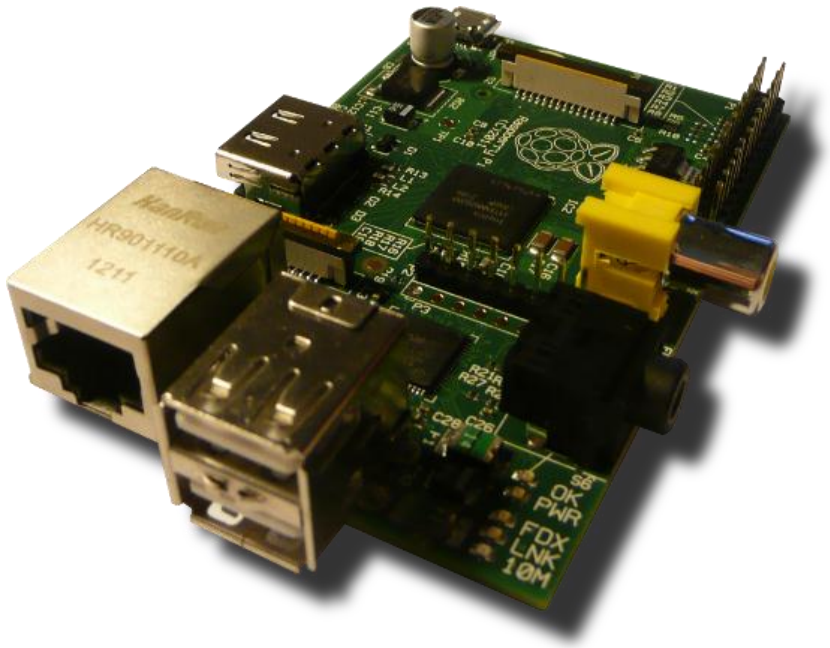
PiMeUp

Raspberry Pi



AddOn

Der Port-Expander MCP23S17 (Teil 1)

Version 1.1,
16.10.2012

www.erik-bartmann.de

Raspberry Pi

Worum geht's?

Hallo zusammen,

in diesem *RasPi-AddOn* möchte ich näher auf den Baustein *MCP23S17* eingehen. Es handelt sich hierbei um einen sogenannten *Port-Expander*, den ihr hier auf dem folgenden Bild seht.

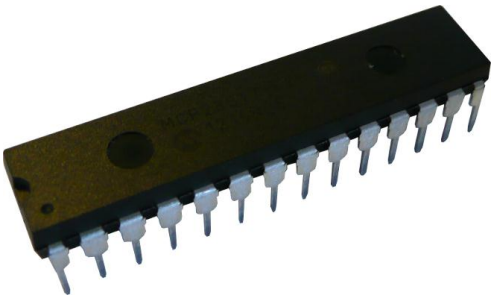


Abbildung 1 Der MCP23S17

Warum wird ein derartiger Baustein benötigt, wirst du dich möglicherweise fragen. Nun, ein *Mikrocontroller* oder auch die *GPIO*-Schnittstelle wie hier beim *Raspberry Pi* verfügt über eine begrenzte Anzahl von *I/O-Ports*. In manchen Situationen kann es also vorkommen, dass du an die physikalischen Grenzen stößt und dir wünscht, dass dir mehr Ports zur Verfügung ständen. Jetzt kommt der *Port-Expander* ins Spiel. Der *MCP23S17* verfügt über zwei Ports mit jeweils *8 I/O* Leitungen. Insgesamt stehen die also *16* Leitungen zur Verfügung. Wir können jetzt natürlich nicht sagen, dass du mit diesem Baustein über *16* zusätzliche *I/O-Ports* verfügst, denn du benötigst zur Ansteuerung des *Port-Expanders* die *SPI* typischen Leitungen der *GPIO*-Schnittstelle des *Raspberry Pi*. Also *MOSI*, *MISO*, *CS* und *CLCK*. Das Interessante ist aber, dass du mehrere *Port-Expander* kaskadieren kannst, so dass du dann wieder über mehr *I/O-Ports* verfügst. In diesem Tutorial wollen wir uns aber erst einmal auf die Ansteuerung eines einzelnen konzentrieren. Wenn du dich noch nicht mit der *SPI*-Schnittstelle vertraut gemacht hast, dann schlage ich vor, dass du dir das *AddOn* zur Ansteuerung des *MCP3008* anschaust, das ich ebenfalls in meinem Downloadbereich anbiete. Schauen wir uns aber jetzt den *Port-Expander MCP23S17* genauer an.

Hier noch ein kurzer Hinweis: Es existiert auch der Baustein *MCP23017*, der über den *I^2C-Bus* angesteuert werden kann. Das ist aber im Moment nicht unser Thema.

Der MCP23S17

Die Pinbelegung des Port-Expanders schaut wie folgt aus:

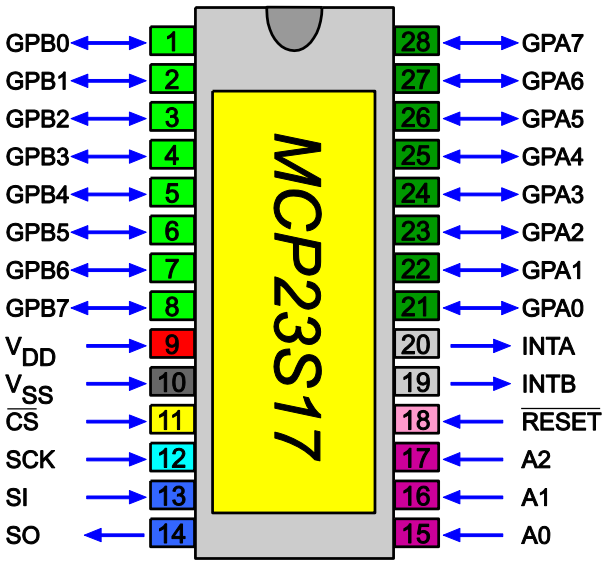


Abbildung 2Die Pinbelegung des MCP 23S17

Das sind ja schon mal eine Menge Pins, die fast an das Volumen eines Mikrocontrollers heranreichen. Aber es handelt sich *nicht* um einen Mikrocontroller. Der *MCP23S17* hat so viele Pins, weil er eben auch so viele Ein- bzw. Ausgänge zu bieten hat und noch so einiges mehr.

Dann kannst du mir doch sicherlich mal erzählen, was dieser Baustein denn alles so leistet.

Yep, das sollte ich tun. Der *MCP23S17* hat folgende Grundfunktionen:

- *16* I/O-Ports mit Interrupt Ausgängen
- Ansteuerung von *8* unterschiedlichen Bausteinen über den *3*-Bit-Bus
- Strom von *25mA* pro I/O
- *SPI*-Clock-Speed bis zu *10MHz*

Die Ein- bzw. Ausgänge sind in zwei getrennten Port-Bänken aufgeteilt. *GPA* und *GPB*, die jeweils über eine *8*-Bit Breite verfügen. Jeder einzelne Pin kann entweder als Ein- oder Ausgang programmiert werden. Wie das funktioniert, das wirst du in Kürze erfahren.

Du kannst ebenfalls mehrere Port-Expander kaskadieren, denn jeder Baustein verfügt über einen *3*-Bit *Adress-Bus*. So ist es dir möglich, 2^3, also *8* unabhängig voneinander arbeitende *Port-Expander* zu betreiben. In unserem Tutorial wollen wir aber lediglich einen einzigen an den *Raspberry Pi* anschließen.

Wie soll das mit dem *Bus* funktionieren? Das habe ich noch nicht ganz verstanden.

Kein Problem! Das ist ganz einfach. Sehen wir uns dazu einfach einmal das komplette *Bus-System* mit allen möglichen *Bus-Adressen* an. Der folgende Plan entspricht *nicht* der Verdrahtung, denn jeder einzelne Baustein wird fest mit einer *Bus-Adresse* versehen. Was du hier siehst, entspricht der *logischen* Adressierung durch die Programmierung. Dazu kommen wir später noch zu sprechen.

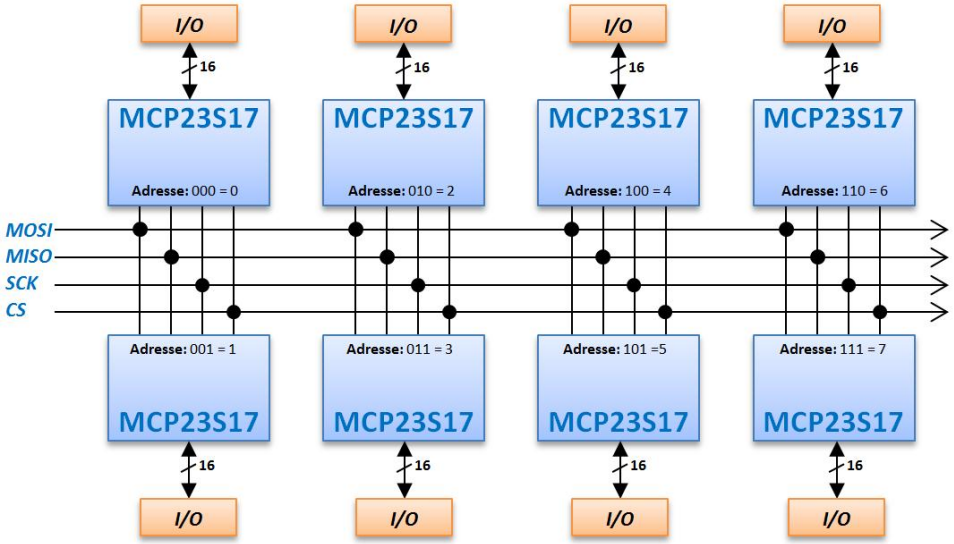


Abbildung 3 Das Bus-System des MCP23S17

Jeder einzelne Bus-Pin (*15*, *16* u. *17*) wird fest mit einem logischen Pegel (*LOW* bzw. *HIGH*) verdrahtet. *LOW* entspricht hierbei V_{SS} (*0V*) und *HIGH* V_{DD} (*3,3V*). Über die Programmierung sagen wir dann, welchen *Port-Expander* wir denn ansprechen möchten. Dazu kommen wir gleich. Sehen wir uns zunächst den Schaltungsaufbau an.

Noch eine Kleinigkeit! Der *MCP3008* hatte u.a. die *SPI*-Bezeichnungen D_{IN} und D_{OUT}. Hier sieht das schon wieder vollkommen anders aus. Es gibt *SI* und *SO*. Wie habe ich das zu verstehen?

Das ist nicht weiter schwer. *SI* bedeutet *Serial Data In* und *SO* bedeutet *Serial Data Out*. Damit du das besser verstehst, setze ich die unterschiedlichen Bezeichnungen einmal untereinander.

Schau her:

Abbildung 4 Allgemeine Bezeichnung der SPI-Kommunikation

Abbildung 5 MCP 3008 Bezeichnung der SPI-Kommunikation

Abbildung 6 MCP 23S17 Bezeichnung der SPI-Kommunikation

Du siehst, es gibt die unterschiedlichsten Bezeichnungen, doch mit ein wenig Überlegung wird es sicherlich funktionieren. Kommen wir nun zum versprochenen Fritzing-Aufbau.

Der Schaltplan

Um einen besseren Überblick zu gewinnen, zeige ich dir am besten zuerst den Schaltplan.

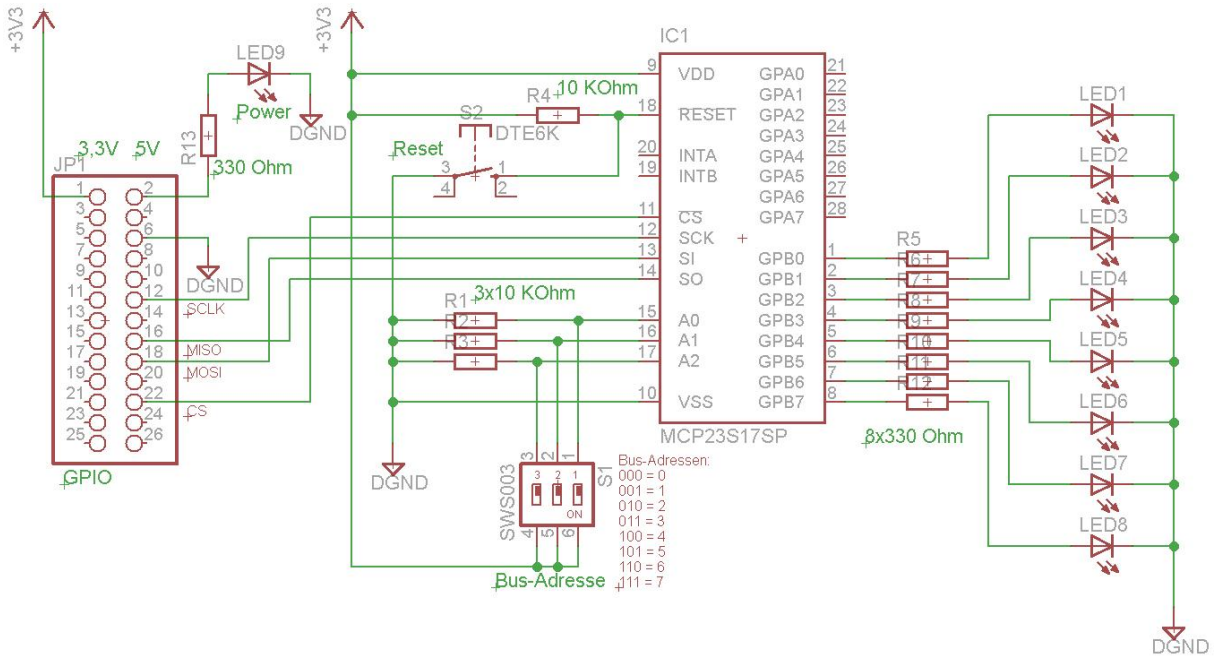


Abbildung 7 Der Schaltplan

Die Stiftleiste auf der linken Seite hat die Bezeichnung *GPIO* und wird über ein geeignetes Flachbahnkabel mit deinem *Raspberry Pi* verbunden. Damit die Sache hinsichtlich der Programmierung mit unterschiedlichen Bus-Adressen besser funktioniert, kannst du sogenannte *DIP-Schalter* verwenden, die direkt auf die Platine aufgelötet werden können. Auf diese Art kannst du die *Bus-Adresse* des *Port-Expanders* sehr einfach und komfortabel anpassen. Die Adressen mit den jeweiligen Schalterstellungen stehen rechts neben dem *DIP-Schalter*, wobei eine *1 = ON* und eine *0 = OFF* bedeutet. Des Weiteren findest du in diesem Schaltplan einen *Reset-Taster*, der den Baustein zurücksetzt. Beide Elemente, also *DIP-Schalter* bzw. *Reset-Taster* habe ich im folgenden *Fritzing-Aufbau* weg gelassen. Du kannst sie aber sehr leicht integrieren.

Der Schaltungsaufbau mit Fritzing

Mit der Software Fritzing, die ich schon beim *AddOn* für den *MCP3008* erwähnt habe, können wir den Schaltungsaufbau sehr gut visualisieren. Ich werde mich in diesem ersten *MCP23S17-AddOn* auf die Ansteuerung der Ausgänge an *Port-B* beschränken. Alles Weitere folgt dann in Kürze in einem weiteren *AddOn*. Wir machen das am besten Schrittchen für Schrittchen, so dass auch niemand auf der Strecke bleibt.

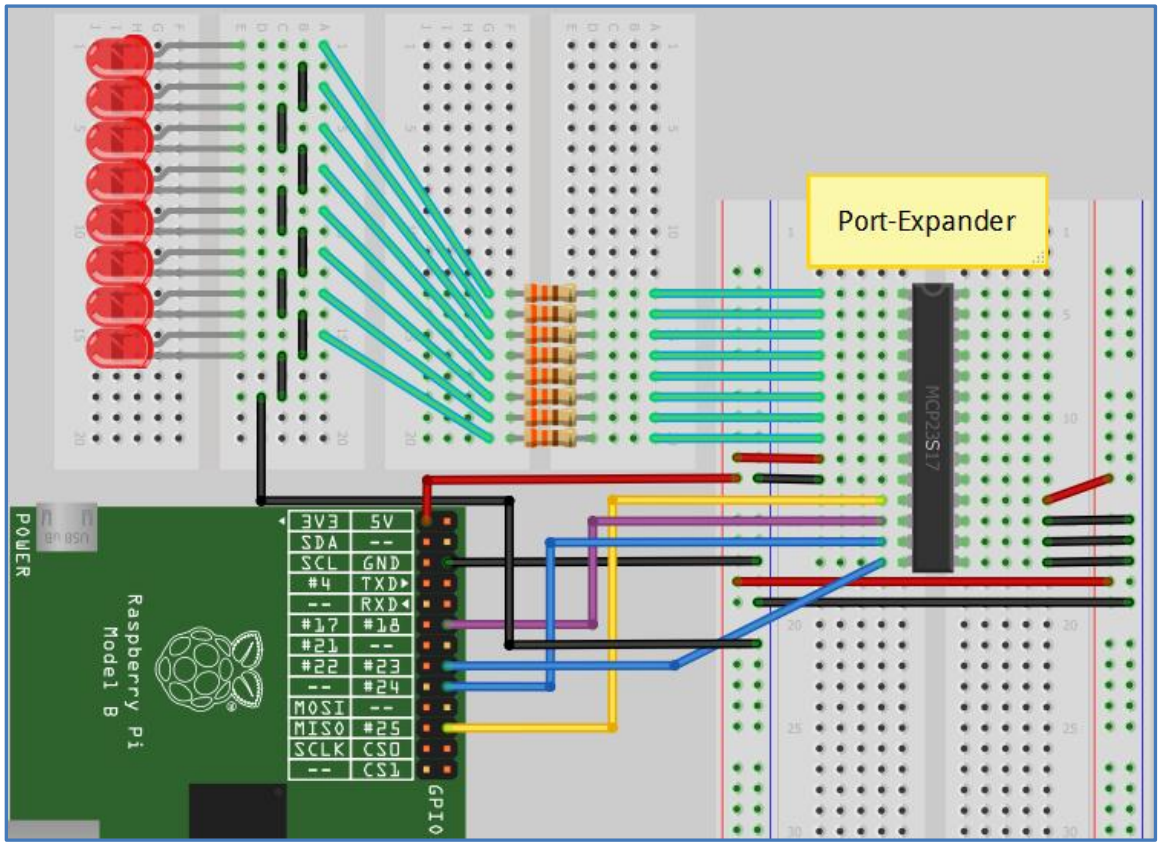


Abbildung 8 Schaltungsaufbau mit Fritzing

Auf der linken Seite siehst du wieder unseren *Raspberry Pi* - jedenfalls ausschnittsweise -, der über die *GPIO-Schnittstelle* mit dem auf einem Breadboard aufgesteckten *Port-Expander MCP23S17* verbunden ist. Fürs Erste wollen wir lediglich die an *Port-B* angeschlossenen LEDs ansteuern. Hast du das verstanden, können wir weiter zu komplexeren Dingen gehen.

Hinsichtlich der Vorwiderstände habe ich noch eine Frage. Warum hast du gerade *330 Ohm* Widerstände genommen? Bist du dir ganz sicher, dass das auch ok so ist? Wir dürfen doch nicht über *25mA* pro Pin kommen!

Das ist ein guter Einwand! Aber du brauchst dir keine Gedanken zu machen, denn wir kommen mit *330 Ohm* nicht an die Grenzen des zulässigen Maximalstroms von *25mA*. Am besten zeige ich dir dazu einmal das Ersatzschaltbild für die Ansteuerung eines einzelnen Pins.

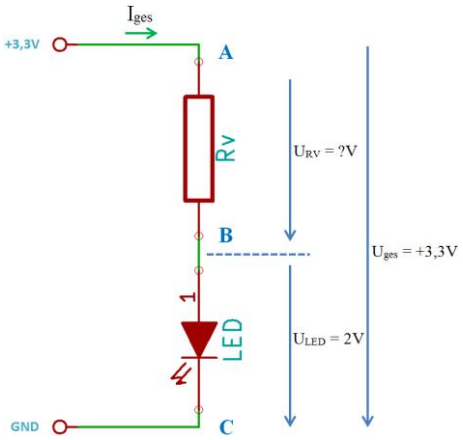


Abbildung 9 Eine LED mit Vorwiderstand

Um den Wert eines Widerstandes zu berechnen, wird die Formel für das *Ohm'sche Gesetz* verwendet, die da lautet:

$$R = \frac{U}{I}$$

An der Leuchtdiode fallen ca. *2V* ab, was dann bedeutet, dass sich U_{RV} wie folgt berechnet:

$$U_{RV} = U_{ges} - U_{LED} = 3{,}3V - 2V = 1{,}3V$$

Jetzt wollen wir mal sehen, ob ich mit meinen *330 Ohm* richtig gelegen habe. Wie berechnet sich nun aber den Strom? Ganz einfach. Wir stellen die Formel nach *I* um und setzten die entsprechenden Werte ein.

$$I = \frac{U}{R} = \frac{U_{RV}}{R_V} = \frac{1{,}3V}{330\Omega} = 3{,}9mA$$

Der Wert von *3,9mA* bewegt sich also vollkommen im grünen Bereich und ist weit unterhalb der zulässigen I_{max} = *25mA*. Hoffentlich ist deine Frage damit beantwortet. Auf diese Weise kannst du auch den Strom für andere Vorwiderstände berechnen, falls dir die LED zu hell oder zu dunkel erscheint. Versuche aber nicht an das Limit von I_{max} heran zu kommen. Ich denke, dass du jetzt genug Informationen beisammen hast, dass wir uns der Programmierung zuwenden können.

Die Programmierung

Wie auch schon beim *MCP3008* kommt es natürlich auf das *Timing* an. Ein Blick in das Datenblatt ist sicherlich ratsam, auch wenn es den einen oder anderen aufgrund der Informationsvielfalt vielleicht verwirren mag. Ich suche dir die interessanten Stellen heraus, so dass es hoffentlich etwas einfacher wird.

Welche Programmiersprache verwenden wir?

Da wir das Ganze wieder in der Programmiersprache *Python 2.7* programmieren werden, hier der Hinweis, dass ich die Library-Version *RPi.GPIO 0.4.1a* verwendet habe.

Vorbereitungen

Folgende Schritte sind notwendig, damit Du auf Deinem *Raspberry Pi* in der Programmiersprache *Python* deine Programme entwickeln kannst.

Python-Dev installieren

Falls es nicht schon geschehen ist, musst du *Python-Dev* über die folgende Kommandozeile installieren:

```
# sudo apt-get install python-dev
```

Die Raspberry Pi GPIO-Library installieren

Diese Library findest Du auf der folgenden Seite im Internet.

http://code.google.com/p/raspberry-gpio-python/downloads/list

Du wirst sehen, dass sich dort eine ganze Anzahl von unterschiedlichen Versionen befindet. Für meine Versuche habe ich – wie schon erwähnt – diesmal die Version *0.4.1a* verwendet. Hast Du die gewünschte Version in Deinem *Home-Verzeichnis* heruntergeladen, dann liegt sie in der folgenden Form vor:

RPi.GPIO-0.4.1a.tar.gz

Du kannst an der Endung *gz* erkennen, dass die Datei noch *komprimiert* ist. Mit dem folgenden Kommando *dekomprimierst* du die Datei:

```
# gunzip RPi.GPIO-0.4.1a.tar.gz
```

Die einzelnen Dateien bzw. Verzeichnisse sind jedoch dann noch in einer einzigen Datei mit der Endung *tar* zusammengefasst bzw. archiviert. Du musst über das folgende Kommando das Archiv entpacken.

 # tar –xvf RPi.GPIO-0-4-1a.tar

Jetzt kannst Du über das *cd*-Kommando in das neu entstandene Verzeichnis wechseln. Dort befindet sich u.a. eine Datei mit dem Namen *setup.py*. Es handelt sich um eine Installationsdatei von *Python*, über die Du das *Python-GPIO*-Paket installieren kannst. Starte über die folgende Befehlszeile die Installation:

```
# sudo python setup.py install
```

Nach der erfolgreichen Installation können wir unmittelbar mit der Programmierung beginnen.

Das Python-Programm

Hinsichtlich der Programmierung in *Python* gibt es die unterschiedlichsten Entwicklungsumgebungen, die du nutzen kannst. In meinem Buch über den *Raspberry Pi* habe ich u.a. auch *Stani's Python Editor* – kurz *SPE* genannt – verwendet. Du kannst ihn mit der folgenden Befehlszeile installieren:

```
# sudo apt-get install spe
```

Das ist schon ein interessantes und mächtiges Werkzeug, doch auch der unter *Raspian* (*Debian Wheezy*) vorinstallierten Texteditor *Nano* ist sicherlich einen Blick wert.

Es ist aber in jedem Fall zu bedenken, dass das Python-Skript mit *Root-Rechten* gestartet werden muss. Also angenommen, du hast das Skript *control_mcp23s17.py* genannt, dann musst Du es wie folgt starten:

```
# sudo python control_mcp23s17.py
```

Ich werde in kleinen Schritten vorgehen und jeden zusammenhängenden Skriptblock detailliert erläutern. Das komplette Skript findest du im Downloadbereich auf meiner Internetseite.

```
11  import time
12  import RPi.GPIO as GPIO
13
14  GPIO.setmode(GPIO.BCM)
15  GPIO.setwarnings(False)
```

Zu Beginn müssen wir *2* Libraries einbinden, damit unser Skript auch ohne Probleme läuft. Die *time*-Library wird später genutzt, um über den *sleep*-Befehl eine Pause im Ablauf einzulegen. Die *RPi.GPIO*-Library kapselt die komplette Funktionalität hinsichtlich der Ansteuerung der *GPIO-Schnittstelle*. Mit Hilfe der *setmode*-Methode teilen wir der Library mit, dass wir anstelle der *Pin-Nummern*, die *GPIO-Nummern* in der Programmierung

verwenden möchten. Statt der Pin-Nummern, möchte ich die *GPIO-Bezeichnungen* verwenden. Deshalb füge ich die folgende Zeile zu Beginn ein:

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

In der *RPi.GPIO-Library 0.4.1a* kannst du jetzt mit *HIGH*- bzw. *LOW*-Pegeln arbeiten. Das erfolgt über die Konstanten *GPIO.HIGH* bzw. *GPIO.LOW*, die du gleich im Programm finden wirst. Kommen wir zu einigen Variablen, die *Adressen* definieren und die wir für die Ansteuerung des *Port-Expanders* benötigen. Ich gehe gleich im Detail darauf ein.

```
17  # MCP23S17 Werte
18  SPI_SLAVE_ADDR = 0x40
19  SPI_IOCTRL     = 0x0A
20  SPI_IODIRA     = 0x00
21  SPI_IODIRB     = 0x01
22  SPI_GPIOA      = 0x12
23  SPI_GPIOB      = 0x13
```

Fangen wir doch einfach mit der *Zeile 18* an. Dort steht

```
SPI_SLAVE_ADDR = 0x40
```

Ich hatte dir gesagt, dass du bis zu *8* separate *Port-Expander* anschließen kannst. Jeder einzelne muss natürlich separat ansteuerbar sein, was über eine sogenannte *Adresse* erfolgt. Der *MCP22S17* verfügt über einen *3-Bit-Bus*, so dass wir vor der Inbetriebnahme der Schaltung, dem Baustein eine Adresse zuweisen müssen. Die niedrigste Adresse lautet *0*, die höchste *7*, was in Summe *8* ergibt. Für unser Beispiel habe ich die niedrigste Adresse *0* verwendet.

Das habe ich verstanden! Doch warum schreibst du den Wert *0x40* und weist ihn der Slave-Adresse zu? Das ist mir schleierhaft!

Das ist wahrlich etwas merkwürdig, hat aber einen ganz einfachen Hintergrund. Bevor wir mit dem Versenden von Informationen hinsichtlich der Ansteuerung der angeschlossenen LEDs beginnen können, müssen wir den *Port-Expander* konfigurieren. Dazu gibt es u.a. ein *Control-Byte* mit einer Breite von *8-Bit*, deren einzelnen Bits jeweils eine besondere Bedeutung zukommt.

Schau her:

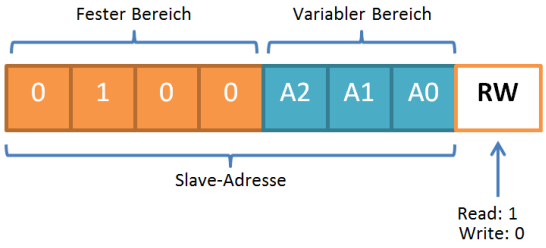
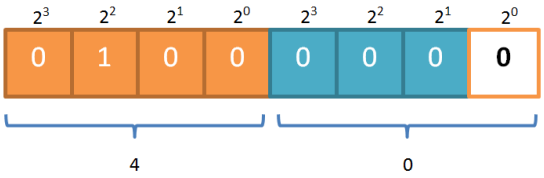


Abbildung 10 Das Control-Register

Beginnen wir mit den linken *4 Bits*, die eine feste Kombination vorweisen, also nicht angepasst werden können, gefolgt von unseren *3 Adress-Bits*, denen wir den Wert *0* zuweisen wollen. Am Ende steht das *Read/Write-Bit*, das wir fest mit *0* versehen, da wir nur schreiben wollen. Das werden wir im *AddOn 2. Teil*, wenn wir auch Werte lesen möchten, noch flexibler gestalten. Wie kommen wir also zu dem Hex-Wert *0x40*? Ganz einfach!



Wie und wann wir diesen Wert und auch die anderen der Programmzeilen *18* bis *23* zum *Port-Expander* versenden, dass erkläre ich etwas später. Dieser *8-Bit* breite Block wird *OP-Code* (*Operation-Code*) genannt. Er besagt, welcher *Port-Expander* angesprochen werden soll. Den Hex-Wert müssen wir per *SPI* an den *Port-Expander* versenden. Wie das funktioniert… das wirst du gleich sehen, wenn ich dir alle notwendigen Informationen gegeben habe.

Danach geht es darum zu bestimmen, welche Ports bzw. Pins als *Eingänge* bzw. *Ausgänge* arbeiten sollen. Diese und noch viele andere Informationen werden in internen Registern vorgehalten. Jedes dieser Register ist mit einer *Adresse* versehen. Die *Adresse* habe ich ja zu Beginn des Programms definiert. Es sind einige dabei, die wir im Moment nicht benötigen. Da wir *Port B* komplett als *Ausgang* programmieren möchten, schauen wir uns die *Zeile 21* an. Dort steht die Adresse *0x01*.

```
20    SPI_IODIRA      = 0x00
21    SPI_IODIRB      = 0x01
```

Genau *den* Wert müssen wir ebenfalls per *SPI* an den *Port-Expander* versenden. Der *8-Bit* breite Hex-Code wird - wer hätte es gedacht – *Adresse* genannt.

Zu guter Letzt müssen wir noch die *Daten* an das ausgewählte Register senden. Welche Pins an *Port B* sollen denn als Ausgänge programmiert werden? Na alle!

Dafür sind die die folgenden Wert vorgesehen:

- ***0***: Ausgang
- ***1***: Eingang

Demzufolge müssen wir, da alle Pins als Ausgänge arbeiten sollen, den Wert *0x00* per *SPI* an den *Port-Expander* versenden. Wir nennen diesen Hex-Wert *Daten*.

Wir können also zusammenfassend sagen, dass wir immer drei *8-Bit* breite Blöcke per *SPI* an den *Port-Expander* versenden, um mit ihm in Verbindung zu treten.

Bevor wir weiter auf die SPI-Kommunikations-Funktionen eingehen, hier der Definitionsblock, der festlegt, über welche Pins die SPI-Kommunikation erfolgt.

```
25  # MCP23S17-Pins
26  SCLK        = 18 # Serial-Clock
27  MOSI        = 24 # Master-Out-Slave-In
28  MISO        = 23 # Master-In-Slave-Out
29  CS          = 25 # Chip-Select
```

Schauen wir uns jetzt den entsprechenden Code zum Versenden der Informationen per *SPI* an den *Port-Expander* genauer an.

```
35  def sendValue(value):
36      # Value senden
37      for i in range(8):
38          if (value & 0x80):
39              GPIO.output(MOSI, GPIO.HIGH)
40          else:
41              GPIO.output(MOSI, GPIO.LOW)
42          # Negative Flanke des Clocksignals generieren
43          GPIO.output(SCLK, GPIO.HIGH)
44          GPIO.output(SCLK, GPIO.LOW)
45          value <<= 1 # Bitfolge eine Position nach links schieben
46
47  def sendSPI(opcode, addr, data):
48      # CS aktive (LOW-Aktiv)
49      GPIO.output(CS, GPIO.LOW)
50
51      sendValue(opcode) # OP-Code senden
52      sendValue(addr)   # Adresse senden
53      sendValue(data)   # Daten senden
54
55      # CS nicht aktiv
56      GPIO.output(CS, GPIO.HIGH)
```

Der Aufruf erfolgt über die Funktion *sendSPI* mit den eben genannten Informationen, also

- OP-Code
- Adresse
- Daten

die als *Argumente* in der angegebenen Reihenfolge der Funktion übergeben werden müssen. Die Funktion *sendSPI* ruft ihrerseits die Funktion *sendValue* auf, die für die eigentliche Übertragung über die *GPIO-Schnittstelle* verantwortlich ist.

Sie schiebt die einzelnen Bits des übergebenen Wertes in Richtung *Port-Expander*, die dort an *Pin 13* ankommen und verarbeitet werden. Ein paar Worte noch zum *Timing*. Jedes einzelne Bit wird innerhalb der Funktion *sendValue* mit einer *negativen Flanke* des *Clock-Signals* quittiert, das beim *Port-Expander* an *Pin 12* ankommt. Die komplette Sequenz von *OP-Code*, *Adresse* und *Daten* wird durch ein entsprechendes *ChipSelect*-Signal innerhalb der Funktion *sendSPI* begleitet, was *LOW-Aktiv* ist und wie folgt ausschaut:

Abbildung 11 Zeitlicher Verlauf der Datenübertragung im Zusammenspiel mit dem CS-Signal

Zwischen den Zeitmarken t_0 und t_1 findet die Übertragung statt.

Kommen wir zum Hauptprogramm:

```
58  def main():
59      # Pin-Programmierung
60      GPIO.setup(SCLK, GPIO.OUT)
61      GPIO.setup(MOSI, GPIO.OUT)
62      GPIO.setup(MISO, GPIO.IN)
63      GPIO.setup(CS,   GPIO.OUT)
64
65      # Pegel vorbereiten
66      GPIO.output(CS,   GPIO.HIGH)
67      GPIO.output(SCLK, GPIO.LOW)
68
69      # Initialisierung de MCP23S17
70      sendSPI(SPI_SLAVE_ADDR, SPI_IODIRB, 0x00) # GPPIOB als Ausgaenge programmieren
71      sendSPI(SPI_SLAVE_ADDR, SPI_GPIOB, 0x00)  # Reset des GPIOB
72
73      while True:
74          for i in range(len(ledPattern)):
75              sendSPI(SPI_SLAVE_ADDR, SPI_GPIOB, ledPattern[i])
76              time.sleep(0.5)
```

In den Zeilen *60* bis *63* werden die für die *SPI-Kommunikation* benötigten Pins konfiguriert, was über die *GPIO.setup*-Funktion erfolgt. Sie legt fest, welche Pins der *GPIO-Schnittstelle* als *Eingänge* und welche als *Ausgänge* arbeiten sollen.

Um einen definierten Ausgangszustand der *ChipSelect*- und *Clock-Signale* zu erhalten, werden diese in den Zeilen *66* und *67* vorbereitet.

Die Initialisierung des *Port-Expanders* erfolgt in den Zeilen *70* und *71* mit den schon beschriebenen Hintergründen.

In der Endlosschleife, die in Zeile *73* beginnt, werden die einzelnen LEDs angesteuert. Das erfolgt in meinem Beispiel nach einem Muster, das ich in einem *Python-Tupel* mit mehreren Elementen hinterlegt habe.

```
31  ledPattern = (0b01010101, 0b10101010, 0b01010101, 0b10101010, \
32                0b00000001, 0b00000010, 0b00000100, 0b00001000, \
33                0b00010000, 0b00100000, 0b01000000, 0b10000000)
```

Jedes einzelne Bit steht für eine LED an *Port B* des *Port-Expanders*. Die *for*-Schleife iteriert über das *Tupel* und wählt somit jedes einzelne Element an und übergibt sie der Funktion *sendSPI*. Die angeschlossenen LEDs beginnen entsprechend zu blinken. Spiele ein wenig mit diesen Werten, um die Auswirkungen zu erkennen und zu verstehen.

Bezugsquellen

Den *MCP23S17* kannst du z.B. unter der folgenden Adresse beziehen:

http://www.reichelt.de/ICs-M-MN-/MCP-23S17-E-SP/3//index.html?ACTION=3&GROUPID=2914&ARTICLE=90047&SHOW=1&START=0&OFFSET=16&

Vorschau

Ich plane weitere *AddOns* zur Ansteuerung des *Port-Expanders*, denn wir haben uns ja bisher nur mit der *Ausgabe*, also der Ansteuerung von LEDs befasst. Was ist mit der *Eingabe*? Wir wollen z.B. *Taster* anschließen. Aus diesem Grund werde ich ein spezielles *Prototyping-Board* vorstellen. Schau her:

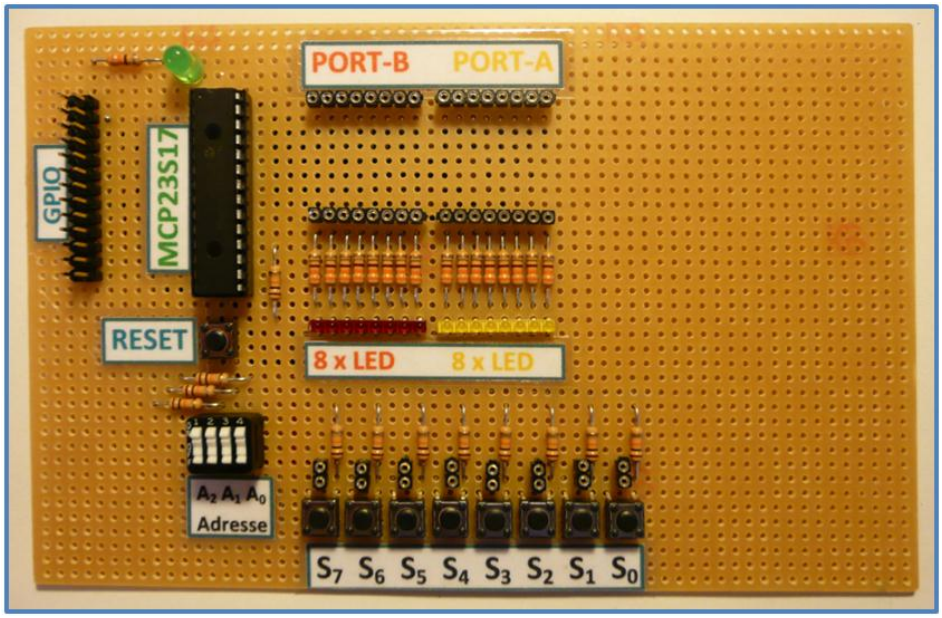


Abbildung 12 Das Prototyping-Board

Dieses Board wird über ein Flachbahnkabel direkt mit deinem *Raspberry Pi* verbunden. Es besitzt *2 x 8 LEDs* und *8 Taster*, die flexibel mit den *16-Ports* des *Port-Expanders* über flexible Steckbrücken verbunden werden können.

Auch die Ansteuerung über eine grafische Oberfläche mittels *PyQt* ist sicherlich sehr spannend. Es lohnt sich also, von Zeit zu Zeit mal einen Blick auf meine Internetseite zu werfen. Natürlich werde ich neue *AddOns* auch über *Twitter* oder *facebook* posten bzw. ankündigen.

Schlusswort

Jetzt wünsche ich dir viel Spaß beim Experimentieren und ich würde mich freuen, wenn du von Zeit zu Zeit einen Blick auf meine Internetseite werfen würdest. Dort findest du sicherlich ein paar interessante *AddOns* zu meinen verschiedenen Themen bzw. Büchern.

Erik Bartmann

www.erik-bartmann.de

http://www.oreilly.de/catalog/raspberrypiger/

http://www.oreilly.de/catalog/elekarduinobasger/

http://www.oreilly.de/catalog/processingger/