RFID-Karten mit dem Raspberry Pi 5 und RC522 auslesen

Dieses Tutorial führt dich durch den Prozess, einen RC522 RFID-Reader über die SPI-Schnittstelle mit einem Raspberry Pi 5 zu verbinden und mit einer minimalistischen Python-Bibliothek die eindeutige ID (UID) von RFID-Karten auszulesen. Viele bestehende Anleitungen für den RC522 nutzen die RPi.GPIO-Bibliothek, die auf dem Raspberry Pi 5 nicht mehr ohne Weiteres funktioniert. **Dieses Tutorial löst dieses Problem**, indem es eine schlanke Python-Bibliothek nutzt, die auf der modernen und kompatiblen gpiod-Bibliothek aufbaut.

Kernfunktionen der Bibliothek

- **Schlankes Design**: Alle notwendigen Funktionen sind in einer einzigen, leichtgewichtigen Datei zusammengefasst.
- **Einfache Integration**: Die Bibliothek kann einfach durch Kopieren der Datei in dein Projekt eingebunden werden.
- **Raspberry Pi 5 Kompatibel**: Nutzt die moderne gpiod-Bibliothek anstelle des veralteten RPi.GPIO.
- **SPI-Fokus**: Optimiert für eine stabile und schnelle SPI-Kommunikation.

Hardware: Voraussetzungen und Verkabelung

Zuerst verbinden wir den RC522-Reader mit dem Raspberry Pi.

Benötigte Hardware

- Raspberry Pi 5
- RC522 RFID-Reader-Modul
- RFID-Karten oder -Tags (z.B. MIFARE Classic 1K)
- Jumper-Kabel (Dupont-Kabel)

Verkabelungsplan (SPI)

Verbinde das RC522-Modul wie in der folgenden Tabelle beschrieben mit den GPIO-Pins deines Raspberry Pi.

RC522 Pin	Pi 5 Pin (Physisch)	GPIO (BCM)	Beschreibung
SDA/SS	Pin 24	GPIO 8	Chip Select (CS)
SCK	Pin 23	GPIO 11	SPI Clock
MOSI	Pin 19	GPIO 10	Master Out -> Slave In
MISO	Pin 21	GPIO 9	Master In <- Slave Out
RST	Pin 15	GPIO 22	Reset
GND	Pin 6, 9, etc.	–	Ground (Masse)
VCC	Pin 1 oder 17	–	3.3V Stromversorgung

Wichtiger Hinweis: Schließe das RC522-Modul **ausschließlich an einen 3.3V-Pin** an. Die Verwendung eines 5V-Pins kann das Modul dauerhaft beschädigen!

Software: System einrichten

Bevor wir den Code ausführen können, müssen die SPI-Schnittstelle aktiviert und die notwendigen Bibliotheken installiert werden.

a) SPI-Schnittstelle aktivieren

Falls noch nicht geschehen, aktiviere die SPI-Schnittstelle auf deinem Raspberry Pi.

1. Öffne ein Terminal und gib sudo raspi-config ein.
2. Navigiere zu 3 Interface Options -> I4 SPI.
3. Bestätige die Frage, ob die SPI-Schnittstelle aktiviert werden soll, mit „Ja“ oder „Yes“.
4. Beende das Konfigurationstool. Ein Neustart kann erforderlich sein.

b) Systemabhängigkeiten installieren

Installiere die Python-Bibliotheken spidev für die SPI-Kommunikation und libgpiod für die GPIO-Steuerung.

```
sudo apt update
sudo apt install python3-spidev python3-libgpiod -y
```

Bibliothek und Beispiel-Code

Das gesamte Projekt besteht aus nur zwei Dateien: der Bibliothek selbst und einem Beispielskript, das die Anwendung demonstriert. Du findest den Code hier, kannst ihn dir aber auch aus unserem GitHub-Repository herunterladen.

a) Die Bibliotheksdatei: rc522_spi_library.py

Dies ist die eigentliche Bibliothek. Sie ist in sich geschlossen und enthält alle Klassen, Konstanten und Funktionen, um mit dem RC522 zu kommunizieren. Kopiere den folgenden Code und speichere ihn in einer Datei mit dem Namen rc522_spi_library.py.

```
# -*- coding: utf-8 -*-
#
# A lean Python library for the RC522 RFID reader on the Raspberry Pi 5 via SPI.
# Combines the necessary classes and constants for easy integration.
#
# Pollux Labs
# polluxlabs.net
#

import time
import logging

try:
    import gpiod
    import spidev
except ImportError:
    print("Important Note: The hardware libraries 'gpiod' and 'spidev' could not be imported.")
    print("This library is intended for use on a Raspberry Pi with the SPI interface enabled.")
    gpiod = None
    spidev = None

# --- Constants ---
# From `constants.py`

class RC522Registers:
    COMMAND_REG = 0x01
    COM_IRQ_REG = 0x04
    DIV_IRQ_REG = 0x05
    ERROR_REG = 0x06
```

```
    STATUS2_REG = 0x08
    FIFO_DATA_REG = 0x09
    FIFO_LEVEL_REG = 0x0A
    CONTROL_REG = 0x0C
    BIT_FRAMING_REG = 0x0D
    TX_CONTROL_REG = 0x14
    CRC_RESULT_REG_MSB = 0x21
    CRC_RESULT_REG_LSB = 0x22
    VERSION_REG = 0x37
    T_MODE_REG = 0x2A
    T_PRESCALER_REG = 0x2B
    T_RELOAD_REG_H = 0x2C
    T_RELOAD_REG_L = 0x2D
    MODE_REG = 0x11
    TX_AUTO_REG = 0x15

class RC522Commands:
    IDLE = 0x00
    CALC_CRC = 0x03
    TRANSCEIVE = 0x0C
    MF_AUTHENT = 0x0E
    SOFT_RESET = 0x0F

class MifareCommands:
    REQUEST_A = 0x26
    ANTICOLL_1 = 0x93
    SELECT_1 = 0x93
    HALT = 0x50
    READ = 0x30
    AUTH_A = 0x60

class StatusCodes:
    OK = 0
    ERROR = 1
    TIMEOUT = 3
    AUTH_ERROR = 5

DEFAULT_KEY = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]

# --- Exceptions ---
# From `exceptions.py`
```

```
class RC522Error(Exception):
    """Base exception for RC522 operations."""
    pass

class RC522CommunicationError(RC522Error):
    """Exception for communication errors with the RC522."""
    pass

# --- Main Class ---
# Combined and simplified logic from `rc522_reader.py`

class RC522SPILibrary:
    """
    A lean and standalone Python library for the RC522 RFID
reader
    on the Raspberry Pi 5, focusing on SPI communication.
    """

    def __init__(self, spi_bus=0, spi_device=0, rst_pin=22,
debug=False):
        """
        Initializes the reader.

        Args:
            spi_bus (int): The SPI bus (default: 0).
            spi_device (int): The SPI device (default: 0 for
CE0).
            rst_pin (int): The GPIO pin for the reset (BCM
numbering).
            debug (bool): Enables detailed log output.
        """
        self.logger = logging.getLogger(__name__)
        if debug:
            self.logger.setLevel(logging.DEBUG)
        if not spidev or not gpiod:
            raise RC522CommunicationError("The hardware
libraries 'spidev' and 'gpiod' are not available.")

        self.spi = spidev.SpiDev()
        self.spi.open(spi_bus, spi_device)
        self.spi.max_speed_hz = 1000000  # 1 MHz
```

```
        self.spi.mode = 0

        # GPIO setup for the reset pin using gpiod
        try:
            # Chip 'gpiochip4' is for physical pins on the Pi
5.
            # For Pi 4, this might be 'gpiochip0'.
            self.gpio_chip = gpiod.Chip('gpiochip4')
            self.rst_line = self.gpio_chip.get_line(rst_pin)
               self.rst_line.request(consumer="RC522_RST",
type=gpiod.LINE_REQ_DIR_OUT)
        except Exception as e:
            raise RC522CommunicationError(f"Error initializing
GPIO pin via gpiod: {e}")

        self._initialized = False
        self.initialize()

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.cleanup()

    def _write_register(self, reg, value):
        self.spi.xfer2([reg << 1 & 0x7E, value])

    def _read_register(self, reg):
         return self.spi.xfer2([(reg << 1 & 0x7E) | 0x80,
0])[1]

    def _set_bit_mask(self, reg, mask):
        current = self._read_register(reg)
        self._write_register(reg, current | mask)

    def _clear_bit_mask(self, reg, mask):
        current = self._read_register(reg)
        self._write_register(reg, current & (~mask))

    def _reset(self):
        """Performs a hardware reset of the RC522."""
```

```
        self.rst_line.set_value(0)
        time.sleep(0.05)
        self.rst_line.set_value(1)
        time.sleep(0.05)

    def initialize(self):
        """Initializes the RC522 chip."""
        self._reset()
          self._write_register(RC522Registers.COMMAND_REG,
RC522Commands.SOFT_RESET)
        time.sleep(0.05)

        self._write_register(RC522Registers.T_MODE_REG, 0x8D)
         self._write_register(RC522Registers.T_PRESCALER_REG,
0x3E)
          self._write_register(RC522Registers.T_RELOAD_REG_L,
30)
        self._write_register(RC522Registers.T_RELOAD_REG_H, 0)
        self._write_register(RC522Registers.TX_AUTO_REG, 0x40)
        self._write_register(RC522Registers.MODE_REG, 0x3D)
        self.antenna_on()
        self._initialized = True
        self.logger.info("RC522 initialized successfully.")

    def antenna_on(self):
                                                  if      not
(self._read_register(RC522Registers.TX_CONTROL_REG) & 0x03):
             self._set_bit_mask(RC522Registers.TX_CONTROL_REG,
0x03)

    def cleanup(self):
        """Resets the RC522 and releases resources."""
        if self._initialized:
            self._reset()
        if hasattr(self, 'rst_line') and self.rst_line:
            self.rst_line.release()
        if hasattr(self, 'gpio_chip') and self.gpio_chip:
            self.gpio_chip.close()
        self.spi.close()
           self.logger.info("RC522 resources have been
released.")
```

```
    def _communicate_with_card(self, command, send_data,
timeout=0.1):
        """Internal method for card communication."""
        irq_en = 0x77
        wait_irq = 0x30
          self._write_register(RC522Registers.COMMAND_REG,
RC522Commands.IDLE)
        self._write_register(RC522Registers.COM_IRQ_REG, 0x7F)
          self._set_bit_mask(RC522Registers.FIFO_LEVEL_REG,
0x80)

        for byte in send_data:
            self._write_register(RC522Registers.FIFO_DATA_REG,
byte)

          self._write_register(RC522Registers.COMMAND_REG,
command)
        if command == RC522Commands.TRANSCEIVE:
            self._set_bit_mask(RC522Registers.BIT_FRAMING_REG,
0x80)

        start_time = time.time()
        while time.time() - start_time < timeout:
                                                    n     =
self._read_register(RC522Registers.COM_IRQ_REG)
            if n & wait_irq:
                break
         self._clear_bit_mask(RC522Registers.BIT_FRAMING_REG,
0x80)

        if time.time() - start_time >= timeout:
            return StatusCodes.TIMEOUT, [], 0

         if self._read_register(RC522Registers.ERROR_REG) &
0x1B:
            return StatusCodes.ERROR, [], 0
        status = StatusCodes.OK
        back_data = []
        back_len = 0

        if n & 0x01:
```

```
            status = StatusCodes.ERROR

        if command == RC522Commands.TRANSCEIVE:
                                                fifo_size   =
self._read_register(RC522Registers.FIFO_LEVEL_REG)
                                                last_bits   =
self._read_register(RC522Registers.CONTROL_REG) & 0x07
            if last_bits != 0:
                back_len = (fifo_size - 1) * 8 + last_bits
            else:
                back_len = fifo_size * 8

            if fifo_size == 0:
                fifo_size = 1

            if fifo_size > 16:
                fifo_size = 16

            for _ in range(fifo_size):
back_data.append(self._read_register(RC522Registers.FIFO_DATA_
REG))

        return status, back_data, back_len

    def request(self):
        """
        Scans for cards in the antenna field.
        """
         self._write_register(RC522Registers.BIT_FRAMING_REG,
0x07)
                             status,   back_data,    _    =
self._communicate_with_card(RC522Commands.TRANSCEIVE,
[MifareCommands.REQUEST_A])
        if status != StatusCodes.OK or len(back_data) != 2:
            return StatusCodes.ERROR, None
        return status, back_data

    def anticoll(self):
        """
         Performs an anti-collision procedure to get a card's
UID.
```

```
        """
         self._write_register(RC522Registers.BIT_FRAMING_REG,
0x00)
                              status,   back_data,   _   =
self._communicate_with_card(RC522Commands.TRANSCEIVE,
[MifareCommands.ANTICOLL_1, 0x20])
        if status == StatusCodes.OK and len(back_data) == 5:
            # Checksum of UID
            checksum = 0
            for i in range(4):
                checksum ^= back_data[i]
            if checksum != back_data[4]:
                return StatusCodes.ERROR, None
            return StatusCodes.OK, back_data[:4]
        return StatusCodes.ERROR, None
```

b) Die Beispieldatei: example.py

Dieses Skript importiert die soeben erstellte Bibliothek und zeigt, wie man eine Kartenerkennung und das Auslesen der UID in einer einfachen Schleife implementiert. Speichere diesen Code im **selben Verzeichnis** wie die Bibliotheksdatei unter dem Namen example.py.

```
# -*- coding: utf-8 -*-
# Pollux Labs
# polluxlabs.net

import time
import logging
# Importiere die zuvor erstellte Bibliothek
from rc522_spi_library import RC522SPILibrary, StatusCodes

# Konfiguriere das Logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s')

def main():
    """
```

```
     Allgemeines Beispiel zum Auslesen der UID von einer beliebigen RFID-Karte.
    """
    print("Starte den RFID-Kartenleser...")
    print("Halte eine beliebige RFID-Karte vor den Reader.")
    print("Drücke STRG+C zum Beenden.")

    reader = None
    try:
        # Initialisiere die Bibliothek.
        # RST-Pin 22 (BCM) entspricht dem physischen Pin 15.
        reader = RC522SPILibrary(rst_pin=22)
         # Speichert die UID der zuletzt gesehenen Karte, um ständige Wiederholungen zu vermeiden
        last_uid = None

        while True:
            # 1. Suche nach einer Karte im Feld
            status, _ = reader.request()

            if status == StatusCodes.OK:
                 # 2. Wenn eine Karte im Feld ist, hole ihre UID (Anti-Kollision)
                status, uid = reader.anticoll()
                if status == StatusCodes.OK:
                     # Reagiere nur, wenn es eine neue Karte ist
                    if uid != last_uid:
                        last_uid = uid
                         # Konvertiere die UID in ein lesbares Format
                         uid_str = ":".join([f"{i:02X}" for i in uid])
print("\n================================")
                        print(f"Karte erkannt!")
                        print(f"  UID: {uid_str}")
print("================================")
                         print("INFO: Du kannst diese UID nun in deinem eigenen Code verwenden.")
            else:
                # 3. Wenn keine Karte mehr im Feld ist, setze
```

```
`last_uid` zurück
                if last_uid is not None:
                     print("\nKarte entfernt. Der Reader ist
bereit für die nächste Karte.")
                    last_uid = None

            # Kurze Pause zur Reduzierung der CPU-Last
            time.sleep(0.1)

    except Exception as e:
           logging.error(f"Ein unerwarteter Fehler ist
aufgetreten: {e}")
    except KeyboardInterrupt:
        print("\nProgramm wird beendet.")
    finally:
          # Stelle sicher, dass die Ressourcen am Ende
freigegeben werden
        if reader:
            reader.cleanup()
            print("RC522-Ressourcen erfolgreich freigegeben.")

if __name__ == '__main__':
    main()
```

Anwendung und Test

Jetzt ist alles bereit, um den RFID-Reader zu testen.

1. Navigiere in deinem Terminal in das Verzeichnis, in dem du die beiden Python-Dateien (rc522_spi_library.py und example.py) gespeichert hast.
2. Führe das Beispiel-Skript aus:Bashpython3 example.py
3. Das Programm startet und fordert dich auf, eine Karte an den Reader zu halten.
4. Wenn du eine RFID-Karte in die Nähe des Lesegeräts hältst, sollte deren UID auf dem Bildschirm erscheinen.

Beispiel-Ausgabe:

```
Starte den RFID-Kartenleser...
Halte eine beliebige RFID-Karte vor den Reader.
Drücke STRG+C zum Beenden.

================================
Karte erkannt!
  UID: 4A:F3:8B:1E
================================
INFO: Du kannst diese UID nun in deinem eigenen Code verwenden.
```

Diese UID kannst du nun kopieren und in deinen eigenen Projekten für Zugangskontrollen, zur Identifikation von Objekten oder zum Auslösen von Aktionen verwenden.