Raspberry Pi zum Anfassen

Chemnitzer Linux-Tage 2014

Andreas Heik

andreas.heik@linux-tage.de

im März 2014

Aus der Vision, Computertechnik für den schmalen Geldbeutel technisch interessierten Jugendlichen verfügbar zu machen entstand ein kreditkartengroßen Einplatinencomputer, der *Raspberry Pi* [1].
Mit diesem Artikel zum Workshop möchten wir den Gedanken aufgreifen und mit praktischen Hinweisen zum anfassen und experimentieren anregen.

1 Raspberry Pi

Der Raspberry Pi ist als kreditkartengroßer Einplatinen-Rechner erhältlich. Passende Gehäuse werden als Zubehör angeboten.

Auf der Platine arbeitet ein Prozessor der *ARM11-Familie* zusammengefasst mit einer *VideoCore IV* Grafikeinheit in einem SoC *(System on a Chip)*. Das Betriebssytem, natürlich ein Linux sowie Anwendungen und Daten finden auf einer SD-Karte Platz. Über einen Ethernetport und 2 USB-Anschlüsse kommuniziert der Raspberry Pi mit seiner Umgebung. In der GPU dekodierte Videos in HD-Auflösung werden über HDMI oder Composite Video ausgegeben. [2]
An einer Stiftleiste herausgeführte Low-Level Peripherie wie GPIO, UART, I^2C- und SPI-Bus laden zum basteln und programmieren ein. Ein Kameramodul vervollständigt die Liste der Erweiterungen.

2 Quickstart

Für die Inbetriebnahme des Raspberry Pi ist ein Netzteil mit Micro-USB Stecker und einer Belastbarkeit von wenigstens *1 A* erforderlich. Das Betriebssystem wird auf einer SD-Karte mit mindestens 2 GB, besser 4 GB Speicherkapazität installiert. Bei der Auswahl empfiehlt sich eine SD-Karte mit hoher Schreib-/Lesegeschwindigkeit. Eine eingekreiste Zahl auf der SD-Karte gibt dazu Auskunft.

Der HDMI-Ausgang des Raspberry Pi verbindet man am einfachsten mit einem Monitor oder TV-Gerät. Alternativ kann die Grafikausgabe auch über den Composite-Ausgang erfolgen. Tastatur und gegebenenfalls eine Maus werden an die USB-Ports angeschlossen. Eine Funktastatur mit Maus reduziert den Stromverbrauch und hält den zweiten USB-Port beispielsweise für einen USB-WLAN Stick frei.

Als Betriebssystem wird von der Raspberry Pi Foundation die auf Debian basierte Distribution *Raspbian Wheezy* [3] empfohlen. Eine interessante Möglichkeit, verschiedene Distributionen auf einer SD-Karte bereitzustellen bietet der NOOBS-Installer *(New Out of Box Software)*. Die Installation des entpackten Image auf eine SD-Karte erfolgt am einfachsten mit dem Werkzeug *dd*.

```
sudo dd if=2014-01-07-wheezy-raspbian.img of=/dev/SD-Karte bs=1M
```

Das vorgeschaltete *sudo* beschafft die erforderlichen Rechte für das blockweise Schreiben auf die Gerätedatei der SD-Karte.
Nach übertragen des Image befinden sich auf der SD-Karte eine 56 MB große */boot-Partition* (vfat-formatiert) und eine ca. 2 GB große Partition mit dem *Root-Filesystem* (ext4-formatiert).
SD-Karten mit vorinstalliertem *wheezy*-Image sind auch erhältlich.

Beim ersten Start des Raspberry Pi erscheint ein Konfigurationsmenü. Ist eine Tastatur angeschlossen empfiehlt sich jetzt die Konfiguration der *Tastatur*, der *Spracheinstellung* und der *Zeitzone*.

Wurde der Raspberry Pi an ein kabelgebundenes Netzwerk mit DHCP-Server angeschlossen ist zu diesem Zeitpunkt auch die Anmeldung mittels Secure Shell (*SSH*) möglich.
Die Anmeldung erfolgt mit dem Loginnamen *pi* und dem Passwort *raspberry*.
Das Konfigurationsmenü kann auch in einer SSH-Sitzung mit dem Kommando *sudo raspi-config* aufgerufen werden.

Mit der Option *expand_rootfs* wird das Root-Filesystem an die Größe der SD-Karte angepasst. Die Aktualisierung des Systems erfolgt anschließend mit den Kommandos:

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

An dieser Stelle sind alle Voraussetzungen erfüllt, den Raspberry Pi zu erkunden. Wer sich von den Multimedia-Fähigkeiten der GPU überzeugen möchte spielt ein Video im H.264-Format, vielleicht *Big Bug Bunny*[1] ab.
Die Kommandozeile mit Ausgabe des Tons über den HDMI-Ausgang lautet dazu:

```
omxplayer --adev hdmi big_buck_bunny_1080p_h264.mov
```

3 Der Raspberry Pi im WLAN

Die *wheezy*-Distribution hat bereits Treiber und Firmwarepakete für gängige USB-WLAN Sticks und den *wpa_supplicant* für verschlüsselte WLANs an Board.

Nach anstecken des WLAN-Sticks geben das Kommando */usr/bin/lsusb* und ein Blick in */var/log/syslog* Auskunft über das USB-Gerät.
Sichtbare, Funknetzwerke zeigt das Kommando */sbin/iwlist scan* an. Die Konfiguration des *wpa_supplicant* wird durch das Kommando *wpa_passphrase* vereinfacht.

```
wpa_passphrase "essid" "passphrase" \
    >> /etc/wpa_supplicant/wpa_supplicant.conf
```

Die folgenden Zeilen in der Datei */etc/network/interfaces* konfigurieren das Netzwerkinterface *wlan0* für den automatisch Start

```
auto wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

und *sudo ifup wlan0* startet das Netzwerkinterface *wlan0*.

[1] `http://www.bigbuckbunny.org/`

Raspberry Pi als Access Point

Mit Hilfe des *hostapd*-Daemon lässt sich der Raspberry Pi in einen Access Point verwandeln. Bei der Auswahl des USB-WLAN Sticks ist zu beachten, dass der Treiber den Access Point-Mode[2] unterstützt.

Die erforderlichen Pakete werden mit dem folgendem Kommando installiert:

```
sudo apt-get install hostapd bridge-utils
```

Die Konfiguration des *hostapd* wird unter */etc/hostapd/hostapd.conf* abgelegt. Eine ausführlich kommentierte Vorlage befindet sich unter */usr/share/doc/hostapd/examples/hostapd.conf.gz*.
Das folgende Listing stellt eine minimale *hostapd*-Konfiguration bereit.

```
interface=wlan0
driver=nl80211
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
ssid=clttest
hw_mode=g
channel=1
auth_algs=3
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=GanzGeheim
wpa_key_mgmt=WPA-PSK
```

Die Konfiguration der Bridge zwischen kabelgebundenem Interface (*eth0*) und WLAN-Interface (*wlan0*) erfolgt mit dem *brctl*-Werkzeug. Dabei werden die Interfaces *eth0* und *wlan0* ohne IP-Adresse gestartet. Die Manangement IP-Adresse wird auf die Bridge (*br0*) konfiguriert.
Der Start des *hostapd* versetzt das WLAN-Interface in den Access Point Mode.

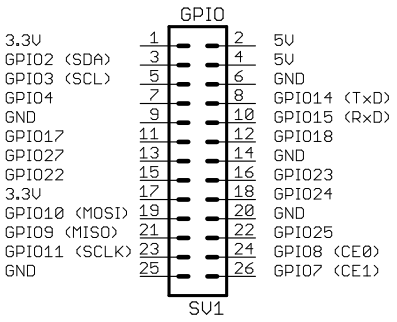
```
ifconfig eth0 0.0.0.0 up
ifconfig wlan0 0.0.0.0 up
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 wlan0
ifconfig br0 192.168.0.20 netmask 255.255.255.0 up
/usr/sbin/hostapd /etc/hostapd/hostapd.conf
```

[2] `http://wireless.kernel.org/en/users/Drivers`

4 Low-Level Peripherie

Die an einer Stiftleiste herausgeführten GPIO-Pins [4] machen den Raspberry Pi zu einer interessanten Plattform für hardwarenahe Projekte. Doch bevor diese Anschlüsse mit Tastern, LEDs, Sensoren oder anderen Bausteinen verbunden werden beachten Sie unbedingt die Hinweise zu Belastbarkeit und Spannungsfestigkeit.

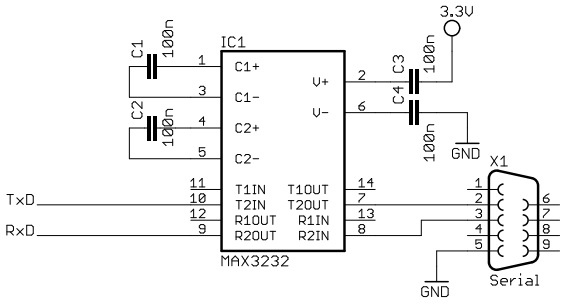
- 3.3V Spannung mit $50mA$ belastbar
- 5V wird vom Netzteil versorgt
- Signalpegel der GPIOs ist 3.3V *(*Eingänge sind nicht 5V-tolerant)
- Pinbelegung von der Board-Revision abhängig



Pinbelegung der Stiftleiste (Board Revision 2)

Serielle Schnittstelle

Soll der Raspberry Pi ohne Tastatur und Monitor betrieben werden erweist sich ein Zugang über die serielle Konsole als recht nützlich. Die erforderlichen Anschlüsse sind über die Pins *TxD* und *RxD* an der Stiftleiste herausgeführt. Für eine Verbindung mit der RS232-Schnittstelle eines PC ist jedoch eine Pegelanpassung erforderlich. Diese realisiert der IC *MAX 3232* (*3.3V-Typ*) mit minimaler Außenbeschaltung.

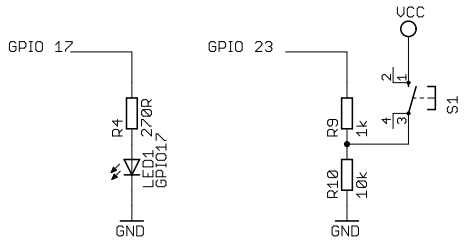


Als Terminalprogramme auf der PC-Seite eignen sich beispielsweise *minicom* oder *screen*. Die Datenrate ist auf 115200bit/s und das Datenformat auf 8N1 voreingestellt. Der Raspberry Pi gibt die Meldungen des Kernel auf der seriellen Schnittstelle aus und der init-Prozess bindet ein *getty* an die serielle Schnittstelle.

```
screen /dev/ttyUSB0 115200
```

GPIO-Pins

Die über die Stiftleiste herausgeführten GPIO-Pins sind frei programmierbar. Damit lässt sich jeder dieser Anschlüsse als Eingang oder Ausgang nutzen. Um bei offenen Eingängen einen definierten Pegel zu erhalten können integrierte Pullup-Widerstände softwareseitig aktiviert werden.
Zum Schutz vor Überlastung empfiehlt sich an den Ausgängen der Einsatz von Low-Current-LEDs oder Treibertransistoren.



Für die Programmierung der GPIO-Pins sind Module und Bibliotheken für viele Programmiersprachen verfügbar. Einfache Tests lassen sich in Shell-Skripten beispielsweise über das */sys-Filesystem* implementieren.

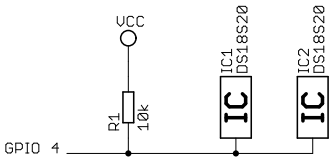
```
# GPIO 17 als Ausgang
echo "17" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio17/direction

# GPIO 17 auf High-Pegel
echo "1" > /sys/class/gpio/gpio17/value
```

Bussysteme

Bei der Anbindung von Bausteinen wie Speicher, Controller, Sensoren und dergleichen kommen oft Bussysteme zum Einsatz. Zu den typischen Vertretern gehören neben I^2C- und SPI- auch der 1-Wire-Bus. Die beiden erstgenannten Bussysteme werden hardwareseitig im SoC des Raspberry Pi unterstützt und deren Anschlüsse sind über die Stiftleiste herausgeführt.

1-Wire-Bus

Bausteine am 1-Wire-Bus werden neben der Stromversorgung mit nur einer Signalleitung angeschlossen. Die Datenübertragung erfolgt asynchron wobei der Raspberry Pi die Rolle des Masters übernimmt. Die Adressierung der Slaves basiert auf eindeutigen 64bit ROM-Adressen.

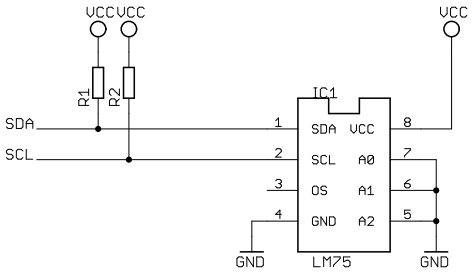


Der Kernel der *wheezy*-Distribution enthält Module für die Unterstützung des 1-Wire-Bus am *GPIO 4* und für verschiedene 1-Wire-Geräteklassen.
Das folgende Beispiel zeigt das Laden der Kernelmodule für den 1-Wire-Bus und einen Temperatursensor vom Typ *DS18S20*. Der 1-Wire-Bus präsentiert sich danach mit seinen Slaves im */sys-Filesystem*.

```
modprobe wire
modprobe w1-gpio
modprobe w1-therm
cat /sys/bus/w1/devices/22-00000022a744/w1_slave
 58 01 4b 46 7f ff 08 10 f9 : crc=f9 YES
 58 01 4b 46 7f ff 08 10 f9 t=21500          <-- 21.5 Grad
    Celsius
```

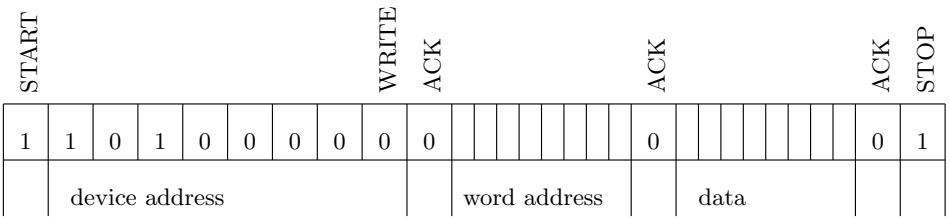
I²C-Bus

Der auch als Two-Wire-Interface (*TWI*) bezeichnete Bus wird häufig für die Kommunikation zwischen Bausteinen benutzt. Zwei Signalleitungen für Takt und Daten (*SCL* und *SDA*) bilden den Bus. Die Datenübertragung erfolgt synchron nach dem Master-Slave-Prinzip und unterstützt Taktraten zwischen 100 kHz und 3.4 MHz.



Die Pullup-Widerstände der Signalleitungen sind bereits auf dem Raspberry Pi-Board integriert.

Die Adressierung der Slaves erfolgt meist mit 8bit Adressen wobei das niederwertigste Bit über Lese- oder Schreiboperation auf dem Slave entscheidet. Das Beispiel zeigt die Kommunikationssequenz mit einem EEPROM-Baustein *24CXX* an der Geräteadresse $0x50$ beim Beschreiben einer Adresse mit einem Datenbyte.



In der *wheezy*-Distribution enthaltene Werkzeuge wie die *i2c-tools* benötigen als Schnittstelle zum I²C-Bus die zugehörigen Gerätedateien im */dev-Filesystem*.
Durch laden der Kernelmodule *i2c-bcm2708* und *i2c_dev* wird diese Schnittstelle angelegt. Damit die Module beim nächsten Start automatisch geladen werden entfernen Sie *i2c-bcm2708* aus der Blacklist */etc/modprobe.d/raspi-blacklist.conf* und fügen Sie *i2c_dev* der Liste der zu ladenden Module */etc/modules* hinzu.
Das Werkzeug *i2cdetect* scannt den I²C-Bus auf angeschlossene Slaves. Die Kommunikation mit den Slaves unterstützen Programme wie *i2cget*, *i2cdump* und *i2cset*.

```
apt-get install i2c-tools
modprobe i2c-bcm2708
modprobe i2c_dev
# scan bus 1
i2cdetect -y 1
```

Hinweis: Der Rapberry Pi Revision 2 stellt den I²C-Bus mit der *ID 1* an der Stiftleiste bereit. Die Adressierung der Slaves erfolgt *ohne* Schreib/Lese-Bit.

Das Python-Module *python-smbus*[3] setzt ebenfalls auf die I²C-Gerätedateien */dev/i2c-?* auf und stellt Funktionen für die Kommunikation mit I²C-Slaves bereit.

Erweiterte Funktionalität bietet das Python-Module *quick2wire*[4] mit Funktionen für die Programmierung von GPIO-Pins, den I²C- und SPI-Bus.

Mit *bcm2835*[5] ist eine C-Bibliothek für die Programmierung der Low-Level Peripherie des Raspberry Pi verfügbar. Dabei erfolgt der Hardwarezugriff über in den Speicher gemappte Register (*mmap()*).

[3] http://wiki.erazor-zone.de/wiki:linux:python:smbus:doc
[4] http://quick2wire.com/
[5] http://www.open.com.au/mikem/bcm2835/

Im folgenden Code-Beispiel wird ein EEPROM-Baustein *24CXX* mit der Geräteadresse $0x50$ programmiert. Die Funktion *write_i2c_block_data()* füllt den EEPROM mit der übergebenen Daten-Liste ab Adresse *reg*. Da der Schreibzyklus den EEPROM laut Datenblatt für maximal 5ms blockiert wartet ein *sleep*. Die Iterationsfunktion über die Länge der Daten-Liste liest nun byteweise ab Adresse *reg* und gibt die Werte wieder aus.

```
# EEPROM 24C02
import sys, time, smbus

# bus, address, register
bus = 1
address = 0x50
reg = 0x01
data  = [ 0x48, 0x61, 0x6c, 0x6c, 0x6f, 0x20, 0x43, 0x4c, 0x54 ]

eeprom = smbus.SMBus(bus)

print "I2C: write to device 0x%02X" % address
try:
  eeprom.write_i2c_block_data(address, reg, data)
except IOError, err:
  print "Error accessing I2C address: 0x%02X" % address
  exit(-1)
time.sleep(0.1)

print "I2C: read from device 0x%02X" % address
for i in range(len(data)):
  try:
    ret = eeprom.read_byte_data(address, reg+i)
  except IOError, err:
    print "Error accessing I2C address: 0x%02X" % address
    exit(-1)
  print "%c" % ret
```

Literatur

[1] *Raspberry Pi*
http://www.raspberrypi.org

[2] *Wikipedia Artikel über Raspberry Pi*
http://en.wikipedia.org/wiki/Raspberry_Pi

[3] *Raspbian Distribution*
http://www.raspbian.org/

[4] *Low-level peripherals*
http://elinux.org/RPi_Low-level_peripherals