CPU-Lastanzeige mit LEDs am Portexpander

Das nächste Experiment liefert auf acht LEDs eine Pegelanzeige der aktuellen CPU-Auslastung.

Bauen Sie die Schaltung wie in der Abbildung auf. Der Schaltungsaufbau entspricht dem ersten Experiment mit dem Portexpander, wo ebenfalls einfach acht LEDs an den acht Pins der GPIOA-Schnittstelle angeschlossen sind. Der Schaltungsaufbau entspricht somit dem Experiment mit dem Lauflicht, also dem Experiment 16 im Buch.

Benötigte Bauteile

- 2 x Steckplatine
- 1 x Portexpander MCP23017
- 2 x LED rot
- 2 x LED gelb
- 2 x LED grün
- 2 x LED blau
- 4 x Verbindungskabel
- 14 x Drahtbrücke (unterschiedliche Längen)

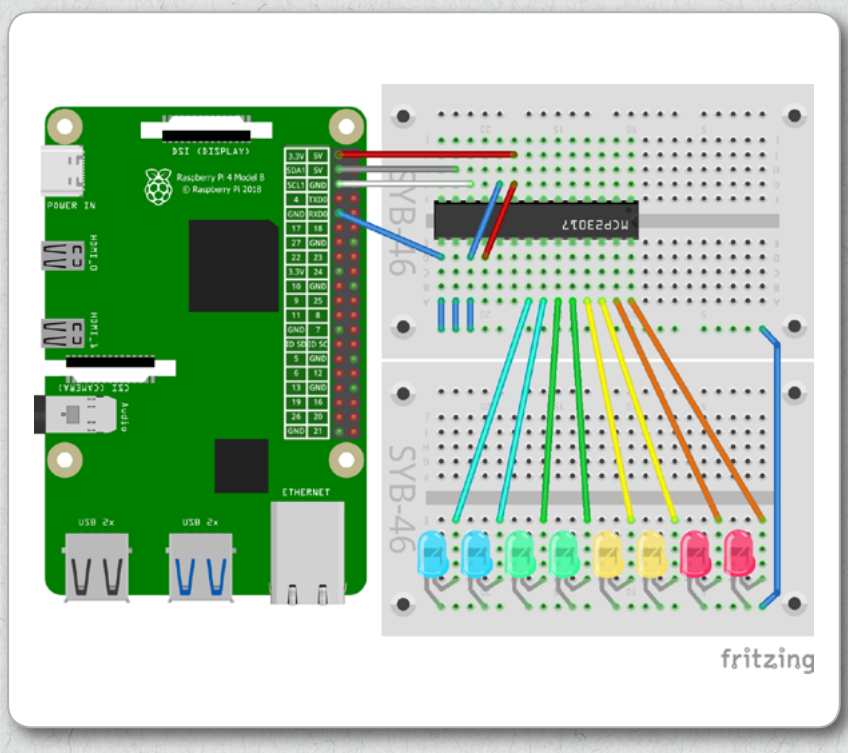

Abb. 1: *CPU-Lastanzeige mit MCP23017-Portexpander.*

Raspberry Pi 4

Beim gegenüber den Vorgängermodellen deutlich leistungsstärkeren Raspberry Pi 4 lässt sich die CPU-Auslastung durch einfache Mausbewegungen oder Starten von Programmen kaum höher als auf einen einstelligen Prozentwert treiben, sodass auf der CPU-Lastanzeige meistens nichts zu sehen ist. Öffnen Sie im Browser eine aufwendige Webseite wie z. B. `www.youtube.com` oder starten Sie in Mathematica eine der komplexeren Demos, um ein bisschen CPU-Last zu sehen. Bei früheren Raspberry-Pi-Modellen lässt sich bereits durch Öffnen von Programmen oder durch hektische Mausbewegungen erkennbare CPU-Last erzeugen.

Je nach CPU-Auslastung leuchten ein bis acht LEDs. Das Programm `19cpu.py` basiert auf dem Programm aus dem Experiment mit dem Lauflicht und verwendet zusätzlich einige interessante Funktionen zur Berechnung der CPU-Last.

```
#!/usr/bin/python
import time, os, smbus

DEVICE = 0x20
IODIRA = 0x00
GPIOA  = 0x12

bus = smbus.SMBus(1)
bus.write_byte_data(DEVICE,IODIRA,0x00)
bus.write_byte_data(DEVICE,GPIOA,0x00)

while True:
    cpu1 = os.popen("vmstat 1 3").readlines()
    cpu2 = 2 ** int((100 - int(cpu1[4].split()[14])) * 0.08) - 1
    bus.write_byte_data(DEVICE,GPIOA,cpu2)
```

So funktioniert es

Die ersten Programmzeilen zur Initialisierung des i2c-Bus sind bereits bekannt. In diesem Fall wird nur zusätzlich das Modul `os` zum Zugriff auf Systemfunktionen importiert.

```
while True:
    cpu1 = os.popen("vmstat 1 3").readlines()
```

Das Programm läuft in einer Endlosschleife. Sie ruft in jedem Durchlauf zuerst das Linux-Kommando `vmstat 1 3` auf, womit sich diverse Statusangaben des Systems auslesen lassen. Der Parameter `3` bedeutet, dass die Statusdaten dreimal hintereinander gelesen werden. Beim ersten Lesen ergeben sich verfälschte Informationen, da der Aufruf des Kommandos selbst einiges an Systemleistung frisst.

Die Funktion `os.popen()` schreibt das Ergebnis eines Linux-Kommandos zeilenweise in ein Array mit Zeichenketten, hier die Variable `cpu1`.

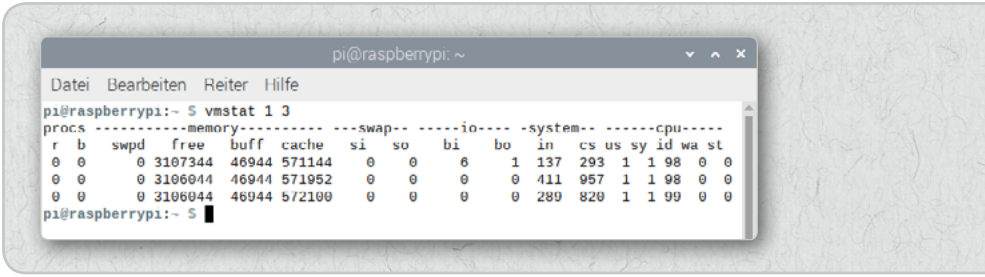

Abb. 2: *Ausgabe des Befehls vmstat 1 3 in einem Konsolenfenster.*

Die nächste Zeile ermittelt mit einer interessanten Berechnungsformel den Wert, der mit den LEDs angezeigt werden soll, und ihn diesen in der Variablen `cpu2`.

```
cpu2 = 2 ** int((100 - int(cpu1[4].split()[14])) * 0.08) - 1
```

Die dritte Statusabfrage steht nach den beiden Tabellenüberschriften in der fünften Zeile und damit im fünften Element des Arrays, das bei Zählung ab `0` die Nummer `4` trägt.

```
cpu1[4].split()
```

Diese Zeichenkette wird mit der Methode `split()` an den Leerzeichen in einzelne Zeichenketten aufgespalten. Interessant ist das Element der Nummer `[14]`, das, mit der Zählung wieder bei `0` beginnend, den Inhalt der Spalte `id` enthält. Diese Spalte gibt die ungenutzte CPU-Zeit in Prozent an.

```
100 - int(cpu1[4].split()[14])
```

Subtrahiert man diesen Wert von 100, erhält man die verbrauchte CPU-Zeit in Prozent, also die Auslastung der CPU. Zuvor wird die Zeichenkette mit der `int()`-Funktion in eine Ganzzahl umgewandelt.

```
(100 - int(cpu1[4].split()[14])) * 0.08)
```

Eine Multiplikation mit dem Faktor `0,08` ergibt zur Darstellung auf den acht LEDs eine Zahl zwischen 0 und 8 statt zwischen 0 und 100.

Zur Darstellung über den Portexpander braucht man 8-Bit-Zahlen. In diesem Fall einer Pegelanzeige sollen LEDs in durchgehender Folge leuchten. Die Tabelle zeigt alle Zahlen zwischen 0 und 8, die aus einer durchgehenden Folge von Einsen in der Binärdarstellung bestehen.

BINÄR	DEZIMAL	BERECHNUNGSFORMEL
0000 0000	0	2 ** 0 – 1
0000 0001	1	2 ** 1 – 1
0000 0011	3	2 ** 2 – 1
0000 0111	7	2 ** 3 – 1
0000 1111	15	2 ** 4 – 1
0001 1111	31	2 ** 5 – 1
0011 1111	63	2 ** 6 – 1
0111 1111	127	2 ** 7 – 1
1111 1111	255	2 ** 8 – 1

Über die Berechnungsfunktion in der rechten Spalte der Tabelle werden die Werte ermittelt, die binärcodiert die entsprechenden LEDs ansteuern. Das Symbol `**` steht in Python für die Exponentialfunktion.

```
bus.write_byte_data(DEVICE,GPIOA,cpu2)
```

Das in der Variablen `cpu2` gespeicherte Ergebnis wird auf den `GPIOA`-Port des Portexpanders geschrieben und mit den LEDs angezeigt. Danach startet die Endlosschleife neu und ermittelt wiederum die aktuelle CPU-Last.