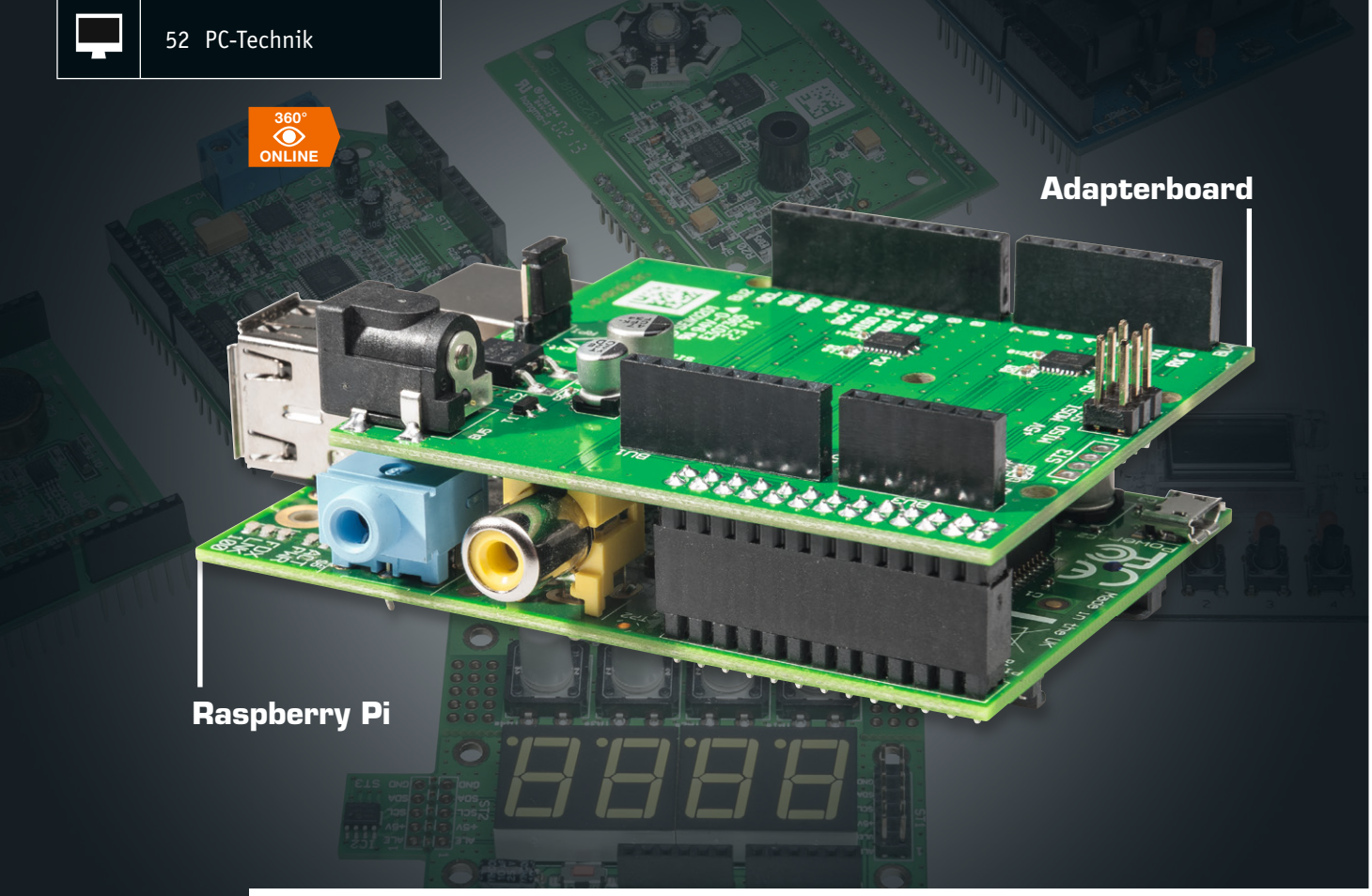


Arduino-Shields am Raspberry Pi einsetzen – Shield-Adapterboard

Infos zum Bausatz
im ELV-Web-Shop
#1321

Es gibt unzählige Arduino-Shields – warum diese nicht auch am Raspberry Pi betreiben? So kann man sich in vielen Fällen den Entwurf und die Realisierung von Hardware-Zusätzen ersparen und vielleicht sogar schon vorhandene Arduino-Shields am ARM-Rechner weiter nutzen. Das hier vorgestellte Adapterboard ermöglicht dies auf einfache Weise und stellt zusätzlich Logik-Pegelwandler und die auf dem Raspberry Pi fehlenden A/D-Wandler-Ports zur Verfügung.

Welten verbinden

Für den weit verbreiteten Arduino-Mikrorechner gibt es Erweiterungsboards, Shields genannt, quer durch die Elektronik – von der Netzwerkerweiterung über Displayboards bis hin zu Motor- und Lichteffektsteuerungen. Da sich der stets neugierige Techniker ständig weiterentwickelt, kommt wohl bei den meisten Arduino-Nutzern früher oder später der Schritt zum komplexeren ARM-Mikrorechner, eben dem Raspberry Pi oder Kompatiblen. Diese Rechnerklasse hat quasi den gleichen Hintergrund wie der Arduino – es sollen möglichst viele (junge) Menschen auf einfache Weise an die Technik herangeführt werden. Der Schritt zur externen Erweiterung des Systems ist ebenso logisch wie beim Arduino. Auch für den Raspberry Pi gibt es bereits sehr viele Shields, aber erstens noch lange nicht in der Breite des länger auf dem Markt befindlichen Arduino, und zweitens wäre es für bisherige Arduino-Nutzer kontraproduktiv, vorhandene Shields des Systems beim Systemwechsel quasi in der Schublade verschwinden zu lassen.

Schon gibt es spezialisierte Raspberry-Pi-kompatible Plattformen, die die Umsetzung auf Arduino-Boards bis hinein in die Mega-Klasse an Bord haben, so z. B. der PCDuino, das Udoo-Board oder das Embedded-Pi-Board.

Wer allerdings schon einen Raspberry Pi sein Eigen nennt, muss deswegen kein neues Board kaufen, unser Adapterboard macht den direkten Anschluss vieler Arduino-Shields möglich. Auf der Hardwareseite ist dabei die Aufgabe relativ einfach: Das GPIO-Layout des Raspberry Pi muss ins Shield-Anschlusslayout des Arduino umgesetzt werden, es ist eine Pegelanpassung zwischen beiden Systemen nötig, und

Daten

Geräte-Kurzbezeichnung:	RPi-AA1
Versorgungsspannung:	7–9 VDC oder 5 V vom Raspberry Pi 3,3 V vom Raspberry Pi
Stromaufnahme:	500 mA max.
Umgebungstemperatur:	5 bis 35 °C
Abmessungen (B x H x T):	60 x 24 x 72 mm
Gewicht:	27 g

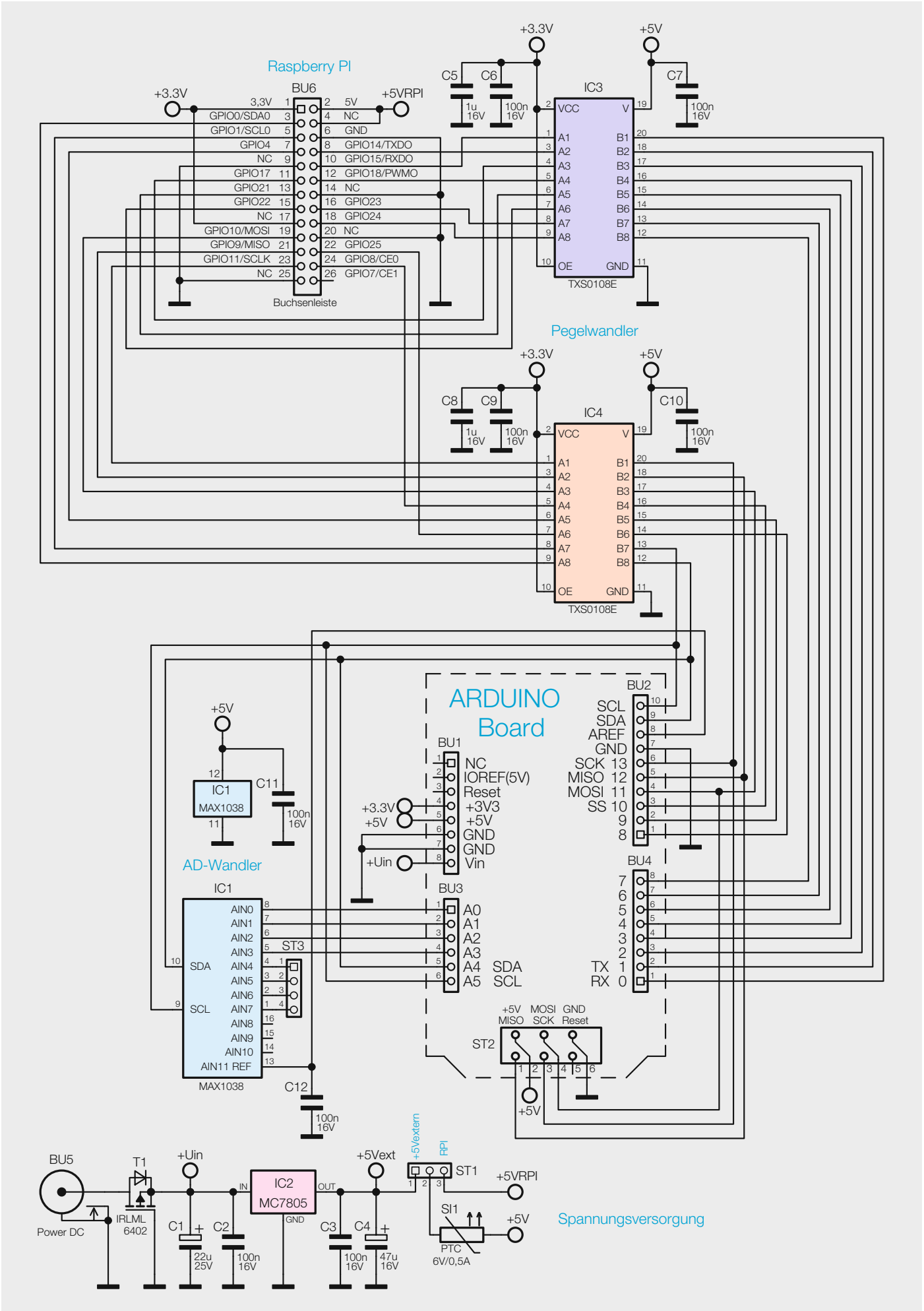


Bild 1: Das Schaltbild des Shield-Adapterboards RPi-AA1

die aufzusteckenden Shields müssen mit Spannung versorgt werden. Etwas trickreicher ist hingegen die Software-Anpassung, aber auch hier erleichtern speziell angepasste Software-Bibliotheken, die Libs, die Kontaktaufnahme.

Beides stellen wir mit diesem Adapter-Projekt zur Verfügung.

Da ist einmal das Adapterboard: Es trägt die für beide Systeme erforderlichen Steckverbinder, bidirektionale Pegelwandler, die den Logikpegel des 5-V-Arduino-Systems auf den des 3,3-V-Raspberry-Pi-Systems umsetzen.

Zusätzlich ist noch ein praktisches Feature untergebracht, das dem Raspberry Pi fehlt: ein A/D-Wandler, der dem Raspberry Pi 8 Analogeingänge mit einer Auflösung von je 8 Bit zur Verfügung stellt.

Die Spannungsversorgung des Boards und darauf aufgesteckter Shields erfolgt direkt über den Raspberry Pi oder kann mittels einer externen Spannungsquelle realisiert werden. Das ist wichtig bei leistungshungrigeren bzw. dem Einsatz mehrerer Shields.

Da der Raspberry Pi und das Arduino-Board sich verständlicherweise in der Hardware unterscheiden, bieten beide Systeme natürlich auch unterschiedliche Funktionen, so dass nicht alle Funktionen des Arduino auf dem Raspberry Pi möglich sind und umgekehrt. Deshalb sind auch nicht alle Shields mit diesem Adapter verwendbar. Bei einigen gibt es gewisse Einschränkungen, da z. B. bestimmte Pins nicht verfügbar oder mit anderen Funktionen belegt sind. Die von uns angepassten Beispiele für die Arduino-Shields wurden für den Raspberry Pi portiert und sind auf der Produktseite [1] zu finden.

In der Anwendungs- bzw. Softwarebeschreibung gehen wir darauf detailliert ein.

Schaltungsbeschreibung

Die Schaltung (Bild 1) ist, wie eingangs angedeutet, recht einfach gehalten und besteht im Wesentlichen aus der Spannungsversorgung, Pegelwandler und einem A/D-Wandler sowie den Steckverbindern zum Raspberry-Pi-Board und den Arduino-Shields.

Die Spannungsversorgung kann wahlweise über ein externes Netzteil oder über den Raspberry Pi erfolgen, eine zentrale Rolle spielt dabei die Stiftleiste ST1. Über diese lässt sich mit dem beiliegenden Jumper die externe Spannungsquelle (Jumper auf Pin 1 und Pin 2) oder die Spannungsversorgung über den Raspberry Pi (Jumper auf Pin 2 und Pin 3) auswählen. Dazu ist der Jumper auf ST1 entsprechend zu setzen.

Über BU5 kann ein externes Netzteil angeschlossen werden, T1 dient als Verpolungsschutz, der Linearregler IC2 erzeugt dann aus der Eingangsspannung von 7 bis 9 V eine Gleichspannung von 5 V. Die Kondensatoren C1 bis C4 dienen zur Stabilisierung.

Der Linearregler ist intern vor Überlastung und Übertemperatur geschützt. Die selbstrückstellende Sicherung SI1 schützt vor zu großen Strömen.

Achtung!

Der Spannungsregler IC2 kann je nach Belastung sehr heiß werden! Ggf. kann die Kühlung mit einem aufgesetzten Kühlkörper (aufkleben mit Wärmeleitfolie oder Wärmeleitkleber) unterstützt werden.

Pegelwandlung

Da der Raspberry Pi lediglich mit Logikpegeln von 3,3 V arbeitet, die Arduino-Shields jedoch meist mit 5 V, muss eine Pegelanpassung erfolgen, dazu wurden hier bidirektionale Pegelwandler von Texas Instruments verwendet, welche eine automatische Richtungserkennung beinhalten. So können die IOs beliebig als Ein- oder Ausgänge verwendet werden.

Die Pegelwandler beinhalten eine Treiberstufe, welche schnelle Flankenwechsel ermöglicht, so dass auch höhere Taktfrequenzen ohne weiteres möglich sind. Diese Treiberstufen sind jedoch nur in der Lage, kleinere Kapazitäten von 100 pF an den Ausgängen zu schalten, bei größeren Kapazitäten kann es zu Störungen in der Datenkommunikation kommen. Aus diesem Grund können z. B. die ELV-Shields GLD1 und RTC-DCF nicht auf dem Adapter betrieben werden, da dort die Kapazitäten größer ausfallen.

A/D-Wandler

Da der Raspberry selbst keine Analogeingänge zur Verfügung stellt, wurde ein A/D-Wandler IC1 vom Typ MAX1038 zusätzlich in die Adapterschaltung integriert. Dieser lässt sich über die I²C-Schnittstelle ansprechen und stellt acht Analogeingänge mit einer Auflösung von je 8 Bit zur Verfügung.

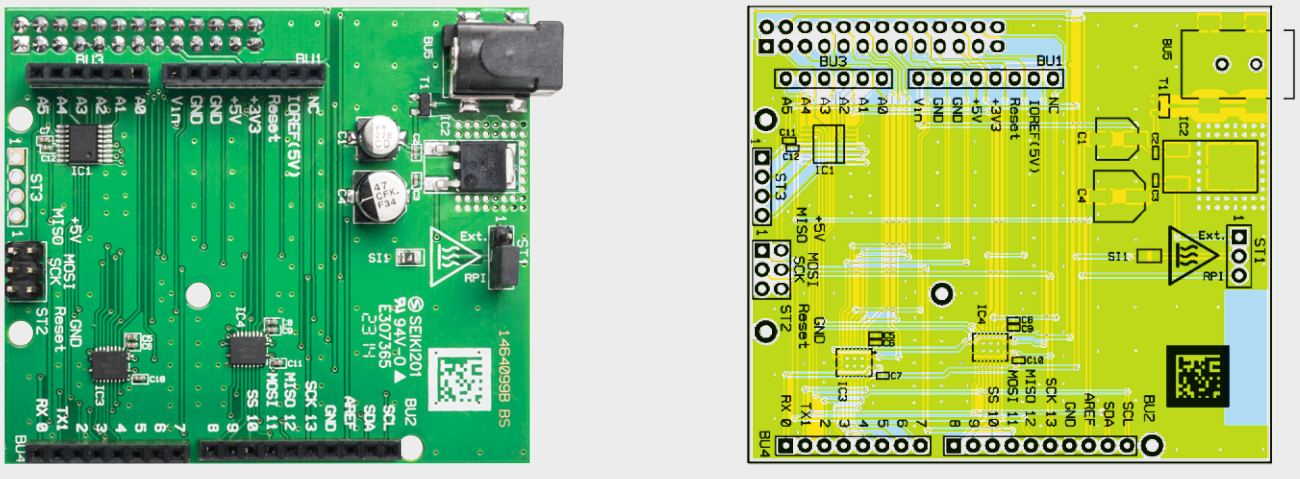


Bild 2: Das vollständig bestückte Adapterboard mit dem zugehörigen Bestückungsplan

Es besteht die Möglichkeit, statt der Betriebsspannung von 5 V auch eine externe Referenzspannung zu nutzen, diese muss aber zwischen 0 V und der Betriebsspannung liegen. Für die Nutzung dieser Option sollte das Datenblatt des MAX1038 zu Rate gezogen werden [2].

Nachbau

Da die meisten Komponenten als bereits ab Werk bestücktes SMD-Bauteil ausgeführt sind, beschränkt sich der Zusammenbau auf das Einsetzen der Stift- und Buchsenleisten BU1 bis BU4, BU6 und ST1 und ST2. Die jeweilige Bestückung ergibt sich aus dem Platinenfoto (Bild 2), dem Bestückungsplan und der Stückliste.

Bei der Bestückung ist darauf zu achten, dass die Bauteile jeweils plan auf der Platine aufliegen. Bild 3 zeigt die Platinenoberseite und die plane Lage der Buchsen.

Bei der Buchsenleiste BU6 (Bild 4) ist die Bauhöhe so gewählt, dass die Platine RPi-AA1 auf der Netzwerkbuchse des Raspberry genau aufliegt, dafür ist auf der Unterseite der Platine extra ein Bereich freigestellt worden, so dass dort keine ungewollten Verbindungen zwischen Netzwerkbuchse und Signalleitungen/Versorgungsspannung zustande kommen können.

Wichtiger Hinweis:

Für einen ausreichenden Schutz vor elektrostatischen Entladungen ist der Einbau in ein geeignetes Gehäuse erforderlich, damit die Schaltung nicht durch eine Berührung mit den Fingern oder mit Gegenständen gefährdet werden kann.

Stückliste

Kondensatoren:	
100 nF/16 V/SMD/0402	C2, C3, C6, C7, C9–C12
1 µF/16 V/SMD/0402	C5, C8
22 µF/25 V	C1
47 µF/16 V	C4
Halbleiter:	
MAX1038/SMD/QSOP-16	IC1
MC7805CDT/SMD	IC2
TXS0108ERGYR/SMD	IC3, IC4
IRLML6402/SMD	T1
Sonstiges:	
Buchsenleiste, 1x 8-polig, print, gerade	BU1, BU4
Buchsenleiste, 1x 10-polig, print, gerade	BU2
Buchsenleiste, 1x 6-polig, print, gerade	BU3
Hohlsteckerbuchse, 6,5/2,1 mm, SMD	BU5
Buchsenleiste, 2x 13-polig, gerade	BU6
Stiftleiste, 1x 3-polig, gerade, 20,3 mm, print	ST1
Stiftleiste, 2x 3-polig, gerade, print	ST2
PTC, 0,5 A, 6 V, SMD, 0805	SI1
1 Jumper ohne Grifflasche, geschlossene Ausführung	
2 Ferrit-Ringkerne, 25 (15) x 12 mm	

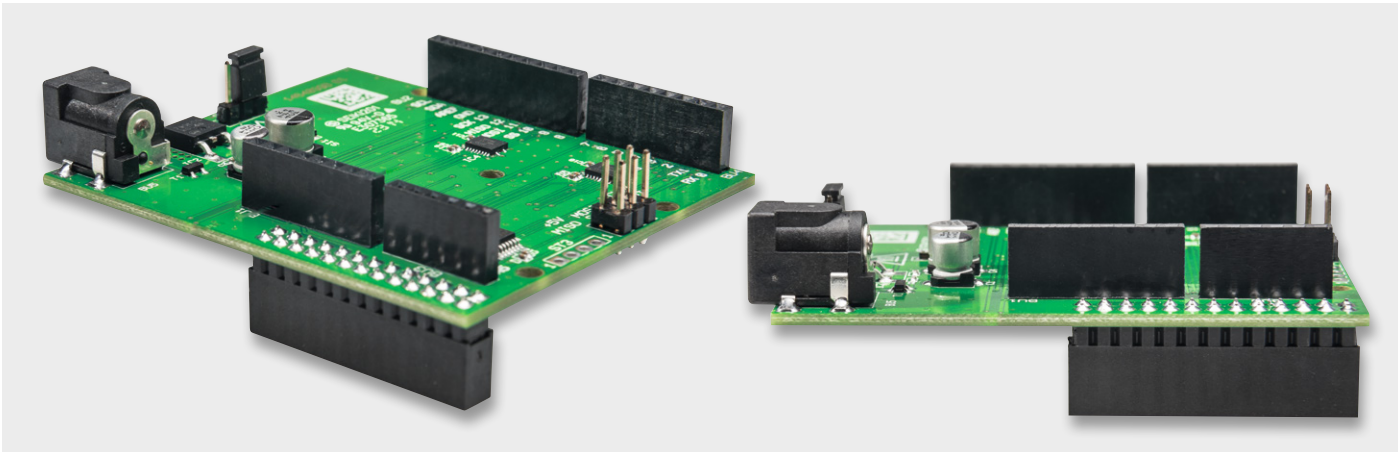


Bild 3: Die bestückte Platine, rechts ist die plane Lage der Buchsen auf der Platine gut zu sehen

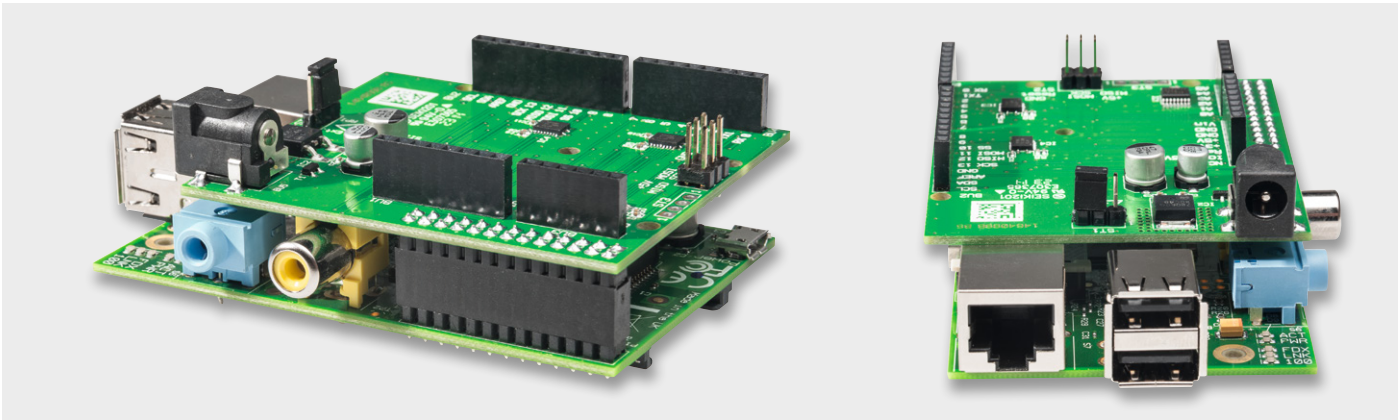


Bild 4: Die flache Buchsenleiste BU6 ermöglicht ein dichtes Aufsetzen des Adapterboards auf den Raspberry Pi.

Wichtiger Hinweis:

Zur Gewährleistung der elektrischen Sicherheit muss es sich bei der speisenden Quelle um eine Sicherheits-Schutzkleinspannung handeln. Außerdem muss es sich um eine Quelle begrenzter Leistung gemäß EN60950-1 handeln, die nicht mehr als 15 W liefern kann. Üblicherweise werden beide Forderungen von handelsüblichen Steckernetzteilen mit entsprechender Leistung erfüllt.

Vorbereitungen Raspberry Pi – Raspbian-Installation

Achtung: Zum Wechseln der Shields Raspberry ausschalten und von der Spannungsversorgung trennen!

Um mit den Beispielen arbeiten zu können, müssen wir auf dem Raspberry Pi das Betriebssystem Raspbian installieren, dafür wird eine SD-Karte mit mindestens 2 GB benötigt. Zusätzlich ist eine Internetverbindung zum Raspberry Pi erforderlich.

Das Raw Image zu Raspbian gibt es auf der offiziellen Website [3] als Download. Nach dem Entpacken muss das Image auf die SD-Karte geschrieben werden. Das Image aus dem Download kann nicht direkt auf die SD-Karte kopiert werden, sondern ist wie im Kasten Elektronikwissen beschrieben zu behandeln.

Bevor nun die Spannungsversorgung an den Raspberry angeschlossen wird, setzen wir die vorbereitete SD-Karte ein und schließen eine USB-Tastatur und einen Bildschirm via HDMI an.

Nach einigen Initialisierungssequenzen erscheint beim erstmaligen Start eine grau-blaue Oberfläche samt Menü, in der einige Einstellungen mittels Pfeiltasten, Enter und Escape durchgeführt werden können.

Mittels Expand Filesystem stellen wir dem Raspbian auch den bisher ungenutzten Speicherbereich der SD-Karte zur Verfügung. Diese Änderungen werden erst nach einem Neustart wirksam:

Internationalisation Options
„Change Keyboard Layout“ – dort stellen wir eine deutsche Tastatur ein. Es wird empfohlen, „Classmate PC“ zu wählen. Danach folgt das Keyboardlayout, dort wählen wir zunächst „Other“ und anschließend „German“ (ohne Zusätze). Nun folgen weitere Einstellungen: The default for the keyboard layout, No compose key und STRG + ALT + Entfernen No.

Change Locale
„de_DE.UTF-8“ UTF-8 mittels Leertaste bestätigen, dann mit Tab auf OK. Für die korrekte Ausgabe von anderssprachigen Anwendungen wählen wir „en_GB.UTF-8“ und bestätigen mit Enter.

Elektronikwissen

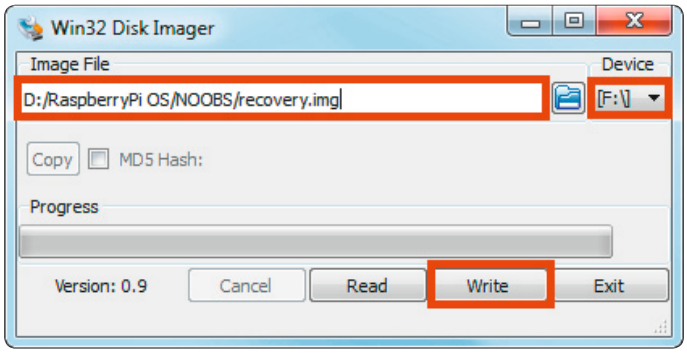
Raspbian auf SD-Karte kopieren

Betriebssystem MS Windows:
Unter Windows ist das Tool „Win32DiskImager“ [4] zu verwenden. Win32DiskImager ist nach dem Download zu entpacken und als Administrator auszuführen.

Es erscheint das Programmfenster (rechts), dort wird bei „Image File“ das Image von Raspbian angegeben. Unter „Device“ ist das Laufwerk der SD-Karte auszuwählen.

Danach wird der Schreibvorgang mittels „Write“ gestartet, dieser Vorgang dauert einige Zeit.

Nachdem der Vorgang abgeschlossen ist, kann das Programm beendet werden und die SD-Karte ist bereit für den Einsatz im Raspberry Pi.



Das Programmfenster von Win32DiskImager mit den notwendigen Einstellungen und Bedienschritten

Betriebssystem Linux:
Unter Linux ist das Tool „dd“ [5] zu verwenden. Mittels df–h lassen wir alle Laufwerke auflisten, nun stecken wir unseren Kartenleser an und lassen uns die Laufwerke erneut auflisten. Bei dem neu erscheinenden Laufwerk handelt es sich um die SD-Karte, typisch sind Bezeichnungen wie „/dev/mmcblk0p1“ oder „/dev/sdd1“. Das p1 oder 1 am Ende ist die Kennzeichnung der Partition, da die ganze SD-Karte genutzt werden soll, müssen diese in den folgenden Schritten entfernt werden „/dev/mmcblk0“ oder „/dev/sdd“.

Es können auch mehrere Partitionen auf einer SD-Karte vorhanden sein. Um die Karte neu beschreiben zu können, müssen alle Partitionen freigegeben werden:

```
umount /dev/sdd1
```

In dem folgenden Befehl sind die Parameter zu beachten. Bei if= ist der Pfad zu unserem Image anzugeben, bei of= ist die SD-Karte einzutragen:

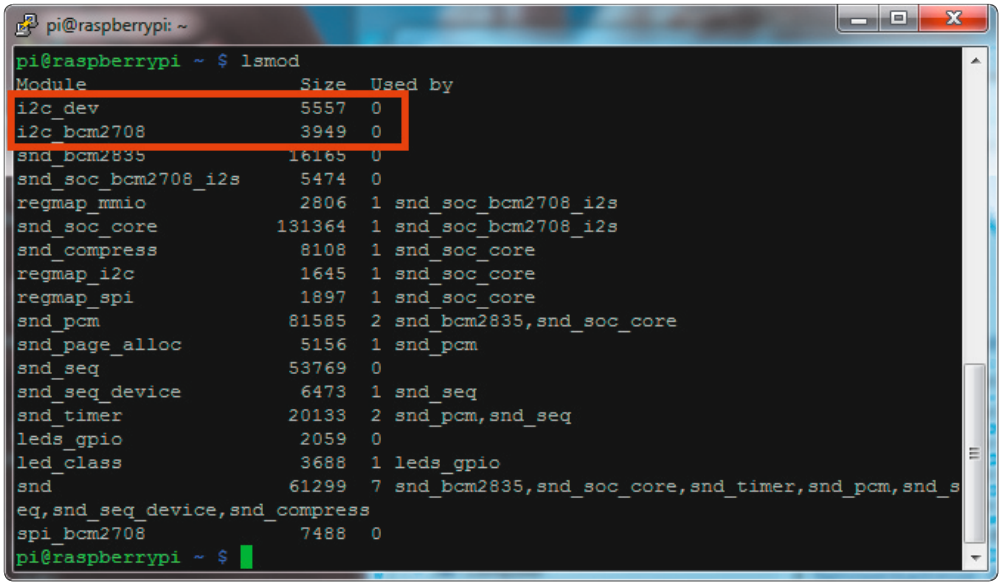
```
sudo dd bs=4M if=~/2012-12-16-wheezy-raspbian.img of=/dev/sdd
```

Um sicher zu gehen, dass alles auf die Karte geschrieben wurde und wir diese entfernen können:

```
sudo sync
```

Nun ist die Karte für den Einsatz im Raspberry Pi bereit.

Bild 5: Das Ausgabebeispiel für die Ausgabe von lsmod



Change Timezone
Zeitzone auf Europe und Berlin einstellen.

Advanced Options SSH Enable
Damit wird der SSH-Dienst automatisch beim Starten aktiviert, und so kann der Raspberry Pi über SSH im Netzwerk bedient werden.

Monitor und Tastatur sind ab hier nun nicht mehr zwingend erforderlich.

Es folgt nun ein Neustart mit:

Benutzername: pi
Passwort: raspberry

Es empfiehlt sich, nach dem Neustart des Raspberry eine Aktualisierung durchzuführen, dazu werden in der Konsole folgende Befehle ausgeführt (Internetverbindung vorausgesetzt):

```
sudo apt-get update
sudo apt-get upgrade
```

Dieser Schritt kann einige Zeit in Anspruch nehmen, da die Updates erst heruntergeladen werden müssen und anschließend installiert werden.

Weitere Hilfen sind auf der Raspberry-Homepage [3] und vielen anderen Quellen zu finden.

Verwendung der I²C-Schnittstelle

In unserem Beispiel gehen wir von einer Raspbian-Installation aus. In dieser werden die I²C-Module für das Betriebssystem beim Starten nicht automatisch geladen. Deshalb müssen diese Module bei Verwendung der I²C-Schnittstelle manuell nach jedem Neustart des Raspberry wie folgt geladen werden:

```
sudo modprobe i2c-bcm2708
sudo modprobe i2c-dev
```

Oder die Module werden für den Autostart in die Modulliste eingetragen. Dies geschieht z. B. mit dem Texteditor nano

```
sudo nano /etc/modules
```

Dort müssen nun die zwei folgenden Einträge ergänzt werden.

```
i2c-bcm2708
i2c-dev
```

Nach einem Neustart sollten die I²C-Module nun gestartet sein. Überprüfen können wir dies mit dem Befehl „*lsmod*“, welcher alle aktiven Module auflistet (Bild 5).

Verwendung der UART-Schnittstelle

Beim Raspbian-System wird standardmäßig der komplette Bootvorgang samt Login zusätzlich auf der UART-Schnittstelle ausgegeben. Wollen wir nun aber ein Arduino-Shield mit UART-Schnittstelle ansprechen, kann dies zu Problemen führen, deswegen müssen diese Meldungen auf der UART-Schnittstelle deaktiviert werden. Dazu gehen wir wie folgt vor:

Ändern des Files /boot/cmdline.txt:
Sicherheitshalber erstellen wir vor dem Editieren eine Sicherheitskopie:

```
sudo cp /boot/cmdline.txt /boot/cmdline_backup.txt

sudo nano /boot/cmdline.txt
```

In der Datei ist folgende Zeile enthalten.

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200
kgdboc=ttyAMA0,115200 console=tty1 root=/dev/...
```

Die auf „console=tty1“ folgenden Parameter müssen unbedingt erhalten bleiben, diese sind hier nicht weiter aufgeführt, da diese Parameter Pfadangaben auf dem aktuellen System enthalten und deshalb unterschiedlich sind. Lediglich die Parameter, welche die UART-Schnittstellen (ttyAMA0): enthalten, werden entfernt:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/...
```

Zusätzlich muss in der /etc/inittab die folgende Zeile mittels Voranstellen einer # auskommentiert werden:

```
sudo nano /etc/inittab

#Spawn a getty on Raspberry Pi serial line
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Nach dem Editieren sieht diese so aus:

```
#Spawn a getty on Raspberry Pi serial line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Nach einem Neustart werden diese Einstellungen übernommen.

sudo reboot

Programmieren auf dem Raspberry Pi

Programme werden hier wie in der Arduino-IDE in der Programmiersprache C++ verfasst, zum Kompilieren wird der im Raspbian installierte g++ compiler verwendet.

Programme können mit einem Editor wie z. B. nano erstellt werden, den wir für die Einrichtung unseres Systems bereits verwendet haben.

Der g++ compiler wird mit den Optionen „*-lrt*“ und „*-lpthread*“ aufgerufen, damit die rt Lib und pthread Lib geladen werden, denn diese sind z. B. für die Interrupts nötigt, da Interrupts in eigenen Threads abgehandelt werden.

In *test.cpp* ist unser Testprogramm enthalten. Der Parameter –o gibt an, wohin das Kompilat geschrieben werden soll.

g++ -lrt –lpthread test.cpp –o test

Da der Benutzter „pi“, unter dem wir auf dem Raspbian-System arbeiten, keine vollen Zugriffsrechte besitzt, lässt sich unser erstelltes Programm jedoch nicht direkt ausführen.

Der Befehl „sudo“ bewirkt, dass wir mit erweiterten Zugriffsrechten arbeiten können.

Unser Programm führen wir dementsprechend wie folgt aus:

sudo ./test

„./“ gibt den aktuellen Ordner als Pfad an.

Es müssen immer alle zu dem Projekt gehörenden Quellcodes kompiliert werden.

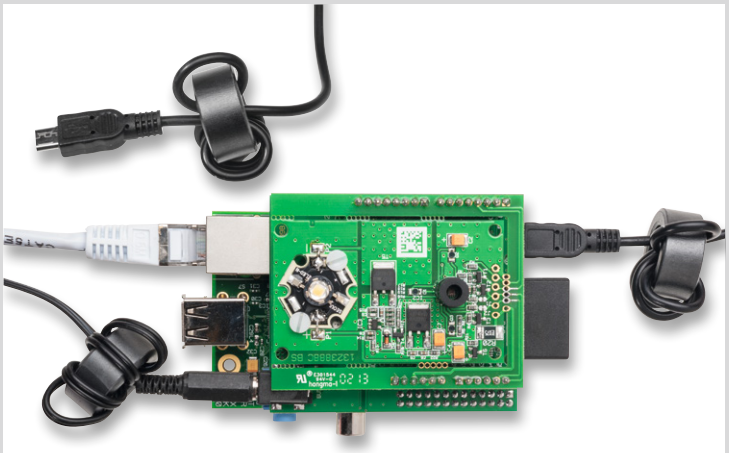
In dem Beispiel für ADCtest werden neben der ADCtest.cpp auch die Bibliotheken ADCPi und arduPi benötigt, der Compiler-Aufruf sieht dann folgendermaßen aus:

g++ -lrt –lpthread ADCPi.cpp arduPi.cpp ADCtest.cpp –o ADCtest

Achtung – Störaussendung:

Beim Einsatz des Adapters mit einem Raspberry Pi ist im Hinblick auf die Störaussendung noch etwas zu beachten. Da der Raspberry Pi beim Anschluss von externen Komponenten dazu neigt, Störsignale über die Zuleitungen auszusenden, muss ein Ferritring in die Zuleitung der Versorgungsspannung eingebracht werden. Hierzu wird die Zuleitung viermal durch den Ferritring geführt.

Gleiches gilt auch für die Zuleitung bei externer Versorgung des Adapters. Das Bild zeigt die zwei Beispiele dazu.



Weitere Hilfen zur Programmierung finden sich auf zahlreichen Raspberry-Pi-Seiten und -Foren im Internet, deswegen gehen wir hier nicht näher darauf ein. Zusammen mit den Beispielen ist dies für den Einstieg ausreichend.

Einsatzbeispiele mit Arduino-Shields

Nachdem der Raspberry Pi mit dem Raspbian-Betriebssystem und den Schnittstelleneinstellungen vorbereitetet wurde, können wir uns den Beispielen für die Arduino-Shields widmen.

Für die Umsetzung der aus dem Arduino-Bereich gewohnten Befehle wird hier eine angepasste Version der arduPi-Bibliothek von cockinghacks [6] eingesetzt. Diese wurde an einigen Stellen angepasst, um zum Beispiel mit dem hier eingesetzten A/D-Wandler zu kommunizieren.

Die Dokumentation zu der Bibliothek „arduPi“ ist unter [6] zu finden, die Pinzuordnungen und auch die Schnittstellen I²C, SPI und UART sind so umgesetzt, dass sich viele Funktionen in vom Arduino gewohnter Weise verwenden lassen.

Beachtet werden müssen dabei jedoch z. B. die Clock-Teiler für I²C und SPI. Da hier nun mit dem Takt des Raspberry Pi gearbeitet wird, müssen diese entsprechend angepasst werden, um auf die gleiche Taktgeschwindigkeit zu kommen.

Deshalb sind in der Bibliothek die Definitionen der SPI-Teiler angepasst worden, damit diese denen eines Arduino Uno mit 16-MHz-Takt entsprechen.

Auch bei I²C wurde die Taktgeschwindigkeit auf 100 kHz als Defaultwert geändert, beim Raspberry Pi wäre eine Taktfrequenz von 166 kHz standardmäßig.

Hinweis: Da die analogen Eingänge beim RPi-AA1 nicht als digitale I/O-Ports verwendet werden können, muss man bei Einsatz der Arduino-Shields teilweise mit Einschränkungen rechnen, so können z. B. beim ASA1 die SD-Kartenerkennung und die LEDs nicht verwendet werden.

Nach der notwendigen Einleitung kommen wir nun zu einer direkten Anwendung.

Für die von ELV entwickelten Arduino-Shields stehen entsprechende Beispiele auf der Produktseite [1] zum Download bereit. Diese müssen lediglich auf die Speicherkarte des Raspberry kopiert und dann auf dem Raspberry kompiliert und ausgeführt werden.

Im Ordner „Libs“ sind die Bibliotheken abgelegt, dort sind die Dateien für ADCPi, PWMPi und arduPi zu finden, zusätzlich sind dort in Unterordnern für die einzelnen Shields weitere Bibliotheken abgelegt.

Im Ordner „Examples“ befinden sich die einzelnen Beispiele für die Shields, in jedem Beispiel ist ein „Makefile“ abgelegt, in dem die Compiler-Aufrufe mit allen benötigten Parametern und Dateien aufgeführt sind.

Da die Bibliotheken in anderen Ordnern liegen, sind die Pfade mit anzugeben und über den Parameter „–I“ die Pfade zusätzlich als Include-Ordner anzugeben.

Ein kompletter Compiler-Aufruf für das ADCtest-Beispiel sieht im Makefile so aus:

sudo g++ -lrt –lpthread -I ./../../libs/ ./../../libs/ADCPi.cpp ./../../libs/arduPi.cpp ADCtest.cpp –o ADCtest

Mit Hilfe des Makefiles kann einfach durch einen Aufruf von „make“ das entsprechende Beispiel in den Examples-Ordner kompiliert werden.

Folgende Beispiele stehen zur Verfügung, nähere Erläuterungen zu den Modulen selbst und ihren Funktionen finden sich in den jeweiligen Dokumentationen zu den Modulen:

ADC (ADC auf dem RPi-AA1, Bild 6)
Das Beispiel „ADCtest“ ermöglicht das Einlesen der Werte an den acht Analogeingängen. Weitere Erläuterungen dazu finden sich im Kapitel „ADCPi-Bibliothek“.

PWM (Ausgabe auf dem RPi-AA1)
„PWMtest“
Das Beispiel erzeugt ein PWM-Signal mit sich ändernden Pulsweiten an Pin 4 (BU4).

ASA1 (ELV-ASA1 auf dem RPi-AA1, Bild 7)
„ASA1_sinetest“
Das Beispiel lässt den MP3-Decoder des ASA1-Moduls einen Sinus-Testton erzeugen.

„ASA1_flash“
Das Beispiel beinhaltet das Abspielen eines im Code abgelegten Sounds.

„ASA1_simple“
Hier erfolgt das Abspielen der Dateien „001.mp3“ in einer Schleife.

„ASA1_extended“
Hier kann man das Abspielen der Dateien „001.mp3“ bis „100.mp3“ in einer Schleife starten.

TwoWireFlipSign (ELV-I²C-Flip-Anzeige auf dem RPi-AA1, Bild 8)
„SwitchWithoutTimer“
Hier erfolgt das Umschalten der Flip-Anzeige mittels Delays.

„ToggleWithTimer“
Das Beispiel zeigt das Umschalten der Flip-Anzeige über einen integrierten Timer.

TwoWireLCD (ELV-I²C-LCD auf dem RPi-AA1, Bild 9)
„runningLED“
Das Beispiel erzeugt ein Lauflicht mit den LEDs des Moduls.

„runningText“
Hier wird ein Lauftext auf dem LC-Display generiert.

„simpleClock“
Das Beispiel erzeugt eine Uhrzeitanzeige auf dem LC-Display.

„simpleVoltmeter“
Mit diesem Beispiel wird eine Voltmeter-Anzeige über einen Analogeingang realisiert.

I2C_4DLED (ELV-I²C-4DLED auf dem RPi-AA1, Bild 10)
„setBrightness“
Das Beispiel erzeugt eine Helligkeitsänderung der LED-Anzeige.

„showTemperatureWithConversion“
Hier erfolgt eine Temperaturanzeige, als Sensor dient der auf dem Modul integrierte Temperatursensor MCP9801.

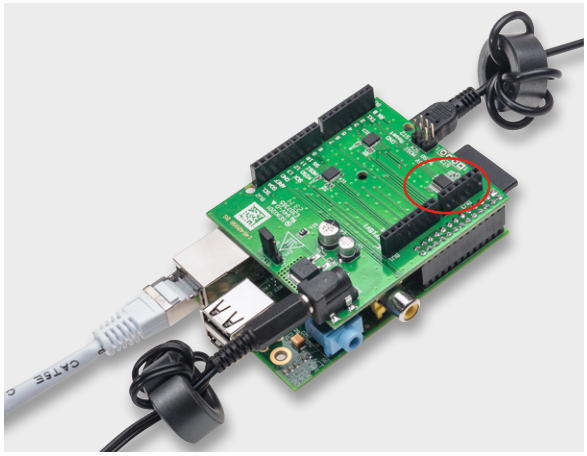


Bild 6: Das Einlesen von Analogwerten erfolgt über den auf dem RPi-AA1 integrierten A/D-Wandler und die Kontakte der BU3.

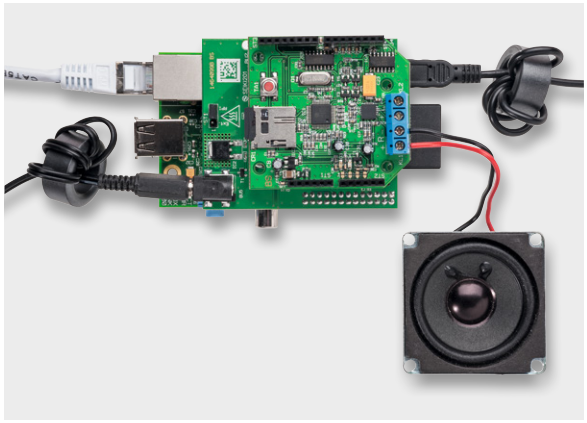


Bild 7: Anwendungsbeispiel mit dem ELV-Audioshield ASA1 (Best.-Nr. J6-10 59 22)

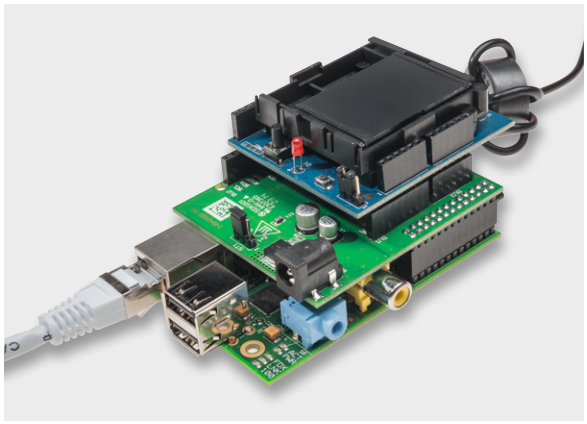


Bild 8: Hier ist der Einsatz der I²C-Flip-Anzeige (Best.-Nr. J6-10 48 63) zu sehen.

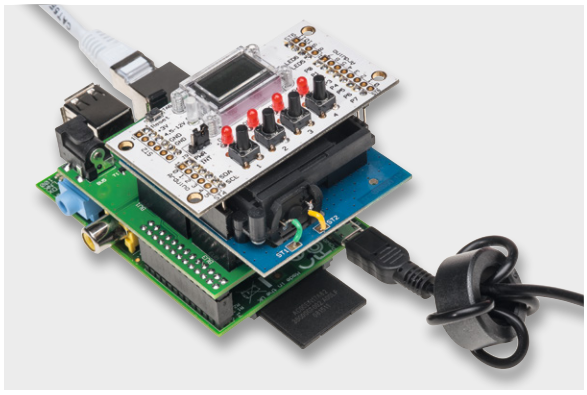


Bild 9: Auch für das I²C-LCD-Modul werden diverse Anwendungsbeispiele bereitgestellt (Best.-Nr. J6-09 92 53).

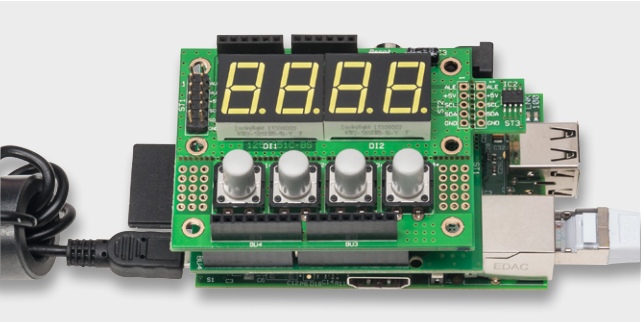


Bild 10: Durch den integrierten I²C-Temperatursensor besonders vielseitig einsetzbar – das I²C-4-Digit-LED-Display (Best.-Nr. J6-10 56 97).

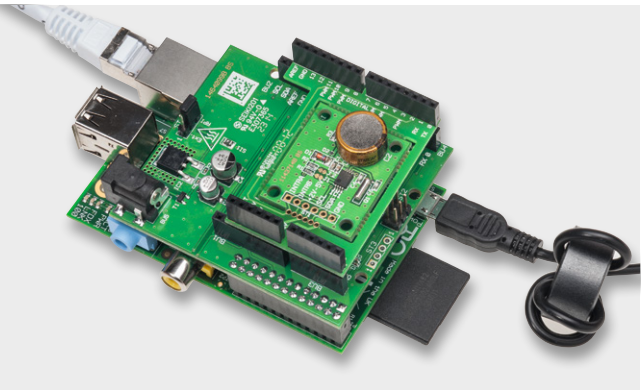


Bild 11: Vielseitiger Zeitgeberbaustein – auch der I²C-RTC-Baustein von ELV ist auf dem Adapter einsetzbar (Best.-Nr. J6-10 34 13).

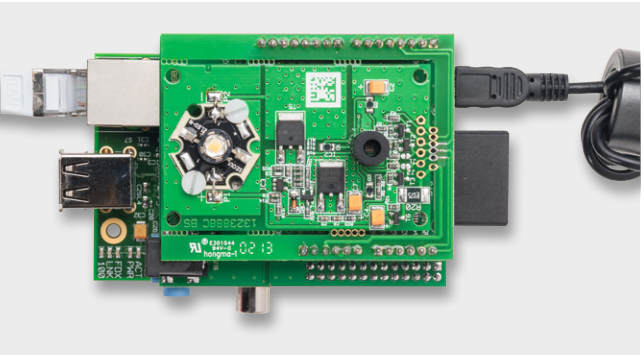


Bild 12: Ein Beispiel für die PWM-Steuerung ist der GLD1 (Best.-Nr. J6-13 02 25).

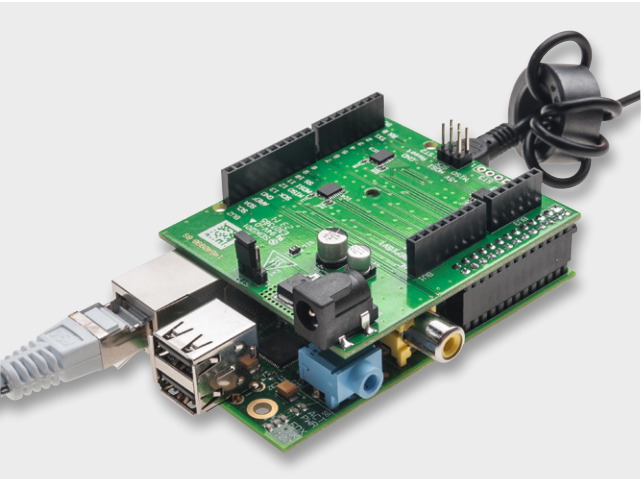


Bild 13: Die betriebsbereite Kombination aus Raspberry Pi mit Anschlusskabeln sowie dem RPi-AA1-Adapterboard.

Für dessen Nutzung sind drei weitere Beispiele unter „MCP9801“ vorhanden:
„configuration“ – Konfiguration des Sensors
„getTemperature“ – Einlesen eines Temperaturwertes
„usingAlertOutput“ – Alarmausgabe bei Erreichen eines Alarmwertes

„stopWatch“
Der Name sagt es, hier wird eine Stoppuhrfunktion realisiert.

„testDisplay“
Das Beispiel erzeugt einen Displaytest des LED-Displays.

TwoWireRTC (ELV-I²C-RTC auf dem RPi-AA1, Bild 11)
„oscillationWasHalted“
Das Beispiel demonstriert das Stoppen des Zeitgebers.

„periodicInterrupt“
Hier kann eine Interrupt-Funktion aktiviert werden, die man z. B. für die Taktgenerierung oder Weckzeiten heranziehen kann.

„SettingAlarm“
Das Beispiel setzt eine Weckzeit an, zu der ein Alarm ausgegeben wird.

„SettingTime“
Hiermit kann man die Zeit der internen Uhr einstellen.

ADCPi-Bibliothek
Der auf dem Adapterboard befindliche A/D-Wandler erweitert die Funktionalität des Raspberry Pi. Angeschlossen ist dieser an den I²C-Bus, die Adresse lautet „0x65“. Die ADCPi-Bibliothek kann für den hier eingesetzten A/D-Wandler genutzt werden.

Wie bereits aufgeführt, ist in „Examples“ ein entsprechendes Beispiel „ADCtest“ zur Verwendung der ADCPi-Bibliothek enthalten.

Die wichtigsten Funktionen zur Nutzung der ADCPi-Bibliothek lauten:

„ADC.begin();“ – Start der Erfassung
„ADC.analogRead(Kanalnummer);“ – Einlesen des Analogwertes eines Kanals

Beim A/D-Wandler wird in der Grundeinstellung die 5-V-Versorgungsspannung als Referenzspannung verwendet. Die Kanalnummer entspricht den Pin-Nummern der Analogeingänge 0 bis 8.

PWMPi-Bibliothek
Das Modul PWM.cpp erzeugt auf dem Hardware-PWM-Pin des Raspberry Pi ein PWM-Signal und gibt dies auf Pin 4 (BU4) des RPi-AA1 aus (siehe Anwendungsbeispiel PWM). Damit lässt sich z. B. die LED des GLD1 (Bild 12) ansteuern.

Bei dem GLD1 kann der Sensorchip aufgrund von Übertragungsproblemen mit dem auf dem Board eingesetzten Pegelwandler leider nicht verwendet werden. Alternativ ist der GLD1 entsprechend seiner Dokumentation nur als LED-Treiber für andere LEDs verwendbar.

Wichtiger Hinweis:

Die Nutzung des Hardware-PWMs kann zu Störungen bei der Audioausgabe auf der 3,5-mm-Klinkenbuchse führen, da für die Audioausgabe und den PWM-Pin die gleiche Peripherie genutzt wird.

Die wichtigsten Funktionen des PWMPi-Moduls sind:
Begin(); – Initialisierung des Hardware-PWM0 und PINs (GPIO18)
Achtung: GPIO18 wird ab jetzt als Ausgang verwendet!
setClockDivider(uint32); – Teiler des Grundtaktes (default 16 -> 1.2MHz)
setRange(uint32); – Auflösung des PWM (default 1024)
setData(uint32); – aktueller Wert des PWM (0 bis range-1)

Wer ein PWM-Signal auf einem anderen Pin oder mehrere PWM-Ausgänge benötigt, sollte sich näher mit Software-PWM beschäftigen, eine gute Bibliothek wäre z. B. PiBlaster.

Eine weitere Möglichkeit wäre der Einsatz eines externen Hardware-PWM-Bausteins PCA9685.

Andere Arduino-Beispiele für Raspberry vorbereiten:
Um den Einsatz der Beispiele möglichst einfach zu halten, kopieren wir alle Dateien der Arduino-Bibliothek in einen Ordner.

Header-Files wie Arduino.h, wire.h und SPI.h ersetzen wir durch die bereits erwähnte arduPi-Bibliothek mit arduPi.h, diese beinhaltet die wesentlichen Funktionen für den Umgang mit einem Arduino-Shield.

Bei Nutzung der Wire-Funktionen muss beachtet werden, dass mehrere Daten über die Wire-Funktionen nicht einzeln, sondern in einem Block geschrieben werden müssen. Dies liegt am Aufbau der I²C-Hardware im Raspberry Pi.

Ein Beispiel:
Folgende 2 Byte über I²C an ein Gerät mit der Adresse 0x65 senden:

```
char data1 = 0x23;
char data2 = 0xAC;
```

Einzelnes Senden bei Arduino:

```
Wire.beginTransmission(0x65);
Wire.write(data1);   //einzelnes Datenbyte senden
Wire.write(data2);   //einzelnes Datenbyte senden
Wire.endTransmission();
```

Daten im Block senden bei Raspberry

```
char temp_data[2];   //Array mit den zu sendenden Daten anlegen und füllen
temp_data[0] = data1;
temp_data[1] = data2;

Wire.beginTransmission(0x65);
Wire.write(&temp_data[0],2);   //Adresse und Länge der Daten übergeben
Wire.endTransmission();
```

In den mitgelieferten Beispielen für die Arduino-Shields wurde dies in den Bibliotheken bereits geändert. Es müssen, wie bereits erwähnt, immer alle zu dem Projekt gehörenden Quellcode-Dateien beim Kompilieren angegeben werden.

Bild 14 zeigt abschließend noch einmal eine Übersicht über alle in den Beispielen eingesetzten ELV-Modulbausteine. ELV

Weitere Infos:

[1] www.elv.de: Webcode #1326
[2] Datenblatt MAX1038: http://datasheets.maximintegrated.com/en/ds/MAX1036-MAX1039M.pdf
[3] www.raspberry.org
[4] http://sourceforge.net/projects/win32diskimager/
[5] http://en.wikipedia.org/wiki/Dd_(Unix)
[6] www.cooking-hacks.com/documentation/tutorials/raspberry-pi-to-arduino-shields-connection-bridge#step4

❶ *I²C-RTC-Baustein (Best.-Nr. J6-10 34 13)*
❷ *Audioshield ASA1 (Best.-Nr. J6-10 59 22)*
❸ *GLD1 (Best.-Nr. J6-13 02 25)*
❹ *I²C-Flip-Anzeige (Best.-Nr. J6-10 48 63)*
❺ *I²C-4-Digit-LED-Display (Best.-Nr. J6-10 56 97)*
❻ *I²C-LCD-Modul (Best.-Nr. J6-09 92 53)*

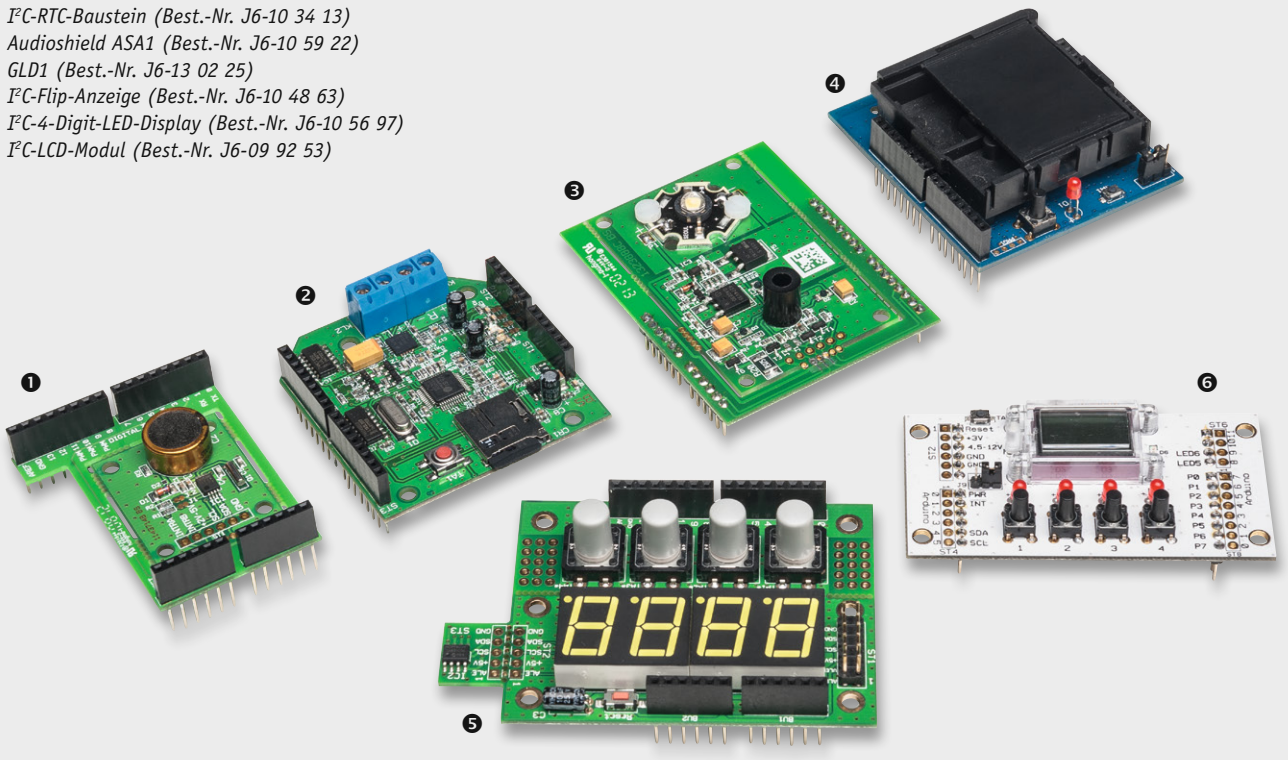


Bild 14: Übersicht über die auf dem Adapterboard einsetzbaren ELV-Bausteine, für die Programmbeispiele zur Verfügung gestellt werden. So ist ein schneller Einstieg möglich.