Vorwort

Alle Schaltungen, Programme und Bauteile wurden vorher von uns geprüft. Trotzdem können wir leider Fehler im Buch, den Schaltungen oder Programmen nicht ausschließen. Bei Fragen oder Problemen gibt es unten die Kontaktmöglichkeiten. Mehr Informationen findest du auch auf der Internetadresse der Seitenzahl.

LEDs

LEDs sollten nicht aus kurzer Distanz direkt angesehen werden. Ein direkter Blick kann zu Schäden an den Netzhäuten führen. Dies kann gefährlich sein, auch wenn die Verletzungen nicht sofort klar erkennbar sind. Die LEDs dürfen nur wie in den Anleitungen beschrieben verwendet werden, höhere Ströme oder Stromspannungen sind zu vermeiden.

Dieses Produkt entspricht den geltenden Europäischen Richtlinien und trägt ein CE-Kennzeichen. Der richtige Gebrauch ist in dem beiliegenden Buch erklärt.

CE

Baue Schaltungen immer wie beschrieben auf, achte auch auf die verschiedenen GPIO Pins vom Raspberry Pi.

Herausgeber

Jugend Programmiert
Coding World UG (haftungsbeschränkt)
Liese-Meitner-Str. 2, 24941 Flensburg
www.codingworld.io
Support & Feedback: support@cw42.de

Moin Moin und Willkommen

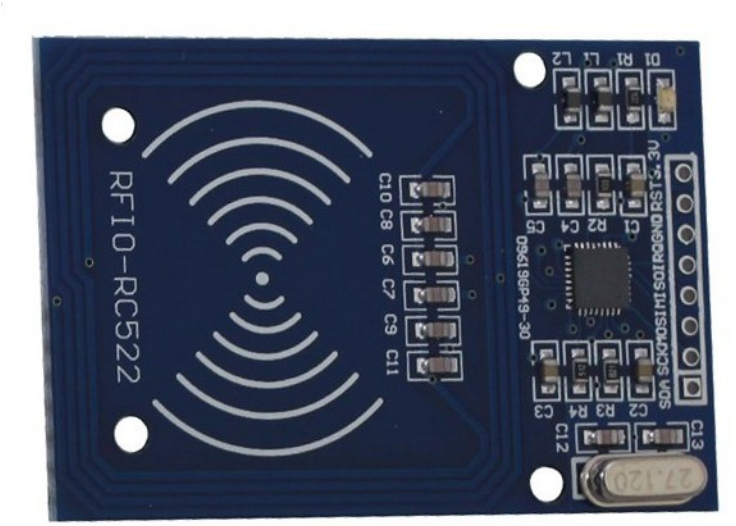
In diesem Kit werden uns mit dem ganzen Themenkomplex RFID beschäftigen. Du wirst lernen, was RFID überhaupt ist und wie du mithilfe deines Pis und einem RFID-Reader Karten auslesen und beschreiben kannst. Natürlich haben wir das ganze praxisorientiert und spaßig verpackt. Damit dein Projekt Tagebuch nicht leer aus geht, haben wir uns auch noch mehrere Anwendungsfälle ausgedacht, bei denen wir mit RFID großartige Dinge machen können, aber das wirst du schon schnell genug erfahren.

Damit du das alles machen kannst, ist folgendes im Kit enthalten:

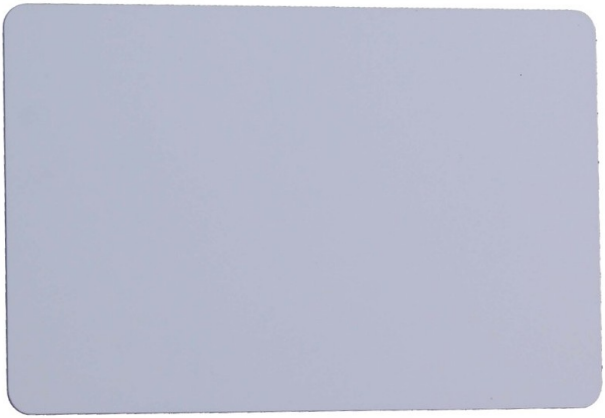
- RFID Reader
- RFID Karten (sehen aus wie weiße EC-Karten)
- RFID Schlüsselanhänger

RFID Grundlagen

Bevor wir direkt mit dem Pi und RFID anfangen, brauchen wir erstmal ein paar Grundlagen für die Funktionsweise. RFID ist kurz für **r**adio-**f**requency **id**entification oder auf Deutsch: Identifizierung mit elektromagnetischen Wellen. Mit dieser Technik können Daten von einem Transponder mithilfe eines Schreib-/Lesegerät (wir werden die Abkürzung RFID-Reader benutzen) gelesen werden.



Wie schon gesagt, gehört zu einem RFID System immer der RFID-Reader und ein Transponder. Ein Transponder kann dabei in vielen verschiedenen Formen und Größen kommen. Wir benutzen in diesem Fall RFID-Karte, RFID-Anhänger und NFC-Sticker (dazu später mehr). Alle davon sind Transponder. Die Transponder, die wir benutzen, sind alle gleich aufgebaut. Sie lassen sich in zwei verschiedene Komponenten aufteilen, einmal die Antenne, diese befindet sich an den Außenseiten und dem Speicher und der Logik in der Form eines Chips in der Mitte des Transponders



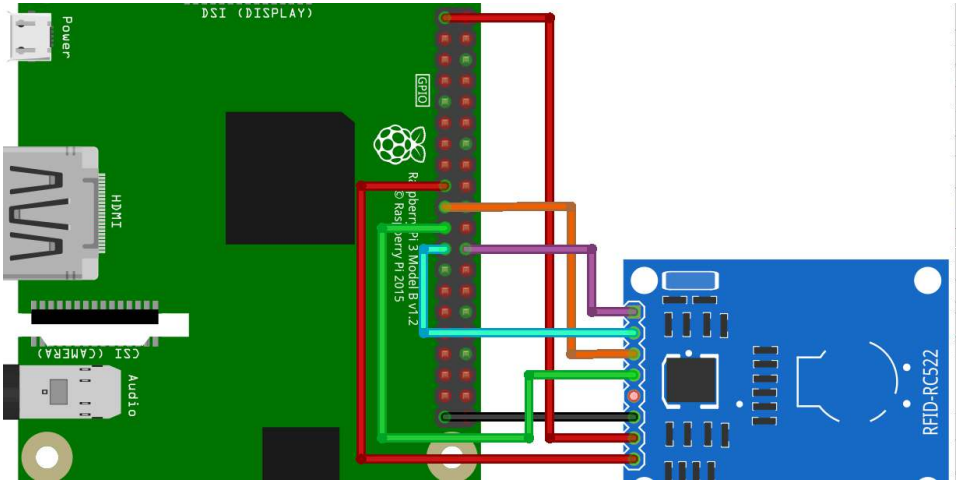
Die Transponder, die wir benutzen, haben keine eigene Stromversorgung, denn diese kommt durch den RFID-Reader. Wenn dieser angeschlossen ist, wird ein hochfrequentes elektromagnetisches Feld erzeugt. Und sobald sich der Transponder in diesem Feld befindet, wird er darüber mit Strom versorgt. Die Energie wird dabei durch die Antennen aufgenommen und dann an den Chip weitergegeben. Wenn die Stromversorgung sichergestellt ist, startet die Kommunikation zwischen dem RFID-Reader und dem Transponder. Die hier verwendeten Transponder können beschrieben oder gelesen werden. Der Transponder erzeugt dabei kein eigenes Feld oder Wellen, sondern manipuliert nur das vorhandene Feld mithilfe der Antennen. Dieses kann dann durch den RFID-Reader interpretiert und ausgewertet werden. Der Vorteil ist, dass der Transponder keine eigene Stromversorgung braucht und deswegen sehr klein sein kann. Der Nachteil ist, dass bei diesem Verfahren der RFID-Reader und der Transponder nicht weit von einander entfernt sein dürfen, damit die Übertragung funktioniert.

NFC

Jetzt haben wir schon viel über RFID geredet und dabei auch noch NFC erwähnt. Das steht für **N**ear **F**ield **C**ommunication oder Deutsch: Nahfeldkommunikation. NFC ist eine Unterart von RFID, die nur mit 13,56 MHz arbeitet. Wir können dies an unserem RFID-Reader benutzen, weil auch der in diesem Frequenzbereich arbeitet. NFC ist eine neuere Technik und erlaubt auch, dass zwei Geräte wie Smartphones miteinander kommunizieren und Daten austauschen können. Auch kontaktloses Bezahlen ist dadurch häufig möglich.

RFID am Raspberry Pi

Jetzt haben wir genug Grundlagen gehabt und wollen diese Technik nun am Pi benutzen. Dafür müssen wir zuerst den RFID-Reader an den Pi anschließen. Für die Kommunikation zwischen dem Pi und dem Reader wird das SPI Protokoll benutzt. Wie du das einrichtest, erfährst du unter http://cw42.de/spi (auch wenn du SPI schon aktiviert hast, musst du teilweise noch weitere Einstellungen treffen, das gilt vor allem bei älteren Raspbian Versionen). Bei allen aktuellen Betriebssystem Versionen solltest du aber über `sudo raspi-config` -> 9 Advanced Options -> A5 SPI -> YES (alles bestätigen und dann neustarten) schon einfach SPI aktiviert haben.



RC522 Modul	Raspberry Pi
1 - SDA	GPIO8 CE0
2 - SCK	GPIO11 SCKL
3 - MOSI	GPIO10 MOSI
4 - MISO	GPIO9 MISO
5 - IRQ	
6 - GND	GND
7 - RST	3.3V

8 - 3.3V	3.3V

Bibliotheken hinzufügen

Das Anschließen des Bauteils haben wir jetzt hinter uns und können uns nun wieder um die Software kümmern. Für die Kommunikation zwischen Pi und dem Reader wird das SPI Kommunikationsprotokoll benutzt. Im Gegensatz zum 5510 Display, welches im Starterkit enthalten ist, senden wir diesmal nicht einfach nur Daten an den Reader, sondern müssen diese natürlich auch empfangen, um die einzelnen Karten wieder auslesen zu können. Deswegen ist die Software auch ein wenig komplizierter, aber natürlich gibt es dafür auch schon eine Bibliothek, die wir nur für unsere Zwecke anpassen müssen. Es gibt zwei verschiedene Komponenten, die wir runterladen müssen, zuerst die Bibliothek mit der Python über die GPIO Pins SPI kommunizieren kann:

Bash
```
$ git clone https://github.com/lthiery/SPI-Py.git
```

Bash
```
$ cd SPI-PY
```

Bash
```
$ sudo python setup.py install
```

Und jetzt die Bibliothek für die Kommunikation mit den Reader.

Bash
```
$ git clone
https://github.com/coding-world/MFRC522-python.git
```

Bash
```
$ cd MFRC522-python
```

Wenn du jetzt **Read.py** ausführst und die RFID-Karte oder den RFID-Anhänger an den Reader am Pi hältst, solltest du folgendes sehen:

Bash
```
$ sudo python Read.py
```

```
Card detected
Card read UID: 219,160,58,213
Size: 8
```

Wenn nichts angezeigt wird, obwohl du mit den RFID-Transpondern auch ein wenig länger (1-2 Sekunden) in der Nähe der Vorderseite bleibst, ist entweder etwas beim Anschließen falsch gelaufen oder beim Aktivieren/Installieren der Bibliotheken. Am besten versuchst du einfach erstmal alles nochmal.

Arbeiten mit RFID-Karten

Dein RFID-Reader sollte jetzt funktionieren. In diesem Teil zeigen wir dir, wie man das Read.py und Write.py Script so ändert, dass wir eigene Namen oder ID's auf die Karten schreiben und diese später auch wider auslesen können.

Alle RFID-Karten haben eine eigene UUID, also eine eigene eingebaute Kennung, die auch auf der Karte gespeichert ist. Wir können aber auch noch weitere Daten auf die Karten schreiben und auslesen. So lassen sich zum Beispiel Karten für die Zeiterfassung bauen. Wenn jemand ins Büro kommt, registriert er sich mit seiner RFID-Karte. Wir steigen einfach ein. Als erstes modifizieren wir das Write.py und Read.py Programm, so dass wir eigene Daten auf die RFID Speicher schreiben und lesen können.

Daten auf RFID Schreiben

Unsere RFID-Karten haben 16 Sektoren. In jedem Sektor können wir eine Zahl von 1 bis 255 speichern. Wenn du das Script Read.py ausführst und eine RFID-Karte vor den Leser hältst, dann sollte die Ausgabe so aussehen:

```
Sector 8 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

16 Sektoren, die alle auf 0 stehen. Maschinencode ist zwar toll, aber wer kann sich das merken? Wir wollen lieber Buchstaben speichern. Wenn wir 0 als A nehmen und B als 1 und C als 2 dann sind wir schnell durch. Dazu kommt dann noch das kleine a,b,c, die Zahlen und Sonderzeichen. Das klingt nach viel Mühe. Zum Glück haben andere das Problem schon gelöst. Es gibt den **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, kurz ASCII. Da sind alle Zeichen enthalten, die wir brauchen und das beste ist, Python kann das übersetzen (Nur auf die deutschen

Umlaute und Sonderzeichen müssen wir verzichten, aber das ist ja auch nicht so schlimm ;)). Mehr zu ASCII gibt es auch auf https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange.

Wir erstellen jetzt ein kleines Script, das eine Eingabe in ASCII Code umwandelt. Diesen Code speichern wir dann auf eine RFID-Karte.

Erstelle eine neue Datei und schreibe folgenden Inhalt hinein: **nano ASCII.py**

Python
```
name=raw_input("Gib einen Kartennamen ein: ")
if len(name)>16:
  name=name[:16]
data = [ord(x) for x in list(name)]
print data
```

Ok, das sieht komplizierter aus als es ist. In der ersten Zeile holen wir uns die Eingabe des Nutzers und speichern diese in der Variablen name. Mit `ran_input("Frage")` können wir im Terminal dem Benutzer des Programms eine Frage stellen und die Antworten dann in einer Variable speichern und später weiterverwenden. Wir können 16 Sektoren also 16 Zahlen von 0 bis 255 speichern. Falls der Nutzer aber mehr Zeichen eingegeben hat zum Beispiel 18 oder 20, dann ist das zu viel. In der zweiten Zeile prüfen wir das. `if len(name)>16:` Bedeutet, wenn die Länge von `name` > (größer als) 16 ist, dann mache Folgendes. Die Länge ist dabei natürlich die Anzahl der Zeichen. Die Funktion `len()` gibt uns die Länge einer Variable aus. Das kannst du selbst testen mit

Python
```
print len(„Das ist ein TEST“)
```

Falls unsere Variable name länger ist als 16 Zeichen, dann wird der eingerückte Teil ausgeführt und zwar: `name=name[:16]` . Übersetzt heißt das: Speichere in die Variablen `name` nur die ersten 16 Stellen aus der Variablen name. Mit Variable[anfangsWert:Endwert] können wir einzelne Zeichen aus einer Zeichenkette ausschneiden und ausgeben oder wieder in einer neuen oder der alten Variablen speichern.

Jetzt kommt der Kern des Ganzen, das Übersetzen der Zeichen in ASCII.

Python
```
data = [ord(x) for x in name]
```

data ist die Variable, in der nachher alle Zeichen gespeichert werden. `[ord(x) for x in list(name)]` besteht aus zwei Teilen. `ord(x)` und `for x in name`. Das

erste ist eine Python Funktion, die wir auch so aufrufen können mit `print ord("A")`. Das zweite, die for-Schleife, ist ähnlich wie die while-Schleife und lässt uns einzelne Elemente durchgehen. `print ord(„A“)` gibt uns den ASCII Wert, also eine Zahl aus, bei **A** ist das **97** Jetzt bleibt der kryptische Teil for `x in`. Das bedeutet, dass es jedes Element in der Variablen **name** ord(x) ausführt. Das x wird dabei durch den Wert aus der Liste ersetzt. Für unser **['h', 'a', 'l', 'l', 'o']** wäre das also ord(„h“) ord(“a“) ord(„l“) usw. Das ergibt zusammen: **[104, 97, 108, 108, 111]**

Nun haben wir also erfolgreich die Buchstaben in ein Format umgewandelt, welches wir auf der Karte speichern können, aber dafür müssen wir das natürlich auch wirklich tun.

Um nicht alles neu schreiben zu müssen, benutzen wir als Grundlage die **Write.py**-Datei. Dazu kopieren wir diese erstmal in eine neue Datei.

Bash
```
$ cp Write.py MyWrite.py
```

Jetzt können wir **MyWrite.py** bearbeiten und immer noch auf die alte Datei zugreifen. Diese öffnen wir jetzt mit nano und fügen unter `ìmport signal*, also in Zeile 8 unser vorheriges Programm ein, die Datei müssten bei dir dann so aussehen:

Python
```
#!/usr/bin/env python
# -*- coding: utf8 -*-

import RPi.GPIO as GPIO
import MFRC522
import signal

name=raw_input("Gib einen Kartennamen ein: ")
if len(name)>16:
 name=name[:16]
data = [ord(x) for x in list(name)]

continue_reading = True
```

Dazu müssen wir noch Zeile 56 und 77 die neuen *data* Variablen auskommentieren, weil diese ansonsten unsere Variablem die wir am Anfang definiert haben, überschreiben. Das machst du ganz einfach, indem du vor diese Zeilen ein # schreibst. Wenn du nano als deinen Editor benutzt, kannst du mit `Strg/Ctrl + w` einen Suchbegriff eingeben und dann mit Enter suchen, und nano springt dann zu dieser Zeile.

Sollte sich bei dir der Fehlerteufel eingeschlichen haben, kannst du mit dem folgenden Befehl das Programm, mit den Änderungen, herunterladen:

```Bash
$ wget cw42.de/p/MyWrite.py
```

Führe das Script nun aus mit `python MyWrite.py` Gib einen Namen oder eine ID für deine RFID-Karte ein, bestätige das mit ENTER und halte dann deine RFID-Karte an den Reader.

Die Ausgabe sollte ca so aussehen:

```
Sector 8 looked like this:
Sector 8 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Data written
It now looks like this:
Sector 8 [104, 97, 108, 108, 111, 255, 255, 255, 255, 255, 2
```

Die Zahlen nach Sector 8 sind unser **hallo** . Die 255 sind Platzhalter die automatisch erstellt werden.

Jetzt bist du in der Lage, deine Karten mit eindeutigen Namen und ID's zu belegen. Die Karten lassen sich übrigens mehr als 10.000 mal wieder beschreiben. Du kannst sie also jederzeit neu beschreiben!

RFID-Karte lesen

In diesem Teil werden wir die Informationen einer RFID-Karte auslesen und den ASCII Zahlen-Code in lesbaren Code umwandeln. Bevor wir weitermachen, müssen wir dabei zuerst etwas bei der Hauptdatei verändern, die wir auch zum Schreiben benutzt haben. Wenn wir die eingebauten Lese-Funktionen ausführen, werden zwar die Daten der Karten im Terminal ausgegeben, aber das bringt uns erstmal nichts, wenn wir mit diesen Daten noch etwas ausführen wollen. Deswegen bessern wir da nochmal etwas nach. Öffne die Datei `MFRC522.py` mit nano MFRC522.py und bei Zeile 377 fügst du folgende Zeile ein: `return backData` Das sollte dann so bei dir aussehen und dann natürlich nicht das Speichern vergessen: `python if len(backData) == 16: print "Sector "+str(blockAddr)+" "+str(backData) return backData`

Auch diese Datei kannst du dir natürlich mit folgendem Befehl herunterladen:

Bash
```
$ wget cw42.de/p/MFRC522.py
```

Als nächstes wollen wir die Karte auslesen. Um uns da die Tipparbeit zu sparen, benutzen wir **Read. py** als Grundlage und kopieren den Inhalt in eine neue Datei.

Bash
```
$ cp Read.py MyRead.py
```

Dann öffnen wir **MyRead.py** und ändern den folgenden Programmcode:

Python
```
# Check if authenticated
        if status == MIFAREReader.MI_OK:
            MIFAREReader.MFRC522_Read(8)
```

Vor `MIFAREReader.MFRC522_Read(8)` schreiben wir data =, damit das so aussieht: (natürlich speichern nicht vergessen)

Python
```
# Check if authenticated
        if status == MIFAREReader.MI_OK:
            data = MIFAREReader.MFRC522_Read(8)
```

Und warum haben wir das Ganze jetzt so gemacht? Mit `MIFAREReader.MFRC522_Read(8)` wird eine Funktion in der Datei MFRC522.py aufgerufen. Diese Funktion gibt normalerweise mit print den Wert auf der Karte aus. Wir haben zusätzlich das `return` ausgegeben. Dadurch wird der Wert nun auch übermittelt und wir können Ihn in die Variable `data` speichern. In der Variablen `data` liegt nun die Liste mit den Nummern. Diese Liste wollen wir nun von ASCII wieder zurück wandeln in lesbare Zeichen. Daher fügen wir unter dem `data = MIFAREReader.MFRC522_Read(8)` folgendes ein:

Python
```
text = "".join(chr(x) for x in data)
print text
```

Jetzt müssen wir natürlich wieder die ganze `data`-Variable durchgehen und alle Zahlen, die ein ASCII Code sind, wieder zurück in Zeichen umwandeln die wir auch wirklich benutzen können. Also genau das umgekehrte Verfahren, das wir schon in die **MyWrite.py** Datei gebaut haben. Dafür nutzen wir die Funktion **join** und **chr** . Join zieht die Werte zusammen und `chr(x) for x in data)` macht ein chr(x) für jeden Eintrag, also für jeden Zahlenblock wieder ein Zeichen in unserer Liste. `chr()`

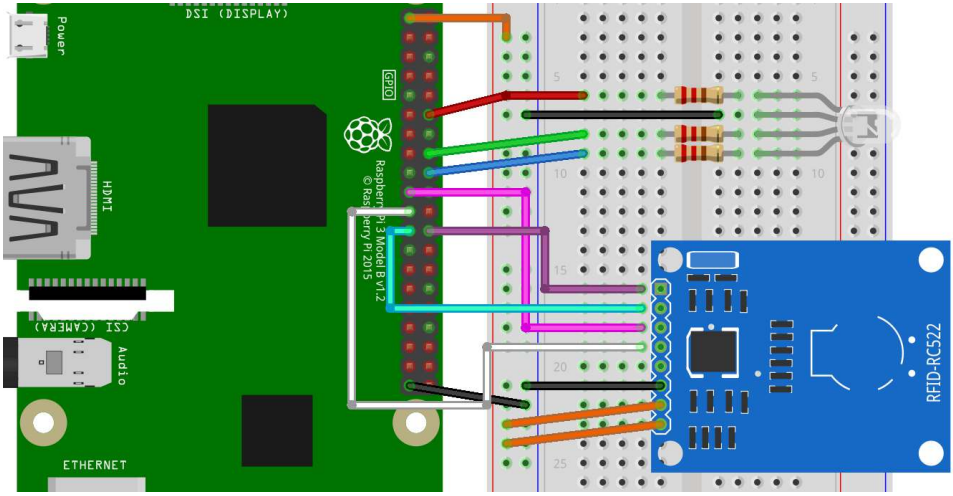
ist dabei der Gegensatz zu `ord()`, wir wandeln also ASCII Zahlen in Buchstaben um. Damit wir dann nicht "h", "a", "l", "l", "o" lesen, setzen wir das Join davon. Die Ausgaben ist somit "hallo". (Natürlich auch nur, solange du zu Beginn Hallo auf die Zahlen geschrieben hast.

RFID Projekte

Nachdem die Grundlagen für das Arbeiten mit RFID-Modulen gegeben sind, wollen wir uns jetzt ansehen wie wir Projekte ganz praktisch umsetzen können. Dafür haben wir kleine Beispiel-Projekte für dich ausgearbeitet.

Kontaktlos LEDs zum Leuchten bringen

Fangen wir doch zuerst damit an, LEDs ohne Kontakt zum Leuchten zu bringen. Das Ziel ist es, jeder Karte eine Farbe zuzuordnen und eine LED entsprechend leuchten zu lassen. So sparen wir uns das nervige Umschalten. Natürlich lässt sich dieses Projekt auch beliebig erweitern z.B. mit einem RGB-LED Stripe. um dein ganzes Zimmer zum Leuchten zu bringen. Legen wir gleich mal los. Zuerst musst du eine RGB-LED an deinen Pi anschließen, im folgenden findest du den Schaltplan.



Anschlüsse Pi	Anschlüsse Breadboard
GPIO 18	1. Bein RGB LED
GND	2. Bein (das längste) RGB LED
GPIO23	3. Bein RGB LED
GPIO 24	4. Bein RGB LED

Als nächstes kopieren wir wieder die **Read.py** Datei in die **farben.py** Datei, mit der

wir jetzt arbeiten werden. Insgesamt gibt es drei Code Stellen, die du einfügen musst, um die LEDs zum Leuchten zu bringen.

Wichtig ist, dass wir auch wirklich mit dem folgenden Befehl die `RPi.GPIO` Bibliothek importieren, wenn dort etwas anderes in Zeile steht, kannst du das ersetzen.

farben.py Zeile 1

Python
```
1 import RPi.GPIO as gpio
```

Im nächsten Schritt müssen wir dem Pi erstmal sagen, dass wir RGB-LEDs angeschlossen haben. Dafür definieren wir zuerst Variablen mit den GPIO Anschlüssen und legen dann den Zustand der Pins fest. Im nächsten Schritt definieren wir die Funktion **ledsOff()**. Die macht nichts anderes als der Name schon sagt. Sie schaltet alle LEDs aus. Warum diese Funktion nützlich ist, werden wir später noch erfahren. Bei der Funktion **end_read()** ist noch das `gpio.cleanup` wichtig, damit werden die GPIO Pins aufgeräumt und können dann wieder beim nächsten Start ordentlich benutzt werden.

Zeile 11 bis 28

Python
```
1  # LED Setup
2  rot = 24
3  gruen = 23
4  blau = 18
5  gpio.setmode(gpio.BCM)
6  gpio.setup(rot, gpio.OUT)
7  gpio.setup(gruen, gpio.OUT)
8  gpio.setup(blau, gpio.OUT)
9
10 def ledsOff():
11     gpio.output(rot, gpio.LOW)
12     gpio.output(gruen, gpio.LOW)
13     gpio.output(blau, gpio.LOW)
14
15 def end_read(signal,frame):
16     global continue_reading
17     continue_reading = False
18     gpio.cleanup()
```

Jetzt, da wir die LED angeschlossen haben, müssen wir natürlich noch Auslesen, welche Karte zu welcher Farbe gehört. Dafür haben wir drei verschiedene if-Bedingungen, heißt für jede Möglichkeit jeweils eine Bedingung. Zuerst aber kontrollieren wir mit `if type(data) is list:`, ob sich in der Variable *data* überhaupt eine Liste befindet. Das ist wichtig, denn teilweise wird zwar eine Karte erkannt, aber der Inhalt der Karte kann nicht ordentlich gelesen werden. Das kann passieren wenn du zum Beispiel die Karte zu schnell von dem Lesegerät entfernst. Im nächsten Schritt wandeln wir, wie bekannt, die ASCII Zahlen in Buchstaben um, dann kommen die Bedingungen. Dort benutzen wir `if "gruen" in text`, um zu kontrollieren ob sich der String "gruen" in der Variablen *text* befindet. Das müssen wir machen, denn wenn wir nur "gruen" == text machen würden, könnte es dort teilweise zu falschen Ergebnissen kommen, wenn der Rest der Karte mit noch anderen Inhalten befüllt ist. In der Bedingung schalten wir zuerst alle LEDs aus und schalten dann immer die LED an, die wir auch anschalten wollen.

Python

```
if status == MIFAREReader.MI_OK:
    data = MIFAREReader.MFRC522_Read(8)
    MIFAREReader.MFRC522_StopCrypto1()
    if type(data) is list:
        text = "".join(chr(x) for x in data)
        print text
        if "gruen" in text:
            ledsOff()
            gpio.output(gruen, gpio.HIGH)
        if "rot" in text:
            ledsOff()
            gpio.output(rot, gpio.HIGH)
        if "blau" in text:
            ledsOff()
            gpio.output(blau, gpio.HIGH)
    else:
        print "Es gab einen Fehler :("
```

So einfach ist der Programmcode dann auch ;) Bevor du dein Programm aber testen kannst, musst du erst noch deine Karten mit der **MyWrite.py** Datei so beschreiben, dass auf diesen auch wirklich gruen, rot und blau steht, sonst kann natürlich das

Programm auch nicht funktionieren. Gegen den Fehlerteufel kannst du dir das Programm auch nochmal herunterladen:

Bash

```
$ wget cw42.de/p/farben.py
```

Das RFID Soundboard

Gerade eben haben wir uns schon damit beschäftigt die GPIOs und damit eine LED zu kontrollieren, aber natürlich haben wir damit auch noch andere Möglichkeiten. Eine davon ist ein interaktives Soundboard. Wir haben als Beispiel einfach mal Tiergeräusche genommen, aber natürlich lassen sich auch andere Geräusche abspielen, die im Zweifelsfall auch situationsbedingt weiter helfen können. Damit du nicht selbst alles im Internet zusammen googlen musst, gibt es hier eine kleine Zusammenfassung von uns, die du dir einfach herunterladen kannst.

```Bash
$ git clone https://github.com/coding-world/animal-sounds
```

Noch mehr Sounds findest du auch auf https://www.freesound.org/. Dort gibt es nur Creative Common Sounds, heißt du kannst diese einfach benutzen und verändern wie du willst.

Im nächsten Schritt müssen wir noch das Audio richtig einstellen. Der Raspberry Pi hat selbst keine Lautsprecher, deswegen brauchst du entweder Kopfhörer oder externe Lautsprecher, die du an den Pi anschließen kannst. Wir haben in unseren Fall Lautsprecher genommen, die mit einem USB-Anschluss mit Strom versorgt werden und haben die Verbindung zwischen Raspberry Pi und den Boxen mit einem Klinkenstecker vorgenommen. Wenn du diese Art der Verbindung nutzt, musst du teilweise noch einmal die Einstellungen ändern, damit auch wirklich Sounds über den Kopfhöreranschluss abgespielt werden.

`sudo raspi-config` -> 9 Advanced Options -> A9 Audo -> 1 Force 3.5mm [...]

Falls dein Monitor oder Fernsehr Lautsprecher eingebaut hat, die über HDMI funktionieren, kannst du auch HDMI als Sound-Ausgang wählen. Um zu testen, ob alles richtig verbunden/angeschlossen ist, kannst du mit dem folgenden Befehl einzelne Töne über das Terminal abspielen:

```Bash
$ aplay animal-sounds/cow.wav
```

Wenn das klappt, solltest du das Muhen einer Kuh hören. Jetzt können wir uns daran machen, das Python Programm dafür zu schreiben. Wir benutzen dabei eine Python Bibliothek, die auf jedem Pi vorhanden ist, die `pygame` Bibliothek. Mit dieser können grafischen Spiele programmiert werden. Das machen wir jetzt nicht, nutzen aber den Sound-Teil dieser Bibliothek, den `mixer`, um Sounds ab zu spielen.

```python
import pygame
pygame.mixer.init()
pygame.mixer.music.load("animal-sounds/cow.wav")
pygame.mixer.music.play()
while pygame.mixer.music.get_busy() == True:
    continue
```

Das Programm ist ganz simpel, in Zeile 1 wird die Bibliothek importiert und in Zeile 2 wird dann die Mixer Komponente initialisiert. In Zeile drei wird festgelegt, wo sich die Sound Datei befindet, du musst hier den Pfad von deinem Python Programm zu der Sound Datei angeben. In Zeile 5 wird diese Datei dann abgespielt. Da Python sich immer beendet, sobald die letzte Zeile des Programms ausgeführt wurde, können wir mit der while-Schleife in Zeile 6 das Programm solange pausieren, wie die Sound Datei noch nicht komplett abgespielt wurde. Wenn du das Programm ausführst, solltest du wieder das Muhen einer Kuh hören. Wenn du in den Ordner animal-sounds wechselst (`cd animal-sounds`), kannst du dir mit `ls` alle Sounds anzeigen lassen, die du abspielen kannst.

Der nächste Schritt ist, diese Funktion in das Farben Python Programm einzubauen, wir benutzen dabei als Grundlage die **farben.py** Datei. Deswegen kopieren wir diese am besten in **rfid-sounds.py**. In diesem müssen wir am Anfang zuerst die Bibliothek importieren und dann initialisieren. Zeile 3-6 müssen wir dann die if-Bedingungen einbauen. Wenn du das machst, müsste jetzt beim Vorhalten einer Karte der Sound abgespielt werden. Wir haben hier mal etwas vorbereitet.

```python
if "kuh" in text:
    pygame.mixer.music.load("animal-sounds/cow.wav")
    pygame.mixer.music.play()
    while pygame.mixer.music.get_busy() == True:
if "huhn" in text:
    pygame.mixer.music.load("animal-sounds/chicken.wav")
    pygame.mixer.music.play()
    while pygame.mixer.music.get_busy() == True:
if "elefant" in text:
    pygame.mixer.music.load("animal-sounds/elephant.wav")
    pygame.mixer.music.play()
    while pygame.mixer.music.get_busy() == True:
```

Das Ganze funktioniert ähnlich wie mit den Farben. Aber anstatt eine LED zu schalten, spielen wir dieses Mal Sounds ab. Natürlich musst du dafür auch die Karten neu beschreiben!

Weitere Möglichkeiten

Unser Hauptprojekt ist mal wieder zu Ende, aber der wahre Spaß fängt jetzt natürlich erst wirklich an! Falls dir die Ideen fehlen, hätten wir hier noch ein paar:

Twitter Bot

Mit Twitter kannst du schnell kleine Nachrichten von maximal 140 Zeichen in die Welt senden. Wie wäre es also mit verschiedenen Karten, um der Welt deine Gefühlslage (gelangweilt, glücklich, <emotion einfügen>) mitzuteilen und das nur mit dem RFID-Readern und Karte.

Zugangskontrolle

Wie wäre es mit einer smarten Zugangskontrolle, die dir genau protokolliert, wann du wie dein Zimmer betreten hast, solche Systeme findest du auch bei größeren Arbeitgebern und bei Schließsystem. Sowas ließe sich natürlich auch mit einem Display oder weiterem verbinden ;)

Musik-Player

Dein Lieblingslied wird abgespielt, wenn du eine RFID Karte über den Reader hältst. Das ist fast wie früher als man CD's einlegen musste und macht mächtig viel Eindruck. Außerdem ist es ein riesen Spaß für Alle.

Weitere RFID Karten bekommst du übrigens super günstig auf CodingWorld.io

Auf Wiedersehen

Wir hoffen, dir hat dieses kleine Kit gefallen! Wenn du noch mehr erleben und arbeiten willst, findest du viele weitere spannende Projekte auf codingworld.io Dort helfen wir dir auch gerne bei Fragen und Problemen rund um dein Projekt.