LC-Display am Portexpander

Das LC-Display nutzt je nach eingestelltem Modus vier oder acht Datenleitungen, auf denen es bitcodierte Daten empfängt. Der Portexpander verfügt ebenfalls über zwei mal acht Datenleitungen, auf denen bitcodierte Daten gesendet werden können. Da bietet es sich geradezu an, das LC-Display mit dem Portexpander zu verbinden. Man braucht zum Anschluss des Displays am Raspberry Pi neben der Stromversorgung nur noch zwei Leitungen für den i2c-Bus und nicht mehr sechs bzw. zehn GPIO-Ports.

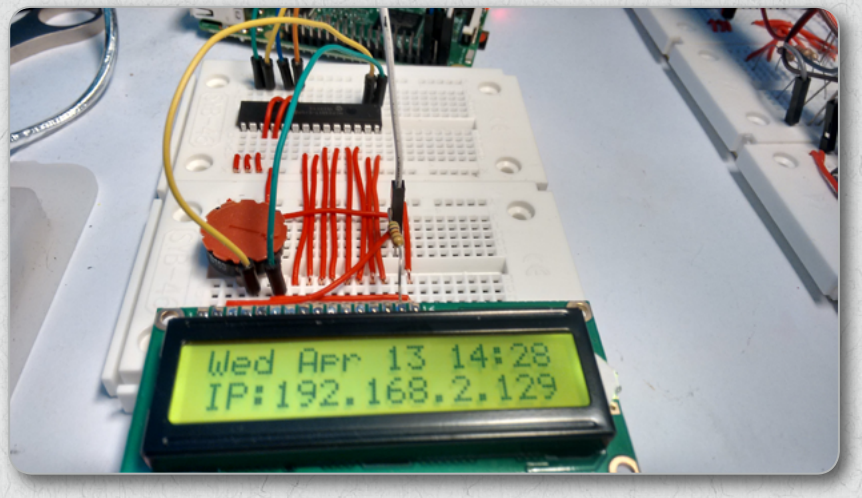

Abb. 1: *LC-Display mit Portexpander am Raspberry Pi.*

Für den Anschluss am Portexpander empfiehlt sich der 8-Bit-Modus des LC-Displays, da die Zeichen direkt byteweise anstatt in zwei Blöcken auf das Display geschrieben werden können, was das Programm deutlich vereinfacht. Die acht Datenleitungen des `GPIOA`-Ports des Portexpanders werden eins zu eins mit den Datenleitungen des LC-Displays verbunden. Zwei der Leitungen des `GPIOB`-Ports werden für die Steuersignale `RS` und `E` verwendet.

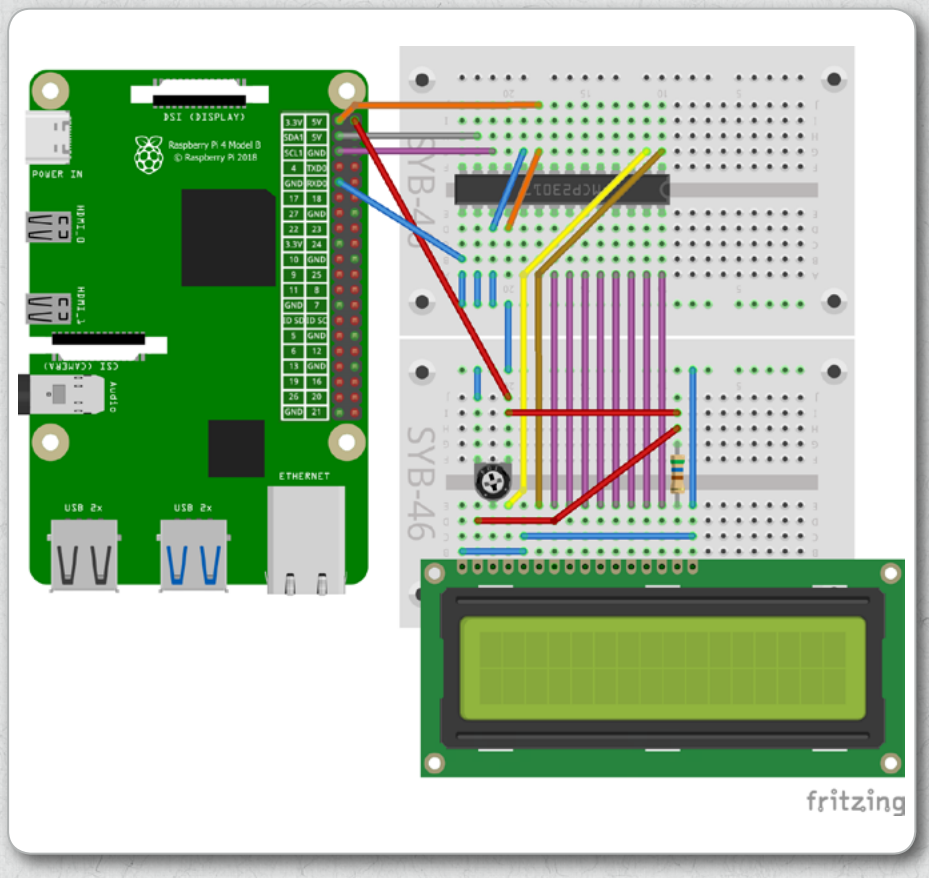

Abb. 2: *LC-Display am Portexpander.*

Benötigte Bauteile

- 2 x Steckplatine
- 1 x Portexpander MCP23017
- 1 x LC-Display
- 1 x 560-Ohm-Widerstand (Grün-Blau-Braun)
- 1 x Potenziometer
- 5 x Verbindungskabel
- 23 x Drahtbrücke (unterschiedliche Längen)

Zwei verschiedene Stromversorgungen

Achten Sie beim Aufbau besonders genau auf den korrekten Anschluss der Stromversorgung, da die Schaltung beide Stromversorgungsleitungen des Raspberry Pi nutzt, +3,3 V für den Portexpander und +5 V für das LC-Display.

Das Programm `20i2cuhr.py` übernimmt die grundlegende Struktur der Statusanzeige `20status.py`, wobei einige Funktionen für die Steuerung über den Portexpander angepasst wurden. Zusätzlich sind die bereits bekannten Initialisierungsroutinen für den i2c-Bus enthalten, dafür fällt die Initialisierung der GPIO-Schnittstelle weg.

```
#!/usr/bin/python
import time, smbus, os
bus = smbus.SMBus(1)

DEVICE = 0x20
IODIRA = 0x00
IODIRB = 0x01
GPIOA  = 0x12
GPIOB  = 0x13
```

```
LCD_WIDTH  = 16
LCD_LINE_1 = 0x80
LCD_LINE_2 = 0xC0
LCD_CHR = 1
LCD_CMD = 0
LCD_RS  = 0x02
LCD_E   = 0x01
E_PULSE = 0.00005
E_DELAY = 0.00005
INIT  = 0.01
pause = 2

def lcd_byte(bits, mode):
    bus.write_byte_data(DEVICE,GPIOB,LCD_RS*mode)
    bus.write_byte_data(DEVICE,GPIOA,bits)
    time.sleep(E_DELAY)
    bus.write_byte_data(DEVICE,GPIOB,LCD_E+LCD_RS*mode)
    time.sleep(E_PULSE)
    bus.write_byte_data(DEVICE,GPIOB,LCD_RS*mode)
    time.sleep(E_DELAY)

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

bus.write_byte_data(DEVICE,IODIRA,0x00)
bus.write_byte_data(DEVICE,IODIRB,0x00)
bus.write_byte_data(DEVICE,GPIOA,0x00)
bus.write_byte_data(DEVICE,GPIOB,0x00)

def lcd_anzeige(z1, z2):
    lcd_byte(LCD_LINE_1, LCD_CMD)
    lcd_string(z1)
    lcd_byte(LCD_LINE_2, LCD_CMD)
    lcd_string(z2)
```

```
LCD_INIT = [0x33, 0x33, 0x32, 0x3C, 0x0C, 0x06, 0x01]
for i in LCD_INIT:
    lcd_byte(i,LCD_CMD)
    time.sleep(INIT)

while True:
    zeile1 = time.asctime()
    zeile2 = os.popen("hostname -I").readline()[:-2]
    lcd_anzeige(zeile1, zeile2)
    time.sleep(pause)
```

So funktioniert es

Am Anfang werden die Module `smbus` für den i2c-Bus sowie `time` und `os` importiert, die für die Statusanzeigen benötigt werden. Die GPIO-Bibliothek wird nicht gebraucht.

Die Konstanten für den Portexpander sind bereits bekannt, die Konstanten für das LC-Display wurden weitestgehend übernommen, um die Änderungen am Programmcode so gering wie möglich zu halten.

```
LCD_RS = 0x02
LCD_E  = 0x01
```

Die beiden Signale `LCD_RS` (*Register Select*) und `LCE_E` (*Enable*) sind jetzt keine GPIO-Pins mehr, sondern Bitcodes, die über den `GPIOB`-Port an die entsprechenden Steuerleitungen des LC-Displays gesendet werden. Die ehemaligen Konstanten für die vier oder acht Datenleitungen sowie die Initialisierung der GPIO-Ports entfallen ersatzlos.

Die größten Änderungen finden sich in der Funktion `lcd_byte()`, die der Einfachheit halber mit der ehemaligen Funktion `lcd_enable()` zusammengefasst wurde.

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

Der Funktionsaufruf bleibt kompatibel zum ursprünglichen Programm, damit braucht das Hauptprogramm nicht verändert zu werden. Die Funktion `lcd_byte()` bekommt weiterhin zwei Parameter. 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.

```
bus.write_byte_data(DEVICE,GPIOB,LCD_RS*mode)
```

Als Erstes wird das Display auf Kommandomodus oder Zeichenmodus gesetzt. Dazu wird das Bitmuster für das zweite Bit des `GPIOB`-Ports (`0x02`), das in der Konstanten `LCD_RS` gespeichert ist, mit dem Parameter `mode`, der 0 oder 1 sein kann, multipliziert. Auf diese Weise wird dieses Bit für den Kommandomodus ausgeschaltet und für den Zeichenmodus eingeschaltet.

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

Diese Zeile schreibt das an das Display zu sendende Byte auf den `GPIOA`-Port. Bei Verwendung des Portexpanders entfällt die Umrechnung des Zeichens in Bitcode.

```
time.sleep(E_DELAY)
bus.write_byte_data(DEVICE,GPIOB,LCD_E+LCD_RS*mode)
```

Jetzt braucht das Display das Enable-Signal, einen Wechsel von `1` auf `0` auf der `E`-Leitung, um die Daten auf den Datenleitungen einzulesen und darzustellen. Nach einer kurzen Verzögerung, um die Displayträgheit zu überbrücken, wird das in der Konstanten `LCD_E` festgelegte erste Bit des `GPIOB`-Ports auf `1` gesetzt. Da auf dem Portexpander immer ein komplettes Byte auf einmal ausgegeben werden muss und der zuvor gesetzte Modus nicht verändert werden darf, wird dieser Modus zum Wert `LCD_E` addiert und so wieder mitgesendet.

```
time.sleep(E_PULSE)
bus.write_byte_data(DEVICE,GPIOB,LCD_RS*mode)
```

Nach einer weiteren kurzen Verzögerung wird das Bit `LCD_E` wieder auf `0` gesetzt. Dazu sendet die Funktion einfach nur den aktuellen Modus `0x02` oder `0x00` an das Display. Dabei ist das Bit `LCD_E` automatisch `0`.

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

Wie im ursprünglichen Programm folgt eine kurze Verzögerung, damit keine neuen Daten am Display ankommen können, bevor die aktuellen Daten dargestellt sind.

Die Funktionen `lcd_string()` und `lcd_anzeige()` greifen nicht direkt auf die Hardware des LC-Displays zu und können unverändert übernommen werden.

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

Nach der Definition der Funktionen beginnt das eigentliche Programm mit der bereits bekannten Initialisierung des i2c-Bus sowie der beiden Ports des Portexpanders `GPIOA` und `GPIOB`.

```
LCD_INIT = [0x33, 0x33, 0x32, 0x3C, 0x0C, 0x06, 0x01]
```

Der Initialisierungsstring für das Display enthält diesmal die Codes für den 8-Bit-Modus. Die Schleife zur Initialisierung des Displays sowie die Hauptschleife des Programms können unverändert übernommen werden, da auch diese nur über Funktionen und nicht direkt auf die Hardware des LC-Displays zugreifen.