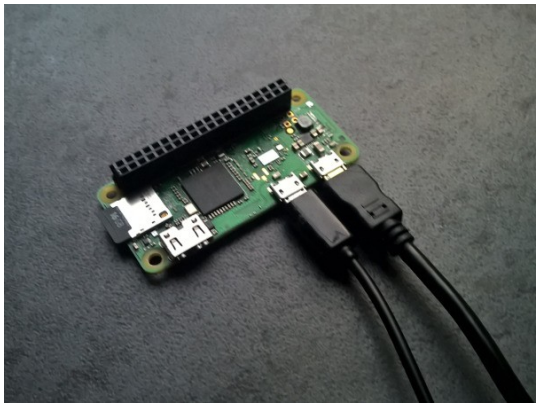
Raspberry Pi Zero W als smarter USB-Stick

Durch sein Built-In-WiFi kann man aus dem Pi Zero W mit recht einfachen Mitteln einen USB-Speicherstick mit erweiterten Funktionen machen. Unser Ziel im kleinen Projekt war ein „Pi-Stick“, den man aus der Ferne mit immer neuen Inhalten beladen kann, ohne ihn abziehen zu müssen. Und wie von einem handelsüblichen USB-Stick gewohnt, mit einem einzigen USB-A-Stecker verbinden kann, um auf sonstige Verkabelung verzichten zu können. Der SD-Karte des Pi wird dabei Platz für den Massenspeicher abgezweigt. Anleitungen zu diesem Thema gibt es schon, hier soll alles von Grund auf für den Einsteiger nachvollzogen und die Lösung auf eigene Bedürfnisse angepasst werden.

[b06.jpg] „Pi-Stick“

Für das grundlegende Setup bestücken wir den Pi zuerst mit einer MicroSD-Karte mit Raspbian-Image. Um den Zero nicht an Tastatur und Monitor anschließen zu müssen, kann man wie gewohnt einige Dateien editieren, so dass er sich mit dem eigenen WLAN verbindet und nach dem ersten Boot Zugriff per *ssh* möglich wird. Dafür gibt es Anleitung im Web.

Alternativ kann man ihn auch mit X-Desktop hochfahren und an einem angeschlossenen Monitor Netzwerk- und sonstige Einstellungen vornehmen. Nachdem wir wie immer einen Hostnamen festgelegt und die Userpartition auf der SD-Karte erweitert haben, soll noch eingestellt werden, dass das nächste Mal standardmäßig in die Kommandozeile gebootet wird. Nach dem Neustart reicht uns für die Kommunikation ab da an eine *ssh*-Verbindung im Netzwerk.



[b01.jpg] Klassische Beschaltung am OTG- und Stromversorgungsanschluss

Eine vollständige Raspbian-Installation bringt immer auch die Desktopanwendungen mit, die wir hier nicht benötigen. Wer will, kann deshalb etwas Platz schaffen und zum Beispiel Libreoffice, Wolfram und Mathematica entfernen:

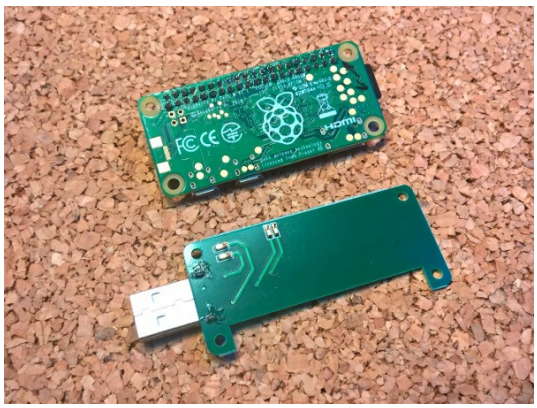
```
$ sudo apt-get purge libreoffice*
$ sudo apt-get purge wolfram-engine
$ sudo apt-get clean
$ sudo apt-get autoremove
```

Viel zu konstruieren haben wir in unserem Projekt nicht. Um einen „richtigen“ USB-Dongle herzustellen, nutzen wir einen Bausatz, welcher ein USB-A-Addon und Gehäuseschalen enthält. Schön, dass sich die

Addon-Platine mit Pogo-Pins zu vier Inseln auf der Unterseite des Pi verbindet, so braucht nichts gelötet zu werden.

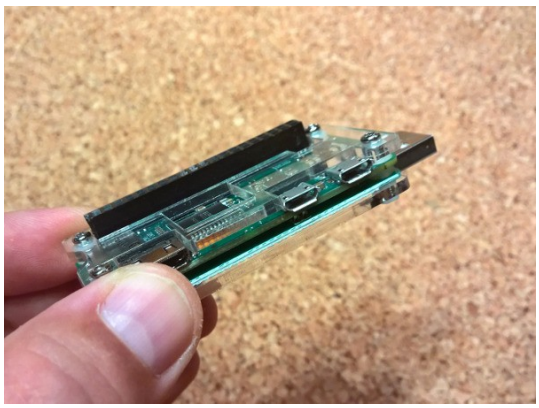
[b02.jpg] Bauteile der USB-Addon-Platine

Beim näheren Hinschauen kann man die Beschaltung nachvollziehen. Die für die Stromversorgung des Pi zuständige Micro-USB-Buchse ist mit den beiden Versorgungsleitern des großen USB-A-Steckers des Addon verbunden. Die Datenleitungen des Steckers führen wiederum auf die zweite Micro-USB-Buchse des Pi. Eine gute Idee, um doppelte Bestromung zu vermeiden und sich wie bei einem gewöhnlichen Stick nur noch über den USB-Stecker sowohl für Stromversorgung als auch Datenaustausch zu verbinden.



[b03.jpg] USB-Addon und Pi über Pogo-Pins miteinander verbinden

Beim Pi Zero funktioniert das bei fast allen Geräten, da sein Strombedarf deutlich unter der von USB-Anschlüssen bereitgestellten 500mA liegt. Natürlich kommt man damit in ein neues Problem rein, denn wenn das genutzte Gerät plötzlich abgeschalten wird, verliert auch der Pi seine Stromversorgung, und es kann zu Beschädigungen der SD-Karte kommen. Um das zu verhindern, kann man jedoch wie gehabt ein externes Netzteil für den Pi nutzen.



[b04.jpg] Der Stick ist mechanisch zusammengebaut

Wo nun das Konstruktive erledigt ist, soll es mit den speziellen softwaretechnischen Belangen weitergehen. Es muss der USB-Treiber für den Gadget-Mode aktiviert werden, dazu zuerst die Datei */boot/config.txt* editieren und an deren Ende die Zeile

dtoverlay=dwc2

anfügen. Die */etc/modules* erhält den Eintrag

dwc2 .

Oder diesen Eintrag auskommentieren, sollte er etwa schon nach der „i2c-dev“-Zeile stehen. Nach dem nächsten Start sollte der dwc2-Treiber geladen sein.

Um einen Massenspeicher zu erhalten, muss unserem Gadget zuerst ein Container-File verpasst werden. So zweigt man 4GB von der SD-Karte des Pi Zero ab und erzeugt ein leeres Binärfile „piusb.bin“:

```
$ sudo dd bs=1M if=/dev/zero of=/piusb.bin count=4096
```

Anschließend nutzt man „mkdosfs“, um es als FAT32-Filesystem zu formatieren:

```
$ sudo mkdosfs /piusb.bin -F 32 -I
```

Jetzt kann es gemountet und permanent nach *fstab* eingetragen werden. Wir mounten hier nach */mnt/usb_share* :

```
$ sudo mkdir /mnt/usb_share
```

In die Partitionstabelle unter */etc/*fstab am Ende die Zeile eintragen:

```
/piusb.bin /mnt/usb_share vfat users,umask=000 0 2
```

Mittels

```
$ sudo mount -a
```

laden wir die modifizierte Tabelle nach.

Falls noch nicht gemacht, sollte man jetzt den Pi herunterfahren und schon einmal an einen USB-Port vom Laptop oder an ein TV-Gerät anstecken. Unser Stick holt sich nun die Stromversorgung von der USB-Buchse vom Gerät. Da dafür schon alles vorbereitet wurde, greifen wir spätestens jetzt per *ssh* von einem entfernten Rechner im heimischen Netzwerk zu. Auf einem Windows-Rechner kann man dafür *Putty* verwenden, auf einem Linux-Rechner öffnet man ein Terminal und gibt wie gewohnt

```
$ ssh pi@[piname]
```

und im Anschluss das Passwort ein.

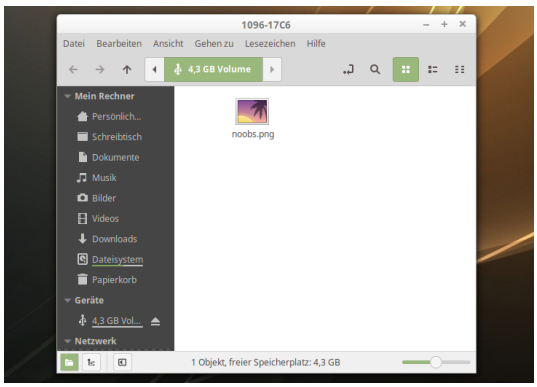
Wer jetzt den Massenspeichermode testen möchte, kann nun schon ein File nach */mnt/usb_share* downloaden:

```
$ cd /mnt/usb_share
$ wget https://github.com/raspberrypi/documentation/blob/master/installation/images/noobs.png
$ sync
```

Damit haben wir uns eine PNG-Datei von einem Server in unser gemountetes Verzeichnis geholt. Jetzt muss der Massenspeichermodus aktiviert werden:

```
$ sudo modprobe g_mass_storage file=/piusb.bin stall=0 ro=1
```

Und es wird richtig spannend, denn wenn bisher alles funktioniert hat, sollte der Smart-TV nun die Bilddatei erkennen oder am Laptop ein Filedialog aufpoppen, der das Volume (bei uns 4GB groß) und die Datei darin anzeigt. Geschafft!



[b05.png] Geschafft – Der Host zeigt den Inhalt des Smartstick an.

Den smarten Stick nun von Ferne mit immer neuen Bildern oder anderen Inhalten zu bestücken, das kann man nicht nur per *ssh* bewerkstelligen. Mit dem Pi als Unterbau hat man da alle Möglichkeiten, die man auch bei anderen Netzwerkrechnern hat. So ist es nun möglich, unseren Stick als Samba-Share ins Netzwerk einzubinden, so dass man ihn als Freigabe im Netzwerkordner auf anderen Rechnern finden kann. Dort kann man dann einfach per Dateiexplorer auf dem Desktop neue Dateien einstellen. Oder man macht das textbasiert auf der Kommandozeile, gerne auch automatisiert per Skript. Und per *ftp* geht das Ganze auch.

Bleibt zu organisieren, wie man die Auffrischung der Inhalte dem Host mitteilt, ohne unseren Stick vom jeweiligen Gerät abzuziehen und wieder anzustecken, denn auf einem Smart-TV beispielsweise wird nur nach dem Anstecken die Anzeige aktualisiert. So einen Mechanismus kann man auch selbst bauen, aber es gibt bereits gute Lösungen dazu. Deshalb soll auf ein Python-Skript, auf Git von David Honess veröffentlicht, zurückgegriffen werden, was sich relativ einfach installieren und unter */usr/local/share* ablegen lässt:

```
$ cd /usr/local/share
$ sudo wget http://rpf.io/usbzw -O usb_share.py
$ sudo chmod +x usb_share.py
```

Außerdem wird noch die Python-Library für den Watchdog benötigt:

```
$ sudo apt-get install python-watchdog
```

Was passiert in diesem Skript? Der Pfad */mnt/usb_share* , unter welchem unser Massenspeicher gemounted ist, wird mittels Observer überwacht und der Eventhandler hängt beim Kopieren oder Löschen von Dateien, das Volume neu ein, genau so, wie wir es beim ersten Mal mittels *modprobe* oben schon änfänglich gemacht haben.

Dazu muss das Ganze im Background-Service laufen und soll deshalb noch die entsprechenden Einträge in einem neuen File (bei uns *usbshare.service*) unter */etc/systemd/system* bekommen:

```
$ cd /etc/systemd/system
$ sudo nano usbshare.service
```

Nun per Editor eintragen:

```
[Unit]
Description=USB Share Watchdog

[Service]
Type=Simple
ExecStart=/usr/local/share/usb_share.py
Restart=always

[Install]
WantedBy=multi-user.target
```

Abschließend muss der Service noch registriert und eingeschalten werden:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable usbshare.service
$ sudo systemctl start usbshare.service
```

Damit haben wir nun bewerkstelligt, dass im Abstand von 30 Sekunden ein Reconnect zum Gerät erfolgt, an dem wir unseren Smartstick gesteckt haben. Smart-TV oder elektronischer Bilderrahmen zeigen neue Bilder zuverlässig an. So funktioniert in der Praxis unser Stick nun in Kombination mit einem Bilderrahmen. Der ist per Zeitschaltuhr ab Nachmittag aktiv.

[b07.jpg] Bilderrahmen mit „Pi-Stick“

Das wäre dann eine von diversen Einsatzmöglichkeiten. Nebenbei löst sich hier auch unser eingangs erwähntes Problem der unkontrollierten Abschaltung. Denn ein paar Minuten, bevor der Bilderrahmen samt USB-Anschluss durch die Zeitschaltuhr am Abend wieder stromlos gemacht werden, fährt der Pi auf dem Stick per Cronjob kontrolliert herunter und es drohen keine Beschädigungen der SD-Karte.

Unser Bilderrahmen bekommt damit „WLAN-Anschluss“, gerät ins „Internet der Dinge“ und lässt sich aus unserer Hausautomation heraus beeinflussen. Tag für Tag können frische Bilder aus dem Netzwerk nach bestimmten Regeln hochgeladen werden. So hat man immer eine neue Stimmung, wenn man nach Hause kommt. Und bestimmt fallen euch noch weitere Anwendungen für so einen „Pi-Stick“ ein...