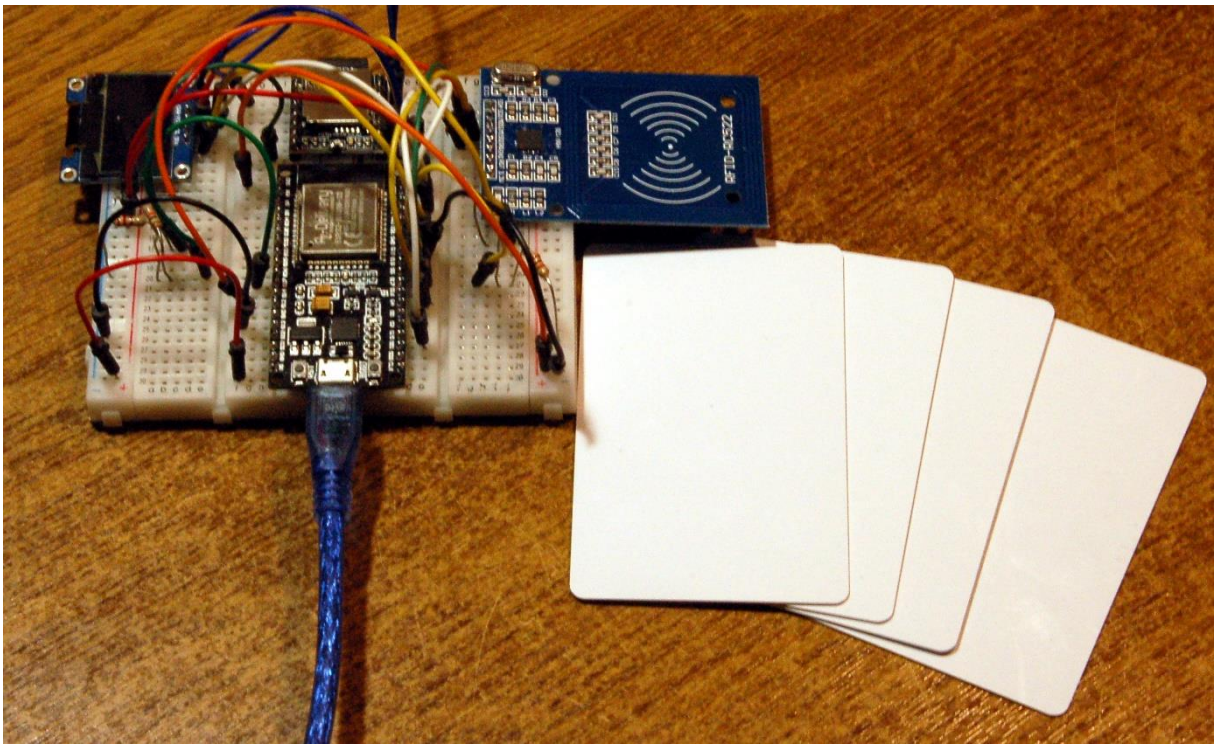

MP3-Player - Aufbau

Dieser Beitrag ist auch als PDF-Dokument verfügbar.

Es ist schon eine ganze Weile her, dass ich eine Weihnachtsverlosung mit RFID-Tags für den ESP32 programmiert habe. Nicht ganz so lang liegt der Beitrag mit der Sprachausgabe von Messergebnissen zurück. Die Sprachfetzen hatte ich mit dem Handy aufgenommen und auf eine SD-Karte übertragen. Nun könnte man natürlich auch Musikstücke auf der SD-Karte speichern und über RFID-Tags abrufen – fertig ist ein MP3-Player mit Ausbaupotenzial. Für die Programmierung habe ich verschiedene Ansätze ausprobiert. Was anfangs vielversprechend und interessant aussah, ein Ansatz mit Threading, musste schließlich zugunsten der herkömmlichen Programmierung durch Schleifen aufgegeben werden, weil die Performance nicht meinen Vorstellungen entsprach. Diese Lösung stelle ich Ihnen jetzt vor, mit dem neuen Beitrag aus der Reihe

MicroPython auf dem ESP32 und ESP8266

Heute:

MP3-Player mit RFID

Genau genommen kam ich auf die Idee zu diesem Beitrag durch die Artikel zum Thema serielle Schnittstellen am ESP32/ESP8266 dem Raspberry Pi Pico und dem PC. Ich wollte noch einige Anwendungen der RS232-Ports zeigen. Nun, der DFPlayer mini, der hier zum Einsatz kommt, wird über die serielle Schnittstelle angesteuert. Und weil ein bidirektionaler Datenaustausch gebraucht wird, kommen eigentlich nur ein ESP32 oder ein Raspberry Pi Pico als Controller in Frage. Über einen RS232-TTL-Converter wäre auch ein PC in der Lage direkt mit einem DFPlayer mini zu parlieren, aber das lasse ich mal außen vor. Damit sind wir auch schon bei der Hardwareliste angekommen.

1	ESP32 Dev Kit C unverlötet oder ESP32 NodeMCU Module WLAN WiFi Development Board oder NodeMCU-ESP-32S-Kit
1	1,3 Zoll OLED I2C 128 x 64 Pixel Display kompatibel mit Arduino und Raspberry Pi oder 0,96 Zoll OLED SSD1306 Display I2C 128 x 64 Pixel
1	RFID Kit RC522 mit Reader, Chip und Card für Raspberry Pi und Co. (13,56MHz)
1	Mini MP3 Player DFPlayer Master Module
1	2 Stück 3 Watt 8 Ohm Mini-Lautsprecher
1	Battery Expansion Shield 18650 V3 inkl. USB Kabel
1	Breadboard Kit - 3 x 65Stk. Jumper Wire Kabel M2M und 3 x Mini Breadboard 400 Pins kompatibel mit Arduino und Raspberry Pi
1	Jumper Wire Kabel 3 x 40 STK. je 20 cm M2M/ F2M / F2F
2	Widerstand 1,0kΩ
3	Widerstand 2,2kΩ
1	Potentiometer 10kΩ linear zum Beispiel: Mehrgang rotary Potentiometer mit Schutzwiderstand 3590S 10K Ohm
optional	RFID Keycard Card 13,56MHz Schlüsselkarte Karte MF S50 (13,56 MHz) – 10x RFID Karte

Der Aufbau auf zwei Mini-Breadboards schaut durch die vielen Jumperkabel vogelwild aus. Immerhin werden vier Schnittstellen benutzt, SPI (RFID-Reader), I2C (OLED-Display), RS232 (DFPlayer mini) und ADC (Poti).

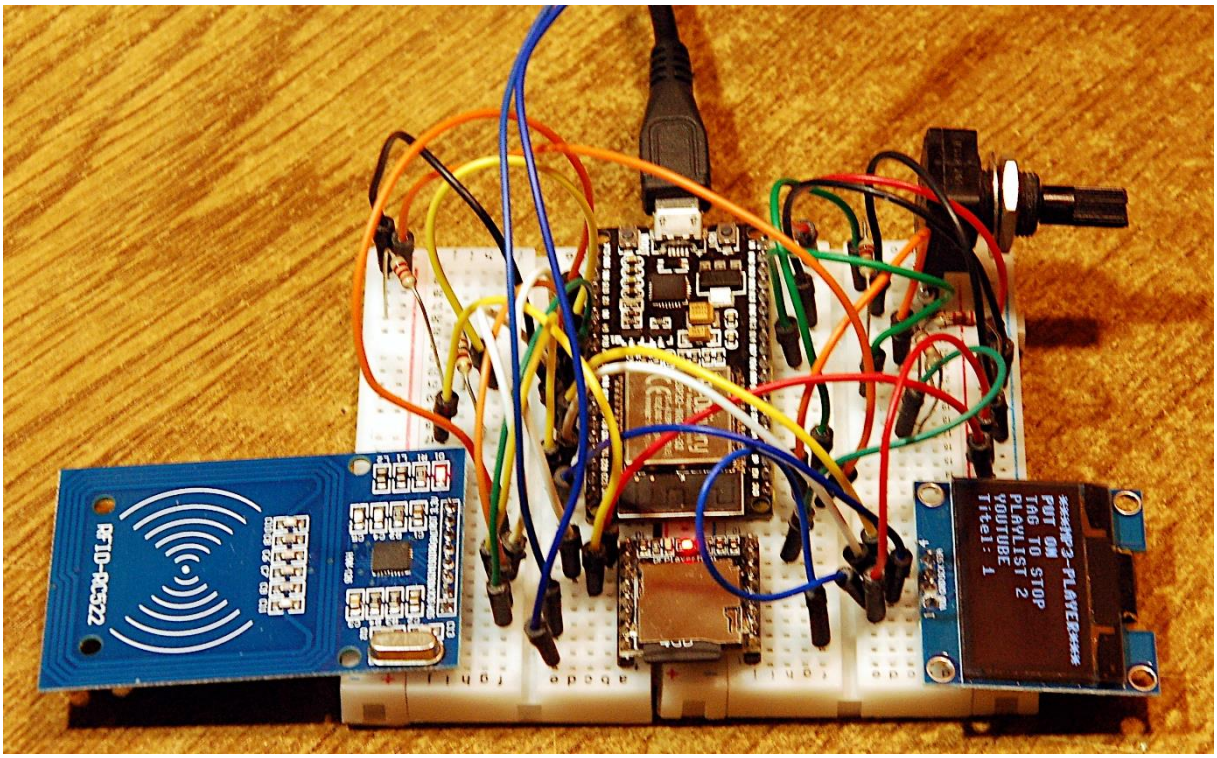

Abbildung 1: MP3-Player - Aufbau mit SH1106

Dabei gibt es einiges zu beachten. Der DFPlayer mini muss mit 5V versorgt werden. Daher liegen an seinem Pin TX auch Impulse mit einem Pegel von 5V an, ebenso am Busy-Pin. Weil der ESP32 aber nur 3,3V an seinen GPIOs verträgt, müssen die

Pegel durch Spannungsteiler reduziert werden. Ich habe 1kΩ und 2,2kΩ gewählt. Sie können auch andere Werte nehmen, die etwa im Verhältnis 1:2 stehen. Umgekehrt kommt der DFPlayer am RX-Eingang gut mit den 3,3V-Signalen vom ESP32 zurecht.

Wenn die Lautsprecher mit der schwarzen Leitung an GND gelegt werden, fließt im 5V-Kreis in Strom von ca. 450mA! Das liegt daran, dass die Anschlüsse SP1 und SP2 5V-Pegel führen. Schließen Sie daher die beiden Speaker nur an SP1 und SP2 an, nicht an GND. Auch wenn Sie nur einen Lautsprecher verwenden, gehören dessen Leitungen an SP1 und SP2 (in Abbildung 2 gelb). Andernfalls müsste man die Lautsprecher über einen Elko anschließen.

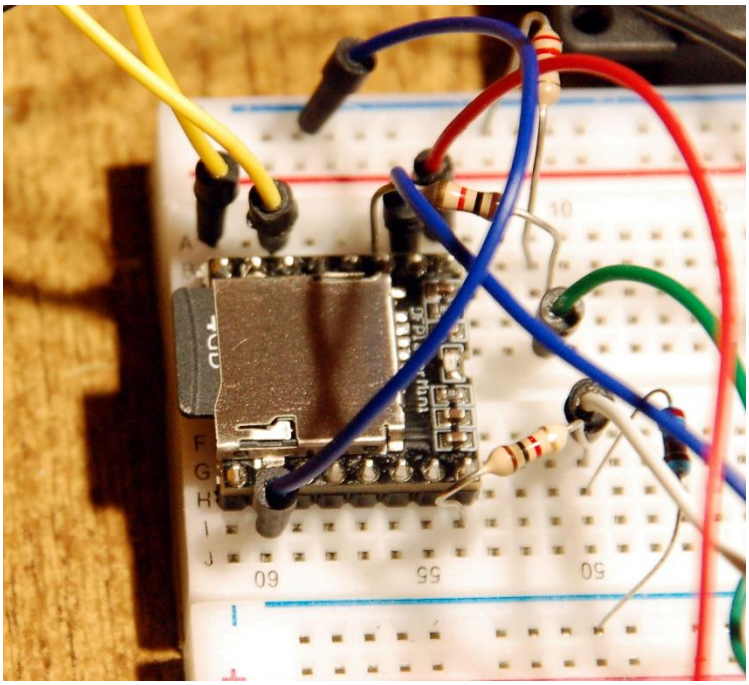

Abbildung 2: DFPlayer mini mit 4 GB SD-Karte

Der ADC-Eingang am ESP32 kann Spannungen bis 2,4V verarbeiten. Das Poti liegt aber an 3,3V. Daher reduziere ich die abgreifbare Spannung durch Vorschalten eines Widerstands von 2,2kΩ. Damit das Poti auf das Breadboard gesteckt werden kann, habe ich die Anschlüsse mit einer 5-poligen Stiftleiste versehen.

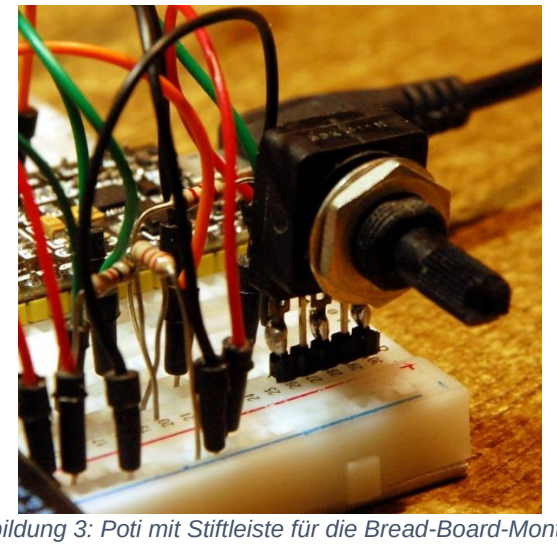

Abbildung 3: Poti mit Stiftleiste für die Bread-Board-Montage

Besonders aufpassen muss man bei den genannten Displays. Die Anschlüsse für SDA und SCK stimmen zwar überein, aber die Pins für +Vcc und GND beim 96"-Modul (Stiftleiste oben) sind beim 1,3"-Modul (Stiftleiste unten) gegeneinander vertauscht. In der Schaltung von Abbildung 4 habe ich ein 1,3"-Modul verwendet.

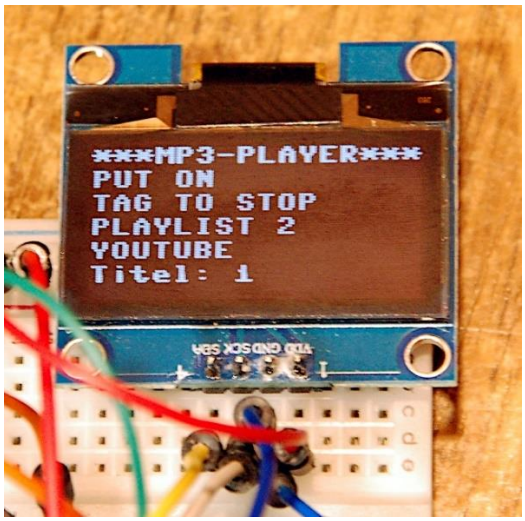

Abbildung 4: SH1106 in Betrieb

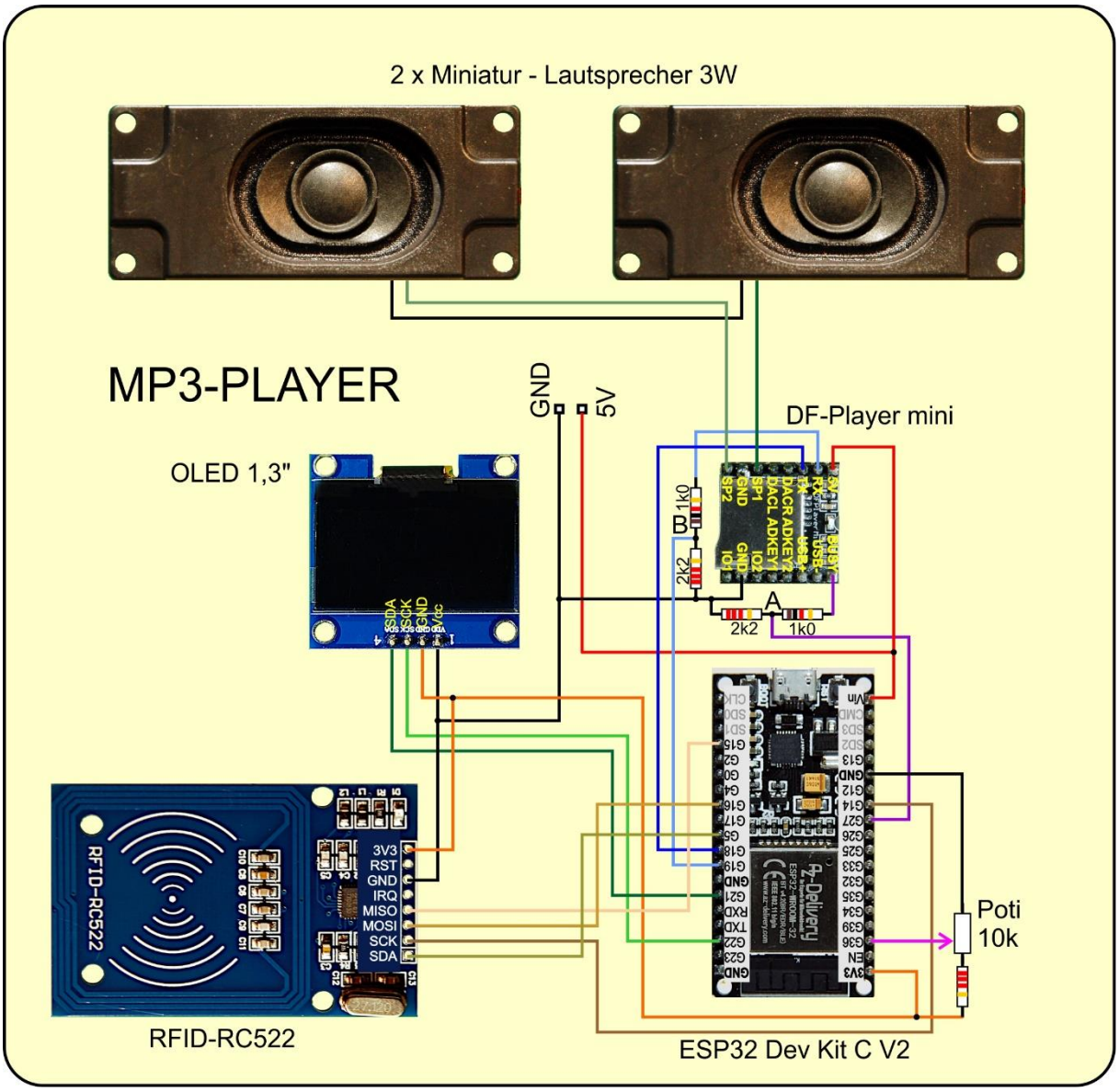

Abbildung 5: MP3-Player - Schaltung

Für den autonomen Betrieb habe ich einen Batteriehalter für einen LI-Ionen-Akku vom Typ 16850 vorgesehen. Er kann mit dem beiliegenden Kabel direkt mit dem ESP32 verbunden werden, der dann die Versorgung der anderen Module übernimmt.

Abbildung 6: Batteriehalter 18650

Für das Zusammenstöpseln der Schaltung kann der Verbindungsplan vielleicht hilfreich sein.

ESP32	RFID RC522	DF-Player	OLED
Vin		5V	
GND	GND	GND	GND
3V3	3,3V		Vcc
18		RX	
19		B	
27		A	
22			SCL
21			SDA
15	MISO		
16	MOSI		
14	SCK		
5	SDA		

Abbildung 7: Verbindungsplan

Den Chefposten übernimmt ein ESP32. Für dessen Einsatz bedarf es zweier Breadboards, die mit einer Stromschiene in der Mitte verbunden werden, damit man mit den Abständen der Pinreihen hinkommt und am Rand auch noch Kabel gesteckt werden können.

Der kleine Bruder des ESP32, der ESP8266, ist für unseren Zweck ungeeignet, er kann nämlich keine zweite, volle UART-Schnittstelle bieten, und die brauchen wir zur Ansteuerung des DFPlayers.

UART ist das Akronym für **Universal Asynchronous Receiver / Transmitter**. Über so ein Interface sprechen wir auch mit unserem Controller vom Terminal aus in Thonny. Der ESP32 /ESP8266 ist über das USB-Kabel mit dem PC verbunden und ein USB-RS232-TTL-Konverter auf dem Controllerboard gibt den Datenverkehr an die Schnittstelle UART0 des Controllers weiter. Der wiederum speist damit den Datenstrom **sys.stdin**. Aus diesem Datei-Objekt wird der input-Befehl bedient. **sys.stdout** sendet Ausgabedaten über UART0 an das Terminal von Thonny. Deshalb können wir UART0 nicht benutzen, um Befehle an den Mini-MP3-Player zu senden und Daten von ihm zu empfangen. Eine UART-Schnittstelle kann immer nur ein Gegenüber haben. Der ESP8266 hat zwar eine zweite UART-Schnittstelle, aber das ist eigentlich nur eine halbe, weil nur eine TXD-Leitung (Sendeleitung) und keine RXD-Leitung (Empfangsleitung) zur Verfügung steht. In Frage käme aber auch ein Raspberry Pi Pico (W), der zwei frei verfügbare UARTs im Angebot hat. Das Programm ist dasselbe, nur die Pinzuordnung würde eine andere.

Die Software

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft

Verwendete Firmware für einen ESP32:

Micropython Firmware Download
v1.19.1 (2022-06-18) .bin

Die MicroPython-Programme zum Projekt:
ssd1306.py Hardwaretreiber für das 0,96"-OLED-Display und
oled.py API für OLED-Display oder

sh1106.py Hardwaretreiber für das 1,3"-OLED-Display und
oled_SH1106.py API für OLED-Display

dfplayer.py Treibermodul für den DFPlayer mini
mfrc522.py Treibermodul für den RFID-Reader
mp3player.py Betriebsprogramm
timeout.py Softwaretimer nicht blockierend

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung (english version). Darin gibt es auch eine Beschreibung, wie die Micropython-Firmware (Stand 25.01.2024) auf den ESP-Chip gebrannt wird.

MicroPython ist eine Interpretersprache. Der Hauptunterschied zur Arduino-IDE, wo Sie stets und ausschließlich ganze Programme flashen, ist der, dass Sie die MicroPython-Firmware nur einmal zu Beginn auf den ESP32 flashen müssen, damit der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang hier beschrieben.

Sobald die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm kompilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Makro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

Autostart

Soll das Programm autonom mit dem Einschalten des Controllers starten, kopieren Sie den Programmtext in eine neu angelegte Blankodatei. Speichern Sie diese Datei unter boot.py im Workspace ab und laden Sie sie zum ESP-Chip hoch. Beim nächsten Reset oder Einschalten startet das Programm automatisch.

Programme testen

Manuell werden Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5 gestartet. Das geht schneller als der Mausklick auf den Startbutton, oder über das Menü **Run**. Lediglich die im Programm verwendeten Module müssen sich im Flash des ESP32 befinden.

Zwischendurch doch mal wieder Arduino-IDE?

Sollten Sie den Controller später wieder zusammen mit der Arduino-IDE verwenden wollen, flashen Sie das Programm einfach in gewohnter Weise. Allerdings hat der ESP32/ESP8266 dann vergessen, dass er jemals MicroPython gesprochen hat. Umgekehrt kann jeder Espressif-Chip, der ein kompiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer so, wie hier beschrieben.

Das Programm

Ein Überblick

Die Auswahl der Playlists wird in diesem Projekt durch RFID-Tags vorgenommen. Die verwendeten Tags können neben der Karten-ID bis zu 1kByte Daten speichern. Es wäre also denkbar, die Titel der Songs, die mit dem Einlesen der ID ausgewählt werden, auf der Karte zu speichern. Das geht aber eindeutig über den Umfang dieses Beitrags hinaus und könnte vielleicht Gegenstand einer anderen Blogfolge werden. Anhand der ID, einer 8-stelligen Hexadezimalzahl, kann der ESP32 den DFPlayer mini anweisen, die MP3-Dateien aus einem Ordner auf der SD-Card abzuspielen. Ordnername und -Nummer werden im Display zusammen mit der Titelnummer angezeigt. Im Display werden auch Benutzeranweisungen dargestellt.

Der RFID-Reader

Durch den SPI-Bus ist die Verdrahtung zusätzlich aufwendiger als alleine beim I2C-Bus mit seinen 2 Leitungen. SPI-Bus-Geräte haben keine Hardware-Geräteadresse, dafür haben sie einen Chip-select-Anschluss (CS), der auf LOW liegen muss, wenn das Gerät angesprochen werden soll. Auch der Datentransfer läuft etwas anders, es wird stets gleichzeitig gesendet und empfangen. Das nähere Procedere zu klären ist hier nicht notwendig, denn die Klasse **MFRC522** erledigt das für uns. Dem Konstruktor werden wir nur die Anschlussbelegungen und die Übertragungsgeschwindigkeit mitteilen. Der Transfer arbeitet mit flotten 3,2MHz. Zum Vergleich, I2C arbeitet auf bis zu 400kHz.

Unsere eigene Funktion **readUID**() liest die eindeutige Kennung einer Karte aus und gibt sie zurück und zwar als Dezimalzahl und als hexadezimalen Ziffernstring. Die Karten werden über das OLED-Display angefordert. Damit die Funktion nicht den gesamten Ablauf blockiert, sorgt ein Timeout für einen geordneten Rückzug. In diesem Fall wird statt der Karten-ID der Wert None zurückgegeben.

Damit die Playlist-Karten ins Spiel kommen, brauchen wir eine Masterkarte. Dazu nehmen wir eine beliebige Karte oder einen Chip aus dem Stapel, lesen die ID aus und belegen damit gleich am Programmbeginn die Variable mit dem Dezimalwert, hier:

```
MasterID=4217116188.
```

Beim ersten Start stellt der ESP32 fest, dass noch keine Datei mit den Playlist-Karten -Daten besteht und verlangt die Masterkarte. Nachdem diese erkannt wurde, wird eine Playlist-Karte angefordert. Nach dem Auslesen der ID wird diese in die Datei geschrieben und erneut die Masterkarte verlangt. Das Einlesen wird bis zur letzten Playlist-Karte fortgesetzt. Wird nach der Anforderung der Masterkarte 10 Sekunden lang keine Playlist-Karte angeboten, wird der Erfassungsvorgang beendet, und das Hauptprogramm startet. Um komplett von vorne zu beginnen, können wir die Datei **slavecards.txt** mit den Playlist-Karten-IDs über die Thonny-Console löschen. Die Datei kann natürlich auch im Editorfenster von Thonny bearbeitet werden.

DFPlayer mini

Die Kommunikation mit dem DF-Player geschieht stets in der gleichen Weise. Es werden Blöcke von 10 Bytes gesendet und empfangen. Der Aufbau eines solchen Blocks sieht wie folgt aus. Die Methode **sendCommand**() der Klasse **DFPlayer** kümmert sich um die korrekte Übermittlung.

Startbyte 0x7E
Versionsnummer 0xFF
Länge der Payload 0x06 (von Versionsnummer bis Parameter 2 inkl.)
Commandbyte
Feedback 0x00 (nein) oder 0x01 (ja)
Parameter high
Parameter low
Checksum high
Checksum low
Endebyte 0xEF

Die Befehlscodes kann man im Datenblatt finden. Dieses habe ich benutzt, um die wesentlichsten Codes in Methoden des Moduls **dfplayer.py** umzusetzen. Wir brauchen für das aktuelle Projekt nur ein paar davon. Werfen Sie aber ruhig mal einen Blick in die Datei **pfplayer**.py (Rev 1.3 vom 24.01.2024)

Die Bearbeitung von Sounddateien ist mit dem Freewaretool Audacity sehr leicht möglich Auch das Umcodieren von Sounddateiformaten nach mp3 ist damit ein Kinderspiel. Einen größeren Aufwand bedeutet es, die Dateinamen so zu ändern, dass diese den Vorgaben von DFPlayer mini entsprechen. Die Dateien befinden sich in Ordnern, die zweistellig benannt sind. Das sind quasi die Playlists.

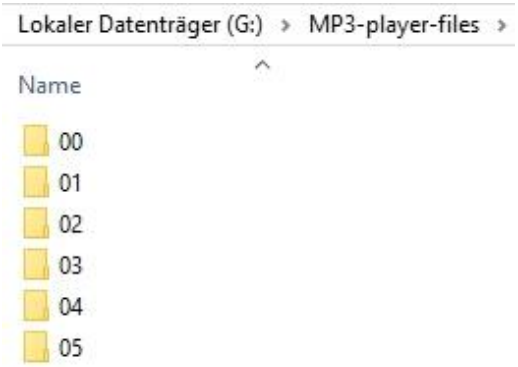

Abbildung 8: Ordneransicht der SD-Karte

Die Dateinamen können ihre alte Bezeichnung behalten, müssen aber mit einem Prefix versehen werden, das aus einer dreistelligen Dezimalzahl besteht.

Abbildung 9: Dateinamen mit Prefix

Für ein paar Dateien mag die manuelle Umbenennung noch angehen, bei mehr als zehn wird das sehr lästig. Deshalb habe ich das mit einem weiteren kostenlosen Tool gemacht, Advanced Renamer. Die Installationsdatei laden Sie vom Chip-Server herunter.

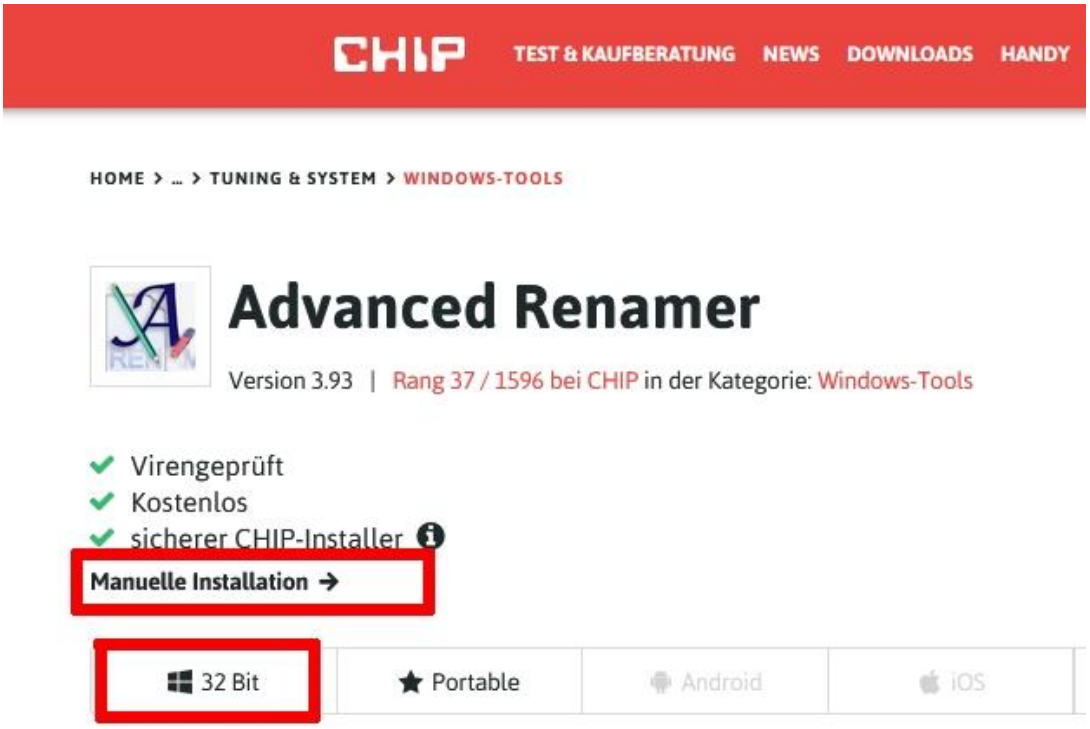

Abbildung 10: Advanced Renamer herunterladen

Zur Installation folgen Sie einfach dem Assistenten. Die für einen Ordner auf der SD-Card ausgewählten Dateien sammeln Sie am besten in einem Arbeits-Verzeichnis oder gleich auf der SD-Card. Markieren Sie dort alle Dateien im Ordner und ziehen sie das Paket auf das **Renamer**-Fenster. Als **Batch-Methode** wählen Sie **Hinzufügen** und stellen dann im Filterfenster **1: Hinzufügen** die Eigenschaften wie in Abbildung 11 ein.

Abbildung 11: Filtereinstellung in Advaced Renamer

Jede Änderung in diesem Fenster wirkt sich sofort auf die Vorschau aus. Zur Übernahme klicken Sie rechts oben auf **Batch starten**. Das war's.

Abbildung 12: Quellnamen und Ergebnis

MP3-Player

Nach den ganzen Vorbereitungen wird es Zeit für die Codierung des Projekts. Legen wir also los. Wie immer beginnt das Programm mit dem Importgeschäft. Das Beste wird es sein, wenn Sie das Programm mp3player.py schon mal in Thonny im Editorfenster öffnen, dann können Sie dort im Überblick mitlesen, während wir hier die Arbeitsweise durchgehen.

```
# mp3player.py
# workes with RC522 @ 13,56MHz
from dfplayer import DFPlayer
import mfrc522
from machine import Pin, SoftI2C, SPI, ADC, reset
# from oled import OLED # Vorsicht: GND Vcc SCK SDA
from oled_SH1106 import OLED # Vorsicht: Vdd GND SCK SDA
from time import sleep
from sys import exit
from timeout import *
```

Von **dfplayer.py** importieren wir die Klasse DFPlayer, während von **mfrc522.py** alles importiert wird. Diese beiden Dateien müssen in den Flash des ESP32 hochgeschaufelt werden. Im Filemanager von Thonny beide markieren, Rechtsklick, **Upload to /**.

Aus dem Modul **machine** kommen die Klassen für die Bedienung der Schnittstellen. Je nachdem, welches Display-Modul eingesetzt wird, muss eine der beiden folgenden Zeilen aktiviert beziehungsweise auskommentiert werden.

Für kurze, passive Pausen holen wir **sleep**() aus dem Modul time. Einen sauberen Programmausstieg sichern wir uns mit der Funktion **exit**() aus **sys**. Aus dem Modul **timeout** holen wir alles (*****) in unseren Scope, was für nichtblockierende Softwaretimer gebraucht wird. Durch den Stern entfällt beim Aufruf einer Methode aus diesem Modul der Vorsatz des Modulnamens.

Ohne Stern zum Beispiel:
Timeout.TimeOutMs(1000)

Mit Stern:
TimeOutMs(1000)

In entsprechenden Paketen instanziieren wir als Nächstes unsere Objekte. Wir starten mit dem ADC-Objekt, das wir für das Einlesen der Spannung vom Poti brauchen. Wir werden damit die Lautstärke des DFPlayer mini kontrollieren.

```
vol=ADC(Pin(36))
vol.atten(vol.ATTN_11DB)
vol.width(vol.WIDTH_10BIT)
```

Der Schleifkontakt des Potis ist an GPIO36 angeschlossen. Wird stellen den Abschwächer auf 11 db und erreichen damit eine Abtastung bis 2,4V. Die Auflösung richten wir auf 10-Bit ein.

```
RX_Pin = 16
TX_Pin = 17
busyPin = 27
df=DFPlayer(busyPinNbr=busyPin,
            txd=TX_Pin,
            rxd=RX_Pin,
            volume=15)
sleep(2) # wait for player initializing
numOfTitles=df.getNumberOfFiles()
print("Titel gesamt:",numOfTitles)
```

Die Klasse DFPlayer richtet sich den UART1 des ESP32 selbst ein. Der Konstruktor braucht dazu nur die Nummern der GPIOs an denen **busy**-Pin, **rx**-Pin und **tx**-Pin angeschlossen sind. UART1 liegt standardmäßig an Pins, die für die Unterhaltung des ESP32 mit seinem Onboard-Flash reserviert sind. Durch die Möglichkeit, die Anschlüsse auf beliebige, geeignete GPIOs umzulegen, können wir die Schnittstelle trotzdem nutzen. Wir legen RX am ESP32 auf GPIO16 und TX auf GPIO17. Nach zwei Sekunden testet der ESP32 die Verbindung, indem er die Gesamtanzahl von Dateien auf der SD-Card abfragt.

```
spi=SPI(2,baudrate=3200000)
#                           cs=sda
rdr = mfrc522.MFRC522(spi, cs=5, baudrate=3200000)
```

Die SPI2-Hardware-Schnittstelle belegt die GPIOs 18 (SCK), 23 (MOSI) und 19 (MISO), ferner 5 (CS). Der Chip-Select-Anschluss auf dem RFID-Board ist mit SDA benannt. Die Baudrate legen wir auf 3200000Hz fest.

```
i2c=SoftI2C(scl=Pin(22),sda=Pin(21))
d=OLED(i2c)
```

Für das Display instanziieren wir ein I2C-Objekt, das wir dem Konstruktor des OLED-Objekts mit auf den Weg geben.

Die Master-Karte ist das RFID-Tag, mit welchem dem System die Playlist-Tags bekanntgemacht werden. Den Zahlenwert ermitteln wir, sobald das Programm soweit fertiggestellt ist und tragen ihn dann an dieser Stelle ein.

```
MasterID=4217116188  #  0XFB5C161C
```

Die Liste **listen** enthält die Namen der Playlists.

```
listen=[
        "COUNTRY",
        "BLACKMOORSNIGHT",
        "YOUTUBE",
        "SCHLAGER",
        "HITS",
        "VON CD"
        "OLDIES"
       ]
```

Es folgen die Deklarationen einiger Funktionen. **readUID**() liest die Identifikationsnummer des Tags aus. Wir übergeben das OLED-Display-Objekt, einen String für den Kartentyp und eine Zeitdauer in Millisekunden, nach deren Ablauf die Funktion in jedem Fall verlassen wird. Die Variable **song** wird global deklariert, damit der in der Funktion zugewiesene Wert im Hauptprogramm verfügbar wird. Auf alle Ausgaben, sowohl im Display als auch in REPL, gehe ich nicht weiter ein, sie dienen teils der Information des Benutzers und der Fehlersuche während der Entwicklung und erklären sich durch ausgegebenen Texte von selbst.

Der nichtblockierende Softwaretimer **readTimeOut** wird durch die Methode **TimeOutMs**() eingestellt. In **TimeOutMs**() verbirgt sich die Funktion **compare**(), eine sogenannte Closure. **TimeOutMs**() gibt eine Referenz auf **compare**() zurück, die wir dem Bezeichner **readTimeOut** zuweisen. Anders als bei **sleep**() können, während der Timer tickt, andere Anweisungen ausgeführt werden. Die while-Schleife wird also so lange durchlaufen, bis die Funktion **readTimeOut**(), alias **compare**(), **True** zurückgibt. Die Hintergründe zu diesem Vorgehen können Sie im Dokument Closures an Decorators nachlesen.

```
def readUID(display,kartentyp,timeout):
    global song
    display.clearFT(0,1,15,2,False)
    display.writeAt("PUT ON",0,1,False)
    display.writeAt(kartentyp,0,2)
    readTimeOut=TimeOutMs(timeout)
    while not readTimeOut():
        (stat, tag_type) = rdr.request(rdr.REQIDL)
        if stat == rdr.OK:
            (stat, raw_uid) = rdr.anticoll()
            if stat == rdr.OK:
                display.clearFT(0,3,15,3)
                display.writeAt("Card OK",0,3)
                userID=0
                for i in range(4):
                    userID=(userID<<8) | raw_uid[i]
                userIDS="{:#X}".format(userID)
                display.writeAt(userIDS+"     ",0,4)
                sleep(2)
                df.reset()
                song=0
                return userID,userIDS
    return None
```

Mit **rdr.request**() stoßen wir einen Einlese-Prozess an. War die Anfrage erfolgreich, holen wir im nächsten Schritt die ID ab. War auch dieser Schritt erfolgreich, machen wir uns an die Auswertung. Vom MFRC522 bekommen wir ein **bytes**-Objekt zurück, das wir in vier Schritten in eine 32-Bit-Zahl umwandeln. Das bisherige Ergebnis in **userID** wird um jeweils 8 Bit-Positionen nach links geschoben und das nächste Element der bytes-Folge dazu oderiert. Das Gesamtergebnis verwandeln wir anschließend in einen Hexadezimalstring, der in der Regel 8-stellig sein wird.

Zwei Sekunden für das Ablesen des Displays, dann setzen wir den DFPlayer mini zurück und den Songzähler auf 0, denn es soll mit dem Einlesen der ID ja auch eine

neue Playlist gestartet werden. Statt der ID in Zahlen- und String-Form wird None zurückgegeben, falls der Leseversuch nicht erfolgreich war oder einfach die Zeit abgelaufen ist.

Wenn Sie das Programm bis hierher eingegeben haben, können Sie es bereits starten, um die ID der Master-Karte auszulesen. Alternativ können Sie das Programm **mp3player.py** herunterladen und in Zeile 73 folgende Anweisung eintragen. Danach speichern und starten Sie das Programm im Editorfenster.

```
exit()
```

Nach dem der REPL-Prompt wieder erschienen ist, geben Sie folgende Anweisung ein. Halten Sie jetzt Ihre Master-Karte an den Leser. Den ausgegebenen Zahlenwert tragen Sie oben als **MasterID** ein.

>>> readUID(d,"TEST",6000)
0XFB5C161C
*(**4217116188**, '0XFB5C161C')*

Diese Master-Karte benutzt die nächste Funktion **addUID**(), die für das Registrieren der Playlist-Karten zuständig ist. Sie wird vom Hauptprogramm beim ersten Start aufgerufen, wenn noch keine Datei mit dem Namen **slavecards.txt** im Dateisystem des ESP32 existiert. Aber auch von Hand kann **addUID**() aufgerufen werden, um die Datei anzulegen und/oder weitere Playlist-Tags zu registrieren.

Als Erstes fordern wir die Master-Karte an und lesen die ID ein. **m** erhält das Tupel aus dem dezimalen Wert und dem Hex-String, wenn die Aktion erfolgreich war, sonst **None**. Das Tupel dröseln wir durch Entpacken in die Zahl und den String auf und gleichen auf die Master-ID ab. Es verbleiben drei Sekunden zum Auflegen einer neuen Playlist-Karte, sobald die Leseanweisung erfolgt ist.

```
def addUID(display):
    display.clearAll()
    m=readUID(display,"Master",3000)
    print("Master",m)
    if m is not None:
        mid,mids= m
        if mid==MasterID:
            print("Master OK")
            u=readUID(display,"Slavecard",3000)
            if u is not None:
                uid,uids=u
                if uid is not None and uid != MasterID:
                    with open("slavecards.txt","a") as f:
                        f.write("{}\n".format(uids))
                    display.writeAt("New slave written",0,4)
                    display.writeAt("{}".format(uids),0,5)
                    sleep(5)
                    return True
            else:
                display.writeAt("ERROR!!!",0,3)
                display.writeAt("Card not added!",0,4)
```

```
                return False
        else:
            display.writeAt("ERROR!!!",0,3)
            display.writeAt("Not mastercard",0,4)
            sleep(3)
    return False
```

Wie vorher **m**, so enthält nun **u** das ID-Tupel, wenn der Lesevorgang erfolgreich war. Die Karte wird registriert, wenn **uid** nicht **None** ist und die Karte nicht die Masterkarte war. Mit der **with**-Anweisung wird ein Datei-Objekt angelegt und zum Anhängen von Textzeilen unter dem Bezeichner **f** geöffnet. Wir schreiben auch gleich den Hex-String der eingelesenen Karte mit einem angehängten Zeilenende-Zeichen "\n" = 0x0A. Mit dem Verlassen des **with**-Blocks wird die Datei automatisch geschlossen. Wir müssen uns also nicht selbst um diese notwendige Maßnahme kümmern. Nach fünf Sekunden Lesezeit für die Displaymeldung gibt die Funktion **True** zurück. Zwei mögliche Fehler müssen behandelt werden, ein Lesefehler vom RC522 und eine falsche Masterkarte.

Beim Programmstart wird **readTags()** aufgerufen und nach einer Datei **slavecards.txt** im Root-Verzeichnis des ESP32-Dateisystems gefahndet. Die leere Liste **tags** soll die Hex-Strings der Karten aufnehmen. Kann die **with**-Anweisung die Datei nicht öffnen, weil sie noch nicht vorhanden ist, bleibt **tags** leer, und der else-Teil des if-Konstrukts gibt None zurück. Existiert die Datei, dann wird Zeile für Zeile eingelesen, das Zeilenende-Zeichen entfernt und der String an die Liste angehängt, bis die Schleife das Dateiende erkennt. In diesem Fall ist die Liste nicht leer und wird zurückgegeben.

```
def readTags():
    tags=[]
    with open("slavecards.txt","r") as f:
        for line in f:
            tags.append(line.strip("\n"))
    if tags:
        return tags
    else:
        return None
```

Jeder Ordner auf der SD-Karte kann unterschiedlich viele Musiktitel enthalten. Wir müssen die Anzahl kennen, um sie in einer **for**-Schleife abspielen zu können. **getNumberOfTitels**() ermittelt die Anzahl an Titeln in allen Ordnern und legt sie in der Liste **fileCount** ab. Wir starten mit der leeren Liste und lassen uns erst einmal die Anzahl von Ordnern flüstern. Um jeden Ordner anzusprechen spielen wir in der for-Schleife den ersten Titel darin kurz an, und damit man nicht durch Knack- und sonstige Geräusche irritiert wird, schalten wir zuvor die Lautstärke auf 0.

```
def getNumberOfTitels():
    fileCount=[]
    nof=df.getNumberOfFolders()
    df.volume(0)
    sleep(0.2)
    for t in range(nof):
        df.play(t,0)
        sleep(0.2)
        df.stop()
        sleep(0.2)
        n=df.getNumberOfFilesInFolder()
        d.writeAt("Track:{}".format(t),0,4,False)
        d.writeAt("Songs:{}".format(n),0,5)
        fileCount.append(n)
    df.reset()
    volume=int(vol.read()*100/1024)
    df.volume(volume)
    sleep(0.2)
    return fileCount
```

In der for-Schleife hängen wir die Titelanzahl an die Liste hinten dran. Nach dem Rücksetzen des DFPlayers restaurieren wir die Lautstärke auf den Wert, den das Poti vorgibt. Die Liste der Titelzahlen wird zurückgegeben.

Als letzte Funktion deklarieren wir **player**(). Während der Laufzeit der Wiedergabeschleife findet ein stetes Einlesen von Playlist-Karten statt. Eine neue Karte bricht die gerade abgespielte Playlist ab. Die Variable **acard** gibt in diesem Fall den leeren String an das Hauptprogramm zurück. Damit das aus der Funktion heraus möglich ist, deklarieren wir die Variable global. Damit der Player richtig startet, stoppen wir ihn erst einmal. Dann geht es in die for-Schleife. Der Player verhält sich hier störrisch und überspringt den ersten Titel, deswegen lasse ich die Schleife mit dem Laufindex **s** bei -1 starten. **isPlaying**() sollte jetzt **False** zurückgeben, was wir gleich überprüfen und daraufhin den Titel **s** im Ordner **t** abspielen lassen.

```
def player(d,t,nof):
    global acard
    df.stop()
    for s in range(-1,nof):
        if not df.isPlaying():
            df.play(t,s)
            d.clearFT(0,5,15,5,False)
            d.writeAt("Titel: {}".format(s),0,5)
            while df.isPlaying():
                volume=int(vol.read()*100/1024)
                df.volume(volume)
                u=readUID(d,"TAG TO STOP",100)
                if u is not None:
                    d.clearFT(0,3,15)
                    df.stop()
                    acard=""
                    return
    d.clearFT(0,3,15)
    df.stop()
```

Während der Titel läuft, gibt es ständig einiges zu tun. Das Poti muss abgefragt und der Wert an den DFPlayer weitergegeben werden. Dann geben wir einen Leseauftrag an den RC522 mit der Botschaft für einen potenziellen Abbruch. Kommt nach 100 Millisekunden = 0,1 Sekunden **None** zurück, geht es in die nächste Runde der while-Schleife. Wurde aber eine neue Karte registriert, stoppen wir den DFPlayer und bereiten mit **acard** = "" das Abspielen einer neuen Playlist vor. Falls die Playlist komplett abgespielt wurde, löschen wir den unteren Teil der Anzeige und geben noch einmal einen Stoppbefehl an den DFPlayer. Mitunter ignoriert dieser in schändlicher Weise alle möglichen Befehle. Das hat bei der Entwicklung des Programms verschiedene enttäuschende Momente ergeben, die stets das Eröffnen neuer Lösungsansätze erforderlich machten.

Für das Hauptprogramm müssen einige Variablen initialisiert werden.

```
cards=[] # Liste der Karten-IDs leeren
acard="" # aktuelle Karten-ID
ncard="" # neue Karten-ID
song=0   # Stets mit Song 0 starten
```

Die erste Handlung ist jetzt das Einlesen der Karten-IDs, falls es denn die Datei **slavecards.txt** gibt. Beim ersten Start des Programms ist hier Fehlanzeige. Deshalb sichern wir den Aufruf von **readTags**() mit **try** und **except** ab. Im except-Block legen wir die Datei an. Wir zaubern also einen nichtblockierenden Softwaretimer **allRead**() mittels **timeOutMs**() aus der Retorte. Während der Timer abläuft, rufen wir zyklisch **addUID**() auf. Die Arbeitsweise dieser Funktion haben wir ja oben schon besprochen. Wenn die Routine **True** zurückgibt, wurde eine Karte registriert, und der Timer wird neu gestartet. Gibt es keine weitere Karte einzulesen, warten wir einfach den Ablauf des Timers ab und holen in einem zweiten Versuch die IDs in die Liste **cards**.

```
try:
    cards=readTags()
except OSError as e:
    allRead=TimeOutMs(10000)
    while not allRead():
        if addUID(d):
            allRead=TimeOutMs(10000)
    d.clearFT(0,3,15,4,False)
    d.writeAt(" ALL CARDS READ",0,3)
    cards=readTags()
```

Wenn die Liste nicht leer ist, geht es in die Hauptschleife, die permanent durchlaufen wird, auch während ein Titel abgespielt wird.

```
if cards:
    d.writeAt("***MP3-PLAYER***",0,0,False)
    d.writeAt("PUT ON MP3-TAG",0,1,False)
    d.writeAt("Hole die Anzahl",0,2,False)
    d.writeAt("Songs/Folder",0,3)
```

```
    filesInFolder=getNumberOfTitels()
    d.clearFT(0,2,15)
    df.stop()
    while 1:
        uid=readUID(d,"PLAYLIST-TAG",1000)
        if uid is not None:
            ncard=uid[1]
        volume=int(vol.read()*100/1024)
        df.volume(volume)
```

Wir versuchen, eine Karte einzulesen. Hat das geklappt, weisen wir den Hex-String der Variablen **ncard** zu und fragen als Nächstes das Poti ab.

Der Abspielvorgang einer neuen Playlist wird gestartet, wenn der Inhalt von **acard** nicht mit dem von **ncard** übereinstimmt. In diesem Fall wird **acard** schon einmal auf den neuesten Stand gebracht.

```
        if acard != ncard:
            acard=ncard
```

Dann prüfen wir, ob die ID in der Liste **cards** enthalten ist. Wenn ja, dann ermitteln wir den Index des Eintrags und haben somit die Nummer des Ordners auf der SD-Card gefunden. Die Organisation zum Abspielen der Playlist übergeben wir der Funktion **player**(). Sie bekommt die Referenz auf das OLED-Objekt, die Ordnernummer und die Anzahl Titel mit auf den Weg.

```
            if acard in cards:
                d.clearFT(0,2,15,5, False)
                t=cards.index(acard) # Ordnernummer
                d.writeAt("PLAYLIST {}".format(t),0,3,False)
                d.writeAt(listen[t],0,4)
                player(d,t,filesInFolder[t])
```

Wurde die Karten-ID nicht in **cards** gefunden, kriegen wir eine Fehlermeldung bevor es in die nächste Runde der Mainloop geht.

Für einen autonomen Betrieb, ohne Anschluss an den PC, muss das Programm unter dem Namen **main.py** in den Flash des ESP32 hochgeladen werden. Der Vorgang ist oben beschrieben. Alternativ können Sie auch in Thonny **Save as** aus dem **File**-Menü aufrufen. Als Zielort wählen sie dann **MicroPython device**. Wenn Sie jetzt den ESP32 am Batterie-Board anstecken, startet das Programm ohne PC.

Damit sind wir mit der Besprechung am Ende. Für die Ausstattung des Players gibt es aber noch weitere Möglichkeiten. Wer es lauter haben möchte, setzt einen Verstärker an die Stereo-Ausgänge des DFPlayer mini. Mit ein paar Tasten könnte man in den Titeln blättern und mit einer WLAN-verbindung kann man die volle Kontrolle mit einer Android-App übernehmen, die leicht mit dem MIT App-Inventor 2 erstellt werden kann.

Als Einsatzzweck für den Player kommen alle Situationen in Frage, in denen man durch die Karten, die ausgehändigt werden, das Abspielprogramm einschränken oder steuern möchte. Wenn die Karten in eine Schautafel eingebaut werden, kann durch Auflegen des Players ein entsprechender Informationstext abgespielt werden. Oder lassen Sie den Teddybären verschiedene Gute-Nacht-Geschichten erzählen…

Viel Vergnügen beim Basteln, Programmieren und Umsetzen vieler weiterer Ideen.