Inhaltsverzeichnis

Grundsätzliches

Diese Micropython Multitasking Framework (MMF) ist ein experimenteller Code, der nur Tests zur Machbarkeit eines solchen Frameworks erlauben soll.

So soll die Anlage funktionieren

- Eine grüne Led zeigt die Betriebsbereitschaft an. Die Anlage ist 5 Sekunden nach dem Einschalten bereit.
- Ein PIR - Sensor erfasst Bewegungen und löst einen Alarm aus.
- Im Alarmzustand blinkt die rote Led und der Buzzer erzeugt einen Sirenenton.
- Mit einem kurzen Druck auf Taster A verstummt der Buzzer, die rote Led blinkt weiter.
- Mit einem langen Druck auf Taster A wird der Alarm beendet. Die Anlage geht 5 Sekunden später wieder in die Bereitschaft zurück.
- Mit einem kurzen Druck auf Taster B wird ein Probealarm ausgelöst.
- Mit einem langen Druck auf Taster B wird das Programm beendet.

Die Hardware

Raspberry Pi Pico W
Bei den Experimenten wird ein ***Raspberry Pi Pico W*** verwendet. Der WiFi - Teil des Boards wird aber nicht verwendet, so dass ein ***Raspberry Pi Pico*** ebenfalls verwendet werden kann.
https://www.raspberrypi.com/products/raspberry-pi-pico/

Pico Explorer Base von Pimoroni
Für die Experimente wurde der Pico noch um dieses Board erweitert. Dadurch stehen unter anderem ein Display und 4 Buttons zur Verfügung.
https://www.raspberrypi.com/products/raspberry-pi-pico/

Sonstiges
Zusätzlich wurden noch 3 farbige Leds und ein Potentiometer angeschlossen.

Verwendete Pins

GPIO (General Purpose Input Output)
Diese Anschlüsse können als digitale Ein- und Ausgänge konfiguriert werden.
Alle Ausgänge sind PWM fähig und haben bereits einen 100 Ohm Widerstand in Serie eingebaut.

GP0 = 0 (Led rot)
GP2 = 2 (Led grün)
GP3 = 3 (Buzzer)
GP4 = 4 (PIR)

Taster
Die Taster belegen die GPIOs 12 - 15 und verbinden zu GND.

Button A = 12
Button B = 13

Die Taster können direkt ohne Angabe des Ports mit ButtonA() und ButtonB() erzeugt werden.

Die Alarmanlage

Eine erste Version (Alarm1.py)

Das Grundgerüst
Wir erstellen eine erste Version des Programms mit den wichtigsten Komponenten.

```
from MMF_Explorer import *

app = Application()

# Buzzer vorbereiten
app.buzzer = Buzzer(3)

# MMF - Komponenten
rot = DigitalOut(0)
gruen = DigitalOut(2)

taster_a = ButtonA()
taster_b = ButtonB()

app.add_components(rot, gruen, taster_a, taster_b)

# Funktionen
def alarm_ein():
    print('Alarm ein')

def alarm_aus():
    print('Alarm aus')

def programmende():
    alarm_aus()
    app.stop()

# Nachrichtenempfänger
def on_message(sender, topic, data):
    pass

# Hauptprogramm
alarm_aus()
app.run(message=on_message)
```

Taster und Leds
Wir realisieren die ersten Teile der Funktionsbeschreibung.

- **Eine grüne Led zeigt die Betriebsbereitschaft an. Die Anlage ist 5 Sekunden nach dem Einschalten bereit.**
- Ein PIR - Sensor erfasst Bewegungen und löst einen Alarm aus.
- **Im Alarmzustand blinkt die rote Led** und der Buzzer erzeugt einen Sirenenton.
- Mit einem kurzen Druck auf Taster A verstummt der Buzzer, die rote Led blinkt weiter.
- **Mit einem langen Druck auf Taster A wird der Alarm beendet. Die Anlage geht 5 Sekunden später wieder in die Bereitschaft zurück.**
- **Mit einem kurzen Druck auf Taster B wird ein Probealarm ausgelöst.**
- **Mit einem langen Druck auf Taster B wird das Programm beendet.**

Für die Taster benötigen wir den Nachrichtenempfänger.

```
def on_message(sender, topic, data):
    if sender == taster_a and topic == 'released':
        if data > 1000:
            alarm_aus()
        else:
            pass
    elif sender == taster_b and topic == 'released':
        if data < 1000:
            alarm_ein()
        else:
            programmende()
```

Jetzt müssen wir die Funktionen noch mit Leben füllen. Da wir und momentan nur um die Leds kümmern, sind die beiden Alarmfunktionen sehr einfach.

```
def alarm_ein():
    gruen.high = False
    rot.blink = 2

def alarm_aus():
    rot.high = False
    gruen.high = True
```

Die Alarmfunktion soll erst 5 Sekunden nach dem Ausschalten wieder aktiv werden. Dazu brauchen wir eine kleine Modifikation in der alarm_aus() Funktion.

```
def alarm_aus():
    rot.high = gruen.high = False
    app.after(5000, gruen.set_high, True)
```

Wir stellen sicher, dass beide Leds ausgeschaltet werden. Nach 5 Sekunden schaltet sich die grüne Led wieder ein.

Damit hätten wir das erste Ziel erreicht.

Die Sirene
- Eine grüne Led zeigt die Betriebsbereitschaft an. Die Anlage ist 5 Sekunden nach dem Einschalten bereit.
- Ein PIR - Sensor erfasst Bewegungen und löst einen Alarm aus.
- Im Alarmzustand blinkt die rote Led **und der Buzzer erzeugt einen Sirenenton**.
- **Mit einem kurzen Druck auf Taster A verstummt der Buzzer, die rote Led blinkt weiter.**
- Mit einem langen Druck auf Taster A wird der Alarm beendet. Die Anlage geht 5 Sekunden später wieder in die Bereitschaft zurück.
- Mit einem kurzen Druck auf Taster B wird ein Probealarm ausgelöst.
- Mit einem langen Druck auf Taster B wird das Programm beendet.

Der Buzzer soll einen Sirenenton erzeugen. Leider kann er das nicht von Haus aus, wir müssen das programmieren. Er kann lediglich einen Ton mit einer bestimmten Frequenz erzeugen.

Wir beginnen mit 300 Hz und steigern die Frequenz alle 50 ms um 20 Hz. Bei etwa 1000 Hz senken wir die Frequenz jeweils wieder um 20 Hz.

Dazu legen wir 2 neue Variablen für den Buzzer an und setzen sie gleich auf vernünftige Anfangswerte.
```
app.buzzer = Buzzer(3)
app.buzzer_freq = 300
app.buzzer_freq_schritt = 20
```

Alle 50 ms soll ein Schritt ausgeführt werden. Dazu erstellen wir eine Funktion, die als wiederholte Funktion der App hinzugefügt wird.
```
def buzzer_alarm():
    app.buzzer.set_tone(app.buzzer_freq)
    app.buzzer_freq += app.buzzer_freq_schritt
    if app.buzzer_freq > 1000 or app.buzzer_freq < 300:
        app.buzzer_freq_schritt *= -1

# Hauptprogramm
app.buzzer_alarm = app.add_function(50, buzzer_alarm)
alarm_aus()
```

Der Buzzer wäre jetzt aber dauernd aktiv. Wir müssen das noch steuern.
```
def buzzer_aus():
    app.buzzer_alarm.active = False
    app.buzzer.set_tone(-1)

def buzzer_ein():
    app.buzzer_alarm.active = True
```

alarm_aus() und alarm_ein() benutzen jetzt die Funktionen.
```
def alarm_ein():
    gruen.high = False
    rot.blink = 2
    buzzer_ein()

def alarm_aus():
    rot.high = False
    gruen.high = True
    buzzer_aus()
```

Für den stillen Alarm modifizieren wir noch den Nachrichtenempfänger.
```
    ...
    if sender == taster_a and topic == 'released':
        if data > 1000:
            alarm_aus()
        else:
            buzzer_aus()
    ...
```

Der PIR - Sensor
Es bleibt nur eine Zeile aus der Anforderungsliste übrig.

- **Ein PIR - Sensor erfasst Bewegungen und löst einen Alarm aus.**

Ein PIR ist sehr einfach. Sobald er eine Bewegung erfasst, gibt er ein HIGH - Signal aus. Sonst ist der Anschluss auf LOW.

```
pir = DigitalIn(4, pullup=False)
...
app.add_components(rot, gruen, taster_a, taster_b, pir)
```

Da der Pir beide Zustände zuverlässig setzt, müssen wir den internen PullUp ausschalten.

Der Pir sendet und eine Nachricht, wenn sich der Zustand ändert. Daher erweitern wir on_message().

```
    ...
    elif sender == pir:
        if gruen.high:
            alarm_ein()
```

Der Pir aktiviert den Alarm nur, wenn die Anlage scharf ist (grüne Led leuchtet).

Damit wäre die Aufgabe vollständig erfüllt.

Das vollständige Programm

```
"""
Micropython Multitasking Framework (MMF)
Alarmanlage

https://github.com/hobbyelektroniker/MMF
https://community.hobbyelektroniker.ch
https://www.youtube.com/c/HobbyelektronikerCh

Der Hobbyelektroniker, 07.04.2023
MIT License gemäss Angaben auf Github
"""

from MMF_Explorer import *

app = Application()

# Buzzer vorbereiten
app.buzzer = Buzzer(3)
app.buzzer_freq = 300
app.buzzer_freq_schritt = 20

# MMF - Komponenten
rot = DigitalOut(0)
gruen = DigitalOut(2)
pir = DigitalIn(4, pullup=False)

taster_a = ButtonA()
taster_b = ButtonB()

app.add_components(rot, gruen, taster_a, taster_b, pir)

# Wiederholte Funktionen
def buzzer_alarm():
    app.buzzer.set_tone(app.buzzer_freq)
    app.buzzer_freq += app.buzzer_freq_schritt
    if app.buzzer_freq > 1000 or app.buzzer_freq < 300:
        app.buzzer_freq_schritt *= -1

# Gewöhnliche Funktionen
def buzzer_aus():
    app.buzzer_alarm.active = False
    app.buzzer.set_tone(-1)

def buzzer_ein():
    app.buzzer_alarm.active = True

def alarm_ein():
    gruen.high = False
    rot.blink = 2
    buzzer_ein()

def alarm_aus():
    rot.high = gruen.high = False
    buzzer_aus()
    app.after(5000, gruen.set_high, True)

def programmende():
    alarm_aus()
    app.stop()

# Nachrichtenempfänger
def on_message(sender, topic, data):
    if sender == taster_a and topic == 'released':
        if data > 1000:
            alarm_aus()
```

```
        else:
            buzzer_aus()
    elif sender == taster_b and topic == 'released':
        if data < 1000:
            alarm_ein()
        else:
            programmende()
    elif sender == pir:
        if gruen.high:
            alarm_ein()

# Hauptprogramm
app.buzzer_alarm = app.add_function(50, buzzer_alarm)
alarm_aus()
app.run(message=on_message)
```