Abbildung 1: Türgong in der Entwicklung

Diesen Beitrag gibt es auch als PDF-Datei zum Download.

Unser Türgong mit einem SAB600-Chip tut seit vielen, vielen Jahren seinen Dienst. Seit wir umgezogen sind, nervt das Teil, weil es einfach zwischendurch meint, Laut geben zu müssen, auch wenn niemand den Klingelknopf gedrückt hat. Vermutlich sind es Spannungsspitzen oder HF-Impulse aus dem Haus oder von der Nachbarschaft, die ganz sporadisch blinden Alarm auslösen. Das ist vor allem mitten in der Nacht übel, da stehst du senkrecht im Bett. Es half auch nichts, den Eingang des Moduls zu entschärfen. Also suchte ich schon seit längerer Zeit nach einer alternativen Lösung mit einem unempfindlicheren Eingang.

Als ich kürzlich ein Mini-MP3-Player-Modul von ca. 2x2 cm Kantenlänge in die Finger bekam, hatte ich sie auch schon, die Lösung. Der SAB600 darf in Rente gehen und ein ESP8266 wird zusammen mit dem DF-Player mini die Stellung an der Haustür übernehmen. Und weil ich gerade am Bearbeiten der MQTT-Blogbeiträge bin, war es naheliegend, dieses Miniprojekt mit in dieses System einzubinden. Seien Sie gespannt und herzlich willkommen zur Reihe

Server und Clients unter MicroPython auf dem Raspi und der ESP-Familie

heute mit dem Thema

Der etwas andere Türgong

Sie ahnen es wahrscheinlich schon. Dieser Türwächter kann nicht nur einfach ding-dang-dong, freilich kann er das auch, aber dann kann er auch noch Hundegebell, Hahnenschrei, weinendes Baby, die Glocken von Big Ben, eigene Ansagen, den Sound von Ennio Morricone und vieles mehr. Der Phantasie sind keine Grenzen gesetzt. OK, was brauchen wir?

Hardware:

1	Raspberry Pi Bundle mit Pi 1 B+, Pi Pico und Zubehör oder
1	Raspberry Pi (1B+, 2, 2B, 3, 3B) ff. als Raspi bezeichnet
1	SD-Karte 8-16GB passend zum ausgewählten Raspi
1	Breadboard
diverse	Jumperkabel
1	Raspi Netzteil, 5V, 2,5A
1	Monitor (nur zur Einrichtung des Raspi)
1	Tastatur (nur zur Einrichtung des Raspi)
1	Maus (nur zur Einrichtung des Raspi)
1	HDMI-Kabel (nur zur Einrichtung des Raspi)
1	Netzwerkkabel
1 *	D1 Mini NodeMcu mit ESP8266-12F WLAN Modul
1 *	KY-009 RGB LED SMD Modul Sensor
1 *	Widerstand 680Ω (für rote LED)
1 *	Widerstand 1,5kΩ (für blaue LED)
1 *	Widerstand 2,7kΩ (für grüne LED)
1 *	Spannungsversorgung 5V, 500mA für den ESP8266
1 *	Mini MP3 Player DFPlayer Master Module
1 *	KY-004 Taster Modul Sensor Taste Kopf
1	kleiner Lautsprecher mit ca. 3 – 5W

Die mit einem "*" versehenen Teile in der Liste beziehen sich auf das aktuelle Projekt. Die anderen Teile waren schon in den vorangegangenen Folgen im Einsatz. Der Raspberry Pi und die damit zusammenhängenden Teile beziehen sich auf die Integration des Türgongs in das MQTT-Gesamtprojekt. Durch die Einbindung ins MQTT-Netz wird der Klingelersatz smart! Im zweiten Teil dieser Beschreibung erweitern wir die bestehende Broker-Konfiguration. Dadurch werden Klingelevents erfasst, gespeichert und (fern-) abrufbar.

Damit ich nicht alles noch einmal erzählen muss, finden Sie nachfolgend eine kurze Zusammenstellung dessen, was im MQTT-Projekt bisher alles gemacht wurde.

- In Folge 1 - MQTT - Raspberry Pi – Mosquitto und Node-RED haben wir Node-RED und den MQTT-Server Mosquitto auf dem Raspberry Pi installiert.
- In Folge 2 Ein ESP8266 als MQTT-Client und
- Folge 3 Erweiterung des MQTT-Home-Systems entstanden ein zweiter MQTT-Client mit dem ESP8266 sowie ein MQTT-Monitor mit einem ESP32 zum Steuern und Überwachen des Systems.
- Um die übersichtliche Darstellung der erfassten Daten durch das Tool Node-RED ging es in Folge 4. Dort wurde auch genau der Umgang mit der Weboberfläche von Node-RED behandelt.
- In der 5. Folge hatten wir uns mit der Absicherung des Datenverkehrs beschäftigt, sofern uns das die beschränkten Möglichkeiten des ESP8266 erlauben.

Software

Nachfolgend sind alle Softwareteile aufgeführt, die zum MQTT-Projekt und zum Türgong gebraucht werden.

Software für ESP32/8266:
Thonny oder
µPyCraft

Firmware:
MicroPython passend zum ESP32 oder ESP8266

MicroPython Module und Programmfiles:
umqttsimple.py
dfplayer.py
doorbell.py mit MQTT-Nutzung
Node-RED-Flow keller+heizung.json aus Folge 5
doorbell_wo_mqtt.py, falls Sie den Türgong ganz ohne MQTT nutzen möchten.

Software für den Raspi:
Raspian-Image
Imager.exe Brennprogramm für die SD-Card
Mosquitto-Broker
Node-RED

Software für die Windowsmaschine
xming
putty

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine **ausführliche Anleitung**. Darin gibt es auch eine Beschreibung, wie die **MicropythonFirmware (Stand 15.12.2021)** 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, bevor 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.

Der alternative Türgong – die Schaltung

Die Schaltung ist sehr übersichtlich, sie besteht gerade mal aus 4 Modulen und 3 Widerständen. Beschränkt man sich nur auf die Signaleinheit, dann fallen auch noch LED und Widerstände weg. Das heißt die LED wird nur für Statussignale bei der Funkübertragung gebraucht.

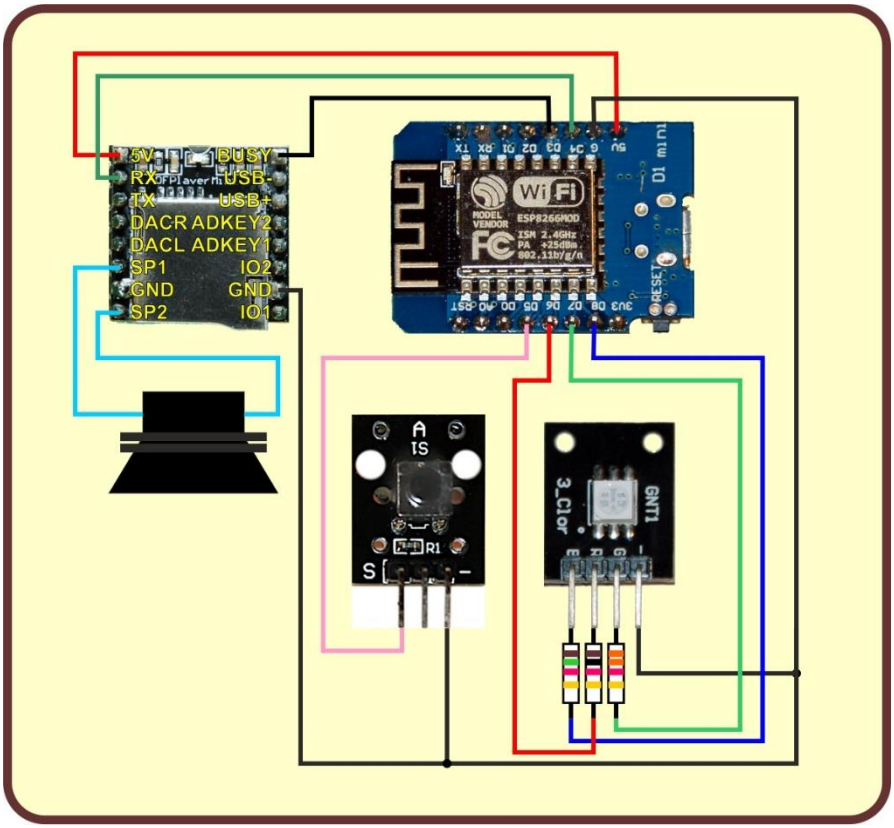

Abbildung 2: Schaltung Türgong mit DF-Player mini

Der Taster dient während der Entwicklungsphase als Türklingelknopf, damit wir nicht bei jedem Test zum Hauseingang sausen müssen. Auch der Taster kann später entfernt werden.

DF-Player und ESP8266 D1 mini werden mit den Steckleisten versehen. Das geschieht am besten in einem Steckbrett, da stimmen dann später Abstand und Ausrichtung. Zum Löten finden Sie eine Anleitung auf meiner AVR-Seite. Dort gibt es auch eine Anleitung zum Herstellen von Leiterplatten. Ich verwende hier eine Trägerplatine, weil ESP8266 und DF-Player außer den Steckleisten keine Befestigungsmöglichkeiten bieten. Aber darauf komme ich später zurück, bauen wir die Schaltung erst einmal auf einem Breadboard auf. Für den Taster reichen zwei Leitungen, Signal und Masse aka Ground (GND). Den Anschluss für die Versorgungsspannung sparen wir uns, denn wir schalten den internen Pullup-Widerstand im ESP8266 ein, der die Signalleitung auf +3,3 V zieht. Es kann aber später im Einsatz nötig werden, den Eingang zusätzlich mit einem externen Pullup und einem Kondensator gegen Masse zu puffern, falls wir beim Anschluss einer langen Klingelleitung feststellen, dass Störungen auftreten. Wird die Leitung durch den Pullup zu hochohmig abgeschlossen, wirkt sie wie eine Antenne und fängt eventuell störende hochfrequente Impulse ein, die erneut zu Problemen mit der Stabilität führen können.

Die LED gibt uns durch die drei Farben Informationen über den Boot-Vorgang. Blau blinkt sie, wenn der ESP8266 nach einem lokalen WLAN sucht, mit dem er sich verbinden möchte. Grünes Blinken ist der Heartbeat. Es sagt uns, dass das Programm ordnungsgemäß läuft. Und die rote LED sagt uns, dass ein Fehler vorliegt.

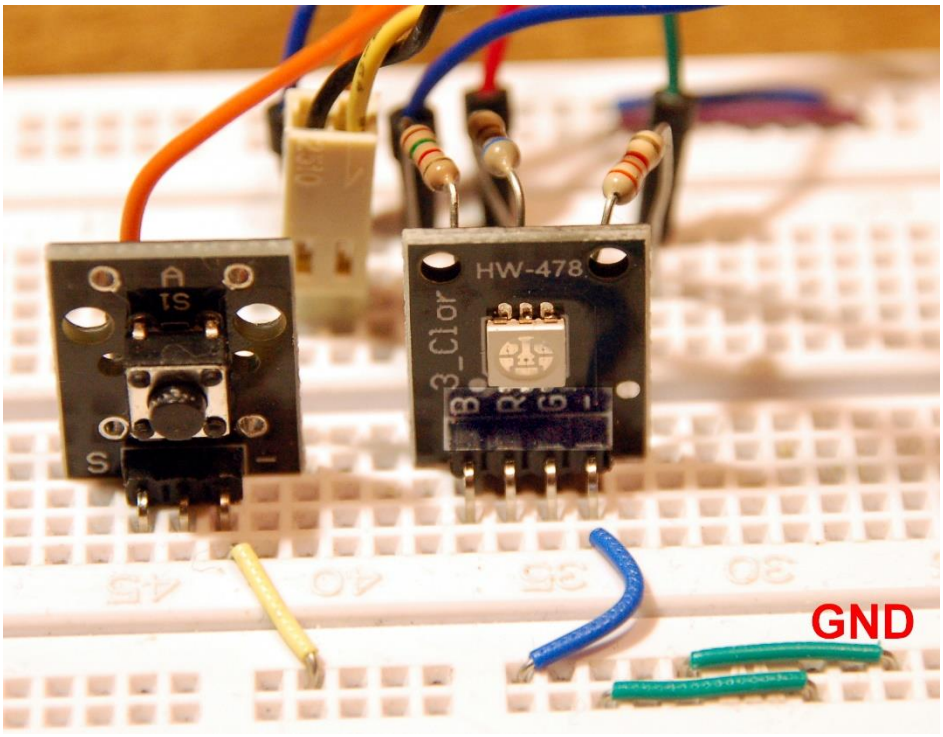

Abbildung 3: Taste und LED

Vom DF-Player mini brauchen wir 6 Anschlüsse, 5V, GND, zweimal Speaker, RS232-Eingang RX und BUSY. Alle Ein- und Ausgänge laufen am ESP8266 D1 mini zusammen.

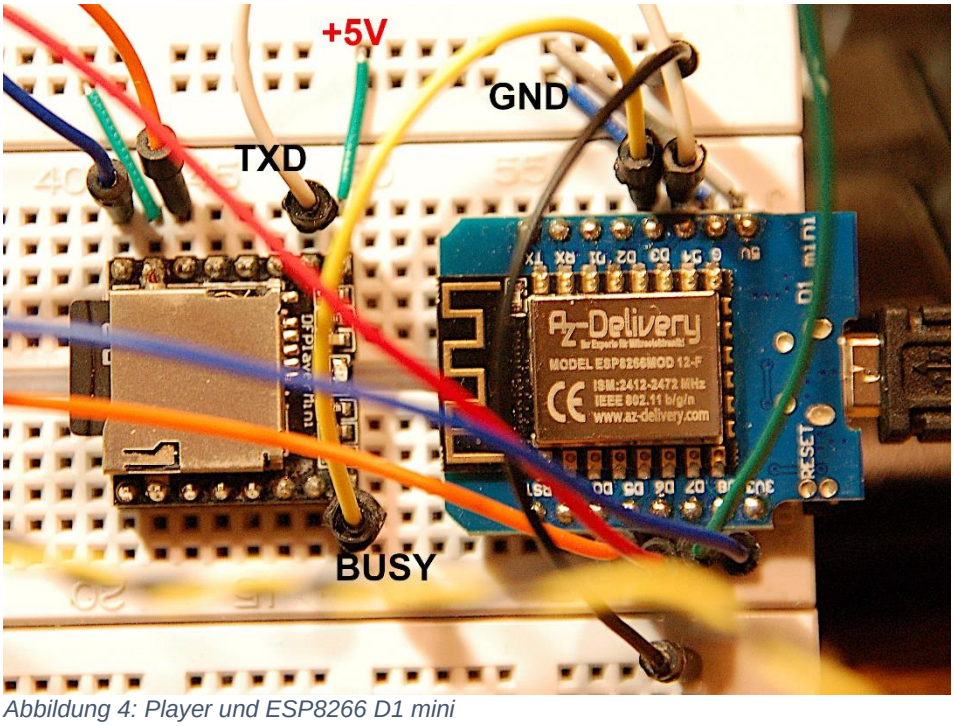

Abbildung 4: Player und ESP8266 D1 mini

Während der Testphase wird der gesamte Aufbau über das USB-Kabel vom PC versorgt. Im Produktionssystem muss eine eigene 5V-Versorgung verwendet werden. Bei herkömmlichen Haustürklingeln kann auch der Klingeltrafo zusammen mit einem entsprechenden Netzteil (Gleichrichter, Siebung und 5V-Regler) die Schaltung mit Energie füttern. Eine schlechte Nachricht gibt es allerdings auch. **Bei Signalanlagen mit Gegensprecheinheiten, die über ein Bussystem arbeiten, wie die BTicino, ist unser Türgong leider nicht so ohne weiteres einsetzbar**. Immerhin wäre aber ein Einsatz unter Verwendung des Klingelknopfs an der Wohnungstür denkbar.

Damit auch die Ohren etwas von unserem Türgong haben, müssen wir dem Aufbau natürlich noch einen kleinen Lautsprecher spendieren. Ich habe meinen in ein Gehäuse eingebaut, damit auch die Bässe besser rüberkommen. Die Leistung des Speakers sollte zwischen 3 und 5 Watt liegen, sein Widerstand zwischen 4 und 8 Ohm. Für die Tests benutze ich den Speaker eines kleinen Musikman-Würfels, dessen sonstige defekte Innerei entfernt wurde.

Abbildung 5: Türgong in der Entwicklung

Im Hintergrund befindet sich das 7"-Display meines Raspberry Pi auf dem der Broker läuft. Allerdings bediene ich diesen meistens über SSH-Terminals vom PC aus. Wie das funktioniert, habe ich im Teil 1 dieser Folge detailliert beschrieben. Kommen wir jetzt zur Software.

Das Programm für den Türgong

Das Programm **doorbell.py** deckt die zwei Standbeine der Anwendung ab, den Betrieb des DF-Players und die MQTT-Anbindung an den Mosquitto-Broker, der auf dem Raspberry Pi läuft und vom ESP8266 über WLAN kontaktiert wird.

Als Broker wird der Serverprozess unter MQTT bezeichnet. Das MQTT-Protokoll setzt wie HTTP auf dem TCP/IP-Protokoll auf, stellt also eine gesicherte Verbindung mit Handshake bereit. Clients können Nachrichten an den Broker senden (publishen) und beziehen (subscriben). Zu einer Nachricht gehört ein Thema, das als Topic bezeichnet wird und eine Nutzlast, genannt Payload. Unsere Einheit wird die Topics haus/doorbell/folder, haus/doorbell/title und haus/doorbell/volume abonnieren, also subscriben und unter den Topics haus/doorbell/ringed und haus/doorbell/title/new Nachrichten versenden. Topics haben eine ähnliche Baumstruktur wie Verzeichnisse unter Windows oder Linux.

Im DF-Player befindet sich eine Mini-SD-Karte mit 4 GB, eine mit weniger Speicherumfang ist kaum mehr zu haben. Die 32 Sound-Dateien mit verschiedenen Geräuschen sind in 4 Ordnern zu je 8 Files untergebracht. Die Dateien müssen per Vorgabe die Namen 000.mp3 bis 999.mp3 tragen, die Ordner werden mit 00 bis 99 bezeichnet. Normale Filenamen sind nicht zulässig. Die Sounds habe ich teilweise aus dem Internet gesammelt. Bei verschiedenen Datenbanken kann man die Sounds ohne Gebühren herunterladen, sofern man sie nicht für kommerzielle Zwecke einsetzt. Zwei Links dazu befinden sich im Programm doorbell.py. Andere Files habe ich selber aufgenommen und mit dem freien Tool Audacity bearbeitet. Die Spieldauer liegt zwischen 2 bis 8 Sekunden.

```
import os,sys       # System- und Dateianweisungen
from machine import Pin, reset, I2C
from time import sleep, ticks_ms
import network
from umqttsimple import MQTTClient
import esp
esp.osdebug(None)
import gc
gc.collect()
from dfplayer import DFPlayer
```

Das Importgeschäft bindet eine Reihe von Modulen ein. Zwei davon sind externe. Die Dateien dfplayer.py und umqttsimple.py müssen vor dem ersten Programmstart zum ESP8266 hochgeladen werden. Das geschieht mit Thonny. Die restlichen Module sind bereits im Kern von MicroPython enthalten.

```
bellButton=Pin(14, Pin.IN, Pin.PULL_UP)
statusLed=Pin(15,Pin.OUT,value=0) # blau=2
onairLed=Pin(13,Pin.OUT,value=0)  # gruen=1
errorLed=Pin(12,Pin.OUT,value=0)  # rot=0
led=[errorLed,onairLed,statusLed ]
red,green,blue=0,1,2
```

Hier werden die Anschlüsse für den Taster und die LEDs deklariert. Die Liste **led** erlaubt den Zugriff auf die LEDs über Zahlen. Zur Indizierung können auch die Variablen **red**, **green** und **blue** eingesetzt werden.

Die am Arduino orientierten Anschlussnamen auf dem ESP8266 D1 mini müssen für MicroPython in die Bezeichnungen von Espressif übersetzt werden. Zur Information kann die Tabelle dienen.

```
# Pintranslator fuer ESP8266-Boards
# Arduino-Pins D0 D1 D2 D3 D4 D5 D6 D7 D8
# ESP8266 Pins 16  5  4  0  2 14 12 13 15
#                SC SD
```

```
df=DFPlayer(busyPinNbr=0, volume=15)

topicBell=    "haus/doorbell/ringed"
topicFolder=  "haus/doorbell/folder"
topicTitle=   "haus/doorbell/title"
topicVolume=  "haus/doorbell/volume"
topicNewTitle="haus/doorbell/title/new"
topicNewFolder="haus/doorbell/folder/new"
folders=[
         "Gong",
         "Stimmen",
         "Sprache",
         "Musik",
        ]
folder=0
numOfTitles=8
title=0
vol=20
b=0
```

Wir erzeugen ein DFPlayer-Objekt und legen Variablen für die Topics an. Die Liste Folders legt lesbare Namen für die Soundordner vor. Wir starten mit Folder 0, legen fest, dass es jeweils 8 Titel pro Folder gibt (0…7) und dass wir mit Titel 0 starten. 20 ist der Wert für die Lautstärke während der Tests, im Produktivsystem sollte hier 70 bis 85 stehen. Wählen Sie den Wert so, wie es Ihrem Lautstärkeempfinden am besten entspricht.

```
# ------------------Funktionen deklarieren --------------
def blink(pulse,wait,col,inverted=False):
    pass
    if inverted:
        led[col].off()
        sleep(pulse)
        led[col].on()
        sleep(wait)
    else:
        led[col].on()
        sleep(pulse)
        led[col].off()
```

```
        sleep(wait)

def TimeOut(t):
    start=ticks_ms()
    def compare():
        return int(ticks_ms()-start) >= t
    return compare
```

Wir definieren ein paar Funktionen. **blink**() lässt eine LED, deren Farbe in **col** als Zahl 0…2 übergeben wird, für **pulse** Sekunden aufblitzen. Danach bleibt die LED für **wait** Sekunden aus. Danach wird das Programm fortgesetzt.

TimeOut() erzeugt und startet einen Software-Timer, der den Programmablauf nicht blockiert, anders als **sleep**(). Es wird die Referenz auf eine Funktion zurückgegeben. Diese Funktion gibt ihrerseits den Wert True zurück wenn die Zeit abgelaufen ist, deren Dauer in Millisekunden **TimeOut**() übergeben wurde.

Beispiel:

```
>>> abgelaufen=TimeOut(10000)

>>> abgelaufen()
```
Ausgabe: `False`

```
>>> abgelaufen()
```
Ausgabe: `False`

```
>>> abgelaufen()
```
Ausgabe: `True`

liefert 10000 ms später den Wert True zurück, vorher False. **TimeOut**() ist eine sogenannte **Closure**. Auch in anderen Programmiersprachen, zum Beispiel Node-MCU LUA, gibt es derartige Strukturen. Mehr dazu können Sie in diesem PDF-Dokument erfahren.

```
def hexMac(byteMac):
    """
    Die Funktion hexMAC nimmt die MAC-Adresse im Bytecode
    entgegen und bildet daraus einen String fuer die Rueckgabe
    """
    macString =""
    for i in range(0,len(byteMac)):
        macString += hex(byteMac[i])[2:]
        if i <len(byteMac)-1 :
            macString +="-"
    return macString
```

hexMac() gibt die MAC-Adresse der WLAN-Schnittstelle STA zurück. Die werden wir brauchen, wenn unser MQTT-System am WLAN-Router angemeldet wird. Ohne diese Angabe wird uns der Router nicht ins Netz lassen.

```
def connect2Broker():
    # Client-Instanz erzeugen
    client = MQTTClient(myID, myMQTTserver, user="doorbell",\
                        password="5t6z7u")
    client.set_callback(messageArrived)
    client.connect()
    client.subscribe(topicFolder)
    client.subscribe(topicTitle)
    client.subscribe(topicVolume)
    print("Connected to:", myMQTTserver)
    print("Subscribed to:",topicFolder,topicTitle,topicVolume)
    return client
```

Wir kommen langsam zum Kern der Anwendung. **connect2Broker**() macht genau das, was der Name sagt. wir erzeugen eine MQTTClient-Instanz. Die IP von unserem Raspberri Pi ist die Broker-IP, unser Client hat den Namen **doorbell** und das Passwort **5t6z7u**. Name und Passwort sind frei wählbar, müssen aber dem Broker bekanntgemacht werden. Das tun wir später. Dann sagen wir dem Clientobjekt, welche Funktion es aufrufen muss, wenn eine Nachricht eintrudelt. Die Callback-funktion heißt **messageArrived**. Wir schleusen auf diesem Weg eine Referenz auf diese Funktion in die ansonsten abgeschlossene Klasse MQTTClient ein. In der Klasse gibt es eine Variable self.cb, der diese Referenz zugewiesen wird. Wird self.cb() beim Empfang von Zeichen als Methode aufgerufen, dann läuft genau das ab, was wir in der Funktion **messageArrived**() deklariert haben. Mit diesem Trick erweitern wir die Funktionalität der Klasse MQTTClient selbst nach Belieben, ohne in den Programmcode der Klasse eingreifen zu müssen.

Das Objekt **client** verbindet sich mit dem Broker und abonniert die Topics, die in den drei Variablen **topicFolder**, **topicTitle** und **topicVolume** hinterlegt sind. Wir werden über den Verlauf informiert, und das client-Objekt wird von der Funktion zurückgegeben.

```
def restartClient():
    print("connection to {} failed - Rebooting".\
          format(myMQTTserver))
    sleep(10)
    reset()
```

Bei Verlust der Verbindung zum Broker wird eine Exception geworfen und infolge **restartClient**() aufgerufen. Wir erhalten eine Meldung im Terminalbereich von Thonny und nach 10 Sekunden wird das System neu gestartet.

```
def messageArrived(topic,msg):
    global folder, title, vol
    topic=topic.decode()
    msg=msg.decode()
    if topic == topicFolder:
        try:
            folder=int(msg)
```

```
        except:
            folder=0
        print("Folder: {}. {}".format(msg, folders[folder]))
        client.publish(topicNewFolder, msg+folders[folder])
    if topic == topicTitle:
        try:
            title = int(msg)
        except:
            pass
        print("Titel:",title)
        client.publish(topicNewTitle, str(title))
    if topic == topicVolume:
        try:
            vol = int(msg)
        except:
            vol = 85
        print("Volume:",vol)
        df.volume(vol)
```

Die Funktion **messageArrived**() wird ausgeführt, wenn eine Nachricht eingetroffen ist. **folder**, **title** und **vol** sind globale Variablen, die außerhalb von messageArrived() gebraucht werden. daher sind sie als global ausgewiesen. Das übergebene Topic und die Nutzlast müssen vom Bytes-Typ in einen String übersetzt werden, das macht die Methode **decode**(). Nun prüfen wir, zu welchem Topic eine Nachricht vorliegt und leiten die jeweilige Aktion ein. Wir setzen die übermittelte Foldernummer und geben Nummer und Name zurück, die unter dem Topic in **topicNewFolder** abgelegt ist. Oder wir setzen den gewünschten Titel. Auch dessen Nummer wird wieder veröffentlicht. Schließlich kann auch die Lautstärke auf einen neuen Wert gesetzt werden. Falls Fehler bei den Aktionen auftreten, wird der Wert in den except-Zweigen gesetzt.

```
def buttonPressed(pin):
    global pressed
    bellButton.irq(handler=None)
    pressed = True
```

Die Funktion **buttonPressed**() wird aufgerufen, wenn der Eingang GPIO14 von 1 (3,3V durch Pullup-Widerstand) auf 0 wechselt. Das wird in der Regel durch Drücken des Klingelknopfs geschehen. Die Auslösung des Aufrufs passiert durch einen sogenannten Interrupt (IRQ), eine Unterbrechung des Hauptprogramms. Damit das nicht sofort wieder passiert, falls die Taste prellt (was jede mechanische Taste tut), wird der IRQ zunächst ausgeschaltet und das Flag **pressed** auf True gesetzt. In der Hauptschleife fragen wir den Zustand ab und testen gleichzeitig, ob die Taste immer noch gedrückt ist. Auf diese Weise können wir feststellen, dass nicht nur ein kurzer Störimpuls auf der Eingangsleitung der Auslöser des IRQ war, sondern ein echter Tastendruck.

```
myMQTTserver = "10.0.1.99"
mySSID = 'Here_goes_your_SSID'
myPass = 'Here_goes_your_Password'

# Unbedingt das AP-Interface ausschalten
nac=network.WLAN(network.AP_IF)
nac.active(False)
nac=None
```

Hier werden die IP-Adresse des Systems, auf dem der Broker läuft und die Credentials für den Netzzugang eingetragen. Ersetzen Sie bitte die Platzhalter durch die Daten, die Ihr WLAN-Router vorgibt. Die nachfolgenden drei Zeilen eliminieren den Fehler, den das Accesspoint-Interface im ESP8266 verursacht und der zu ständigen Neustarts des Controllers führt. Das AP-Interface (Accesspoint) muss deaktiviert werden, bevor das STA-Interface (Station) gestartet wird. Eine weitere Fehlerquelle ist beim ESP8266 die von Haus aus aktivierte webREPL-Funktion. Zum Deaktivieren muss nach jedem Mal neu Flashen Folgendes geschehen.

Eingabe auf der Kommandozeile:
>>> *import webrepl_setup*
WebREPL daemon auto-start status: disabled

Would you like to (E)nable or (D)isable it running on boot?
(Empty line to quit)
> *d*
No further action required

Die nächsten 30 Zeilen schaffen den Zugang zum WLAN und sind im Programm ausreichend durch Kommentare erklärt, sodass ich sie hier übergehe.

Bevor das Programm in die Hauptschleife eintritt, versucht es, die Verbindung zum Broker herzustellen. Die Variable **client** übernimmt das MQTT-Objekt, welches von der Funktion **connect2Broker**() erzeugt und zurückgegeben wird. Gelingt das aus irgendeinem Grund nicht, erfolgt nach 10 Sekunden ein Neustart.

```
try:
    client = connect2Broker()
except OSError as e:
    restartClient()

pressed=False
bellButton.irq(handler=buttonPressed, trigger=Pin.IRQ_FALLING)
df.volume(vol)
```

Wir setzen das Flag **pressed** auf False und schalten den IRQ scharf. Wir übergeben eine Referenz auf die ISR, der Auslöser soll die fallende Flanke an GPIO14 sein. Die Lautstärke stellen wir auf die anfangs vorgegebenen 20%, den Wert in der Variablen **vol**.

```
while True:
    try:
        client.check_msg()
        ring=bellButton.value()
        if pressed and ring == 0:
            nachricht=folders[folder]+";"+str(title)
            client.publish(topicBell, nachricht)
            df.play(folder,title)
            title=(title+1) % (numOfTitles)
            print("naechster Titel",title)
            client.publish(topicNewTitle, str(title))
            bellButton.irq(handler=buttonPressed, \
                      trigger=Pin.IRQ_FALLING)
            pressed=False
    except OSError as e:
        restartClient()
    b+=1
    if b>= 5000:
        blink(0.05,0.05,green)
        b=0
```

In der Hauptschleife prüfen wir als Erstes, ob eine Nachricht vom Broker eingetroffen ist. Die Funktion **messageArrived**() sorgt dann automatisch für deren Abarbeitung. Hat jemand den Klingelknopf gedrückt, dann ergeben **ring** und **pressed** zusammen **True**, der Block der if-Struktur wird durchlaufen.

Wir bereiten die Payload vor, die wir in der folgenden Zeile unter dem Topic in der Variablen **topicBell** (haus/doorbell/ringed) publishen. Dann stoßen wir das Abspielen des Titels an, der durch die Variablen **folder** und **title** ausgewählt ist. Wir erhöhen die Titelnummer und publishen diese als String unter dem Topic in **topicNewTitle**. Danach schalten wir den IRQ erneut scharf, und setzen **pressed** auf **False**. Sollte während der Ausführung des try-Blocks ein Fehler aufgetreten sein, führt das zum Neustart des Systems in 10 Sekunden. Der Zähler **b** dient der Erzeugung des Heartbeats. Nach 5000 Schleifendurchläufen blitzt die grüne LED für 50ms auf und der Zähler wird wieder auf 0 gesetzt. Die Dunkelzeit beträgt etwa 3 Sekunden. Ein Schleifendurchlauf dauert somit ca. 600µs.

Einbindung ins MQTT-System

Bisher wird durch das MQTT-System (Message Queuing Telemetry Transport) im Lagerkeller die Temperatur und Luftfeuchte erfasst und ein Ventilator geschaltet, außerdem werden in der Heizung Vor- und Rücklauftemperatur gemessen sowie Umwälzpumpe und Brenner geschaltet. Die MQTT-Anbindung des Türgongs selbst habe ich bereits beschrieben, jetzt folgt die Einbindung am Broker und die Behandlung durch Node-RED. Mit MQTT wird die komfortable Bedienung des Systems über einen Browser auf einfache Weise durch ein Baukastensystem geschaffen. Außerdem loggen wir die Klingelaktionen mit Datum und Uhrzeit in einer Datei.

Beginnen wir mit dem Mosquitto-Broker. Der wird durch drei Dateien konfiguriert, zwei davon müssen wir durch Einträge ergänzen.

Bei der Einführung der Benutzerauthentifizierung hatten wir in Folge 5 im Verzeichnis **/etc/mosquitto/conf.d** eine Datei mit Name-Passwort-Paaren angelegt (**mosquitto_users.txt**), gesichert (**mosquitto_users.txt.sic**) und dann die Passwörter in der Originaldatei verschlüsselt. Wir nutzen jetzt diese Klartext-Sicherung, um den neuen User **doorbell** anzulegen. Wir öffnen mit Putty ein Terminal von der Windowskiste auf den Raspi. Die meisten der folgenden Anweisungen müssen als **root** erledigt werden, daher das einleitende **sudo**, das den nachfolgenden Befehl unter Administratorrechten ausführt.

```
cd /etc/mosquitto/conf.d
```

```
sudo nano mosquitto_users.txt.sic
```

Der nano-Editor öffnet sich mit der Passwort-Datei

Abbildung 6: Ergänzung der User-Passwort-Datei

In der Abbildung ist der User **doorbell** bereits ergänzt. Bitte beachten Sie, dass in der **letzten Zeile kein Zeilenvorschub** erfolgen darf. Die Datei wird mit Strg+O abgespeichert, der Editor mit Strg+X geschlossen.

Die geänderte Sicherung kopieren wir jetzt auf die Arbeitsdatei mosquitto_users.txt.

```
sudo cp -a mosquitto_users.txt.sic mosquitto_users.txt
```

Dann lassen wir uns die Passwörter erneut verschlüsseln.

```
sudo mosquitto_passwd -U mosquitto_users.txt
```

Die zweite zu ändernde Datei ist **acl_liste.txt**.

```
sudo nano acl_liste.txt
```

Die Zeilen im roten Rahmen müssen ergänzt werden, damit der User doorbell seine programmierten Aktionen ausführen darf.

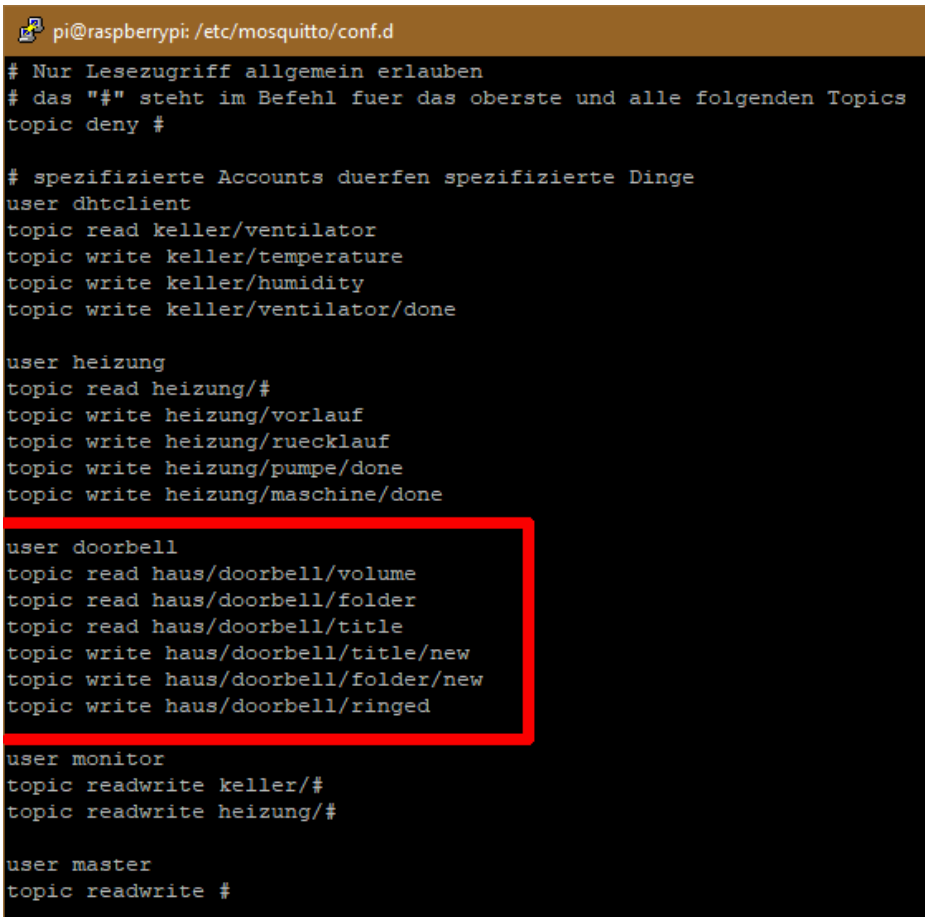

Abbildung 7: Die neue ACL-Liste

Wir speichern wieder ab Strg+O und schließen den Editor Strg+X. Damit unsere Änderungen greifen, müssen wir mosquitto neu starten, damit der Broker die neue Konfiguration einliest.

```
sudo service mosquitto restart
```

Kopieren Sie jetzt bitte den Inhalt der Datei doorbell.py in ein leeres Editorfenster, speichern Sie es unter **boot.py** ab und laden Sie diese Datei zum ESP8266 hoch. Steckt die SD-Karte mit den Soundfragmenten im Slot des DF-Players? Dann sollte nach einem Reset der Controller automatisch unser Programm starten. Die blaue LED blinkt lang-kurz bis die Verbindung zum WLAN-Router steht. Danach sollte die grüne LED etwa im 3-Sekunden-Rhythmus aufblitzen. Im Moment hängt der ESP8266 ja noch am USB-Anschluss. Meldungen des Controllers sollten daher im Terminal von Thonny erscheinen.

Ausgabe:

Player-Constructor
 UART(1, baudrate=9600, bits=8, parity=None, stop=1, rxbuf=15, timeout=0, timeout_char=2)
Client-ID **ec-fa-bc-c5-31-69**
#5 ets_task(4020ee60, 28, 3fff92d0, 10)
1.1.1.
connected: True

Verbindungsstatus: STAT_GOT_IP
STA-IP: 10.0.1.94

STA-NETMASK: 255.255.255.0
STA-GATEWAY: 10.0.1.20
ADDR ('10.0.1.99', 1883)
Connected to: 10.0.1.99
Subscribed to: haus/doorbell/folder haus/doorbell/title haus/doorbell/volume

Die markierte Zeile enthält die Client-ID, welche beim Herstellen der Verbindung zum Brocker übermittelt wird. In unserem Fall ist das die MAC-Adresse des STA-Interfaces. Genau diese sechs Hexadezimalwerte müssen im Router im Bereich WLAN-Sicherheit eingetragen werden, damit der ESP8266 Zugang zum Funknetz erhält. Ziehen Sie bitte dafür das Handbuch Ihres Routers zu Rate.

Es ist Zeit für einen ersten Test. Wir öffnen ein zweites Terminal zum Raspi und subscriben mit **-t "haus/#"** alle Topics unterhalb **haus**. Dann drücken wir, nicht zu kurz, den Klingelknopf.

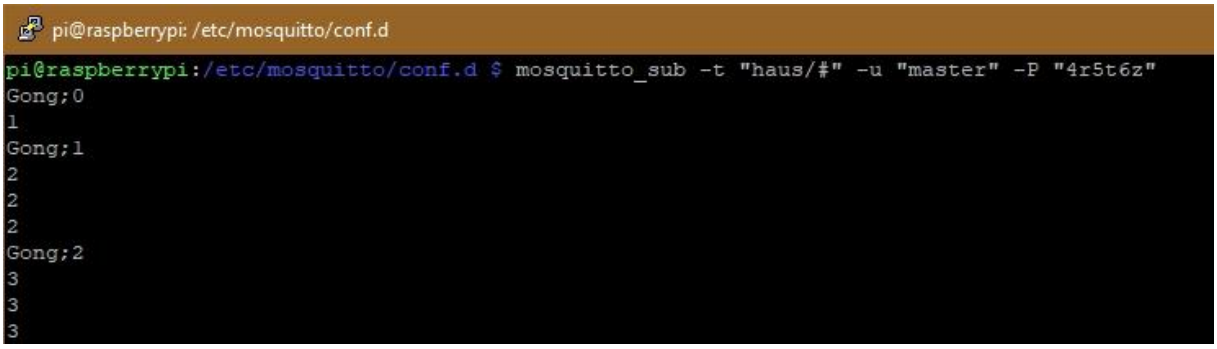

Abbildung 8: Die erste Antwort vom Türgong

Neben dem Klang aus dem Lautsprecher sollten im Terminal Ausgaben, ähnlich wie in Abbildung 7 erscheinen.

Im zweiten Terminal veröffentlichen wir folgende Nachrichten.

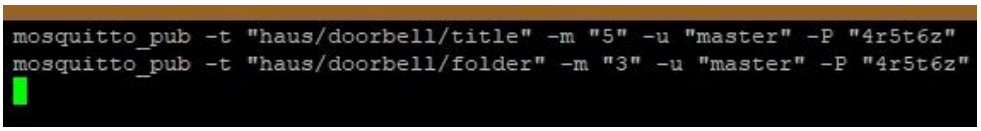

Abbildung 9: Aufträge an doorbell

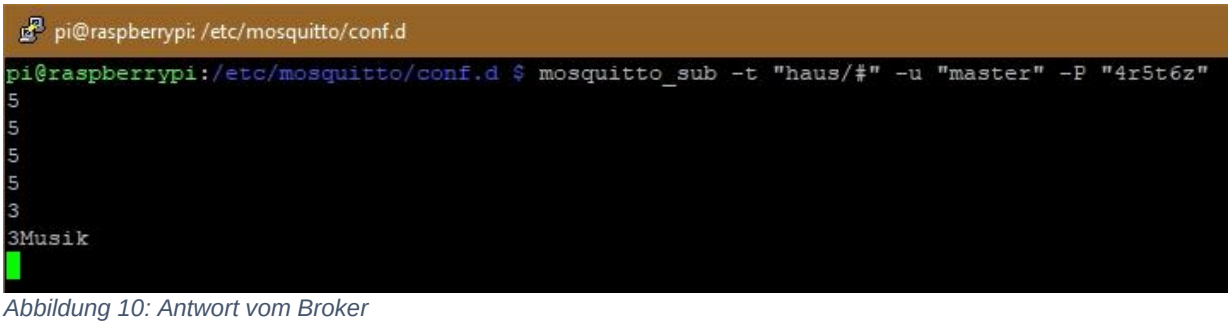

Abbildung 10: Antwort vom Broker

Funktioniert? Ausgezeichnet, dann erklimmen wir die dritte Stufe, lassen Sie uns das Ganze in Node-RED in einem neuen Tab platzieren.

Wir müssen einen Browser öffnen und als Erstes Node-RED auf unserem Raspi aufrufen, URL: 10.0.1.99:1880. Die json-Datei mit dem Flow aus der Folge 5 finden Sie über den Link. In Episode 4 und 5 ist ausführlich gezeigt, wie ein Flow angelegt wird. Außerdem erfahren Sie dort grundlegende Dinge über den Umgang mit den Elementen von Node-RED und den Fensteraufbau.

Um den Inhalt der json-Datei in Node-RED zu importieren, führen Sie bitte folgende Schritte durch.

- Laden Sie die json-Datei herunter, klicken Sie irgendwo in den Text und markieren und kopieren Sie den Inhalt des Browserfensters in die Zwischenablage (Strg+A, Strg+C).
- Öffnen Sie im Node-RED-Fenster das **Menü** (drei waagrechte Balken neben **deploy**)
- Wählen Sie **Import.**
- Kopieren Sie die Zwischenablage in das rosa Fenster.
- Klicken Sie **Aktueller Flow** und dann **Import**.
- Entfernen Sie alle weiteren Flows aus dem Arbeitsbereich.

Ergänzen sie jetzt die Nodes, die in Abbildung 10 dargestellt sind aus den jeweiligen Karten. Wir gehen jetzt einzeln auf jeden Typ und dessen Konfiguration ein. **Ich empfehle aber dringend, die Folgen 4 und 5 dieser Reihe durchzuarbeiten, wenn Sie bisher noch keine Erfahrung mit Node-RED sammeln konnten**. Von den Nodes sind einige aus der Palette **Dashboard**. Diese ist nicht im Standardumfang von Node-RED, sondern muss nachgerüstet werden. Eine Schritt-für-Schritt-Anleitung dazu finden Sie in der Blogfolge 4 ab Seite 7.

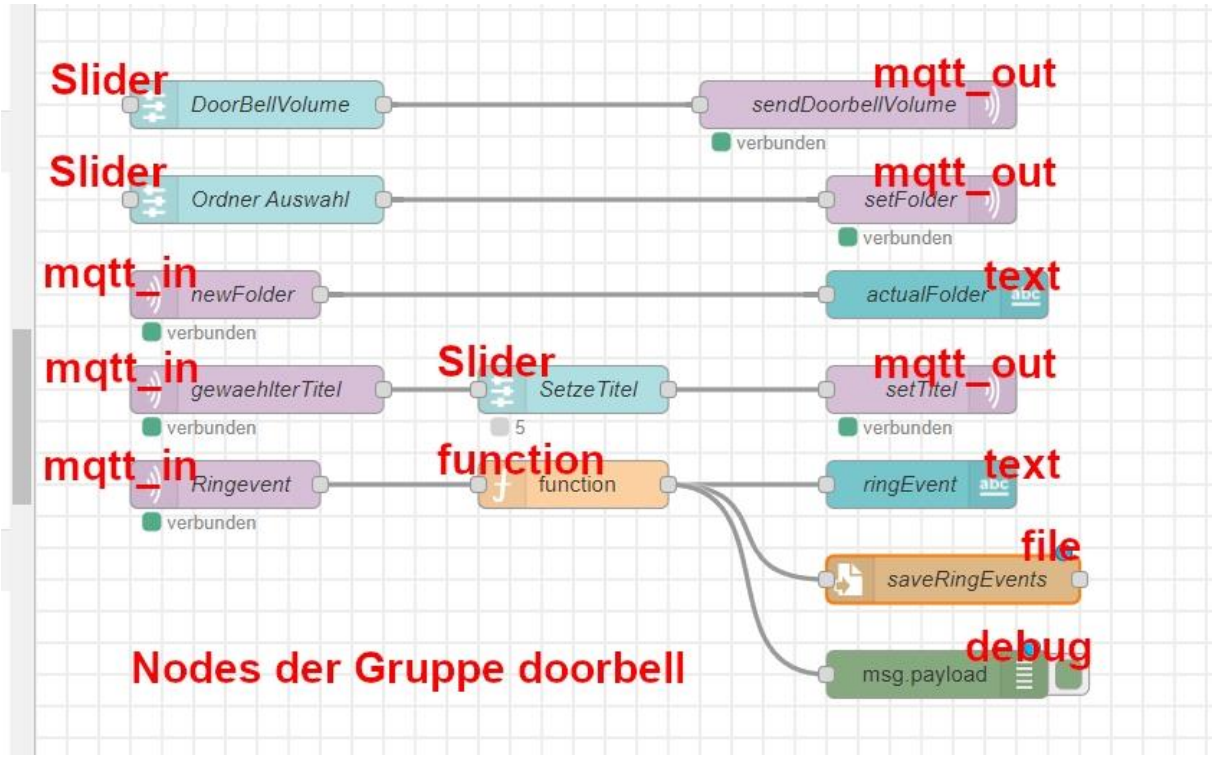

Abbildung 11: Die Gruppe doorbell

Öffnen Sie nun in der rechten Seitenspalte die Dashboard-Ansicht.

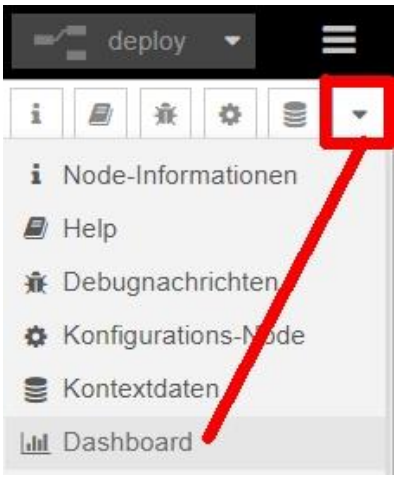

Abbildung 12: Dashboardansicht öffnen

Dort erzeugen wir einen neuen Tab,

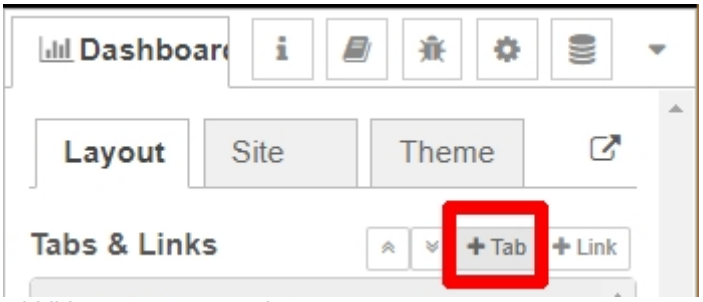

Abbildung 13: Neuen Tab erzeugen

den wir auch gleich bearbeiten.

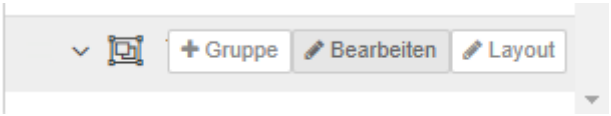

Abbildung 14: Neuen Tab bearbeiten

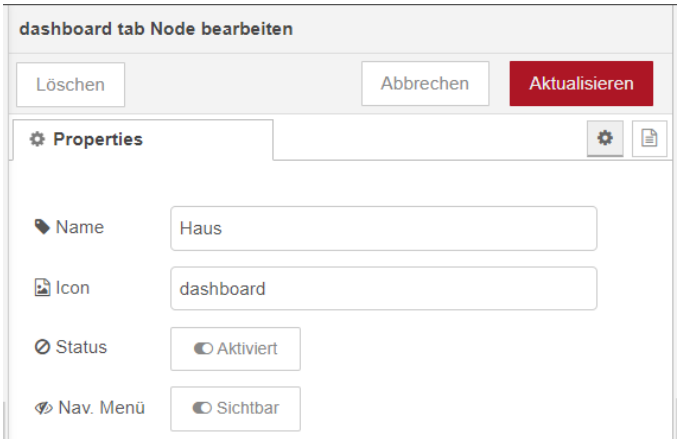

Abbildung 15: tab Node bearbeiten

Aktualisieren und im neuen Tab eine neue Gruppe anlegen.

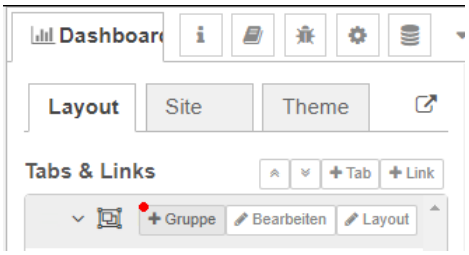

Abbildung 16: Neue Gruppe anlegen

Die Gruppe bearbeiten wir, danach **Aktualisieren.**

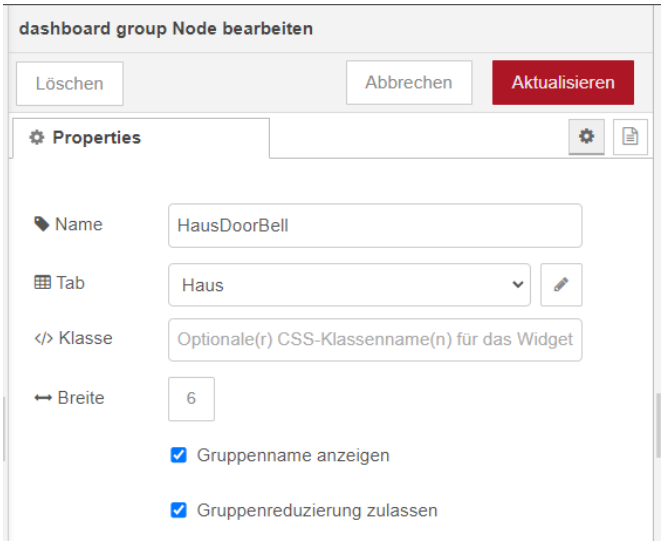

Abbildung 17: group Node bearbeiten

Jetzt werden die neuen Nodes aus der Palette hereingezogen. Wir beginnen mit den Slidern (Schieberegler). Doppelklick auf den ersten und hier ist die Konfiguration für die Lautstärke.

slider am Beispiel doorBellVolume

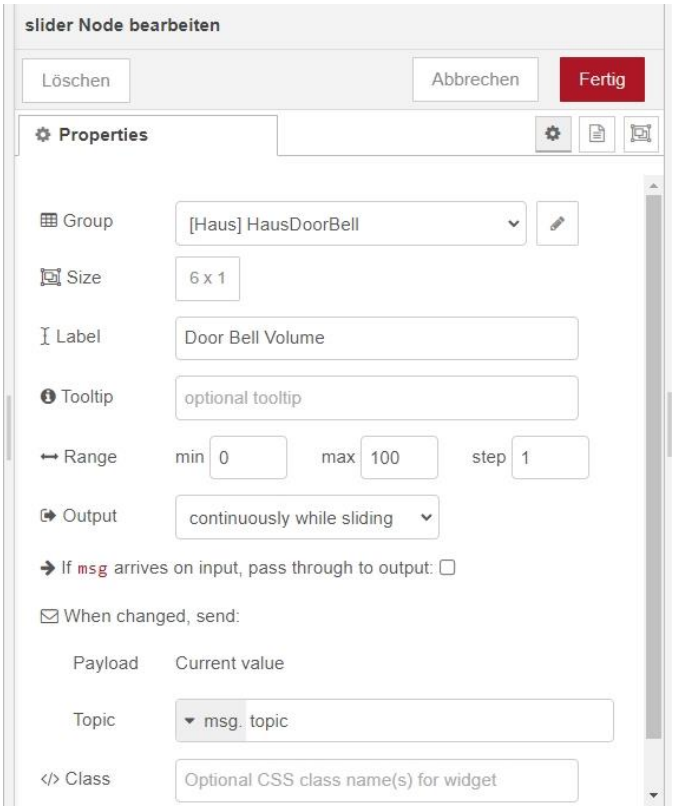

Abbildung 18: slider-Properties

Für **OrderAuswahl** und **Setze Titel** ist die Konfiguration ähnlich, die Maximalwerte sind 3 und 7. Die Gruppenzuordnung ist für alle Dashboard-Nodes **HausDoorBell**, die Beschriftung wird der Bedeutung angepasst.

mqtt in am Beispiel newFolder

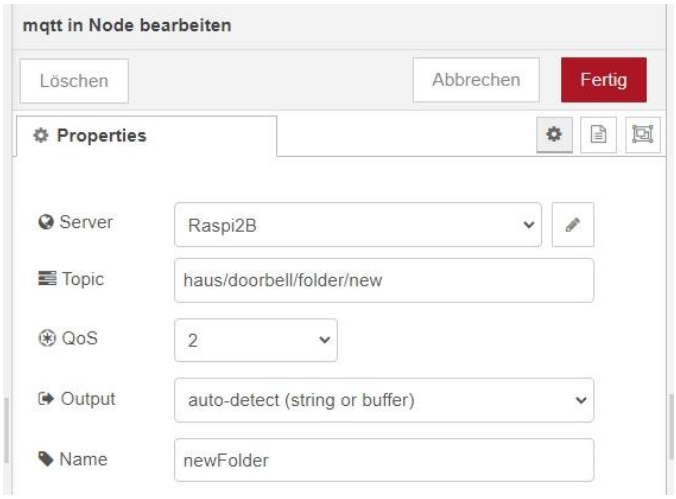

Abbildung 19: mqtt_in Node bearbeiten

Der Server ist für alle mqtt-Nodes Raspi2B, das Topic wird so eingetragen wie es in doorbell.py definiert wurde. Der Name wird "sprechend" ausgewählt (siehe Abbildung 11).

mqtt out am Beispiel sendDoorbellVolume

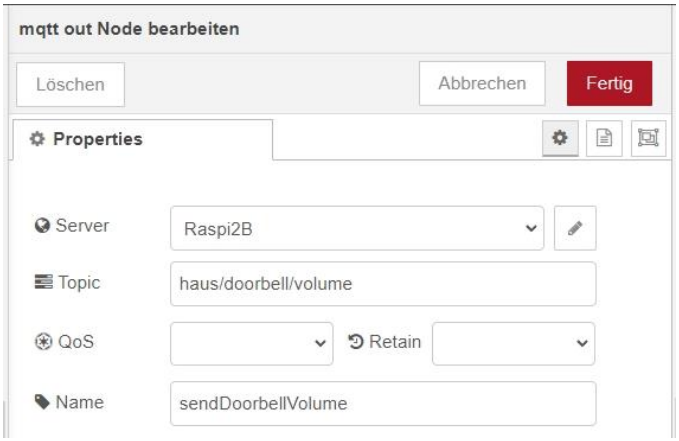

Abbildung 20: mqtt_out Node bearbeiten

text am Beispiel actualFolder

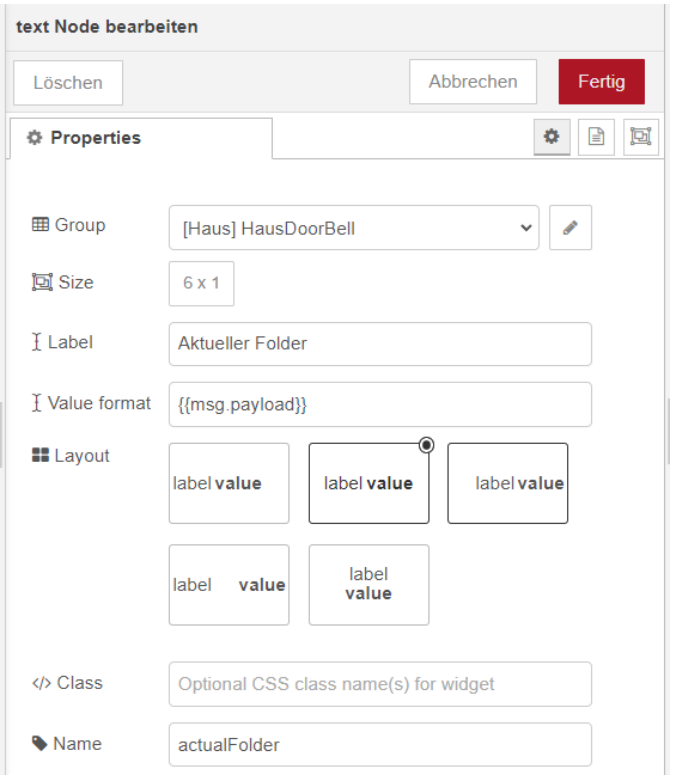

Abbildung 21: text Node bearbeiten

Label und Name werden der Aufgabe entsprechend vergeben.

function

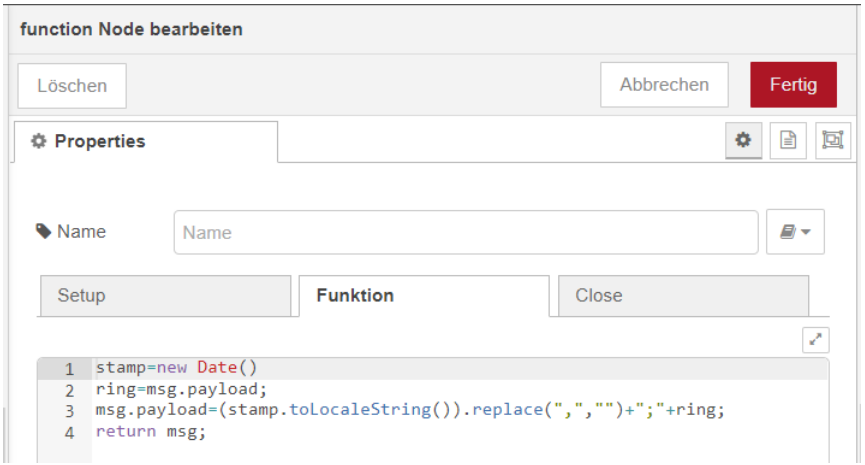

Abbildung 22: function Node bearbeiten

Unter dem Reiter **Funktion** wird der Körper einer Java-Script-Funktion eingegeben. Das Eingabeargument kommt vom user doorbell. Es enthält Folder und Titel, des gespielten Sounds. Als Erstes erzeugen wir einen Zeitstempel und sichern die bisherige Nutzlast der Nachricht vom mqtt_in-Node in der Variablen **ring**. Die Methode **toLocalestring**() stellt aus der Zeitangabe in Millisekunden seit dem 01.01.1970 einen Klartextstring zusammen. Das darin befindliche Komma ersetzen wir durch nichts. Wir addieren einen Strichpunkt und die gesicherte Payload der Eingangs-Nachricht. Alles zusammen weisen wir wieder der Nutzlast der Nachricht zu und geben diese zurück. Sie gelangt an den Ausgang des function-Nodes und wird an drei Ziele versandt. Den Typ Textanzeige kennen wir schon.

file

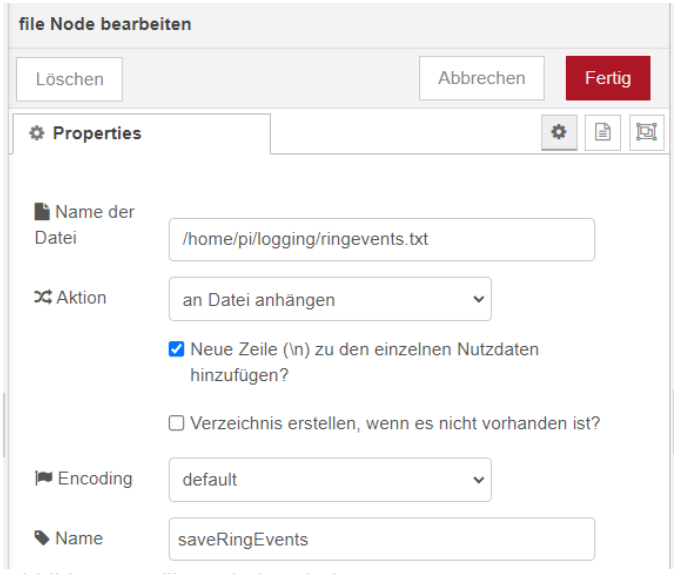

Abbildung 23: file node bearbeiten

Wir vergeben einen Filenamen, ich empfehle eine absolute Pfadangabe vom Rootverzeichnis aus. Weil es eine Logdatei werden soll, wählen wir als Aktion **anhängen**.

debug

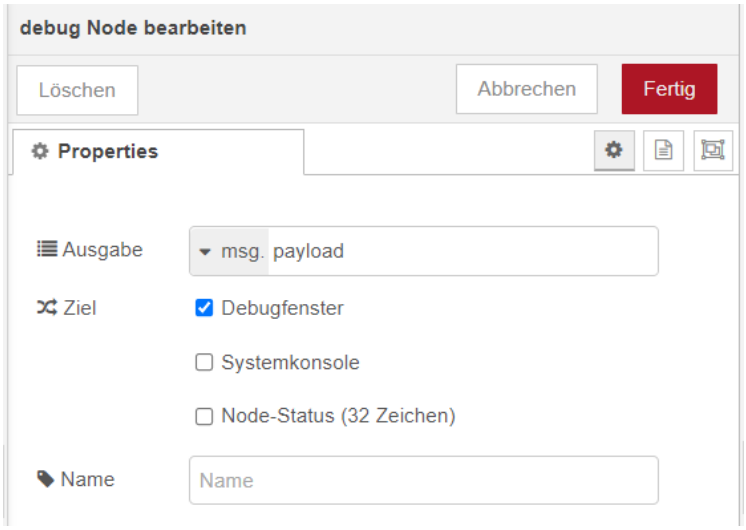

Abbildung 24: debug Node bearbeiten

Ein debug-Node gibt das Topic und die Payload der eingegangenen Nachricht im Debug-Fenster der rechten Seitenleiste aus. Eingeschaltet wird das über den Reiter mit der kleinen Wanze (Bug) oder den Tastencode Strg+g+d.

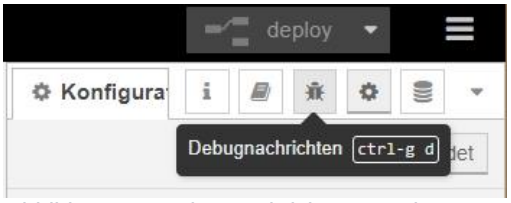

Abbildung 25: Debugnachrichten anzeigen

Wenn alles richtig eingestellt wurde, können wir den ersten Test starten. Die doorbell-Einheit läuft? Mosquitto wurde neu gestartet? Im Node-RED-Fenster wurde der deploy-Button geklickt? Das Debug-Fenster ist aktiviert? Dann drücken sie doch einmal den Klingelknopf.

So sollten die Meldungen in der Debug-Spalte aussehen.

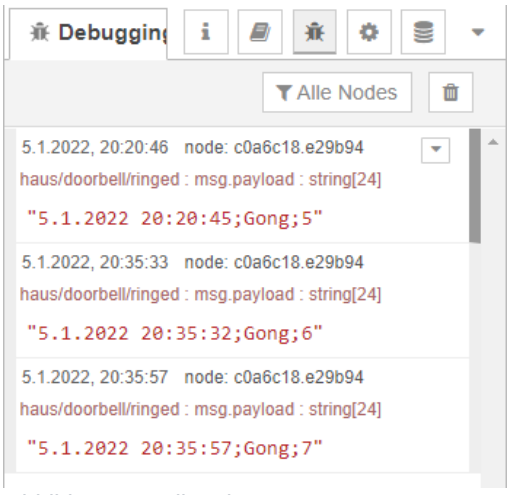

Abbildung 26: Klingel-Events

Falls der Versuch fehlschlägt, gibt es hier die json-Datei von meinem Lösungsvorschlag zum Download. Ich empfehle aber, damit Ihre Arbeit nicht für die Katz war, dass Sie Ihren Entwurf zuerst exportieren. Das geht wieder über das Menü, Auswahl: Exportieren.

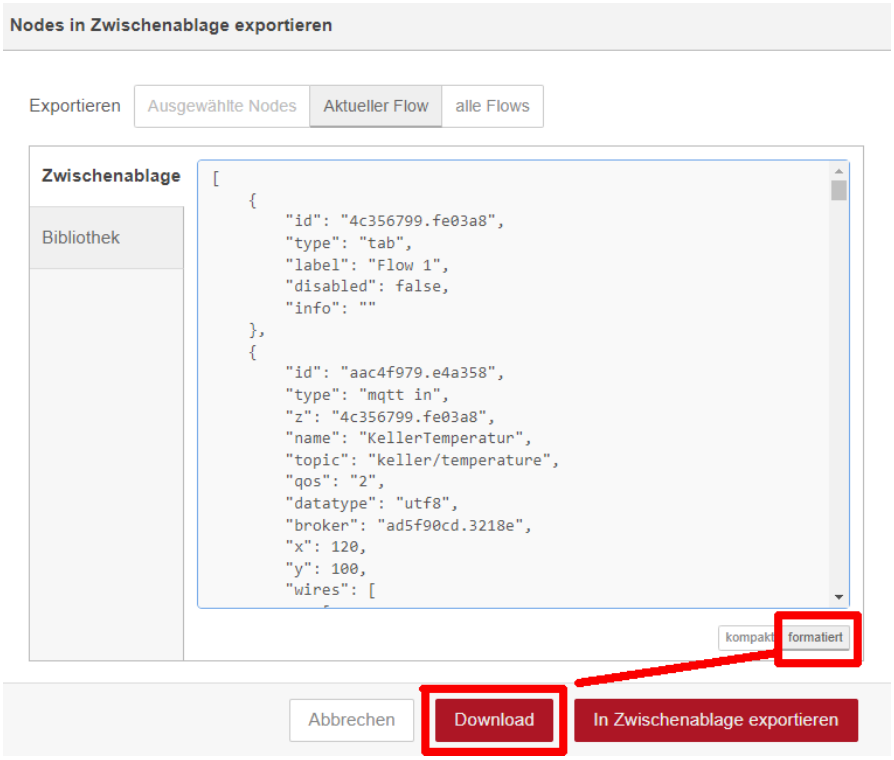

Abbildung 27: Flow exportieren

Entfernen Sie dann Ihren Flow (Menü – Flows – Löschen) und importieren Sie die heruntergeladene json-Datei. Wenn Sie jetzt ins Dashboard-Fenster schalten, die URL ist http://10.0.1.99:1880/ui/, dann sollten Sie im Tab Haus die folgende Darstellung finden.

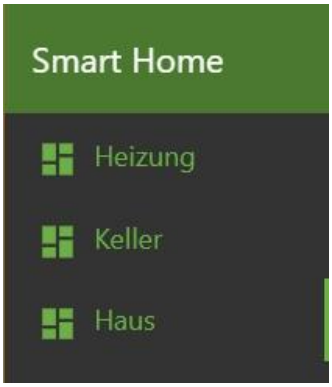

Abbildung 28: Die aktuelle Tabauswahl

Abbildung 29: doorbell - Dashboardansicht

Hier kann man die Lautstärke einstellen, den Themen-Folder vorgeben sowie den Titel. Weil die Titelnummer in unserem Programm **doorbell**.py automatisch nach jedem Läuten erhöht wird, und dieser Wert zurück an Node-RED gesandt wird, habe ich den Eingang vom Titel-slider mit dieser zurückgemeldeten Nummer gekoppelt. Dadurch wird die Marke des Sliders nach jedem Gong neu positioniert. Natürlich kann man die Fähigkeiten durch weitere Bedienelemente ausbauen: Ein Tasteranschluss für eine weitere Tür mit anderem Signal, Fernauslösung eines Signals, Stummschaltung manuell oder zeitgesteuert, automatische Nachtabschaltung durch einen LDR, zufallsgesteuerte Signalwahl, Folderweiterschaltung …Sie sehen, da ist noch viel Musik drin!

Zum Abschluss habe ich noch ein Bonbon für Sie. Damit man die beiden Kernmodule stabil in einem Gehäuse montieren kann, gibt es hier den Download zu einer PDF-Datei mit dem Layout einer Trägerplatine.

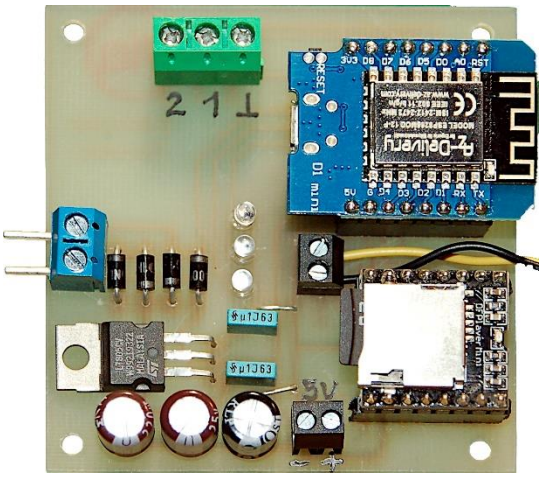

Abbildung 30: Bestückte Platine mit Netzteil 12V zu 5V

Damit sind wir vorerst am Ende der Reihe **Server und Clients unter MQTT in MicroPython auf dem Raspi und der ESP-Familie** angekommen. Ich hoffe, dass ich einen kleinen Eindruck von der Leistungsfähigkeit des Übertragungsprotokolls MQTT vermitteln und zeigen konnte, wie man ESP32 und Co. sowie den Raspberry Pi dazu in Dienst nehmen kann. Ähnlich wie unter UDP, kann man natürlich auch auf dem Raspi unter MQTT ohne Node-RED Clients für besondere Aufgaben entwickeln – selbstverständlich auf der Basis von Python, der Muttersprache des Pi. Wir haben in dieser Reihe nur ein bisschen an der Oberfläche gekratzt, aber ich würde mich freuen, wenn es Ihnen ebenso viel Spaß gemacht hat, wie mir. Hier finden Sie übrigens eine Zusammenfassung aller bisher veröffentlichten Blogfolgen zum Thema MicroPython mit dem ESP8266 und ESP32. Darin gibt es ganz einfache Beispiele zum allerersten Umgang mit MicroPython bis hin zu anspruchsvollen Projekten.

Kleine Vorschau auf das nächste Thema:

Arduino goes ESP via nRF24L01+
Ein MicroPython-Modul auf dem ESP32/ESP8266 ermöglicht die Verbindungsaufnahme mit der Arduino-Familie. Als kabelloses Bindeglied werden wir die Funkstrecke zwischen zwei nRF24L01-Modulen verwenden.