Der komplette Quellcode der Projekte für Arduino™ und Raspberry Pi zum Download.

FRUIT UP YOUR FANTASY

WILFRIED KLAAS

BUSSYSTEME IN DER PRAXIS

Serielle Schnittstelle, SPI, I²C, 1-Wire, USB, KNX und CAN: Schnittstellen verstehen und in Projekten mit Arduino™ und Raspberry Pi direkt einsetzen.

FRANZIS

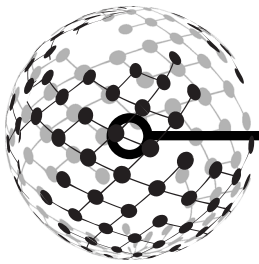
WILFRIED KLAAS

BUSSYSTEME IN DER PRAXIS

Wilfried Klaas studierte Elektrotechnik an der Ruhr Universität Bochum (Abschluss Dipl.-Ing.) und arbeitet bei der EASY Software AG als System-Architekt. Außer mit Musik beschäftigt er sich intensiv mit Mikrocontrollern sowie deren Programmierung und sucht dabei immer die Verbindung zu seinem zweiten großen Hobby, dem Modellbau. Als freier Autor hat er diverse Publikationen zu diesen Themen verfasst.
Kontakt: w.klaas@gmx.de, http://www.rcarduino.tk

FRUIT UP
YOUR
FANTASY

WILFRIED KLAAS



BUSSYSTEME IN DER PRAXIS

Serielle Schnittstelle, SPI, I²C, 1-Wire, USB, KNX und CAN: Schnittstellen verstehen und in Projekten mit Arduino™ und Raspberry Pi direkt einsetzen.

FRANZIS

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

Autor: Wilfried Klaas
Programmleitung: Dr. Markus Stäuble
Satz: DTP-Satz A. Kugge, München
art & design: www.ideehoch2.de
Druck: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn
Printed in Germany

ISBN 978-3-645-65310-7

Vorwort

Schnittstellen und Bussysteme sind sehr wichtige Grundlagen der Mikrocontroller-Programmierung, ermöglichen sie doch die Kommunikation mit der »Außenwelt«.

In diesem Buch stelle ich Ihnen zunächst die verschiedenen relevanten Schnittstellen und Bussysteme vor, die für diese Kopplung wichtig sind.

Im zweiten Teil steigen Sie dann in die Programmierung ein. Ich werde mich auf zwei der wichtigsten Systeme beschränken: Arduino und Raspberry Pi. Die vermittelten Grundlagen können Sie später auf andere Mikrocontroller übertragen.

Alle hier vorgestellten Schaltungen und Programme wurden nach bestem Wissen erstellt und getestet. Jedoch kann für die korrekte Funktion keine Haftung übernommen werden.

Ich wünsche viel Spaß bei der Lektüre.

Wilfried Klaas

Inhaltsverzeichnis

1

Die Hardware

1.1 Arduino™

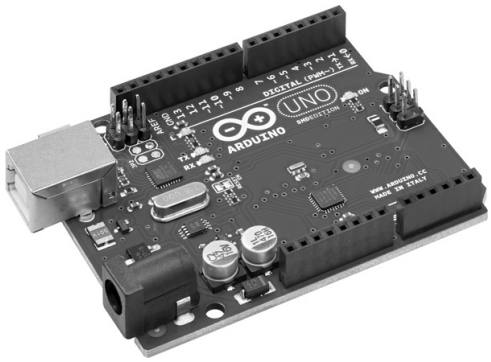


Bild 1.1: Arduino Uno

Wohl der am weitesten verbreitete Mikrocontroller im DIY-Bereich ist der Arduino. Er ist zwar schon etwas älter, hat aber nichts von seiner Attraktivität verloren. Im Kern handelt es sich um einen Atmel-Chip aus der ATmega-Reihe. Je nach Board findet ein ATmega328 (Arduino Uno), ein ATmega32U16 (Arduino Leonardo) oder ein ATmega256 (Arduino Mega) Verwendung. Die ATmegas sind 8-Bit-Prozessoren mit einer Taktfrequenz von 16 MHz. Die Ressourcen wie RAM und Flash sind natürlich begrenzt, reichen aber für die üblichen Anwendungen aus. Große Vorteile der Arduinos sind die weite Verbreitung, die große Gemeinschaft und die niedrigen Hard-

und Softwarekosten. Ein offizielles Board kostet zwar fast so viel wie ein Raspberry Pi, aber man bekommt ein komplettes Entwicklungssystem mit dazu. Und mittlerweile gibt es preisgünstige kompatible Varianten. Wer sich etwas eingearbeitet hat, kann später auch eigene Boards designen und direkt an seine Bedürfnisse anpassen. Ein ATmega-Basis-Board kann man dann selber auf einer Lochrasterplatine für weniger als 5 € herstellen.

Der Arduino hat bereits eine Vielzahl von Schnittstellen integriert. So finden Sie bei einem originalen Board neben SPI und I^2C auch direkt eine funktionierende USB-Anbindung. Diese ist als reine Seriell/USB-Konvertierung ausgelegt.

Einzig der Arduino Leonardo kann als echter USB-Slave benutzt werden. Falls man einen echten USB-Slave verwenden möchte, müssen neben der Programmierung im Mikrocontroller auch die entsprechenden Treiber für das Betriebssystem erstellt werden. Und wer sich schon einmal mit der Treiberprogrammierung unter Windows oder Linux auseinandergesetzt hat, weiß, dass das kein einfaches Unterfangen ist. Der Arduino Leonardo kann aber z. B. direkt als HID (Human Interface Device) eingesetzt werden und die bereits vorhandenen HID-Treiber verwenden.

Dies ist zwar nicht seine bevorzugte Anwendung, aber selbst der Arduino kann mit entsprechender Hardware eine Internetverbindung aufbauen und so z. B. Messwerte übers Internet zur Verfügung stellen. Dabei sollte man allerdings stark auf den Ressourcenverbrauch achten.

Für dieses Buch sollten Sie schon etwas Übung im Umgang mit dem Arduino haben. Grundlegende Kenntnisse der IDE und Programmierkenntnisse sind Voraussetzung.

1.1.1 Schnittstellen

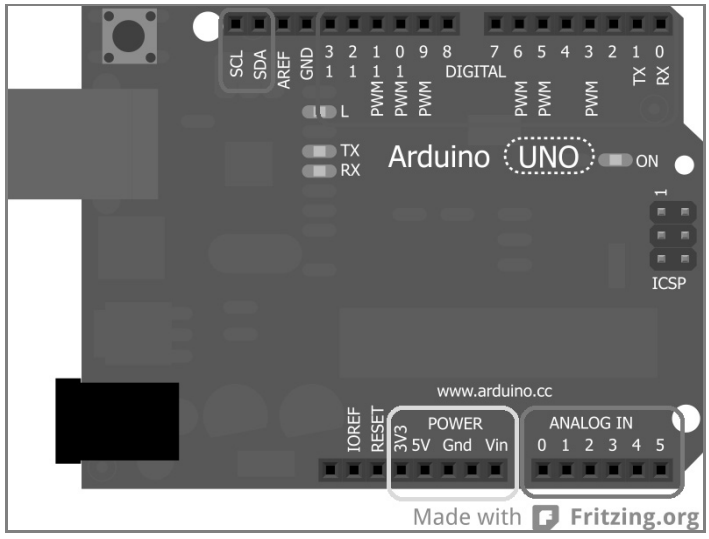


Bild 1.2:
Schnittstellen des Arduino Uno

Die nachfolgende Tabelle beschreibt die einzelnen Schnittstellen auf der Arduino-Platine.

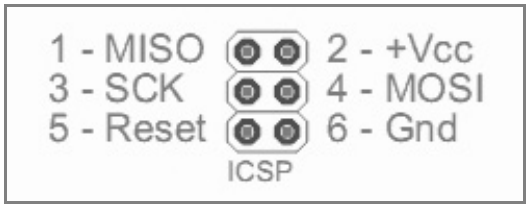
USB	Die große silberne Buchse links oben. Der USB-Anschluss ist automatisch mit der 1. seriellen Schnittstelle verbunden.
1. serielle Schnittstelle	rot, Signale TX und RX
2. serielle Schnittstelle	Beim Arduino Uno kann die 2. serielle Schnittstelle nur über eine zusätzliche Bibliothek per Softwareemulation erzeugt werden. Gut geeignet ist die AltSoftSerial-Bibliothek[1] von Paul Stoffregen, da sie im Interruptmodus arbeitet. TX ist dann auf D9, RX auf D8. PWM auf D10 funktioniert dann wegen des in der Bibliothek benutzten Timers nicht mehr. Der Pin D10 kann aber für andere Aufgaben weiter benutzt werden.
SPI	rot oder ICSP, Signale MOSI-Pin 11 oder ICSP-4, MISO-Pin 12 oder ICSP-1, SCK-Pin 13 oder ICSP-3, SS-Pin 10. Kann normalerweise frei im roten (D0-D13) oder grünen Bereich (D14-D19) gewählt werden.  Bild 1.3: ICSP-Header
I²C	lila (grün), Signale SCL (A4), SDA (A5)
1-Wire	rot, Pin kann normalerweise frei im Bereich von D0-D19 gewählt werden.

Tabelle 1.1: Anordnung der Schnittstellen beim Arduino

[1] *https://github.com/PaulStoffregen/AltSoftSerial*

1.2 Raspberry Pi

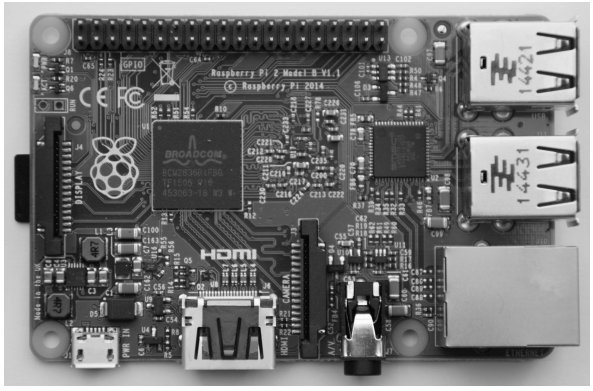


Bild 1.4: Raspberry Pi 2, Model B

Der Raspberry Pi ist ein vollständiger Computer. Als Betriebssystem wird normalerweise das Linux-Derivat Raspbian[2] verwendet. Der direkte Zugriff auf die Port-Pins (GPIO, General Purpose Input Output) ist möglich und ermöglicht die eigene Implementierung von Bussystemen. Doch Vorsicht, Linux ist kein Echtzeitbetriebssystem, d. h. es kann durchaus passieren, dass inmitten der eigenen Routine, die Zeichen von der seriellen Schnittstelle liest, das Betriebssystem plötzlich "etwas ganz wichtiges Anderes erledigen muss" und dann leider Zeichen verloren gehen. Das kann man durch direkte Treiberprogrammierung verhindern, die aber leider bei Linux nicht weniger komplex als in einem Windows-System ist. Zum Glück gibt es bereits fertige Treiber für die wichtigsten Schnittstellen. Den Raspberry Pi gibt es mittlerweile in verschiedenen Varianten. Ich verwende in diesem Buch hauptsächlich den Raspberry Pi 2, Model B. Er funktioniert im Prinzip genau wie der ältere Raspberry Pi B+, jedoch wurden der Arbeitsspeicher und die USB-Schnittstellen verdoppelt. Außerdem ist – sehr wichtig für unsere Anwendungen – der Prozessor ein Quadcore-Prozessor. D. h. statt bisher nur einer CPU stehen dem Raspberry Pi nun vier CPUs zur Verfügung. Das macht sich in der Geschwindigkeit aber auch in der Verfügbarkeit deutlich bemerkbar. Denn nun kann das Betriebssystem nebenbei Operationen ausführen, ohne unmittelbar die Ausführung unseres Codes zu beeinflussen. Mittelbar natürlich schon, weil die Ressourcen geteilt werden. Schnittstellen für SPI, I^2C und Seriell sind schon vorhanden.

[2] *https://www.raspbian.org/*

Raspberry A+B

		Pin			
	3,3V	1	2	5V	
I²C: SDA	GPIO 02	3	4	5V	
I²C: SCL	GPIO 03	5	6	GND	
GCLK	GPIO 04	7	8	GPIO 14	TX 0
	GND	9	10	GPIO 15	RX 0
GEN0	GPIO 17	11	12	GPIO 18	GEN1
GEN2	GPIO 27	13	14	GND	
GEN3	GPIO 22	15	16	GPIO 23	GEN4
	3,3V	17	18	GPIO 24	GEN5
SPI: MOSI	GPIO 10	19	20	GND	
SPI: MISO	GPIO 09	21	22	GPIO 25	GEN6
SPI: CLK	GPIO 11	23	24	GPIO 08	SPI: SS0
	GND	25	26	GPIO 07	SPI: SS1

Raspberry B+ & 2

		Pin			
	3,3V	1	2	5V	
I²C: SDA	GPIO 02	3	4	5V	
I²C: SCL	GPIO 03	5	6	GND	
GCLK	GPIO 04	7	8	GPIO 14	TX 0
	GND	9	10	GPIO 15	RX 0
GEN0	GPIO 17	11	12	GPIO 18	GEN1
GEN2	GPIO 27	13	14	GND	
GEN3	GPIO 22	15	16	GPIO 23	GEN4
	3,3V	17	18	GPIO 24	GEN5
SPI: MOSI	GPIO 10	19	20	GND	
SPI: MISO	GPIO 09	21	22	GPIO 25	GEN6
SPI: CLK	GPIO 11	23	24	GPIO 08	SPI: SS0
	GND	25	26	GPIO 07	SPI: SS1
I²C: ID EEPROM	ID_SD	27	28	ID_SC	I²C ID EEPROM
	GPIO 05	29	30	GND	
	GPIO 06	31	32	GPIO 12	
	GPIO 13	33	34	GND	
	GPIO 19	35	36	GPIO 16	
	GPIO 26	37	38	GPIO 20	
	GND	39	40	GPIO 21	

Tabelle 1.2: Belegung der GPIO-Pins

Für den Raspberry-Pi-Teil sind Kenntnisse der Grundlagen im Umgang mit dem Raspberry Pi und dem Betriebssystem Raspbian Voraussetzung, ebenso Grundkenntnisse der Sprache Python. In diesem Buch wird abgesehen von ein paar Terminalscripten hauptsächlich mit der Sprache Python gearbeitet.

4

Anwendungsbeispiele mit dem Raspberry Pi

Das Betriebssystem des Raspberry Pi, Raspbian, hat als Basis ein Debian-Linux. Es gibt verschiedene Möglichkeiten, für den Raspberry Pi Programme zu erstellen. Im Folgenden werden hauptsächlich Programme in der Sprache Python beschrieben, weil die Struktur dieser Sprache recht einfach ist und sich daher auch für den weniger geübten Programmierer eignet. Viele der hier beschriebenen Beispiele und Bibliotheken gibt es natürlich auch in C oder einer anderen Sprache.

4.1 Seriell und USB

Der Raspberry Pi hat zwei bzw. vier USB-Host-Anschlüsse. An diese lassen sich diverse USB-Geräte anschließen. Beim Modell A lässt sich ein Anschluss auf OTG (USB On-The-Go) umschalten und der Raspberry Pi damit als USB-Slave betreiben. Die Programmierung dazu ist allerdings recht komplex. Noch dazu unterstützt diese Vorgehensweise nur das Modell A des Raspberry Pi. Deswegen verzichte ich an dieser Stelle auf eine weitere Vertiefung. Anders sieht das bei der seriellen Schnittstelle aus. Der Raspberry Pi besitzt eine serielle Schnittstelle auf dem GPIO-Stecker.

```
  # mit den entsprechenden Einstellungen für den 24er-Ring
  strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
LED_INVERT, LED_BRIGHTNESS)
  # Initialisierung der Bibliothek. Muss einmal
  # aufgerufen werden, bevor die Bibliothek
  # verwendet werden kann.
  strip.begin()

  print 'Press Ctrl-C to quit.'
  while True:
    strip.setBrightness(255)
    showTime()
    time.sleep(0.2)
```

4.6 Internet

Der Raspberry Pi kann noch viel mehr. Wie wäre es mit einer Darstellung von Messwerten im WWW oder auch der Steuerung von Aktoren? Dazu benötigt man nicht viel. In den vorherigen Kapiteln haben sich einige Python-Scripts angesammelt, die nun ins WWW gebracht werden. Dazu benötigt man natürlich zunächst einen Web-Server. Hier wird das Python-Webframework Flask vorgestellt und verwendet.

4.6.1 Installation und Test des Python-WebServers Flask

Zunächst muss Flask erst einmal installiert werden. Dazu benötigt man den Python-Paket-Installer PIP. Auch der muss zunächst installiert werden:

```
sudo apt-get install python-pip
```

Jetzt kann man auch `flask` installieren:

```
sudo pip install flask
```

Zum Testen wird dieses kleines Pythonscript dienen:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Zunächst werden die für dieses Script benötigten Flask-Module geladen.

Dann wird ein neues Flask-Objekt erzeugt. Mit `@app.route("/")` wird der Einstiegspunkt der Webapplikation erzeugt, in diesem Fall also das Webroot.

Es folgt eine Funktion, die einen einfachen String zurückgibt.

Mit `if __name__ == "__main__":` wird dann geprüft, ob das Python-Modul direkt aus einer Kommandozeile ausgeführt wird. Wenn ja, wird der WebServer gestartet. Dieser wird auf alle verfügbaren IP-Adressen gebunden und horcht auf Port 80. Zusätzlich werden alle Fehler und Debug-Ausgaben ausgegeben.

Und schon kann es losgehen: Einfach das Script mit `sudo python test.py` starten. Danach einen Browser öffnen und die Adresse `http://127.0.0.1/` eingeben. Jetzt sollte ein "Hello World!" im Browser stehen.

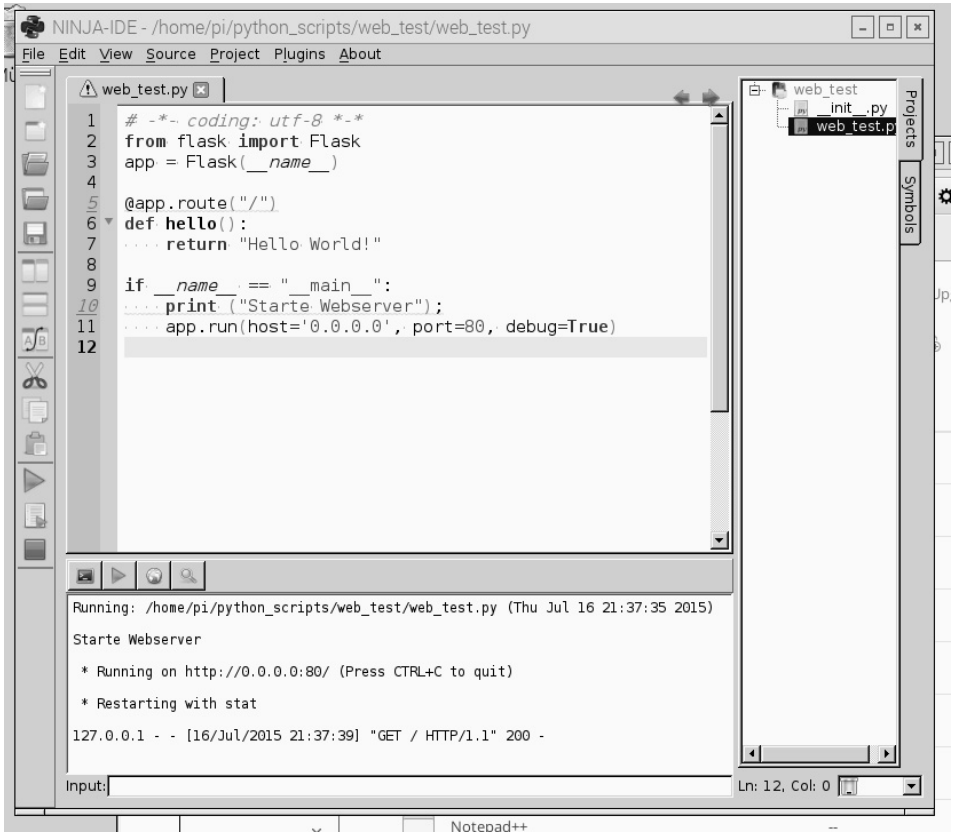


Bild 4.28: Ninja-IDE mit dem Hello-World-Testprogramm

Bild 4.29:
Hello-World-Browserausgabe

4.6.2 Benutzung von WebTemplates

Ganze Webseiten mit einem Python-Script zu erzeugen ist sehr aufwendig. Deswegen ist im Flask-Framework auch die Template-Engine Jinja2 (siehe Kapitel 7, Internetseite 27) enthalten. Mit dieser Engine lassen sich Vorlagen von Webseiten mit Inhalten verknüpfen. Z. B. erstellt man sich eine Webseite zur Ausgabe der Temperatur eines Sensors. Die Webseite soll folgendermaßen aussehen:

Datei `index.html`:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Mein Raspberry: {{ title }}</title>
</head>
<body>
Die Temperatur um {{ time }} beträgt {{ temperatur }} °C.
</body>
</html>
```

Diese Datei wird in ein Unterverzeichnis mit dem Namen »templates« unterhalb unserer Python-Datei gestellt. Die Python-Datei wird wie folgt geändert:

```
from flask import Flask, render_template
import datetime
app = Flask(__name__)

def getTemperatur():
  ... # hier Temperatur besorgen und zurückgeben

@app.route("/")
def temp():
   now = datetime.datetime.now();
   strTime = now.strftime("%H:%M %d.%m.%Y");
   temperatur = getTemperatur();
   strTemperatur = '{:02.2f}'.format(temperatur);
   templateData = {
      'title' : 'Tempsensor 1',
```

```
016       'temperatur' : strTemperatur,
017       'time': strTime
018       };
019    return render_template('index.html', **templateData);
020
021 if __name__ == "__main__":
022    app.run(host='0.0.0.0', port=80, debug=True);
```

Vieles ist schon bekannt. Die Bibliothek `datetime` enthält Datums- und Zeitmethoden. Da das aktuelle Datum und die Uhrzeit ausgegeben werden sollen, wird diese Bibliothek benötigt.

`getTemperatur()` ist die Funktion, in der der Temperatursensor angesprochen wird. Wie das funktioniert, ist in den vorherigen Kapiteln bereits behandelt worden.

In der Hauptmethode `temp()` werden zunächst das aktuelle Datum und die Uhrzeit geholt und entsprechend formatiert. Dann wird die Temperatur besorgt und auch entsprechend formatiert (hier mit zwei Vor- und zwei Nachkommastellen).

Es wird eine Struktur (Record) mit den verschiedenen Werten zusammengebaut und mithilfe der Renderengine zu einer HTML-Seite verarbeitet. Dabei werden die Platzhalter in der HTML-Datei (gekennzeichnet durch das `{{…}}`) durch die entsprechenden Texte im Record ersetzt.

Das Ergebnis sieht dann so aus:

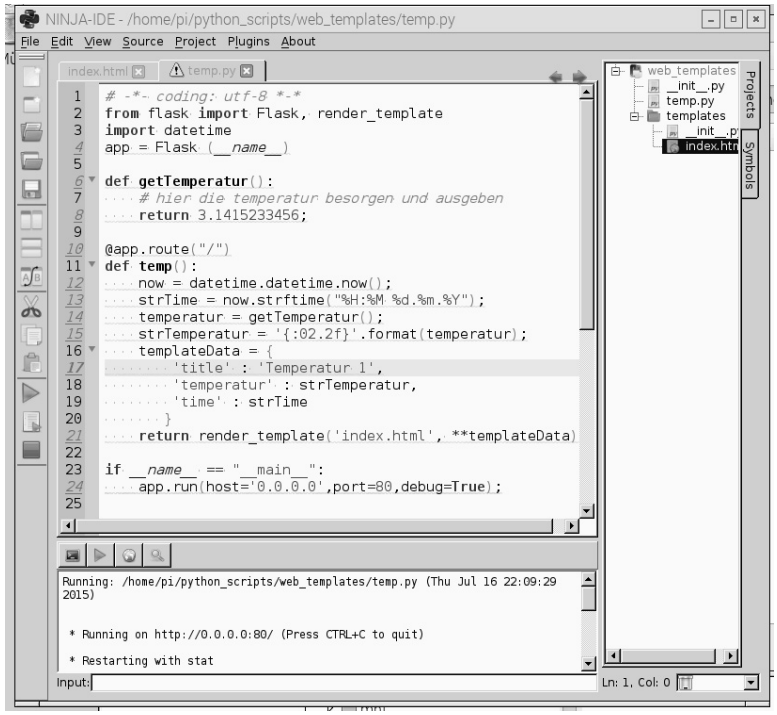


Bild 4.30: Ninja-IDE mit dem Web-Temperatur-Programm

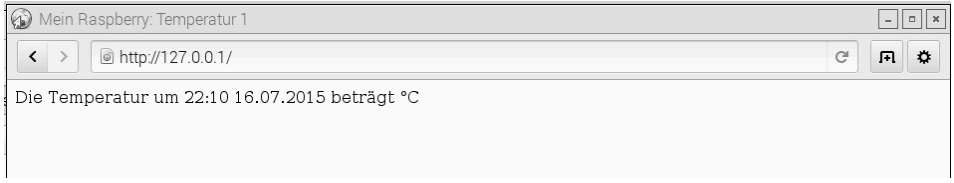


Bild 4.31: Browserausgabe Web-Temperatur

4.7 Raspberry Pi und Arduino™, zwei Welten verbunden

Bereits im Kapitel 4.1.2 wurde eine mögliche Kopplung von Arduino und Raspberry Pi beschrieben. Dabei wurde die serielle Schnittstelle verwendet. Auch mittels einer Funkstrecke (Kapitel 3.2.9 und 4.2.5) kann eine Kopplung beider Systeme erfolgen. Diese Kopplung wird z.B. dann benötigt, wenn der Raspberry Pi den Echtzeitanforderungen einer Schnittstelle nicht mehr genügt. Das Linux-Betriebssystem ist eben kein Echtzeitbetriebssystem, so kann es sein, dass mitten in der Routine zur Aktualisierung einer WS2812B-Kette ein höher priorisierter Systemtask dazwischen funkt und das

zeitkritische Protokoll stört. Das macht sich durch Flackern der LED-Kette bemerkbar. Natürlich kann man um diese Probleme herum programmieren. Im Falle der WS2812B wird per DMA und PWM eine vom Betriebssystem ungestörte Steuerung ermöglicht. Manchmal ist das aber nicht möglich. Mithilfe eines Arduinos, der die Echzeitsteuerung übernimmt, kann man dann das Update und die Schnittstellenkoordination dem Arduino überlassen. Die Kontrolle aber kann weiterhin der Raspberry Pi behalten. Exemplarisch für eine solche Schnittstelle sei hier das bereits im Kapitel 4.1.2 gezeigte Beispiel genannt.

Wie die Kopplung tatsächlich aussieht, kann je nach Anwendungsfall variieren: seriell oder SPI oder eventuell per Funk. Die Grundlagen dazu sind in den vorherigen Kapiteln beschrieben worden. Nun gilt es nur noch, neben der hardware technischen Anbindung ein entsprechendes Software-Protokoll zu finden oder selber zu entwickeln.

Stichwortverzeichnis

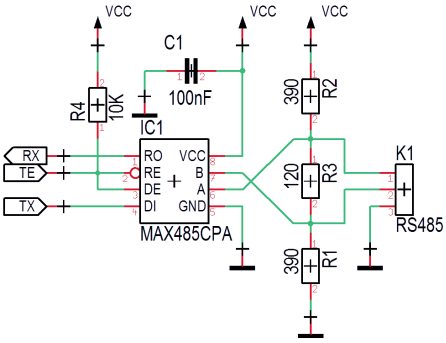
WILFRIED KLAAS

BUSSYSTEME IN DER PRAXIS

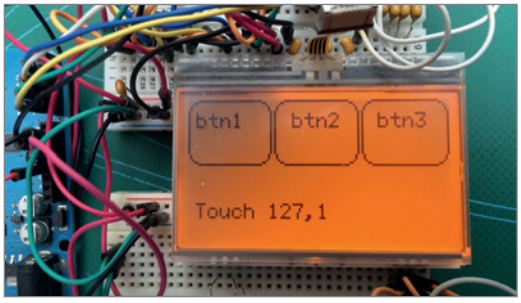
Aus dem Inhalt:

- Schnittstellen des Arduino™ und des Raspberry Pi
- Bussysteme: Serielle Schnittstelle, SPI, I²C, 1-Wire, USB, CAN und KNX
- SB-Kommunikation mit dem Arduino™ und dem Raspberry Pi
- DMX mit dem Arduino™
- Schieberegister, Sieben-Segment-Display, Grafik-LCD und Funkverbindung über SPI
- Gyroskop, Beschleunigungssensor, Temperatur- und Luftdruckmessung über I²C
- Temperaturmessung mittels 1-Wire
- Arduino™ und Raspberry Pi miteinander verbinden
- Arduino™ und Raspberry Pi programmieren

Geringe Anschaffungskosten, eine große Community und gut zugängliche Schnittstellen sind die Erfolgsfaktoren von Arduino™ und Raspberry Pi. Ob Sensoren, Displays oder andere Hardware – die genaue Kenntnis von Schnittstellen wie SPI und I²C ist für erfolgreiche Projekte unumgänglich. Egal ob Sie Arduino™ oder Raspberry Pi bevorzugen, in diesem Buch lernen Sie die Funktionsweise dieser Schnittstellen im Detail kennen und nutzen sie in Praxisprojekten für beide wichtigen Plattformen. Für den schnellen Einstieg steht der komplette Quellcode zum Download bereit.



Alle wichtigen Details zu den Schnittstellen werden erklärt: Aufbau, Funktionsweise und Programmierung.



Projekte für den Arduino™ und den Raspberry Pi verdeutlichen die Nutzung der einzelnen Schnittstellen.

Projektpraxis für Arduino™ und Raspberry Pi

Mit Stromlaufplänen, Blockdiagrammen und Protokollbeschreibungen werden die wichtigsten Schnittstellen für eigene Maker-Projekte beschrieben. Damit Sie das Wissen auch direkt für Arduino und Raspberry Pi anwenden können, stellt Ihnen Klaas zahlreiche nachvollziehbare Praxisprojekte mit Schaltplan und Quellcode vor: Der Anschluss von Sensoren über I²C, die Ansteuerung von Displays über SPI oder die Ansteuerung eines GPS-Moduls über die serielle Schnittstelle sind nur einige Beispiele aus dem großen Projektteil. In einem extra Kapitel stellt Ihnen Klaas außerdem die Programmierumgebungen für Arduino™ und Raspberry Pi vor.

Der komplette Quellcode aus dem Buch auf www.buch.cd

9 783645 653107

Besuchen Sie unsere Website
www.franzis.de

FRANZIS