FEN LOGIC LTD.

Gertboard Benutzerhandbuch

Gert van Loo und Myra VanInwegen

Revision 2.0

Das Gertboard ist ein GPIO-Erweiterungbord für den Raspberry Pi-Computer. Es besitzt eine Vielzahl unterschiedlicher Komponenten, einschließlich Schalter, LEDs, A/D- und D/A Konverter, einer Motorsteuerung , und einem Atmel- AVR- Microcontroller. Es gibt Testprogramme für das Gertboard, die in C und Python geschrieben und frei im Web verfügbar sind. Dieses Handbuch beschreibt sowohl, wie das Gertboard für verschiedene Anwendungen konfiguriert wird, als auch wie die Testprogramme arbeiten.

Übersetzt ins Deutsche von Hans-Dieter Busch, mit freundlicher Genehmigung der Autoren.

Inhaltsverzeichnis

Gertboard Überblick

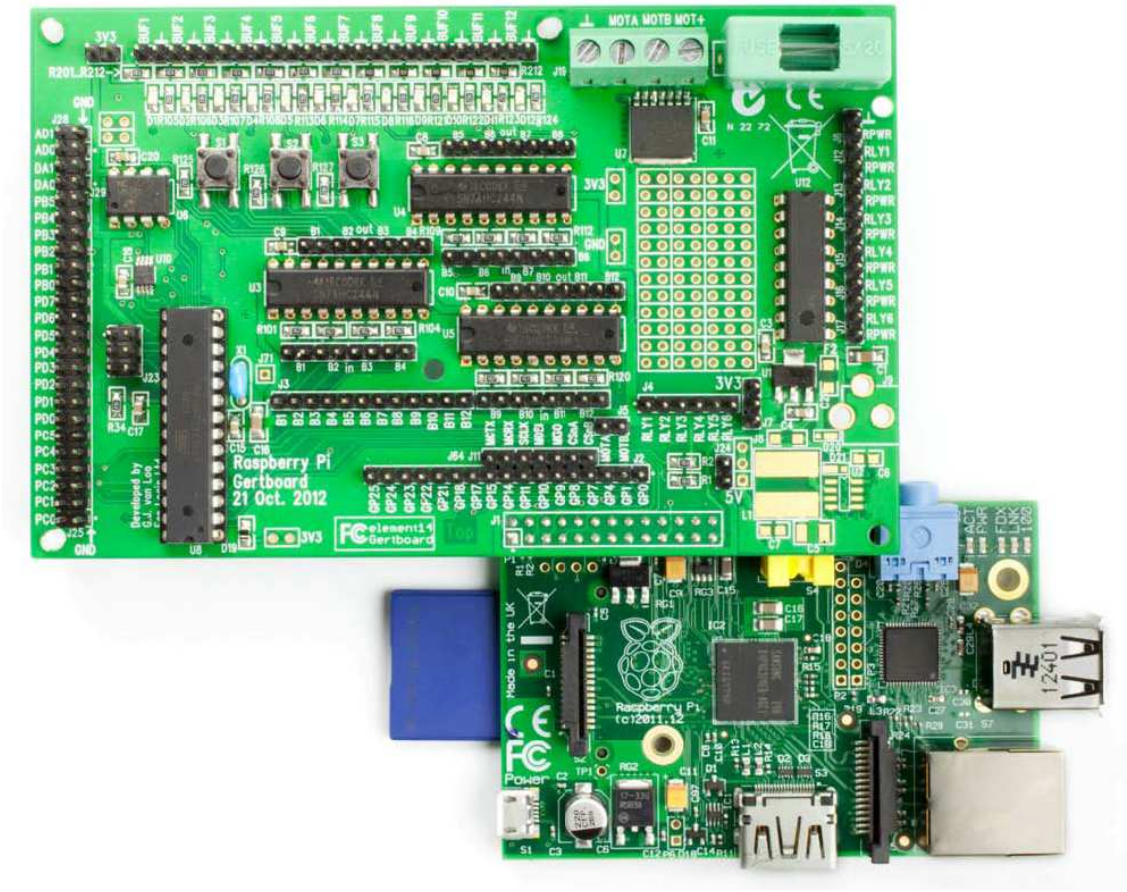


Bild 1: Gertboard und Raspberry Pi

Das Gertboard ist ein Ein-/Ausgangs (I/O)- Erweiterungsbord für den Raspberry Pi-Computer. Es passt auf die GPIO-Pins (General Purpose I/O – Ein-/Ausgänge für allgemeine Zwecke) des Raspberry Pi (die Doppelreihe Pins in der oberen linken Ecke) über einen Sockel auf der Rückseite des Gertboards. Etwas Vorsicht ist beim Verbinden der beiden Komponenten geboten. Es passiert leicht, dass nur eine Reihe von Pins in den Sockeln steckt, aber es müssen beide Reihen verbunden werden. Da das Gertboard die Stromversorgung über diese GPIO-Pins bekommt, benötigt das Raspberry Pi (RPi) eine Stromversorgung, die mindestens 1A Strom liefern kann.

Das Gertboard besitzt eine Reihe von Funktionsblöcken, die miteinander auf vielfältige Weise über Pfostenstecker verbunden werden können. Die Funktionsblöcke sind:

- 12 x gepufferte Ein-Ausgänge (I/O)
- 3 x Druckschalter
- 6 x Open-Kollektor-Treiber (50V, 0,5A)
- 18V, 2A Motorsteuerung
- 28-Pin dual-in-line ATmega-Microcontroller
- 2-Kanal 8, 10 oder 12 Bit digital- zu analog- Konverter
- 2-Kanal 10 Bit analog- zu digital- Konverter

Die Anordnung dieser Blöcke ist in Bild 2 zu sehen.

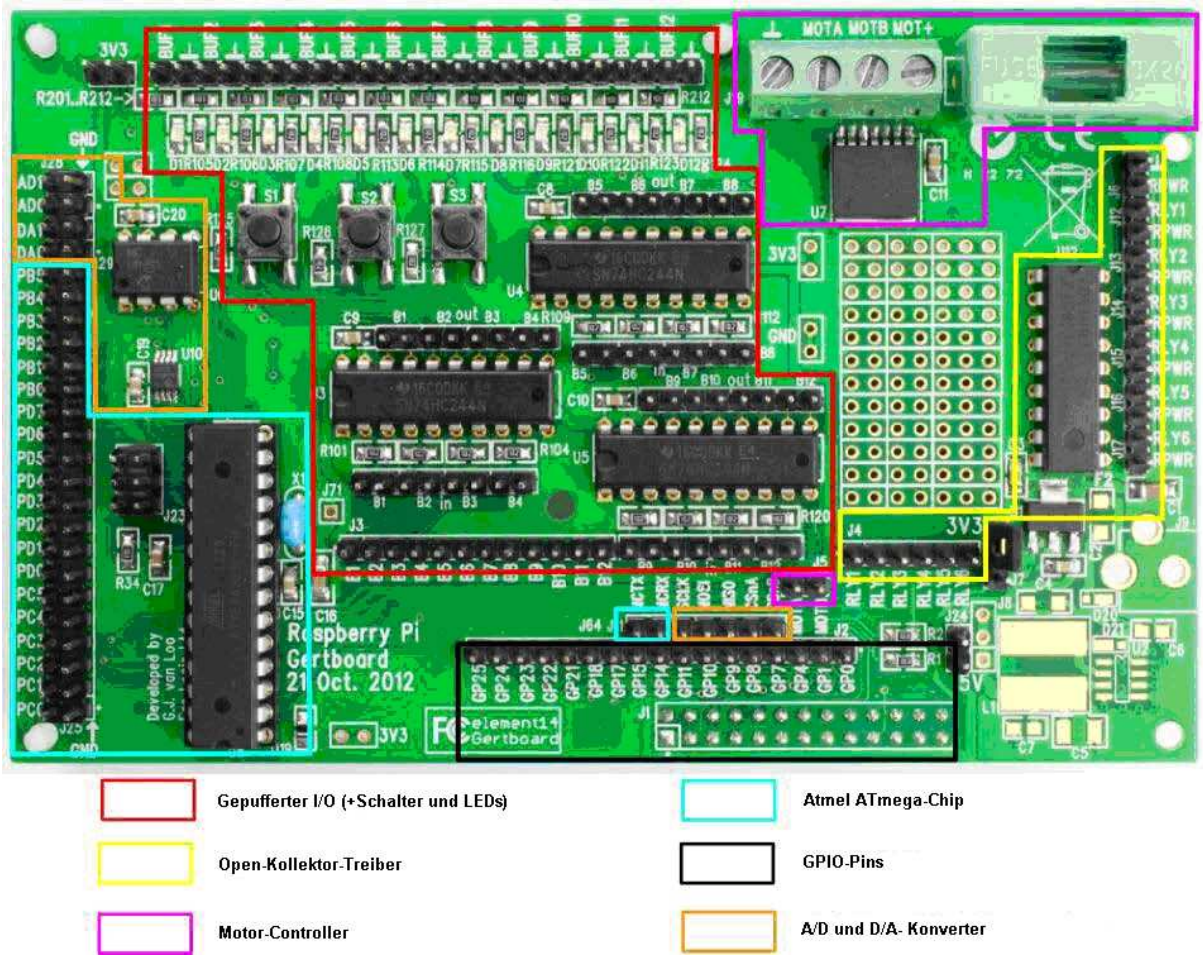


Bild 2: Anordnung der Funktionsblöcke: die Schlüsselblöcke sind farblich umrandet. Bitte beachten Sie, dass die Erscheinungsform mancher Komponenten abweichen kann.

Dieses kommentierte Foto eines voll bestückten Gertboards zeigt, wo die Funktionsblöcke liegen. Einige Funktionsblöcke haben zwei markierte Bereiche. Zum Beispiel zeigen die türkisen Linien den Atmel ATmega-Chip nicht nur um den Chip selbst und die Pins daneben (links unten), sondern auch zwei Pins in der Mitte nahe dem unteren Rand des Bords. Diese beiden Pins sind mit dem Atmel-Chip verbunden und liefern einen einfachen Weg, die GPIO-Signale vom Raspberry Pi (die sich in der „black box“ befinden) mit dem Atmel-Chip zu verbinden.
Es gibt keine anderen Verbindungen (außer Betriebsspannung und Masse) zwischen den einzelnen Funktionsblöcken auf dem Gertboard. Die vielen Kontaktpfosten (Pins) auf dem Bord erlauben die Herstellung dieser Verbindungen über Verbindungsdrähte oder Kontaktbrücken (Jumper). Siehe Bild 11 auf Seite 18 für ein Beispiel, wie die Blöcke miteinander verbunden werden können.

Bild 3: Verbindungsdrähte und –brücken (Jumper). Verbindungsdrähte verbinden zwei Teile des Gertboards miteinander, während Brücken zwei nebeneinander liegende Pins verbinden.

Beschriftung und Diagramme

Wie Sie sehen können, werden Sie beim Herstellen der Verbindungen zwischen den Funktionsblöcken intensiv durch die weiße Beschriftung auf dem Bord unterstützt. Hier möchten wir das Schema, das in Bild 5 gezeigt wird, erläutern. An Hand dieses Schemas zeigen wir Ihnen die Verdrahtung Ihres Gertboards für die Testprogramme.

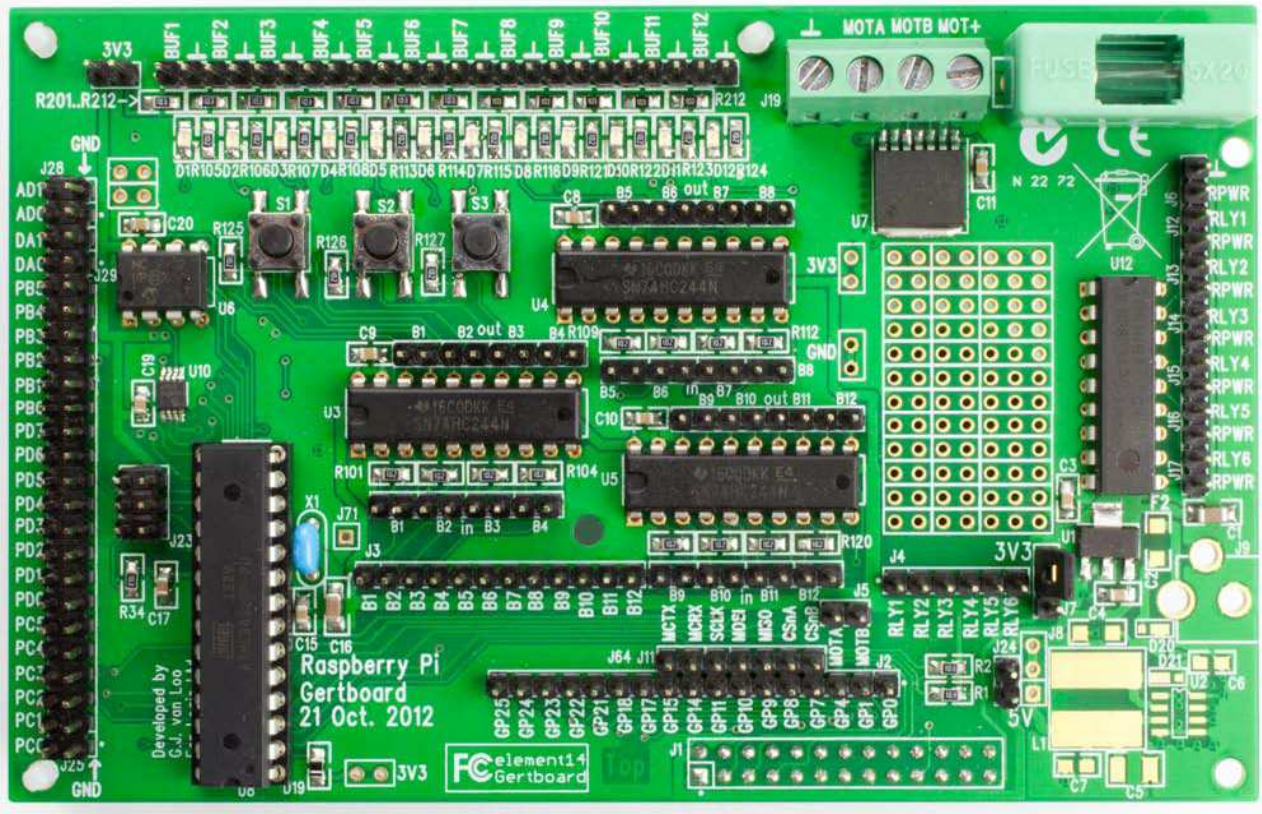


Bild 4: Foto des Gertboards

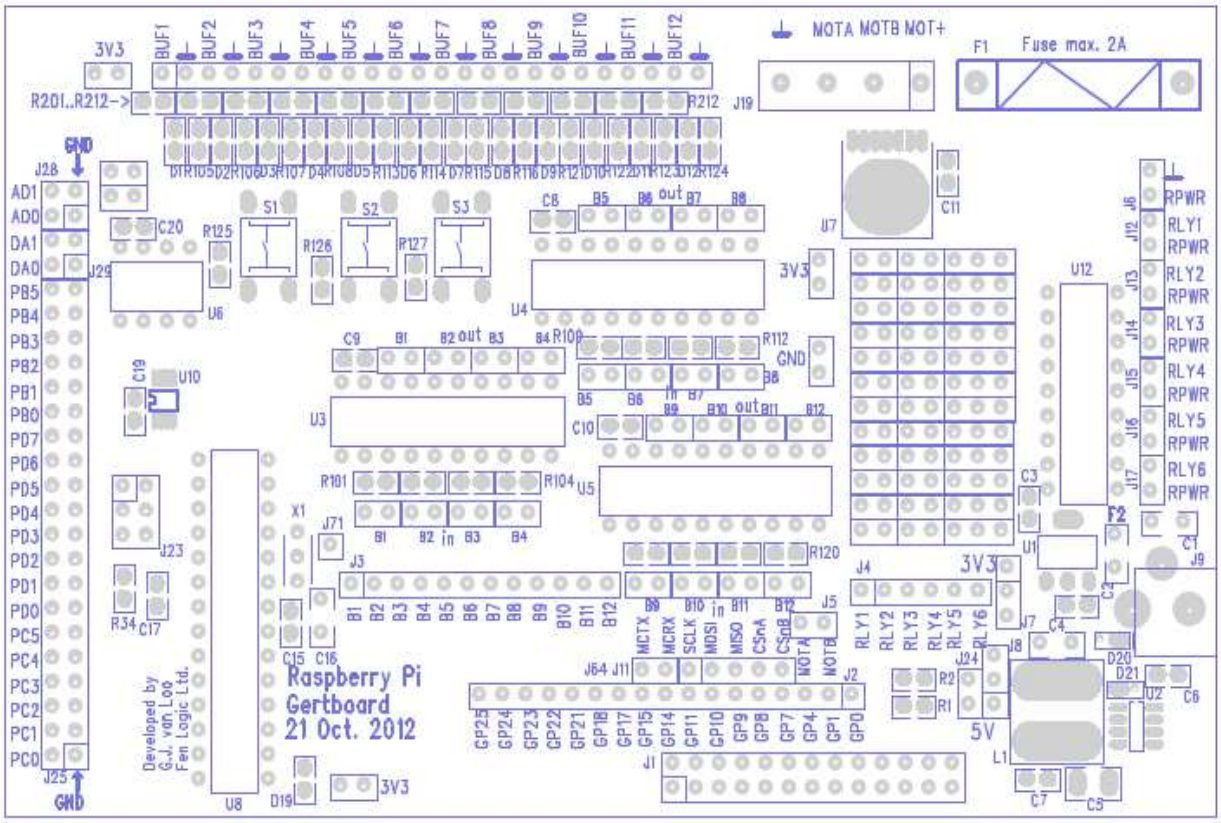


Bild 5: Das Diagramm zeigt ein leeres Gertboard. Die blauen Elemente korrespondieren mit den weißen Linien und dem Text, die grauen Elemente korrespondieren mit den silberfarbenen Anschlüssen auf dem Bord.

Das Diagramm in Bild 5 wurde aus den Daten erstellt, die für das Design des Gertboard benutzt wurden. Das Blau im Diagramm zeigt die weißen Linien und den Text auf dem Bord. Im Schema grau sind alle Kontaktstellen auf der Oberseite des Bords. Diese blauen und grauen Angaben benutzen wir als Basis für unseren Verdrahtungsplan, der Ihnen die Pins zeigt, die für jedes Testprogramm miteinander verbunden werden müssen. Dieses Schema ist wesentlich klarer als ein Foto von einem voll bestückten und verdrahteten Bord.

Sehen Sie sich den weißen Text auf dem Foto in Bild 4 näher an (oder den weißen Text auf Ihrem Gertboard oder den blauen Text in Bild 5). Diese Beschriftung liefert die erforderlichen Informationen, um die verschiedenen Blöcke auf dem Gertboard miteinander zu verbinden. Fast alle Komponenten haben eine Beschriftung - und wichtiger, die Kontaktpfosten (Pins) haben eine Beschriftung. Es ist nicht notwendig, sich zu sehr für einige Komponenten zu interessieren, wie z. B. Widerstände und Kondensatoren (bezeichnet mit R*n* und C*n*, wobei *n* eine Zahl ist). Jedoch sind die Bezeichnungen für Kontaktpfosten, ICs, Dioden und Schalter wichtig.

Dioden sind mit D*n* bezeichnet. D1 bis D12 sind für Sie interessant, das sind die LEDs. Die LEDs sind nahe der linken Oberseite des Bords platziert. Die Beschriftung ist ein wenig eng. Jede LED hat einen Widerstand daneben und folglich hat jedes D*n* ein R*m* daneben. Die LEDs sind leicht zu finden, wenn das Bord mit Spannung versorgt ist, da sie eine Reihe von hellen roten Lichtern bilden. Siehe unten, im Kapitel Stromversorgung auf dem Gertboard (Seite 9) für Informationen, wie die Versorgung des Gertboards erfolgt.

Druckschalter sind beschriftet mit S1, S2 und S3 (sie sind gleich in der Nähe der LEDs angeordnet).

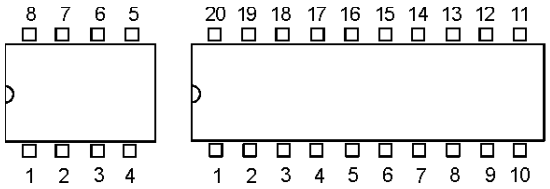


Bild 6: Zwei Beispiele von ICs – ein 8-Pin und ein 20-Pin dual-inline-Gehäuse (DIP). Bei diesem Gehäusetyp ist Pin 1 immer das erste Pin entgegen dem Uhrzeiger von der Markierungskerbe.

Integrierte Schaltkreise (ICs oder Chips), werden mit U*n* bezeichnet Z. B. sind die Ein-/Ausgabe-Puffer (I/O buffer) Chips U3, U4 und U5 (diese sind nahe der Mitte auf dem Bord), während der Atmel Microcontroller U8 ist (dieser ist unterhalb und links von U3 bis U5). Es ist wichtig, die IC Pin-Nummerierung zu verstehen. Wenn der Chip so angeordnet ist, dass sich die halbrunde Markierung links befindet, ist Pin 1 das linke Pin in der unteren Reihe. Pin-Nummern werden entgegen dem Uhrzeigersinn hochgezählt, wie in Bild 6 gezeigt. Wenn man das weiß, kann man die Schaltpläne im Anhang A immer auf die ICs auf dem Gertboard beziehen.

Pfostenleisten sind eine oft benutzte Komponente auf dem Gertboard. Sie sind mit J*n* bezeichnet. Z. B. gibt es eine Anzahl von Pfostenleisten entlang der linken Seite des Bords. Diese erlauben Ihnen den Zugriff auf die drei ICs auf der linken Seite des Bords: J28 ganz oben für den analog- zu digital- Chip, J29 darunter für den digital- zu analog- Chip, und J25 darunter für den Atmel-Microcontroller. Es ist ein bisschen schwierig, die Grenzen zwischen diesen Steckern auf einem voll bestückten Bord zu erkennen; Es ist viel klarer auf dem Schema mit der blauen und grauen Beschriftung im Bild 5 zu erkennen. Auf dem Gertboard ist von jedem Stecker mit mehr als zwei Pins das Pin 1 mit einem Quadrat um das Pin und einem Punkt daneben markiert. Die Punkte sind sehr hilfreich bei einem bestückten Bord, aber sie erscheinen nicht im Bild 5, so dass Sie hier die Quadrate nutzen können, um Pin 1 zu finden.

Nicht jeder beschriftete J*n* hat mehrere Pins. J1, am unteren Rand des Bords gelegen, ist der Stecker, der das Gertboard mit dem Raspberry Pi verbindet. J19, am oberen Rand des Bords (rechts der Mitte) ist ein Block von Schraubanschlüssen, die Ihnen den Anschluss einer Stromversorgung und eines Motors erleichtern.

Stromversorgung auf dem Gertboard

Die Pins für die Stromversorgung sind mit dem Spannungswert beschriftet, z. B. 5V oder 3V3 (das bedeutet 3,3V). Eine 5V-Versorgung kommt vom Raspberry Pi auf das Bord. Wenn Sie diese Spannung benötigen, kann sie vom unteren Pin (beschriftet mit 5V) am Stecker J24 auf der rechten unteren Ecke des Bords abgegriffen werden. Masse ist beschriftet mit GND oder dem Symbol ⊥.

Die Versorgungsspannung (die Spannung, die als logisch 1, Ein oder high fungiert) beträgt 3,3V. Diese wird aus dem 5V-Pin im Stecker J1 durch die Komponenten in der rechten unteren Ecke des Bords erzeugt. Um die 3,3V-Versorgung an die Komponenten auf dem Gertboard zu liefern, müssen Sie eine Brücke (Jumper) auf die oberen beiden Pins von Stecker J7 stecken. Er ist in der Nähe der rechten unteren Ecke des Bords, siehe Foto und Schema in Bild 7. Die Open-Kollektor- und Motor-Controller können höhere Spannungen verarbeiten und haben Anschlüsse für eine externe Stromversorgung.

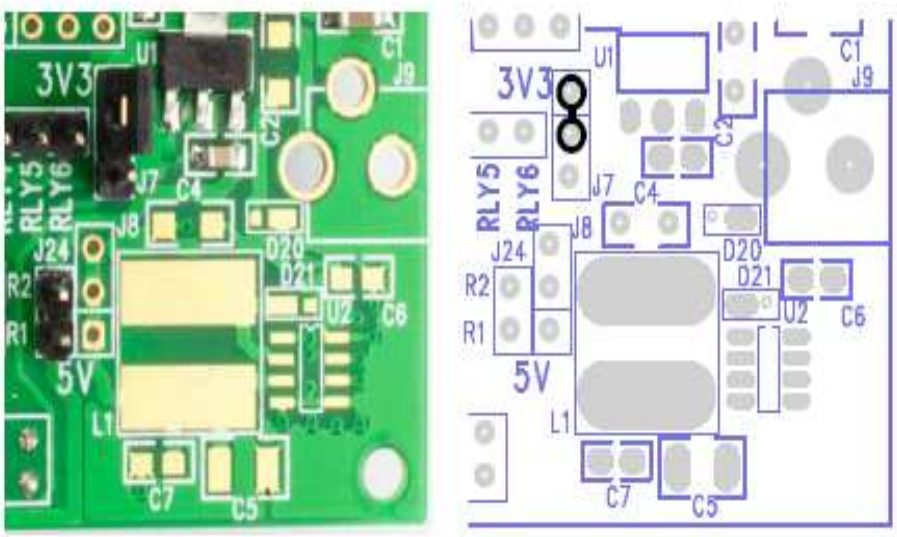


Bild 7: Jumper für die Stromversorgung gesteckt in J7: Foto links, Diagramm rechts

Das Schema auf der rechten Seite von Bild 7 ist unser erstes Beispiel eines Verdrahtungsplanes basierend auf dem blauen und grauen Bordschema. Diese Schemata markieren die Pins mit schwarzen Kreisen um die entsprechenden Pins auf dem Bord und zeigen Verbindungen mit schwarzen Linien zwischen den Kreisen an. Ob dabei ein Verbindungsdraht oder ein Jumper benutzt wird, wird hier nicht dargestellt. Generell sollte eine Brücke (Jumper) benutzt werden, wenn die zwei zu verbindenden Pins direkt nebeneinander liegen.

GPIO Pins

Der Stecker J2, rechts vom Text “Rasberry Pi Gertboard”, ermöglicht den Zugriff auf alle I/O-Pins des GPIO-Steckers. Der Stecker J1 hat 26 Pins (der Stecker, der das Gertboard mit dem Rasberry Pi verbindet), aber der Stecker J2 hat nur 17 Pins: 3 der Pins von J1 sind Stromversorgung (3,3V and 5V) und Masse, 6 Pins sind DNC (do not connect – nicht verbinden). Die Beschriftung dieser Pins, GP0, GP1, GP4, GP7 usw., sieht anfangs ein wenig willkürlich aus, weil es offensichtlich ein paar Lücken gibt und die Zahlen nicht mit den Pin-Nummern auf dem GPIO-Stecker J1 übereinstimmen. Diese Beschriftungen sind jedoch wichtig: Sie korrespondieren mit den Signalnamen, die vom BCM2835, dem Prozessor auf dem Raspberry Pi (RPi) benutzt werden. Das Signal GPIO*n* im BCM2835- Datenblatt korrespondiert mit dem Pin mit der Bezeichnung GP*n* auf dem Stecker J2. Zumindest war das so bei der ersten Version des Raspberry Pi (“rev1”). Beginnend ab September 2012 wurde die Version 2 des Raspberry Pi (“rev2”) ausgeliefert. Bei rev2 des RPis wurden einige der GPIO-Pins geändert. Der GPIO-Port, der gewöhnlich durch GPIO21 gesteuert wurde, wird nun durch GPIO27 gesteuert; Die Ports, die gewöhnlich durch GPIO0 and GPIO1 gesteuert wurden, werden nun durch GPIO2 and GPIO3 gesteuert. Der Rest blieb derselbe. Die ersten drei Spalten von Tabelle 1 unten fassen die aktuelle Situation zusammen.

Einige der GPIO-Pins haben eine alternative Funktion, von der in manchen Testprogrammen Gebrauch gemacht wird. Diese werden ebenfalls in Tabelle 1, in den letzten beiden Spalten gezeigt. Die Ports, die in der Spalte “Alt function” (alternative Funktion) keine Angaben haben, werden nur für allgemeine Ein-/Ausgabezwecke benutzt. In den C-Testprogrammen benutzen wir Makros, um Zugriff auf die alternativen Funktionen der GPIO-Ports zu erlangen. Dies wird im Kapitel für analog- zu digital- und digital- zu analog- Konverter (D/A- und A/D- Test in C, Seite 35) erklärt. In Python benutzen wir Packages (Programmpakete), um Zugriff auf die alternativen Funktionen zu erhalten.

Wir erwähnen die I^2C-Bus-Benutzung von GPIO0 und GPIO1 (oder GPIO2 und GPIO3 für rev2 RPis) in Tabelle 1 nicht deshalb, weil der I^2C-Bus in den Testprogrammen benutzt wird, sondern weil jeder einen 1800Ω pull-up-Widerstand auf dem Raspberry Pi hat, und das ihre Benutzung durch die Druckschalter verhindert (siehe Kapitel „gepufferter I/O, LEDs und Druckschalter“ für weitere Informationen).

Label on GB	Port on RPi1	Port on RPi2	Alt function (which alt)	Purpose
GP0	GPIO0	GPIO2	SDA	I^2C bus
GP1	GPIO1	GPIO3	SCL	
GP4	GPIO4	GPIO4		
GP7	GPIO7	GPIO7	SPI_CE1_N (alt 0)	SPI bus
GP8	GPIO8	GPIO8	SPI_CE0_N (alt 0)	
GP9	GPIO9	GPIO9	SPI_MISO (alt 0)	
GP10	GPIO10	GPIO10	SPI_MOSI (alt 0)	
GP11	GPIO11	GPIO11	SPI_SCLK (alt 0)	
GP14	GPIO14	GPIO14	TXD0 (alt 0)	UART
GP15	GPIO15	GPIO15	RXD0 (alt 0)	
GP17	GPIO17	GPIO17		
GP18	GPIO18	GPIO18	PWM0 (alt 5)	pulse width modulation
GP21	GPIO21	GPIO27		
GP22	GPIO22	GPIO22		
GP23	GPIO23	GPIO23		
GP24	GPIO24	GPIO24		
GP25	GPIO25	GPIO25		

Tabelle 1: Zuordnung GPIO-Ports zu "GP" Beschriftung und alternative Funktionen der GPIO-Ports (GB bedeutet Gertboard, RPi1 bedeutet Raspberry Pi rev1, RPi2 bedeutet Raspberry Pi rev2)

Schemata

Obwohl es Schaltpläne oder Schemata für einige der Funktionsblöcke des Bords im Handbuch gibt, sind sie doch Vereinfachungen der aktuellen Schaltungen. Während diese vereinfachten Schemata und die Erläuterungen im Text für die meisten Anwendungen des Gertboards gut genug sind, wird es gelegentlich Fragen geben, die nur dann beantwortet werden können, wenn man das Bord genau kennt. Darum haben wir am Ende des Handbuchs im Anhang A die genauen Schaltpläne veröffentlicht. Diese Seiten sind im Querformat. Die Seitennummern (A-1, A-2 usw.) sind in der linken unteren Ecke der Seiten (wenn die Seite mit der Schrift aufrecht gehalten wird).

Testprogramme - Überblick

Es gibt Testprogramme für das Gertboard, geschrieben in C und in Python. C erlaubt den direktesten Zugriff auf die Funktionen des Gertboards, aber es ist für den Anfänger keine so leichte Programmiersprache. Verschiedene Programmpakete wurden erstellt, um in Python geschriebenen Programmen den Zugriff auf die GPIO-Pins des Raspberry Pi und die alternativen Funktion dieser Pins, wie den seriellen peripheren Interface (SPI)-Bus und die Pulsweitenmodulation (PWM) zu ermöglichen. Indem Sie diese Programmpakete einsetzen, können Sie die meisten Funktionen des Gertboards mit Python benutzen. Im Moment der Erstellung dieses Handbuches ist die einzige nicht mit Python programmierbare Hauptfunktion der Atmel-Microcontroller.

C Code - Überblick

Um die Gertboard C-Software herunterzuladen, gehen Sie zu
`http://www.gertbot.com/other_products.html` oder
`http://www.element14.com`
und suchen Sie nach "Gertboard", indem Sie das Eingabefeld oben auf der Webseite benutzen. Der Link, den Sie anklicken sollten, sollte so ähnlich lauten wie "Application Library for Gertboard". Von dort können Sie die Datei herunterladen, die den C-Code enthält; Sie hat einen Namen wie etwa `gertboard_sw_20120725.zip`. Wie Sie erkennen, handelt es sich um eine zip-Datei, die Sie anschließend auspacken müssen.

Um die Software zu erhalten, platzieren Sie die Datei an die gewünschte Stelle für die Gertboard-Software auf Ihrem Raspberry Pi-Computer. Anschließend entzippen Sie die Datei, indem Sie die folgende Kommandozeile in einem Terminalfenster des RPi eintippen (ersetzen Sie den Dateinamen aus dem Beispiel durch den tatsächlichen Namen der zip-Datei, die Sie heruntergeladen haben):

```
unzip gertboard_sw_20120725.zip
```

Es wird ein neues Verzeichnis angelegt: `gertboard_sw`. Wechseln Sie in dieses Verzeichnis (mittels `cd gertboard_sw`) und lassen Sie den Inhalt anzeigen (`ls`). Sie sehen eine Reihe von C-Dateien und eine Make-Datei. C-Dateien sind Programmdateien (Quellcode), die erst übersetzt (compiliert) werden müssen, bevor sie auf dem Prozessor ausgeführt werden können. Im Fall des Raspberry Pi ist der Prozessor ein ARM11. Die Make-Datei sorgt für die Übersetzung des C-Quellcodes in ein ausführbares Programm. Tippen Sie:

```
make all
```

Dieser Befehl übersetzt den Quellcode in ein ausführbares Programm. Um es zu starten (z. B. das Programm `leds`, das die LEDs testet), tippen Sie:

```
sudo ./leds
```

Das `sudo` muss verwendet werden, weil der Zugriff auf die GPIO-Ports spezielle Rechte erfordert. `sudo` stattet das Programm `leds` mit diesen Rechten aus. Die Eingabe ./ vor `leds` bedeutet den Aufruf des Programms aus dem aktuellen Verzeichnis.

Jeder Funktionsblock hat mindestens ein Testprogramm, das mit ihm funktioniert. Jedes Testprogramm wird aus zwei oder mehr C-Quelldateien übersetzt. Die Datei `gb_common.c` (die eine Header-Datei `gb_common.h` referenziert) enthält Code, der von allen Funktionsblöcken auf dem Bord benutzt wird. Jedes Testprogramm hat eine C-Datei mit speziellem Code nur für diesen Test (folglich finden Sie die Hauptfunktion `main` hier). Einige Testprogramme benutzen eine spezielle Schnittstelle (interface) (z. B. den SPI-Bus). Diese Testprogramme haben eine zusätzliche C-Datei, die spezifischen Code für dieses Interface enthält (diese Dateien sind `gb_spi.c` für den SPI-Bus und `gb_pwm` für den Pulsweitenmodulator).

In jedem Abschnitt über die einzelnen Funktionsblöcke ist der spezifische Code für den Block erläutert. Da alle Testprogramme die gemeinsame Quelldatei `gb_common.c` verwenden , wird hier ein Überblick über diese Datei gegeben. Um das Gertboard über die GPIO-Ports zu benutzen, muss das Testprogramm zuerst `setup_io` aufrufen. Diese Funktion belegt diverse Speicherbereiche (Arrays) und ruft anschließend `mmap` auf, um die Arrays mit den Geräten zu assoziieren, die gesteuert werden sollen, wie z. B. den GPIO-Port, SPI-Bus, PWM (Pulsweitenmodulator) usw. Im Resultat schreibt das Programm Daten in diese Arrays, um die Geräte zu steuern oder sendet Daten zu ihnen oder es liest Daten aus den Arrays, um Statusbits abzufragen oder Daten von den Geräten zu lesen. Am Ende eines Testprogramms sollte `restore_io` aufgerufen werden, das den belegten Speicher wieder freigibt.

Makros

In `gb_common.h, gb_spi.h` und `gb_pwm.h` sind eine Zahl von Makros, die den verschiedenen Teilen der belegten Speicherbereiche (Arrays) sprechende Namen geben. Diese Makros werden benutzt, um alles mögliche zu tun, vom Setzen eines GPIO-Ports als Eingang oder Ausgang bis zur Steuerung der Taktfrequenz des Pulsweitenmodulators.

Makroname	Typ	Zweck	Seite
`INP_GPIO(n)`	E	aktiviert GPIO-Pin n (als Eingang)	13
`OUT_GPIO(n)`	E	benutzt nach INP_GPIO(n), setzt Pin n als Ausgang	13
`SET_GPIO_ALT(n, a)`	E	nach INP_GPIO(n), setzt alternative Funktion für Pin n	29
`GPIO_PULL`	W	setzt den pull-Code (pull = ziehen)	19
`GPIO_PULLCCLK0`	W	Auswahl, welchen Pins der Pull-Code zugeordnet ist	19
`GPIO_IN0`	R	Eingangswerte lesen	19
`GPIO_SET0`	W	Auswahl, welche Pins auf 1 (high) gesetzt werden	21
`GPIO_CLR0`	W	Auswahl, welche Pins auf 0 (low) gesetzt werden	21

Tabelle 2: Allgemein benutzte Makros, ihr Typ, Zweck und Ort in diesem Handbuch.

Tabelle 2 zeigt eine Zusammenfassung der öfter benutzten Makros und gibt die Seitennummer an, auf der die Makros näher erläutert werden. Der Typ gibt an, wie das Makro benutzt wird: 'E' bedeutet „executed" (ausgeführt) und die Benutzung ist wie in:

```
INP_GPIO(17);
```

'W' bedeutet "written to" (zugewiesen an), die Benutzung ist wie in:

```
GPIO_PULL = 2;
```

'R' bedeutet "read" (lesen von), die Benutzung ist wie in:

```
data = GPIO_IN0;
```

Das Makro `INP_GPIO(n)` muss für eine Pin-Nummer n aufgerufen werden, damit das Pin benutzt wird. Standardmäßig wird der Mode des Pins dabei auf Eingang (input) gesetzt. Wenn das Pin als Ausgang benutzt werden soll, muss `OUT_GPIO(n)` aufgerufen werden, nachdem `INP_GPIO(n)` aufgerufen wurde.

Python Code - Überblick

Die Python-Software (zusammen mit den Teilen dieses Handbuchs, die die Python-Programme beschreiben) wurden von Alex Eames von Raspi.TV geschrieben. Wir sind ihm sehr dankbar für seine Hilfe.

Diese Software ist in Python 2.7 geschrieben und sie ist kompatibel mit allen aktuellen Revisionen des Raspberry Pi-Computers und Gertboards. Es erfordert `sudo-` Benutzerrechte, um den SPI-Bus und die GPIO-Ports zu benutzen.

Der Python-Code enthält verschiedene Software-Pakete, um auf die unterschiedlichen Funktionen des Gertboards zugreifen zu können. Um die GPIO-Ports anzusprechen, muss entweder RPi.GPIO oder WiringPi für Python installiert sein (oder beides). Um den digital- zu analog- und den analog- zu digital-Konverter zu benutzen, der den SPI-Bus benutzt, muss der Modul `py-spidev` installiert sein. Eine Anleitung, wie das erfolgt, ist im Python-Programm enthalten.

Herunterladen der Software

Um die Python Gertboard-Software mit einem Internetbrowser oder mit einem anderen Computer als einem Raspberry Pi herunterzuladen, besuchen Sie
`http://www.gertbot.com/other_products.html` oder
`http://raspi.tv/downloads`. Um sie direkt mit Ihrem mit dem Internet verbundenen Raspberry Pi zu bekommen, wechseln Sie zuerst in das Verzeichnis, in das die Programme installiert werden sollen. Anschließend tippen Sie in die Kommandozeile:

```
wget http://raspi.tv/download/GB_Python.zip
```

Das sollte die kleine Datei `GB_Python.zip` herunterladen. Tippen Sie nun:

```
unzip GB_Python.zip
cd GB_Python
ls
```

Das `ls` listet alle Dateien im Verzeichnis auf. Die meisten von ihnen enden mit .py und sind Python-Programme. Die Datei README.txt enthält (neben anderen Informationen) eine Anleitung, wie das Software-Paket installiert wird, das Sie zum Ausführen der Python-Programme benötigen.

Wenn Sie das benötigte Paket (engl. package) installiert haben, können Sie die Programme ausführen. Wenn Sie z. B. das Programm `leds-rg.py` ausführen möchten, das die LEDs testet (für die Benutzung des RPi.GPIO-Pakets siehe unten) tippen Sie in die Kommandozeile:

```
sudo python leds-rg.py
```

Warum unterschiedliche Programmversionen?

Es gibt zwei General Purpose Input Output (GPIO)-Pakete für Python: RPi.GPIO und WiringPi für Python. Die Programme, die in zwei Versionen kommen (wie `leds-rg.py` und `leds-wp.py`) benutzen jeweils diese unterschiedlichen Pakete.

Es ist sinnvoll, beide Pakete zu installieren, weil keines von beiden den vollen Umfang an Möglichkeiten bietet. Der Schwachpunkt von RPi.GPIO ist das Fehlen des PWM (Pulsweitenmodulator), der mit dem Motorprogramm benutzt wird. Der Schwachpunkt von WiringPi ist das Fehlen der pull-up-Funktion, die für die Benutzung der Schalter benötigt wird. Wenn Sie die gesamte Funktionalität des Bords benutzen wollen, müssen Sie beide Pakete installieren. In manchen Programmen (z. B. `leds` und `ocol`) wird keine spezielle Funktion benutzt und Sie können entweder das eine oder das andere Paket einsetzen. Die Programme, die das RPi.GPIO-Paket benutzen, heißen `filename-rg.py`, während die, die das WiringPi für Python-Paket benutzen, `filename-wp.py` heißen.

Hier ist eine Liste aller Python-Testprogramme (zur Zeit der Erstellung des Handbuchs):

Buttons-rg.py	– Schalter-Programm, benutzt RPi.GPIO
leds-rg.py	– LED-Programm, benutzt RPi.GPIO
leds-wp.py	– LED-Programm, benutzt WiringPi
butled-rg.py	– Schalter- und LED-Programm, benutzt RPi.GPIO
motor-rg.py	– Motor-Programm, benutzt die Software-PWM und RPi.GPIO
motor-wp.py	– Motor-Programm, benutzt Hardware-PWM und WiringPi
ocol-rg.py	– Relais-Schalt-Programm, benutzt RPi.GPIO
ocol-wp.py	– Relais-Schalt-Programm, benutzt WiringPi
atod.py	– Test für analog- zu digital- Konverter, benutzt SPI mit spidev
dtoa.py	– Test für digital- zu analog- Konverter, benutzt SPI mit spidev
dad.py	– Test für D/A and A/D, benutzt SPI mit spidev
potmot.py	– Test für A/D und Motor, benutzt WiringPi und spidev

Gepufferte Ein-Ausgabe (buffered I/O), LEDs und Druckschalter

Es gibt 12 Pins, die als Ein- oder Ausgabeports dienen können. Jedes kann als Eingang oder Ausgang konfiguriert werden, indem ein Jumper gesteckt wird. Beachten Sie, dass die Begriffe Eingang und Ausgang aus der Sicht des Raspberry Pi gesehen sind: beim Eingang liest das Pin Daten in den RPi; beim Ausgang gibt der RPi Daten aus. Es ist wichtig, das im Hinterkopf zu behalten, wenn das Gertboard konfiguriert wird: ein Ausgang vom Gertboard ist ein Eingang für den Raspberry Pi, und somit muss der Eingangs-Jumper gesteckt werden.

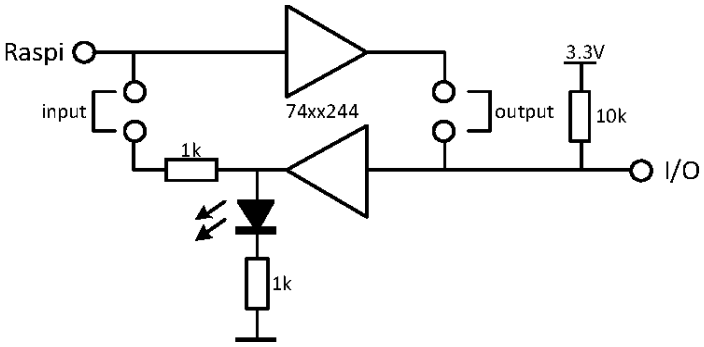


Bild 8: Schaltplan für die I/O-Ports 4-12.

Die beiden weißen Dreieck-Symbole im Diagramm oben repräsentieren Puffer; Sie halten logische Werte (low und high) in der Richtung, in die das Dreieck zeigt. Die Rechtecke sind Widerstände, das schwarze Dreieck mit den beiden Pfeilen ist eine LED, und die kleinen hohlen Kreise sind Pins. Um den Port als Eingang zum Raspberry Pi zu konfigurieren, installieren Sie den Eingangs-Jumper (im Diagramm als ‚input' bezeichnet): dann fließen die Daten vom 'I/O'-Punkt zum 'Raspi'-Punkt. Damit der Port als Ausgang fungiert, muss der Ausgangs-Jumper installiert werden: die Daten fließen dann vom 'Raspi'-Punkt zum 'I/O'-Punkt. Wenn beide Jumper installiert sind, schadet das dem Bord nicht, aber es arbeitet nicht richtig.

Sowohl im Eingangs- als auch im Ausgangsmodus zeigt die LED den logischen Wert am I/O-Pin an. Die LED leuchtet, wenn der Wert 'high' (1) ist und sie ist aus, wenn der Wert 'low' (0) ist. Es gibt eine dritte Möglichkeit, den Port zu nutzen: wenn weder der Eingans- noch der Ausgangs-Jumper steckt, kann das I/O-Pin als einfacher Logikdetektor benutzt werden. Das I/O-Pin kann mit einem anderen logischen Punkt verbunden werden (z. B. einem, der entweder 3,3 oder 0V hat) und die LED zeigt dann, ob der Punkt 1 ist oder 0.

Der Widerstand auf der rechten Seite von Bild 8 ist ein pull-up-Widerstand. Wenn er nicht vorhanden wäre, würde die LED mit der kleinsten Spannungsänderung an und aus gehen, z.B. wenn das Bord berührt wird. Das Einschalten der LED, wenn sie nicht betrieben wird, verhindert dieses ziemlich zufällige Verhalten und dient außerdem der Anzeige, dass das Gertboard richtig mit Strom versorgt ist. Beachten Sie, dass das zufällige Verhalten dann auftritt, wenn der Ausgangs-Jumper steckt, aber der 'Raspi'-Punkt nicht belegt ist.

Es gibt einen Serienwiderstand zwischen dem Eingangspuffer (das nach links zeigende Dreieck) und dem 'Raspi'-Punkt. Dieser ist dazu da, den BCM2835 (der Prozessor auf dem Raspberry Pi) zu schützen für den Fall, dass der Nutzer den GPIO-Port als Ausgang programmiert, aber den Eingangs-Jumper steckt. Der BCM2835-Eingang ist hochohmig und darum produziert ein 1K Serienwiderstand keinen nennenswerten Spannungsabfall, wenn der Port als Eingang arbeitet.

Druckschalter (Pushbuttons)

Das Gertboard hat drei Druckschalter, die mit den Ports 1, 2 und 3 verbunden sind. Der Schaltplan für diese Ports ist in Bild 9 zu sehen. Der Plan ist prinzipiell derselbe wie im Bild 8, mit dem Unterschied, dass links ein Druckschalter und ein Widerstand vorhanden sind. Wenn der Schalter gedrückt wird, wird der 'Raspi'-Punkt mit Masse verbunden und liest somit ‚low' (0).

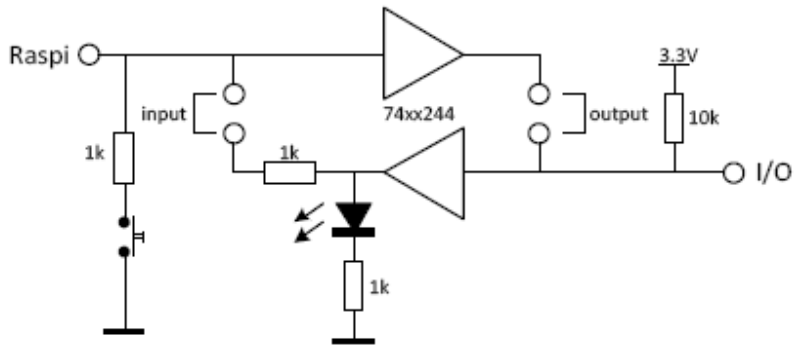


Bild 9: Schaltplan für die I/O-Ports 1, 2 und 3 (mit Druckschalter).

Um einen Druckschalter zu benutzen, darf der Eingangs-Jumper nicht gesteckt sein, auch wenn der Port als Eingang für den Raspberry Pi arbeitet. Wenn der Jumper installiert ist, verhindert der Ausgang des unteren Puffers die ordnungsgemäße Funktion des Druckschalters. Um den Zustand des Schalters mit der LED anzuzeigen, kann der Ausgangs-Jumper gesteckt werden (LED An bedeutet Schalter oben, LED Aus bedeutet Schalter gedrückt). Um die Druckschalter benutzen zu können, muss ein pull-up-Widerstand an die Raspberry Pi GPIO-Pins angeschlossen werden (siehe unten, Seite 19), so dass logisch 1 (high) gelesen wird, wenn der Schalter nicht gedrückt ist.

Lage der I/O-Ports auf dem Gertboard

Im Funktionsblock-Foto (Bild 2 auf Seite 6) sind die Komponenten, die die gepufferte Ein-/Ausgabe implementieren, rot umrandet. Die ICs, die die Puffer enthalten, sind U3, U4 und U5 nahe der Mitte des Bords. Die LEDs sind beschriftet mit D1 bis D12; D1 wird durch Port 1 getrieben, D2 durch Port 2 usw. Die Druckschalter (die silbernen Rechtecke mit schwarzen Kreisen) sind bezeichnet mit S1 bis S3; S1 ist mit Port 1 verbunden usw.

Die Pins, die mit ‘Raspi’ in den Bildern 8 und 9 bezeichnet sind, sind B1 bis B12 auf der Pfostenreihe J3 oben und rechts von den Worten ‘Raspberry Pi’ auf dem Bord (B1 bis B3 sind die ‘Raspi’-Punkte im Bild 9, und B4 bis B12 sind die ‘Raspi’-Punkte im Bild 8). Sie werden ‘Raspi’ genannt, weil sie die einzigen sind, die normalerweise mit den Pins der Pfostenreihe J2 verbunden sind, die direkt verbunden sind mit den Pins von J1, und die dann letztendlich verbunden sind mit den GPIO-Pins auf dem Raspberry Pi. Die Pins, die mit ‘I/O’ bezeichnet sind (rechts im Schaltplan, Bild 8 und 9), sind BUF1 bis BUF12 auf der einzelnen, unbeschrifteten Pfostenleiste am oberen Rand vom Gertboard.
In der Gertboard-Schaltung sind die I/O-Puffer auf Seite A-2 zu sehen. Die Puffer-Chips U3, U4 und U5 sind gut beschriftet. Es sollte klar sein, dass die Ports 1 bis 4 zum Chip U3, die Ports 5 bis 8 zum Chip U4 und die Ports 9 bis 12 zum Chip U5 gehören. Die ‘Raspi’-Punkte in den Schaltplänen oben werden als die Signale BUF_1 bis BUF_12 links auf der Seite dargestellt, und die ‘I/O’-Punkte sind BUF1 bis BUF12 rechts der Puffer-Chips. Die Eingangs-Jumper sind die blauen Rechtecke, die mit P1, P3, P5, P7 usw. bezeichnet sind, rechts von den Puffer-Chips, und die Ausgangs-Jumper sind die blauen Rechtecke, die mit P2, P4, P6, P8 usw. bezeichnet sind, rechts von den Puffer-Chips. Die Druckschalter S1, S2 und S3 werden separat gezeigt, auf der rechten Seite, in der Nähe der Mitte. Unter den Druckschaltern sind die pull-up-Widerstände zu sehen.

Die gepufferten I/O-Ports können mit (fast) jedem GPIO-Pin benutzt werden – sie brauchen nur mit Verbindungsdrähten verbunden zu werden. Wenn Sie z.B. Port 1 mit GPIO17 benutzen möchten, wird eine Verbindung zwischen dem B1-Pin an J3 und dem GP17-Pin an J2 hergestellt. Beachten Sie, dass die Druckschalter nicht mit GPIO0 oder GPIO1 (GP0 und GP1 in der Pfostenreihe J2 auf dem Bord) benutzt werden können, da diese beiden Pins einen 1800Ω pull-up-Widerstand auf dem Raspberry Pi haben. Wenn der Schalter gedrückt wird, ist die Spannung am Eingang

$$3.3V \times \frac{1000\Omega}{1000\Omega + 1800\Omega} = 1.2V$$

Das ist keine I/O-Spannung, die verlässlich als low (0) angesehen werden kann.

Die Ausgangs- und Eingangs-Jumper sind ober- und unterhalb der U3, U4 und U5 Puffer-Chips angeordnet. Die Eingangs-Jumper mussten auf der Pfostenreihe unterhalb der Chips angeordnet werden (auf dem Bord mit dem Text ‘in’ beschriftet; sie sind vom zugehörigen Chip durch vier kleine Widerstände getrennt), und die Ausgangs-Jumper mussten auf der Pfostenreihe oberhalb der Chips angeordnet werden (mit dem Text ‘out’). Bei näherer Betrachtung (es ist ersichtlicher auf dem blauen und grauen Schema) sieht man, dass jede Reihe von 8 Pfosten-Pins ober- und unterhalb der Puffer-Chips in 4 Pin-Paare aufgeteilt ist. Die Paare an U3 sind mit B1 bis B4 bezeichnet, die an U4 sind B5 bis B8, und die an U5 sind B9 bis B12. Das B1-Pin ist für Port 1, B2 für Port 2 usw.

Um Port n als Eingang zu nutzen (aber nicht für die Benutzung der Druckschalter, wenn n 1, 2 oder 3 ist), wird ein Jumper auf das Pin-Paar in B*n* in der Reihe, die mit ‘in’ bezeichnet ist, gesteckt (unterhalb des entsprechenden Puffer-Chips). Um Port n als Ausgang zu betreiben, wird ein Jumper auf das Pin-Paar in B*n* in der Reihe, die mit ‘out’ bezeichnet ist, gesteckt (oberhalb des entsprechenden Puffer-Chips).

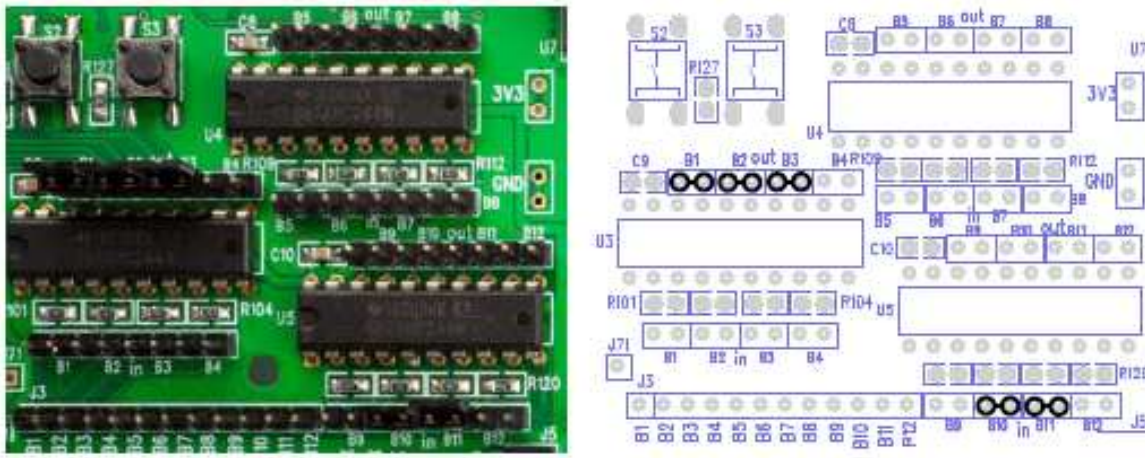


Bild 10: Beispiel der Port-Konfiguration, wobei die Ports 1 bis 3 als Ausgang und die Ports 10 und 11 als Eingang gesetzt sind

Ein Beispiel sehen Sie in Bild 10: Die Ports 1, 2 und 3 sind als Ausgang konfiguriert (die Jumper sind auf B1, B2 und B3 auf der ‘out’-Seite des Chips U3 gesteckt). Die Ports 10 und 11 sind als Eingang konfiguriert (die Jumpers sind auf B10 und B11 auf der ‘in’-Seite von U5 gesteckt). Bild 10 zeigt auch, warum wir lieber Schemas anstatt Fotos benutzen, um die Verbindungen auf dem Bord zu veranschaulichen. Der Eingangs-Jumper auf B10 ist auf dem Foto fast nicht zu sehen, weil er kürzer ist als die anderen.

In den Testprogrammen werden die benötigten Verbindungen angezeigt, bevor der Test gestartet wird. Die Eingangs- und Ausgangs-Jumper werden auf folgende Weise bezeichnet: U3-out-B1 bedeutet einen Jumper auf den B1-Pins auf der ‘out’-Seite vom Puffer-Chip U3. Somit werden die 5 Jumper im oberen Bild bezeichnet mit U3-out-B1, U3-out-B2, U3-out-B3, U5-in-B10 und U5-in-B11.

Testen der Druckschalter (pushbuttons)

Es gibt Testprogramme für die Schalter in C und Python. Die C-Version heißt `buttons` und die Python-Version heißt `buttons-rg.py`. Um die Testprogramme laufen zu lassen, muss das Gertboard so konfiguriert werden wie es in Bild 11 zu sehen ist (das Schema in Bild 12 zeigt dasselbe übersichtlicher). Es gibt Verbindungsdrähte, die die Pins B1, B2 und B3 des Pfostensteckers J3 mit den Pins GP25, GP24 und GP23 des Steckers J2 verbinden. Folglich liest GPIO25 den links liegenden Druckschalter, GPIO24 den mittleren und GPIO23 den rechten Druckschalter. Die Jumper auf der ‘out’-Seite von U3 (U3-out-B1, U3-outB2, U3-out-B3) sind optional: Wenn sie gesteckt sind, dann zeigen die linken 3 LEDs den Zustand der Schalter an.

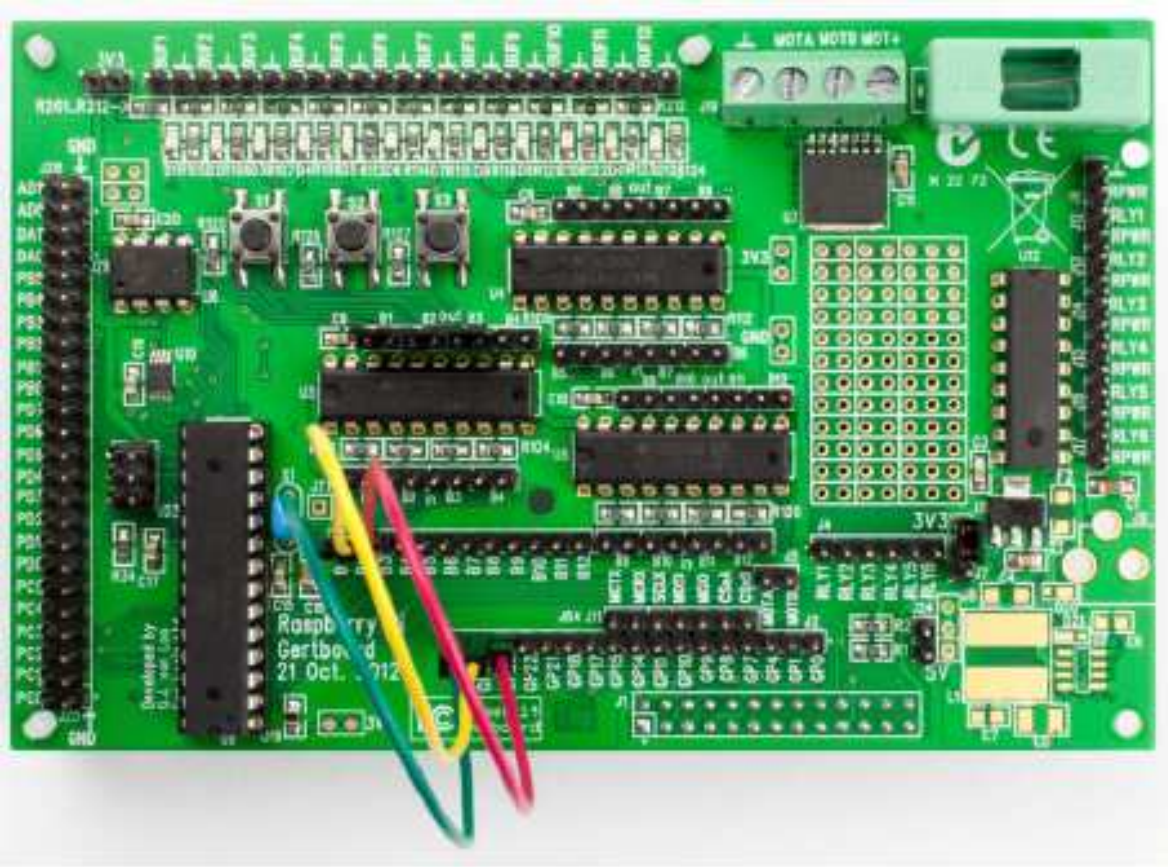


Bild 11: Das Foto zeigt die Verbindungen für das Schalter-Testprogramm. Für den Nachbau eignet sich die schematische Darstellung in Bild 12 besser.

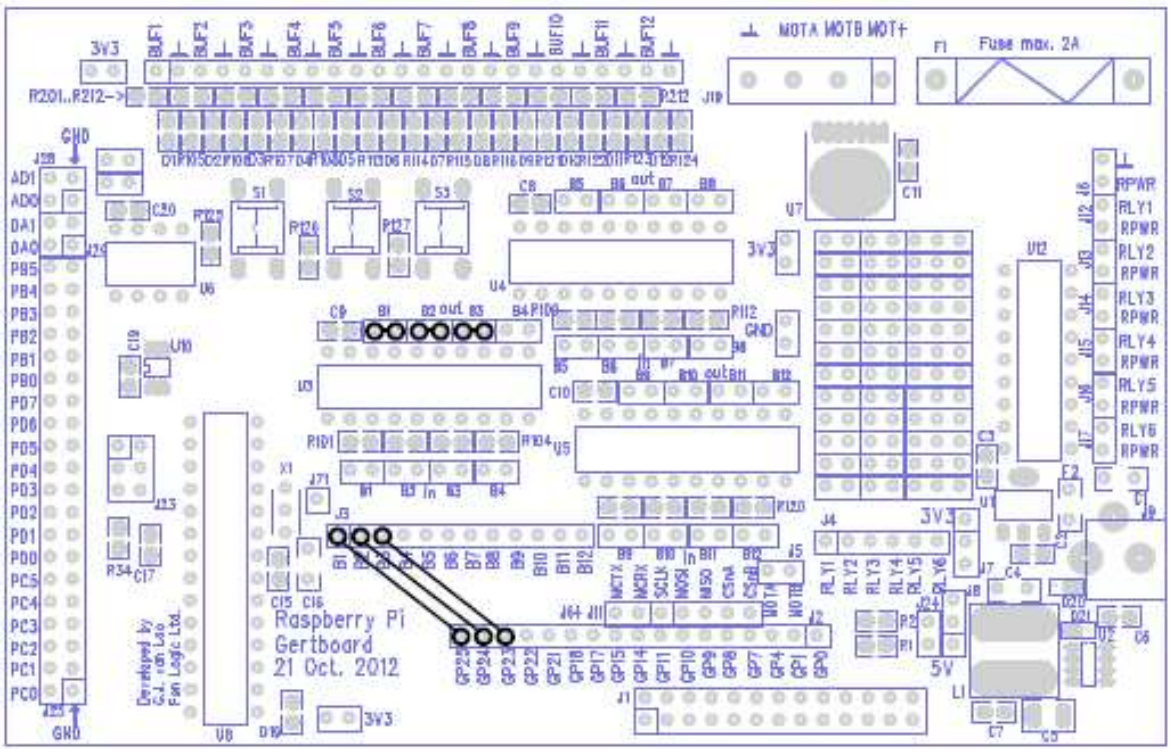


Bild 12: Schematische Darstellung der Verbindungen für das Schalter-Testprogramm. Diese Darstellung ist für den Nachbau besser geeignet als das Foto in Bild 11.

Im Bild 12 zeigen schwarze Kreise und Linien, welche Pins miteinander verbunden werden müssen. Nebeneinander liegende Pins werden am besten mit einem Jumper verbunden.

Schaltertest in C

Der spezielle Quellcode für den Schaltertest befindet sich in `buttons.c`. Im Hauptprogramm (`main`-Routine) werden zuerst die benötigten Verbindungen für diesen Test auf dem Bildschirm angezeigt (wie weiter oben bereits beschrieben). Nachdem Sie die Verbindungen hergestellt bzw. geprüft haben, wird `setup_io` (siehe Seite 12) aufgerufen, um alles bereit zu setzen.

`setup_gpio` wird anschließend aufgerufen, um die GPIO-Pins 1 bis 3 für die Benutzung als Druckschalter-Eingänge zu aktivieren. Dazu wird zuerst das Makro `INP_GPIO(n)` (wobei n die GPIO-Pinnummer ist) aufgerufen, um diese 3 Pins als Eingang zu konfigurieren.

Die Pins müssen auf high (1) gezogen werden (pull-up): Die Schalter setzen beim Drücken die Spannung auf 0V, deshalb muss im nicht gedrückten Zustand die Spannung hoch sein. Dies wird durch Setzen von GPIO_PULL auf 2, der Wert für pull-up, erreicht. Sollte es jemals erforderlich sein: der Wert für pull-down (ziehen auf 0V) ist 1. Der Wert für nicht ziehen (no pull) ist 0; Das erlaubt dem Pin, als Ausgang zu fungieren, nachdem es als Schalter-Eingang benutzt wurde. Um diese Werte den entsprechenden Pins zuzuweisen, setzen Sie `GPIO_PULLCCLK0 = 0X03800000`. Diese 32-Bit hexadezimale Zahl (0X am Anfang bedeutet hexadezimal – binär lautet sie 0000 0011 1000 0000 0000 0000 0000 0000) hat die Bits 23, 24 und 25 (ganz rechts ist Bit 0) auf 1 gesetzt, der Rest ist 0. Das bedeutet, der „pull“-Wert wird auf die GPIO-Pins 23, 24 und 25 angewendet. Ein `short_wait` gibt dem Befehl Zeit, umgesetzt zu werden und anschließend werden GPIO_PULL und GPIO_PULLCLK0 zurück auf 0 gesetzt.

Zurück im Hauptprogramm (`main`-Routine), wird eine Schleife ausgeführt, in der die Schalter zyklisch abgefragt werden (unter Benutzung des Makros `GPIO_IN0`). Die Bits 23, 24 und 25 werden ausgewertet (über logische Verknüpfung mit einer Maske und unter Benutzung von Schiebeoperatoren) und bei einer Veränderung eines Schalterzustandes wird dies auf dem Bildschirm angezeigt.(‚1’ für oben – high, und ‚0’ für unten – low). Diese Schleife wird solange durchlaufen, bis genügend Statuswechsel erkannt worden sind.

Nach der Schleife wird `unpull_pins` aufgerufen, das den pull-up an den Pins zurücksetzt, danach die Funktion `restore_io` in `gb_common.c`, um „aufzuräumen“.

Schaltertest in Python

Dieses Programm (`buttons-rg.py`) ist z. Z. nur unter Benutzung des RPi.GPIO-Paketes verfügbar, weil die pull-up-Funktion in WiringPi für Python noch nicht existiert. Die Schalter können ohne diese Funktion nicht benutzt werden.

Zuerst importiert das Programm den RPi.GPIO-Modul, der für die GPIO-Steuerung benötigt wird. Danach setzt das Kommando `GPIO.setmode(GPIO.BCM)` das BCM-Nummerierungs-Schema für die Pins. Das Ergebnis ist, dass die Pin-Nummern im Python-Code die Nummern sind, die der BCM2835 (der Raspberry Pi-Prozessor) benutzt, um die Pins zu referenzieren. Andererseits beziehen sich die Pin-Nummern im Python-Code auf die Pin-Nummern des Pfostensteckers J1 (was dieselben sind wie die des P1-Steckers auf dem RPi).

Die nächsten zwei Zeilen konfigurieren die Ports 23-25 mit pull-up-Widerständen:

```
for i in range(23, 26):
    GPIO.setup(i, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Beachten Sie bitte, dass in Python bei “for”-Schleifen der “bis”-Wert um 1 höher sein muss als die Schleife laufen soll. Wenn im obigen Beispiel die Variable i von 23 bis 25 laufen soll, muss die obere Grenze also 26 betragen.

Als nächstes werden die Verdrahtungsangaben auf dem Bildschirm angezeigt. Wenn der Benutzer diese Daten bestätigt hat, werden die Anfangswerte für die Variablen `button_press` und `previous_status` gesetzt.
Nun läuft eine Schleife (while), bis 19 Mal ein Schalter ein- bzw. ausgeschaltet wurde. Bei jedem Schlei-

fendurchlauf wird der Status jedes Schalters gelesen und, wenn er gedrückt ist, eine 1, und wenn er nicht gedrückt ist, eine 0 in der Array-Variablen `status_list` für diesen Schalter gespeichert.

Anschließend werden die Werte in der Variablen `status_list` mit den Werten in der Variablen `previous_status` (hier sind die Werte des vorherigen Schleifendurchlaufs gespeichert) verglichen: Die Zeile

```
if current_status != previous_status:
```

führt bei einer Veränderung des aktuellen Status gegenüber dem letzten Status einen kleinen Abschnitt im Programm aus, der den neuen Wert auf dem Bildschirm anzeigt und den Zähler `button_press` hochzählt (diese Variable enthält die Zahl der Schaltvorgänge). Die Schleife startet von vorn, bis 19 Schaltvorgänge erreicht sind, dann bricht sie ab.

Die `while`-Schleife ist in einen `try`-Block eingeschlossen:

```
try:
      <while block>
except KeyboardInterrupt:
```

das erlaubt dem Programm, auf die Tastenkombination Ctrl-C zu reagieren und die Ports zurückzusetzen sowie das Programm vorzeitig zu beenden.

Wenn das Programm normal nach 19 Schaltvorgängen beendet wird, werden die Ports zurückgesetzt. Wenn wir nicht `try: … except` benutzt hätten, würde der Tastatur-Interrupt (Ctrl-C) das Programm zwar beenden, aber die Ports geöffnet lassen. Das führt zu Fehlern, wenn die Ports erneut geöffnet werden sollen.

Vorschläge zum Experimentieren. Versuchen Sie die folgenden Änderungen und sehen Sie, was passiert...

- `while button_press < 20`: – ändern Sie die 20 in eine andere Zahl
  (Anm. d. Ü.: Erläuterung der Programmzeile: solange die Variable `button_press` kleiner ist als 20, wird die `while`-Schleife durchlaufen. `button_press` wird bei jedem Schaltvorgang um 1 erhöht)
- `button_press += 1` – ändern Sie die 1 in eine andere Zahl
  (Anm. d. Ü.: Erläuterung der Programmzeile: die Variable `button_press` wird um 1 erhöht. Es handelt sich um die Kurzschreibweise von `button_press = button_press + 1`)

Testen der LEDs

Das C-Testprogramm für die LEDs heißt `leds`. Die Python-Versionen sind `leds-rg.py` und `leds-wp.py`. Um das Gertboard für diesen Test vorzubereiten, schauen Sie sich den Verdrahtungsplan im Bild 13 an. Jeder I/O-Port ist als Ausgang geschaltet, somit sind alle ‘out’-Jumper (die oberhalb der Puffer-Chips) installiert. Drähte werden für die folgenden Verbindungen benutzt: (wobei alle ‘GP’-Pins auf Stecker J2 und alle ‘B’-Pins auf Stecker J3 liegen): GP25 zu B1, GP24 zu B2, GP23 zu B3, GP22 zu B4, GP21 zu B5, GP18 zu B6, GP17 zu B7, GP11 zu B8, GP10 zu B9, GP9 zu B10, GP8 zu B11 und GP7 zu B12. Mit anderen Worten, die 12 linken ‘GP’-Pins werden verbunden mit den ‘B’-Pins, außer dass GP14 und GP15 vermisst werden: sie sind bereits auf den UART-Modus durch Linux gesetzt, so dass sie am besten nicht beachtet werden.

Wenn es nicht genug Jumper oder Verbindungsdrähte gibt, um alle benötigten Verbindungen für den Test herzustellen, machen Sie sich keine Sorgen, verbinden Sie so viel wie möglich und starten Sie den Test. Wenn er beendet ist, können die Jumper / Verbindungsdrähte umgesteckt und der Test erneut gestartet werden. Es passiert nichts Schlimmes, wenn auf ein unverbundenes Pin geschrieben wird.

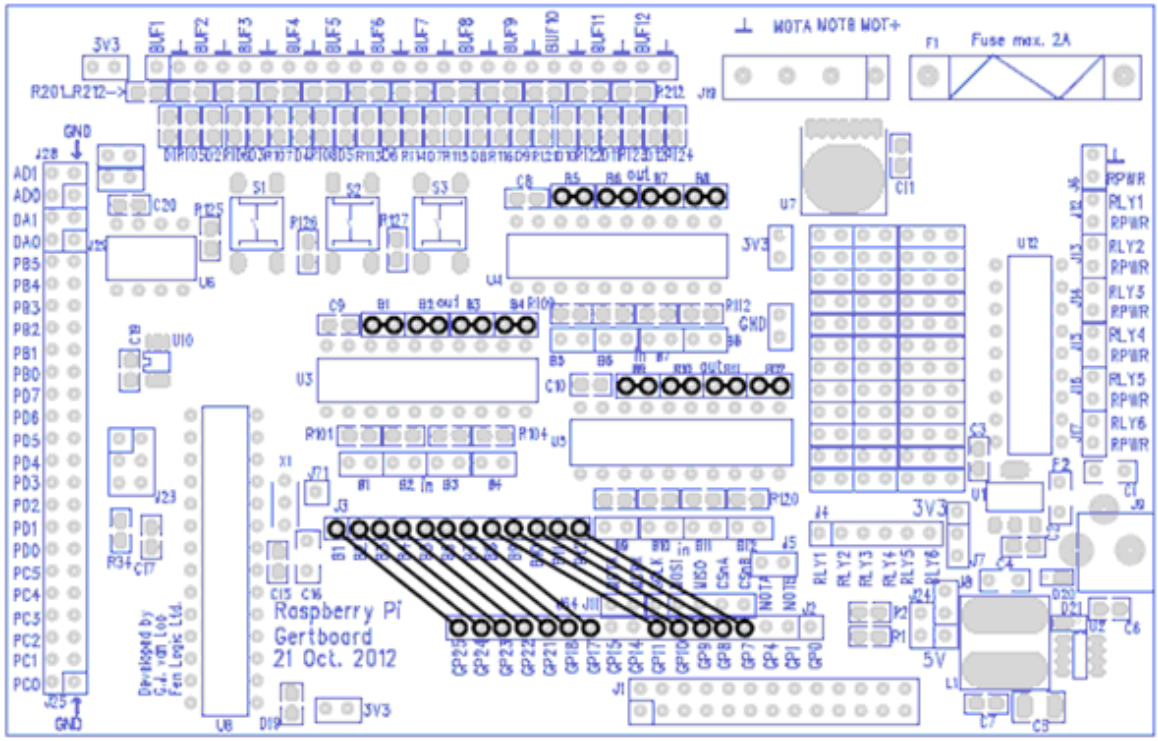


Bild 13: Verbindungsschema für das LED-Testprogramm

Test der LEDs in C

Der Code in `leds.c` ruft zuerst `setup_io`, um alles zu initialisieren. Dann wird `setup_gpio` aufgerufen, das die 12 GPIO-Pins vorbereitet, um als Ausgänge zu arbeiten (alle 12 I/O-Ports erfordern Steuerung). Alle GPIO-Signale außer GPIO 0, 1, 4, 14 und 15 werden benutzt. Um sie als Ausgänge zu konfigurieren, muss zur Initialisierung zuerst `INP_GPIO(n)` aufgerufen werden (wobei n die GPIO-Pinnummer ist), und zwar für jedes der 12 Pins. Da dieser Befehl die Ports außerdem als Eingang öffnet, muss anschließend `OUT_GPIO(n)` für jedes der 12 Pins aufgerufen werden, um sie als Ausgang zu betreiben.

Die LEDs werden durch das Makro `GPIO_SET0` umgeschaltet: Der Wert, der an `GPIO_SET0` zugewiesen wird, setzt das GPIO-Pin n auf high (1), wenn das Bit n auf diesen Wert gesetzt wird. Wenn ein GPIO-Pin auf high (1) gesetzt wird, wird der damit verbundene I/O-Port ebenfalls auf high (1) gesetzt und die LED an diesem Port geht an. Somit setzt die folgende Programmzeile

```
GPIO_SET0 = 0x180;
```

die GPIO-Pins 7 und 8 auf high (da die Bits 7 und 8 auf 1 gesetzt werden: hexadezimal 0X180 = 0001 1000 0000 in binärer Schreibweise). Entsprechend dem Verdrahtungsplan oben gehen die Ports 11 und 12 auf high (1) (das sind die mit GP7 und GP8 verbundenen Ports), und somit leuchten die zwei rechten LEDs.

Um die LEDs auszuschalten, wird das Makro `GPIO_CLR0` benutzt. Das arbeitet in analoger Weise wie `GPIO_SET0`, aber hier zeigen die auf 1 gesetzten Bits im zugewiesenen Wert an, welche GPIO-Pins auf low (0) gesetzt werden (und damit, welche Ports auf 0 gesetzt werden und damit, welche LEDs ausgeschaltet werden). Zum Beispiel setzt die Programmzeile

```
GPIO_CLR0 = 0x100;
```
(hexadezimal 0x100 = 0001 0000 0000 binär → Bit 8 ist gesetzt)

das GPIO-Pin 8 auf low und schaltet somit die LED für Port 11 aus, weil dieser Port mit GP8 verbunden ist. (In `leds.c` werden immer alle LEDs zusammen ein- oder ausgeschaltet, aber das kann man natürlich anders machen)

Das Testprogramm lässt die LEDs in drei Mustern leuchten. Die Muster werden durch eine Anzahl von globalen Arrays (Array = Feld von Werten gleichen Typs, z.B. von Zahlen) mit gegebenen Werten festgelegt. Die Zahl in jedem Array bestimmt, welche der LEDs an dieser Stelle im Muster eingeschaltet wird – daher werden die Werte des Arrays nacheinander ausgegeben, um das Muster an leuchtenden LEDs zu erzeugen. Jedes Muster wird zweimal durchlaufen. Das erste Muster lässt die LEDs einzeln von links nach rechts leuchten. Das zweite Mustern tut dasselbe, aber beim Erreichen der rechten LED wechselt es die Richtung und die LEDs leuchten von rechts nach links auf. Das dritte Muster startet bei der linken LED und lässt die LEDs jeweils eingeschaltet, wenn die nächste LED aufleuchtet, bis alle LEDs leuchten. Anschließend werden alle LEDs, von links beginnend, wieder ausgeschaltet.

Zum Schluss werden alle LEDs ausgeschaltet und das Programm ruft `restore_io` auf, um die Hardware auf einen definierten Status zu setzen.

Test der LEDs in Python

Dieser Test ist für RPi.GPIO und WiringPi für Python verfügbar. Der einzige Unterschied zwischen `leds-rg.py` und `leds-wp.py` ist die Art, wie die GPIO-Ports behandelt werden und der Raspberry Pi-Bord- Revisionsprüfer.

Im Programm wird zuerst die Raspberry Pi-Bord- Revision geprüft und, abhängig vom Resultat, werden die korrekten Ports für die LEDs in einer Liste namens `ports` definiert. Da wir die Liste vorwärts und rückwärts durchlaufen müssen, machen wir eine Kopie und invertieren diese – sie heißt `ports_rev`. Nun setzen wir die Ports als Ausgang, indem wir die Liste `ports` durchlaufen.

Danach definieren wir die Hauptroutine des Scripts, die Funktion `led_drive()`, die drei Argumente erfordert `(reps, multiple, direction)`.

1. `reps` (von repeats – Wiederholungen) definiert, wie oft der Prozess durchlaufen wird (1 oder 3 in dieser Demo)
2. `multiple` (mehrfach) definiert, ob eine LED ausgeschaltet wird, bevor die nächste LED eingeschaltet wird (1 = anlassen, d.h. es leuchten mehrere LEDs)
3. `direction` (Richtung) definiert die Richtung (vorwärts oder rückwärts – es wird die entsprechende Liste zum Durchlaufen der LEDs verwendet – `ports` für vorwärts, `ports_rev` für rückwärts)

Es gibt acht Aufrufe der Funktion `led_drive()` und somit haben wir eine Menge Programmzeilen gespart, weil wir mit der Funktion immer wieder denselben Code mehrfach aufrufen. Alle Aufrufe von `led_drive()` sind in einen `try:... except:` -Block eingeschlossen. Das sichert, dass beim Verlassen des Programms über Ctrl-C die GPIO-Ports zurückgesetzt werden. Das verhindert Warnungen, wenn das nächste Mal ein Programm die Ports öffnen will, diese aber immer noch in Benutzung durch ein anderes Programm sind.

Vorschläge zum Experimentieren: Ändern Sie die Werte und sehen Sie, was passiert...

- `led_drive(3, 0, ports)` – ändern Sie die 3
- `led_drive(3, 0, ports)` – wechseln Sie von `ports` zu `ports_rev`
- `led_drive(3, 0, ports)` – ändern Sie die 0 in eine 1

Test der Ein-/Ausgabe (I/O)

Unsere zwei bisherigen Beispiele haben nur die Ports für den Zugriff auf die Druckschalter und LEDs benutzt. Das nächste Beispiel namens `butled` (für BUTton LED) in C bzw. `butled-rg.py` (in Python), zeigt uns, wie ein Port nur als Eingang dient. Die Idee ist, dass ein Port (zusammen mit seinem Schalter) benutzt wird, um ein Signal zu generieren und die Software anschließend dieses Signal an einen anderen Port sendet, der nur als Eingang (zum Raspberry Pi) dient. Wir lesen beide Ports ein und zeigen Sie auf dem Bildschirm an.

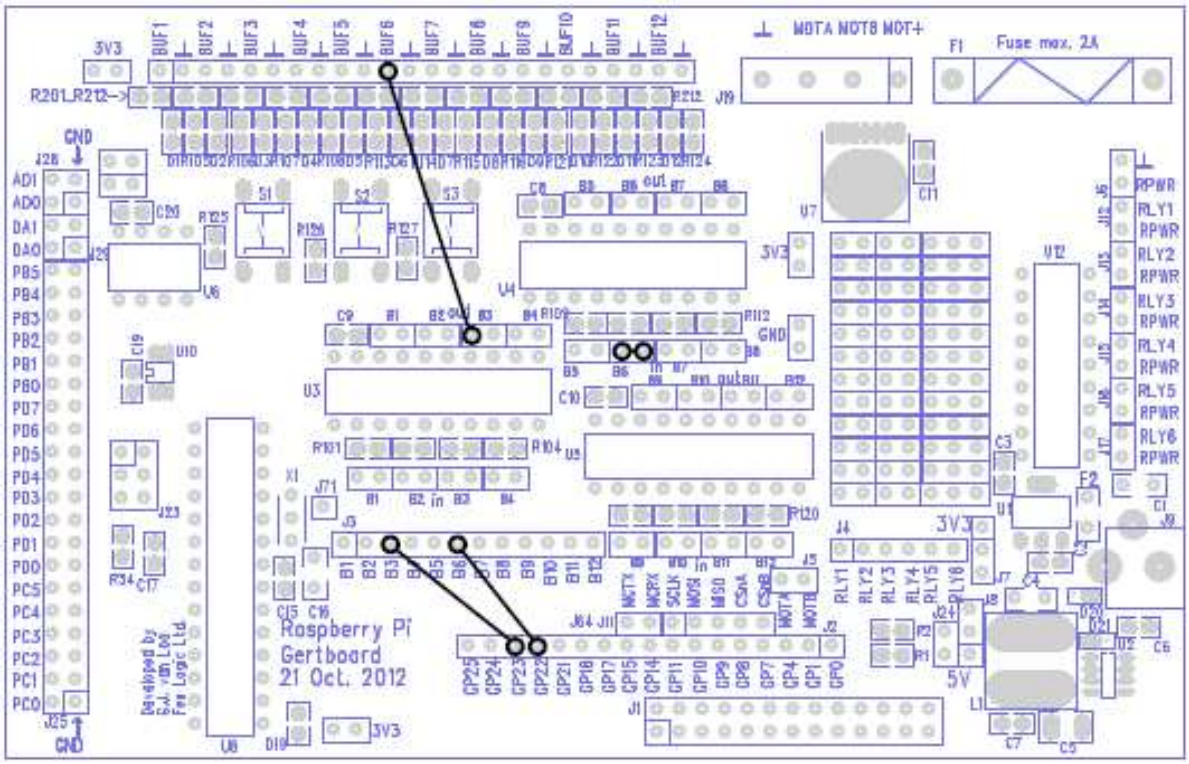


Bild 14: Verdrahtungsplan für das Testprogramm butled, das einen Schalterdruck erkennt und anschließend diesen Schalterzustand auf dem Bildschirm und einer LED anzeigt

Im Bild 14 ist der Verdrahtungsplan zu sehen: Pin GPIO23 steuert I/O-Port 3 und GPIO22 steuert I/O-Port 6, somit wird GP23 auf dem Stecker J2 mit Pin B3 auf Stecker J3 verbunden und GP22 wird mit B6 verbunden. Beachten Sie: Der Druckschalter an Port 3 soll hier benutzt werden, aber die LED für Port 3 soll nicht benutzt werden. Somit wird der Ausgangs-Jumper für Port 3 (der an U3-out-B3 gesteckt würde) nicht installiert.

Wenn man den Schaltplan auf Seite A-2 betrachtet, wird klar, dass der Ausgangspuffer für Port 3 an Pin 14 des Pufferchips U3 geht. Dieser ist mit Pin 1 von U3-out-B3 verbunden (als P6 im Schema zu sehen). Es ist nicht offensichtlich, welches Steckerpin auf dem Bord Pin 1 ist, aber da es mit Pin 14 des Chips verbunden ist, können wir schlussfolgern, dass es das Pin genau oberhalb von Pin 14 ist. Ein einfacher Test zeigt, dass es tatsächlich das richtige Pin ist, und wir verbinden dieses Pin mit dem BUF6-Pin oben auf dem Bord. Das erlaubt dem Schalter, ein Signal zu generieren, das dann an Port 6 gesendet wird. Ein Jumper wird auf U4-in-B6 gesteckt, um das Signal einlesen zu können. Der Wert des Schalters von Port 3 wird ebenfalls eingelesen, und diese zwei Signale sollten identisch sein (jedenfalls meistens).

Test von butled in C

In `butled.c` benutzen wir `INP_GPIO`, um GPIO22 und GPIO23 auf Eingang zu setzen und `GPIO_PULL` und `GPIO_PULLCLK0`, um pull-up-Widerstände an GPIO23 zu setzen. Das wird auf Seite 19 im Abschnitt Schaltertest näher beschrieben. Dann werden die GPIO-Werte wiederholt eingelesen, und die Binärwerte von GPIO22 und GPIO23 werden angezeigt (GPIO23 zuerst), wenn sie sich seit der letzten Abfrage geändert haben. Wenn also '01' auf dem Bildschirm angezeigt wird, können wir sehen, dass GPIO23 low ist und GPIO22 high. (Beachten Sie, dass die LED für Port 6, mit D6 bezeichnet, Aus sein sollte, wenn der Schalter3 gedrückt ist und Ein, wenn er nicht gedrückt ist.)

Nun, wenn die Werte für GPIO22 und GPIO23 immer dieselben sind, sollte immer nur '00' oder '11' angezeigt werden. Aber gelegentlich sehen wir '01' oder '10'. Wenn das passiert, wechselt die erste Stelle zum neuen Wert und unmittelbar danach die zweite. Aber die Werte von GPIO22 und GPIO23 werden gleichzeitig gelesen, wieso haben wir dann verschiedene Werte an den GPIO-Pins? Die Antwort liegt in der kleinen Zeitspanne, die das Signal vom Druckschalter (der mit GPIO23 verbunden ist) durch die Puffer benötigt, um an GPIO22 zu gelangen. Manchmal lesen wir, nachdem sich GPIO23 geändert hat, aber nicht genug Zeit vergangen ist, um die Änderung an GPIO22 zu sehen!

Test von butled in Python

Das Programm (`butled-rg.py`) arbeitet z. Z. nur mit RPi.GPIO. Es ist ein sehr ähnliches Programm wie `buttons`, aber mit zwei Schaltern weniger und einer LED, die leuchtet, wenn der Schalter gedrückt ist.

Zwei GPIO-Ports werden benutzt. Ein Port (23) arbeitet mit einem pull-up-Widerstand für den Schalter und der andere (22) wird als normaler Eingang betrieben. In der Haupt- (while-) Schleife fragt das Programm beide Ports ab und inkrementiert bei einer Änderung des Zustands die Variable `button_press` und zeigt die neuen Werte auf dem Bildschirm an. Der Draht von U3-out-B3 Pin 1 zu BUF6 im oberen Stecker verbindet die LED mit dem Schalter, damit sie leuchtet und der andere Eingang (Port 22) auf high geht, wenn der Schalter gedrückt wird. Die Werte werden auf dem Bildschirm angezeigt wie beim Programm `buttons`.

Normalerweise müssten wir immer gleiche Werte für GPIO22 und GPIO23 sehen, nämlich '00' oder '11'. Aber, wie im C-Beispiel, sehen wir '01' oder '10'. Wenn das passiert, wechselt die zweite Stelle zuerst auf den neuen Wert und kurz darauf die erste Stelle. Das ist umgekehrt wie im C-Beispiel. Warum? Im Python-Programm werden die Werte in folgender Zeile eingelesen:

```
status_list = [str(GPIO.input(23)),str(GPIO.input(22))]
```

Das verursacht das Lesen von GPIO23 vor GPIO22. Wenn der Schalter gedrückt oder geöffnet wird, behält zwischen den beiden Lesevorgängen GPIO23 noch den alten Wert, aber der neue Wert wird von GPIO22 gelesen. Der neue Wert wird nicht von GPIO23 gelesen, bis die while-Schleife das nächste Mal durchlaufen wird.

Open-Kollektor-Treiber

Das Gertboard benutzt sechs Ports eines ULN2803A, um Open-Kollektor-Treiber bereitzustellen. Diese werden benutzt, um Geräte ein- und auszuschalten, besonders solche, die von einer externen Stromversorgung gespeist werden und eine andere Spannung benötigen oder einen höheren Strombedarf haben, als auf dem Gertboard verfügbar ist. Einfach betrachtet, verbindet ein Open-Kollektor-Treiber die Masse-Seite einer externen Schaltung mit der Masse auf dem Bord und gibt somit der Schaltung Strom. Der ULN2803A kann bis zu 50V aushalten und 500mA an jedem seiner Ports treiben. Jeder Treiber hat eine integrierte Schutzdiode (die oberste Diode im Schaltplan im Bild 15).

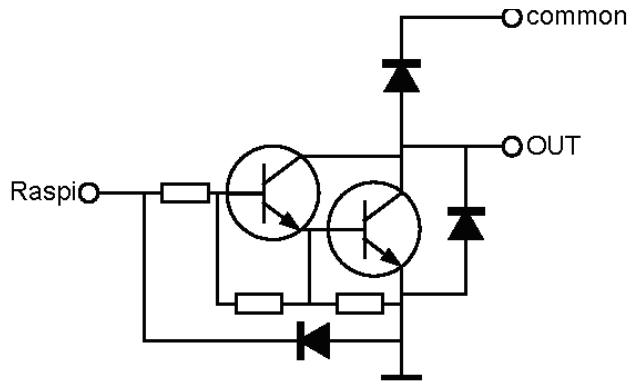


Bild 15: Schaltplan eines Open-Kollektor-Treibers

Das ‘common’-Pin ist, wie der Name sagt, gemeinsam für alle Open-Kollektor-Treiber. Es ist nicht mit etwas anderem auf dem Gertboard verbunden. Wie bei allen Geräten kann die Steuerung für die Open-Kollektor-Treiber (der ‘Raspi’-Punkt) auch mit dem ATmega-Controller verbunden werden, z. B. um Relais oder Motoren anzusteuern.

Die Open-Kollektor-Treiber sind im Schaltplan auf Seite A-3 zu sehen.

Im Blockdiagramm des Gertboards (Bild 2 auf Seite 6) sind die Bereiche, in denen sich die Komponenten für die Open-Kollektor-Treiber befinden, gelb markiert. Die Pins, die mit ‘Raspi’ im Bild 15 bezeichnet sind, sind die Pins RLY1 bis RLY6 auf der Steckerleiste J4; Die Pins, die mit ‘common’ bezeichnet sind, sind die mit RPWR bezeichneten auf der Steckerleiste am rechten Rand des Bords; Und die Pins, die mit ‘OUT’ bezeichnet sind, sind die Pins RLY1 bis RLY6 auf der Steckerleiste J12 bis J17. Wie das Ganze benutzt wird, wird durch ein Testbeispiel veranschaulicht.

Test des Open-Kollektor-Treibers

Das C-Programm `ocol` (für open collector) ermöglicht den Funktionstest der Open-Kollektor-Treiber. Die Python-Version kommt in zwei Varianten, `ocol-rg.py` und `ocol-wp.py`.

Wie benötigten etwas, das der Treiber ein- und ausschalten kann, und so bauten wir eine kleine Schaltung aus zwei LEDs und einem Serienwiderstand (Das ist die kleine Schaltung rechts auf dem Gertboard im Bild 16). Wenn es verbunden ist, beträgt die Spannung über jeder LED ein wenig über 3V und wir benutzten eine 9V-Batterie als Stromversorgung und berechneten einen Serienwiderstand von rund 90Ω, damit ein passender Strom durch die LEDs fließt. Sie können natürlich irgendeine Schaltung für diesen Test benutzen; achten Sie nur darauf, dass die Spannung der Stromversorgung für die Schaltung passt (das ist innerhalb von 50V, 500mA ist die Grenze für die Treiber).

Um die Geräte mittels Open-Kollektor-Treiber ein- oder auszuschalten (sagen wir, unter Verwendung von Treiber 1), muss zuerst geprüft werden, ob das Gerät (die Schaltung) mit der externen Stromversorgung funktioniert. Dann bleibt die Plusseite der Schaltung an Plus der externen Stromversorgung angeschlossen, wird aber zusätzlich mit einem der RPWR-Pins auf dem Stecker an der rechten Seite des Bords verbunden (die sind alle miteinander verbunden). Trennen Sie die Masseseite der Schaltung von der Stromversorgung und verbinden Sie sie mit RLY1 auf dem Stecker J12 rechts auf dem Bord. Verbinden Sie die Masse der externen Stromversorgung mit einem GND- oder ⊥ Pin auf dem Bord. Jetzt benötigen wir ein Signal, um den Treiber zu steuern. Für den `ocol` -Test benutzen wir GPIO4, um den Open-Kollektor-Treiber zu steuern (Sie können natürlich irgendein logisches Signal benutzen). Dazu verbinden wir GP4 auf J2 mit RLY1 auf J4 (um einen anderen Treiber n zu testen, muss die Masseseite der Schaltung mit RLYn auf dem Stecker rechts auf dem Bord verbunden werden und GP4 auf Stecker J2 mit RLYn auf J4).
Nun, wenn RLY1 auf J4 auf low gesetzt wird, erhält die Schaltung keinen Strom und ist aus. Wenn RLY1 auf J4 high wird, verbindet der Open-Kollektor-Treiber über Transistoren die Masseseite der

Schaltung mit der Masse des Bords und, weil diese mit der Masse der externen Stromversorgung verbunden ist, kann Strom durch die Schaltung fließen.

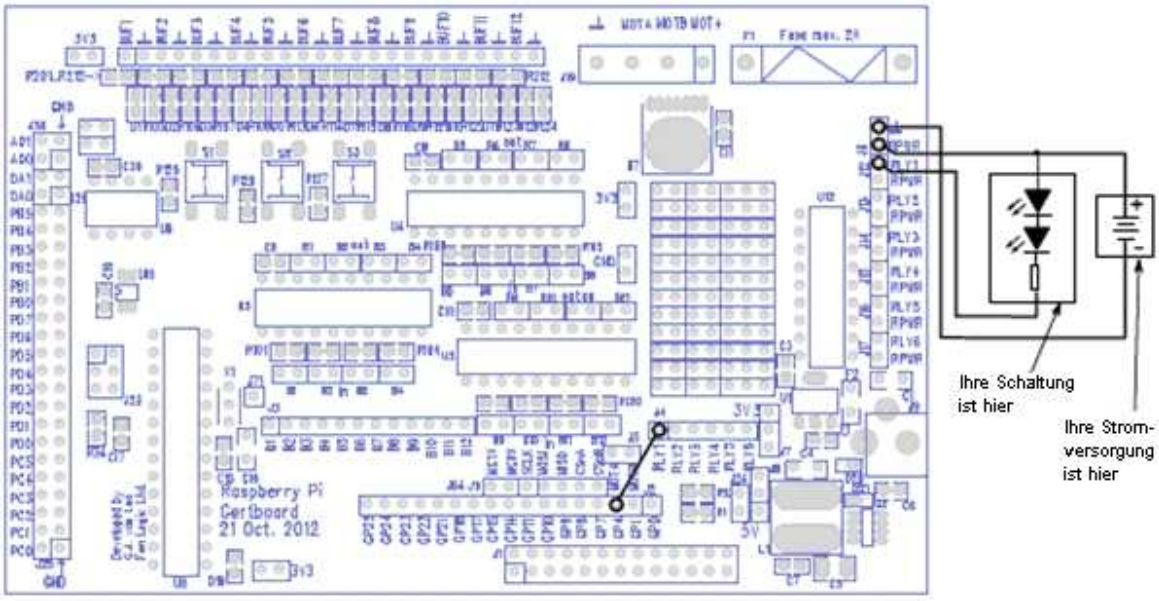


Bild 16: Verdrahtungsschema, um die Open-Kollektor-Treiber zu testen. Rechts ist eine kleine Testschaltung aus zwei LEDs und einem 90 Ω Widerstand. Eine 9V-Batterie dient als Stromversorgung.

Sie werden sich wundern, warum der Pluspol der Stromversorgung mit dem Open-Kollektor-Treiber verbunden werden muss (über das RPWR-Pin). Der Grund ist: wenn die Schaltung eine Induktivität enthält, z. B. einen Motor oder ein Relais, und die Spannung abgeschaltet wird, kann die Induktivität eine Spannung am RLYn-Pin induzieren, die höher sein kann als die Spannung der externen Stromversorgung. Der Chip selbst hat eine interne Schutzdiode, die das RLYn-Pin mit RPWR verbindet (das ist die Diode oben im Bild 15). Das ermöglicht dem Strom, zur positiven Seite der Schaltung zu fließen. Die Energie wird abgeleitet und eine Beschädigung der Schaltung verhindert.

Test des Open Kollektor in C

Der `ocol`- Test ist sehr einfach. Zuerst werden die benötigten Verbindungen auf dem Bildschirm angezeigt (mit Ihrer externen Schaltung und Stromversorgung) und anschließend wird `setup_io` aufgerufen, um die GPIO- Schnittstelle für die Benutzung zu initialisieren und `setup_gpio`, um das Pin GPIO4 als Ausgang zu öffnen (mit den Kommandos `INP_GPIO(4); OUT_GPIO(4);` wie auf Seite 13 beschrieben). Dann benutzt das Programm GPIO_SET0 und GPIO_CLR0 (siehe Seite 21), um GPIO4 zehn Mal high und low zu setzen. Beachten Sie: Das Testprogramm fragt zwar, welcher Treiber getestet werden soll, aber es benutzt diese Information nur, um die dazu benötigten Verbindungen anzuzeigen. Ansonsten wird die Eingabe ignoriert.

Test des Open Kollektor in Python

Die beiden Open-Kollektor-Programme `ocol-wp.py` und `ocol-rp.py` sind identisch, außer dem benutzten GPIO-System. Im Programm organisiert die Funktion `which_channel()` die Kanalauswahl des Benutzers. Wenn eine gültige Eingabe erkannt wird, werden die notwendigen Verbindungen angezeigt. Wenn die Enter-Taste zur Bestätigung der richtigen Verbindungen gedrückt wurde, schaltet das Programm den gewählten Open-Kollektor-Port zehn Mal ein und aus, mit einer Pause von 0,4 Sekunden zwischen jedem Schaltvorgang. Am Ende oder wenn Ctrl-C gedrückt wurde, werden die GPIO-Ports zurückgesetzt.

Vorschläge zum Experimentieren: Ändern Sie die folgenden Werte und sehen Sie, was passiert...

- `sleep(0.4)` - ändern Sie die 0.4
- `for i in range(10)` - ändern Sie die 10
- sehen Sie was passiert, wenn Sie dem Programm mitteilen, Treiber 7 zu benutzen (nachdem das Programm gestartet wurde)
- ändern Sie die Fehlermeldung, die bei der Auswahl einer falschen Portnummer angezeigt wird in eine Meldung Ihrer Wahl

Motor-Controller

Das Gertboard hat einen ROHM BD6222HFP Motor-Controller. Der Motor-Controller ist für Bürsten-Gleichstrom (DC)-Motoren und kann mit einer maximalen Spannung von 18V und einem maximalen Strom von 2A umgehen.

Der Controller hat zwei Eingangs-Pins, A und B (bezeichnet mit MOTA und MOTB auf dem Bord). Die Pins können high oder low getrieben werden und der Motor reagiert entsprechend der unteren Tabelle. Die Geschwindigkeit des Motors kann über eine Pulsweitenmodulation (PWM) entweder am A- oder am B-Pin gesteuert werden.

A	B	Verhalten des Motors
0	0	Keine Bewegung
0	1	Dreht in einer Richtung
1	0	Dreht in die andere Richtung als oben
1	1	Keine Bewegung

Tabelle 3 zeigt das Motorverhalten für verschiedene logische Kombinationen am Motor-Controller

Der Motor-Controller-IC hat eine interne Temperaturüberwachung und eine Schutz vor Überhitzung. Eine Sicherung auf dem Gertboard schützt vor Überstrom. Der Motor-Controller ist im Schaltbild auf Seite A-4 zu sehen.

Im Funktionsblock-Schema (Bild 2 auf Seite 6) ist der Bereich für den Motor-Controller violett markiert. Der Motor-Controller und die Schraubanschlüsse sind nahe beim oberen Rand des Bords und es gibt zwei Pins für die Steuersignale auf J5, einem kleinen Stecker genau oberhalb von GP4 und GP1 auf dem Stecker J2. Die MOTA- und MOTB-Pins auf J5 sind die Eingänge des Motor-Controllers– es sind digitale Signale (low oder high). Die Schraubanschlüsse oben auf dem Bord mit der Bezeichnung MOTA und MOTB sind die Ausgänge des Motor-Controllers: sie liefern den Strom für den Motor. Wenn der Motor mehr Spannung oder Strom benötigt, wird dies über das Gertboard geliefert. Die Schraubanschlüsse oben mit der Bezeichnung MOT+ und ⊥ erlauben den Anschluss einer externen Stromversorgung, die die benötigte Energie liefert: Der Motor-Controller leitet diese Spannung an die Anschlüsse MOTA und MOTB weiter, wobei die Modulation entsprechend der MOTA und MOTB-Eingänge auf J5 erfolgt.

Wenn Sie den Motor nur in beiden Drehrichtungen ein- oder ausschalten möchten, kann dies einfach erfolgen, indem zwei GPIO-Pins mit den MOTA- und MOTB-Eingängen verbunden werden. Jetzt kann man den Motor steuern, indem die Pins entsprechend der Tabelle 3 gesetzt werden. Um die Motor-Geschwindigkeit zu steuern, wird die Pulsweitenmodulation (PWM) benötigt. Das ist eine Baugruppe, die ein Rechtecksignal ausgibt, wie in Bild 17 zu sehen.

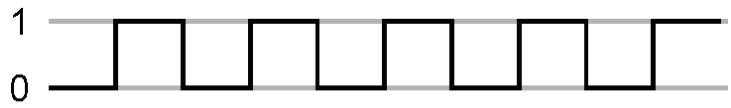


Bild 17: Ein Beispiel für ein PWM-Ausgangssignal. Hier ist das Signal 50% der Zeit auf high, so dass die Taktrate 50% beträgt.

Mit PWM können Sie steuern, wie lange das Signal jeweils high bzw. low ist. Das wird als Taktrate (duty cycle) bezeichnet und in Prozent angegeben. Bild 17 oben zeigt eine Taktrate von 50%, das Bild 18 eine Taktrate von 25%.

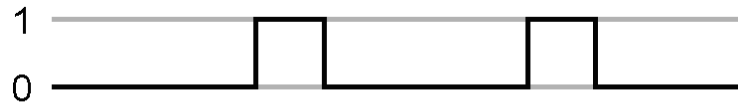


Bild 18: In diesem Beispiel beträgt die Taktrate 25%

Es gibt eine PWM auf dem BCM2835 (dem Raspberry Pi-Prozessor), und der Ausgang kann über die GPIO18 abgegriffen werden (das ist die alternative Funktion 5). Wenn dieses Pin mit einem Eingang des Motor-Controllers verbunden wird (MOTA wird in unserem Motortest benutzt) und der andere Eingang des Motor-Controllers (MOTB in unserem Test) wird auf statisch high oder low gesetzt, dann können die Geschwindigkeit und die Drehrichtung des Motors gesteuert werden.

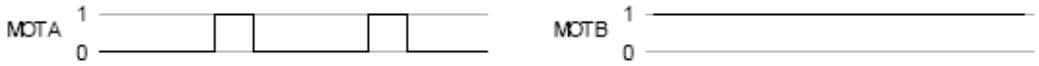


Bild 19: Die Drehrichtung des Motors wird durch MOTB gesetzt. Während MOTA eine Taktrate von 25% hat, bekommt der Motor immer dann Strom, wenn MOTA und MOTB unterschiedlich sind, also für 75% der Zeit.

Ein Beispiel (Bild 19): Wir wechseln zwischen A low / B high und A high / B high. Wenn A low ist, bekommt der Motor Strom und dreht in einer Richtung. wenn A high ist, bekommt er keinen Strom. Das Resultat der Taktrate von 25%, die hier gezeigt wird ist, dass der Motor mit ungefähr ¾ Geschwindigkeit dreht.

Bild 20: In diesem Beispiel dreht der Motor in entgegengesetzter Richtung mit ca. 25% Drehzahl.

Wenn Sie andererseits MOTB auf low setzen, wie in Bild 20, dann bekommt der Motor Strom, wenn MOTA high ist, aber er dreht in die andere Richtung. Ist MOTA low, dann bekommt der Motor keinen Strom. Das Resultat der 25% Taktrate ist, dass der Motor mit ca. ¼ Geschwindigkeit in die andere Richtung als im Bild 19 dreht.

Test des Motor-Controllers

Das C-Testprogramm für den Motor-Controller heißt `motor`. In Python gibt es zwei Versionen, `motor-rg.py` und `motor-wp.py`. Um das Gertboard für den Test vorzubereiten, wird GP17 auf J2 mit dem MOTB-Pin verbunden (das MOTB-Pin auf J5, nicht das am oberen Rand des Bords), und GP18 mit MOTA auf J5. Der Motor wird an die MOTA- und MOTB-Schraubanschlüsse oben auf dem Bord, die Stromversorgung für den Motor wird an die MOT+ und ⊥ Schraubanschlüsse angeschlossen Das ist im Bild 21 zu sehen.

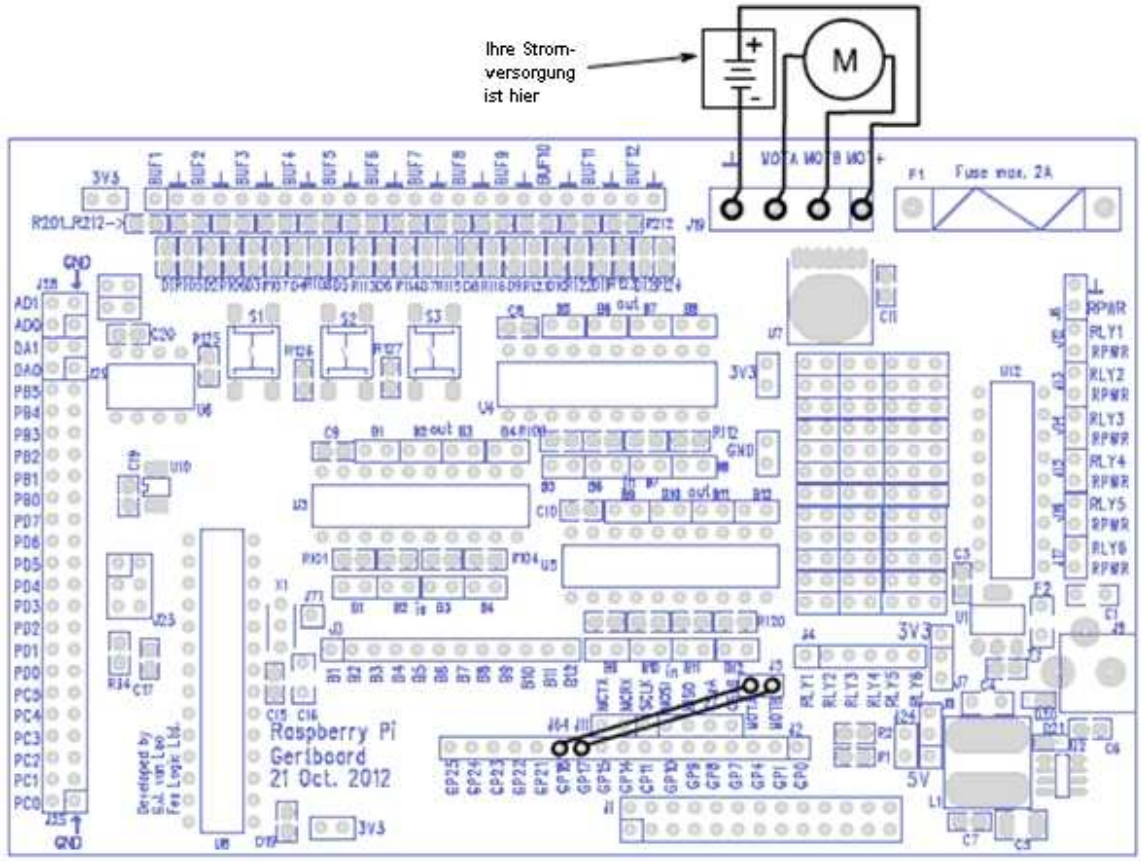


Bild 21: Das Verbindungsschema für den Motortest.

Motortest in C

Der PWM wird über einen Speicherbereich gesteuert, ähnlich den GPIO und dem SPI-Bus. Dieser Speicher ist Teil der Funktion `setup_io` in `gb_common.c`, so dass der PWM entweder benutzt wird oder nicht. Weiterer Setup-Code befindet sich in `gb_pwm.c`, mit einer zugeordneten Header-Datei `gb_pwm.h` (engl. header file – diese Datei enthält z. B. Definitionen von Konstanten). Die Funktion `setup_pwm` in `gb_pwm.c` setzt die Frequenz des PWM-Taktes und den Maximalwert des PWM auf 1024: das ist der Wert, bei dem die Taktrate des PWM 100% ist. Außerdem wird der PWM ausgeschaltet. Die beiden Routinen `set_pwm0` und `force_pwm0` setzen den Wert, der die Taktrate des PWM steuert. `set_pwm0` setzt den Wert (wobei zuerst geprüft wird, ob er zwischen 0 und 1024 liegt), aber da es nur bestimmte Punkte im PWM-Takt gibt, bei denen ein neuer Wert gesetzt wird, hat das Setzen eines Wertes keinen Effekt, wenn kurz danach ein zweiter Wert gesetzt wird. Die Routine `force_pwm0` besitzt zwei Argumente, ein neuer Wert und ein neuer Modus. Es schaltet den PWM ab, setzt anschließend den Wert und schaltet den PWM im neuen Modus wieder ein. Dabei werden an „strategischen“ Punkten Pausen eingelegt, um der Hardware Gelegenheit zu geben, die Werte zu übernehmen. Die Routine `pwm_off` schaltet den PWM aus.

Der Code für das Motorprogramm befindet sich in `motor.c`. In der Hauptroutine `main` werden zuerst die nötigen Verbindungen für den Test auf dem Bildschirm angezeigt, danach wird `setup_io` aufgerufen, um die GPIO-Schnittstelle zu öffnen. Danach wird `setup_gpio` aufgerufen, um GPIO18 als PWM-Ausgang zu konfigurieren und GPIO17 als normalen Ausgang. Für den zweiten werden sowohl `INP_GPIO` als auch `OUT_GPIO` benutzt (siehe Seite 13 für mehr Informationen). Um GPIO18 zu konfigurieren, wird zuerst `INP_GPIO(18)` benutzt, um das Pin zu aktivieren. Eine der alternativen Funktionen von GPIO18 ist es, als Ausgang für den PWM zu fungieren. Das ist Alternative Nr. 5, darum benutzen wir das Makro `SET_GPIO_ALT(18, 5),` um die alternative Funktion zu aktivieren (siehe Tabelle 6-31 des BCM2835 Datenblattes oder die Onlineversion auf `http://elinux.org/RPi_BCM2835_GPIOs` für weitere Informationen über alternative Funktionen der GPIO-Pins. Für eine Zusammenfassung der alternativen Funktionen, die auf dem Gertboard benutzt werden, siehe Tabelle 1 auf Seite 11).

Wir setzen den Ausgang von GPIO17 auf low (um sicherzustellen, dass der Motor nicht läuft) und initialisieren danach den PWM durch den Aufruf von `setup_pwm`. Wir aktivieren den PWM durch Setzen des Modus auf PWM0_ENABLE, wobei `force_pwm0` benutzt wird. Da GPIO17 (Eingang B des Motor-Controllers) auf low gesetzt ist, fängt der Motor an, entgegengesetzt zu drehen (entsprechend der Tabelle auf Seite 27), wenn die Taktrate des PWM (Eingang A des Motor-Controllers) hoch genug ist.

Nun startet eine Schleife, in der der PWM gestartet wird, anfangs mit einer sehr kleinen Taktrate (weil der Wert, der an `set_pwm0` übergeben wird, klein ist), dann schrittweise höher bis zum Maximum (welches auf 0x400 – 1024 – in `setup_pwm` gesetzt wird). Danach wird der an den PWM gesendete Wert verkleinert, um den Motor zu drosseln. Anschließend wird GPIO17 auf high gesetzt, so dass der Motor während der low-Phase des PWM-Signals Strom bekommt. Der PWM wird im Modus PWM0_ENABLE | PWM0_REVPOLAR aktiviert (| bedeutet eine logische ODER-Verknüpfung der beiden Werte). Das umgedrehte Polarisationssignal PWM0_REVPOLAR dreht das PWM-Signal um, so dass ein kleiner Wert, der zum PWM gesendet wird, ein Signal bewirkt, das die meiste Zeit auf high ist. Auf diese Art kann derselbe Code benutzt werden, um die Motordrehzahl langsam zu steigern (aber diesmal in der vorwärts-Richtung, entsprechend der Tabelle auf Seite 27), um dann wieder die Drehzahl zu vermindern. Zum Schluss wird der PWM deaktiviert und die GPIO-Schnittstelle geschlossen.

Motortest in Python

Die Motor-Testprogramme `rg.py` und `motor-wp.py` sind ziemlich unterschiedlich, weil das RPi.GPIO-Paket noch nicht Hardware-Pulsweitenmodulation (PWM) unterstützt, aber WiringPi für Python tut dies (es funktioniert, aber es ist noch nicht dokumentiert). Somit benutzt `motor-rg.py` (die RPi.GPIO-Version) Software-PWM, die sich in der Funktion `run_motor()` befindet. Wenn Sie beide Programme ausprobieren, bekommen Sie wahrscheinlich bessere Ergebnisse mit `motor-wp.py` (der WiringPi für Python-Version).

Beide Versionen benutzen die Python 3-Funktion `print()`, die wie folgt importiert wird

```
from __future__ import print_function
```

Ohne diese Funktion ist es schwierig, Zeilenumbrüche und Leerzeichen bei der Bildschirmausgabe zu verhindern. Das heißt, dass alle Anzeigebefehle als `print()` geschrieben werden müssen anstatt einfach als `print`, weil sie jetzt Funktionsaufrufe sind und nicht einfache Befehle.

motor-rg.py (Software-PWM)

Nach dem Import der benötigten Module definieren wir, welche Ports benutzt werden sollen (18 und 17), wie oft jede Schleife durchlaufen werden soll (Reps), die PWM-Taktfrequenz (Hertz), die Frequenz für die Motor-Taktrate (engl. Motor loop time period) (Freq). Danach werden die Ports als Ausgang konfiguriert und auf Aus (0) gesetzt.

Danach wird die Funktion

```
run_motor(Reps, pulse_width, port_num, time_period)
```

definiert. Sie steuert den GPIO-Port für eine genaue Zeit ein und aus (high oder low). Der Port `port_num` wird für die Zeit `pulse_width` eingeschaltet, dann für die Zeit `time_period` ausgeschaltet und all das wird `Reps`-Mal wiederholt (400 in diesem Fall). Die Hauptschleife in `run_motor()` befindet sich in einem `try: ... except`:-Block. Das ist eine wichtige Sicherheitsmaßnahme, um das Ausschalten des Motors bei einem Tastatur-Interrupt zu gewährleisten. (Das ist sogar noch wichtiger bei Hardware-PWM in der anderen Version). Die Funktion `run_motor()` wird von der Funktion `run_loop()` aufgerufen. Jeder der 400 Aufrufe ist genau ein Schritt auf oder ab in der Motordrehzahl: 400 Wiederholungen bei 2000 Hertz ergeben 0.2 Sekunden.

Nun definieren wir die Funktion `run_loop()`

```
run_loop(startloop, endloop, step, port_num, printchar)
```

Die Argumente `startloop` und `endloop` sind die %-Zeit für den geschalteten Port für EIN bei Beginn und Ende der Schleife, `step` ist der Inkrementalwert für jede folgende Schleife, `port_num` ist der geschaltete Port (17 oder 18), `printchar` ist das Zeichen für Beschleunigung bzw. Abbremsung ('+' oder '-'). Die Funktion `run_loop()` steuert die wiederholten Aufrufe der Funktion `run_motor()`, es erfolgen 400 Aufrufe mit den vorgegebenen Werten.

Nachdem die Funktionen definiert sind, werden die Verdrahtungsinformationen angezeigt und das Programm wartet auf die Enter-Taste, mit der der Benutzer den Test freigibt. Danach wird die Funktion `run_loop()` vier Mal aufgerufen, wobei die Motordrehzahl von 5% bis 95% und dann wieder zurück gesteuert wird, und zwar in jeder Drehrichtung. Danach werden die Ports auf AUS gestellt und zurückgesetzt.

Vorschläge zum Experimentieren: Ändern Sie diese Werte und beobachten Sie, was passiert...

- `run_loop(5, 95, 1, 18,'+')` – ändern Sie das + in ein anderes Zeichen
- `run_loop(5, 95, 1, 18,'+')` – ändern Sie die 5 in eine größere Zahl < 95
- `run_loop(5, 95, 1, 18,'+')` – ändern Sie die 95 in eine kleinere Zahl (aber größer als die Zahl, die an Stelle der 5 steht)

motor-wp.py (Hardware- PWM)

Der Raspberry Pi hat einen verfügbaren Hardware-PWM-Port (GPIO18). Wir benutzen Port 17, um die Drehrichtung zu steuern und Port 18, um das Pulsen zu steuern.

Der erste Teil des Programms importiert die erforderlichen Module, einschließlich der Python 3-Funktion `print()` (erläutert weiter oben im Abschnitt für `motor-rg.py`). Nach der Initialisierung von WiringPi wird Port 18 mittels `wiringpi.pinMode(18,2)` in den PWM-Modus gesetzt und Port 17 wird als normaler Ausgang konfiguriert. Dann werden beide Ports auf 0 gesetzt (AUS). Nun werden die Verdrahtungshinweise angezeigt und der Computer wartet auf die Bestätigung vom Nutzer. Wenn die erfolgt ist, definiert das Programm drei Funktionen, die periodisch ausgeführt werden.

`display(printchar)` sorgt für die korrekte Anzeige der Motorbeschleunigung bzw. -abbremsung unter Benutzung der importierten Python 3-Funktion `print()`.

`reset_ports()` sorgt für das Rücksetzen der Ports beim Programmende. Das ist nicht in WiringPi für Python vorhanden, so dass es hier definiert werden muss. Das ist besonders wichtig im Motorprogramm, weil es einen unkontrollierten Motorlauf nach Programmende verhindert. Das ist wichtige Sicherheit, wenn Sie Dinge wie z.B. Propeller usw. antreiben.

`loop(start_pwm, stop_pwm, step, printchar)` ist die Haupt-PWM-Steuerungsschleife. Wie zuvor wird diese auf vier verschiedene Arten aufgerufen (Drehzahl erhöhen, dann verringern, dann wieder in Gegenrichtung erhöhen und verringern). `start_pwm` ist der 0 bis 1024 PWM-Wert, mit dem die Schleife gestartet wird, `stop_pwm` ist der 0 bis 1024 PWM-Wert, mit dem die Schleife beendet wird.
`step` ist der +/- Differenzwert für jeden folgenden Schleifendurchlauf. `Printchar` ist ein Zeichen, um anzuzeigen, ob der Motor beschleunigt oder abbremst.

Das Hauptprogramm enthält vier Aufrufe von `loop()`, um das Beschleunigen und Abbremsen in jeder Drehrichtung zu demonstrieren, mit einer kleinen Pause zwischen den Schleifendurchläufen, die durch `rest = 0.013` (=0,013s) definiert wird. Wenn Port 17 null ist, dreht der Motor in eine Richtung, ist Port 17 eins, dreht er in die andere Richtung. Wie üblich, wird das Hauptprogramm in einen `try: ... except`: -Block eingeschlossen, um das sichere Rücksetzen der Ports zu garantieren, wenn

der Benutzer das Programm über Ctrl-C beendet.

Vorschläge zum Experimentieren: Ändern Sie die folgenden Werte und beobachten Sie, was passiert...

- `rest = 0.013` – ändern Sie 0.013 zu 0 und sehen Sie, wozu es da ist
- `loop(140, 1024, 1, '+')` – ändern Sie 140 in einen Wert < 140, aber > 0
- `loop(140, 1024, 1, '+')` – ändern Sie 1024 in eine kleinere Zahl > 140
- `loop(140, 1024, 1, '+')` – ändern Sie + in ein anderes Zeichen

Digital- zu analog- und analog- zu digital- Konverter

Im Funktionsblock-Schema des Gertboards (Bild 2, Seite 6) sind die Komponenten des Konverters orange markiert. Sowohl der digital- zu analog- Wandler (D/A) als auch der analog- zu digital- Wandler (A/D) sind 8-Pin Chips von Microchip, obwohl sie seltsamerweise in verschiedenen Gehäusen untergebracht sind. Der A/D- Wandler (bezeichnet mit U6 auf dem Bord) ist in einem dual in-line Gehäuse, während der D/A- Wandler (U10) ein SMD-Gehäuse hat (surface mounted device). Jeder unterstützt 2 Kanäle.

Beide benutzen den SPI-Bus, um mit dem Raspberry Pi Daten auszutauschen. Die SPI-Pins an den zwei Chips sind mit den Pins SCLK, MOSI, MISO, CSnA und CSnB auf dem Stecker genau oberhalb von J2 auf dem Bord verbunden (somit sind diese Pins im Funktionsblock-Schema ebenfalls orange markiert). SCLK ist der Takt, MOSI (Master Output / Slave Input) ist der Ausgang vom RPi und MISO (Master Input / Slave Output) ist der Eingang zum RPi. CSnA ist das Chip-Select-Signal für den A/D- und CSnB ist das Chip-Select-Signal für den D/A- Wandler (das ‘n’ im Signalnamen bedeutet, dass das Signal ‘negativ’ ist, somit ist der Chip nur aktiv, wenn das Signal low (0) ist). Sowohl der A/D- als auch der D/A-Chip haben jeweils einen 10K pull-up-Widerstand an ihrem Chip-Select-Pin, somit sind die Chips nicht aktiv, wenn die Pins nicht verbunden sind.

Die SPI-Pins sind günstigerweise genau oberhalb von GP7 bis GP11auf Stecker J2 platziert, weil eine der alternativen Funktionen dieser Pins das Treiben der SPI-Signale ist. Zum Beispiel ist die Funktion ‘ALT0’ (Alternative 0) von GPIO9 gleich SPI0_MISO - deshalb befindet sich das Pin MISO genau neben dem Pin GP9. Somit ist die Benutzung der A/D- und D/A- Wandler einfach, indem mit Jumpern die Pins GP7 bis GP11 mit den SPI-Pins direkt neben ihnen verbunden werden (obwohl Sie technisch gesehen nur CSnA für den A/D- und CSnB für den D/A- Wandler benötigen).

In der Schaltung sind die D/A- und A/D- Wandler in der linken oberen Ecke auf Seite A-6 zu sehen.

Digital- zu analog- Konverter (D/A)

Das Gertboard benutzt einen MCP48x2 digital- zu analog- Wandler (D/A) von Microchip. Der IC wird in drei unterschiedlichen Typen hergestellt: mit 8, 10 oder 12 Bits. Der auf Ihrem Bord ist abhängig von der Verfügbarkeit der Teile. Um zu bestimmen, welchen Sie haben, sehen Sie sich den kleinen Chip U10 genauer an. Er sollte eine Nummer 48x2 tragen, wobei x entweder 0, 1 oder 2 ist. Wenn x = 0 ist, haben Sie die 8-Bit-Version. Wenn es 1 ist, haben Sie die 10-Bit-Version und bei x = 2 die 12-Bit-Version. Diese ICs sind alle Pin-kompatibel und werden auf dieselbe Weise programmiert. Im Einzelnen: die Routine, die zum D/A-Wandler schreibt, geht vom 12-Bit-Format aus, so dass es wichtig ist, den Wert passend auszuwählen (Details siehe unten im Abschnitt “Test der D/A und A/D- Wandler”). Die maximale Ausgangsspannung des D/A- Wandlers – die Ausgangsspannung, wenn als Eingang alle Bits 1 sind – beträgt 2,04 V.

Die analogen Ausgänge der beiden Kanäle gehen auf Pins mit der Bezeichnung DA0 (für Kanal 0) und DA1 (Kanal 1) auf dem Stecker J29 am linken Rand des Bords. Gleich neben diesen Pins sind die Masse-Pins (GND), die die Referenz liefern.

Analog- zu digital- Konverter (A/D)

Das Gertboard benutzt einen MCP3002 10-Bit analog- zu digital- Wandler von Microchip. Er unterstützt 2 Kanäle mit einer Abtastrate (sampling rate) von ~72k Messungen pro Sekunde (samples per second - sps). Der Maximalwert (1023) wird geliefert, wenn die Eingangsspannung 3,3V beträgt.

Die Analogeingänge für diese zwei Kanäle sind AD0 (für Kanal 0) und AD1 (für Kanal 1) auf dem Stecker J28. Gleich neben diesen Pins sind die Masse-Pins (GND), die die Referenz liefern.

Testen der D/A- und A/D- Konverter

Entsprechend der Datenblätter für die D/A- Wandler wird der Wert am Ausgangspin V_{out} mit folgender Formel berechnet: (für den 8-bit MCP4802):

$$Vout = \frac{D_{in}}{256} \times 2.048V$$

V_{out} für Kanal 0 ist DA0 auf J29; für Kanal 1 ist es DA1.

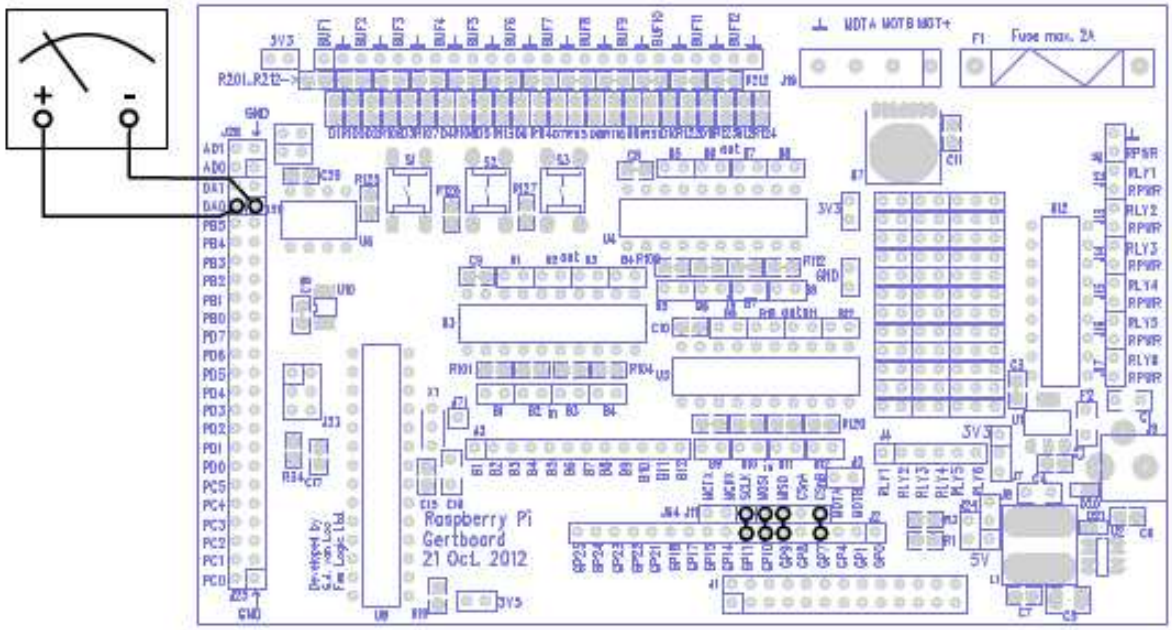


Bild 22: Verdrahtungsschema für den D/A-Test.

Um den D/A- Wandler zu testen, wird ein Multimeter benötigt. Das C-Testprogramm ist `dtoa`. Die Python-Version ist `dtoa.py`. Um das Gertboard für den Test zu konfigurieren, werden Jumper auf den Pins GP11, GP10, GP9 und GP7 platziert, die sie mit den Pins vom SPI-Bus darüber verbinden. Verbinden Sie das Multimeter wie folgt: das schwarze Kabel muss mit Masse verbunden werden. Sie können dazu jedes Pin benutzen, das mit ⊥ oder GND bezeichnet ist.
Das rote Kabel muss mit DA0 verbunden werden (um den D/A-Kanal 0 zu testen, wie unten gezeigt) oder mit DA1 (für Kanal 1). Schalten Sie das Multimeter ein und wählen Sie den Spannungsbereich von 0 bis rund 5V. All das wird im Bild 22 gezeigt.

Das C-Testprogramm für den A/D-Wandler heißt `atod`; die Python-Version ist `atod.py`. Für diesen Test wird eine Spannungsquelle an den analogen Eingängen benötigt. Das wird am besten mit einem Potentiometer realisiert. Die beiden Anschlüsse des Potis werden einmal mit high (3,3V, die Sie von

einem Pin, das mit 3V3 bezeichnet ist, abgreifen können) und zum anderen mit Masse (GND oder ⊥) verbunden. Die Mittelanzapfung des Potis wird mit AD0 (für Kanal 0 wie unten gezeigt) oder mit AD1 (für Kanal 1) verbunden. Die Jumper für die Benutzung des SPI-Busses müssen auf den Pins GP11, GP10, GP9 und GP8 stecken, um diese mit dem SPI-Bus zu verbinden. Das ist in Bild 23 zu sehen.

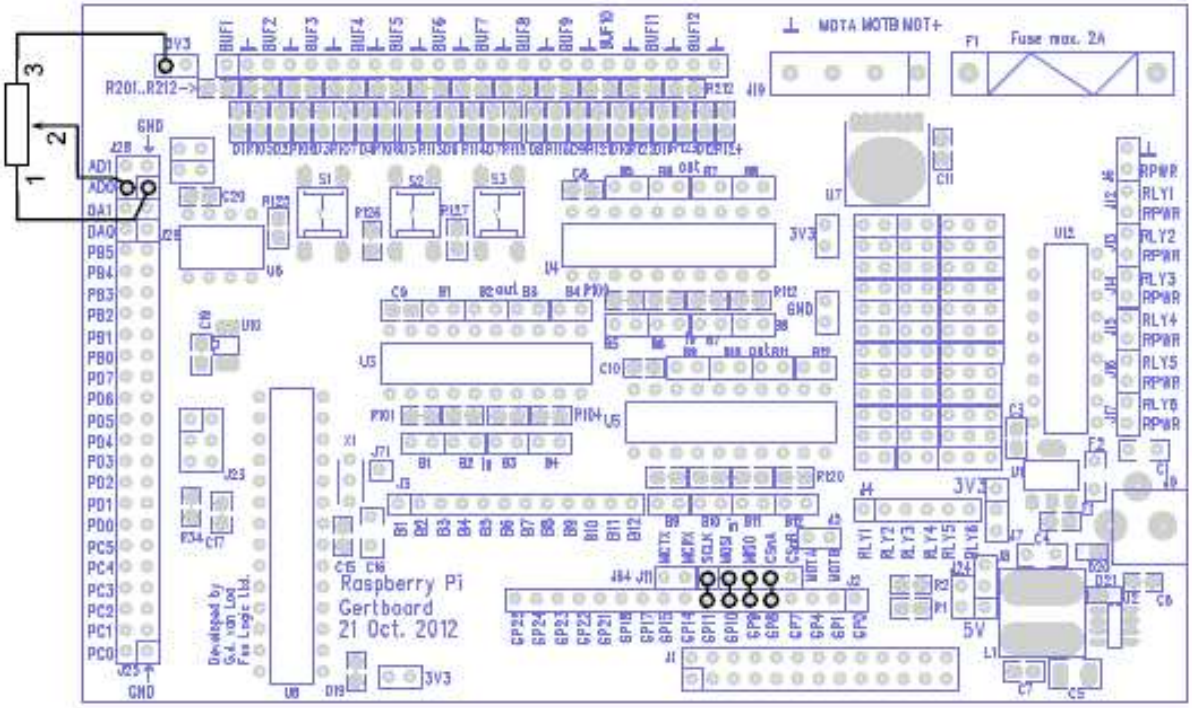


Bild 23: Verdrahtungsschema für das Testprogramm atod.

Sogar ohne Multimeter und ohne Potentiometer können die A/D- und D/A- Wandler getestet werden, indem der Ausgang des D/A- Wandlers mit dem Eingang des A/D- Wandlers verbunden wird. Dieser Test heißt `dad` für digital-analog-digital. Um das Gertboard für diesen Test zu konfigurieren, stecken Sie alle Jumper auf die Pins des SPI-Busses (die GP11 bis GP7 mit den Pins des SPI-Busses darüber verbinden) und stecken einen Jumper zwischen den Pins DA1 und AD0, wie im Schema unten.

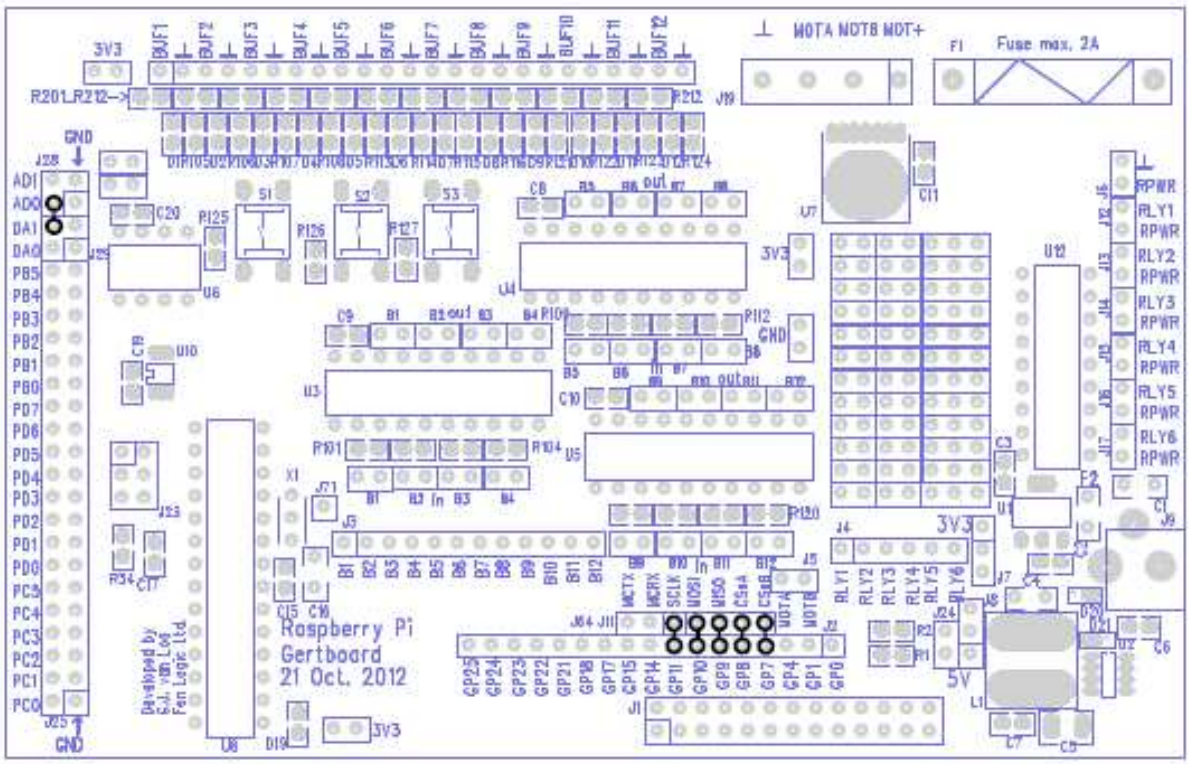


Bild 24: Verdrahtungsschema für das Testprogramm dad, das den Test der A/D- und D/A-Wandler zusammen erlaubt, ohne ein Multimeter oder ein Potentiometer zu benutzen.

Test der D/A- und A/D- Wandler in C

Da die D/A- und A/D- Wandler beide den SPI-Bus benutzen, wurde der allgemeine Code für den SPI-Bus in eine separate Datei ausgelagert - `gb_spi.c`. Es gibt außerdem eine zugeordnete Header-Datei - `gb_spi.h`, die viele Makros und Konstanten enthält, die für die Arbeit mit dem SPI-Bus benötigt werden, genauso wie Deklarationen für die Funktionen in `gb_spi.c`. Diese Funktionen sind `setup_spi, read_adc` und `write_dac`. `setup_spi` setzt die Taktgeschwindigkeit für den Bus und löscht die Statusbits. `read_adc` hat ein Argument, das den Kanal angibt (muss 0 oder 1 sein) und gibt eine Ganzzahl mit dem gelesenen Wert vom A/D- Wandler zurück, Der Rückgabewert liegt zwischen 0 und 1023 (d.h. nur die niedrigsten [least significant] 10 Bits werden gesetzt), wobei der Wert 0 zurückgeliefert wird, wenn die Spannung am Eingang 0V beträgt und 1023 bei 3,3V am Eingang.

Die Routine `write_dac` hat zwei Argumente: die Nummer des Kanals (0 oder 1) und den zu schreibenden Wert. Der zu schreibende Wert erfordert eine Erklärung. Die MCP48xx-Familie von digital- zu analog- Konvertern akzeptieren alle 12 Bit-Werte. Der IC MCP4822 benutzt alle Bits; der MCP4812 ignoriert die letzten zwei und der MCP4802 ignoriert die letzten 4 Bits. Da irgendeiner dieser Chips auf dem Gertboard vorhanden ist (abhängig von der Verfügbarkeit), wurde `write_dac` so geschrieben, dass es mit allen drei ICs arbeitet, indem es einfach den übergebenen Wert an die D/A-Wandler sendet. Wenn das Gertboard mit dem MCP4802 ausgestattet ist, kann es nur Werte zwischen 0 und 255 verarbeiten, aber dieser Wert muss in den Bits 4 bis 11 (das niederwertigste Bit ist Bit 0) des gesendeten Bit-Stromes (bit-stream) stehen. Somit muss die an den D/A-Wandler zu sendende Zahl (sie muss zwischen 0 und 255 liegen) mit 16 multipliziert werden (was die Bits 4 Stellen nach links schiebt), bevor sie an `write_dac` übergeben wird.

dtoa

Um den D/A- Wandler zu testen, fragt das Programm `dtoa` zuerst den zu benutzenden Kanal ab und zeigt die dazu nötigen Verbindungen an. Danach ruft es `setup_io` auf, um die GPIO-Ports zu initialisieren, dann wird `setup_gpio` aufgerufen, um festzulegen, welche Pins wie benutzt werden. In `setup_gpio` wird wie üblich `INP_GPIO(n)` benutzt, (wobei n die Pinnummer ist), um die Pins zu

aktivieren. Das öffnet die Pins als Eingang. Sie sollen jedoch als SPI-Bus benutzt werden, was eine der alternativen Funktionen für diese Pins ist (es ist Alternative 0). Wir benutzen `SET_GPIO_ALT(n, a)` (wobei n die Pinnummer und a die Nummer der alternativen Funktion ist, in diesem Fall 0), um diese alternative Funktion der Pins zu aktivieren. Anschließend sendet das Programm unterschiedliche Werte an den D/A- Wandler und erwartet vom Benutzer, der ein Multimeter verwendet, eine Bestätigung, dass der D/A- Wandler die korrekte Ausgangsspannung erzeugt.

atod

Um den A/D- Wandler zu testen, fragt das Programm `atod` zuerst, welcher Kanal benutzt werden soll und zeigt die dazu nötigen Verbindungen an. Danach ruft es `setup_io` auf, um die GPIO-Schnittstelle zu initialisieren. Anschließend legt `setup_gpio` fest, welche Pins wie benutzt werden. Die Funktion `setup_gpio` in `atod` arbeitet genauso wie die in `dtoa` (außer für die Aktivierung von GPIO8 anstelle von GPIO7).

Nun liest `atod` wiederholt den 10-Bit-Wert vom A/D- Wandler und zeigt den Wert auf dem Bildschirm an, sowohl als absolute Zahl als auch als Balken (der gelesene Wert wird durch 16 geteilt und das Ergebnis wird als Zeichenkette von '#'-Zeichen präsentiert). Beachten Sie, dass auch, wenn das Potentiometer nicht verändert wird, u. U. nicht exakt der gleiche Wert bei mehreren Messungen erscheint. Bei 10 Bit Genauigkeit ist die Messung sehr empfindlich und die kleinste Änderung, wie z.B. ein Strom in einem benachbarten Netzkabel, kann den gelesenen Wert beeinflussen.

dad

Um sowohl den D/A- als auch den A/D- Wandler gleichzeitig zu testen, sendet das Programm `dad` 17 unterschiedliche Digitalwerte an den D/A- Wandler (0 bis 255 in gleichen Abständen, dann zurück bis auf 0). Die Ausgangsspannung wird vom A/D- Wandler eingelesen. Sowohl der digitale Originalwert als auch der zurückgelesene Analogwert werden angezeigt, wobei der Balken den zurückgelesenen Wert darstellt (geteilt durch 16 wie in `atod`). Die Balken sollten ein Dreieck auf dem Bildschirm bilden: die Balken sind anfangs kurz, werden anschließend länger und werden dann wieder kürzer.

D/A- und A/D-Tests in Python

Die analog- zu digital- und digital- zu analog- Wandler sind mit den (SPI)-Ports auf dem Raspberry Pi verbunden. Das sind die alternativen Funktionen der GPIO-Ports 7 und 8.

Um `atod.py, dtoa.py` und `dad.py` zu benutzen, muss SPI eingeschaltet sein und Sie müssen einen Python-Modul namens `py-spidev` installieren. Die Anleitung dafür finden Sie in der Datei README.txt, die in den heruntergeladenen Pythonprogrammen enthalten ist.

atod.py

Der Benutzer wählt den zu benutzenden Kanal für den analog- zu digital- Wandler (A/D) aus, dann werden die Verbindungsinformationen für das Gertboard angezeigt. Die Funktion `get_adc(kanal)` benutzt `spidev`, um den A/D- Wandler zu lesen. Sie empfängt drei Datenbytes und berechnet das Resultat – eine Zahl zwischen 0 und 1023, wobei 0 = 0V und 1023 = 3,3V bedeutet.

Die Hauptschleife läuft 600 Mal mit einer kleinen Pause (sleep) von 0.05s, so dass das Programm ca. 30 Sekunden (600 * 0.05) läuft. Während jedes Schleifendurchlaufs liest das Programm den A/D- Wandler und zeigt die gelesene Zahl und einen Balken aus #-Zeichen, dessen Länge proportional zum Wert ist, auf dem Bildschirm an. Der Wert hängt vom Widerstand des Potentiometers ab.
Für die Anzeige wird die Funktion `display(char, reps, adc_value, spaces)` benutzt.

Eine Veränderung des Potis während des Programmlaufs ändert den gelesenen Wert und die Zahl der #-Zeichen (die Balkenlänge) auf der Anzeige.

Vorschläge zum Experimentieren: Ändern Sie die folgenden Werte und beobachten Sie, was passiert...

- `char = '#'` – ändern Sie das #-Zeichen in ein anderes Symbol (in Zeile 30)
- `sleep(0.05)` – ändern Sie die 0.05 und sehen Sie, was passiert (in Zeile 56)

dtoa.py

Der digital- zu analog- Konverter (D/A) wird gesteuert, indem 2 Bytes (16 Bit insgesamt) über die SPI-Schnittstelle an ihn gesendet werden. Das Programm benutzt einen Python-Modul mit dem Namen `spidev`, um die SPI-Kommunikation zu realisieren. Wir müssen `spidev` zwei Zahlen übergeben, die als Bytes (je 8 Bit) an den D/A gesendet werden. Der D/A- Wandler setzt die Ausgangsspannung entsprechend dem übergebenen Wert. Ein Eingang von 0 ergibt 0V, 255 ergibt 2,048V, und da es linear ist, ergibt 128 eine Spannung von 1,02V.

Die vorgegebenen Werte, die wir an den D/A- Wandler senden, werden in zwei Listen-Variablen gespeichert:

- `num_list[]` – enthält das erste Byte (unterschiedlich für jeden Kanal).
- `common[]` – enthält das zweite Datenbyte (gleich für jeden Kanal).

Der Benutzer wählt den Kanal und danach nimmt das Programm in der Hauptschleife den ersten Wert aus jeder Liste und kombiniert diese Werte (Byte1 mit Byte2). Dann wird dieser Wert an den D/A-Wandler gesendet, der sofort die gewünschte Spannung ausgibt, bis die Taste Enter gedrückt wird. Das Programm läuft nun durch die Liste (alle fünf Werte), sendet die Daten zum D/A und wartet auf die Taste. Beide Kanäle werden am Programmende auf null zurückgesetzt.

Vorschläge zum Experimentieren:

- Keine zu diesem Zeitpunkt. Wenn Sie die Zahlen in den Listen ändern, beschädigen Sie das Programm (die Spannung am Ausgang stimmt dann nicht mehr mit den Zahlen, die auf dem Bildschirm angezeigt werden, überein). Wenn Sie sehen möchten, wie sie abgeleitet werden, können Sie in das Programm `dad.py` in die Funktion `dac_write` schauen

dad.py

Dieses Programm benutzt `spidev`, um den A/D- und den D/A- Wandler unter Benutzung der SPI-Ports auf dem Raspberry Pi zu steuern. Zuerst werden die Verbindungsinformationen für das Bord angezeigt. Dann durchläuft das Hauptprogramm zwei Schleifen (eine für jede Richtung), wobei die Werte von 0 bis 256 und wieder zurück durchlaufen werden, bei einer Schrittweite von 32. Der Wert wird an den D/A-Wandler unter Verwendung von `dac_write(DAC_value)` gesendet, und die Ausgangsspannung erscheint am Ausgangspin DA1. Diese Spannung wird über einen Jumper an AD0 angelegt, einem Eingang des A/D- Wandlers, und die Funktion `get_adc(adc_channel)` wird aufgerufen, um den Wert für diese Spannung zurückzulesen (der Wert liegt zwischen 0 und 1023). Beide Werte werden nun zusammen mit einem Balken (der den gelesenen Analogwert repräsentiert) angezeigt (unter Benutzung der #-Zeichen).

Vorschläge zum Experimentieren:

- Ändern Sie die 32 in beiden Schleifen:
  `for DAC_value in range(0,257,32)` und
  `for DAC_value in range(224,-1,-32)`
- Ändern Sie die 4 in `adc_string = "{0:04d}"`
- Ändern Sie die 3 in `print "%s %s %s" % ("{0:03d}")`
- Entfernen Sie den Jumper zwischen AD0 und DA1 und beobachten Sie, was passiert

Kombinierte Tests

Dieser Abschnitt zeigt an Hand von Beispielen die Benutzung von mehreren Funktionsblöcken gleichzeitig.

A/D und Motor-Controller

Im Testprogramm `potmot` (für Potentiometer-Motor) benutzen wir ein Poti, das mit dem analog- zu digital- Konverter (A/D) verbunden ist, um einen Eingangswert zu erhalten, der dazu dient, einen Motor in Drehzahl und Drehrichtung zu steuern. Es wird so konfiguriert, dass der Motor in Mittelstellung des Potis still steht und in beiden Endlagen mit maximaler Drehzahl läuft, jeweils in entgegengesetzter Richtung.

Um das Gertboard für dieses Beispiel vorzubereiten, werden die Verdrahtungsschemas für den A/D-Wandler und den Motortest kombiniert. Jumper verbinden GP8 bis GP11 mit den Pins direkt oberhalb, um mit GP8 bis GP11 den SPI-Bus zu steuern. Sie müssen das Poti an den AD0-Eingang anschließen. GPIO17 steuert den Motor B-Eingang und GPIO18 den Motor A-Eingang über den Pulsweitenmodulator (PWM). Darum muss GP17 über einen Draht mit MOTB und GP18 mit MOTA verbunden werden. Der Motor und seine Stromversorgung müssen an die Schraubklemmen auf J19 oben auf dem Bord angeschlossen werden, siehe Verdrahtungsschema unten.

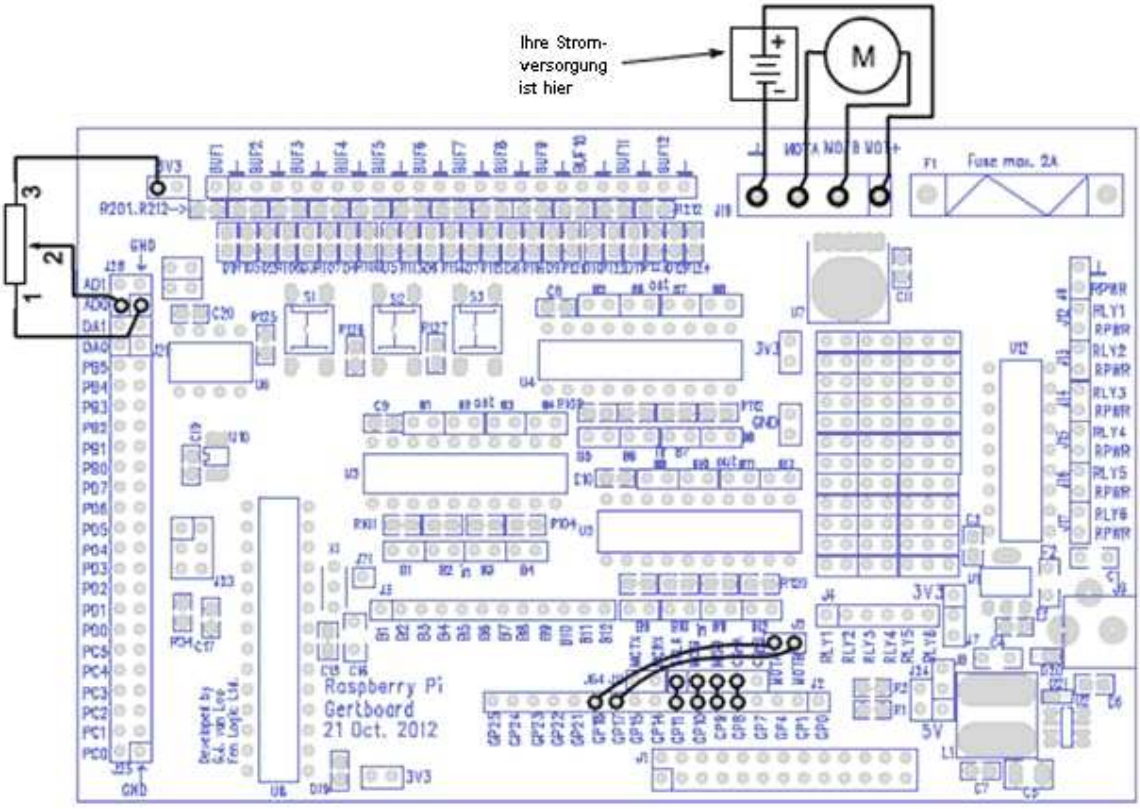


Bild 25: Verdrahtungsschema für das potmot-Programm

Potmot Test in C

Die `main`- (Haupt-) Routine dafür befindet sich in `potmot.c`. Funktionen aus `gb_spi.c` und `gb_pwm.c` werden benutzt, um den SPI-Bus zu steuern (zum Lesen des A/D) und auch den Pulsweitenmodulator (zur Steuerung der Motor-Drehzahl).

In der `main`-Routine für `potmot` werden zuerst die nötigen Verbindungen für das Gertboard angezeigt, anschließend wird `setup_io` aufgerufen, um die GPIO für die Benutzung vorzubereiten. Danach rufen wir `setup_gpio` auf, um die GPIO-Pins zu setzen. Darin werden GPIO8 bis GPIO11 zur Benutzung des SPI-Busses konfiguriert, indem `INP_GPIO` und `SET_GPIO_ALT` wie im Abschnitt über die Konverter (D/A- und A/D-Test in C, Seite 35) benutzt werden. GPIO17 wird als Ausgang gesetzt (mittels `INP_GPIO` und `OUT_GPIO`), und GPIO18 wird als ein PWM gesetzt (mittels `INP_GPIO` und `SET_GPIO_ALT`), wie im Abschnitt über den Motor-Controller (Motor-Test in C, Seite 29). Zurück in `main`, werden `setup_spi` und `setup_pwm` aufgerufen, um den SPI-Bus und den PWM zu konfigurieren und den Motor bereit für die Arbeit zu machen.

Nun lesen wir periodisch den A/D- Wandler und setzen die Geschwindigkeit und die Drehrichtung des Motors abhängig vom gelesenen Wert. Niedrige A/D-Werte (bis zu 511 (erinnern Sie sich, dass der A/D-Chip einen 10-Bit-Wert liefert und der Maximalwert 1023 beträgt) führen dazu, dass der Motor B-Eingang auf high gesetzt wird und der Motor in Drehrichtung 1 dreht, wie in der Tabelle für den Motor-Controller beschrieben (Tabelle 3, Seite 27). Dummerweise wird diese Drehrichtung in den Kommentaren im Programm als “backwards” (rückwärts) bezeichnet! Höhere A/D-Werte (512 bis 1023) führen dazu, dass der Motor B-Eingang auf low gesetzt wird und der Motor in entgegengesetzter Richtung dreht. Diese Richtung wird im Programm mit “forwards” (vorwärts) bezeichnet. Einfache arithmetische Berechnungen transformieren A/D-Werte nahe bei 511 in niedrige Motordrehzahlen und A/D-Werte nahe den Endpunkten (0 und 1023) in hohe Drehzahlen, indem der zum PWM gesendete Wert verändert wird.

Potmot-Test in Python

Die Position des Potis (die durch den A/D- Wandler gelesen wird) bestimmt die Drehrichtung und die Drehzahl des Motors (PWM-Wert) wie folgt: Der mittlere Wert (511) bedeutet Stillstand des Motors, 1023 bedeutet maximale Drehzahl in Richtung 1, 0 bedeutet maximale Drehzahl in entgegengesetzter Richtung.

Das Programm `potmot-wp.py` benutzt `spidev`, um den A/D zu steuern und WiringPi für Python, um den Motor mit Hardware-PWM zu steuern. Im Endeffekt ist `potmot` eine vereinfachte Kombination aus den Programmen `atod.py` und `motor-wp.py`. Es wurde dahingehend vereinfacht, dass es keine Bildschirmausgaben über die A/D-Werte oder die Drehrichtung des Motors gibt.

Zuerst importiert das Programm die benötigten Module, `spidev` und `wiringpi`, anschließend werden die GPIO-Ports 17 und 18 als Ausgang bzw. PWM-Ausgang konfiguriert. Dann werden zwei Funktionen definiert: `get_adc()` liest die Spannung am Potentiometer unter Benutzung des A/D-Wandlers; `reset_ports()` sichert das Zurücksetzen der Ports beim Verlassen des Programms. Nun werden die Anfangswerte der Variablen gesetzt und die Verbindungsinformationen angezeigt. Das Programm wartet auf die Bestätigung des Benutzers, bis es weiterarbeitet.

Wenn der Benutzer die Enter-Taste drückt, wird der SPI-Port geöffnet, um die Spannung am Poti über den A/D einzulesen. Bei einem Wert über 511 wird 17 auf 0 gesetzt, was die Drehrichtung des Motors auf Richtung 1 festlegt. Andernfalls wird die Drehrichtung auf entgegengesetzt festgelegt. Dann wird der PWM-Wert, der an Port 18 gesendet werden soll, berechnet, ausgehend vom gelesenen Wert des A/D. Damit wird die Drehzahl des Motors gesteuert. Nachdem der Wert für den PWM an Port 18 gesendet wurde, wartet das Programm 0,05 Sekunden und wiederholt danach die Hauptschleife mit dem Einlesen des Wertes vom A/D- Wandler. Das passiert 600 Mal, so dass das Programm ca. 30 Sekunden läuft. Die Hauptschleife ist in einen `try: ... except`:- Block eingeschlossen, um das Schließen der Ports

beim Drücken von Ctrl-C auf der Tastatur sicherzustellen.

Vorschläge zum Experimentieren

- im Moment keine

Decoder

Der Decoder, der durch das Programm `decoder` implementiert ist, nimmt die drei Druckschalter als Eingang und schaltet eine von 8 LEDs ein, um die Zahl anzuzeigen, die durch die Stellung der drei Schalter, binär codiert, gebildet wird. Schalter S1 bildet das höchstwertige und Schalter S3 das niederwertigste Bit der Zahl (Schalter S2 das Bit in der Mitte). Für die Anzeige repräsentiert die LED D5 die Zahl 0, D6 die 1 usw. bis LED D12, die die 7 repräsentiert (Mit drei Schaltern können 8 Zustände, 0 bis 7, dargestellt werden). Erinnern Sie sich, dass die Schalter high (1) sind, wenn sie offen sind und low (0) im gedrückten Zustand, so dass LED D12 leuchtet, wenn kein Schalter gedrückt ist (das ergibt binär 111 oder dezimal 7), D6 leuchtet, wenn S1 und S2 gedrückt sind (ergibt binär 001), usw.

Es gibt einiges zu verdrahten bei diesem Beispiel, da wir alle I/O-Ports außer einem benutzen. GPIO25 bis GPIO23 lesen die Druckschalter, somit müssen GP25 mit B1, GP24 mit B2 und GP23 mit B3 verbunden werden. Die 8 am niedrigsten bezifferten GPIO-Pins werden mit den I/O-Ports 5 bis 12 benutzt, so dass GP11 mit B5, GP10 mit B6, GP9 mit B7, GP8 mit B8, GP7 mit B9, GP4 mit B10, GP1 mit B11 und GP0 mit B12 verbunden werden müssen. Zusätzlich, da wir die I/O-Ports 5 bis 12 als Ausgang benutzen, müssen Sie alle Ausgangs-Jumper für die Pufferchips U4 und U5 installieren (erinnern Sie sich, dass die Ausgangs-Jumper diejenigen oberhalb der Chips sind).

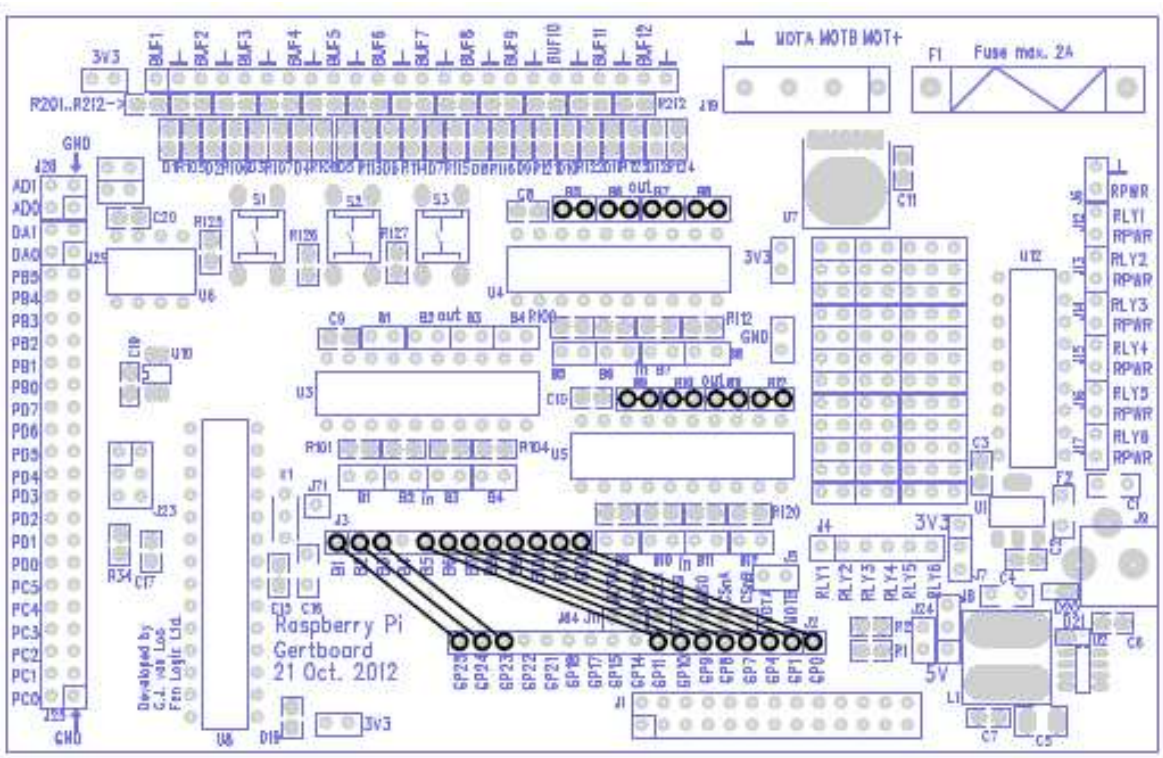


Bild 26: Verdrahtungsschema für den Decodertest.

Decoder Test in C

In der Haupt- (`main`) Routine für `decoder` starten wir wie immer mit der Anzeige der benötigten Verbindungen für das Gertboard. Danach wird `setup_io` aufgerufen, um die GPIO-Ports zu initialisieren. Anschließend rufen wir `setup_gpio` auf, um GPIO25 bis 23 für die Benutzung mit den Druckschaltern zu konfigurieren (indem sie als Eingang gesetzt werden und ein pull-up-Widerstand geschaltet wird, wie auf Seite 19 beschrieben) und um GPIO11 bis GP7, GPIO4, GPIO1 und GPIO0 als Ausgänge zu

setzen (wie auf Seite13 beschrieben). Nun wird eine Schleife abgearbeitet, in der wir den Status der Schalter einlesen und die LEDs entsprechend leuchten lassen (etwaige leuchtende LEDs müssen vorher ausgeschaltet werden). Wir schalten die LEDs mittels `GPIO_SET0` und `GPIO_CLR0` ein und aus, wie auf Seite 21 beschrieben.

Zur Zeit der Erstellung des Handbuches gibt es keinen Decodertest in Python.

ATmega Microcontroller

Das Gertboard besitzt einen Atmel AVR Microcontroller, einen 28-Pin ATmega, bezeichnet mit U8 im unteren linken Bereich des Bords. Das kann einer der folgenden sein: Tmega48A/PA, 88A/PA, 168A/PA oder 328/P in einem 28-Pin DIP-Gehäuse. Normalerweise ist der 168 oder 328 eingebaut. Der Controller hat einen 12 MHz piezokeramischen Resonator an den Pins 9 und 10. Alle Ein- und Ausgangspins sind auf den Stecker J25 auf der linken Seite des Bords verbunden. Es gibt einen separaten 6-Pin-Stecker (J23 auf der linken Seite des Bords), der benutzt werden kann, um den Controller zu programmieren.

Die PD0 / PD1-Pins (ATmega UART TX und RX) sind auf Pins verbunden, die sich angrenzend an die UART-Pins des Raspberry Pi befinden, so dass Sie nur 2 Jumper stecken müssen, um die beiden ICs zu verbinden.
(Anm. d. Ü.:
UART = universal asynchronous receiver / transmitter – ein IC für die asynchrone serielle Kommunikation zwischen Geräten.
TX = transmit – Sendeleitung
RX = receive – Empfangsleitung)

Beachten Sie, dass der ATmega auf dem Gertboard mit 3,3Volt arbeitet. Das ist im Gegensatz zum ‘Arduino’-System, das mit 5V arbeitet (das ist der Grund, warum der Controller keinen 16MHz-Takt hat. Genau genommen liegt bei 3,3V die maximale Taktfrequenz unter 12MHz).
(Anm. d. Ü.: Mehr zum Arduino-Prinzip z.B. unter
https://wiki.attraktor.org/images/f/f1/ARDUINO_Basics_Teil_2.pdf

Warnung: viele der Arduino- Beispiele gehen von +5V für die Schaltung aus. Da wir nur mit 3,3V arbeiten, müssen Sie 3,3V statt 5V benutzen, wenn irgendwo 5V angegeben sind. Wenn Sie 5 V benutzen, riskieren Sie die Zerstörung des Chips.

Der ATmega und die mit ihm verbundenen Steckerleisten sind im Schaltplan auf Seite A-6 abgebildet.

Programmierung des ATmega

Die Programmierung des ATmega-Microcontrollers ist einfach, wenn Sie einmal die Infrastruktur geschaffen haben, aber es erfordert einiges an Software, die auf Ihrem Raspberry Pi installiert werden muss. Wir sind Gordon Henderson von Drogon Systems sehr dankbar für die Ausarbeitung dessen, was notwendig war und die Lieferung der angepassten (customized) Software. Sein System erlaubt es Ihnen, die Arduino-IDE (Integrated Development Environment – integrierte Entwicklungsumgebung) auf dem Raspberry Pi zu benutzen, um Programme für den ATmega-Chip zu entwickeln und auf das Gertboard hochzuladen. Alle benötigte Software, zusammen mit der Anleitung, ist verfügbar unter

https://projects.drogon.net/raspberry-pi/gertboard/

Für den Rest dieses Abschnitts setzen wir voraus, dass Sie die Arduino-IDE heruntergeladen und erfolgreich installiert und konfiguriert haben, so wie es auf Gordons Website beschrieben ist, und wir fahren von hier aus fort.

Um mit dem ATmega-Chip zu beginnen, starten Sie die Arduino-IDE. Das sollte leicht sein: wenn die Installation des Arduino- Pakets erfolgreich war, haben Sie einen neuen Punkt „Arduino IDE” in Ihrem Startmenü, unter “Electronics”. Die genaue Version der IDE hängt von Ihrem Betriebssystem ab. Die

überwiegende Mehrheit der Raspberry Pi-Benutzer verwendet Raspbian, das auf Debian wheezy basiert, und wir gehen von jetzt ab davon aus, dass Sie das verwenden. Die Versionsnummer wird im Titelbalken angezeigt, für wheezy ist sie 1.0.1. Als erstes müssen Sie die IDE konfigurieren, damit sie mit dem Gertboard arbeitet. Gehen Sie zum Menüpunkt Tools > Board und wählen Sie die Gertboard- Option mit dem Chip, den Sie benutzen (Es gibt die Auswahl für den ATmega168 und den ATmega328, die am meisten verwendeten auf dem Gertboard). Gehen Sie anschließend zum Menüpunkt Tools > Programmer und wählen Sie “Raspberry Pi GPIO”.

Arduino-Pins auf dem Gertboard

Alle Ein- und Ausgangspins des ATmega-Chips sind mit dem Stecker J25 links auf dem Bord verbunden. Sie sind mit PC*n*, PD*n* und PB*n* bezeichnet, wobei *n* die Nummer ist. Die Bezeichnung stimmt mit der Pinbelegung des ATmega168/328-Chips überein. Die Arduino-Welt bezieht sich jedoch nicht direkt auf die Pinnummern der Chips. Stattdessen wird eine abstrakte Bezeichnung der digitalen- und analogen Pinnummern verwendet, die unabhängig vom physischen Gerät ist. Das erlaubt es, Code, der für ein bestimmtes Arduino-Bord geschrieben wurde, einfach für ein anderes Arduino-Bord zu verwenden, das einen Chip mit einer anderen Pinbelegung hat. Darum müssen Sie wissen, wie die Arduino- Pinnummern mit den Bezeichnern auf dem Gertboard übereinstimmen, wenn Sie das Gertboard mit der Arduino-IDE benutzen. Die Tabelle unten zeigt diese Zuordnung (“GB” bedeutet Gertboard).

Arduino-Pin	GB-Pin	Arduino-Pin	GB-Pin	Arduino-Pin	GB-Pin
0	PD0	7	PD7	A0	PC0
1	PD1	8	PB0	A1	PC1
2	PD2	9	PB1	A2	PC2
3	PD3	10	PB2	A3	PC3
4	PD4	11	PB3	A4	PC4
5	PD5	12	PB4	A5	PC5
6	PD6	13	PB5		

Tabelle 4: Die Zuordnung zwischen Arduino- Pinnummern und Pins auf dem Gertboard.

In Programmen werden digitale Pins nur mit einer Nummer bezeichnet. Zum Beispiel

```
digitalWrite(13, HIGH);
```

setzt Pin 13 (PB5 auf dem Gertboard) auf logisch 1. (In der Arduino-Welt ist LOW logisch 0 und HIGH logisch 1)

Die analogen Pins werden mit A0 bis A5 bezeichnet. Um also das analoge Pin 0 (PC0 auf dem Gertboard) zu lesen, muss man das folgende Kommando benutzen

```
value = analogRead(A0);
```

Ein paar Programme (sketches), mit denen Sie starten können

Ein “sketch” ist die Bezeichnung, die Arduino für ein Programm benutzt. Es ist der Code, der auf das Arduino-Bord hochgeladen und dort ausgeführt wird (oder, wie in unserem Fall, der ATmega-Microcontroller auf dem Gertboard). Werfen wir einen Blick auf ein einfaches Programm, `Blink`, das eine LED ein- und ausschaltet. Es ist in der Arduino-IDE über den Menüpunkt Datei > Examples > Basics verfügbar. Wenn Sie es auswählen, erscheint ein neues Fenster mit dem `Blink`- Code. Es gibt nur zwei Funktionen in dem Programm, `setup` und `loop`. Diese werden für alle Arduino-Programme benötigt: `setup` wird einmal gleich zu Beginn ausgeführt, und `loop` wird wiederholt ausgeführt, solange der Chip mit Spannung versorgt wird. Beachten Sie, dass Sie keinen Code schreiben müssen, um diese beiden Funktionen aufzurufen; sie werden automatisch bei der Übersetzung und dem Hochladen (upload) des Programms hinzugefügt. Die Programmiersprache, die für diese Programme benutzt wird, basiert auf

C, so dass sie Ihnen vertraut vorkommen sollte, wenn Sie bereits die C-Testprogramme für das Gertboard angeschaut haben.

Hochladen von Programmen unter Benutzung des SPI Busses

Um ein Programm (“sketch”) auf dem ATmega-Chip des Gertboards laufen zu lassen, muss es irgendwie auf den Chip übertragen werden (das wird als Hochladen – engl. upload- des Programms bezeichnet). Es gibt verschiedene Methoden, ATmega-Chips zu programmieren, aber wir wollen den SPI-Bus, der auf den GPIO-Pins 8 bis 11 verfügbar ist, benutzen. Das ist möglich, weil die Arduino IDE das spezielle Programm zum Herunter- / Hochladen (downloader / uploader) `avrdude` benutzt, das Sie von projects.drogon.net bezogen haben. Um dies zu benutzen, müssen Sie die GPIO-Pins, die für den SPI-Bus benutzt werden, mit dem 6-Pin-Stecker J23 verbinden, wie es im Schema unten gezeigt wird. Hier verbinden Sie einfach die SPI-Pins des GPIO mit den entsprechenden SPI-Pins auf dem Stecker. Die Anordnung der Pins auf J23 ist im Bild auf Seite A-6 zu sehen.

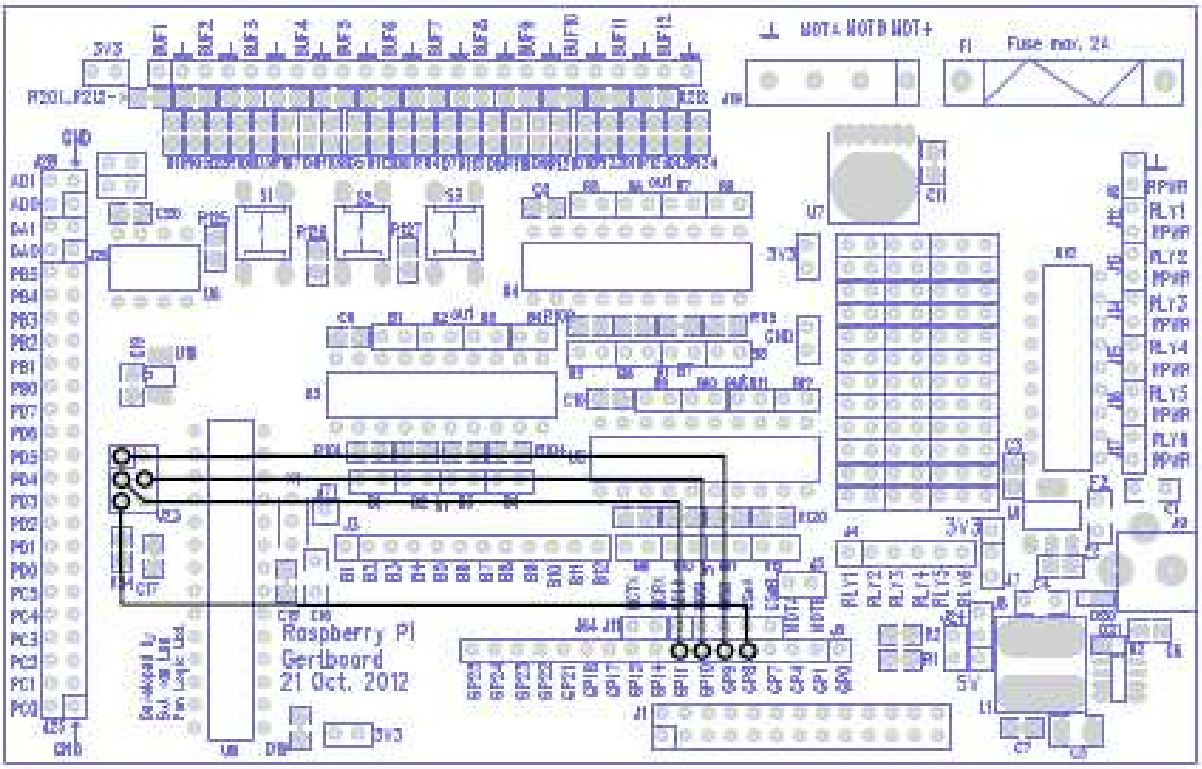


Bild 27: Verdrahtungsschema für das Hochladen von Programmen auf den ATmega-Mikroprozessor.

Um Ihr Programm auf den Chip in der Arduino-IDE hochzuladen, wählen Sie File > Upload Using Programmer. Es dauert einen Moment, um das Programm zu übersetzen (compile) und hochzuladen (upload), und dann wird Ihr Programm auf dem Microcontroller ausgeführt.

Programm Blink

Das Programm `Blink` befindet sich unter dem Menüpunkt File > Examples > Basics. Wählen Sie es aus und laden Sie es auf den ATmega-Chip, indem Sie wie oben beschrieben vorgehen. Das Programm wird jetzt ausgeführt, aber es passiert noch nichts! Auf den meisten Arduino- Bords hat das Pin 13 (das digitale Pin, das in diesem Programm benutzt wird), eine mit ihm verbundene LED, aber nicht das Gertboard. Sie müssen die LED selbst anschließen. Wenn wir in die Tabelle 4 oben schauen, sehen wir, dass das digitale Pin 13 auf dem Gertboard mit PB5 bezeichnet ist, so dass Sie PB5 mit einem der I/O-Ports verbinden müssen. Im Abschnitt über gepufferten I/O, LEDs und Druckschalter haben wir erklärt, dass Sie eine LED benutzen können, um den Zustand eines Signals anzuzeigen, indem Sie das Signal mit einem der Pins BUF1 bis BUF12 auf dem (unbezeichneten) einreihigen Stecker oben auf dem Bord verbinden. Wenn Sie also PB5 mit BUF1 verbinden, wie es unten gezeigt wird, fängt die erste LED an zu blinken.

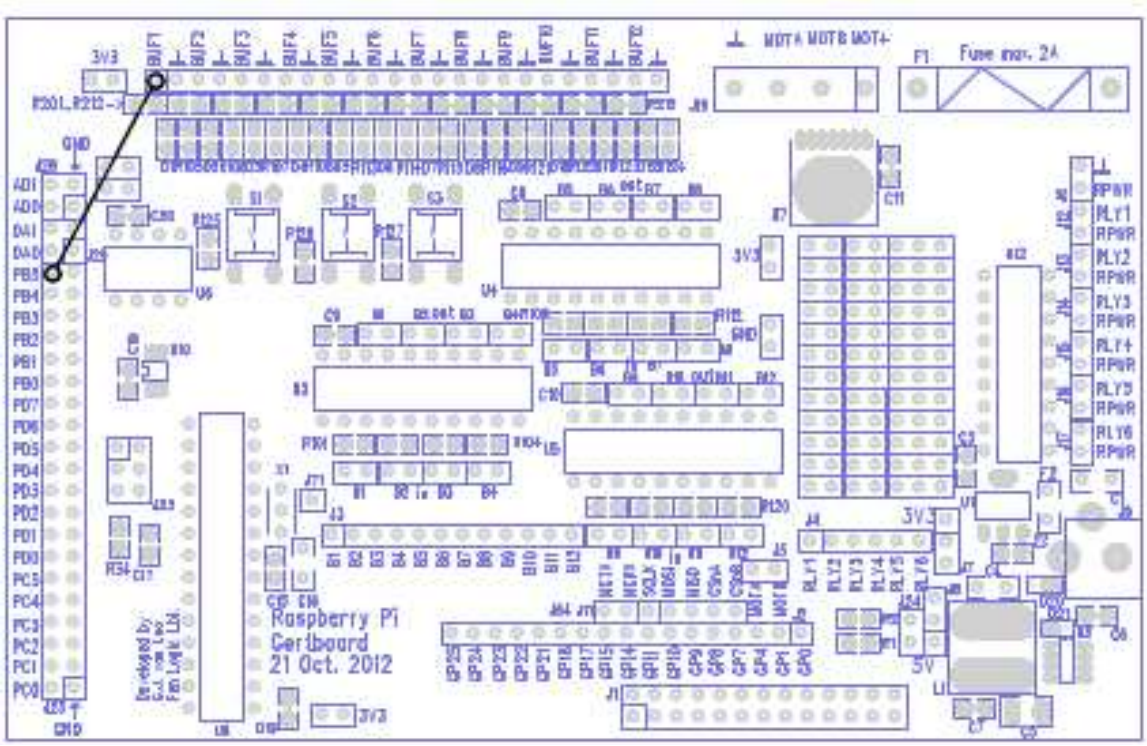


Bild 28: Verdrahtungsschema für das Programm Blink.

Beachten Sie, dass in diesem Schema die Verbindungen zu den SPI-Pins nicht gezeigt werden. Wenn Sie den Code einmal hochgeladen haben, benötigen Sie diese Verbindungen nicht länger und können die Kabel entfernen. Auf der anderen Seite können Sie sie auch stecken lassen, wenn Sie später ein anderes Programm hochladen möchten.

Programm Button (Schalter)

Sehen wir uns ein anderes recht einfaches Programm mit dem Namen Button an, das sich unter File > Examples > Digital befindet, sowohl in 0018 als auch in 1.0.1. Der Kommentar am Anfang des Programms lautet

The circuit: (die Schaltung:)

- LED attached from pin 13 to ground (LED zwischen Pin 13 und Masse)
- pushbutton attached to pin 2 from +5V (Druckschalter zwischen Pin 2 und +5V)
- 10K resistor attached to pin 2 from ground (10K Widerstand zwischen Pin 2 und Masse)

Das Programm `Blink` läuft, die LED ist verbunden, aber was ist mit dem Schalter? Da wie oben erwähnt, der ATmega-Chip auf dem Gertboard mit 3,3V läuft, müssen wir die 5V durch 3,3V ersetzen. Es wird empfohlen, eine Schaltung zu benutzen (wie unten zu sehen), wobei der Wert an Pin 2 logisch 0 ist, wenn der Schalter nicht gedrückt ist (wegen des 10K pull-down-Widerstands) und logisch 1, wenn der Schalter gedrückt ist.

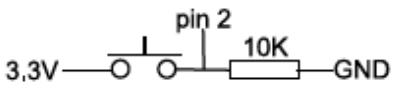


Bild 29: Empfohlene Schaltung für das Programm Button.

Die Schalter auf dem Gertboard werden jedoch wie folgt benutzt:

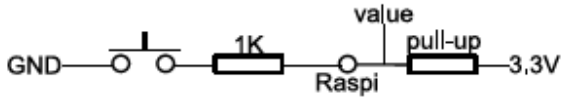


Bild 30: Schaltung, wie sie auf dem Gertboard benutzt wird: ein zusätzlicher Widerstand von 1K, um den Eingang des BCM2835 zu schützen.

Der Widerstand von 1K zwischen dem Schalter und dem 'Raspi'-Punkt dient dem Schutz des BCM2835 (der Prozessor auf dem Raspberry Pi), wenn Sie das GPIO-Pin, das mit 'Raspi' verbunden ist, aus Versehen auf Ausgang anstatt auf Eingang setzen. Die Schaltung rechts vom 'Raspi'-Punkt befindet sich auf dem Raspberry Pi: um den Schalter zu benutzen, schalten wir einen pull-up-Widerstand (gezeigt als Widerstand in der Schaltung oben) auf das Pin, so dass der gelesene Wert logisch 1 ist, wenn der Schalter nicht gedrückt ist (siehe Seite 19 für weitere Informationen über pull-up-Widerstände). Die Schalter auf dem Gertboard werden direkt mit Masse verbunden, so dass es nicht möglich ist, logisch 1 zu lesen, wenn sie gedrückt sind. Wenn Sie einen Schalter auf dem Gertboard mit einem Arduino-Programm so nutzen möchten, dass logisch 1 gelesen wird, wenn der Schalter gedrückt ist, ist es am besten, das Programm zu ändern und den gelesenen Wert zu invertieren. Für den pull-up-Widerstand können wir Vorteile aus den pull-up-Widerständen im ATmega-Chip ziehen. Dafür suchen Sie die folgenden Zeilen im Programm auf:

// initialize the pushbutton pin as an input: (Druckschalter-Pins als Eingang setzen)
```
pinMode(buttonPin, INPUT);
```

und fügen Sie die folgenden zwei Zeilen danach ein:

// set pullup on pushbutton pin (setze einen pull-up-Widerstand an das Schalter-Pin)
```
digitalWrite(buttonPin, HIGH);
```

Um den gelesenen Wert vom Schalter zu invertieren, gehen Sie zu der Zeile:

```
buttonSate = digitalRead(buttonPin);
```

und fügen Sie ein Ausrufezeichen ! (der Negations-Operator in C) wie folgt ein:

```
buttonSate = !digitalRead(buttonPin);
```

Laden Sie nun das geänderte Programm hoch, wie beim Programm `Blink` beschrieben. Wir müssen noch das Arduino- digitale Pin 2 (PD2 auf dem Gertboard, wie Sie aus der Tabelle entnehmen können) mit einem Schalter verbinden, nehmen wir Schalter 3. Das 'Raspi'-Pin in der Schaltung oben, an dem wir den Wert einlesen, ist auf dem Stecker J3.

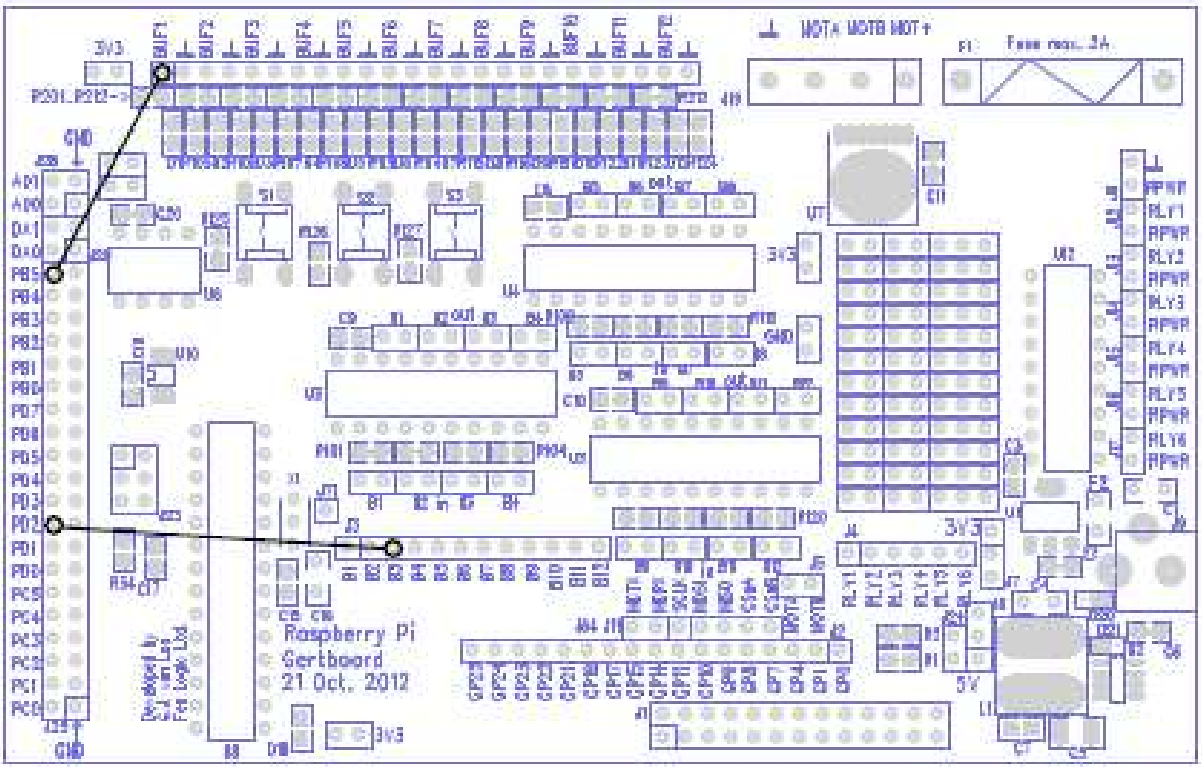


Bild 31: Verdrahtungsschema für das Programm Button.

Wenn Sie das erledigt haben, leuchtet die erste LED, wenn der dritte Schalter gedrückt wird, und ist aus, wenn der Schalter nicht gedrückt ist.

Programm AnalogInput

Versuchen wir nun, ein analoges Pin zu benutzen. Öffnen Sie das Programm `AnalogInput` unter File > Examples > Analog. Dieses Programm liest einen Wert vom analogen Eingang 0 (der bereits vom internen A/D- Wandler in einen Wert zwischen 0 und 1023 konvertiert wurde), und benutzt anschließend diesen Wert als Pause zwischen dem Ein- und Ausschalten einer LED. Somit blinkt die LED umso schneller, desto kleiner die Spannung am analogen Eingang ist. Für dieses Beispiel benötigen Sie ein Potentiometer. Das aus dem Test für den A/D-Wandler funktioniert hier ebenfalls gut. Die Kommentare im Programm `AnalogInput` sagen aus, den Schleifkontakt des Potis mit dem analogen Pin 0 (PC0 auf dem Gertboard) und die äußeren Anschlüsse mit +5V und Masse zu verbinden. Erinnern Sie sich: Sie müssen 3,3V anstatt 5V benutzen, da wir den Chip mit 3,3V betreiben. Das Schema unten zeigt, wie das Gertboard verdrahtet werden muss, um das Programm nach dem Hochladen zum Laufen zu bringen.

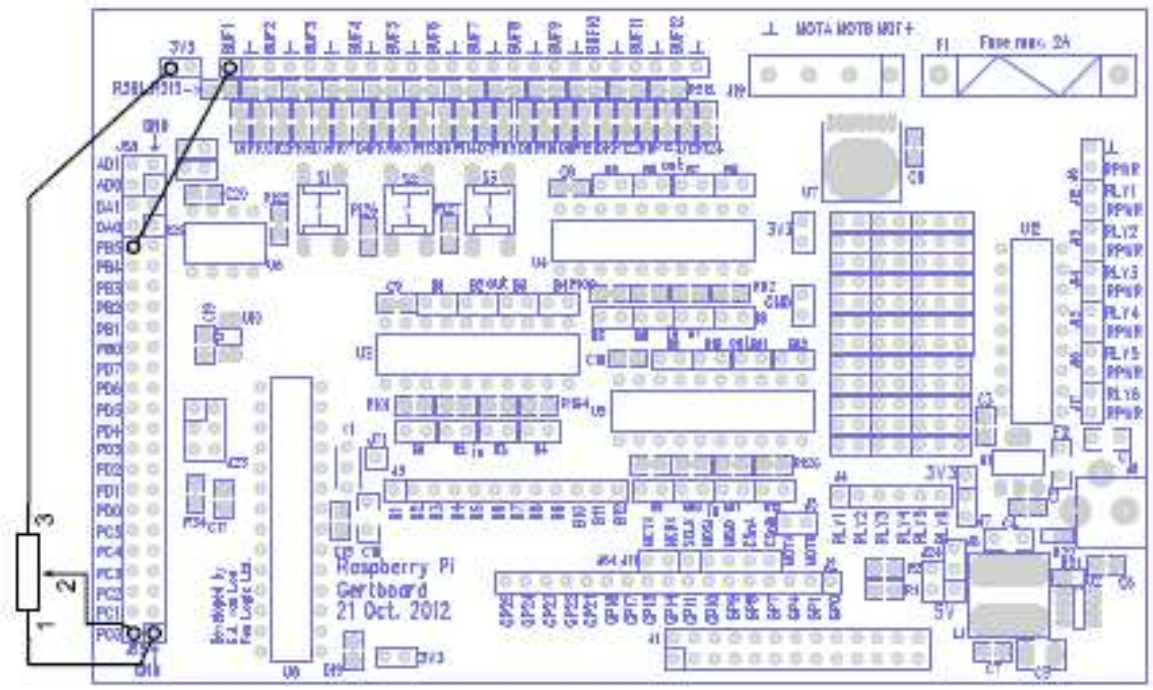


Bild 32: Verdrahtungsplan für das Programm AnalogInput.

Programm AnalogReadSerial unter Benutzung von Minicom

Einige der Arduino-Programme lesen oder schreiben Daten über den seriellen Port bzw. UART. Ein Beispiel ist `AnalogReadSerial`, das sich im Menü File > Examples > Basics befindet. Dieses Programm setzt die Baudrate auf 9600, liest anschließend in einer Schleife einen Wert vom analogen Eingang an Pin 0 and sendet diesen Wert zum seriellen Port (auch als UART bezeichnet). Der eingelesene Wert liegt zwischen 0 und 1023; 0 bedeutet eine Eingangsspannung von 0V, und 1023 bedeutet die Eingangsspannung ist gleich der Versorgungsspannung (3,3V für das Gertboard).

Um das Gertboard für diesen Test vorzubereiten, müssen Sie das Poti am analogen Eingang 0 wie beim Programm `AnalogInput` anschließen. Zusätzlich muss der UART des ATmega-Chips mit dem Raspberry Pi verbunden werden. Das digitale Pin 0 (PD0 auf dem Gertboard) ist RX (receive- empfangen) und das digitale Pin 1 (PD1 auf dem Gertboard) ist TX (transmit - senden). Diese Signale sind auch mit den Pins mit der Bezeichnung MCTX und MCRX genau oberhalb der Pins GP15 und GP14 auf dem Stecker J2 auf dem Gertboard verbunden. Somit können Sie zwei Jumper benutzen, um TX des ATmega-Chips mit GP15 und RX mit GP14 zu verbinden, wie in Bild 33 zu sehen.

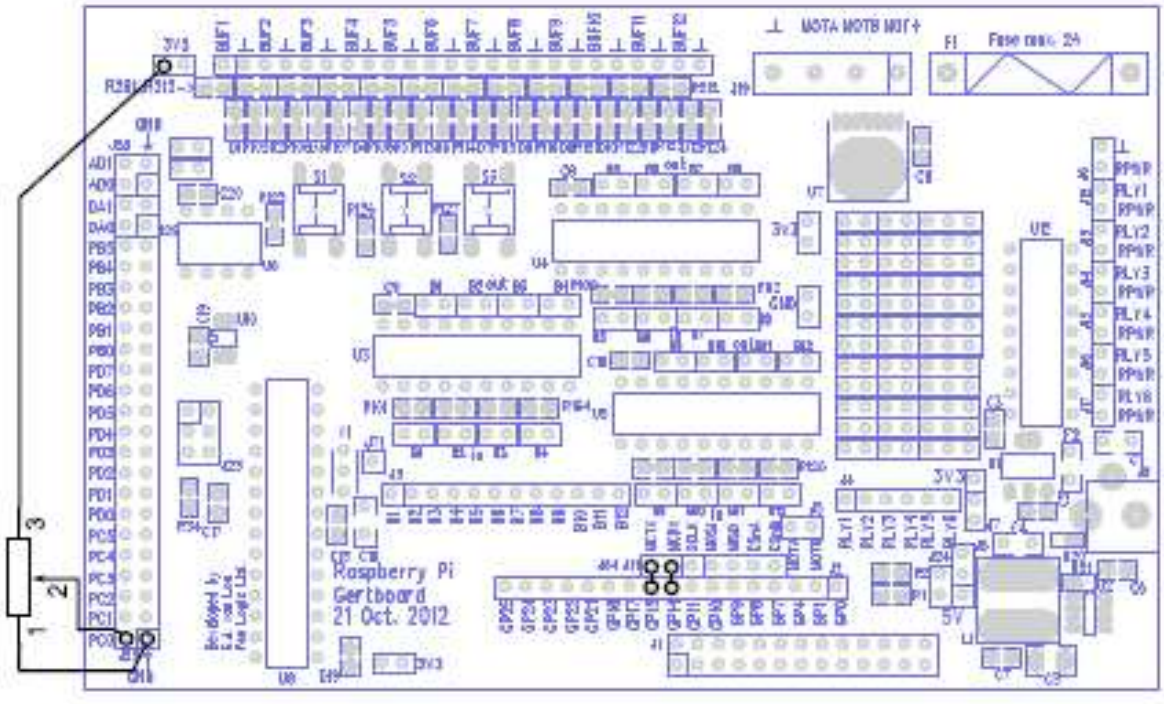


Bild 33: Verdrahtungsplan für das Programm AnalogReadSerial.

GPIO14 und GPIO15 sind die Pins, die der Raspberry Pi für den seriellen Port des UART benutzt. Wenn Sie die Tabelle für die alternativen Funktionen betrachten (Tabelle 1, Seite 11), sehen Sie, dass GPIO14 als TX und GPIO15 als RX aufgelistet sind. Das ist kein Fehler! Diese Vertauschung ist notwendig, weil die Daten, die vom ATmega gesendet werden, vom Raspberry Pi empfangen werden, und umgekehrt.

Nun, wie bringen wir den Raspberry Pi dazu, Daten zu lesen und anzuzeigen, die der ATmega über den seriellen Port gesendet hat? Es gibt eine Schaltfläche (button) namens „Serial Monitor" auf der Werkzeugleiste (toolbar) der Arduino-IDE, aber die funktioniert nicht beim Raspberry Pi. Sie geht von einer USB-Verbindung zu einem Arduino-Bord aus, nicht von einer Verbindung zu einem Gertboard über GPIO. Der einfachste Weg, die Daten zu erhalten, ist die Verwendung des Programms `minicom`. Sie können es einfach installieren, indem Sie auf dem Terminal das folgende Kommando eingeben:

```
sudo apt-get install minicom
```

Sie können Menüs benutzen, um `minicom` zu konfigurieren (durch den Befehl `minicom -s`). Alternativ dazu gibt es eine Datei, die zusammen mit der Gertboard-Software geliefert wird: `minirc.ama0` mit den Einstellungen, die für das Lesen der GPIO-UART-Pins bei 9600 Baud benötigt werden. Kopieren Sie diese Datei (die von Gordon Henderson bereitgestellt wurde) nach `/etc/minicom/` (Sie müssen wahrscheinlich `sudo` dazu verwenden) und rufen Sie `minicom` auf:

```
sudo minicom ama0
```

Nun sollten Sie, wenn Sie das Programm auf den ATmega-Chip hochgeladen haben, den Wert vom Potentiometer auf dem `minicom`- Monitor angezeigt bekommen.

Programm LEDmeter

Dieses Beispiel haben wir speziell für das Gertboard entwickelt, basierend auf dem Programm `AnalogInput`, das oben beschrieben wurde. Im Verzeichnis `gertboard_sw` existiert neben den C-Dateien eine weitere Datei mit dem Namen `LEDmeter.ino`. Das ist ein Arduino-Programm, das alle 12 LEDs benutzt, um einen Balken darzustellen, der die Spannung am analogen Eingang, z. B. an einem

Potentiometer, anzeigt. Zuerst müssen Sie dieses Programm an den richtigen Ort kopieren. Dazu sollten Sie ein Verzeichnis mit dem Namen `sketchbook` in Ihrem Home-Verzeichnis haben. Legen Sie darin ein Unterverzeichnis mit dem Namen `LEDmeter` an und kopieren Sie `LEDmeter.ino` in dieses Verzeichnis. Sie können das mit folgenden Befehlen in ihrem Verzeichnis `gertboard_sw` ausführen:

```
mkdir ~/sketchbook/LEDmeter
cp LEDmeter.ino ~/sketchbook/LEDmeter
```

Nun befindet sich das Programm im Menü File > Sketchbook in der Arduino-IDE. Sie können es öffnen, um den Code zu analysieren. Für dieses Beispiel haben wir zwei extra Funktionen erstellt, `turn_on_leds` und `turn_off_leds`, um zu demonstrieren, dass Sie gewöhnlichen C-Code in den Programmen (sketches) benutzen können, einschließlich Funktionen. Die Pinnummern werden in einem Array gespeichert, um das Ein- und Ausschalten von LEDs zu vereinfachen. Wenn Sie LED 4 ansprechen wollen, benutzen Sie Arduino-Pin `led_pins[4]`. Um das so machen zu können, haben wir eine 0 in das erste Array-Element geschrieben, weil es keine LED 0 gibt. Eine Eigenschaft von C, mit der Sie möglicherweise noch nicht vertraut sind, ist die Benutzung von `static` in der Definition:

```
static int old_max_led = 0;
```

Das Schlüsselwort `static` bedeutet, dass der Wert der Variablen (`old_max_led`) zwischen Funktionsaufrufen immer gleich bleibt. Wenn wir also am Ende eines Aufrufs von `loop()` einen Wert zuweisen, hat `old_max_led` immer noch denselben Wert, wenn das nächste Mal `loop()` aufgerufen wird. Der Anfangswert, 0, wird nur einmal zugewiesen, bevor `loop()` zum ersten Mal aufgerufen wird, nicht jedes Mal, wenn `loop()` aufgerufen wird. Das Verdrahtungsschema für `LEDmeter` ist unten zu sehen.

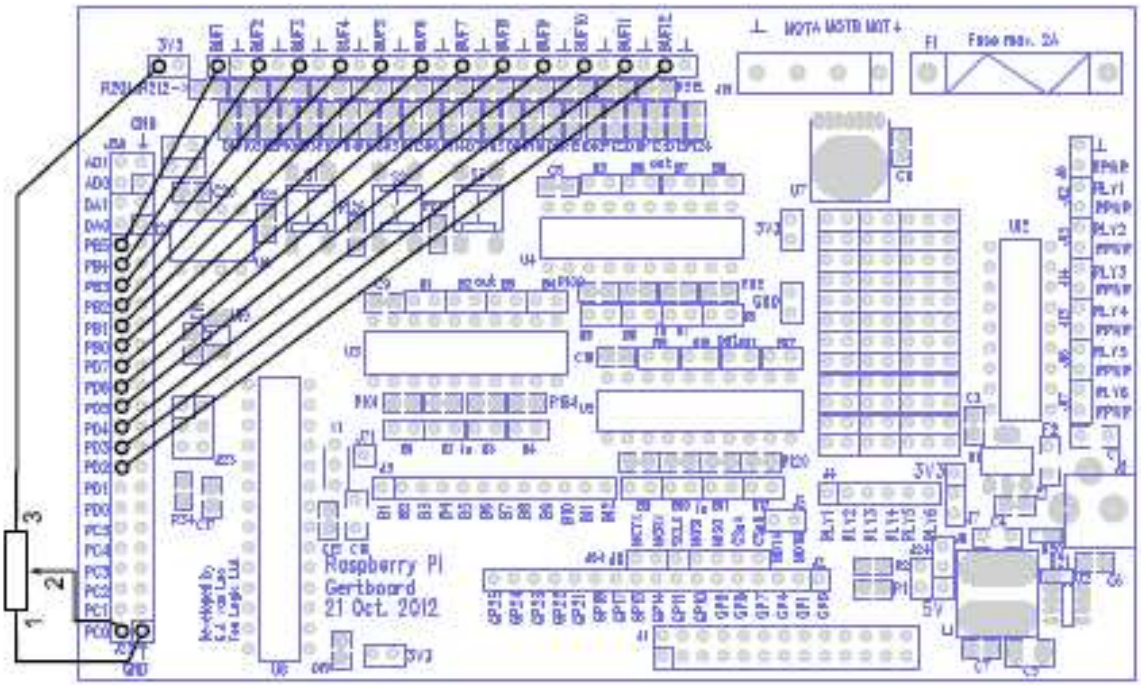


Bild 34: Verdrahtungsschema für das Programm LEDmeter

Hochladen ist der übliche Weg, und die LEDs zeigen die Position des Potentiometers.

Wie geht es weiter?

Diese Beispiele haben die Oberfläche der faszinierenden Welt von Arduino nur angekratzt. Sie erhalten unter
http://arduino.cc/en/Tutorial/HomePage

viel, viel mehr Informationen.

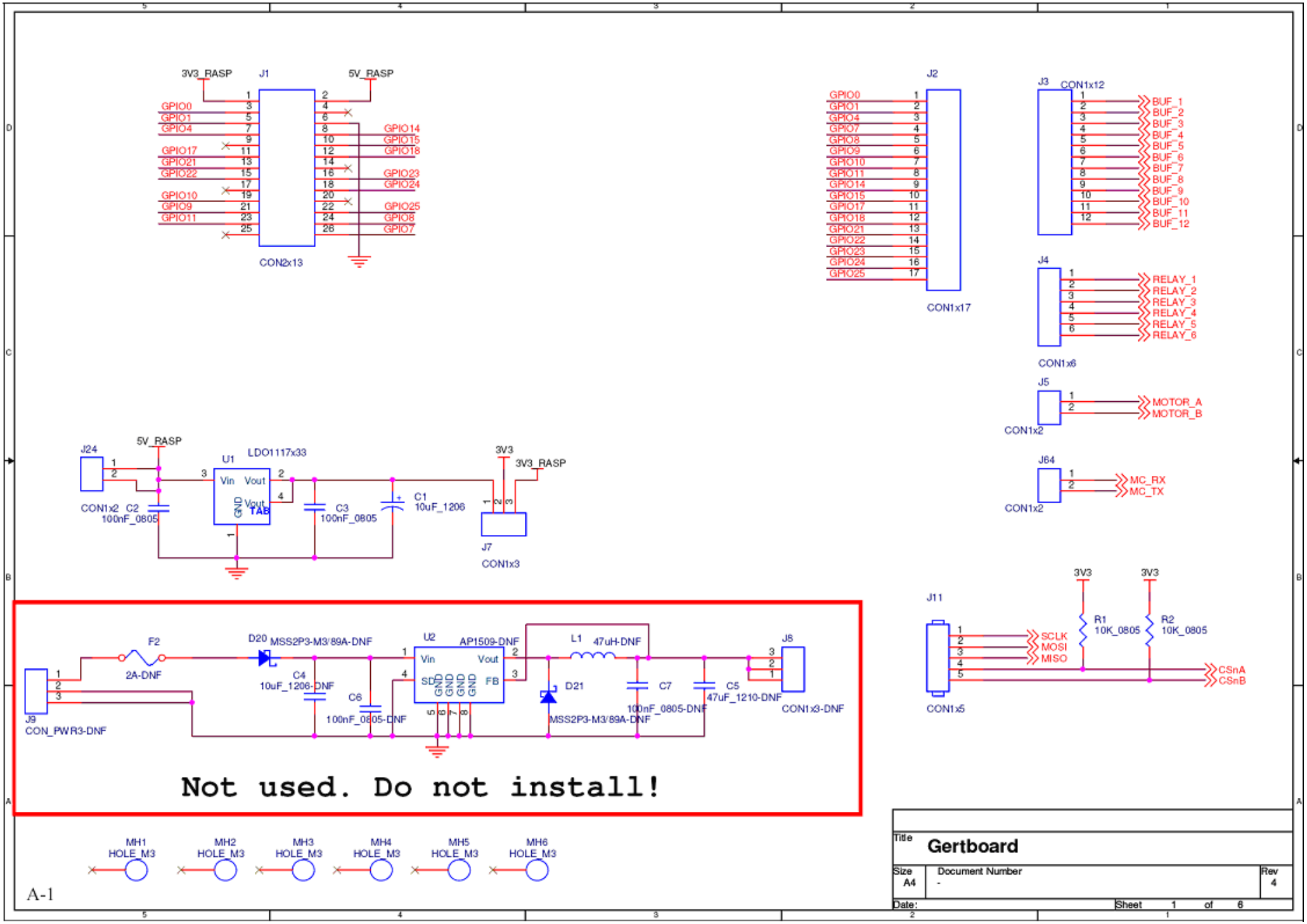
Weitere Informationen

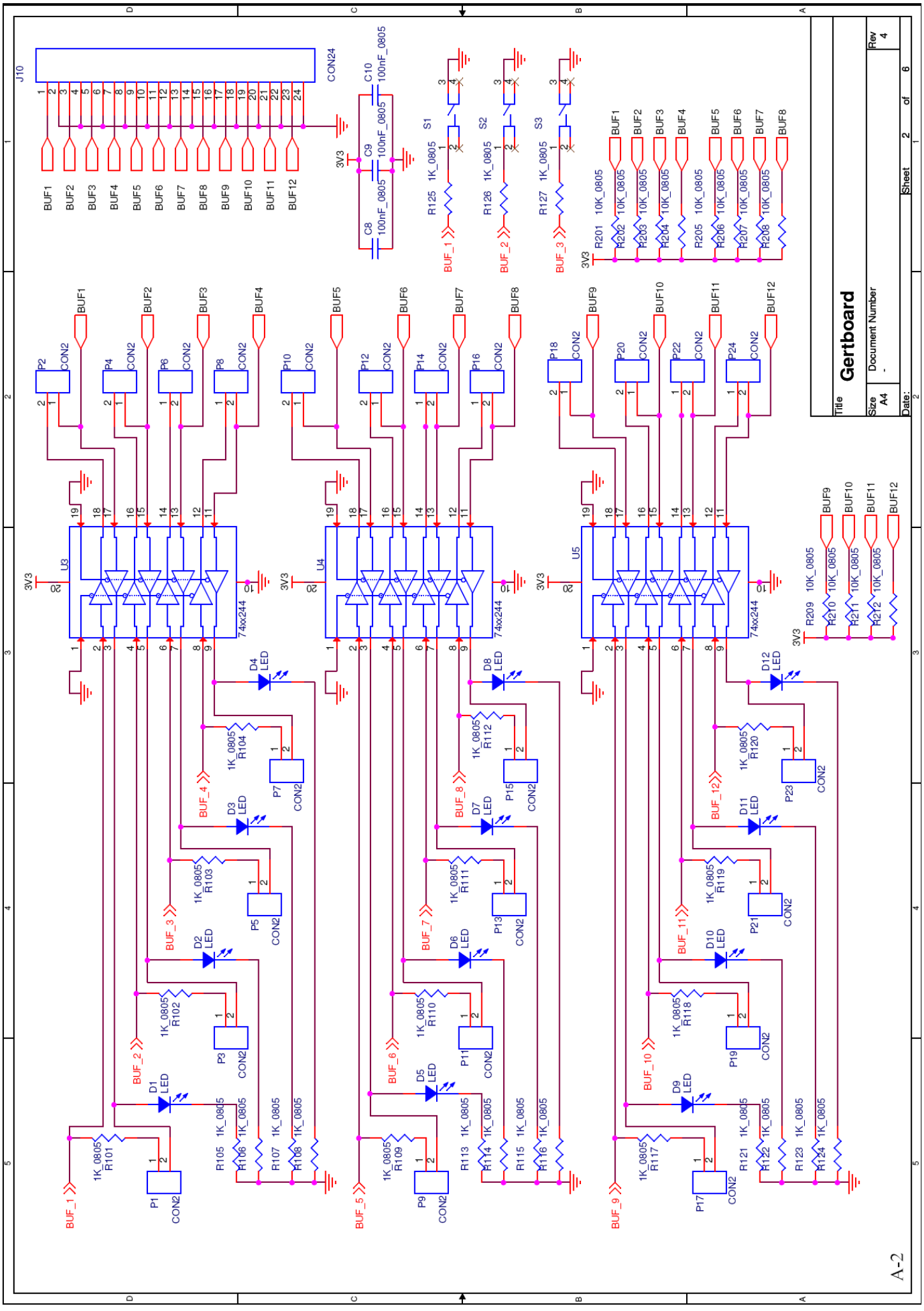
Weitere Informationen über den Raspberry Pi und seine GPIO-Ports sowie das Datenblatt für den Prozessor können Sie hier erhalten:

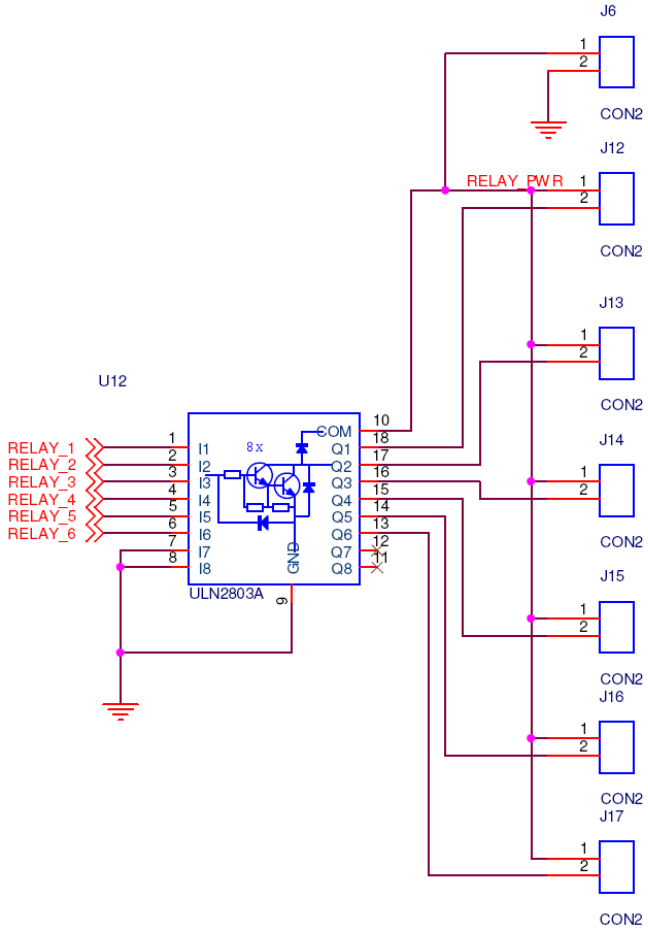
http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

Anhang A: Schaltpläne

Wir haben die Schaltpläne für das Gertboard auf den folgenden Seiten veröffentlicht. Sie sind mit A-1, A-2 usw. nummeriert. Die Seitennummer befindet sich links unten auf jeder Seite.

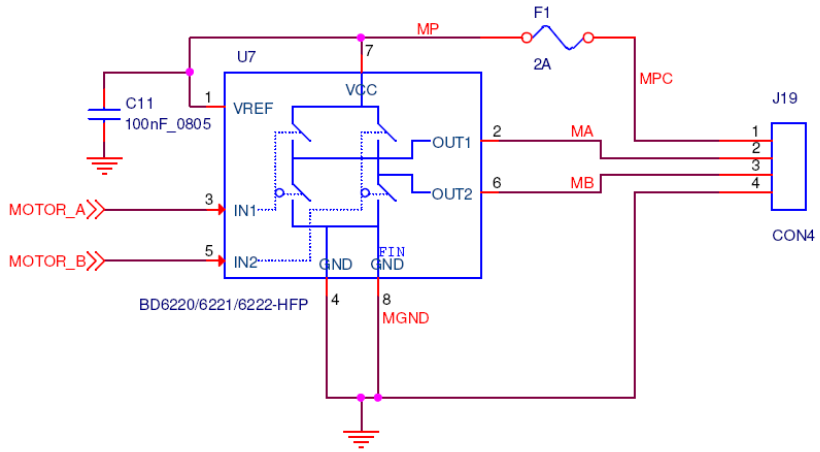

3V3_RASP
J1
5V_RASP
GPIO0
GPIO1
GPIO4
GPIO17
GPIO21
GPIO22
GPIO10
GPIO9
GPIO11
GPIO14
GPIO15
GPIO18
GPIO23
GPIO24
GPIO25
GPIO8
GPIO7
CON2x13
J2
CON1x17
J3
CON1x12
BUF_1
BUF_2
BUF_3
BUF_4
BUF_5
BUF_6
BUF_7
BUF_8
BUF_9
BUF_10
BUF_11
BUF_12
J4
RELAY_1
RELAY_2
RELAY_3
RELAY_4
RELAY_5
RELAY_6
CON1x6
J5
MOTOR_A
MOTOR_B
CON1x2
J64
MC_RX
MC_TX
CON1x2
J24
5V_RASP
U1
LDO1117x33
Vin
Vout
GND
TAB
CON1x2
C2
100nF_0805
C3
100nF_0805
C1
10uF_1206
3V3
3V3_RASP
J7
CON1x3
F2
2A-DNF
D20
MSS2P3-M3/89A-DNF
U2
AP1509-DNF
SDD
GND
FB
L1
47uH-DNF
D21
MSS2P3-M3/89A-DNF
C4
10uF_1206-DNF
C6
100nF_0805-DNF
C7
100nF_0805-DNF
C5
47uF_1210-DNF
J8
CON1x3-DNF
J9
CON_PWR3-DNF
Not used. Do not install!
J11
3V3
3V3
R1
10K_0805
R2
10K_0805
SCLK
MOSI
MISO
CSnA
CSnB
CON1x5
MH1
HOLE_M3
MH2
HOLE_M3
MH3
HOLE_M3
MH4
HOLE_M3
MH5
HOLE_M3
MH6
HOLE_M3
A-1
Title
Gertboard
Size
A4
Document Number
-
Rev
4
Date:
Sheet
1
of
6


Gertboard
Title
Size A4
Document Number -
Rev 4
Sheet 2 of 6
Date:
J10
CON24
BUF1
BUF2
BUF3
BUF4
BUF5
BUF6
BUF7
BUF8
BUF9
BUF10
BUF11
BUF12
C8 100nF_0805
C9 100nF_0805
C10 100nF_0805
3V3
S1
S2
S3
R125 1K_0805
R126 1K_0805
R127 1K_0805
BUF_1
BUF_2
BUF_3
R201 10K_0805
R202 10K_0805
R203 10K_0805
R204 10K_0805
R205 10K_0805
R206 10K_0805
R207 10K_0805
R208 10K_0805
U3
U4
U5
74x244
P2 CON2
P4 CON2
P6 CON2
P8 CON2
P10 CON2
P12 CON2
P14 CON2
P16 CON2
P18 CON2
P20 CON2
P22 CON2
P24 CON2
D1 LED
D2 LED
D3 LED
D4 LED
D5 LED
D6 LED
D7 LED
D8 LED
D9 LED
D10 LED
D11 LED
D12 LED
R101 1K_0805
R102 1K_0805
R103 1K_0805
R104 1K_0805
R105 1K_0805
R106 1K_0805
R107 1K_0805
R108 1K_0805
R109 1K_0805
R110 1K_0805
R111 1K_0805
R112 1K_0805
R113 1K_0805
R114 1K_0805
R115 1K_0805
R116 1K_0805
R117 1K_0805
R118 1K_0805
R119 1K_0805
R120 1K_0805
R121 1K_0805
R122 1K_0805
R123 1K_0805
R124 1K_0805
R209 10K_0805
R210 10K_0805
R211 10K_0805
R212 10K_0805
P1 CON2
P3 CON2
P5 CON2
P7 CON2
P9 CON2
P11 CON2
P13 CON2
P15 CON2
P17 CON2
P19 CON2
P21 CON2
P23 CON2
A-2



A-3

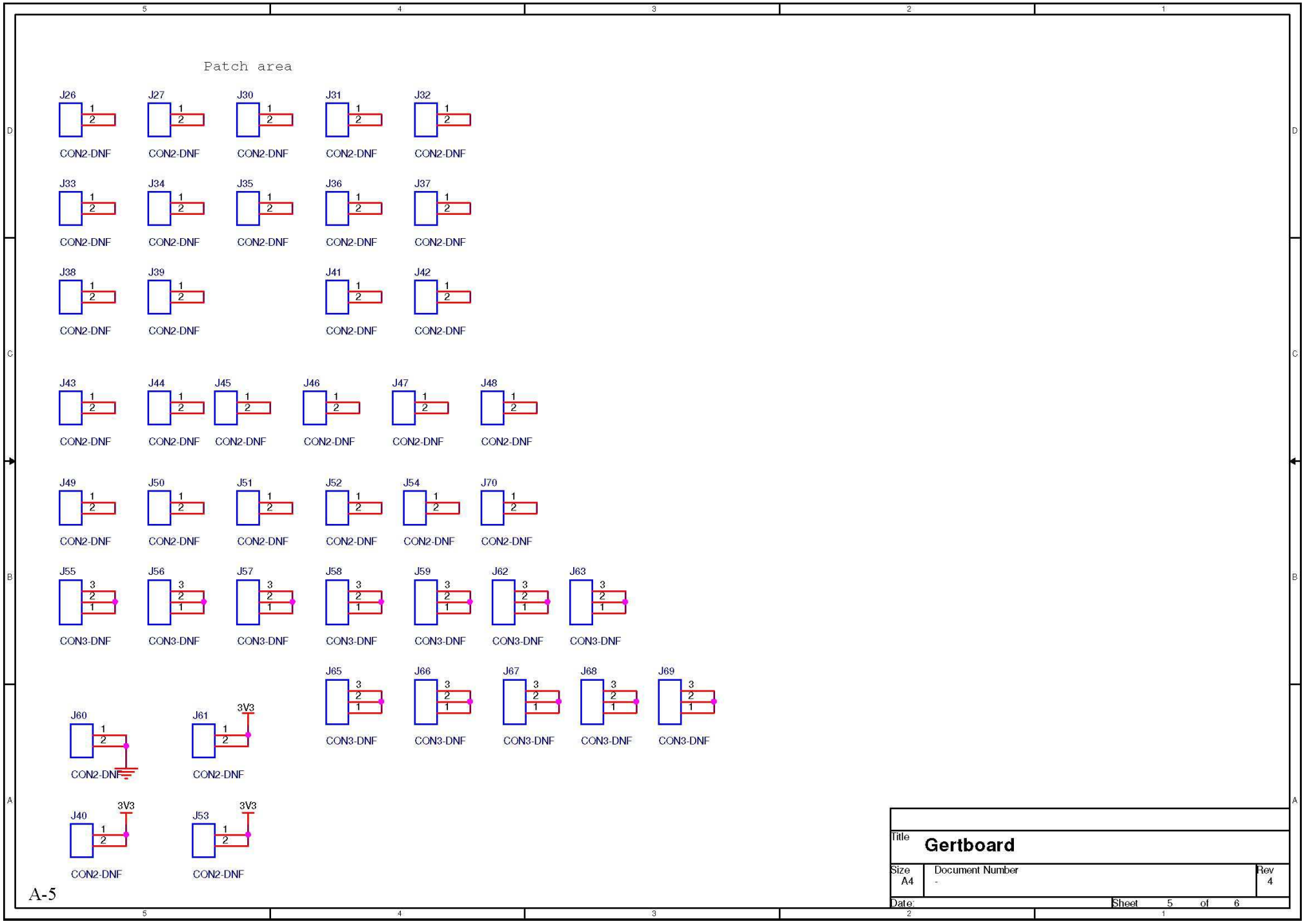
Title	**Gertboard**	
Size A4	Document Number -	Rev 4
Date:	Sheet 3 of 6	

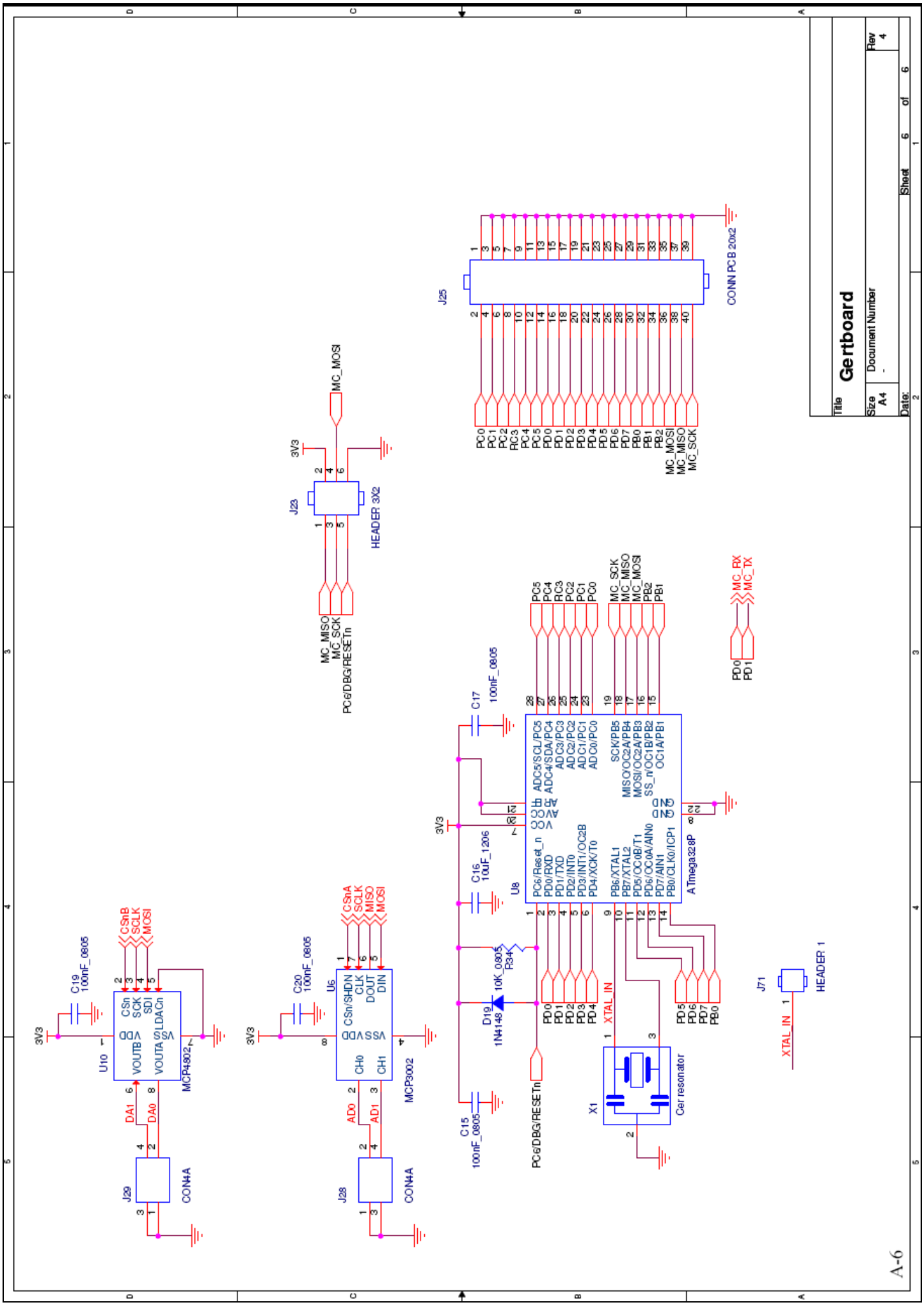


motor power nets named to make high current

Title	Gertboard		
Size A4	Document Number -		Rev 3
Date:		Sheet 4 of 6	

A-4


Patch area
J26
J27
J30
J31
J32
J33
J34
J35
J36
J37
J38
J39
J41
J42
J43
J44
J45
J46
J47
J48
J49
J50
J51
J52
J54
J70
J55
J56
J57
J58
J59
J62
J63
J65
J66
J67
J68
J69
J60
J61
3V3
J40
J53
CON2-DNF
CON3-DNF
A-5
Title
Gertboard
Size
A4
Document Number
-
Rev
4
Date:
Sheet 5 of 6


Gertboard
Document Number
Rev 4
Sheet 6 of 6
Size A4
Date:
CONN PCB 20x2
HEADER 3X2
HEADER 1
ATmega328P
MCP4802
MCP3002
CON4A
Cer resonator
1N4148
10K_0805
100nF_0805
10uF_1206
MC_MOSI
MC_MISO
MC_SCK
MC_RX
MC_TX
XTAL_IN
PC6/DBG/RESETn
A-6