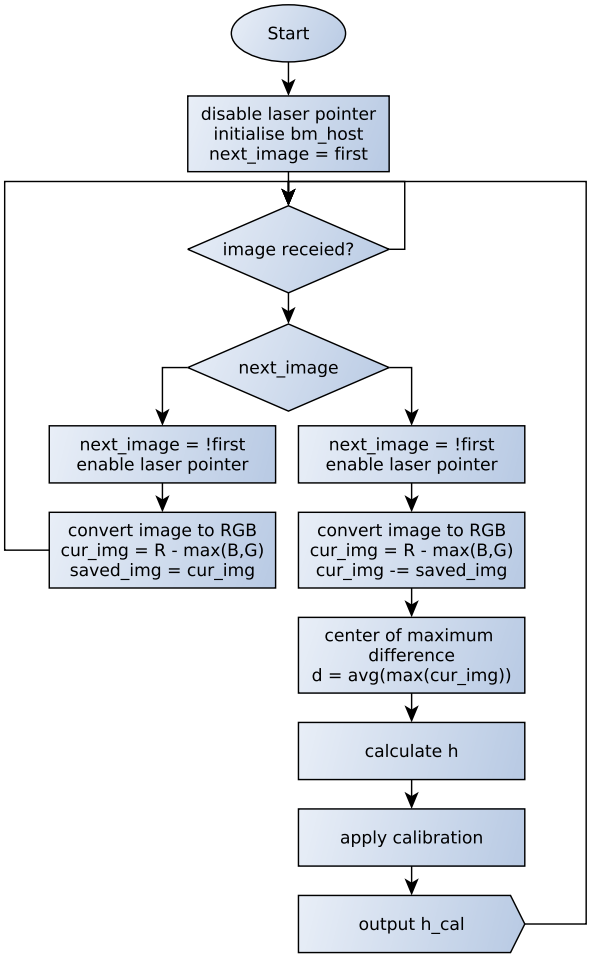


Fig. 6. Flow diagram of the implementation

The maximum frame rate halves because at least two pictures are used to perform one distance measurement. The maximal frame rate of the camera is 30 frames per second which would result in 15 measurements per second. However, the single picture capturing allows only 1-4 measurements per second.

D. Raspberry as black-box system with data interface

A laser range finder is often embedded in other systems like robots or cars. Most normal laser range finders provide an *I2C* or *SPI* interface. This system can also be extended with these interfaces by using a *GPIO* port alternative. There are already ports reserved for these interfaces and working examples are available. There is an article in the *MagPi* magazine about the *I2C* [15] interface. In addition to the interface, the system needs to start automatically while powering up. Unnecessary services can be disabled to decrease the start-up time. The system should perform a showdown sequence to avoid data loss. Since this is not usually used in embedded systems data loss can also be avoided by using a write protected SD card.

In a final step, the laser range finder could be built as a special purpose program and without a Linux operating system. The special purpose program could reach a higher measurement rate but would also increase the system design complexity. But since the *MMAL* abstracts the hardware access to the *VideoCore* the applications have no strong bindings to the Linux kernel.

E. Further development

The current implementation adapts the *raspistill* application to pass images to *OpenCV*. Like in [4, Step 6], the *raspivid* application is faster for image streams and can reach a frame rate of 30 frames per second. However, the maximum resolution is still limited by the computation effort caused by the image processing. Rising the resolution would cause *OpenCV* to drop images which would decrease the frame rate if the current image processing step is not finished. Also the encoding in the camera can be enabled so that the image does not need to be encoded in *OpenCV*.

The laser beam in this example is visible for humans and can be replaced by an infrared laser pointer. An infrared laser beam is not visible but can still injure the eye. Therefore, the impulse duration of the activated laser pointer can be reduced by speeding up the camera. Speeding up the camera also allows to perform more measurements. If the measurement rate is high enough, the laser range finder can be rotated and stopped shortly to perform a two dimensional measurement.

Also a more accurate mount for the laser pointer and the camera needs to be constructed. A finer and more solid mount would calculate more exact results. Also turning the laser pointer to a different angle will change the measurable range. At last a lens behind the camera can optimize the resolution of the laser range finder.

V. Discussion

The major advantage of the Raspberry Pi is the camera connection to the *GPU*. It can achieve a higher frame rate and resolution then a *USB* device on a comparable platform. The camera is small and can be set up in a box or a mount as the user wants it.

Another advantage is the simplicity of the hard- and software. The user still has the raw hardware but does not need special knowledge to get a running setting. Also the software is easy to use and easy to extend. The *raspistill* and *raspivid* application can be read by most hobby programmers and are easy to use for own applications. So the platform has a high potential to meet the requirements for users to build their own projects. Also the price is an advantage in comparison to other camera systems. The costs of the camera and the *Raspberry Pi* are both about $30. Benefiting are also the available interfaces. On the one hand, there is a *GPIO* port for a *I2C*, *SPI* or other data bus for embedded systems. On the other hand, *Ethernet* or *USB* can be used to send images and results to other computers. At least, an analog composite video output through a *Chinch* plug or a digital video output via *HDMI* is useful to display the results of the image processing. The rest of this section shows other ideas which benefit from the shown advantages.

An interesting project in the gaming sector is the *RetroPie* project. This project basically brings an emulator for different older gaming platforms like the *Super Nintendo*. Motion capturing can be used as an input method instead of a gaming controller. A Super Nintendo controller has eight buttons and a directional pad but these keys are used differently in different games. So special gestures need to be assigned to the controlling of each game separately.

It would also be interesting to use two cameras to generate a stereoscopic image but the *VideoCore* is designed to be connected to one camera. A temporary workaround can be done by attaching an *USB* camera or using two *Raspberry Pi*'s. The last option could also be used to just pre-process the images and transmitting them to a more powerful computer to perform a 3D analysis.

Another application which follows from the description of the infrared camera module [16] could be the observation of the photosynthesis of plants using the camera without infrared filter. The power consumption and possible storage extension is well suited to be left for several days. This is also adaptable to other observation tasks like a security camera or a door observation camera which can be connected via Ethernet.

A. Conclusion

The camera for the *Raspberry Pi* is the base for many applications. The interfaces between the GPU and the camera simplify the access to the camera data in embedded systems but also lead in disadvantages for the camera access in Linux. The block device representation is common but is not supported by the GPU and the graphic driver. A workaround is made with adapting the example applications from Broadcom [8]. Different applications can be adapted depending on the required frame rate. The base for own applications is given by embedding the own programm into the existing source code.

The example application shows the camera access for a laser range finder in combination with the GPIO port control. The implementation experiment shows that the concept is working. Although, the results are fine but they can be enhanced by increasing the precision of the mount and the calculation.

The discussion has shown that the advantages of the *Raspberry Pi* with a camera lead to interesting ideas for applications and the future will show if some of these ideas or other applications for the camera show up.

REFERENCES

[1] "Projektgruppe MIPSwarm - Mobile Inverted Pendular Swarm," [accessed 03.01.2014]. [Online]. Available: http://www.uni-oldenburg.de/mipswarm/

[2] archlinux.org, "Kernels," [accessed 03.01.2014]. [Online]. Available: https://wiki.archlinux.org/index.php/Kernels

[3] J. Hughes, "The Raspberry Pi camera - part 1," November 2013, [accessed 07.11.2013]. [Online]. Available: http://magpi.techjeeper.com/The-MagPi-issue-14-en.pdf

[4] "OpenCV and Pi Camera Board," May 2013, [accessed 07.11.2013]. [Online]. Available: http://thinkrpi.wordpress.com/2013/05/22/opencv-and-camera-board-csi/

[5] B. Corporation, "BCM2835 ARM Peripherals," 2012, [accessed 07.11.2013]. [Online]. Available: http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

[6] OmniVision, "OV5647 Datasheet," 2009, [accessed 07.11.2013]. [Online]. Available: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/RaspberryPi/ov5647_full.pdf

[7] B. E. Ltd., "VideoCore," [accessed 03.01.2014]. [Online]. Available: http://www.broadcom.com/products/technology/mobmm_videocore.php

[8] ——, "ARM side libraries for interfacing to Raspberry Pi GPU," [accessed 03.01.2014]. [Online]. Available: https://github.com/raspberrypi/userland

[9] K. Konolige, J. Augenbraun, N. Donaldson, C. Fiebig, and P. Shah, "A low-cost laser distance sensor," in *Robotics and Automation. ICRA. IEEE International Conference*, 2008, pp. 3002–3008.

[10] T. Deyle, "Ultra Low-Cost Laser Rangefinders Actualized by Neato Robotics," Dezember 2009, [accessed 07.11.2013]. [Online]. Available: http://www.hizook.com/blog/2009/12/20/ultra-low-cost-laser-rangefinders-actualized-neato-robotics

[11] MIPI Alliance, Inc, "MIPI Alliance Specification for CSI-2," Deceber 2009, [accessed 07.11.2013]. [Online]. Available: http://electricstuff.co.uk/temp/mipi_CSI-2_specification_v01-01-00_r0-05.pdf

[12] opencv.org, "The OpenCV Reference Manual," [accessed 03.01.2014]. [Online]. Available: http://docs.opencv.org/opencv2refman.pdf

[13] elinux.org, "Raspberry Pi VideoCore APIs," [accessed 03.01.2014]. [Online]. Available: http://elinux.org/Raspberry_Pi_VideoCore_APIs

[14] opencv.org, "OpenCV Open Source Computer Vision," [accessed 03.01.2014]. [Online]. Available: http://opencv.org/

[15] B. E. Hall, "Part 1: Introduction to Pi Matrix and programming the I2C bus," June 2013, [accessed 03.01.2014]. [Online]. Available: http://www.themagpi.com/topic/i2c/

[16] "RASPBERRY PI CAN :: Raspberry Pi - NoIR Kamera," [accessed 26.01.2014]. [Online]. Available: http://www.reichelt.de/Programmer-Entwicklungstools/RASPBERRY-PI-CAN/3/index.html?&ACTION=3&LA=3&ARTICLE=139119&GROUPID=5514

RPI fährt: Motoren, Anbindung, Ansteuerung und Programmierung

Jenny Inge Röbesaat
Carl von Ossietzky Universität Oldenburg

I. Einleitung

In den letzen beiden Jahren hat der Raspberry Pi den Markt immer weiter erobert und ist zu einem beliebten Minicomputer, sowohl für Anfänger als auch für Fortgeschrittene, geworden. Besonders seine vielfältigen Anwendungsmöglichkeiten machen ihn so interessant. So lassen sich mit dem Raspberry Pi und dessen eingebauten GPIO-Schnittstelle verschiedene Sensoren und Aktoren steuern [1]. Der Schwerpunkt dieser Seminarausarbeitung wird die Anbindung von Motoren an den Raspberry Pi, die Ansteuerung der Motoren und dessen Programmierung sein. Um dieses genauer zu untersuchen, wurde ein (rudimentäres) Automodell erstellt. Dieses Automodell soll in der Lage sein, auf eine Ansteuerung zu reagieren und vorwärts und rückwärts fahren zu können. Weiterhin soll es sich links- und rechtherum drehen können.
Für die Untersuchung der Anbindung und Ansteuerung der Motoren an den Raspberry Pi und für die Umsetzung des Automodells waren verschiedene Schritte notwendig. Zunächst wird in Abschnitt II die Anwendungsmöglichkeiten des Raspberry Pi in Bezug auf diese Ausarbeitung beschrieben und mit anderen Alternativen vorab verglichen. Im darauf folgendem Abschnitt III wird kurz die Funktionsweise eines Motors dargestellt und die Auswahlfindung zu den benutzen Motoren dargelegt. Anschließend wird die Anbindung der Motoren an den Raspberry Pi in Abschnitt IV beschrieben. Dabei wird unteranderem auch auf die benutzte GPIO-Schnittstelle eingegangen. In Abschnitt V wird kurz die Konstruktion des Automodells vorgestellt. Um die Motoren ansteuern zu können, werden im Abschnitt VI zwei verschiedene Ansteuerungsmethoden und dessen Programmierung beschrieben. Die erste untersuchte Ansteuerungsmöglichkeit ist die Ansteuerung der Motoren über die Tastatur. Die zweite untersuchte Ansteuerungsmöglichkeit ist die Ansteuerung mit dem von Nintendo entwickeltem Wii Nunchuks. Abschließend folgt in Abschnitt VII eine Darstellung der Ergebnisse sowie zukünftige Erweiterungsmöglichkeiten.

II. Raspberry Pi

Der Raspberry Pi ist ein kreditkartengroßer Minicomputer, der aufgrund seiner vielfältigen Anwendungsmöglichkeiten und seines günstigen Anschaffungspreises besonders bei Bastlern beliebt ist. Das Kernstück des Raspberry Pis ist ein ARM Prozessor mit 700MHz. Strom bezieht er über ein MikroUSB-Netzkabel. Der Raspberry Pi besitzt eine GPIO-Schnittstelle (General Pupose Input / Output Interface), mit der man verschiedenste Aktoren und Sensoren steuern kann und ist daher sehr gut für Elektronikexperimente geeignet [2].

III. Motoren

Ein Motor ist eine antreibende Maschine, die verschiedene Energieformen, wie z.B. elektrische oder thermische Energie in Bewegungsenergie umwandelt und aus dessen Bewegung mechanische Arbeit verrichtet. Es gibt verschiedene Arten von Motoren. Die am häufigsten verwendeten Motoren sind der Verbrennungsmotor und der Elektromotor [3].Ein weiterer bekannter Motor ist die Gasturbine. Diese kommt häufig in der Luftfahrt, Schifffahrt und in Elektrizitätswerken zum Einsatz. Eine Gasturbine wandelt die chemische Energie von Gasgemischen in Bewegungsengerie um. Ihr Wirkungsgrad ist höher als bei vielen anderen Motoren, besonders bei sehr heißen Gasen. Da heiße Gase (Temperaturen von über 1000C) nicht nur gefährlich, sondern auch schwer zu lagern sind und die Anschaffungkosten zudem sehr hoch sind, ist eine Gasturbine ungeeignet für den Modellbau [4]. Der nächste Motor ist der Verbrennungsmotor. Dieser wandelt die chemische Energie von Kraftstoff ebenfalls in Bewegungsenergie um. Jedoch sind auch hier die Anschaffungskosten höher als bei anderen Motorarten. Hinzu kommt der immer weiter steigende Ölpreis und die immer strenger werdenden Emisionsrichtlinien. Zudem ist der Wirkungsgrad niedriger als z.B. bei der Gasturbinen. Daher ist der Verbrennungsmotor auch nicht geeignet für den Modellbau[5, S.V],[6].

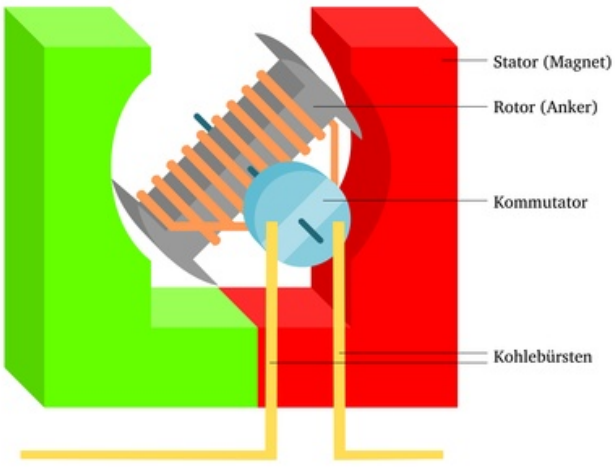


Abbildung 1. Grundaufbau eines Wechselstrom-Elektromotors [16]

Der am meisten für den Modellbau eingesetzte Motor ist der Elektromotor. Dieser hat einen einfachen Aufbau und dessen Anschaffung- und Betriebskosten sind sehr gering.

Für das Automodell werden Gleichstrommotoren verwendet. Gleichstrommotoren können im Gegensatz zu Wechselstrommotoren einfach mit Batterien betrieben werden und werden daher sehr häufig im Modellbau verwendet [8].

Ein Gleichstrom-Elektromotor besteht aus zwei Grundelementen, dem Stator und dem Rotor. Der Stator ist meist ein unbeweglicher Permanentmagnet, der ein Magnetfeld um sich selbst erzeugt. Der Rotor besteht aus einer freibeweglichen Spule mit einem Eisenmagnet, der auch als Anker bezeichnet wird. Abbildung 1 zeigt den Grundaufbau eines Gleichstrom-Elektromotors. Dort befindet sich der Rotor im Innerem des Stators. Über dem Kommutator und den Kohlebürsten wird der Rotor mit Strom versorgt. Dabei ist der Kommutator für die Umpolung des Stromes verantwortlich. Sobald der Rotor mit Strom versorgt wird, erzeugt auch dieser ein Magnetfeld. Dieses Magnetfeld steht in Wechselwirkung mit dem Magnetfeld des Permanentmagnetes. Ohne die Umpolung des Stroms durch den Kommutator würde sich der Rotor so ausrichten, dass das Magnetfeld des Rotors mit dem Magnetfeld des Permantentmagnetes gleichgerichtet wäre und der Rotor würde sich nicht weiter bewegen. Nach der Umpolung stoßen sich die Pole des Rotors und des Stators erneut ab. Durch diesen sich wiederholenden Vorgang ensteht eine Drehbewegung [17][18].

Da ein normaler Gleichstrom-Elektromotor eine sehr hohe Umdrehungszahl pro Zeiteinheit und ein niedrigeres Drehmoment hat, ist dieser aufgrund der hohen resultierenden Geschwindigkeit weniger geeignet für Modelautos. Ein zu schnelles Modelauto lässt sich wenn überhaupt nur schwer steuern und kontrollieren. Daher wird ein Getriebemotor verwendet. Ein Getriebemotor ist in den meisten Fällen ein Elektromotor mit einem vorgeschaltetem Getriebe. Das Getriebe ist ein Maschinenelement, das Bewegungsgrößen ändern kann. Bei den verwendeten Getriebemotoren wird die Umdrehungszahl pro Zeiteinheit verkleinert und dafür das Drehmoment vergrößert.

Eine wichtige Größe für ein Getriebemotor ist das Drehzahlverhältnis. Dieser ist ein Quotient aus der Antriebsdrehzahl und der Abtriebsdrehzahl. Auch das Drehmoment im Abtrieb ist eine wichtige Kennzeichnung für einen Getriebemotor[9].

Die beiden verwendeten Getriebemotoren haben eine maximale Drehzahl von 230 Umdrehungen pro Minute und ein maximales Drehmoment von 5,5kg*cm. Damit sich die Motoren bewegen, benötigen diese einen Strom zwischen 160mA-240mA und eine Spannung zwischen 3V-6V. Zu dem Motor werden passende Räder verwendet (Abbildung 2).

IV. Motoranbindung

Da der Raspberry Pi nur eine Spannung von 3.3V bzw. 5V liefern kann und andere Bestandteile auch Spannung benötigen, aber die ideale Versorgungsspannung bei 6V liegt, wird eine externe Spannungsversorgung benötigt. Dazu werden vier 1.5V Batterien verwendet. Um die Motoren an den Raspberry Pi anzubinden und anschließend ansteuern zu können, benötigt man die GPIO-Anschlüsse (Abbildung3).[11, S. 197-199].

Abbildung 2. Einer der verwendeten Motor mit Rad[7]

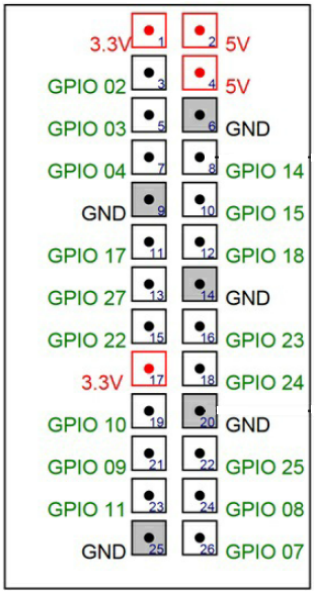


Abbildung 3. GPIO-Schnittstelle [11, S.201]

Bei dem verwendeten Raspberry Pi Model B Revision 2 sind 17 dieser Anschlüsse frei programmierbar. Die restlichen 9 Anschlüsse sind Ground-, 3.3V- oder 5V- Anschlüsse. Von den frei programmierbaren Anschlüssen sind auch zwei Anschlüsse für den IC Datenaustausch geeignet (SDA: GPIO 02 und SCL: GPIO 03). Diese beiden Anschlüsse werden später für die Nunchuk-Anbindung verwendet. Alle frei programmierbaren Anschlüsse können mittels Software auf LOW oder HIGH geschaltet werden.

Versorgt man nun die Motoren mit Spannung, so können sie sich nur in eine Richtung drehen. Um die Richtung ändern zu können, wird eine H-Bridge verwendet.

In Abbildung 4 ist der Schaltplan einer H-Bridge zu sehen. Bei der H-Bridge können verschiedene Transistoren angesteuert werden, wodurch sich die Drehrichtung der Motoren ändern kann. Der blaue bzw. rote Weg gibt jeweils die Stromdurchflussrichtung und damit die Drehrichtung der Motoren an. Um eine H-Bridge-Schaltung zusammen zu bauen, müssen

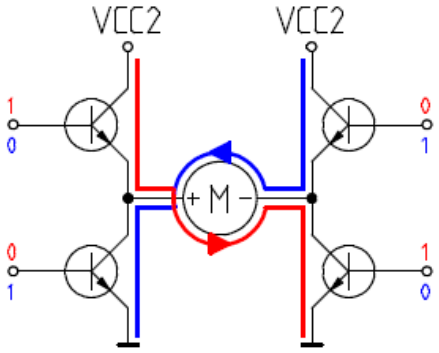


Abbildung 4. H-Bridge [12]

verschiedene Bauteile verlötet werden. Zudem ist die Kurzschlussgefahr einer einfachen H-Bridge-Schaltung sehr hoch [12]. Aus diesen Gründen wird der Motortreiber L293DNE verwendet. Dieser enthält intern zwei H-Bridges, so dass damit zwei Motoren in beide Richtungen betrieben werden können [11, S.668, 669]. Abbildung 5 zeigt die Anschlussbelegung der 16 Anschlüsse des Motortreibers.

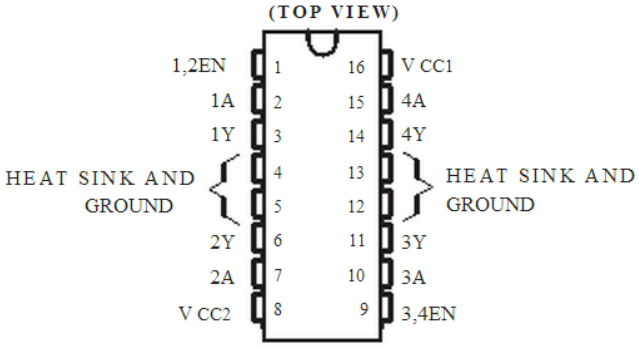


Abbildung 5. Motortreiber L292DNE [14]

Es gibt vier Treiber, welche jeweils einen Eingang und einem Ausgang besitzen. Die vier Treiber sind mit den Zahlen 1-4 durchnummeriert. Die Eingänge der Treiber sind mit den Buchstaben A gekennzeichnet und die Ausgänge mit dem Buchstaben Y. Ein Motor braucht zwei Treiber um in beide Richtungen betrieben zu werden. Die Eingänge der Treiber stellen die Eingänge der H-Bridge dar. Um die Treiber benutzen zu können, müssen diese freigeschaltet werden. Dies geschieht mit dem Enable-Anschluss EN. Die Enable-Anschlüsse schalten jeweils zwei Treiber frei [14]. Um die Treiber freischalten zu können, muss der Enable-Anschluss auf HIGH geschaltet werden. Ist der Enable-Anschluss auf LOW geschaltet, so dreht sich der Motor nicht.

Die jeweils zwei mittleren Anschlüsse auf beiden Seiten sind die Ground-Anschlüsse. Der VCC1-Anschluss ist für die Stromversorgung des Motortreibers da und wird vom Raspberry Pi mit 5V versorgt. Der VCC2-Anschluss bekommt die Stromversorgung von den Batterien und versorgt damit die

Tabelle I
ANSTEUERUNGSVERHALTEN DES MOTORTREIBERS UND DAS VERHALTEN VON MOTOR1

1A	2A	EN1,2	Motorverhalten
X	X	LOW	Motor stoppt
HIGH	LOW	HIGH	Motor dreht sich im Uhrzeigersinn
LOW	HIGH	HIGH	Motor dreht sich gegen den Uhrzeigersinn

Motoren.

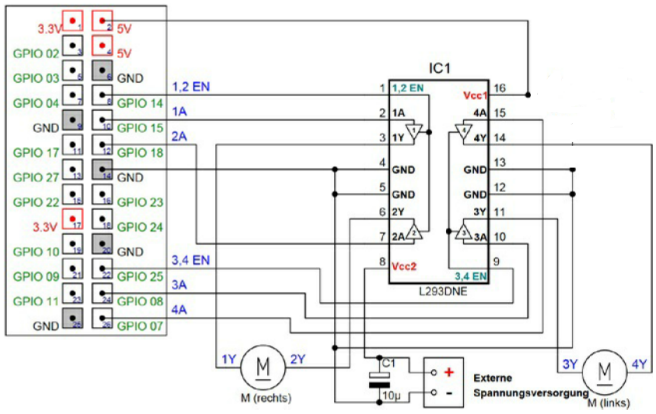


Abbildung 6. Schaltung [11, S.670]

Abbildung 6 zeigt die komplette Schaltung. Dort ist jeder Motor mit zwei Ausgängen der Treiber verbunden. Weiterhin sind alle Ground-Anschlüsse (Raspberry PI, Mototreiber und Batterie) miteinander verbunden. Der 5V-Anschluss vom Raspberry Pi ist dem VCC1 Anschluss des Motortreiber verbunden, wodurch dieser mit Strom versorgt wird. Der Pluspol der Batterie ist mit dem VCC2-Anschluss des Motortreibers verbunden und versorgt damit die Motoren. Um eventuell auftretende Spannungsspitzen oder Spannungsabfällen vorzubeugen, wird ein 10F Elektrolytkondensator parallel zu den Batteriepolen geschaltet. Spannungsspitzen oder Spannungsabfälle treten oft beim An- und Ausschalten bzw. beim Umpolen der Motoren auf. Der Kondensator hält die gewünschte Spannung aufrecht [11, S. 670]. Mit dieser Schaltung kann dann jeder Motor so angesteuert werden, dass dieser sich im Uhrzeigersinn, sich gegen den Uhrzeigersinn oder gar nicht dreht (Tabelle I)

V. KONSTRUKTION DES AUTOMODELLS

Als Grundlage wurde ein Modell verwendet, welches aus einer Plexiglasscheibe besteht, an dem die beiden Motoren montiert wurden. Als Stütze für die Stabilität wurde ein weiteres freibewegliches Rad angebaut. Auf der Plexiglasscheibe wurde ein Breadboard, auf dem sich der Motortreiber befindet, festgeklebt. Ein Fach für die Batterien wurde daneben festgeschraubt. Der Raspberry PI befindet sich über den Batterien und wird mittels einer weiteren Plexiglasscheibe von den Batterien elektrisch isoliert. Über ein MikroUSB-Netzteil wird

Tabelle II
TASTATUREINGABE UND RESULTIERENDES AUTOMODELLVERHALTEN

Tastatureingabe	Automodellverhalten
0	Automodell stoppt
1	Automodell fährt vorwärts
2	Automodell fährt rückwärts
3	Automodell dreht sich links herum
4	Automodell dreht sich recht herum

der Raspberry Pi mit Strom versorgt und ist damit nicht vollständig kabellos. Der Adapter für die Nunchukansteuerung befindet sich ebenfalls auf dem Breadboard. Abbildung 7 zeigt das fertig gebaute Automodell.

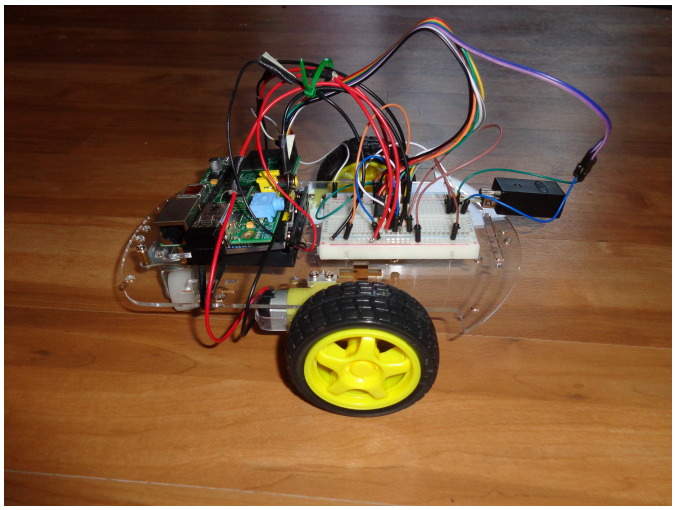


Abbildung 7. Konstruktion des Automodells

VI. STEUERUNG

Für die Steuerung wurde ein Programm geschrieben, welches es ermöglicht, das Automodell anzusteuern. Dabei soll das Automodell in der Lage sein, sowohl sich links und rechts zu drehen, als auch vorwärts und rückwärts zu fahren. Als Programmiersprachen wurde Python verwendet, da der Raspberry Pi bereits einen Python-Interpreter zur Verfügung stellt. Zudem gibt es viele Bibliotheken in Python, die Erweiterungen wie die Verwendung des Wii Nunchuks unterstützen.

A. Tastatur

Die Tastatur ist eine Möglichkeit mit der man die Motoren ansteuern kann. Mit Hilfe der Tastatur sollen Zahlen von 0-4 eingelesen werden. Entsprechend dieser Zahlen soll sich das Automodell dann links drehen, rechts drehen, vorwärts oder rückwärst fahren (Tabelle II). Andere Eingaben sollen ignoriert werden.

1) Programmierung für die Ansteuerung mit Tastatur: Am Anfang des Programmes werden die Bibliotheken "RPi.GPIO", für die Programmierung der GPIO-Schnittstelle, und "sys", für das Einlesen der Tastatureingaben, importiert.

Programmcode Nunchuk: Importierung von Bibliotheken

$import\ RPi.GPIO\ as\ GPIO$
$import\ sys$

Anschließend werden die Anschlüsse, die mit den Eingängen des Motortreiber und den Enable-Anschluss verbunden sind, initialisiert.

Programmcode Tastatur: Initialisierung von Enable-Anschlüssen und Treibereinganänge [11, S.677]

$M1Enable = 14$
$M1\_1a = 15$
$M1\_2a = 18$
$M2Enable = 25$
$M2\_3a = 8$
$M2\_4a = 9$

Als nächstes wird die Anschlussnummerierung der GPIO mit BCM (Broadcom) festgelegt. BCM ist eine Art die GPIO-Anschlüsse zu nummerieren. Dabei werden nur die freiprogrammierbaren Anschlüsse durchnummeriert. Eine andere Nummerierung wäre BOARD, wo alle Anschlüsse durchnummeriert werden.
Danach werden alle Warnmeldungen deaktiviert. Ohne der Deaktivierung der Warnmeldung treten mehrere RuntimeWarnings auf, die störend sind und ignoriert werden können. Im Anschluss werden die zuvor initialisierten Anschlüsse als Ausgänge gesetzt

Programmcode Tastatur: Ausgänge setzen [11, S.677]

$GPIO.setmode(GPIO.BCM)$
$GPIO.setwarnings(False)$
$GPIO.setup(M1Enable, GPIO.OUT)$
$GPIO.setup(M1\_1a, GPIO.OUT)$
$GPIO.setup(M1\_2a, GPIO.OUT)$
$GPIO.setup(M2Enable, GPIO.OUT)$
$GPIO.setup(M2\_3a, GPIO.OUT)$
$GPIO.setup(M2\_4a, GPIO.OUT)$

Das Programm besteht aus drei Funktionen:

- die motor-Funktion
- die move-Funktion
- und der main-Funktion

Programmcode Tastatur: motor()-Funktion [11, S.679]

```
def motor(motor, move):
  if motor == 1:
    if move == 0:
Motor1 stoppt
      GPIO.output(M1_1a, GPIO.LOW)
      GPIO.output(M1_2a, GPIO.LOW)
    if move == 1:
Motor1 dreht sich im Uhrzeigersinn
      GPIO.output(M1_1a, GPIO.HIGH)
      GPIO.output(M1_2a, GPIO.LOW)
    if move == 2:
Motor1 dreht sich gegen den Uhrzeigersinn
      GPIO.output(M1_1a, GPIO.LOW)
      GPIO.output(M1_2a, GPIO.HIGH)
  if motor == 2:
    if move == 0:
Motor2 stoppt
      GPIO.output(M2_3a, GPIO.LOW)
      GPIO.output(M2_4a, GPIO.LOW)
    if move == 1:
Motor2 dreht sich im Uhrzeigersinn
      GPIO.output(M2_3a, GPIO.HIGH)
      GPIO.output(M2_4a, GPIO.LOW)
    if move == 2:
Motor2 dreht sich gegen Uhrzeigersinn
      GPIO.output(M2_3a, GPIO.LOW)
      GPIO.output(M2_4a, GPIO.HIGH)
```

Die motor-Funktion hat zwei Eingabeparameter. Der erste gibt an, um welchen Motor es sich handelt und der zweite gibt an, ob der Motor stoppt, sich im Uhrzeigersinn oder sich gegen den Uhrzeigersinn drehen soll. Entsprechend dieser Parameter werden die Treibereingänge des gewählten Motors so geschaltet, dass sich der Motor in die gewünschte Richtung dreht.

Die zweite Funktion, die move-Funktion, bekommt als Eingabeparameter die Tastatureingabe übermittelt. Abhängig von dieser Eingabe wird die motor-Funktion für jeweils beide Motoren mit dessen gewünschter Drehrichtung aufgerufen. Die gewünschte Drehrichtung wird in Zahlenwerten zwischen 0 und 2 angegeben. Dabei steht 0 für stopp, 1 für im Uhrzeigersinn und 2 für gegen den Uhrzeigersinn.

Programmcode Tastatur: move()-Funktion [11, S.680]

```
def move(eingabe):
  if eingabe == 0:
Das Automodell soll stoppen
    motor(1, 0)
    motor(2, 0)
  if eingabe == 1:
Das Automodell soll vorwärts fahren
    motor(1, 1)
    motor(2, 1)
  if eingabe == 2:
Das Automodell soll rückwärts fahren
    motor(1, 2)
    motor(2, 3)
  if eingabe == 3:
Das Automodell soll sich links drehen
    motor(1, 1)
    motor(2, 2)
  if eingabe == 4:
Das Automodell soll stoppen
    motor(1, 2)
    motor(2, 1)
```

Die letzte Funktion ist die main-Funktion, welche zugleich die Hauptfunktion des Programmes ist. Diese schaltet zunächst die vier Treiber frei. Anschließend wird versucht die Eingabe der Tastatur in einer Endlosschleife zu lesen. Entsprechend der Tastatureingabe wird die move-Funktion mit der entsprechenden Parametereingabe aufgerufen Für den Fall das man das Programm beenden möchte, kann man mit der Tastenkombination CTRL + C einen Keyboard-Interrupt Exception auslösen. Das Automodell wird angehalten und das Programm abgebrochen.

Programmcode Tastatur: main()-Funktion [11, S.680]

```
def main():
GPIO.output(M1Enable, GPIO.HIGH)
GPIO.output(M2Enable, GPIO.HIGH)
  try:
    while True:
      k = sys.stdin.read(1)
      if k =='0': move(0)
      if k =='1': move(1)
      if k =='2': move(2)
      if k =='3': move(3)
      if k =='4': move(4)

  except KeyboardInterrupt:
motor(0)
```

Nach dem Ausführen des Programmes, ließ sich das Automodell wie gewünscht steuern. Lediglich der Bewegungsradius war aufgrund des Netzkabels des Raspberry Pis sehr eingeschränkt.

B. Wii Nunchuck

Der Wii Nunchuk (Abbildung 8) wurde von Nintendo hergestellt und ist eine Ergänzung zum Wii Controller, welcher zur Wii Konsole gehört. Bestandteile des Nunchuks sind ein Joystick, zwei Knöpfen und ein interner Beschleunigungssensor. Aufgrund des niedriegen Preises und der einfachen Anbindung an den Raspberry Pi oder auch an anderen Microcontroller wie Arduino, kommt er immer häufiger, z.B. als Alternative zu einem üblichen Beschleunigungssensor, zum Einsatz. Im Gegensatz zur Tastatur ist der Nunchuk handlicher und entspricht eher der Steuerung eines Ferngesteuerten Autos. Der verwendete Nunchuk ist kein Original von Nintendo, sondern ein von BLAZE hergestellter. Dieser wird mittels einer Bluetooth Schnittstelle ferngesteuert, so dass man sich ein Kabel spart.

Abbildung 8. Wii Nunchuk [19]

1) I^2C: Für die Übertragung der Daten wird das Inter-Integrated Circuit (I^2C) Protokoll verwendet, welches für serielle Übertragungen dient. Es wurde für das Lesen und das Übertragen von Daten zwischen verschiedenen Geräten wie z.B. zwischen Mikrocontrollern und Sensoren, entwickelt. Für die Datenübertagung werden zwei Signalleitungen benötigt. Die eine ist die Serial Clock Leitung (SCL), welche für die Synchronisation der Kommunikation zuständig ist. Die andere Signalleitung ist die Serial Data Leitung (SDA). Diese ist für die eigentliche Datenübertragung zuständig. Für die Übertragung von Daten gibt es immer einen Master und einen oder mehrere Slaves. Der Master koordiniert die Datenübertragung zwischen sich selbst und den Slaves. In diesem Versuch wird der Raspberry Pi als Master agieren und der Nunchuck als Slave. Für die Benutzung des I^2C-Buses gibt es die WiringPi-Python.h Bibliothek für Python, welche mehrere Kommandos für die Initialisierung und Kommunikation zwischen verschiedenen Geräten enthält.

2) Anbindung und Programmierung des Nunchuks: Um den Nunchuk an den Raspberry Pi anzubinden, sind vier Verbindungen notwendig, zwei für die Stromversorgung und zwei für die Datenübertragung (Abbildung 9).

Da es deutlich mehr Aufwand ist, den Nunchuk so zu verbinden, wird ein speziell dafür gebauter Adapter verwendet (Abbildung 10).

Die eine Seite des Adapter schiebt man in die Nunchuckschnittstelle. Auf der anderen Seite des Adapter befinden sich vier Anschlüsse. Zwei Anschlüsse sind für die Stromversorgung zuständig. Der eine wird mit dem 5V-Anschluss des

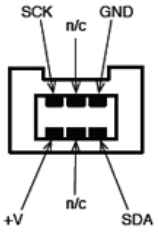


Abbildung 9. Nunchuk Anschlussbelegung [10]

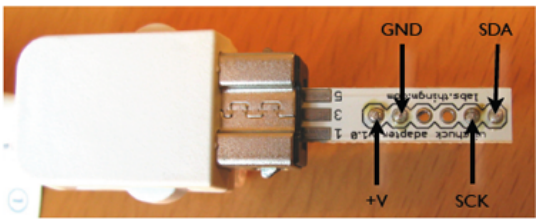


Abbildung 10. Nunchuk-Schnittstelle mit Adapter[10]

Raspberry Pis verbunden und der andere Anschluss wird mit Ground verbunden. Der dritte Anschluss wird mit dem GPIO-Anschluss 02 verbunden, welcher für die SDA-Leitung zur Verfügung steht. Der letzte Anschluss, der für die SCL-Leitung ist, wird mit dem GPIO-Anschluss 03 verbunden.

3) Programmierung für die Ansteuerung mit dem Nunchuck: Das Programm für die Nunchuksteuerung unterscheidet sich etwas von dem mit der Tastatursteuerung. Zunächst müssen zusätzliche Bibliotheken importiert werden. Dazu gehört die time Bibliothek, welche für die sleep-Funktion benötigt wird. Weiterhin wird die wiringpi Bibliothek für die I^2C-Datenübertragung benötigt und eine Nunchuk Bibliothek, welche das Datenauslesen beim Nunchuk unterstützt. Die sys Bibliothek wird dagegen nicht mehr benötigt und weggelassen.

Programmcode Nunchuk: Importierungen [15]

```
import time
import wiringpi
import Nunchuk
```

Wie beim vorherigem Programm werden die Eingänge der Treiber wieder initialisiert und als Ausgänge gesetzt. Auch die motor- und move-Funktionen bleiben unterverändert. Die größten Änderungen gibt es in der main-Funktion. Hier werden erneut die Treiber freigeschaltet. Anschließend wird die Funktion Nunchuk.setup() aufgerufen, welche sich in der Nunchuk-Bibliothek befindet. Die Nunchuk.setup()-Funktion initialisiert und kalibriert den Nunchuk. Nun wird wieder in einer Endlosschleife jede halbe Sekunde versucht, die Daten vom Joystick, des Nunchuks auszulesen. Dabei können die x- und y-Werte einen Wert im Bereich -8 und 8 annehmen. Nachdem die Daten ausgelesen wurden, wird die sleep-

Funktion für 0.5 Sekunden aufgerufen. Anschließend wird der abgelesene Wert ausgewertet. Bewegt man den Joystick nach vorne, so wird ein x-Wert größer 0 ausgelesen und die move-Funktion für vorwärts wird aufgerufen. Bewegt man den Joystick zurück, enthält man einen x-Wert kleiner 0 und das Automodell soll sich rückwärts bewegen. Wird der Joystick seitlich verändert, so ändert sich der y-Wert. Wird der Joystick nach links gedrückt, soll sich das Automodell links herum drehen und falls der Joystick nach rechts bewegt wird, soll sich das Automodell rechts herum drehen. Sind der x-Wert und der y-Wert null, so soll das Automodell stehen. Die Fälle in denen der Joystick schräg nach unten oder oben bewegt wurde und somit der x-Wert und der y-Wert ungleich Null sind, wurden nicht weiter berücksichtigt.

Programmcode Nunchuk: main()-Funktion [15]

```
def main() :
GPIO.output(M1Enable, GPIO.HIGH)
GPIO.output(M2Enable, GPIO.HIGH)
Nunchuk.setup()
  try :
    while True :
        x, y = Nunchuk.read_joy_normalized()
        time.sleep(0.5)
        if x > 0 : move(1)
         if x < 0 : move(2)
         if y > 0 : move(3)
         if y < 0 : move(4)
         else : move(0)
  except KeyboardInterrupt :
    motor(0)
```

Nach dem Ausführen des Programmes, läßt sich das Automodell wie gewünscht steuern. Erneut ist der Bewegungsradius aufgrund des Netzkabels des Raspberry Pis sehr eingeschränkt. Die eingelesenen Werte entsprechen genau dem, was man nach gedrücktem Joystick erwartet. Auch wird die null ausgelesen, wenn man den Joystick nicht bewegt.

VII. Ergebnisse

Das entwickelte Automdell verhält sich so, wie erwartet. Es kann vorwärts und rückwärts fahren und sich drehen. Es lässt sich sowohl mit der Tastatur, als auch mit dem Nunchuk steuern. Ein sehr störender Bestandteil des Automodells war das Netzkabel des Raspberry Pis.
Bei der Entwicklung waren, besonders beim Nunchuck, die Bibliotheken sehr hilfreich und ersparten viel Programmierarbeit. Die kurzen Tests ergaben keine Probleme mit Echtzeitverhalten.

Dennoch traten einige wenige Probleme auf:

- Installation der Bibliotheken: Um die Bibliotheken verwenden zu können, mussten diese vorher heruntergeladen uns installiert werden. Dabei traten häufiger Fehler auf, die darauf basierten, dass andere Programme erstmals geupdated oder installiert werden mussten.
- I^2C aktivieren: Um I^2C verwenden zu können musste I^2C von der raspi-blacklist.conf entfernt werden damit es nicht weiter deaktiviert ist.
- Erno 5 Input/Output Error: Dieser Fehler kam bei der Programmierung mit dem Nunchuk auf und besagt lediglich, dass es sich um einen physikalischer Input/Output Fehler handelt. Das finden dieses Fehlers war sehr aufwendig. Letzten Endes war es eine fehlerhafte Verbindung der SDA- und SCL-Leitungen

Das Automodell ist soweit recht einfach gebaut. Aber es gibt zahlreiche Möglichkeiten das Automodell zu erweitern. Einige Beispiele für Erweiterungen wären:

- Raspberry Pi mit Akku betreiben: Das Netzteil hat einem die Tests mit dem Automodell sehr erschwert. Der Bewegungsradius war sehr eingeschränkt. Es wäre deutlich einfacher das Automodell ohne das Netzteil zu steuern.
- Geschwindigkeitssteuerung: Der Raspberry Pi kann sich nur mit voller oder keiner Geschindigkeit bewegen. Möchte man aber die Geschwindigkeit des Automodell steuern können, was besonders mit dem Nunchuk leicht umsetzbar ist, so kann man dies mittels Pulse-Width-Modulation (PWM) tun.
- Echtzeitverhalten: Wird ein garantiertes Echtzeitverhalten vorausgesetzt, so ist der Raspberry Pi ungeeignet. Besonders bei der Verwendung von PWM kann es schnell zu solchen Problem kommen. Um ein Echtzeitverhalten garantieren zu können, könnte man z.B. einen Arduino mit dem Raspberry Pi verbinden oder ein Real-Time-Clock Modul an den Raspberry Pi anschließen.
- Weitere Erweiterungsmöglichkeiten: Der Raspberry Pi hat viele ungenutzte Ergänzungsmöglichkeiten wie Sound- oder Grafikausgaben. Auch könnte man eine Kamera auf das Automodell integrieren und über das Internet steuern. Mit zusätzlichen Sensoren, wie z.B Infrarotsensor, wäre das Automodell auch in der Lage selbstständig zu fahren.

Der Raspberry Pi hat seine Aufgaben für dieses Projekt erfolgreich bearbeitet. Dennoch sollte man auch Alternativen betrachten, da der Raspberry Pi sehr viele Schnittstellen, für die Verwendungen von z.B Sound, Grafikausgabe oder auch das Abspielen von HD Videos hat, die jedoch nicht für die Anbindung und Steuerung von Motoren notwendig waren. In diesem Fall hätte auch ein preiswerterer Arduino ausgereicht, der zudem leichter und schneller zu programmieren wäre. Möchte man das Automodell erweitern, so wäre der Raspberry Pi sehr gut dafür geeignet, sofern man kein Echtzeitverhalten garantieren muss. Soll das Automodell aber keine weiteren Anforderungen erfüllen, so hätte ein Arduino genügt. Um echtes Echtzeitverhalten und die deutlich besseren Anwendungsmöglichkeiten zu erhalten wäre der Rapsberry Pi in Verbindung mit dem Arduino am Besten geeignet.

Literatur

[1] Die elektronische Welt mit Raspberry Pi entdecken . O'Reilly . Januar 2014 . URL: http://www.oreilly.de/catalog/elekraspberrybasger/top

[2] How To's . Raspberry Pi Guide . Januar 2014 . URL: http://raspberrypiguide.de/HowTo
[3] Definition Motor . Sured . Januar 2014 . URL: http://www.sured.info/defd/motor.html
[4] Gasturbinen . Uni Due . Januar 2014 . URL: https://www.uni-due.de/imperia/md/content/verfahrenstechnik/et42010.pdf
[5] M. Henger, *Zur Betriebsfestigkeit elektrischer Maschinen in Elektro- und Hybridfahrzeugen*, 1. Auflage, Tamm, Deutschland: Springer, 2013.
[6] Verbrennungsmotor . Uni-Protokolle . Januar 2014 . URL: http://www.uni-protokolle.de/Lexikon/Verbrennungsmotor.html
[7] Aliimg . Januar 2014 . URL: http://i01.i.aliimg.com/wsphoto/v0/608170761_1/3PCS-Free-Shipping-High-Quality-6V-Smart-Car-Robot-Plastic-DC-Drive-Gear-Motor-tyre-for.jpg
[8] Elektromotor . GSWN . Januar 2014 . URL : http://www.gswn.de/Projekte/projectpdf/Dokumentationen/Elektromotoren
[9] Insight - How Geared DC Motor works . EngineersGarage . Januar 2014 . URL:http://www.engineersgarage.com/insight/how-geared-dc-motor-works
[10] Physical Computing - Wii Nunchuk . Lizarum . Januar 2014 . URL:http://lizarum.com/assignments/physical_computing/2008/wii_nunchuck.html
[11] E. Bartmann, *Die elektronische Welt mit Rasperry Pi entdecken*, 1. Auflage, Köln, Deutschland: O'Reilly, 2013.
[12] Motoren mit der L293D oder SN754410 H-Brücke ansteuern . Eckard-Gosch.Januar 2014 . URL:http://www.eckhard-gosch.de/de/articles.php?article_id=2
[13] Aktoren . Uni Ulm . Januar 2014 . URL:http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.130/Mitarbeiter/oubbati/RobotikWS1113/Folien/Aktoren.pdf
[14] Texas Instruments, *Datasheet: Quadruple Halfh Drivers L293*, Revision, 2004.
[15] piiwii . Github . Januar 2014 . URL:https://github.com/ChickenProp/piiwii/blob/master/robot.py
[16] Der WissensblogDelfine3d . Januar 2014 . URL:http://www.delfine3d.de/elektromotor-funktionsweise-einfach-erklart/
[17] Elektromotor . Forscherland . Januar 2014 . URL:http://www.forscherland-bw.de/fileadmin/Bilder/Videos_ExperimenteElektromotor_1.pdf
[18] Elektromotor . Wikepedia . Januar 2014 . URL:http://de.wikipedia.org/wiki/Elektromotor
[19] Comunicando com o Nunchuk . Jumpear . Januar 2014 . URL:http://blog.jumpear.com.br/2010/10/29/comunicando-com-o-nunchuk/

Expandable Home Automation with the Raspberry Pi and a Smartphone

Markus Müller
Department of Computer Science
Carl von Ossietzky University Oldenburg
Email: markus.mueller@uni-oldenburg.de

Abstract—The Raspberry Pi has become a popular platform with over two million sold devices[1]. It is today used in many different projects, for example projects that concern with home automation and the internet of things. Therefore several open source frameworks had been published which collects sensor data, actuate on devices, and as well have data processing capabilities. Actual frameworks interconnect the Raspberry Pi with modern smartphones and have their focus on this interface. The subject of this paper is a expandable home automation solution build with Raspberry Pis. Multiple Raspberry Pis build a network that allow interconnection of different devices in different location controlled by a Smartphone. This results in an framework that allows Raspberry Pis to exchange information and send them to a Smartphone.

1 Introduction

The idea behind the Internet of things (IOT) is to connect physical objects with a Network so that they can be controlled and accessed through the Internet. The interest in the field of IOT applications has been increased over the last years, thanks to the recent advances of embedded systems and there falling cost [2].

Because of the latest development and the falling prices of embedded devices like the Raspberry Pi, it is now possible for everyone to build systems that can be controlled over the Internet. These systems can read sensor values, monitor cameras, and manage actuators like a step motor. Since most of these modern embedded platforms can run a modern operating system and have network interfaces a communication over the Internet can be established with standard Internet protocols [3].
The Raspberry Pi is a credit-card sized microcomputer with a lot of interfacing capabilities. Hardware interfaces that are offered by the Raspberry Pi are for example the universal serial bus (USB), a High-Definition Multimedia Interface (HDMI) and General-Purpose inputs/outputs (GPIO). This microcomputer exists in two different models: Model A and Model B, where model B contains twice as much random access memory compared to Model A and has an additional Ethernet network interface. Model B and its interfacing options are shown in the Figure 1.
The Raspberry Pi has a Broadcom BCM2835 System on a Chip (SoC) with a ARM1176JZF-S computation processor unit (CPU) clocked with *700 MHz*, a graphics processing unit (GPU) named Dual Core VideoCore IV®, and *512 MB* shared random access memory (RAM). Since the Raspberry Pi has a fast SoC, it is capable to run different operating systems (OS) like the Linux distribution Raspbian, RISC OS, or FreeBSD 10. [4]

The mentioned operating systems come with a lot of high level programming languages and libraries that enable the users and developers to get quickly started with a project. Because the Raspberry Pi is low priced and offers a lot of functionality with its different hardware interfaces and software frameworks, there is a big community supporting the project.

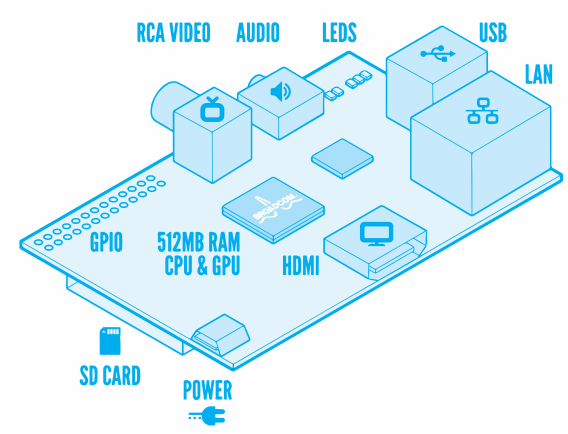


Figure 1. Raspberry Pi Model B [5]

This document gives a short introduction about the Raspberry Pi and it capabilities related to home automation projects. The introduction includes a short description of the hardware, interfaces and the network features which the Raspberry Pi offers. Followed by an overview about Web service standards, protocols and architectures that can often be found in projects with the focus on home automation. The next section will present and discus existing projects that uses a Smartphone to control and

monitor a Raspberry Pi and its connected peripheral. The focus has been kept on projects that use standards and protocols that are related to the Web. The following part is about a concept for an expandable home automation that relies on the Raspberry Pi and a Smartphone.

2 Architecture and Protocols

In home automation and monitoring systems, a simple client- server model is often used as architecture. In this structure, the client can send requests to the server and the server may perform an action and sends a response to the client. One of the most common and widely used client-server protocols is the application layer protocol HyperText Transfer Protocol (HTTP). Other protocols could also be considered as Client-Server protocols. One benefit of HTTP in case of a communication between a Smartphone and a Raspberry Pi is that every modern Smartphone already has an HTTP client built in; the browser. Furthermore most modern Smartphone vendors provide a software development kit (SDK) that includes an application programming interfaces (API) to allow sending and receiving of HTTP messages between a smartphone client and a Web server.

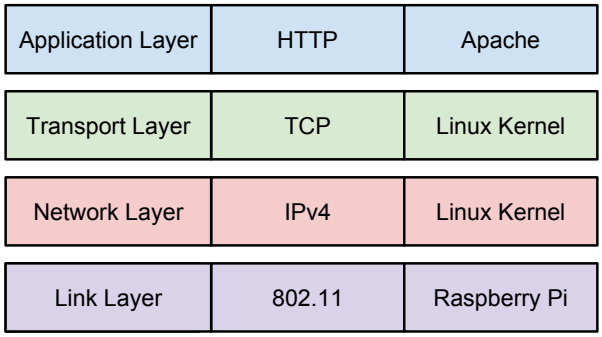


Figure 2. Simplified HTTP Stack on the Raspberry Pi

The Raspberry Pi must run an HTTP server that handles the HTTP messages. Because the Raspberry Pi can run the Linux operating system, it can use different HTTP Web server implementation for this purpose. The two most popular open source implementations are the Apache and the nginx web servers, and both can be used on the raspberry pi. [6] Figure 2 shows how a the HTTP stack is constructed and which communication layer is handled by which part.

2.1 HTTP

The application layer protocol HTTP is used in all Web browsers and Web servers and in an increasing number of applications running on a Smartphone in order to communicate with a backend server. As already mentioned, the HTTP protocol is a classic client-server protocol.

The server hosts resources which can be requested by the client. These resources can be plain text, xml files, jpeg images, or any other file format that exist. Each file has its own uniform resource identifier (URI) which allows the client to address the resource uniquely. A request is formulated with one of the HTTP methods that initiate a transaction. [7] HTTP supports different methods, where some are shown in the Table 1.

Table 1
Most Common HTTP Methods [8]

HTTP method	description
GET	the client requests a resource with an URI
PUT	stores or change content on the server by a given URI
DELETE	request message to delete a specific resource
POST	stores content onto the server into a resource

Figure 3 shows a typical GET request message sentw by a client. In this case the server sends the requested file in its response message body to the client. This a typical client-server dialog.

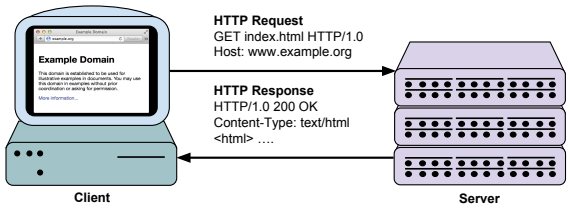


Figure 3. HTTP Client-Server Example Message Exchange

2.2 Data Structures

HTTP is used to send messages between the client and the server, where these messages contain data. Data formats that are transmitted and widely used are the Extensible Markup Language (XML) and JavaScript Object Notation (JSON). Both data structure formats are designed to be readable by humans and machines. Data representation is more flexible in the XML format compared to JSON. However JSON is easier to parse because of the limited support for data types and the lack of attributes.

JSON is suitable for simple data models that are based on a key and value structure. JSON supports basic data types which are arrays, strings, numbers, boolean and objects. Objects can be one or more key-value pairs,

```
[
    {
        "city": "Berlin",
        "Zip":      "13125",
        "Country":  "DE"
    },
    {
        "city": "Oldenburg",
        "Zip":      "26123",
        "Country":  "DE"
    },
]
```

Figure 4. JSON Data Structure Example

where the key is a unique string within the object and the value consist of one of the basic data types [9].

Figure 4 shows an JSON example that consists of an Array containing two objects. Each object holds information that are Strings and Integers.

2.3 Representational State Transfer

The Representational State Transfer (REST) describes a set of rules for how to design a Web service. These rules apply to Web services that has its focus on system resources and their states. The REST concepts and principles do not only refer to HTTP, they can applied to other protocols. The REST concept describes how the state and the resources are processed. The sate is held by the client and the resources are managed by the server. This leads to the requirement that every client request contains all necessary information about the actual state of the resource that are affected by the request [10].

Between the components, there exist a uniform interface that is represented by an uniform resource identifier (URI). A URI points to a resource on the server and should be self descriptive, which means that the URI contains a path which explains what kind of resource is manipulated and how. In Table 2, the REST data elements are listed.
A URI consist of multiple parts. The first part is the *scheme* that indicates the used protocol. In the following URI, the brown part points to the HTTP protocol.

`http://example.org/slave1/LED1/state=?on`

The scheme is followed by the *authority* which is a destination server, on which the Web service is installed and is running. The red highlighted part in the URI is the *path* that points to a resource followed by a query that is optional.

Table 2
REST Data Elements Example

Data Element	Raspberry Pi Server Example
resource	GPIO register that controls a pin
resource identifier	an URI like http://raspberry.pi/GPIO/23/value/
representation	a JSON file
representation metadata	value and direction of the GPIO

3 Popular Existing Projects

There are already existing projects that are connecting the Rapsberry Pi with mobile devices like Smartphones. They all vary in how they work and what they are able to accomplish. But all have in common that they rely on the client server model. Also most projects use a similar HTTP stack that has been shown, in order to offer their service to a Smartphone.

3.1 NetIO

NetIO is a closed source application that can be installed on the Raspberry Pi. The NetIO infrastructure consist of three parts. An external web service, that is called the NetIO UI-Designer that allows the user to design interfaces. An example interface generated with the NetIO UI-Designer is shown in Figure 5. The created interface can be installed on to a any Smartphone that can run the NetIO application. The application is available for the Android and the iOS platform. The Smartphone can establish a connection to the Raspberry Pi after a server application is installed and runs on the Raspberry Pi. All components of the Software are closed source and therefor
.

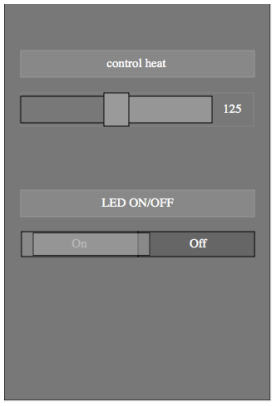


Figure 5. NetIO UI-Designer Generated User Interface

The server application does not support user authentication or any kind of encrypted communication between the

iPhone, the UI-Designer, and the server running on the raspberry pi. The NetIO Application allows to create remote controllers for projects without the need of knowing any programming language[11].

3.2 PiUi

PiUI is an open source project that can be installed on a Raspberry Pi. The PiUI program starts a web server that can turn the Raspberry Pi into a wireless access point. That enables a Smartphone to connect with the Raspberry Pi and the browser can be used to open the interface. The User interface is built with HTML, CSS, and JavaScript, and the website is then rendered by the Smartphone browser.
On the server side, the user interface can be configured and the actions take place when the user pushes a button. This lets the Smartphone control the GPIO of the Raspberry Pi for example. The functionality is served by a REST API that exchange JSON files. That makes it possible to write an application that runs not only in the browser of the Smartphone, instead could realised as dedicated application that runs native source code[12].

3.3 WebIOPi

Another open source project is called WebIOPi. Similar to the PiUI project, WebIOPi runs as a server on the Rasperry Pi and offers a user interface that can be accessed with a browser. In addition, this web server also offers a REST API over HTTP. The user interface can be configured on the server to offer abstraction for different low level protocols like the Serial Peripheral Interface (SPI) or Inter-Integrated Circuit (I2C).

```
request 1 URI: /GPIO/23/function/out
request 2 URI: /GPIO/23/value/1
```

Figure 6. HTTP Request for WebIOPi Web Server

Custom user interface elements can be created and edited with HTML, CSS and Javascript. Also client applications in other programming languages are possible, since the WebIOPi server offers a REST API. An example HTTP POST request in two steps is shown in Figure 6. In the first step, this request sets the direction of the GPIO pin 23 to out and in the next step, it will set the value to a logical 1.
The project comes with a web interface that consists out of HTML, CSS, and Javascript, and can set the direction of a GPIO and the value. The REST API over HTTP enables to build native applications that runs on the Smartphone of the user. The browser and the native application use the REST API that is provided by the Raspberry Pi by running the WebIOPi application. The concept of the infrastructure is illustrated in Figure 7 [13].

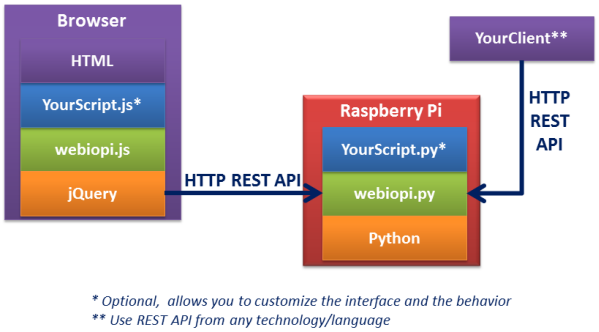


Figure 7. Structure of the WebIOPi Service [13]

In all the mentioned projects, a similar HTTP stack is used to serve and abstract the functionality of the Raspberry Pi. Other projects exist, but they would need additional hardware or external services in order to operate.
One approach would be to extend the Raspberry Pi with additional communication hardware, for example, Near Field Communication (NFC) or Bluetooth. But these need extra specialized hardware or additional services that would need extra attention.

4 Expandable Home Automation

In this section the integration of Raspberry Pi into a home automation processes is presented. As menthioned before, there are many existing frameworks and solutions that bridge the gab between a Smartphone and a Raspberry Pi, where the Raspberry Pi allows to control or monitor elements in a house with a Smartphone.

The expandable home automation project is called Raspberry Pi Home Integration (Raspberry PHI) and provides a Web service that can be installed on the Raspberry Pi in order to allow a Smartphone to control parts in a house. The main advantage of this project is expandability, by control multiple devices in different place.

4.1 Concept

The main idea is to extend the functionality of the Raspberry PHI independent of the location in the house through a self hosted network. In order realize that the Raspberry PHI will operate in an additional wireless network which is exclusive for inter Raspberry PHI device communication. One device is the PHI Master, this device is also an access point. Over this provided network, the master will delegate all requests that are sent from the Smartphone to connected devices. The resulting topology is illustrated in Figure 8.
A benefit of the coverage expansion is that, for example, the garage door opener could be integrated in the home automation system without the need of extra network wires or network configuration. The devices can find each other automatically and exchange their functionality so that the PHI master can propagate that to the Smartphone client.

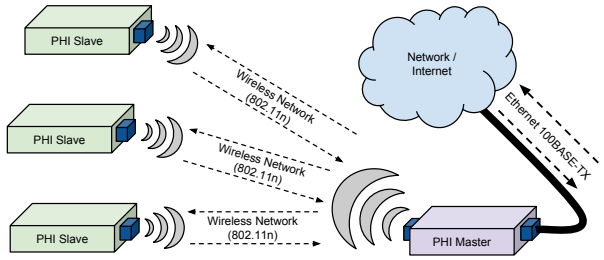


Figure 8. The Physical Communication Layer.

4.2 Architecture

The PHI Master device is connected over an Ethernet connection with the local area network, so it can reach the Internet. The Master is, in addition, a Wireless Access Point (WAP) for the other PHI devices, so the Slave PHI can communicate over a secure encrypted Wi-Fi Protected Access (WPA) connection. The WAP has a one-to-many structure that allows multiple slaves to be connected to the master.

After a slave powers up, it connects with the wireless network spanned by the master and, sends its configured functionality to the master. During the power up phase, the slave acts as a client and the master represents a server, answering the request of the slave.

The PHI slave sends its configured functionality to the PHI master in a JSON formated file. The master then stores the configuration and announces the received functionality of the slave to the Smartphone client.
After the configuration process is completed, the slave migrates into a server mode. This allows the PHI master to send requests to the PHI slave to perform a change in its resource. The request, the master send to the slave is a forwarded request coming from the Smartphone client. That means the PHI master is acting as a server for the Smartphone client, while it is in a client role when it communicate with the PHI slave.

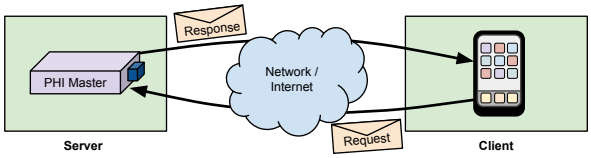


Figure 9. A Smartphone to a PHI Master Communication.

The Smartphone application is a normal client that sends requests to the PHI master which acts as a server, as shown in Figure 9. The structre can be built with an HTTP REST API.

4.2.1 PHI Slave to PHI Master Communication

The wireless network access point, that the master offers, uses the IEEE 802.11 protocol. In order to offer a communication using the IEEE 802.11 standard, the Raspberry Pi needs an additional wireless network connector that can be connected over one of the USB ports.

Since parts of the slave and master software are designed with the programming language Erlang, the connection between the slave and the master is established with some features of the Erlang virtual machine. Figure 10 shows how two Erlang virtual machines can be connected to each other. The connection between the nodes allows the master to directly call function on the slave that will be executed on the slave.

```
(pi@master)>net_kernel:connect(pi@slave)
```

Figure 10. Interconnect two Erlang Nodes

The way the Erlang virtual machine handles the connected nodes makes it easy to develop a distributed system. Also the supervisor principle that is designed into the language framework ensures that, if a slave process dies, it can be restarted by the master automatically [14].

The Slave can be configured to report if a threshold of a connected unit is reached. If, for example, the slave is configured to read out values of a temperature sensor, it notifies the master if a minimum or a maximum value is reached. The master then sends an information to the client.

An additional library that provides high level abstraction for Erlang programs is used for the slaves. This library supports abstraction for protocols like I2C, SPI and pulse wide modulation (PWM) as well as simple GPIO support. The library is offered by the Erlang Embedded Group. [15]

4.2.2 PHI Master

The PHI master, on the one side, offers a web service for the client, and on the other side it establishes a connection to the slaves. Since the connection to the slaves is established using features of the Erlang programming language, the same application can offer the Web service. In order to offer a Web service the master integrates an already developed Web server called Elli. Elli is implemented in Erlang and published under an open source license.

The Elli server allows to implement callbacks for the different HTTP methods that can be called by the client. An example callback function is shown in Figure 11, that consumes the GET request of the client, perform an action and will response to the client with a string. In this example, the URI /GPIO/23/value/1 is called by the

```
handle('GET',[<<"GPIO">>, <<"23">>,
        <<"value">>, <<"1">>], _Req) ->
  %perform action
  io:format("set GPIO pin 23 to 1"),
  %return
  {ok, [], <<"GPIO 23 ON">>};
```

Figure 11. GET Web Server Callback Function

client application.

The Elli Web server supports HTTP Secure (HTTPS), so an encrypted communication is used based on the Secure Socket Layer (SSL) that is based on the public key certificate principle.

4.3 Smartphone Clients

Android and iOS are widely used operating systems, and both offer an SDK for developers. On the Android platform, there is a class, called AndroidHttpClient, that can handle HTTP sessions. On the iOS platform exist an equivalent called NSURLConnection. Both classes do basically the same; handling HTTPS connections.

After the client is started on the Smartphone, it requests the actual state of the PHI devices and their peripherals that are connected. It also locks the system, so it cannot be changed during the interaction. This prevents conflicts in case more then one client tries to change the same resource at the same time. Figure 12 illustrates an example of a sequence of a request and response message exchange between the Smartphone client and the PHI master.

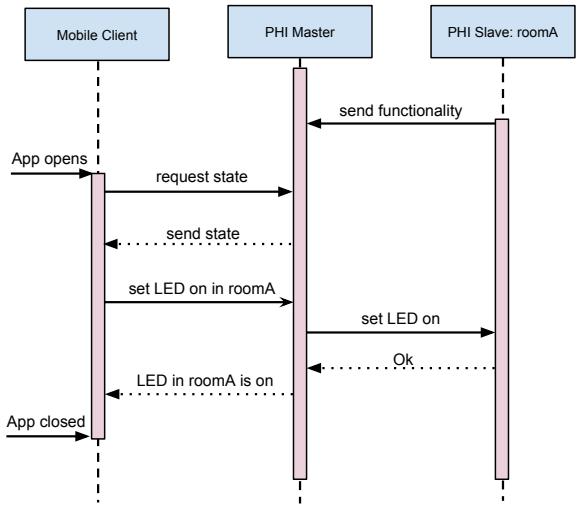


Figure 12. an example communication sequence

First, the client requests the state of the PHI devices. The sate is sent in a JSON style formatted message back to the client. The message is included in an HTML response body. The request and the response is handled by the master that runs the Elli Webserver. If the client request a change on a resource, it is done by the HTTP REST API request, so it is not needed to exchange the complete state again.

The received information are displayed by the Smartphone client with generic interface elements such as labels, switches, and slider. A prototype of a user interface is shown in Figure 13, where the applications interface allows the user to interact with the PHI devices remotely.

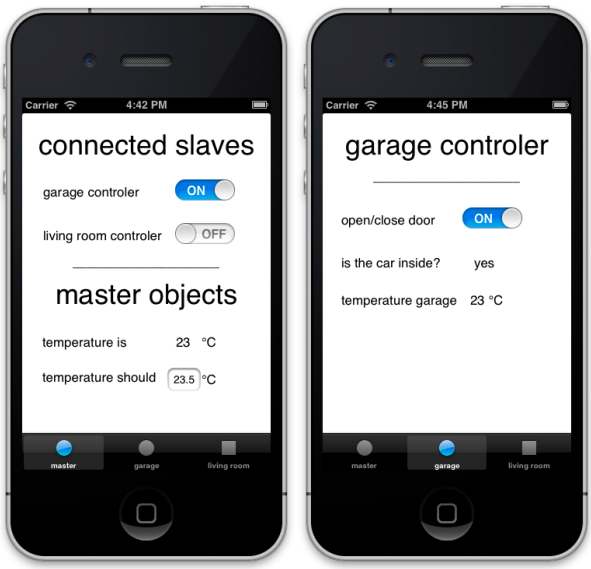


Figure 13. Prototype of the iPhone PHI Client Software

The figure shows on the left the data that represent the state of the master. The state includes for example if a slave is connected. The functionality and the state of the PHI slaves is presented on an extra page, where an example is illustrated on the right of the figure.

5 Conclusion

The Raspberry Pi is well suited for small and lager projects that target to interconnect an environment and to make it accessible over the internet and even controlled by modern Smartphone. Because the Raspberry Pi is an affordable device and can run an operating system, it has become very attractive device for prototyping. A lot of projects has been evolved around the Raspberry Pi. With the Raspberry Pi it is possible to access and control the GPIO pins with a high level language on an embedded device.

Because the Raspberry Pi is a low cost device, it is possible to set up a distributed system without expensive Personal Computers, as the PHI project demonstrates. In order to keep the price low, the PHI slaves can be a Raspberry Model A, which has no Ethernet connector

because the PHI network only make use of the wireless adapter.
The PHI structure is in its actual state limited to a simple star network topology, where the master is in the middle and can reach every slave directly. A conceptual enhancement for future improvement is to expand the functionality of the PHI slaves. In addition to the slaves existing capabilities it can act as a bridge between a PHI master and another slave. This would allow to build even larger networks and would make it possible to reach a wider area. To achieve this functionality the software must be able to route requests and reponses dynamically around the network.

References

[1] Raspberry Pi Foundation. (2013) TWO MILLION! Last visit: 23.1.2013. [Online]. Available: http://www.raspberrypi.org/archives/5265

[2] D. Uckelmann, M. Harrison, and F. Michahelles, *Architecting the Internet of Things*. Springer, 2011.

[3] C. Pfister, *Getting Started with the Internet of Things: Connecting Sensors and Microcontrollers to the Cloud*, ser. Make Series. O'Reilly Media, Incorporated, 2011.

[4] B. Horan, *Practical Raspberry Pi*, ser. Technology in action series. Apress, 2013.

[5] The Raspberry Pi Foundation. (2013) Frequently asked questions, Raspberry Pi Model B. Last visit: 30.12.2013. [Online]. Available: http://www.raspberrypi.org/faqs

[6] NetCraft Ltd. (2013) Web server servey 2013. Last visit: 30.12.2013. [Online]. Available: http://news.netcraft.com/archives/2013/06/06/june-2013-web-server-survey-3.html

[7] R. Fielding. (1999) Hypertext Transfer Protocol – HTTP/1.1, IETF RFC 2616. Last visit: 30.12.2013. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[8] D. Gourley, B. Totty, M. Sayer, A. Aggarwal, and S. Reddy, *HTTP: The Definitive Guide*. O'Reilly Media, 2002.

[9] D. Crockford. (2006) Media Type for JavaScript Object Notation (JSON). Last visit: 30.12.2013. [Online]. Available: http://tools.ietf.org/html/rfc4627

[10] R. T. Fielding, "Architectural styles and the design of network-based software architectures," PhD thesis, University of California, 2000. [Online]. Available: http://www.ics.uci.edu/fielding/pubs/dissertation/top.htm

[11] David Eickhoff. (2012) NetIO Controller Application. Last visit: 3.1.2013. [Online]. Available: http://netio.davideickhoff.de

[12] David Singleton. (2012) Add a Simple Mobile Phone UI to Your RaspberryPi Project. Last visit: 3.1.2013. [Online]. Available: http://blog.davidsingleton.org/introducing-piui/

[13] Eric Ptak. (2012) Internet of Things framework for the Raspberry Pi. Last visit: 30.12.2013. [Online]. Available: https://code.google.com/p/webiopi/

[14] F. Hébert, *Learn You Some Erlang for Great Good! - A Beginner's Guide*, 1st ed. München: No Starch Press, 2013.

[15] Erlang Solutions Ltd. (2013) Erlang Embedded. Last visit: 30.12.2013. [Online]. Available: http://www.erlang-embedded.com/

Raspberry Pi als ISDN–Telefonanlage

Valentin Spreckels
Department für Informatik
Carl von Ossietzky Universität Oldenburg
26111 Oldenburg, Deutschland
Valentin.Spreckels@Informatik.Uni-Oldenburg.DE

***Zusammenfassung*—Der Raspberry Pi ist eine Plattform mit vielen Verwendungsmöglichkeiten. Diese Ausarbeitung schafft die Grundlagen für eine weitere: Die Verwendung als kleine ISDN–Telefonanlage für Heimanwender. Hierfür beschreibt sie, wie der Raspberry Pi mit entsprechender Hardware um zwei ISDN–S_0–Anschlüsse erweitert wird.**

I. Einführung

Telefonie hat eine lange Geschichte: Die ersten Telefongespräche wurden von Hand vermittelt, später wurde das Impulswahl- und schließlich das Mehrfrequenzwahlverfahren eingeführt. Bei diesen Verfahren erfolgte die Übertragung der Sprache analog. Mit der Entwicklung der Computertechnik folgte auch eine Digitalisierung der Telefonie. Zunächst wurde ab den 70er Jahren die Vermittlungstechnik digitalisiert und in den 80er Jahren mit ISDN — Integrated Services Digital Network — auch ein Digitales Verfahren für den Anschluss von Fernsprechteilnehmern entwickelt. In den letzten 15 Jahren erfolgte die Entwicklung der Telefonie über das Internet.

Diese Entwicklung führte dazu, dass in vielen Privathaushalten verschiedene Techniken zu finden sind: Der Router, der den Internetzugang herstellt, hat oft auch einen oder mehrere analoge Telefonanschlüsse und ggf. einen ISDN–Anschluss, sodass er zwischen der Telefonie über das Internet und den vorherigen Telefonietechniken umwandeln kann. Diese integrierten Telefonfunktionen sind in ihrem Funktionsumfang auf Endanwender ohne besondere Anforderungen ausgelegt und bieten daher nur nur wenige Möglichkeiten, die Funktionen individuell zu konfigurieren. Für Firmen gibt es große Telefonanlagen, die ebenfalls zwischen den verschiedenen Techniken vermitteln können; diese Telefonanlagen stellen natürlich weitergehende Funktionen, wie zum Beispiel beliebig konfigurierbare Anrufbeantworter oder Telefonkonferenzen bereit. Zwischen diesen Extremen gibt es Ansätze selbst eine Teleofnanlage zu schaffen, indem man einen Computer mit entsprechender ISDN–Hardware und Software nutzt. Da in der heutigen Zeit Comuter viel Leistungsfähiger als notwendig sind und Energieffizienz immer wichtiger wird, bietet es sich an, eine Raspberry Pi statt eines üblichen Computers zu verwenden. Diese Ausarbeitung untersucht daher, wie dies mit einem Raspberry Pi möglich ist.

Zunächst ist zu betrachten, wie ISDN aufgebaut ist, um zu verstehen, welche Komponente welche Funktionen realisieren kann.

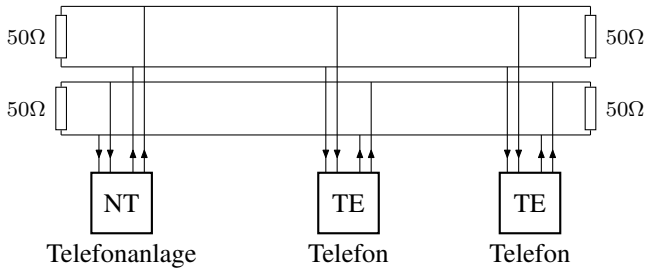


Abbildung 1: Aufbau eines S_0–Busses

II. Grundlagen von ISDN

ISDN ist in mehreren Schichten aufgebaut: Die Schicht 1 ist die Übermittlungsschicht gemäß ISO/OSI–Referenzmodell; Schicht 2 die Sicherungsschicht und Schicht 3 die Vermittlungsschicht. Im folgendem wird ein kurzer Überblick über diese Schichten gegeben, [1] beschreibt die Schichten genauer.

A. Schicht 1: Die Übermittlungsschicht

Die Schicht 1 übernimmt den Transport der Daten auf den Leitungen. Hierfür erfolgt die Verkabelung mit zwei Adernpaaren in Form eines Bus-Systems. Die Funktionen des Adernpaare unterscheiden sich, wie in Abbildung 1 dargestellt, in der Übertragungsrichtung: auf dem einen sendet die Telefonanlage (im ISDN–Referenzmodell nach [2] als NT — Network Termination —bezeichnet), auf dem anderen senden die Telefone (im Referenzmodell TE — Terminal Equipment — genannt). Darauf werden Bits mit einem modifiziertem AMI–Code übertragen[1].

Damit der Empfänger die übertragenen Bits auswerten kann, werden diese in Rahmen verpackt, die in der Richtung von den Telefonen zur Telefonanlage zwei Bits des D- und jeweils zwei Oktetts — also je 16 Bits — der beiden B-Kanäle, sowie Bits zur Signalisierung des Rahmenbeginns (eine bewusste Verletzung der Kodierungsregeln) und zum Wiederherstellen der Gleichspannungsfreiheit. In der anderen Richtung sendet die Telefonanlage zusätzlich einen E-Kanal, in den sie die Daten des von ihr empfangenen D-Kanals spiegelt, sodass Telefone Kollisionen erkennen können, wenn sie selbst ein 0-Bit senden, der E–Kanal aber ein 1–Bit enthält.

[1]Dieser Code übermittelt 1–Bits durch einen Spannungspegel von 0 Volt und 0–Bits alternierend durch $\pm 0,75$ Volt.

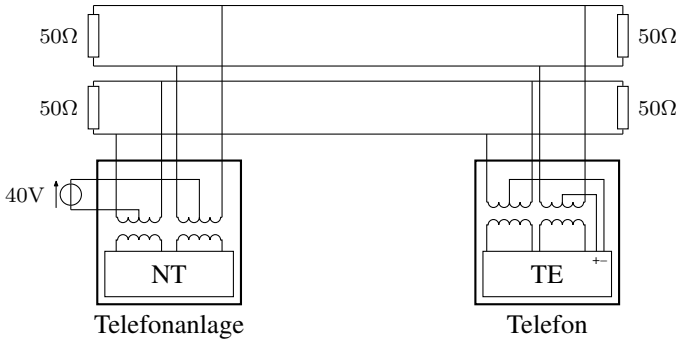


Abbildung 2: Stromversorgung über den S_0–Bus

B. Schicht 2: Die Sicherungsschicht

In der Schicht 2 wird im D–Kanal das Protokoll LAPD — Link Access Procedure on the D-channel — gemäß [3] und [4] benutzt. Dieses benutzt HDLC — High-Level Data Link Control — um auf Basis der Schicht 1, die im D–Kanal einfach nur einzelne Bits überträgt, Pakete zu übertragen. Zusätzlich ermöglicht diese Schicht auch die individuelle Adressierung aller Endgeräte und bietet Mechanismen zur Vergabe und Verwaltung der hierfür benötigten TEI — Terminal Endpoint Identifier.

Im B–Kanal werden in der Schicht 2 abhängig von den zu übertragenden Daten verschiedene Protokolle benutzt. Bei der Sprachübertragung reicht die Schicht 2 unverändert weiter, bei verschiedenen Varianten der Datenübertragung kommen hingegen verschiedene HDLC–Varianten zur Anwendung.

C. Schicht 3: Die Vermittlungsschicht

In der Schicht 3 erfolgt im D–Kanal auf Basis von Paketen der Schicht 2 die Signalisierung von Anrufen und der Auf- und Abbau von Gesprächen, sowie die Bereitstellung weiterer Leistungsmerkmale. Hierfür wurde zunächst ein von der Telekom entworfenes Protokoll benutzt, für das nach dem zugrundeliegendem Standard der Name 1 TR 6 verwendet wird. Dieses wurde nur in Deutschland verwendet und mittlerweile durch das sogenannte Euro–ISDN mit DSS–1 — Digital subscriber Signalling System No. 1 — als Signalisierungsprotokoll ersetzt, das in [5] und [6] beschrieben wird.

Bei den B–Kanälen hängt auch die Ausprägung der Schicht 3 davon ab, welche Art von Inhalten übertragen wird.

D. Stromversorgung

Zusätzlich zur Signalübertragung kann auch eine Stromversorgung der Telefone versorgen. Diese erfolgt mittels Phantomspeisung. Hierfür werden die Adernpaare nicht direkt an die Sende–/Empfangsschaltungen angeschlossen, sondern durch Überträger entkoppelt. Diese stören Wechselspannungssignale nicht, blocken jedoch Gleichspannungen ab. Daher wird am Referenzpunkt NT, also der Telefonanlage bzw. dem NTBA, wie in Abbildung 2 dargestellt, zwischen den Adernpaaren eine Spannung von ca. 40 Volt angelegt, mit der maximal zwei Telefone betrieben werden können.

Das nächste Abschnitt betrachtet Hardware, die es den Raspberry Pi um diese ISDN–Funktionen erweitern kann.

III. ISDN–Hardware für den Raspberry PI

Verschiedene Hersteller stellen oder stellten ISDN–Hardware her. In normalen Computern verwendet man oft PCI–Karten auf Basis von Integrierten Schaltungen der Firma Cologne Chip, da da diese im Gegensatz zu anderen preiswerten ISDN–PCI–Karten nicht nur den Betrieb im TE– sondern auch im NT–Modus beherrschen. Da der Raspberry Pi keine PCI–Schnittstelle hat, kann man keine PCI–Karten verwenden. Dennoch bietet es sich an Integrierte Schaltungen von Cologne Chip zu verwenden. Da diese Firma auf ISDN spezialisiert ist, wohingegen andere Firmen, wie zum Beispiel Siemens keine Produkte für ISDN mehr entwickeln und auch die Vermarktung ihrer Produkte einstellen. Ich habe daher bei Cologne Chip angefragt und habe ein Evaluationsboard mit einer Integrierten Schaltung des Typs XHFC–2SU zur Verfügung gestellt bekommen. Der Aufbau der integrierten Schaltung und die Möglichkeiten ihn an den Raspberry PI anzuschließen werden im folgendem beschrieben.

A. Interner Aufbau des XHFC–2SU

Der XHFC–2SU besteht aus mehreren Komponenten, deren Zusammenwirken in Abbildung 3 dargestellt ist. Die Ein– und Ausgabe von Daten findet über drei verschiedene Arten von Schnittstellen statt: Eine Prozessor–Schnittstelle (unten links in der Abbildung), über die der Anschluss an den Computer erfolgt; eine PCM–Schnittstelle (rechts unten), über die Querverbindungen zu weiterer Telekommunikationshardware hergestellt werden können und den S_0–Schnittstellen, (rechts oben) über die ISDN–Telefone angeschlossen werden können. Die Daten, die über die PCM– bzw. S_0–Schnittstellen, ankommen sind synchron zu einem gemeinsamem Takt[2]. Wann Daten über die Prozessor–Schnittstelle gelesen oder geschrieben werden, ist hingegen vom Programm des Computers, der den XHFC–2SU steuert. Daher werden die in Abbildung 3 als FIFO bezeichneten Ringbuffer benötigt, um die Daten zwischenzuspeichern.

Auf dem Datenpfad zwischen Ringbuffern und PCM– bzw. S_0–Schnittstellen kann eine HDLC–Kodierung bzw. –Dekodierung der Schicht–2–Pakete stattfinden, sowie zwei weitere Arbeitsschritte, um mit Datenkanälen umzugehen, die keine vollständigen Oktetts verwenden, wie zum Beispiel dem D–Kanal[3]. Im Anschluss daran werden die Daten je nach Konfiguration zur PCM–Schnittstelle oder einer der S_0–Schnittstellen geleitet.

Mit der S_0–Schnittstelle werden Funktionen der Schicht 1 bereitgestellt. Mit der HDLC–Kodierung und –Dekodierug wird aus Geschwindigkeitsgründen[4] auch eine kleiner Teil der Schicht 2 in Hardware ausgeführt.

[2]Geringfügige Takt–Abweichungen können ausgeglichen werden.

[3]Ein weiteres Beispiel sind B–Kanäle in den USA, die dort meist nur mit 56 kbit/s verwenden werden und daher nur sieben statt acht Bit verwenden.

[4]HDLC erfordert das Einfügen oder Entfernen von Bits, was auf Prozessoren, die mit Bytes arbeiten, schwierig durchzuführen ist.

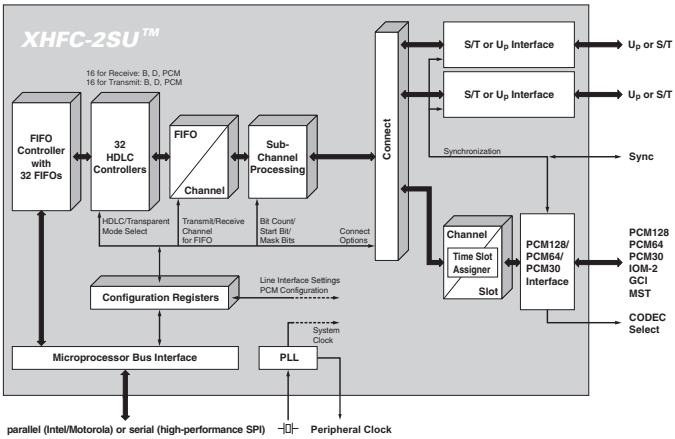


Abbildung 3: Blockdiagramm des XHFC-2SU Chip
Quelle: [7]

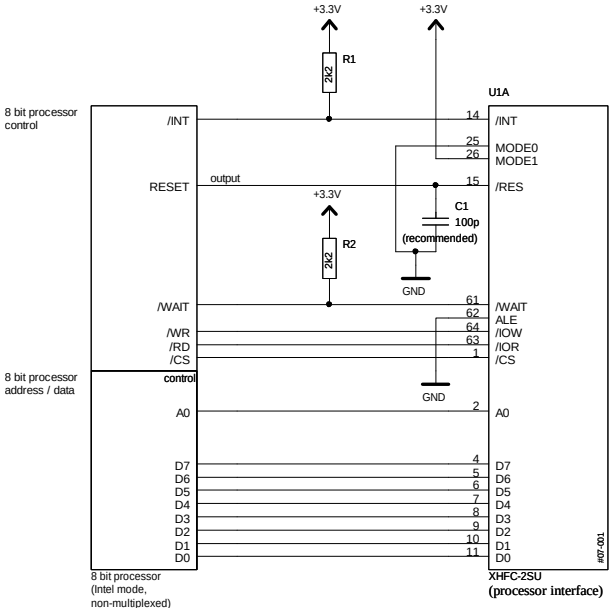


Abbildung 4: Eine der vier Anbindungsvarianten des XHFC-2SU mit einem parallelem Datenbus
Quelle: [7]

Diese Funktionsblöcke müssen für den jeweiligen Anwendungsfall konfiguriert werden. Hierfür haben sie Konfigurationsregister. Diese Register sind teilweise als sogenannte, Array– und/oder Multi–Register ausgelegt. Array–Register werden bei der Konfiguration mehrerer gleichartiger Komponenten verwendet, wie zum Beispiel den beiden S_0–Schnittstellen. Diese werden über die gleiche Registeradresse konfiguriert; für welche der beiden Schnittstelle die Einstellungen, die an diese Adresse geschrieben werden, gelten, ergibt sich aus dem Inhalt eines anderen Registers. Multi-Registern werden verwendet, wenn für verschiedene Betriebsmodi einer Schnittstelle unterschiedliche Einstellungen notwendig sind. Auch hierfür sind die S_0–Schnittstellen ein Beispiel: Sie können auch als U_{PN}–Schnittstellen[5] betrieben werden. Die Betriebsart wird über ein Bit in einem ihrer Konfigurationsregister gesetzt, die Bedeutung der restlichen Bits in diesem Register ist in den beiden Betriebsarten unterschiedlich.

Auch der Zugriff auf Informationen der Ringbuffer erfolgt über eine Registeradressen, hinter denen sich Array–Register verbergen. Dafür muss zuerst in das `R_FIFO`–Register geschrieben werden, auf welchen Ringbuffer zugegriffen werden soll. Anschließend kann dann über die Register `A_Z1` und `A_Z2` bzw. `A_F1` und `A_F2` ermittelt werden, wie viele Oktetts bzw. HDLC–Pakete im Ringbuffer sind, und über das Register `A_FIFO_DATA` auf den Inhalt des Ringbuffers zugegriffen werden.

B. Möglichkeiten der Anbindung

Der XHFC-2SU kann auf verschiedene Arten an den Raspberry Pi angebunden werden. Alle haben gemeinsam, dass eine Leitung zur Signalisierung von Unterbrechungsanforderungen und eine Leitung zum Zurücksetzen des XHFC–2SU benötigt werden.

Bei der Ansteuerung über einen parallelen Bus gibt es vier geringfügig unterschiedliche Varianten. Für alle werden acht Datenleitungen benötigt. Außerdem müssen auch acht Adressbits übertragen werden, um eine der 256 Registeradressen des Chips auszuwählen. Diese Adressbits werden durch Multiplexen mit über die Datenleitungen übertragen. Hierfür wird eine Steuerleitung benötigt. Weitere drei Steuerleitungen werden benötigt, um den Chip auszuwählen, die Art des Zugriffes (Lesen, Schreiben von Daten oder Adressen) festzulegen und um zu signalisieren, ob der Chip die Daten empfangen hat. Die genauen Bedeutungen der Steuerleitungen hängt von der Ansteuerungsvariante ab, eine davon ist in Abbildung 4 dargestellt. Es ist ersichtlich, dass 15 Leitungen zwischen Raspberry Pi und XHFC–2SU benötigt werden würden.

Alternativ ist auch die Anbindung über SPI möglich. Hierfür werden eine Taktleitung, zwei Datenleitungen und eine Leitung zur Auswahl des Chips benötigt. Es werden also, wie in Abbildung 5 ersichtlich, nur sechs Leitungen zur Anbindung benötigt, jedoch ist die Implementierung von SPI aufwendiger als die Verwendung eines parallelen Busses.

Da der Raspberry Pi eine Hardware–SPI–Implementierung hat, ist der größere Implementierungsaufwand nicht relevant und SPI aufgrund der einfacheren Verkabelung vorteilhafter, sodass diese Anbindungsvariante verwendet wird. Die Anschlüsse für die SPI–Daten– und –Taktleitungen am Raspberry PI sind aufgrund der Hardware–SPI–Implementierung fest vorgegeben: Die Datenleitungen müssen an GPIO 9 (SPI_MISO) und GPIO 10 (SPI_MOSI) angeschlossen werden; GPIO 11 muss für die Taktleitung verwendet werden; für die Auswahlleitung kann entweder GPIO 7 oder 8 verwendet werden. Für die Unterbrechungsanforderungsleitung wird GPIO 22 und für die Reset–Leitung GPIO 27 benutzt. Um die Daten schnell genug zu übertragen, sind SPI–Taktfrequenzen im

[5]Einige Telefonanlagen verwenden U_{PN} zum Anschluss von Systemtelefonen oder zur Anbindung von Basisstationen für schnurlose DECT–Basisstationen, da diese Schnittstelle nur ein Adernpaar benötigt.

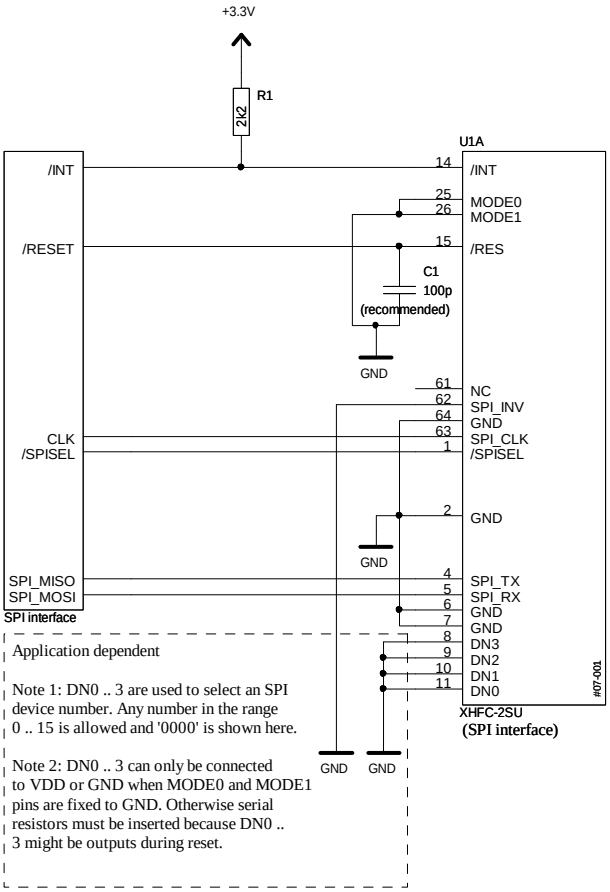


Abbildung 5: Anbindung des XHFC-2SU mittels SPI
Quelle: [7]

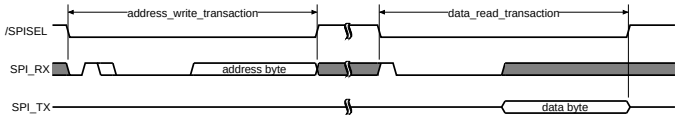


Abbildung 6: Lesezugriff auf ein Register über SPI
Quelle: [7]

MHz–Bereich erforderlich, daher muss bei der Verkabelung darauf geachtet werden, dass diese möglichst kurze Leitungen verwendet, um nicht zu viele Störungen zu verursachen.

Um über SPI auf Register zuzugreifen, wird zunächst immer ein Kontrollbyte übertragen; dieses beschreibt, ob ein lesender oder schreibender Zugriff erfolgen soll, ob die Adresse eines Registers oder dessen Daten übermittelt werden sollen und ob beim Zugriff auf Registerinhalte ein oder vier Byte übertragen werden sollen. Registerzugriffe bestehen also aus vier Bytes. Abbildung 6 zeigt die anhand eines Lesezugriffs: Zunächst wird ein Kontrollbyte, das angibt, dass eine Registeradresse geschrieben werden soll geschickt. Darauf folgt ein Byte mit der Registeradresse. Danach wieder ein Kontrollbyte, das angibt, dass ein Byte Daten aus dem Register gelesen werden soll gesandt. Zuletzt kann dann das Datenbyte, gelesen werden.

Außerdem ist es bei manchen Konfigurationsänderungen erforderlich, abzuwarten, bis diese vom Chip umgesetzt wurden, bevor weitere Register geschrieben werden oder auf Warteschlangen zugegriffen wird. Dafür wird ein Statusregister ausgelesen und aktiv gewartet. Hierbei kann der Lesezugriff verkürzt werden, indem nur beim ersten Zugriff die Adresse geschrieben wird. Danach kann ein verkürzter Lesezugriff erfolgen, bei dem nur das zweite Kontrollbyte, das besagt, dass ein Byte Daten gelesen werden, soll geschrieben und anschließen der Wert des Statusregisters gelesen wird.

Diese Zugriffe müssen, wie im nächsten Abschnitt beschrieben durch den Linux–Betriebssystemkern erfolgen. Ebenso wird auch weitere Software benötigt.

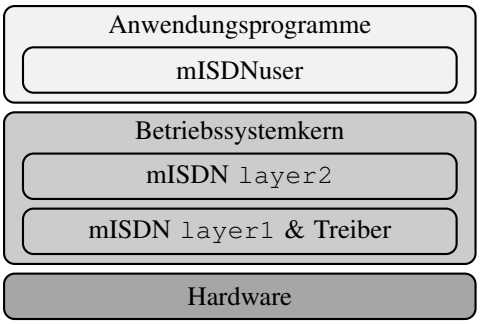


Abbildung 7: Aufbau des mISDN–Protokollstapels

IV. Software für eine ISDN–Telefonanlage

Um die S_0–Schnittstellen für den Betrieb einer Telefonanlage zu verwenden, wird Software benötigt: einerseits muss der Linux–Betriebssystemkern die ISDN–Hardware ansteuern können und andererseite benötigt es Software, die Schicht 3 Funktionen eine Telefonanlage implementiert.

A. Erweiterung des Linux–Betriebssystemkerns

Für Linux gab es im Laufe der Entwicklungsgeschichte mehrere verschiedene ISDN–Protokollstapel. Der aktuellste trägt den Namen mISDN[6] und ist grob in Abbildung 7 dargestellt. Für Anwendungsprogramme gibt es dafür eine Bibliothek mISDNuser, die Programmen ermöglicht, die im Betriebssystemkern enthaltenen mISDN–Funktionen einfacher zu benutzen. Im Betriebssystemkern ist mISDN in Bestandteile der Schichten 1 und 2 und Treiber zur Ansteuerung der ISDN–Hardware aufgeteilt. Einer der Treiber ist `hfcmulti`, dieser wurde für integrierte Schaltungen von Cologne Chip entwickelt und unterstützte ursprünglich HFC–4S, HFC–S8 und HFC–E1–Chips. Der Treiber wurde 2009 um die Möglichkeit, einen XHFC-4SU–Chip in einem bestimmten PowerPC-System anzusteuern, erweitert. Chips der XHFC–Serie werden also prinzipiell schon unterstützt. Der Treiber wurde ursprünglich für HFC–Chips mit PCI-Schnittstelle entwickelt und die Erweiterung unterstützt zwar eine weitere Schnittstelle, die aber nur auf genau genau einer PowerPC–Plattform verfügbar ist. Also ist noch keine Ansteuerung via SPI mit diesem Treiber möglich. Da der Treiber bereits für verschiedene Ansteuerungsarten ausgelegt ist, gibt es hierfür im Treiber

[6] https://www.misdn.eu/wiki/Main_Page

eine abstrakte Schnittstelle in Form von Funktionszeigern als Teil einer Datenstruktur, die eine einzelne ISDN–Karten beschreibt:

```
struct hfc_multi {
  [...]
  void  (*HFC_outb)(struct hfc_multi *hc,
    u_char reg, u_char val);
  u_char  (*HFC_inb)(struct hfc_multi *hc,
    u_char reg);
  u_short (*HFC_inw)(struct hfc_multi *hc,
    u_char reg);
  void  (*HFC_wait)(struct hfc_multi *hc);
  void  (*read_fifo)(struct hfc_multi *hc,
    u_char *data, int len);
  void  (*write_fifo)(struct hfc_multi *hc,
    u_char *data, int len);
  [...]
}
```

Der Programmcode, der diese Funktionszeiger verwendet, um damit Konfigurationen zu setzen, Zähler oder Statusinformationen zu lesen oder auf Warteschlangen zuzugreifen, ist unabhängig von der Art der Anbindung. Daher lag die Vermutung nahe, diese nicht ändern zu müssen, sondern einfach Implementierungen für diese Funktionen, die den Zugriff über SPI durchführen, hinzuzufügen. Die `HFC_outb`– und `HFC_inb`–Funktionen schreiben bzw. lesen ein Byte an bzw. in eine Registeradresse. Die `HFC_inw`–Funktion dient bei HFC–Chips zum Auslesen von 16 Bit–Zählern; da XHFC–Chips nur 8 Bit–Zähler haben, die aber an der gleichen Adresse liegen, muss diese Funktion bei diesen bis auf den Typ des Rückgabewertes gleich der `HFC_inb`–Funktion sein. Das Warten, bis der Chip Konfigurationsänderungen übernommen hat, muss von `HFC_wait` durchgeführt werden. Der lesende und schreibende Zugriff auf die Warteschlangen muss von den Funktion `read_fifo` und `write_fifo` implementiert werden.

Für die Implementierung der Methoden ist es erforderlich, die betriebsystemkerninterne Programmierschnittstelle für SPI zu nutzen. Diese besteht im wesentlichen aus der Funktion `spi_async`, die eine Abfolge von SPI–Operationen startet. Einer ihrer Parameter beschreibt das Gerät, auf dem Operationen ausgeführt werden sollen; der zweite referenzierte eine Datenstruktur beschreibt die Operationen. Diese Funktion stellt eine asynchrone Schnittstelle bereit; oft wird diese jedoch eine synchrone Schnittstelle benötigt. Daher gibt es mit `spi_sync` eine Funktion, die `spi_async` aufruft und dann darauf wartet, dass die SPI–Operationen durchgeführt wurden. Auch um `spi_sync` gibt es Hilfsfunktionen, die die notwendigen Datenstrukturen zur Beschreibung der Operationen anlegen und dann `spi_sync` aufrufen. Mit den Hilfsfunktionen lässt sich `HFC_inb`, wie in Listing 1 gezeigt, implementieren. Zunächst wird ein Buffer angelegt, der die beiden Kontrollbytes und die Registeradresse enthält. Mit `spi_write_then_read` wird dieser Buffer zum Gerät gesandt und anschließend ein Byte gelesen, das den Registerinhalt enthält und zurückgegeben wird.

```
u_char HFC_inb_xhfcspi(
        struct hfc_multi *hc,
        u_char reg) {
    u8 buf[3], ret;

    buf[0] = XHFC_SPI_WRITE
                | XHFC_SPI_ADDR
                | hc->spi_subaddr;
    buf[1] = reg;
    buf[2] = XHFC_SPI_READ
                | XHFC_SPI_DATA;
    spi_write_then_read(hc->spi_dev,
                buf, 3, &ret, 1);

    return ret;
}
```

Listing 1: Implementierung von `HFC_inb` mit einer synchronen Methode

Ein Aspekt, der vor den Anpassungen vermutet und durch Tests bestätigt wurde, ist, dass blockierende SPI–Operationen möglichst nicht in einer Unterbrechungsbehandlungsroutine erfolgen sollten, da diese im Kontext des gerade laufenden Prozesses ausgeführt wird. Der Betriebssystemkern warnt davor, wenn `spi_sync` beim Warten auf das Ende der SPI–Operation versucht einen Prozesswechsel durchzuführen. Als Lösung hierfür gibt es in neueren Linux–Versionen die Möglichkeit, Unterbrechungsbehandlungsroutinen in einem eigenem Prozesskontext laufen zu lassen. Leider zeigte sich, dass der gesamte Treiber in Kontexten läuft, in denen keine Prozesswechsel ausgeführt werden können, da die Schichten 1 und 2 des mISDN–Protokollstapels an vielen Stellen davon ausgehen, dass der Treiber nicht blockieren muss. Daher ergibt sich die Anforderung, alle SPI–Operationen ohne Prozesswechseldurchzuführen.

Als Lösung hierfür liegt aktives Warten nahe, wie in Listing 2, dargestellt. Die Funktion erstellt nun die Datenstrukturen, die zuvor durch die Hilfsfunktion erstellt wurden, selbst und gibt dabei `xhfcspi_complete` als Methode an, die aufgerufen werden soll, sobald die SPI–Operationen durchgeführt wurden[7]. Mit diesen Datenstrukturen wird `spi_async` aufgerufen und darauf gewartet, dass das Statuswort durch die Methode passend gesetzt wird. Diese Funktion wird jedoch vom Treiber mit abgeschalteten Unterbrechungen aufgerufen, sodass das SPI–Subsystem nie die Unterbrechungsanforderung, mit der die Hardware signalisiert, dass der SPI–Datentransfer beendet wurde, erhält. Daher wird die Leseoperation nie beendet und die Schleife nie verlassen[8] Es ist also erforderlich, den gesamten Treiber umzuschreiben, dass er

[7]Eine Beschreibung der Datenstrukturen und Funktionen ist in [8].

[8]Die einzigen Auswege aus dieser Schleife sind: Unterbrechung der Stromzufuhr oder Kurzschließen der Kontakte der P6–Stiftleiste, um der Raspberry Pi neuzustarten.

```
void xhfcspi_complete(void *arg) {
    volatile u_char *status = arg;
    *status = 0;
}

u_char HFC_inb_xhfcspi(
        struct hfc_multi *hc,
        u_char reg) {
    u8 buf[3], ret;
    struct spi_transfer *t;

    buf[0] = XHFC_SPI_WRITE
             | XHFC_SPI_ADDR
             | hc->spi_subaddr;
    buf[1] = reg;
    buf[2] = XHFC_SPI_READ
             | XHFC_SPI_DATA;

    t = hc->spi_transfers;
    t[0].tx_buf = buf;
    t[0].len = 3;
    t[1].rx_buf = &ret;
    t[1].len = 1;
    spi_message_init_with_transfers(
            hc->spi_message, t, 2);
    hc->spi_message->context =
            &hc->spi_status;
    hc->spi_message->complete =
            xhfcspi_complete;
    hc->spi_status = 1;
    spi_async(hc->spi_dev,
            hc->spi_message);
    while (hc->spi_status != 0)
        cpu_relax();

    return ret;
}
```

Listing 2: Implementierung von `HFC_inb` mittels aktivem Warten

die Zugriffe auf das SPI–Gerät nicht sofort ausführt, sondern solange zwischenspeichert, bis er in einem geeignetem Kontext läuft. Hierbei muss jedoch beachtet werden, dass die Reihenfolge der Zugriffe erhalten bleibt, da ansonsten bei Array– und Multi–Registern Probleme auftreten können. Leider muss daher des gesamte Treiber umgeschrieben werden, was

Ein weiterer Bereich, in dem der Treiber modifiziert werden muss, ist die Initialisierung. Der Treiber meldet beim PCI–Subsystem an und teilt ihm eine Liste von Geräte–IDs mit, für die er sich zuständig fühlt. Ähnliches muss er mit dem SPI–Subsystem tun. Dafür werden zwei weitere Funktionen benötigt, die vom SPI–Subsystem aufgerufen werden, wenn ein SPI–Gerät hinzugefügt oder entfernt werden soll. Diese

```
struct spi_board_info spi_dev_info = {
    .modalias       = "XHFC-2SU",
    .platform_data      = NULL,
    .controller_data    = NULL,
    .irq            = 0,
    .max_speed_hz       = 20000000,
    .bus_num        = 0,
    .chip_select        = 0,
    .mode           = SPI_MODE_0,
};

int xhfc_spi_init(void) {
    struct spi_master *spi_master;
    struct spi_device *spi_dev;

    spi_dev_info.irq = gpio_to_irq(22);
    spi_master = spi_busnum_to_master(0);
    if (!spi_master) {
        return -1;
    }

    spi_dev = spi_new_device(
                spi_master,
                &spi_dev_info);
    put_device(&spi_master->dev);
    return spi_dev != NULL ? 0 : -1;
}
```

Listing 3: Anmeldung des SPI–Gerätes beim Linux–Betriebssystemkern

müssen das Gerät initialisieren, die Geräte als ISDN–Hardware bekannt machen und diverse weitere Aufgaben durchführen. Viele der notwendigen Schritte sind von der Art der Anbindung unabhängig. In diesem Bereich ist der Treiber jedoch nicht so gut strukturiert, sodass viele Stellen unnötigerweise davon ausgehen, dass es sich um ein PCI–Gerät handelt. Da SPI anders als PCI keine automatische Erkennung von Geräten ermöglicht, gibt es für jedes System eine Beschreibung, welche Geräte angeschlossen sind. Die Beschreibung für den Raspberry Pi enthält kein SPI–Gerät, da normalerweise keines angeschlossen ist. Daher wird der Programmcode aus Listing 3 benötigt, der den Betriebssystemkern über das SPI–Gerät informiert. Da die SPI–Programmierschnittstelle für Anwendungsprogramme, die SPI–Geräte als Dateien sichtbar macht, sich standardmäßig für alle SPI–Geräte zuständig fühlt, gibt es dadurch einen Konflikt. Also muss diese deaktiviert werden.

B. Anwendungssoftware für die Telefonanlagenfunktionen

Es gibt verschiedene Programme die mISDN verwenden können, um eine Telefonanlage zu implementieren: Asterisk[9] hat hierfür chan_misdn; FreeSWITCH[10] kann mittels

[9] http://www.asterisk.org/
[10] http://www.freeswitch.org/

```
[Extern]
extern
portnum 0

[Intern]
extension
nt
msn 21,22
portnum 1
```

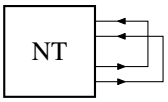
Listing 4: /etc/lcr/interface.conf

FreeTDM mISDN–Geräte verwenden. Nach einigen Tests hat sich gezeigt, dass bei beiden der Schwerpunkt auf Internet–Telefoniefunktionen liegt und bei der Verwendung von mISDN–Geräten nicht erkennbar ist, wieso Probleme auftreten. Es gibt mit Linux–Call–Router jedoch auch ein Programm, dessen Schwerpunkt auf ISDN–Funktionen. Daher ist nachvollziehbar, was geschieht und wobei Probleme auftreten. Da Linux–Call–Router nur rudimentäre VoIP–Funktionen hat, kann man es jedoch nicht direkt benutzen, um Telefone oder Telefonleitungen per VoIP anzubinden. Hierfür würde man noch eine VoIP–Server–Software benötigen.

Linux-Call-Router besitzt verschiedene Konfigurationsdateien. In `/etc/lcr/interfaces.conf` (siehe Listing 4) werden die verschiedenen Schnittstellen beschrieben: Die erste Schnittstelle mit dem Namen `Extern` stellt eine Telefonverbindung zum öffentlichen Telefonnetz dar, was durch das Schlüsselwort `extern` angegeben wird. Die `portnum`–Angabe gibt an, dass sie an der ersten mISDN–Schnittstelle angeschlossen ist. Da keine weitere Angabe erfolgt, wird die Schnittstelle im TE–Modus betrieben. Die zweite Schnittstelle namens `Intern` wird zum Anschluss von Telefonen benutzt. Dies wird durch das Schlüsselwort `extension` angegeben. Hierfür muss die Schnittstelle, wie durch `nt` spezifizert wird, im NT–Modus betrieben werden. Mit dem optionalen `msn`–Angabe wird hier angegeben, das die Telefone an dieser Schnittstelle die Nummer 21 und 22 als Anrufernummer bei Telefonaten verwenden dürfen. Die `portnum`–Angabe gibt hier an, dass die zweite mISDN–Schnittstelle verwendet werden soll.

Damit die Telefonnebenstellen gefunden werden können, müssen für diese Konfigurationsdateien erstellt werden. Hierfür gibt es das Hilfsprogramm `genextension`. Der Aufruf `genextension 21 Intern 21` legt die Nebenstelle 21 an, die an der Schnittstelle `Intern` angeschlossen ist und als Anrufernummer die `21` anzeigen soll, an. Die dadurch in `/var/lib/lcr/extensions/21/` angelegten Dateien können ohne weitere Modifikationen verwendet werden.

Wie genau Anrufe vermittelt werden, legt die Datei `/etc/lcr/routing.conf` fest. Listing 5 gibt diese wieder. Sie besteht aus den Abschnitten `main`, `extern` und `intern`. Linux–Call–Router beginnt im `main` Abschnitt und sucht nach Einträgen, die zur gewählten Nummer oder weiteren Kriterien passen. Bei Telefonanten aus dem öffentlichen Telefonnetz trifft die `extern`–Bedingung zu, bei von Nebenstellen ausgehenden Telefonanten trifft hingegen die `intern`–Bedingung zu. In beiden Fällen ist als Aktion angegeben in einem anderem Abschnitt fortzufahren. In dieser erfolgt die Auswertung auf die gleiche Art und Weise. Wenn an einem der Telefone 0 gewählt wurde, trifft die erste Zeile des `intern`–Abschnitts zu, die das Gespräch ins das öffentliche Telefonnetz vermittelt, dass alle weiteren gewählten Ziffern auswertet. Wenn eine 21 gewählt wurde trifft hingegen die dritte Zeile zu, sodass die `intern`–Aktion das Gespräch zur Nebenstelle 21 vermittelt. Weitere Bedingungen und Aktionen erklärt [9].



XHFC–2SU &
Raspberry Pi

Abbildung 8: Erstes Testszenario: Testschleife

V. Testszenarien

Um die Funktion zu testen, habe ich zwei verschiedene Aufbauten benutzt. Der erste ist minimal und dient nur der Kontrolle der grundlegenden Funktionen der Hardware, der zweite dient zum Test des gesamten Raspberry Pi mit ISDN–Hardware und Telefonanlagensoftware.

A. Testschleife

Bei diesem Testaufbau ist das sendende Adernpaar der S_0–Schnittstelle, wie in Abbildung 8 dargestellt, mit dem empfangenen Adrenpaar verbunden, die S_0–Schnittstelle empfängt also die Daten, die von ihr gesandt werden. Mit dem zu mISDN gehörenden Testprogramme `misdntestlayer1` kann dies geprüft werden. Daher kann damit geprüft werden, ob der Treiber Daten korrekt sendet und empfängt. Mit Fehlerraten um ca 70% war dies leider nicht der Fall.

B. S_0–Bus mit zwei Telefonen

Das zweite Testszenario ist einer nomalen Nutzungssituation nachempfunden. Der S_0–Bus wird hierbei benutzt, um zwei Telefone des Typs Olympia Berlin 2000 anzuschließen. Abbildung 9 stellt dies dar. Da die Stromversorgung über das Evaluationsboard nicht so einfach ist, erfolgt diese stattdessen über einen NTBA — Network Termination for ISDN Basic rate Access, der normalerweise benutzt wird, um die von der Vermittlungsstelle kommend Telefonleitung, die eine sogenannte U_{k0}–Schnittstelle ist, in einen S_0–Bus umzuwandeln. Das in diesem enthaltene Netzteil kann unabhängig von der Signalverarbeitung, die die Umwandlung von der U_{k0}– zur S_0–Schnittstelle vornimmt, betrieben werden, indem am U_{k0}–Anschluss nichts angeschlossen wird. Mit diesem Szenario kann der gesamte Aufbau aus Raspberry Pi, ISDN–Hardware und Software getestet werden.

```
[main]
extern                          : goto ruleset=extern
intern                          : goto ruleset=intern
                                : disconnect cause=31

[extern]
dialing=0,1234                  : intern extension=29
dialing=20-29                   : intern
dialing=81                      : partyline room=42
default                         : disconnect cause=1

[intern]
dialing=0                       : extern
dialing=1                       : extern capability=digital-unrestricted
dialing=21-29                   : intern
dialing=3                       : pick
dialing=5 enblock               : reply
dialing=5                       : reply select
dialing=6 enblock               : redial
dialing=6                       : redial select
dialing=7                       : abbrev
dialing=80                      : vbox-play
dialing=81                      : partyline room=42
dialing=90                      : powerdial
dialing=91                      : callerid
dialing=92                      : calleridnext
dialing=99                      : test
default                         : disconnect cause=1 display="Invalid Code"
```

Listing 5: /etc/lcr/routing.conf

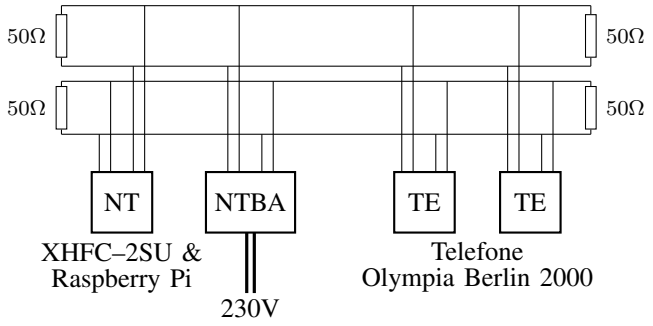


Abbildung 9: Zweites Testszenario: zwei Telefone an einem S_0–Bus

VI. Fazit

Diese Ausarbeitung zeigt, wie sich der Raspberry Pi grundsätzlich als ISDN–Telefonanlage verwenden lässt. Bislang funktioniert dies jedoch noch nicht, da der Treiber noch erweitert werden muss. Ebenso sind noch Experimente erforderlich, wie sich Linux–Call–Router mit einem VoIP–Server kombinieren lässt, um sowohl ISDN als auch Telefonie über das Internet zu verwenden.

Literatur

[1] O. Georg, „Das Diensteintegrierende Digitale Netz (Integrated Services Digital Network; ISDN),“ in *Telekommunikationstechnik*. Springer Berlin Heidelberg, 2000, S. 151–272. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-56985-2_4

[2] ITU-T, *ISDN protocol reference model*, International Telecommunication Union Recommendation I.320, Nov. 1993. [Online]. Available: http://www.itu.int/rec/T-REC-I.320-199311-I/en

[3] ——, *ISDN user-network interface data link layer - General aspects*, International Telecommunication Union Recommendation Q.920, Mar. 1993. [Online]. Available: http://www.itu.int/rec/T-REC-Q.920-199303-I/en

[4] ——, *ISDN user-network interface - Data link layer specification*, International Telecommunication Union Recommendation Q.921, Sep. 1997. [Online]. Available: http://www.itu.int/rec/T-REC-Q.921-199709-I/en

[5] ——, *ISDN user-network interface layer 3 - General aspects*, International Telecommunication Union Recommendation Q.930, Mar. 1993. [Online]. Available: http://www.itu.int/rec/T-REC-Q.930-199303-I/en

[6] ——, *ISDN user-network interface layer 3 specification for basic call control*, International Telecommunication Union Recommendation Q.931, Mai 1998. [Online]. Available: http://www.itu.int/rec/T-REC-Q.931-199805-I/en

[7] *XHFC-2SU Extended ISDN HDLC FIFO controller with two Universal ISDN Ports*, Cologne Chip AG. [Online]. Available: http://www.colognechip.com/xhfc-2su.pdf

[8] L. Torvalds *et al.*, „Linux Device Drivers.“ [Online]. Available: https://www.kernel.org/doc/htmldocs/device-drivers/index.html

[9] A. Eversberg, „Linux-Call-Router.“ [Online]. Available: http://www.linux-call-router.de/download/doc-1.4/linux-call-router-1.4_1.pdf

"Pathfinder" - Raspberry Pi navigiert autonom

B. Sc. Jörn Bellersen
Carl von Ossietzky Universität Oldenburg
Fakultät II Department für Informatik
Seminar: Raspberry Pi inspected - WS2013/14
E-Mail: joern.bellersen@uni-oldenburg.de
28. Januar 2014

***Zusammenfassung*—In dieser Ausarbeitung wird die Entwicklung eines mobilen Roboters für den Außenbereich beschrieben. Dieser Roboter kann mit einer externen Einheit dazu genutzt werden, Mäh- bzw. Sähaufgaben autonom zu bearbeiten. Auf die Konstruktion einer externen Mäh- bzw. Säeinheit wird an dieser Stelle nicht eingegangen. Stattdessen wird das Navigationsproblem im Außenbereich besonders stark thematisiert. Die Entwicklung umfasst ein mobiles System, welches mit Odometrie, GPS und einem Kompass als Sensorik ausgestattet ist und auf einem Raspberry Pi als Steuereinheit aufbaut. Das System wird darauf ausgelegt, eine vorgegebene Grundfläche effizient zu bearbeiten. Dazu wird die Grundfläche in Rechtecke zerlegt, da diese einfach zu bearbeiten sind. Für eine effiziente Bearbeitung wird das Travelling Salesman Problem auf die Rechtecke angewendet. Damit wird der kürzeste geschlossene Pfad über die Grundfläche ermittelt. Durch Minimierung der Wegstrecke wird gleichzeitig die Bearbeitungszeit minimiert.**

Keywords*—*Raspberry Pi, autonomous mobile system, navigation, odometry, Travelling Salesmen Problem

I. Einführung

Bei dem Raspberry Pi handelt es sich um einen Einplatinenrechner, mit der Grundfläche einer Kreditkarte. Es ist möglich den Kleinrechner durch Batterien mit Spannung zu versorgen, wodurch sich dieser für einen mobilen Betrieb eignet. Mit rund 30 € für das Modell A und 40 € für das Model B eignet sich der Raspberry Pi zusätzlich, neben der geringen Größe und Leistungsaufnahme, zur Entwicklung eines mobilen Robotersystems. Dabei steht dem Entwickler eine 32-bit SoC (System on a Chip) Architektur, mit einem 700 MHz ARM11 Prozessor, zur Verfügung. Dadurch wird die Verwendung einer vollwertigen Linux-Distribution als Betriebssystem Unterstützt. Durch den Broadcom BMC2835 SoC verfügt der Raspberry Pi über eine GPIO (General Purpose Input Output) Schnittstelle mit insgesamt 29 Pins. Zusätzlich stellt er zwei USB Host-Ports, sowie analoge und digitale Schnittstellen zur Audio- und Videoübertragung bereit. Auf die GPIO-Schnittstelle kann mit Hilfe von High-Level Linux-Routinen zugegriffen werden. Im Folgenden wird auf die Entwicklung eines mobilen Robotersystems eingegangen, welches die Vorteile des Raspberry Pi als Steuereinheit ausnutzt.

A. Motivation

Roboter und Automaten übernehmen bereits heutzutage Aufgaben des Menschen und erleichtern somit das tägliche Leben. Automaten wie beispielsweise Geschirrspül- oder Waschmaschinen unterstützen den Menschen seit vielen Jahren. Erst seit einiger Zeit unterstützen auch Roboter den Menschen. Aufgrund der vielfältigen Einsatzmöglichkeiten können sie gleichermaßen von Privatpersonen sowie der Industrie eingesetzt werden. Diese Helfer treten z.B. in der Form von Rasenmährobotern auf. Bei Pivatpersonen ersparen diese Roboter das Mähen des Rasens nach Feierabend, wodurch die gewonnene Zeit in andere Aktivitäten investiert werden kann. In landwirtschaftlichen Betrieben können solche Roboter neben dem Mähen auch das Säen von Feldern übernehmen und somit eine Ackerfläche vollautomatisch bestellen und ernten.

B. Aufgabenbeschreibung

Gegenstand dieser Ausarbeitung ist die Entwicklung eines Prototypen eines Roboters, der eine vorgegebene Grundfläche autonom bearbeiten kann. Die Bearbeitung kann beispielsweise das Säen oder das Mähen einer Fläche beinhalten. Es wird dabei ein besonderes Augenmerk auf das Navigationsproblem mobiler Roboter gelegt, da sich die akkurate Positions- und Lagebestimmung des Roboters oft als schwierig und nicht trivial herausstellt. Im Außenbereich können sich Roboter nicht an Fixpunkten, wie beispielsweise Wände oder Gegenstände orientieren, wie es für Roboter im Innenbereich möglich ist. Die Positionsbestimmung muss daher unabhängig von der direkten Umgebung erfolgen, in der sich der Roboter bewegt. Der Roboter wird als Prototyp konstruiert, ohne einen funktionstüchtigen Aktuator, wie beispielsweise ein Mäh- oder Säwerk, zu enthalten.
Das System baut auf einem Raspberry Pi und Navigationselementen zur Positionsbestimmung auf, die zusammen mit einer Spannungsversorgung auf einem fahrbaren Untersatz montiert sind. Das System soll dabei so kostengünstig und simpel gehalten werden wie möglich. Das bedeutet, dass versucht wird, die auf dem Raspberry Pi bereitstehende Peripherie zu verwenden. Daher wird von der Verwendung von externen Erweiterungsplatinen zum Anschließen von analogen Sensoren und Motoren abgesehen und direkt die GPIO-Schnittstelle des Einplatinenrechners verwendet. Diese Schnittstelle beinhaltet keinen Analog-Digital-Wandler und verfügt daher lediglich über digitale Ein- und Ausgänge. Diese Eigenschaft ist bei der Wahl

kompatibler Peripherie zu beachten.
Der Anwender übergibt zur Programmierung des Roboters Koordinaten in einer Konfigurationsdatei, die die Eckpunkte der zu bearbeitenden Grundfläche enthält. Der implementierte Algorithmus berechnet den optimalen Pfad für den Roboter, um sich über die Fläche zu bewegen (Abbildung 1).

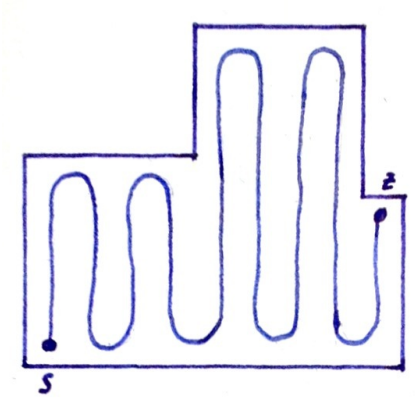


Abbildung 1. Optm. Pfad zum Mähen der Fläche von Punkt S zu Z

Der weitere Aufbau der Ausarbeitung gliedert sich wie folgt: Es wird ein Überblick über aktuelle kommerzielle und private Robotersysteme gegeben. Im Grundlagenkapitel wird der zum Verständnis des vorgestellten Entwurfs dienende theoretische Hintergrund vorgestellt. Außerdem wird der Entwurf und die Architektur des eigenen Systems erläutert. In den Ergebnissen wird das konstruierte System gegen die Anforderungen geprüft. Abgeschlossen wird diese Arbeit mit einem Ausblick auf mögliche Erweiterungen.

II. Anforderungsspezifikation

1) Der Konfigurations- und Installationsaufwand des Roboters selbst soll minimal gehalten werden. Das bedeutet, dass die notwendigen Schritte, um den Roboter (Software) an eine neue Umgebung anzupassen, so gering wie möglich gehalten werden.
2) Die Anpassungsfähigkeit des Systems an verschiedene Grundflächen (zu bearbeitende Rasenfläche) soll im Vergleich zu bestehenden Systemen gesteigert werden. Das bedeutet, dass lediglich Sensorik verwendet wird, die sich an Bord des Roboters befindet. Es soll nicht zusätzlich die zu bearbeitede Fläche präpariert werden.
3) Die Anzahl von mehrfach befahrenen und überfahrenen Bereichen der Grundfläche soll minimiert werden. Die durch das Abfahren des optimalen Pfads verursachten Kosten steigen exponentiell mit der Anzahl der mehrfach bearbeiteten Bereiche.
4) Der Aktuator (beispielsweise das Mäh- bzw. das Säwerk) ist beim Betreten bereits bearbeiteter Bereiche abzuschalten, um Beschädigungen der Grundfläche zu vermeiden (z.B. Übersäung).
5) Es ist der kürzeste geschlossene Pfad über den vorgegebenen Bereich zu wählen, da dies allgemein auch die geringsten zeitlichen Kosten verursacht (geringste Bearbeitungszeit).
6) Der Roboter soll autonom das optimale Bearbeitungsmuster errechnen und anschließend den so erhaltenen Bewegungspfad abfahren. Der Anwender spezifiziert vor dem Einsatz als einzige Tätigkeit das zu bearbeitende Gebiet.
7) Die vom Roboter zu bearbeitenden Bereiche müssen eine Grundfläche aufweisen, deren Eckpunkte durch Kanten verbunden sind, die jeweils entweder einen Winkel von 90 ° oder 270 ° zueinander aufweisen.
8) Die Kanten, die zwei Eckpunkte miteinander verbinden, sind gerade.
9) Die minimale Kantenlänge eines Teilbereiches beträgt mindestens der doppelten Schnittbreite *sb*.
10) Das zu entwickelnde Robotersystem soll in der Lage sein, seine eigene Position mit einer Abweichung im Dezimeterbereich bestimmen zu können.
11) Die Entwicklungskosten sollen so gering wie möglich gehalten werden. Daher soll der Betrag für des gesamte System 200 € nicht überschreiten.

III. Stand der Technik

Es sind bereits einige kommerzielle Robotersysteme auf dem Markt erhältlich, die die Pflege des heimischen Rasens übernehmen können. Zu den Herstellern dieser Systeme gehören beispielsweise Gardena (R40Li) [1] und Bosch (Indego) [2]. Neben den kommerziellen Entwicklungen existieren zudem Systeme, die von Privatanwendern entwickelt worden sind. Zwei Beispiele für nicht kommerzielle Rasenmähroboter sind der Ardumower und der Rasenmähroboter von Werner Stocker [4]. Ardumover ist ein Open-Source-Projekt, welches einen Rasenmähroboter auf Basis eines Arduino-Mikrocontrollers aufbaut [3]. Bei diesen Systemen wird verstärkt auf den Einsatz von Sensorik und der Implementierung einer verbesserten Intelligenz gesetzt.

A. Gardena R40Li

Der Mähroboter der Firma Gardena verfügt über wenig Intelligenz, welches sich in der Mähstrategie widerspiegelt: Er startet von einem beliebigen Punkt und fährt so lange geradeaus, bis er entweder auf ein Hindernis stößt oder eine im Boden vergrabene induktive Begrenzungsschleife erreicht, welche die zu mähende Fläche umrandet. Solch eine Mähstrategie basiert auf dem Zufallsprinzip und benötigt viel Zeit, um die gesamte Rasenfläche einmal komplett abzufahren. Bei diesem Ansatz kann es vorkommen, dass die Rasenfläche in einer vorgegebenen Zeit nicht vollständig und gleichmäßig gemäht wird. In [6] sind die Eigenschaften des Gardena R40Li zusammengefasst:
Vorteile: Roboter findet die Ladestation anhand des Begrenzungskabels. Geeignet für Rasenflächen bis max. 700 m^2.
Nachteile: Benötigt Flächen mit Stromversorgung (Begrenzungsdraht). Maximale Länge das Begrenzungskabels von 250 m. Mäht nach dem Zufallsprinzip. Verhältnismäßig teuer.
Besonderheiten: Fährt automatisch zur Ladestation, wenn die Leistung des Akkus nachlässt. Diebstahlschutz

B. Bosch Indego

Das Gerät der Firma Bosch setzt einen anderen Ansatz als Mähstrategie um: Bei ihm ergibt sich die gefahrene Route nicht nach dem Zufallsprinzip, sondern durch ein geordnetes Abfahren der Rasenfläche. Dazu wird beim Einrichten des Gerätes die Mähfläche abgefahren und gleichzeitig kartographiert. Es wird jedoch wie bei dem R40Li ein Begrenzungsdraht benötigt. Beim Mähen der Fläche fährt der Indego nicht "kreuz und quer", sondern in parallelen Bahnen, da der Roboter über das Wissen verfügt, welche Bereiche des Rasens bereits abgearbeitet wurden. Dieses Wissen bezieht er durch eine Positionsbestimmung mittels Odometrie, die sich in seinen Reifen befindet. Diese Strategie bringt den Vorteil mit sich, dass die Zeit, die benötigt wird um den Rasen vollständig zu mähen, im Vergleich zum R40Li deutlich minimiert werden kann. In [6] sind die Eigenschaften des Bosch Indego zusammengefasst:
Vorteile: Der Roboter findet die Ladestation anhand des Begrenzungskabels.
Nachteile: Benötigt Flächen mit Stromversorgung (Begrenzungsdraht). Unterbricht den Mähvorgang, um die interne Karte nachzuladen. Verhältnismäßig teuer.
Besonderheiten: "Logicut"kartographiert die Umgebung. Diebstahlschutz

C. Ardumover

Der Ardumover verwendet, wie auch die bereits vorgestellten kommerziellen Systeme, einen Begrenzungsdraht zur Definition des Arbeitsbereiches. Dieser Roboter verfügt zudem über eine Vielzahl von Sensoren, die in [5] dargestellt sind. Zu den verbauten Sensoren zählt unter anderem ein so genannter Rasensensor, welcher erkennt, ob sich der Roboter gerade über Rasen oder eine andere Oberfläche bewegt und schaltet dann den Mähmotor ab. Außerdem verfügt der Ardumower über Ultraschallsensoren, mit denen eine Hinderniserkennung ermöglicht wird.
Vorteile: Der Ardumower verwendet eine Vielzahl von Sensoren, wodurch die Genauigkeit der Positionsbestimmung gesteigert wird. Es handelt sich um ein OpenSource-Projekt.
Nachteile: Sehr viele einzelne Komponenten die mit sehr viel Aufwand (elektrisch) verbunden werden müssen.
Besonderheiten: Ist über ein Smartphone steuerbar. Verfügt über einen Rasensensor.

D. Werner Stocker

Der Mähroboter von Werner Stocker verwendet ebenfalls einen Begrenzungsdraht, um die Rasenfläche zu definieren. Zusätzlich verfügt dieses System über Sensoren, um die Position und Orientierung des Roboters zu bestimmen. Die primäre Sensorik zur Positionsbestimmung wird durch Odometrie realisiert. Da dieses Verfahren jedoch die aktuelle Position des Roboters durch Messung der zurückgelegten Wegstrecken relativ zu einem definierten Startpunkt bestimmt, summieren sich die Fehler des Odometers auf. Um diese relativen Fehler zu verringern, werden Sensoren verwendet, die die absolute Position im Bezug auf ein ortsfestes Koordinatensystem angeben. GPS wird hierbei verwendet, um die Position des Roboters mit Hilfe der Messwerte des auf absolute Koordinaten zu beziehen. Dadurch wird der relative Fehler des Odometers verringert.
Vorteile: Kartographie der Rasenfläche und detaillierte Auswertung, welche Bereiche bereits bearbeitet wurden
Nachteile: Hoher Energiebedarf durch verhältnismäßig hohes Gewicht.
Besonderheiten: Bestimmung der Position mit einer Genauigkeit von $0.25m^2$ (laut Hersteller).

Die Stärken und Schwächen der einzelnen Systeme sind in folgender Tabelle dargestellt (Tabelle I). In der letzten Spalte ist der Roboter aufgeführt, der in dieser Ausarbeitung entwickelt wird.

Tabelle I. Übersicht über verschiedene Robotersysteme

System	GPS	Kompass	Odometrie	Begrenzungsdraht	Hindernisdetektion	intelligentes Mähen	Fernsteuerung	WLAN
Gardena R40Li				X	X			
Bosch Indego	X	X	X	X	X	X		
Ardumover	X	X	X	X	X	X	X	
Werner Stocker	X		X	X		X		
Raspberry Pi	X	X	X			X		X

Der zu entwickelnde Roboter soll zu jedem Zeitpunkt seine aktuelle Position kennen. Die Verwendung von Begrenzungsschleifen wird bei der Entwicklung dieses Systems nicht berücksichtigt, da die Spezifikation des Systems darauf ausgelegt ist, den Konfigurationsaufwand zu minimieren und die Anpassungsfähigkeit, im Vergleich zu anderen Systemen, zu steigern. Das System bewegt sich mit Hilfe von Servomotoren fort und bestimmt die aktuelle Position durch Odometrie, die durch eine Integration von GPS und Kompass im Navigationsprozess unterstützt wird. Die Servomotoren werden durch ein puls-weitenmoduliertes Signal angesteuert, wodurch die Drehrichtung und Drehgeschwindigkeit gesteuert wird. Die Navigation erfolgt dabei entlang des optimalen Pfades über die Grundfläche, welcher mit Hilfe des so genannten Travelling Salesman Problem berechnet wird. Der optimale Pfad ergibt sich aus der Zerlegung der Grundfläche in einfach zu bearbeitende Rechtecke. Der Roboter fährt dabei die zerlegten Rechtecke sequenziell ab. Die Entfernungen der jeweiligen Ein- und Austrittspunkte der Rechtecke werden als Kosten interpretiert. Diese Kosten gilt es mit dem Travelling Salesman Problem zu minimieren, da verschiedene Kombinationen in der sequenziellen Anordnung der Rechtecke, verschiedene Kosten für einen Pfad verursachen. Als Besonderheit verfügt dieser Roboter über eine WLAN-Schnittstelle. Im folgenden Abschnitt sollen die benötigten Grundlagen motiviert werden.

IV. Grundlagen

Im folgenden Abschnitt werden die theoretischen Grundlagen des Robotersystems vorgestellt und erläutert.

A. Travelling Saleslman Problem

Das Travelling Salesman Problem (TSP) ist ein Optimierungsproblem, welches in praxisnahen Problemstellungen oft Anwendung findet. Das TSP beschreibt das Problem der Rundreise eines Handelsreisenden auf einem geschlossenen Pfad [17]. Das bedeutet, dass der Ausgangs- und Endpunkt des Reisenden derselbe ist. Die Optimierung des gewählten Pfades über die Orte (Knoten eines Graphen) erfolgt über die verursachten Kosten der Wegstücke zwischen den Orten (Kanten eines Graphen). Das TSP versucht die Summe der verursachten Kosten auf dem gesamten Pfad zu minimieren.

B. Puls-Weiten-Modulation

Puls Weiten Modulation (PWM) ist ein Verfahren, um den Effektivwert einer analogen Spannung digital abzubilden. Die digitalen Zustände werden durch 0 V (logisch 0) und + 5 V (logisch 1) definiert. Repräsentiert werden diese Zustände durch ein Rechtecksignal mit der entsprechenden Amplitude. Um einen Wert zu übertragen, wird die Pulsweite ins Verhältnis zu einer festgelegten Grundfrequenz gesetzt (Abbildung 2). Diese Grundfrequenz beträgt im Modellbau typischerweise 50 Hz (Periodendauer 20 ms). Die Länge des Rechteckpulses innerhalb der Periodendauer der Grundfrequenz ist proportional zum Effektivwert der dadurch erzeugten analogen Spannung. Mathematisch gesehen wird die Rechteckspannung über der Periodendauer der Grundfrequenz integriert. Auf diese Weise kann die analoge Spannung von 0 V bis 5 V eingestellt werden.

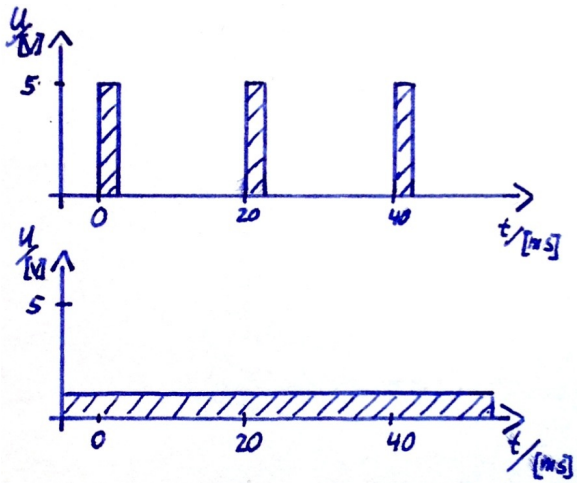


Abbildung 2. Darstellung einer PWM- und Analogspannung

C. Servomotoren

Servomotoren bestehen aus einem kleinen Elektromotor und einer Getriebeeinheit. Durch die integrierte Getriebeeinheit wird aus dem geringen Drehmoment und der hohen Drehzahl des Elektromotors, ein hohes Drehmoment und eine geringe Drehzahl der Antriebswelle erreicht. Servomotoren aus dem Modellbau weisen typischerweise einen Drehwinkel von ± 90 ° auf. Der Drehwinkel wird mechanisch durch einen Bolzen im Getriebe begrenzt, damit das Potentiometer, das zur Detektion des Drehwinkels dient, nicht überdreht wird.

D. Navigationssysteme

In diesem Abschnitt wird eine Übersicht über verschiedene Arten der Positionsbestimmung gegeben.

1) GPS: Das Funktionsprinzip eines gewöhnlichen GPS-Systems besteht darin, die Signale mehrerer GPS-Satelliten (mindestens drei für die reine Positionsbestimmung) auszuwerten. GPS Satelliten übermitteln stetig die aktuelle Position, sowie einen hochakkuraten Zeitstempel und befinden sich in einem Orbit in ungefähr $20{\cdot}10^6$ m über der Erdoberfläche. Elektromagnetische Wellen bewegen sich im Vakuum mit rund 299792458 ms. Daraus folgt, dass Signale ungefähr $\frac{20{\cdot}10^6 m}{299792458\,\frac{m}{s}} \approx 66.71ms$ nach dem Aussenden vom GPS-Empfänger detektiert werden.
Aus den Abweichungen in den Signallaufzeiten, sowie der Phasenverschiebung der einzelnen Satelliten und der jeweils vom Satelliten übertragenen Position auf dessen Orbit, lässt sich die Position des Empfängers auf der Erdoberfläche bestimmen, sowie die Höhe darüber [7]. GPS-Empfänger ohne Mechanismen, die eine Steigerung der Genauigkeit hervorrufen, weisen typischerweise Genauigkeiten im Bereich von ±(10 bis 15) m auf [8]. Eine Möglichkeit der Genauigkeitssteigerung wird durch die Verwendung eines differentiellen GPS-Empfängers gegeben.

2) DGPS: Bei einem differentiellen GPS wird das System um einen ortsfesten GPS-Empfänger erweitert, dessen genaue Position bekannt ist (sog. Referenzpunkt). Solche Referenzpunkte senden ein Funksignal aus, welche die Differenzen aus theoretischer und realer Signallaufzeit beinhalten. Differentielle GPS-Empfänger können dieses Signal empfangen, dekodieren und auswerten. Durch die Einbeziehung der Daten aus dem Korrektursignal ist das mobile GPS-System in der Lage, seinen Fehler beim Empfang der GPS-Signale der Satelliten zu bestimmen und den relativen Positionsfehler zu verringern [7]. Dadurch wird die Genauigkeit gesteigert und liegt laut Aussage von [8] im Bereich von ±(3 bis 5) m für DGPS-Systeme.

3) Odometrie: Die Odometrie beschreibt die Messung der zurückgelegten Wegstrecke eines mobilen Systems mit Hilfe des Drehwinkels der Antriebsräder. Das System kann sich dabei entweder durch Räder oder einen Kettenantrieb fortbewegen. Die Wegstrecke Δs berechnet sich für die Drehung um einen Kreisausschnitt mit dem Winkel φ wie folgt: [9]

$$\Delta s \;=\; \varphi \cdot r \tag{1}$$

Die Variable r repräsentiert den Radius eines Antriebsrades. Es ist der Drehwinkel φ im Bogenmaß anzugeben. Um einen Winkel im Gradmaß in einen Winkel im Bogenmaß umzurechnen wird die folgende Formel benutzt [9].

$$\frac{\varphi_{deg}}{180\,^\circ} \;=\; \frac{\varphi_{rad}}{\pi} \tag{2}$$

Drehen die Antriebsräder nicht um einen Kreisausschnitt, sondern vollführen mehrere vollständige Umdrehungen, so wird die Formel (1) zur Berechnung der Wegstrecke modifiziert.

$$\Delta s = 2 \cdot \pi \cdot r \cdot n \tag{3}$$

Die Anzahl der Umdrehungen, die ein Rad im Messzeitraum zurückgelegt hat, wird durch n repräsentiert. Der Wert für den Drehwinkel φ wurde auf einen statischen Wert von $2 \cdot \pi$ festgelegt, welches im Bogenmaß einer vollständigen Umdrehung entspricht.
Anhand der vorgestellten Grundlagen wird im Folgenden der Entwurf der Soft- und Hardware des Roboters durchgeführt.

V. Entwurf

Der Entwurf des Robotersystems geht auf die verwendeten Komponenten und Lösungsansätze ein. Es wird die Software- und Hardwarearchitektur beschrieben, sowie die Art und Weise, wie diese implementiert wurden.

A. Material & Methoden

1) Sensorik: Der Roboter verfügt über insgesamt drei verschiedene Arten von Sensoren, die abhängig von ihrer Priorität in folgender Abbildung 3 dargestellt sind. Die Prioritäten gliedern sich dabei von oben (größte Priorität) bis unten (niedrigste Priorität).

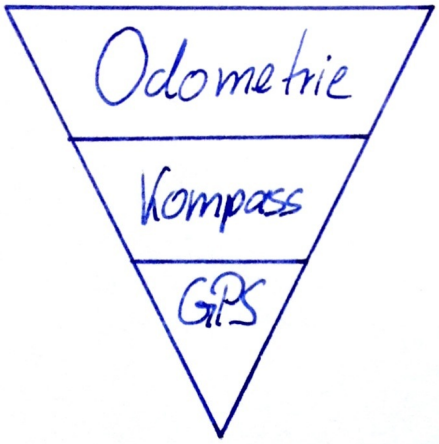


Abbildung 3. Hierarchische Darstellung der verwendeten Sensoren

Es könnten noch weitere Sensoren wie beispielsweise Ultraschallsensoren oder Kameras verwendet werden, um eine Positionsbestimmung durchzuführen. Die Integration und Auswertung dieser Sensoren in das System würde jedoch im Vergleich zum Nutzen zu viel Zeit in Anspruch nehmen und wird daher nicht durchgeführt.

Die Reihenfolge der Sensoren aus Abbildung 3 bezieht sich auf die Gewichtung im Navigationsprozess. Das primäre Navigationswerkzeug bildet die Odometrie. Abweichungen zwischen der berechneten und wahren Position können bei diesem Verfahren durch Schlupf der Antriebsräder und durch Unebenheiten im Untergrund auftreten. Diese relativen Abweichungen werden durch die Verwendung einer Kombination aus einem Kompass und differentiellem GPS verringert. Der Kompass dient dazu, die Abweichung in der Bewegungsrichtung des Roboters zu detektieren. Das GPS wird dazu genutzt, als Referenz genutzt zu werden, falls die Abweichung der Position durch die relative Navigationskomponente größer als der bekannte Fehler des GPS wird. Dadurch wird sichergestellt, dass der Fehler in der Positionsbestimmung im schlechtesten Fall dem bekannten Fehler des GPS entspricht, da über den Fehler der relativen Komponente keine Aussage getroffen werden kann.
Die Umsetzung der Odometrie erfolgt über einen Opto-Reflex-Koppler und einer Encoderscheibe, die jeweils in einem Rad auf der linken und rechten Seite des Roboters untergebracht sind. Auf der Encoderscheibe befinden sich schwarze und weiße Felder, die von dem Opto-Reflex-Koppler abgetastet werden (Abbildung 4). Es befinden sich insgesamt 16 Felder auf der Scheibe, wodurch sich eine Auflösung von $\frac{360\,^\circ}{16} = 22.5\,^\circ$ als kleinstes Inkrement ergibt. Durch das Auswerten dieser Schwarz-Weiß-Übergänge über einer gewissen Zeitspanne kann die Anzahl der Inkremente bestimmt und daraus die zurückgelegte Wegstrecke berechnet werden.

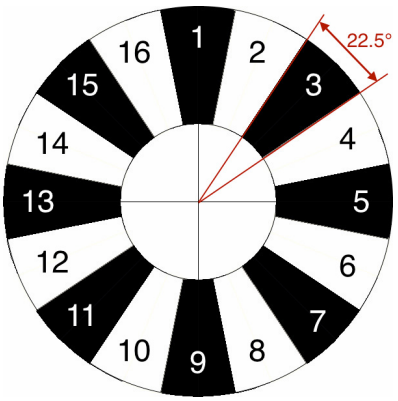


Abbildung 4. Die für die Odometrie genutzte Encoderscheibe

2) Modifizierung von Servomotoren:
Um einen Servomotor zum Antrieb eines Roboters nutzen zu können, muss das Servo modifiziert werden, um eine kontinuierliche Drehbewegung ausführen zu können. Es sind bei den verwendeten RS-2 Servos der Firma Modellcraft [10] zwei Modifikationen vorzunehmen. Die erste Veränderung bezieht sich auf einen Bolzen in der Getriebeeinheit, welcher die Drehbewegung physikalisch begrenzt. Dieser Bolzen ist zu entfernen, um eine kontinuierliche Drehbewegung der Antriebsachse zu ermöglichen. Die zweite Veränderung bezieht sich auf den Holm, der zur Aufnahme der Welle des Potentiometers dient und dieses bei einer Winkeländerung verstellt. Um die Winkelstellung des Potentiometers bei einer Drehbewegung der Antriebsachse nicht mehr zu verändern, ist die physikalische Verbindung der Bauteile zu trennen. Dazu ist der Holm, in den die Potentiometerwelle greift, aufzubohren. Der auf diese Weise entstandene Hohlraum dreht sich nun um die Potentiometerwelle herum und bewegt diese nicht mehr.

3) Ansteurung der Servomotoren: Angesteuert wird ein Servo über drei Leitungen. Eine Leitung liefert eine Span-

nung von 5 V für die Elektronik, sowie den Elektromotor. Eine weitere Leitung dient als Steuerleitung für das PWM-Signal, mit dem die Drehzahl des Motors gesteuert wird. Die dritte Leitung dient als Masse für Versorgungsspannung und Steuersignal.
Die Hardwareschnittstelle zu den Motoren bildet die GPIO-Schnittstelle des Raspberry Pi. Bei den Ausgängen dieser Schnittstelle handelt es sich um digitale Ausgänge die das PWM-Signal erzeugen.
Als Softwareschnittstelle dient ein Modul, das in der Programmiersprache C geschrieben wurde und Software-PWM-Signale erzeugen kann [11]. Zum Übertragen wird diese Schnittstelle als Datei geöffnet und eine Zeichenkette der Form `"P1-GPIOPort = Wert_in_µs / 10"` geschrieben.

B. Architektur

1) Software: Die grobe Darstellung der Softwarearchitektur des Systems ist in der Abbildung 5 abgebildet.

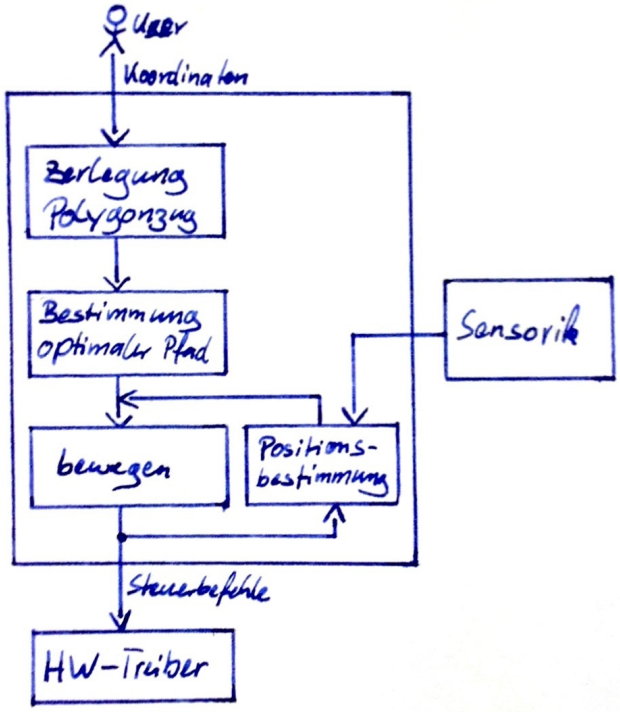


Abbildung 5. Softwarearchitektur des Robotersystems

Die Mensch-Maschinen-Schnittstelle wird durch die Übergabe der Koordinaten der zu bearbeitenden Fläche gebildet. Diese Koordinaten werden als ein geschlossener Polygonzug interpretiert und in Teilbereiche zerlegt. Es wird der kürzeste Weg über die einzelnen Teilbereiche berechnet, um den optimalen Pfad zu erhalten (siehe Abbildung 7). Auf diese Weise entsteht eine Liste von Punkten die nacheinander abgefahren werden, um die Fläche komplett zu bearbeiten. Während der Bewegung des Roboters werden die Daten der Sensoren permanent in den Prozess der Positionsbestimmung eingebunden, um die Steuersignale zu generieren, die an die Motoren gesendet werden.

2) Hardware: In folgendem Diagramm sind die verwendeten Komponenten, sowie die Verbindungen untereinander dargestellt.

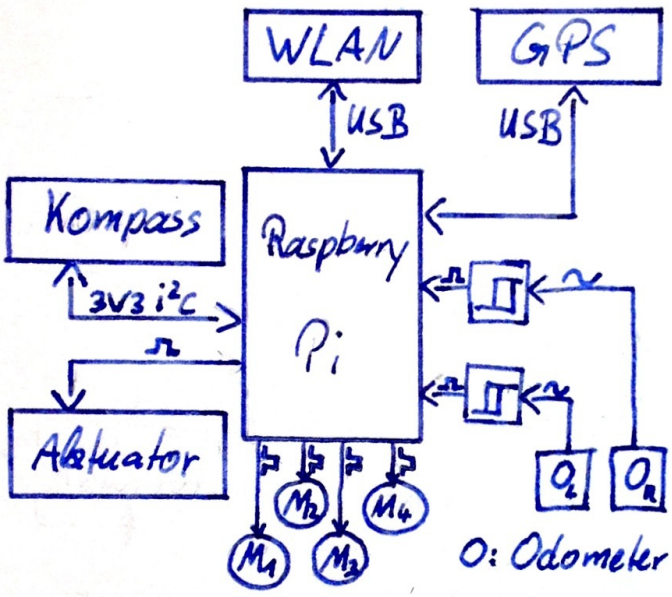


Abbildung 6. Hardwarearchitektur des Robotersystems

Das System besteht aus relativ wenigen Komponenten. Als Intelligenz und Steuereinheit des Roboters wird ein Raspberry Pi Modell B der 2. Revision verwendet. Zur Kommunikation mit der Peripherie wird keine zusätzliche Hardware verwendet, sondern die Stiftleisten auf der Platine des Raspberry Pi. Mit den Stiftleisten werden direkt die Motoren, der Kompass, sowie die Odometereinheiten verbunden. Eine Odometereinheit besteht aus einer auf Papier gedruckten Encoderscheibe, die von einem Opto-Reflex-Koppler abgetastet wird. Das GPS-Modul wird als einziger Sensor über USB ausgelesen. Über USB ist zusätzlich noch ein WLAN-Adapter mit dem Raspberry Pi verbunden, um eine drahtlose Kommunikation mit einem Computer herstellen zu können. Diese Verbindung wird zum Übertragen der Konfigurationsdatei und dem Starten des Mähauftrages benötigt. Zusätzlich wird durch die Verbindung ein drahtloses Monitoring des Roboters ermöglicht.

C. Implementierung

1) Algorithmus: Der Benutzer übergibt der Software eine Konfigurationsdatei, die die GPS-Koordinaten der Eckpunkte der zu bearbeitenden Fläche enthält. Die übergebenen Koordinaten können als Polygonzug aufgefasst werden und bilden eine geschlossene Fläche. Als Restriktion wird vorgegeben, dass die Eckpunkte in einer geordneten Reihenfolge angegeben werden müssen. Der jeweils nächste an den Algorithmus übergebene Eckpunkt ist gleichzeitig auch geographisch gesehen der folgende Eckpunkt. Ein Eckpunkt ist dadurch definiert, dass die an ihn grenzenden zwei Kanten einen Winkel von entweder 90 ° oder 270 ° zueinander aufweisen.

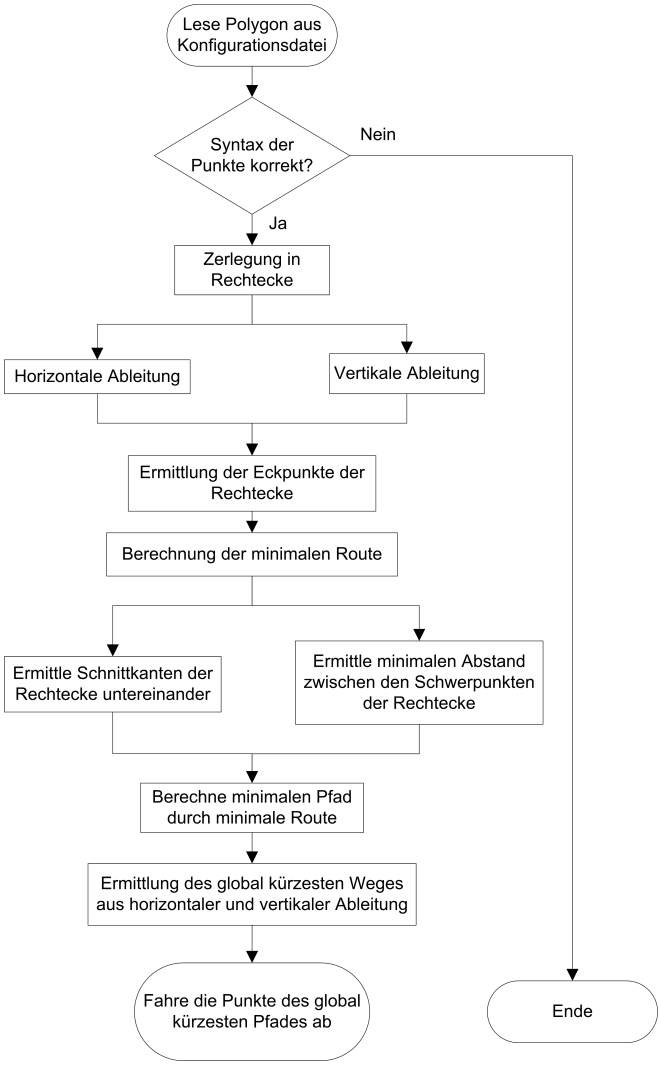


Abbildung 7. Algorithmus

Der in Abbildung 7 angegebene Algorithmus zerlegt den übergebenen Polygonzug in Rechtecke. Die Zerlegung erfolgt dabei parallel auf zwei unterschiedliche Arten (Abbildung 8). Das eine Mal findet die Zerlegung in horizontaler Richtung statt, das andere Mal in vertikaler Richtung. Im Anschluss daran werden die Eckpunkte der erzeugten Rechtecke für beide Ableitungen errechnet. Mit dieser Zerlegungsstrategie soll sichergestellt werden, dass der optimale Pfad zur Bearbeitung der gesamten Fläche gefunden wird.

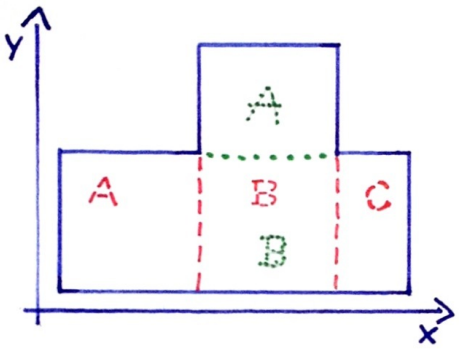


Abbildung 8. Zerlegung horizontale (··) und vertikale (- -) Ableitung

Jedes bei der Zerlegung entstandene Rechteck besitzt folgende Eigenschaften: Jedes Rechteck besitzt vier Eckpunkte, von denen jeweils einer ein Ein- und Austrittspunkt ist. Es gibt keinen Punkt, der gleichzeitig Ein- und Austrittspunkt ist. Außerdem wurde in der Anforderung (9) spezifiziert, dass die Kantenlänge eines Rechteckes $\geq 2 \cdot sb$ ist. Das sequenzielle Abfahren der Rechtecke, unter Berücksichtigung der Ein- und Austrittspunkte, kann als TSP verstanden werden. Dabei wird die kürzeste Route gesucht, die alle Rechtecke einschließt. Am Ende einer Route kehrt der Roboter wieder zum Startpunkt zurück. Mit dem TSP wird für jede Ableitung die Bearbeitungsreihenfolge der Rechtecke getrennt festgelegt. Es wird zusätzlich der minimale Pfad durch die TSP-Route berechnet. Dabei wird jedes besuchte Rechteck mit dem Wert 1 gewichtet. Der Abstand zwischen dem Austrittspunk des vorherigen Rechtecks und dem Eintrittspunkt des nachfolgenden Rechtecks, wird mit dem Betrag der Distanz in [m] gewertet. Anschließend wird der globale minimale Pfad aus den verursachten Kosten der horizontalen und vertikalen Ableitung ermittelt. Dabei wird der Pfad mit den geringsten Kosten gewählt und anschließend abgefahren.

2) Chassis: Die Basis des mechanischen Aufbaus bildet ein Aluminiumgehäuse mit abnehmbarem Deckel und Seitenteilen. Das Gehäuse bietet Einschübe für Europlatinen mit den Maßen 100 mm x 160 mm. Auf einer solchen Europlatine findet die Elektronik, sowie der Raspberry Pi Platz. Die Servomotoren sind direkt mit der Unterseite des Gehäuses verschraubt. Die Räder sind direkt mit den Adaptern, aus dem Zubehör der Servos, verschraubt und an den Antriebswellen der Servos befestigt. Die zwei CNY 70 Opto-Reflex-Koppler [12] sind jeweils am Gehäuse eines sich auf der linken und rechten Seite befindenden Servos montiert. Die Encoderscheibe ist starr im Radinneren montiert. Das analoge Signal des CNY 70 wird mit Hilfe des Schmitt-Triggers CD 4093, der Firma Texas Instruments [13], in ein logisches Signal umgewandelt. Der Maximalwert des Signals beträgt 3.3 V. Als Kompass wird das HMC 5883L 3-Achs-Magnetometer der Firma Honeywell verwendet [14]. Als GPS-Modul kommt das Navilock NL-651 EUSB [15] zum Einsatz. Die Spannungsversorgung wird durch einen Akkumulator mit 5 V Ausgangsspannung und einem maximalen Ausgangsstrom von 2 A realisiert. Der Raspberry Pi wird nicht wie vorgesehen über die Micro-USB-Buchse mit Spannung versorgt, sondern über den +5 V und GND Pin des GPIO-Headers. Um den Raspberry Pi vor einer Überlastung zu schützen, ist eine flinke 680 mA Feinsicherung installiert.
Im Folgenden wird das Robotersystem gegen die Anforderungen geprüft und die Ergebnisse präsentiert.

VI. Ergebnisse

Durch die Verwendung einer Konfigurationsdatei, die die Punkte (GPS-Koordinaten) der Grundfläche enthält, wurde eine Schnittstelle geschaffen, die die Anpassung des Roboters an andere Flächen leicht gestaltet. Damit ist

der Punkt (1) der Anforderungsspezifikation erfolgreich abgeschlossen. Da auf die Verlegung eines Begrenzungsdrahtes bei der Vorbereitung der Grundfläche verzichtet wurde, besteht nicht die Notwendigkeit, die Rasenfläche zu präparieren. Dadurch wurde die Anpassungsfähigkeit, wie in (2) gefordert, im Vergleich zu bestehenden Systemen verbessert. Die Anforderung (3) wurde bisher noch nicht erfüllt, da der Algorithmus noch nicht kodiert wurde. Das Abschalten der Aktuators, wie in (4) gefordert, wurde in der Hardware implementiert, existiert jedoch noch nicht in der Software. Die Anforderung (5) wird erfüllt, da die Grundflüche in mehrere Teilbereiche zerlegt wird und das Abarbeiten dieser Bereiche als TSP aufgefasst wird, welches den kürzesten Weg findet. Der Verwendung einer Konfigurationsdatei und die Anwendung des TSP erfüllt (6). Die Definition des Arbeitsbereiches (Anforderung (7)) ist in einer Syntax festgehalten, jedoch noch nicht im Softwaremodul, zum Einlesen der Konfigurationsdatei, implementiert. Anforderung (8) ist durch die Geometrie des Arbeitsbereiches erfüllt. Die minimale Kantenlänge eines Teilbereiches ist in der Software durch eine Definition (`#define`) festgelegt und erfüllt damit (9). Die Erfüllung der Anforderung (10) konnte noch nicht nachgewiesen werden, da noch keine ausführlichen Testfälle durchgeführt werden konnten. Mit einem Betrag von $\sim$190 € ist auch (11) erfüllt.Des weiteren hat sich durch den Vergleich mit bereits bestehenden Systemen gezeigt, dass GPS als Positionierungssystem nicht die erste Wahl ist, wenn eine Positionierung im Dezimeterbereich benötigt wird. Differentielle GPS-Empfänger, in einem Preisbereich von ~ 50 € weisen unter optimalen Bedingungen Abweichungen von minimal ± 3 m auf [15]. Es hat sich gezeigt, dass sich Odometrie laut [9] eignet, um eine relativ gute Approximation der zurückgelegten Wegstrecke zu erhalten. Wird Odometrie durch GPS und einen Kompass unterstützt, kann die Genauigkeit in der Positionsbestimmung gesteigert werden. Bei dem Abfahren einer Teststrecke auf unwegsamen Gelände hat sich herausgestellt, dass die Kursbestimmung durch den Kompass, aufgrund der Bewegung des Roboters durch Bodenwellen, nicht zuverlässig funktioniert. Durch eine Veränderung der Lage des Roboters im Raum ändert sich auch die Feldstärke, die für die einzelnen Achsen des Roboters bestimmt wird.
Abschließend wird ein Ausblick auf mögliche Erweiterungen und Verbesserungen des Systems gegeben.

VII. Ausblick

Eine mögliche Erweiterung des Systems besteht in der Kompensierung des Kurses. Ein möglicher Ansatz zur Behebung des Problems wäre die Kompensierung der Lage des Kompass im Raum. Zur Lagebestimmung ist ein weiterer Sensor in das System zu integrieren, mit dem die Bewegung des Roboters entlang der Längs- und Querachse detektiert werden kann. Dazu eignet sich beispielsweise ein Drei-Achs-Beschleunigungssensor, der die Orientierung des Roboters in absoluten Koordinaten bestimmen kann. Eine kostengünstige Variante wäre die Verwendung eines ADXL345 Beschleunigungssensors der Firma Analog Devices [16] für <10 €.
Es wurde außerdem in dieser Ausarbeitung lediglich der Algorithmus zur Zerlegung der Grundfläche und Lösung der TSP vorgestellt. Der Algorithmus ist noch in der Programmiersprache *C* zu kodieren.
Zusätzlich wäre eine Fernsteuerung des Roboters sinnvoll, um nicht korrekt bearbeitete Stellen "manuell"nacharbeiten zu können. Diese Steuerung könnte beispielsweise durch ein Smartphone erfolgen, welches die Steuerbefehle über WLAN an den Roboter überträgt.

Literatur

[1] **Gardena R40Li**
http://www.gardena.com/ddoc/GARO/GARO2013_EUde/GARO2013_EUde__1155909-51.pdf
zugegriffen am 2. Dezember 2013

[2] **Bosch Indego**
http://www.bosch-garden.com/de/de/bosch-gartengeraete/gartengeraete/indego-3165140644303-209530.pdf
zugegriffen am 2. Dezember 2013

[3] **ArduMower Projekt**
http://www.ardumower.de/index.php/de/
zugegriffen am 3. Dezember 2013

[4] **Werner Stocker**
http://members.aon.at/wstocke6/Positionsbestimmung2.pdf
zugegriffen am 4. Dezember 2013

[5] **Ardumover Sensorik**
http://www.ardumower.de/images/ardumower_photo.jpg
zugegriffen am 15. Dezember 2013

[6] **Chip.de**
http://www.chip.de/video/Maehroboter-von-Gardena-und-Bosch-Test-Video_63120870.html
zugegriffen am 28. Dezember 2013

[7] **Dr. Jan Wendel**
Integrierte Navigationssysteme - Sensordatenfusion, GPS und Iertiale Navigation, 2te überarbeitete Auflage, S. 84f.
Oldenbourg Verlag München, 2011

[8] **Kowoma.de**
http://www.kowoma.de/gps/Genauigkeit.htm
zugegriffen am 22. Januar 2014

[9] **Odometrie**
http://www.lxrobotics.com/berechnung-der-position-durch-odometriedaten
zugegriffen am 20. Dezember 2013

[10] **Servomotor**
http://www.servodatabase.com/servo/modelcraft/rs-2
zugegriffen am 15. November 2013

[11] **Software-PWM**
https://github.com/richardghirst/PiBits.git
zugegriffen am 21. November 2013

[12] **Opto-Reflex-Koppler**
http://www.vishay.com/docs/83751/cny70.pdf
zugegriffen am 15. November 2013

[13] **Schmitt-Trigger**
http://www.ti.com/lit/ds/symlink/cd4093bc.pdf
zugegriffen am 15. November 2013

[14] **Magnetometer**
http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf
zugegriffen am 16. November 2013

[15] **GPS-Modul**
http://www.navilock.de/produkte/G_61845/merkmale.html
zugegriffen am 11. November 2013

[16] **Beschleunigungssensor**
http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf
zugegriffen am 15. Dezember 2013

[17] **Travelling Salesman Problem**
http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo40.php
zugegriffen am 23. Dezember 2013