Christian Immler

Raspberry Pi SERVERBUCH

Hannah Bernauer

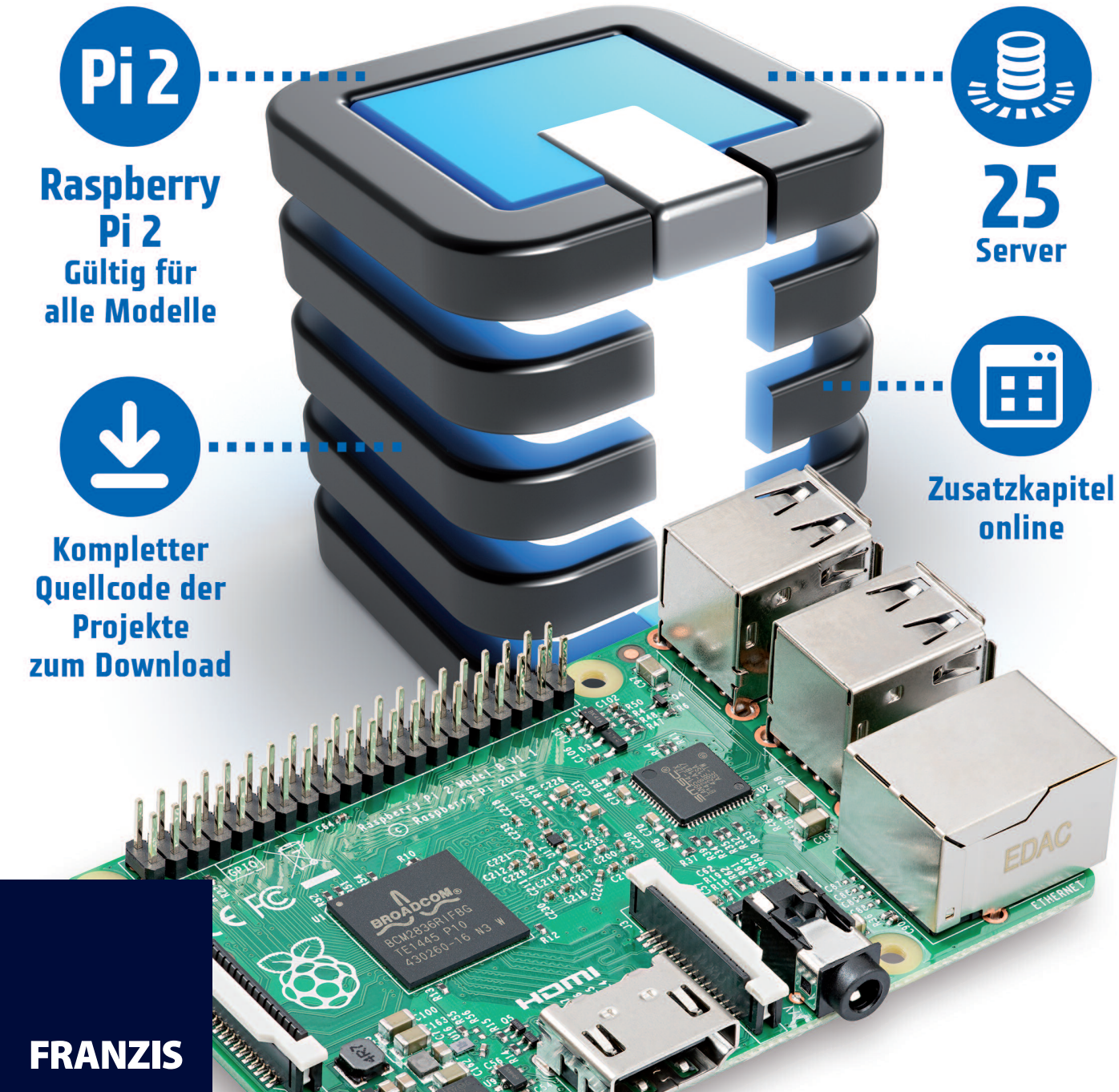
Pi 2

Raspberry Pi 2
Gültig für alle Modelle

25
Server

Kompletter Quellcode der Projekte zum Download

Zusatzkapitel online

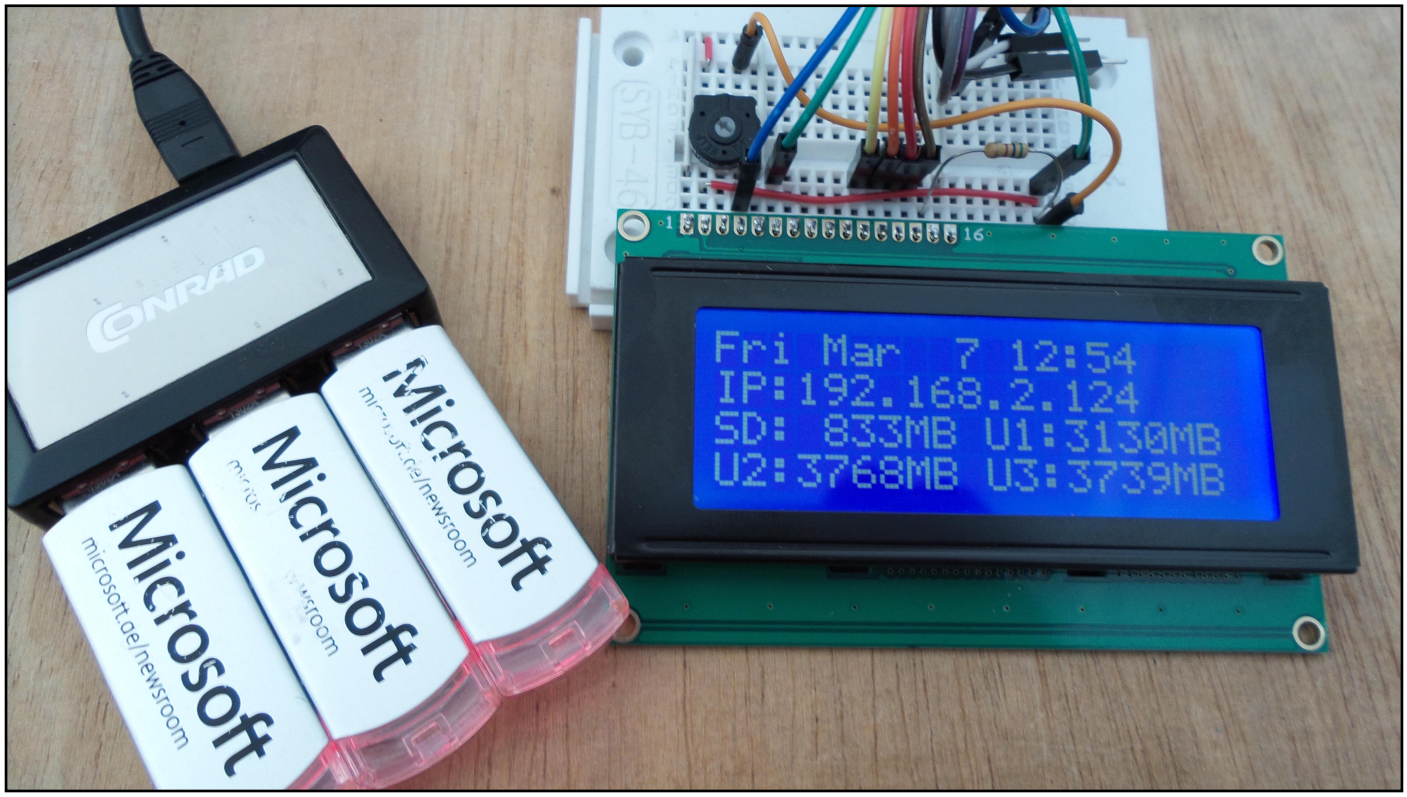


FRANZIS

Statusanzeige mit LCD-Display

Betreibt man einen Raspberry Pi headless ohne Monitor, ist es nicht immer ganz einfach, die IP-Adresse herauszufinden, die man für eine SSH-Verbindung zur Administration des Servers benötigt.

Mit einem LCD-Display am GPIO-Port des Raspberry Pi und einem Python-Script lässt sich diese Aufgabe elegant lösen. Die in diesem Kapitel beschriebene Schaltung und das Python-Skript bilden die Grundlage für die bei vielen Servern in diesem Buch erwähnten Statusanzeigen.

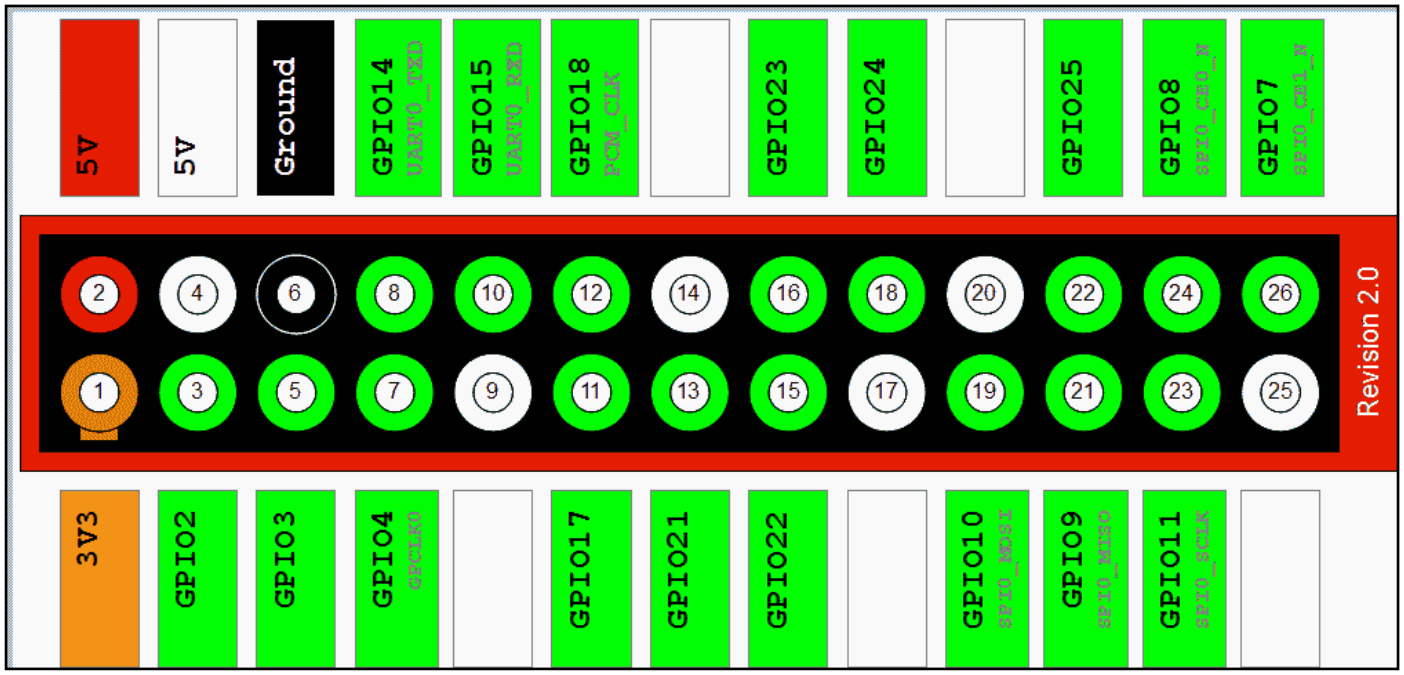


Ein handelsübliches LCD-Display zeigt über ein Python-Script die IP-Adresse, die aktuelle Serverzeit und weitere Informationen an.

Der GPIO-Anschluss am Raspberry Pi

Die 26-polige Stiftleiste in der Ecke des Raspberry Pi, als GPIO bezeichnet, bietet die Möglichkeit, direkt Hardware anzuschließen. Die englische Abkürzung *General Purpose Input Output* bedeutet auf Deutsch einfach *Allgemeine Ein- und Ausgabe*.

Von den 26 Pins lassen sich 17 wahlweise als Eingang oder Ausgang programmieren und so für vielfältige Hardwareerweiterungen nutzen. Die übrigen sind für Stromversorgung und andere Zwecke fest eingerichtet.



Belegung der GPIO-Schnittstelle. Die graue Linie oben und links bezeichnet den Rand der Platine. GPIO-Pin 2 liegt also ganz außen in der Ecke des Raspberry Pi.

Vorsicht

Verbinden Sie auf keinen Fall irgendwelche GPIO-Pins miteinander und warten ab, was passiert, sondern beachten Sie unbedingt folgende Hinweise:

Einige GPIO-Pins sind direkt mit Anschlüssen des Prozessors verbunden, ein Kurzschluss kann den Raspberry Pi komplett zerstören. Verbindet man über einen Schalter oder eine LED zwei Pins miteinander, muss immer ein Vorwiderstand dazwischen geschaltet werden.

Verwenden Sie für Logiksignale immer den Pin 1, der +3,3 V liefert und bis 50 mA belastet werden kann. Pin 6 ist die Masseleitung für Logiksignale.

Jeder GPIO-Pin kann als Ausgang (z. B. für LEDs) oder als Eingang (z. B. für Taster) geschaltet werden. GPIO-Ausgänge liefern im Logikzustand 1 eine Spannung von +3,3 V, im Logikzustand 0 0 Volt. GPIO-Eingänge liefern bei einer Spannung bis +1,7 V das Logiksignal 0, bei einer Spannung zwischen +1,7 V und +3,3 V das Logiksignal 1.

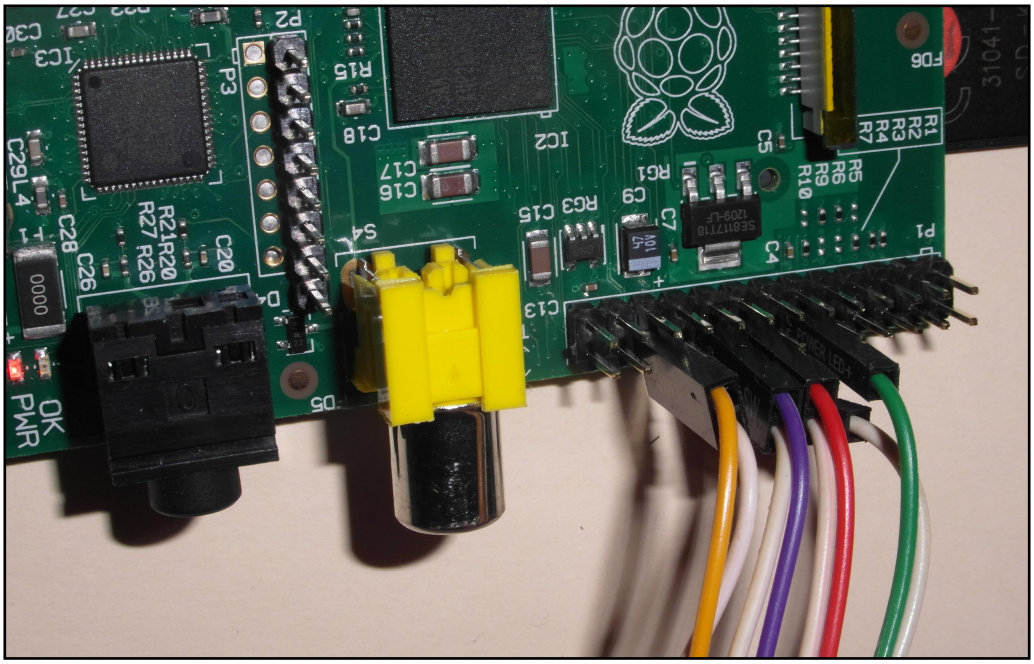
Pin 2 liefert +5 V zur Stromversorgung externer Hardware. Hier kann so viel Strom entnommen werden, wie das USB-Netzteil des Raspberry Pi liefert. Dieser Pin darf aber nicht mit einem GPIO-Eingang verbunden werden. Die Pins 9, 14, 17, 20, 25 sind für spätere Erweiterungen vorgesehen.

Voraussetzungen

Am einfachsten bauen Sie die Schaltung auf einer Steckplatine auf. Hier können elektronische Bauteile direkt in ein Lochraster mit Standardabständen eingesteckt werden, ohne dass man löten muss. Bei diesen Platinen sind die äußeren Längsreihen mit Kontakten alle miteinander verbunden. Diese Kontaktreihen werden üblicherweise als Plus- und Minuspol zur Stromversorgung der Schaltung genutzt. Die anderen Kontaktreihen sind jeweils quer miteinander verbunden, wobei in der Mitte der Platine eine Lücke ist.

- Steckplatine
- LCD-Display 20x4 HD44780-kompatibel
- Pfostenverbinder 16-polig
- 560 Ohm Vorwiderstand (nur bei Display mit Hintergrundbeleuchtung)
- 8 Verbindungskabel (m/w)
- Kurze Drahtbrücken
- 15 kOhm Potentiometer zur Kontrasteinstellung (optionaler Luxus, bei manchen Displays erforderlich)

Alternativ können Sie die Schaltung auch auf einer kleinen Experimentierplatine oder ganz „fliegend" zusammenlöten. Zur Verbindung des Raspberry Pi mit der Steckplatine oder einzelnen elektronischen Bauteilen können Sie den Kabelstrang für die Front-LEDs und Reset-Taster aus einem ausgedienten PC-Gehäuse verwenden. Die Kontakte, mit denen diese Kabel am Motherboard angeschlossen sind, passen auf den GPIO-Port.

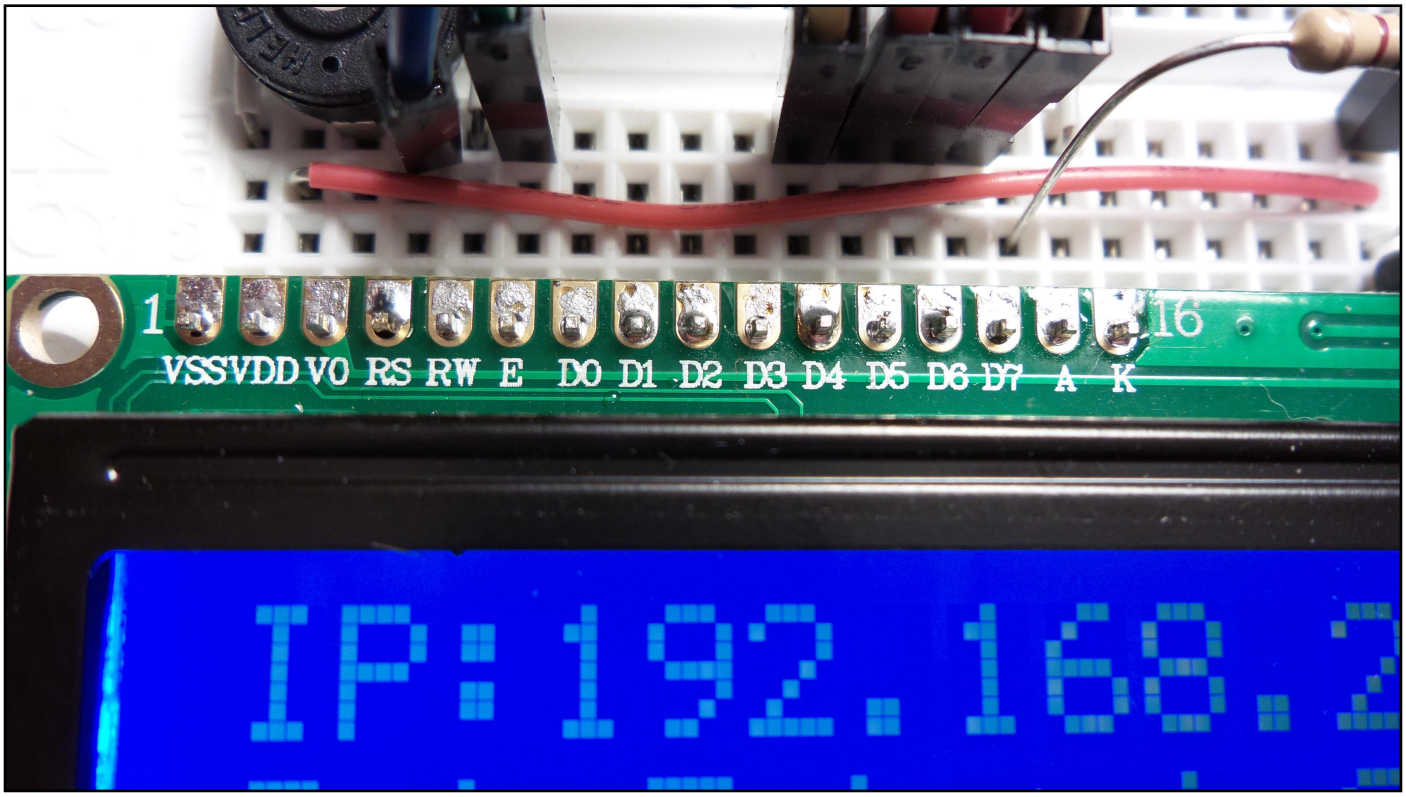


Kabel aus einem ausgedienten PC lassen sich gut für die GPIO-Ports verwenden.

So funktioniert das LCD-Display

Zweizeilige LCD-Displays, wie sie für wenige Euro bei diversen Onlinehändlern erhältlich sind, sind fast alle zum Quasi-Standard HD44780 kompatibel. Dabei handelt es sich um die Bezeichnung des in den Displays eingebauten Controllers, der Zeichen in ein bis vier Zeilen darstellt, ohne dass sich der Benutzer um die Ansteuerung der einzelnen Pixel zu kümmern braucht. Die vierzweiligen Displays sind zwar nicht ganz so weit verbreitet, arbeiten aber nach dem gleichen Standard. Für das Programm verwenden wir ein bekanntes Display vom Typ 2004A, wie sie bei chinesischen Onlinehändlern für wenige Euro zu bekommen sind. Das 20x4-Display hat vier Zeilen mit je 20 Zeichen. Die Anschlussbelegung entspricht einem 16x2-Standarddisplay. Intern werden die erste und dritte Zeile sowie die zweite und vierte wie je eine 40 Zeichen lange Zeile behandelt.

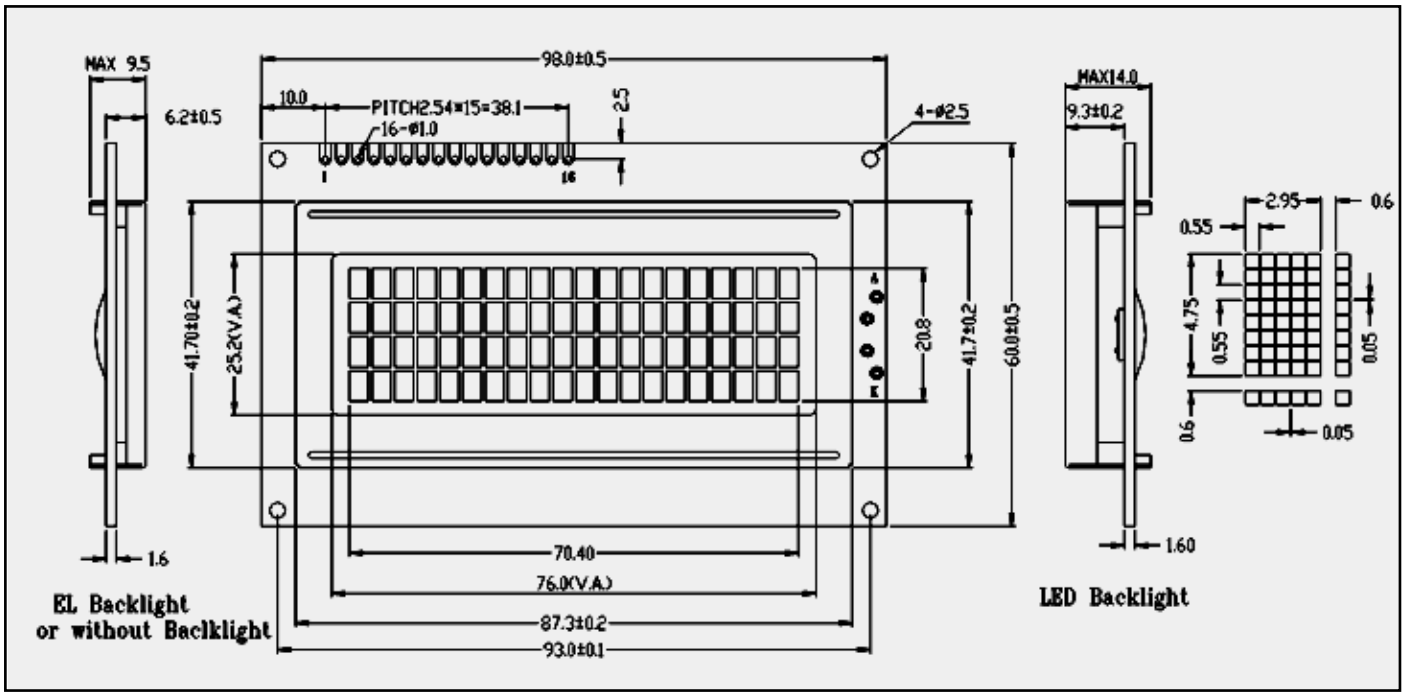
Diese Displays haben eine 16-polige Anschlussleiste, an denen Kabel oder ein Pfostenverbinder zum Aufstecken auf eine Steckplatine angelötet werden können. Displays ohne Hintergrundbeleuchtung haben nur 14 Anschlüsse.



Die Anschlüsse sind auf den meisten Displays genau beschriftet.

In der Abbildung ist ein Pfostenverbinder am Display angelötet, mit dem sich dieses direkt auf die Steckplatine stecken lässt.

Wenn die vollständige Beschriftung auf dem Display fehlt, sind zumindest Pin 1 oder Pin 16 gekennzeichnet. Im Zweifelsfall sehen Sie sich das Datenblatt zum Display an. Bei den meisten Displays sind die Pins fortlaufend angeordnet, in einigen Fällen kommen sie aber auch in der Reihenfolge: *14...1,16,15 vor*.



Ausschnitt aus dem Datenblatt eines kompatiblen Displays

Pinbelegung eines HD44780 kompatiblen Displays

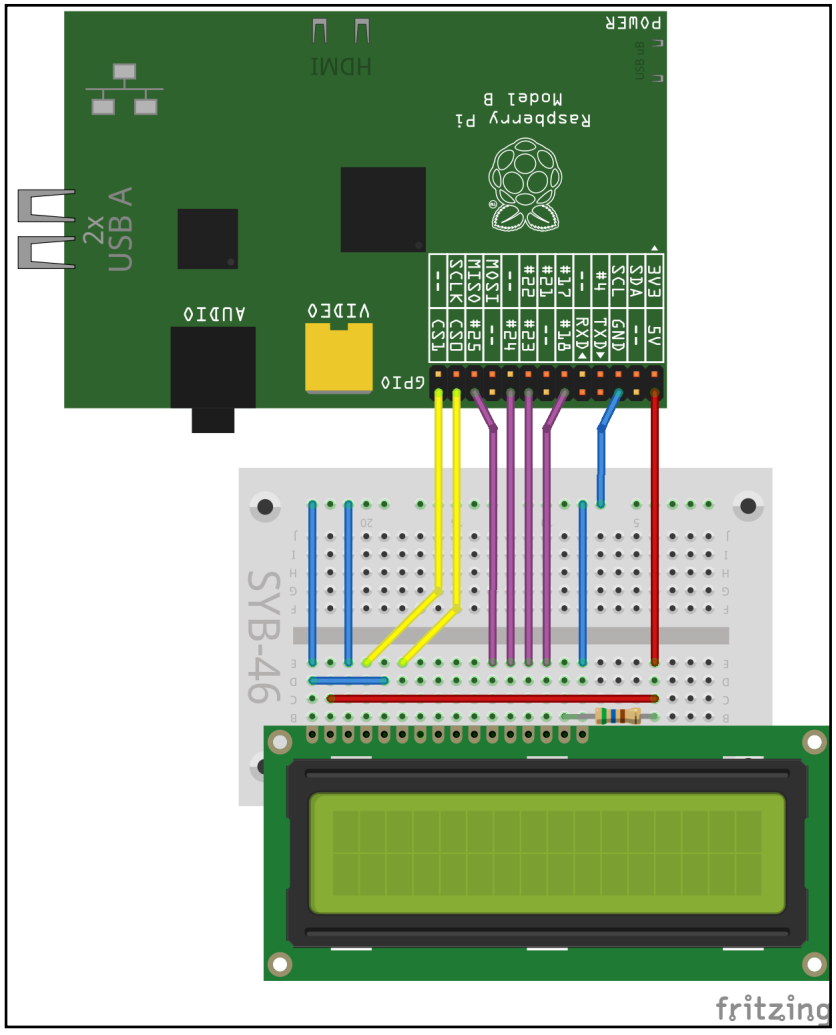
Pin	**Funktion**	***Beschreibung***
1	VSS	*Stromversorgung Masseleitung 0 V*
2	VDD	*Stromversorgung +5 V*
3	V0	*Kontrasteinstellung, 0 V ... 5 V*
4	RS	*Register Select*
5	RW	*Read/Write, wenn vom Display nichts ausgelesen wird, mit 0 V verbinden*
6	E	*Enable (Umschaltsignal)*
7	D0	*Datenbit 0 (im 4-Bit-Modus nicht benötigt)*
8	D1	*Datenbit 1 (im 4-Bit-Modus nicht benötigt)*
9	D2	*Datenbit 2 (im 4-Bit-Modus nicht benötigt)*
10	D3	*Datenbit 3 (im 4-Bit-Modus nicht benötigt)*
11	D4	*Datenbit 04*
12	D5	*Datenbit 05*
13	D6	*Datenbit 06*

Pin	Funktion	***Beschreibung***
14	D7	*Datenbit 07*
15	A	*Hintergrundbeleuchtung, Vorwiderstand 560 Ohm erforderlich*
16	K	*Hintergrundbeleuchtung Masseleitung*

Die Displays unterstützen zwei Modi zur Übertragung der Daten. Im 8-Bit-Modus kann ein komplettes Zeichen auf einmal übertragen werden. Meistens wird aber der 4-Bit-Modus verwendet, da dieser vier Ports am sendenden Gerät spart. Hier werden die Daten eines Zeichens in zwei Blöcken hintereinander übertragen.

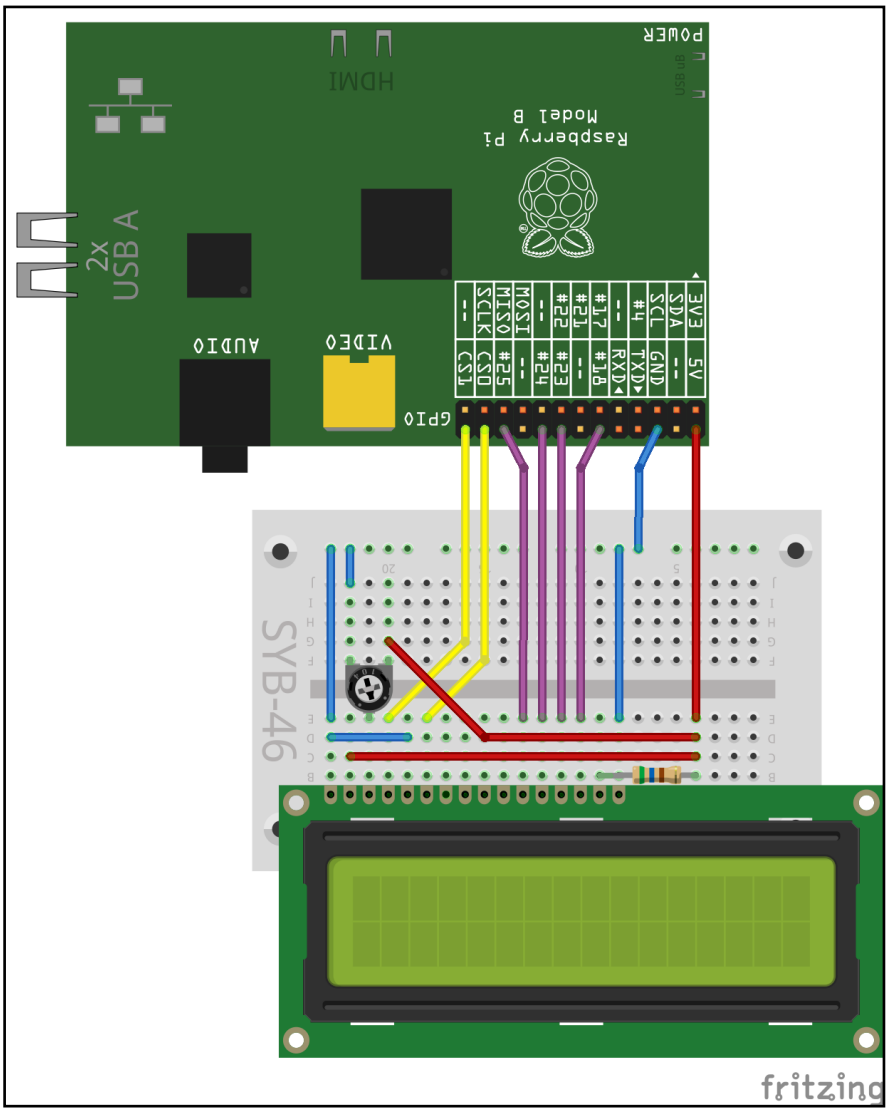
Aufbau der Schaltung

Zusätzlich zu den Verbindungskabeln zum Raspberry Pi benötigen Sie fünf Drahtbrücken unterschiedlicher Länge, mit denen mehrere Pins für die gemeinsame Stromversorgung und Masseleitung zusammengeführt werden.



Gelb: Steuerleitungen
Violett: Datenleitungen
Rot: +5 V
Blau: Masse

Vor der LED der Hintergrundbeleuchtung – Pin 15 - wird ein 560-Ohm.Widerstand als Schutz gegen Überlastung vorgeschaltet. Bei Displays ohne Hintergrundbeleuchtung fallen dieser Widerstand sowie die Masseleitung an Pin 16 weg.



Das Potentiometer zur Kontrastregelung ersetzt die Drahtbrücke zwischen 0 V und Pin 3.

Sollte das Display schwer zu erkennen sein, was bei Modellen mit sehr heller Hintergrundbeleuchtung oft der Fall ist, bauen Sie ein 15-kOhm-Potentiometer ein, das dem Pin 3 eine Spannung zwischen +5 V und 0 V zuführt. Beachten Sie dabei das Datenblatt des Displays. Bei einigen Displays darf die maximale Spannung am Kontrast-Pin nur weniger als +5 V betragen. Drehen Sie also das Potentiometer nie ganz auf.

Alle Anschlüsse genau überprüfen

Bevor Sie den Raspberry Pi und damit die Schaltung mit Strom versorgen, prüfen Sie alle Anschlussdrähte noch einmal genau. Ein falsch angeschlossene 5-V-Leitung kann das Display beschädigen.

Python-Script für die Statusanzeige

Zur Ansteuerung der GPIO-Ports mit dem LCD-Display verwenden wir ein Python-Script. Python ist eine besonders einfach zu erlernende Programmiersprache, die auf dem Raspberry Pi vorinstalliert ist und ihm auch einen Teil seines Namens gab.

Das Wort Pi steht für Python Interpreter. Python überzeugt durch seine klare Struktur, die einen einfachen Einstieg in das Programmieren erlaubt, ist aber auch eine ideale Sprache, um „mal schnell" etwas zu automatisieren, was man sonst von Hand erledigen würde.

Da keine Variablendeklarationen, Typen, Klassen oder komplizierte Regeln zu beachten sind, macht das Programmieren wirklich Spaß.

Haben Sie am Raspberry Pi Monitor und Tastatur angeschlossen, können Sie die mitgelieferte Entwicklungsumgebung IDLE verwenden.

Im Headless-Betrieb erstellen Sie ein Python-Script auf dem PC mit einem beliebigen Texteditor und kopieren es dann per SFTP in das Verzeichnis `/home/pi` auf dem Raspberry Pi. Achten Sie bei Windows-Editoren besonders darauf, dass die Texte unformatiert und mit UTF-8 Codierung gespeichert werden.

Python Flashcards

Python ist die ideale Programmiersprache, um den Einstieg in die Programmierung zu erlernen. Nur die Syntax und die Layoutregeln sind etwas gewöhnungsbedürftig. Zur Hilfestellung im Programmieralltag werden die wichtigsten Syntaxelemente der Sprache Python auf den Python Flashcards von David Whale in Form kleiner „Spickzettel" kurz beschrieben.

Was es damit genau auf sich hat, finden Sie bei: bit.ly/pythonflashcards. Diese Flashcards erklären nicht die technischen Hintergründe, sondern beschreiben nur anhand ganz kurzer Beispiele die Syntax, also, wie etwas gemacht wird.

BEDINGUNGEN 8

```
a=1
if a==1:
     print "gleich"
if a!=1:
     print "nicht gleich"
if a<1:
     print "kleiner"
if a>1:
     print "größer"
if a<=1:
     print "kleiner oder gleich"
if a>=1:
     print "größer oder gleich"
```

python(1) V2 (deutsch) - softwarehandbuch.de

IF ELSE 9

```
alter=10
if alter>17:
     print "Du darfst Auto fahren"
else:
     print "Du bist nicht alt genug"
```

python(1) V2 (deutsch) - softwarehandbuch.de

IF ELIF ELSE 10

```
alter=10
if alter<4:
     print "Du bist in der Kinderkrippe"
elif alter<6:
     print "Du bist im Kindergarten"
elif alter<10:
     print "Du bist in der Grundschule"
elif alter<19:
     print "Du bist im Gymnasium"
else:
     print "Du hast die Schule verlassen"
```

python(1) V2 (deutsch) - softwarehandbuch.de

AND/OR BEDINGUNGEN 11

```
a=1
b=2
if a>0 and b>0:
     print "Beide sind nicht Null"

if a>0 or b>0:
     print "Mindestens eine ist nicht Null"
```

python(1) V2 (deutsch) - softwarehandbuch.de

Ausschnitt aus den Python Flashcards

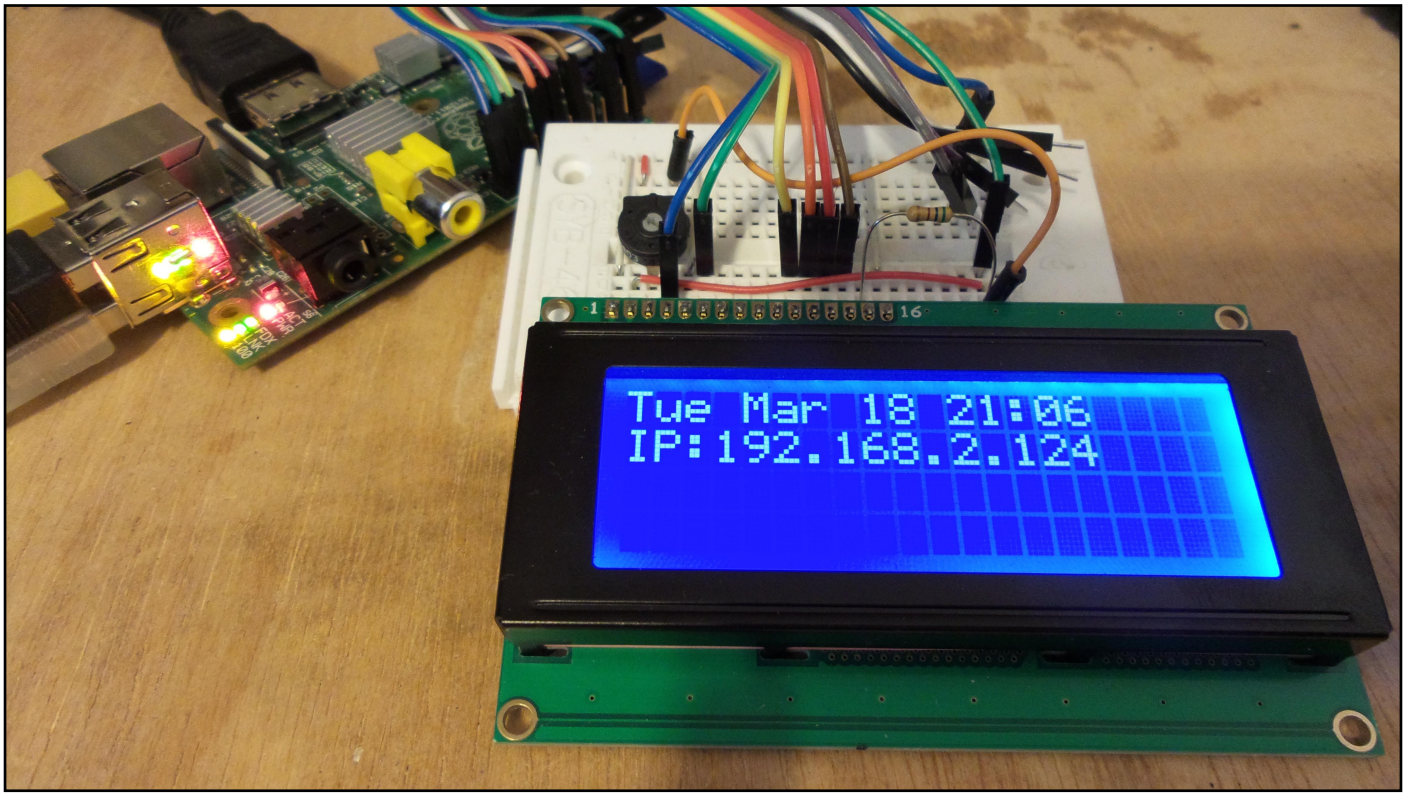
Schreiben Sie folgendes Python-Script auf dem PC oder laden Sie sich die fertige Datei *serverstatus0.py* bei *www.buch.cd* herunter.

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time, os, subprocess

LCD_RS = 7
LCD_E  = 8
LCD_D4 = 25
LCD_D5 = 24
LCD_D6 = 23
LCD_D7 = 18
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(LCD_E,  GPIO.OUT)
GPIO.setup(LCD_RS, GPIO.OUT)
GPIO.setup(LCD_D4, GPIO.OUT)
GPIO.setup(LCD_D5, GPIO.OUT)
GPIO.setup(LCD_D6, GPIO.OUT)
GPIO.setup(LCD_D7, GPIO.OUT)


LCD_WIDTH = 40
LCD_W2 = LCD_WIDTH / 2
LCD_LINE_1 = 0x80
LCD_LINE_2 = 0xC0
LCD_CHR = True
LCD_CMD = False
E_PULSE = 0.00005
E_DELAY = 0.00005
w = 2


def lcd_enable():
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)


def lcd_byte(bits, mode):
  GPIO.output(LCD_RS, mode)
  GPIO.output(LCD_D4, bits&0x10==0x10)
  GPIO.output(LCD_D5, bits&0x20==0x20)
  GPIO.output(LCD_D6, bits&0x40==0x40)
  GPIO.output(LCD_D7, bits&0x80==0x80)
  lcd_enable()
  GPIO.output(LCD_D4, bits&0x01==0x01)
  GPIO.output(LCD_D5, bits&0x02==0x02)
  GPIO.output(LCD_D6, bits&0x04==0x04)
  GPIO.output(LCD_D7, bits&0x08==0x08)
  lcd_enable()
```

```
def lcd_string(message):
  message = message.ljust(LCD_WIDTH,“ „)
  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

def lcd_anzeige():
    z1 = zeile1.ljust(LCD_W2,“ „) + zeile3.ljust(LCD_W2,“ „)
    z2 = zeile2.ljust(LCD_W2,“ „) + zeile4.ljust(LCD_W2,“ „)
    lcd_byte(LCD_LINE_1, LCD_CMD)
    lcd_string(z1)
    lcd_byte(LCD_LINE_2, LCD_CMD)
    lcd_string(z2)
    time.sleep(w)

LCD_INIT = [0x33, 0x32, 0x28, 0x0C, 0x06, 0x01]
for i in LCD_INIT:
  lcd_byte(i,LCD_CMD)

try:
  while True:
    zeile1 = time.asctime()[0:16]
    zeile2 = „IP:“ + subprocess.check_output([„hostname“,“-I“])[:-2]
    zeile3 = „“
    zeile4 = „“
    lcd_anzeige()

except KeyboardInterrupt:
  GPIO.cleanup()
```

Kopieren Sie diese Datei in das Verzeichnis `/home/pi` auf dem Raspberry Pi. Dieses Basisprogramm nutzt nur die oberen beiden Zeilen des Displays und zeigt IP-Adresse und Serverzeit. Probieren Sie das Programm einfach gleich aus, Sie werden sehen, es funktioniert.




```
sudo python serverstatus.py
```

Die GPIO-Ports des Raspberry Pi sind wie unter Linux für alle Geräte üblich, wie Dateien in die Verzeichnisstruktur eingebunden. Zum Zugriff auf diese Dateien braucht man root-Rechte. Starten Sie deshalb den Python-Interpreter mit `sudo`.

Das Display zeigt IP-Adresse, Datum und Uhrzeit über ein Python-Script an.

So funktioniert es

Dass das Script funktioniert, lässt sich einfach ausprobieren. Jetzt stellen sich natürlich die Fragen: Was passiert im Hintergrund? Was bedeuten die einzelnen Programmzeilen?

```
#!/usr/bin/python
```

Diese Zeile steht am Anfang (fast) jedes Python-Scripts, um die Datei als solches zu identifizieren.

Sie wäre zwar in diesem Fall nicht unbedingt nötig, aber gewöhnen Sie sich an, sie immer als erste Zeile in ein Python-Script zu schreiben.

```
import RPi.GPIO as GPIO
import time, os, subprocess
```

Der Standard-Funktionsumfang von Python kann durch Einbinden zusätzlicher Bibliotheken erweitert werden. In diesem Fall handelt es sich um die Bibliotheken:

- `RPi.GPIO` - Zugriff auf die GPIO-Schnittstelle
- `time` - Funktionen für Datum und Zeit
- `os, subprocess` - Zugriff auf Betriebssystemfunktionen

```
LCD_RS = 7
LCD_E  = 8
LCD_D4 = 25
LCD_D5 = 24
LCD_D6 = 23
LCD_D7 = 18
```

Diese Zeilen definieren die Konstanten, in denen die Nummern der GPIO-Ports für Steuerleitungen und Datenleitungen des Displays gespeichert sind.

```
GPIO.setmode(GPIO.BCM)
```

Am Anfang jedes Programms muss definiert werden, wie die GPIO-Ports bezeichnet sind. Üblicherweise verwendet man die Standardnummerierung BCM.

Nummerierung der GPIO-Ports

Die Bibliothek `RPi.GPIO` unterstützt zwei verschiedene Methoden zur Bezeichnung der Ports. Im Modus `BCM` werden die bekannten GPIO-Portnummern verwendet, die auch auf Kommandozeilenebene oder in Shell-Scripten verwendet werden. Im alternativen Modus `BOARD` entsprechen die Bezeichnungen den Pin-Nummern auf der Raspberry Pi-Platine von 1 bis 26.

```
GPIO.setup(LCD_E,  GPIO.OUT)
GPIO.setup(LCD_RS, GPIO.OUT)
GPIO.setup(LCD_D4, GPIO.OUT)
GPIO.setup(LCD_D5, GPIO.OUT)
GPIO.setup(LCD_D6, GPIO.OUT)
GPIO.setup(LCD_D7, GPIO.OUT)
```

Die Funktion `GPIO.setup` initialisiert einen GPIO-Port als Ausgang oder als Eingang. Der erste Parameter bezeichnet den Port. Der zweite Parameter kann entweder `GPIO.OUT` für einen Ausgang oder `GPIO.IN` für einen Eingang sein. Die Portnummern werden hier über die zuvor definierten Variablen angegeben. Diese Methode hat den Vorteil, dass die tatsächlichen GPIO-Ports nur an dieser einen Stelle im Programm auftauchen. So können Sie das Programm ganz einfach umbauen, wenn Sie andere GPIO-Ports nutzen möchten.

```
LCD_WIDTH = 40
LCD_W2 = LCD_WIDTH / 2
LCD_LINE_1 = 0x80

LCD_LINE_2 = 0xC0
LCD_CHR = True
LCD_CMD = False
E_PULSE = 0.00005
E_DELAY = 0.00005
```

Diese Zeilen definieren weitere Konstanten, die im Programm verwendet werden.

Konstante	Bedeutung
`LCD_WIDTH`	Länge der logischen Zeile in Zeichen
`LCD_W2`	Länge der physischen Zeile in Zeichen
`LCD_LINE_1`	Speicheradresse der 1. Zeile
`LCD_LINE_2`	Speicheradresse der 2. Zeile
`LCD_CHR`	Schaltet das Display auf Pin RS in den Zeichenmodus
`LCD_CMD`	Schaltet das Display auf Pin RS in den Steuerungsmodus
`E_PULSE`	Dauer eines Steuerimpulses auf Pin E
`E_DELAY`	Wartezeit zwischen zwei Steuerimpulsen auf Pin E

Adressierung vierzeiliger Displays

Um eine möglichst hohe Kompatibilität zum verbreiteten Standard der zweizeiligen Displays zu erreichen, verwenden vierzeilige Displays intern auch nur zwei Zeilen. Die dritte Zeile des 20 Zeichen breiten Displays entspricht der zweiten Hälfte einer 40 Zeichen langen ersten Zeile. Genauso entspricht die vierte Zeile intern der zweiten Hälfte der zweiten Zeile.

```
w = 2
```

Die Variable `w` enthält die Wartezeit, bis der Systemstatus erneut abgefragt wird. Kürzere Abfrageraten zeigen eine veränderung schneller, lasten das System aber stärker aus. Standardmäßig ist eine Aktualisierung alle 2 Sekunden eingestellt.

```
def lcd_enable():
```

Danach wird eine Funktion definiert, die später im Programm aufgerufen wird. Diese sendet einen kurzen Steuerimpuls an den Pin E (*enable*). Damit schaltet das Display im 4-Bit-Modus zwischen den oberen und den unteren Bits eines darzustellenden Bytes um.

Einrückungen sind in Python wichtig

In den meisten Programmiersprachen werden Funktionsblöcke, Programmschleifen oder Entscheidungen eingerückt, um den Programmcode übersichtlicher zu machen. In Python dienen diese Einrückungen nicht nur der Übersichtlichkeit, sondern sind auch für die Programmlogik zwingend nötig. Dafür braucht man hier keine speziellen Satzzeichen, um Schleifen oder Entscheidungen zu beenden.

```
    time.sleep(E_DELAY)
```

Zuerst wird eine kurze, in `E_DELAY` gespeicherte Zeit (0,05 ms) gewartet. Das Display hat eine gewisse Trägheit. Man kann nicht unmittelbar nach dem letzten Zeichen auf das andere Halbbyte umschalten.

```
    GPIO.output(LCD_E, True)
```

Jetzt wird auf den Pin E das Logiksignal `True` ausgegeben. GPIO-Ausgänge können nur die digitalen Werte `True` oder `False` annehmen.

```
    time.sleep(E_PULSE)
    GPIO.output(LCD_E, False)
```

Das Signal liegt über die in `E_PULSE` gespeicherte Zeit am Pin E an, danach wird der Pin wieder auf `False` zurückgesetzt.

```
    time.sleep(E_DELAY)
```

Das Programm wartet noch die in `E_DELAY` gespeicherte Zeit ab, bevor die Funktion beendet wird, damit danach sofort Daten an das Display gesendet werden können.

```
def lcd_byte(bits, mode):
```

Diese Funktion sendet ein Byte an das Display. Dieses Byte kann ein Zeichen oder ein Steuerbefehl sein. Im Parameter `bits` bekommt die Funktion das zu sendende Byte, der Parameter `mode` gibt an, ob es sich um ein Zeichen oder um einen Steuerungsbefehl handelt. Die beiden Werte, die `mode` annehmen kann, sind in den Konstanten `LCD_CHR` (Zeichen) und `LCD_CMD` (Steuerungsbefehl) festgelegt.

```
    GPIO.output(LCD_RS, mode)
```

Der zu verwendende Modus wird auf den Pin RS (*Register Select*) ausgegeben. Damit wird das Display in den Zeichenmodus oder den Steuerungsmodus geschaltet.

```
    GPIO.output(LCD_D4, bits&0x10==0x10)
    GPIO.output(LCD_D5, bits&0x20==0x20)
    GPIO.output(LCD_D6, bits&0x40==0x40)
    GPIO.output(LCD_D7, bits&0x80==0x80)
```

Das Programm verwendet den 4-Bit-Modus des Displays. Zunächst werden die oberen 4 Bit des zu sendenden Bytes auf den Datenleitungen ausgegeben. Dazu wird der Python-Operator `&` (bitweise UND) verwendet. Nacheinander wird für jedes der vier Bits geprüft, ob es `1` oder `0` ist. Das Ergebnis der Prüfung ist entsprechend `True` oder `False`. Dieser Wert wird dann auf der jeweiligen Datenleitung ausgegeben.

Bit-Nr.	4 high	3 high	2 high	1 high	4 low	3 low	2 low	*1 low*
Pin	D7	D6	D5	D4	D7	D6	D5	*D4*
Binär	1000 0000	0100 0000	0010 0000	0001 0000	0000 1000	0000 0100	0000 0010	*0000* *0001*
Dezimal	128	64	32	16	8	4	2	*1*
Hex	*0x80*	*0x40*	*0x20*	*0x10*	*0x08*	*0x04*	*0x02*	*0x01*

```
    lcd_enable()
```

Diese zuvor definierte Funktion schaltet jetzt das Display auf das untere Halbbyte um.

```
    GPIO.output(LCD_D4, bits&0x01==0x01)
    GPIO.output(LCD_D5, bits&0x02==0x02)
    GPIO.output(LCD_D6, bits&0x04==0x04)
    GPIO.output(LCD_D7, bits&0x08==0x08)
```

Nach dem gleichen Schema werden jetzt die unteren 4 Bit des zu sendenden Bytes auf den Datenleitungen ausgegeben …

```
    lcd_enable()
```

… und danach wieder auf das obere Halbbyte umgeschaltet.

```
def lcd_string(message):
```

Diese Funktion schreibt eine Zeichenfolge auf das Display. Diese Zeichenfolge wird vom aufrufenden Programm im Parameter `message` an die Funktion übergeben.

```
    message = message.ljust(LCD_WIDTH," ")
```

Die Zeichenkette wird bis auf die Länge des Displays rechts mit Leerzeichen aufgefüllt. Die Länge des Displays, in unserem Fall 40 Zeichen, ist in der Konstante `LCD_WIDTH` gespeichert. Die Methode `.ljust` kann in jeder Zeichenkette angewendet werden, der zweite Parameter gibt das Zeichen an, mit dem kürzere Zeichenketten aufgefüllt werden.

```
    for i in range(LCD_WIDTH):
      lcd_byte(ord(message[i]),LCD_CHR)
```

Diese Schleife läuft so oft durch, wie das Display Zeichen hat. In jedem Durchlauf wird ein Zeichen der Zeichenkette `message` zunächst mit der Python-Funktion `ord()` in seinen ASCII-Zahlenwert umgewandelt und dann im Zeichenmodus (`LCD_CHR`) über die zuvor definierte Funktion `lcd_byte()` auf das Display ausgegeben.

```
def lcd_anzeige():
    z1 = zeile1.ljust(LCD_W2," ") + zeile3.ljust(LCD_W2," ")
    z2 = zeile2.ljust(LCD_W2," ") + zeile4.ljust(LCD_W2," ")
    lcd_byte(LCD_LINE_1, LCD_CMD)
    lcd_string(z1)
    lcd_byte(LCD_LINE_2, LCD_CMD)
    lcd_string(z2)
    time.sleep(w)
```

Die Darstellung auf dem Display wird von der Funktion `lcd_anzeige()` gesteuert, die aus vier einzeln definierten Textzeilen `zeile1` bis `zeile4` in den Variablen `z1` und `z2` die Zeichenketten für die beiden logischen, 40 Zeichen langen Zeilen für die Anzeige zusammensetzt. `z1` enthält die Anzeigezeilen 1 und 3, `z2` die Anzeigezeilen 2 und 4. Die vier Zeichenketten werden alle mit Hilfe der Methode `.ljust()` auf eine Länge von je 20 Zeichen gebracht. Sollte die erste oder zweite Zeile kürzer sein, passt sonst der Zeilenumbruch am Ende der Anzeige nicht.

Zuletzt werden die beiden Zeichenketten in die beiden logischen Zeilen des Displays geschrieben und danach die in der Variable `w` festgelegte Zeit gewartet. Danach kann das Programm die Daten für die zweite Seite ermitteln und die Funktion erneut aufrufen.

```
    lcd_byte(LCD_LINE_1, LCD_CMD)
```

Dabei wird zuerst die Adresse der ersten Zeile des Displays im Steuerungsmodus (`LCD_CMD`) auf das Display ausgegeben. Damit wird festgelegt, dass die folgenden Ausgaben im Zeichenmodus auf der oberen Zeile erscheinen.

```
    lcd_string(z1)
```

Danach wird die Zeichenkette `z1` auf das Display geschrieben. Die erste Hälfte erscheint auf der ersten Zeile, die zweite Hälfte auf der dritten Zeile.

```
    lcd_byte(LCD_LINE_2, LCD_CMD)
```

Diese Zeile schaltet auf die untere Zeile des Displays um.

Am Ende wartet die Funktion die voreingestellte Zeit und springt danach zum Hauptprogramm zurück, wo neue Werte für die Ausgabe ermittelt werden.

Nachdem die vier Funktionen `lcd_enable()`, `lcd_byte()`, `lcd_string()` und `lcd_anzeige()` definiert sind, startet das eigentliche Programm.

```
LCD_INIT = [0x33, 0x32, 0x28, 0x0C, 0x06, 0x01]
```

Bevor das Display verwendet werden kann, muss es zunächst initialisiert werden. Eine Folge von Initialisierungskommandos bringt es in einen eindeutig definierten Zustand und setzt es auf den 4-Bit-Modus.

```
for i in LCD_INIT:
  lcd_byte(i,LCD_CMD)
```

Diese Schleife schreibt nacheinander die in der Liste `LCD_INIT` eingetragenen Initialisierungskommandos im Steuerungsmodus (`LCD_CMD`) auf das Display. Zur Datenübertragung wird die weiter oben definierte Funktion `lcd_byte()` verwendet.

```
try:
  while True:
```

Dieses Konstrukt startet eine Endlosschleife, die so lange wiederholt wird, bis die weiter unten bei `except` festgelegte Abbruchbedingung eintritt.

Im Hauptprogramm tauchen neue Elemente auf. Innerhalb der Endlosschleife werden in vier Variablen, `zeile1` bis `zeile4` die Zeichenketten für die vier Anzeigezeilen der ersten Seite zusammengesetzt.

```
    zeile1 = time.asctime()[0:16]
```

`zeile1` enthält das aktuelle Datum und die Uhrzeit. Dazu werden nur die ersten 16 Zeichen der Ausgabe von `time.asctime()` verwendet. Diese Funktion liefert Datum und Uhrzeit in einer Zeichenfolge, z. B.: `Mon Mar 31 22:01:59 2014`. Die ersten 16 Stellen dieser Zeichenfolge zeigen Wochentag, Monat, Tag, Stunden und Minuten. Die Anzeige der Sekunden schneiden wir ab, da für eine Sekundenanzeige in Echtzeit die Schleife sehr oft durchlaufen werden müsste, was sinnlos Performance kostet, die dem Server verloren geht. Da die verwendeten Systemabfragen unter Umständen einige Sekundenbruchteile dauern können, kann es bei einem Schleifendurchlauf pro Sekunde passieren, dass in der Anzeige gerade eine Sekunde übersprungen wird. Man müsste also die Schleife drei- bis viermal pro Sekunde laufen lassen, um eine gleichmäßige Sekundenanzeige zu gewährleisten.

```
    zeile2 = „IP:“ + subprocess.check_output([„hostname“,“-I“])[:-2]
```

`zeile2` enthält die IP-Adresse. Dafür wird die Funktion `check_output()` aus dem Modul `subprocess` verwendet, die die Ausgabe eines beliebigen Kommandozeilenbefehls als Zeichenkette liefert. Diese Zeichenkette wird bis auf die letzten zwei Zeichen verwendet, das Zeilenendezeichen `\n` der Ausgabe sowie ein Leerzeichen am Ende werden abgeschnitten.

```
    zeile3 = „“
    zeile4 = „“
```

`zeile3` und `zeile4` sind im Basisprogramm leer. Die bei den jeweiligen Servern beschriebenen Programmerweiterungen geben hier zusätzliche Daten aus.

```
    lcd_anzeige()
```

Zum Schluss werden diese Daten mit der Funktion `lcd_anzeige()` auf dem Display ausgegeben. Nach Ablauf der in der Funktion eingestellten Wartezeit beginnt die Endlosschleife erneut.

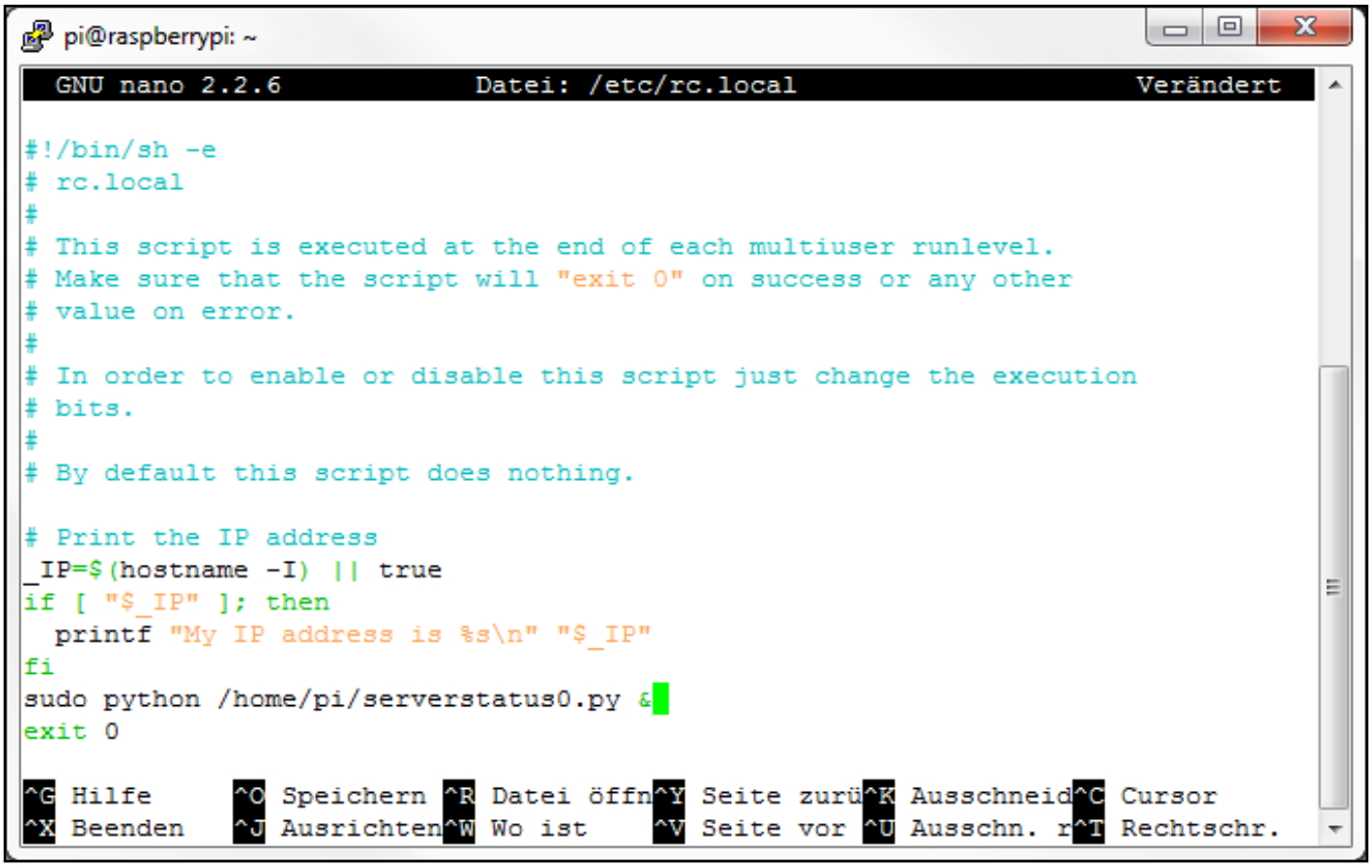
```
except KeyboardInterrupt:
  GPIO.cleanup()
```

Drückt der Benutzer die Tastenkombination [Strg] + [C], wird ein `KeyboardInterrupt` ausgelöst und die Schleife automatisch verlassen. Die letzte Zeile schließt die verwendeten GPIO-Ports. Danach wird das Programm beendet. Nach dem Beenden des Programms bleibt die Anzeige auf dem Display stehen, die Uhr läuft aber nicht mehr weiter. Die Anzeige selbst wird durch den im Display eingebauten Controller aufrecht erhalten.

Durch das kontrollierte Schließen der GPIO-Ports tauchen beim nächsten Start eines Programms, das die GPIO-Schnittstelle nutzt, keine Systemwarnungen oder Abbruchmeldungen auf, die den Benutzer verwirren könnten.

Script automatisch starten

Wirklich sinnvoll ist dieses Programm nur, wenn es beim Start des Raspberry Pi automatisch startet und ohne Zutun des Benutzers IP-Adresse und Systemzeit auf dem Display anzeigt.



Um automatisch zu starten, muss das Script in der Datei `/etc/rc.local` aufgerufen werden. Öffnen Sie diese Datei mit dem nano-Editor.

```
sudo nano /etc/rc.local
```

Fügen Sie in diese Datei vor der Zeile:

```
exit 0
```

eine neue Zeile ein:

```
sudo python /home/pi/serverstatus0.py &
```

Das &-Zeichen am Ende startet das Python-Script im Hintergrund, damit die Datei /etc/rc.local kontrolliert bis zum Ende abgearbeitet werden kann. Auf die gleiche Weise können Sie auch Scripte von der Kommandozeile im PuTTY-Terminal im Hintergrund starten und im Terminal sofort weitere Eingaben machen.

Die Datei /etc/rc.local startet das Python-Script nach dem Booten des Raspberry Pi automatisch.

Speichern Sie die Datei und starten den Raspberry Pi neu.

```
sudo reboot
```

Nach dem Neustart werden auf dem Display automatisch IP-Adresse, Datum und Uhrzeit angezeigt.