Practical 9

RFID stands for Radio Frequency IDentification. For our information, there are a few things required as parts of an operational RFID System in place:

- A reader, that is connected to (or integrated with) embedded system
- An antenna, that sends out / receives in a radio signal
- A tag (or transponder) that returns the signal with information added
- An algorithm that is coded / programmed in an embedded system for specific features.

RFID systems use a wide range of frequency bandwidth. The antennas on both the reader and the tag are tuned to a specific frequency to enable basic interoperability. Frequencies that are often used in access control are (examples in bracket)

- Low Frequency (120 - 1355kHz)	: HID Prox, EM, Nedap NeXS ...
- High Frequency (13.56 MHz)	: MIFARE Classic, DESfire ...
- Ultra High Frequency (860 - 980 MHz)	: RAIN RFID / EPC Gen 2 ...
- Microwave (2.45 GHz)	: Nedap TRANSIT ...

Only store serial number but not data to store

Only can store 1kb of data

Open door, start motorbike

Our Grove RFID Reader (Left) is running on 125kHz for PART 1. If you are interested to explore more in this RFID system, you can get a low cost reader RC522 for 13.56MHz that comes in a set (Right) for PART 2.

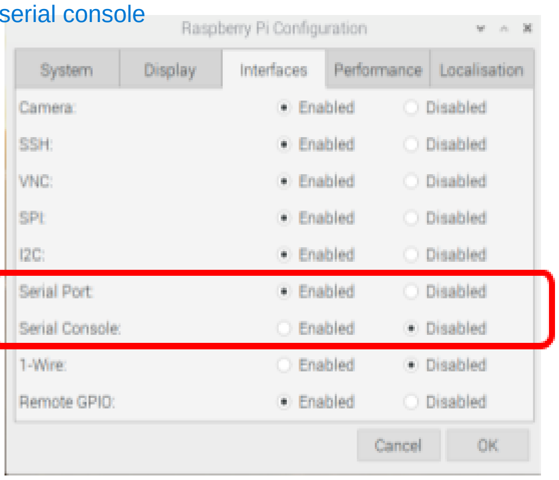
PART 1: RFID Scanner (125kHz)

This Grove-125KHz RFID Reader is a module used to read uem4100 RFID card information with two output formats: Uart and Wiegand. It has a sensitivity with maximum 7cm sensing distance. It can help us with projects like the access control systems.

* Reference: https://wiki.seeedstudio.com/Grove-125KHz_RFID_Reader/

Our Grove-125kHz RFID Reader is using serial communication (i.e., UART). Enable your Raspberry Pi Serial Port in the Raspberry Pi Configuration > Interfaces page (MUST disable the Serial Console) and restart.

Raspberry Icon -> Preferences -> Raspberry Configuration
-> Interfaces, enable serial port and disable serial console



Camera:	[✓ Enabled]	Disabled
SSH:	[✓ Enabled]	Disabled
VNC:	[✓ Enabled]	Disabled
SPI:	[✓ Enabled]	Disabled
I2C:	[✓ Enabled]	Disabled
Serial Port:	[✓ Enabled]	Disabled
Serial Console:	Enabled	[✓ Disabled]
1-Wire:	Enabled	[✓ Disabled]
Remote GPIO:	[✓ Enabled]	Disabled

sudo thonny /boot/config.txt
It will pop out notepad++ config.txt

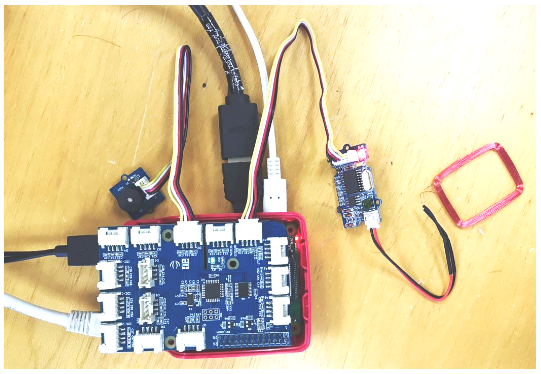
add on "enable_uart=1" if it doesn't exists in the text file

sudo reboot

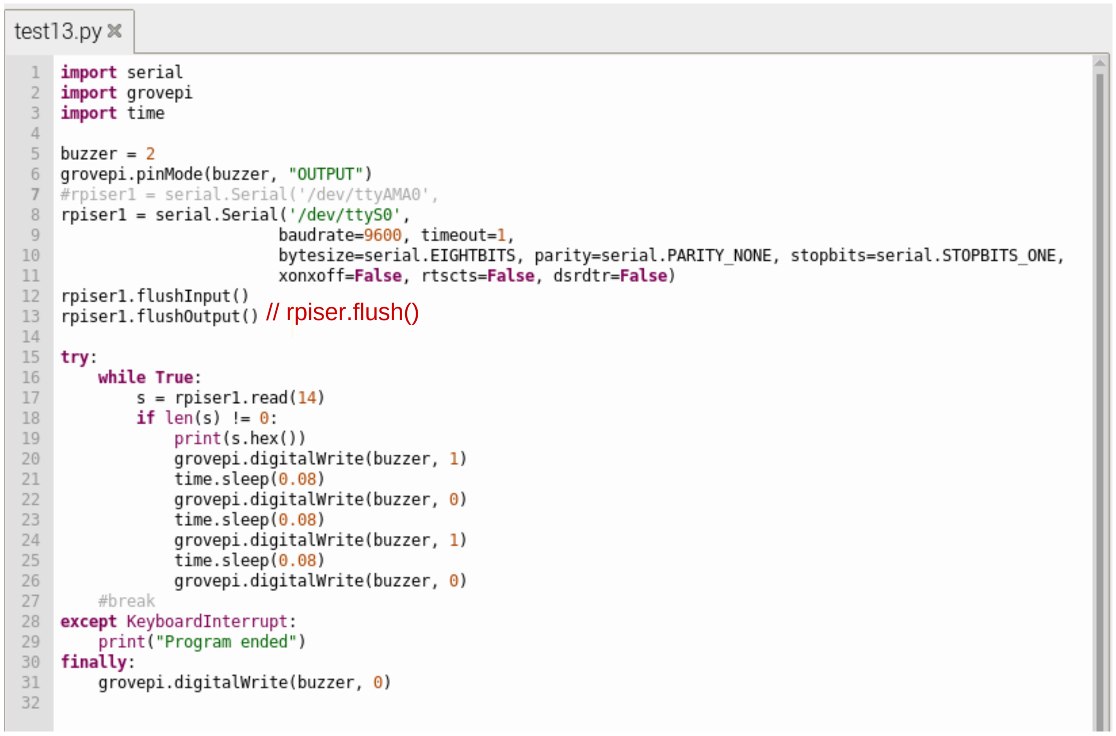
In Raspberry Pi version 1, 2, 3 and 4, there are several pre-configuration required to ensure the serial data is readable from the memory (i.e., ttyS0 or ttyAMA0). In Pi version 4, the bluetooth is enabled by default (then we have to use ttyS0) or set to disable-uart / mini-uart to use ttyAMA0 (refer to the reference on steps). For more information, please refer to the reference to understand on Raspberry Pi UART architecture:

* Reference: https://www.raspberrypi.org/documentation/configuration/uart.md

Step 1: We use Raspberry Pi Serial Port (RPISER port on Grove Pi). Connect the Buzzer at ***Port D2*** and Grove 125kHz RFID module at the ***RPISER*** port [**not SERIAL** Port].



Step 2: In **Thonny Python (ID)**, click “New” to create a new python file and Save As “test13.py”. Write the following codes in the python file.


test13.py

```python
import serial
import grovepi
import time

buzzer = 2
grovepi.pinMode(buzzer, "OUTPUT")
#rpiser1 = serial.Serial('/dev/ttyAMA0',
rpiser1 = serial.Serial('/dev/ttyS0',
                        baudrate=9600, timeout=1,
                        bytesize=serial.EIGHTBITS, parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE,
                        xonxoff=False, rtscts=False, dsrdtr=False)
rpiser1.flushInput()
rpiser1.flushOutput() // rpiser.flush()

try:
    while True:
        s = rpiser1.read(14)
        if len(s) != 0:
            print(s.hex())
            grovepi.digitalWrite(buzzer, 1)
            time.sleep(0.08)
            grovepi.digitalWrite(buzzer, 0)
            time.sleep(0.08)
            grovepi.digitalWrite(buzzer, 1)
            time.sleep(0.08)
            grovepi.digitalWrite(buzzer, 0)
    #break
except KeyboardInterrupt:
    print("Program ended")
finally:
    grovepi.digitalWrite(buzzer, 0)
```

Step 3: Run the codes with a 125kHz tag (i.e., access card or key chain).

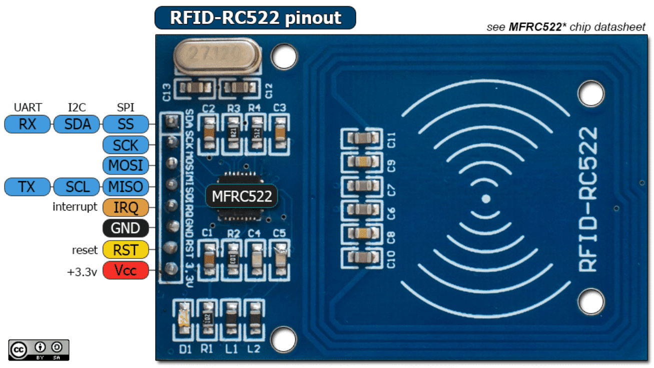
Task 1: Modify the code to accept a "authorized" access card

- Check the tag number of the access card (14 bytes) and hard coded it.

Task 2: Add on a Relay module to control a Solenoid Lock

- Simulate the door lock to be opened with an authorized access card.

PART 2: RFID Scanner (13.56MHz) - subject to availability

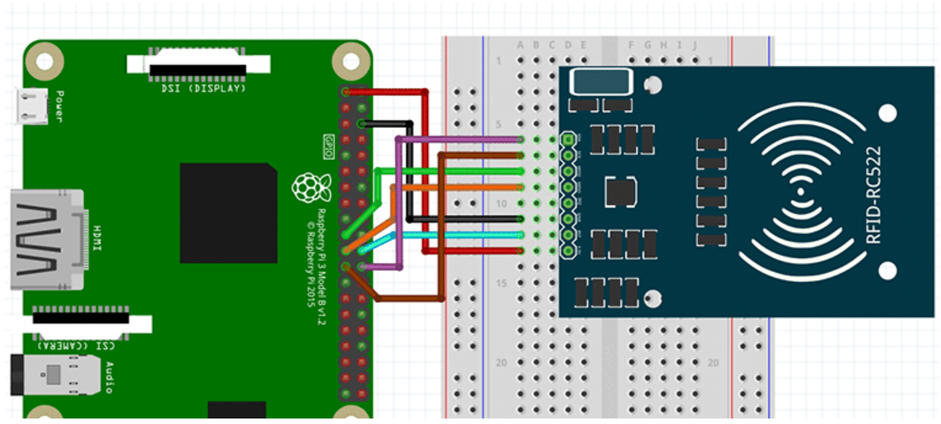


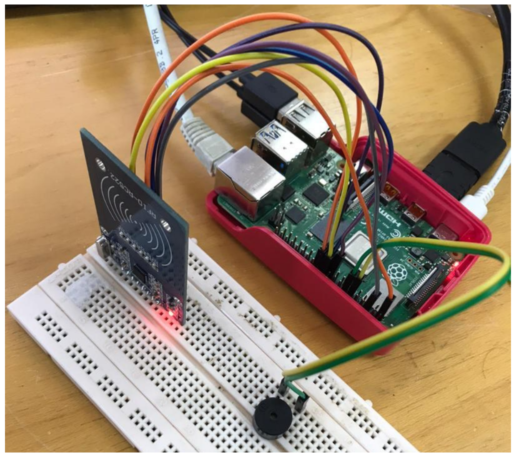
* Reference: https://pimylifeup.com/raspberry-pi-rfid-rc522/

Step 1: If you are using your own Raspberry Pi, do install mfrc522 libraries by typing:

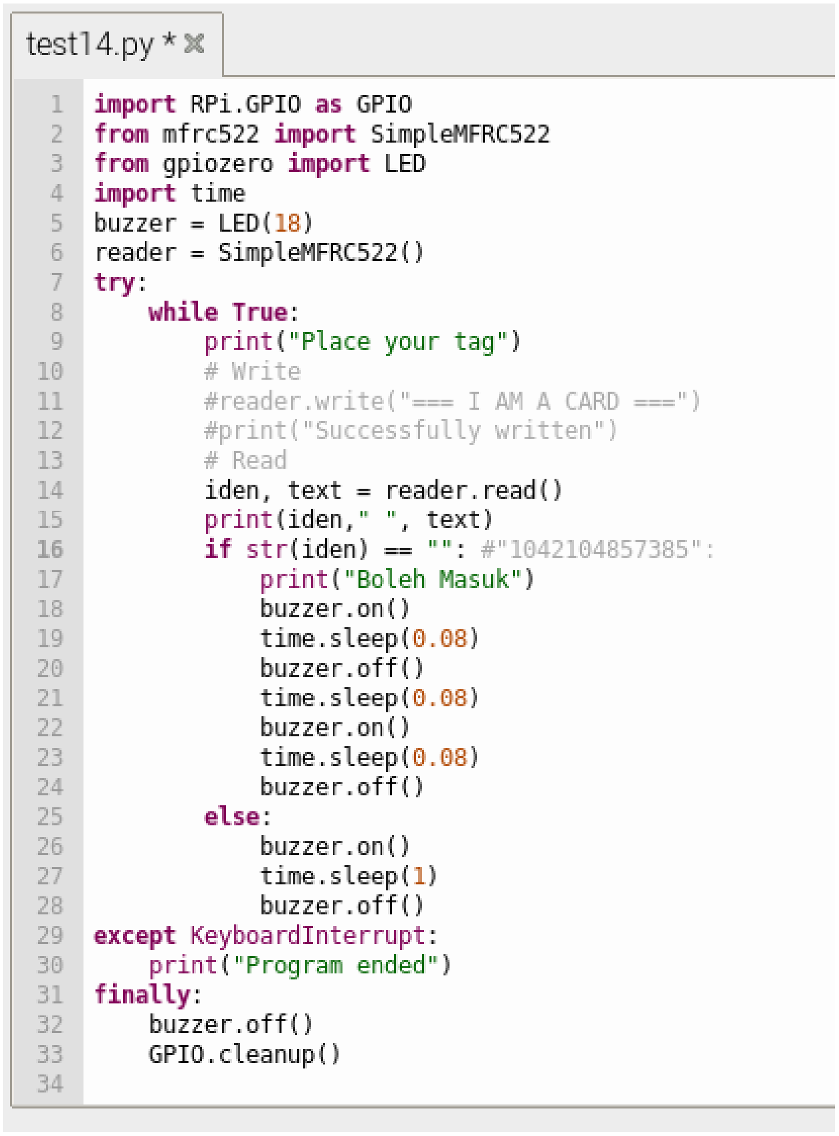
```
      sudo pip3 install mfrc522
```

Step 2: Follow the guidelines provided by the reference to connect RFID-RC522 with respective Raspberry Pi GPIO pins, add on a buzzer at GPIO 18:





Step 3: In **Thonny Python (ID)**, click “New” to create a new python file and Save As “test14.py”. Write the following codes in the python file.



test14.py *

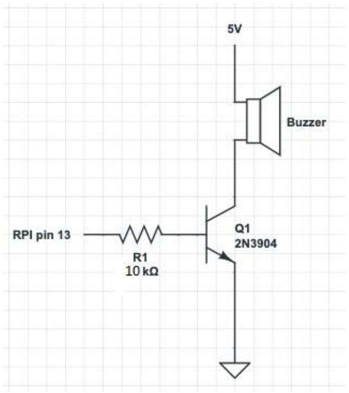
```python
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522
from gpiozero import LED
import time
buzzer = LED(18)
reader = SimpleMFRC522()
try:
    while True:
        print("Place your tag")
        # Write
        #reader.write("=== I AM A CARD ===")
        #print("Successfully written")
        # Read
        iden, text = reader.read()
        print(iden," ", text)
        if str(iden) == "": #"1042104857385":
            print("Boleh Masuk")
            buzzer.on()
            time.sleep(0.08)
            buzzer.off()
            time.sleep(0.08)
            buzzer.on()
            time.sleep(0.08)
            buzzer.off()
        else:
            buzzer.on()
            time.sleep(1)
            buzzer.off()
except KeyboardInterrupt:
    print("Program ended")
finally:
    buzzer.off()
    GPIO.cleanup()

```

Task 1: Modify the code to accept an “authorized” access card and run it.

<u>**Additional Notes**</u>

The buzzer directly connected to the Raspberry Pi GPIO pin may sink about ~20mA of current, and it exceeds the Raspberry Pi pin specifications (16mA max). Therefore, it is suggested to construct the circuit with an additional NPN transistor (2N3904) and a 10k ohm resistor, as shown at the diagram.


5V
Buzzer
RPI pin 13
R1
10 kΩ
Q1
2N3904