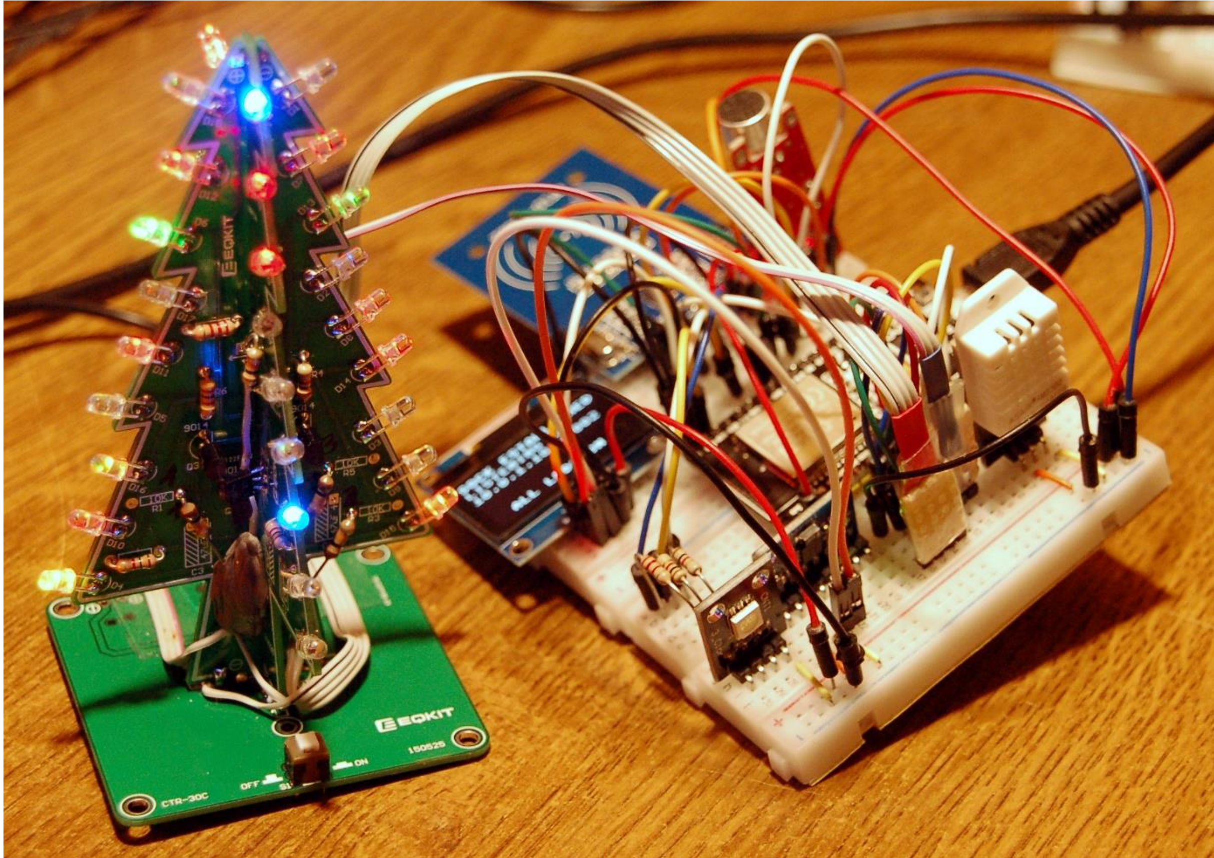

Abbildung 1: Die Baum-Geschichten

Diesen Beitrag gibt es auch als [PDF in Deutsch](#).
This episode is also availlable as [PDF-document in english](#).

Stories usually start with "Once upon a time". The following stories take place today and now. Today we're going to develop a few projects around a very special kind of Christmas tree.

What is meant is a tree approx. 11.5 cm high with 36 (37) colored lights on it. They flash in all colors merrily. But they more or less always do it in the same way. That gave me the idea to bring some order into the chaos with a few additional components. The result is a controller with an ESP32 and various sensors, which illuminate the LEDs for various measurement tasks or simply cause astonishment. In the following chapters, I will introduce you to part of what would be possible as a whole. All applications are of course programmed in MicroPython. So welcome to the

MicroPython stories about a small Christmas tree

Contents:

1. The parts list
2. The software
3. We build the tree and wire it up
4. Targeted illumination
5. The OLED display for plain text information
6. Graduated glow
7. The enchanted tree
8. Who is making such a noise?
9. On the trail of the sapling claw
10. ESP32 and DHT22 / DHT11 wish you a pleasant stay
11. The somewhat different Christmas raffle
12. The Christmas tree app

The part list

1	DIY LED Weihnachtsbaum Kit
1	KY-009 RGB LED SMD Modul Sensor oder KY-016 FZ0455 3-Farben RGB LED Modul 3 Color
1	1,3 Zoll OLED I2C 128 x 64 Pixel Display kompatibel mit Arduino und Raspberry Pi
1	DHT22 AM2302 Temperatursensor und Luftfeuchtigkeitssensor
1	KY-021 Magnet Schalter Mini Magnet Reed Modul Sensor
1	RFID Keycard Card 13,56MHz Schlüsselkarte Karte MF S50 (13,56 MHz) – 10x RFID Karte
1	RFID Kit RC522 mit Reader, Chip und Card für Raspberry Pi und Co. (13,56MHz)
1	Breadboard Kit - 3 x 65Stk. Jumper Wire Kabel M2M und 3 x Mini Breadboard 400 Pins kompatibel mit Arduino und Raspberry Pi
1	KY-038 Klangerfassungsmodul Mikrofon Voice- Ton Sensor
1	ESP32 NodeMCU Module WLAN WiFi Development Board mit CP2102
1	GY-521 MPU-6050 3-Achsen-Gyroskop und Beschleunigungssensor alternativ Rüttelkontakt KY-020 oder KY-002 (*)
1	Jumper Wire Kabel 3 x 40 STK. je 20 cm M2M/ F2M / F2F
3	Widerstand 1,0kΩ

(*) The use of the vibrating contacts requires different programming.

2. The Software

For flashing and programming the ESP32:
Thonny oder
µPyCraft
packetsender zum Testen des ESP32/ESP8266 als UDP-Server
Browser: Opera oder Chrome

Verwendete Firmware:

MicropythonFirmware
Bitte eine Stable-Version aussuchen

MicroPython-Programs of the project:

MIT-Lizenz-Text
Device driver:
gy521rc.py
mfrc522.py
sh1106.py
oled.py

Projektdateien:
movementalarm.py
noisy.py
rfid.py
roomclimate.py
steigerung.py
verzaubert.py
webcontrol.py

MicroPython - Language - Modules and Programs

You can find detailed instructions for installing Thonny here. There is also a description of how the Micropython firmware is burned onto the ESP chip.

MicroPython is an interpreter language. The main difference to the Arduino IDE, where you always and exclusively flash entire programs, is that you only have to flash the MicroPython firmware once at the beginning on the ESP32 before the controller understands MicroPython instructions. You can use Thonny, µPyCraft or esptool.py for this. I have described the process for Thonny here.

As soon as the firmware is flashed, you can have a casual conversation with your controller, test individual commands and immediately see the answer without first having to compile and transfer an entire program. This is exactly what bothers me about the Arduino IDE. You simply save an enormous amount of time if you can do simple tests of the syntax and hardware through to trying out and refining functions and entire program parts via the command line before you knit a program out of it. For this purpose I also like to create small test programs over and over again. As a

kind of macro, they combine recurring commands. From such program fragments, entire applications can develop.

Autostart

If the program is to start autonomously when the controller is switched on, copy the program text into a newly created blank file. Save this file under boot.py in the workspace and upload it to the ESP chip. The program starts automatically the next time it is reset or switched on.

Testing programs

If the program is to start autonomously when the controller is switched on, copy the program text into a newly created blank file. Save this file under boot.py in the workspace and upload it to the ESP chip. The program starts automatically the next time it is reset or switched on.

In between, Arduino IDE again?

If you want to use the controller together with the Arduino IDE again later, simply flash the program in the usual way. However, the ESP32 / ESP8266 then forgot that it ever spoke MicroPython. Conversely, every Espressif chip that contains a compiled program from the Arduino IDE or the AT firmware or LUA or ... can easily be provided with the MicroPython firmware. The process is always as described here.

3. We build the tree and wire it up

There is a video showing how to assemble the tree. Of course, after assembling the tree, you can sit back and relax and admire its work. It's even more fun if we change the assembly process in a few places. In three places in the video we have to proceed differently for our project. The 4.7kΩ resistors mentioned there are those with 10kΩ in the package of parts. And these three per board A and B are only soldered at the points on the board, as shown in Figures Fig.2 and Fig.3, the other end of these resistors remains free for the time being. Later we solder thin cables (for example flat ribbon) to these free ends for the connection to the ESP32. This applies to both boards, A and B. The electrolytic capacitors are completely removed.

Abbildung 2: Teil_A	Abbildung 3: Teil_B

The rest of the construction can be done exactly according to the video template. When the base plate is on, we solder the cables to the free ends of the 10kΩ resistors. The length should be between 25 and 30 cm. At the other end of the cable we solder a piece of pin header so that the thing can be plugged in.

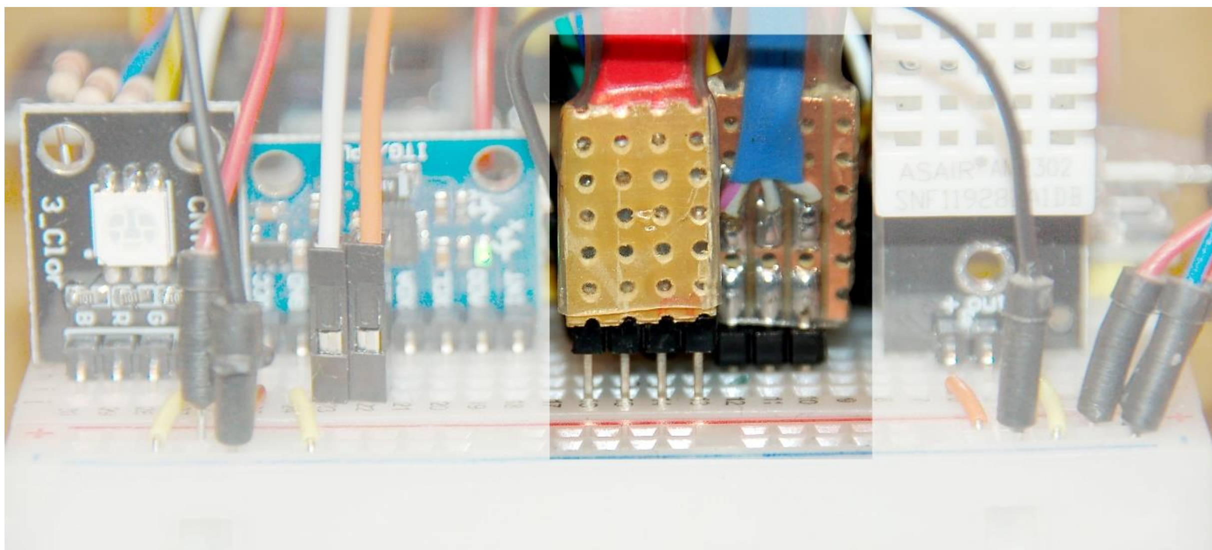

Abbildung 4: The Tree-Connection

The assignment of the connections on the tree to the GPIOs of the ESP32 can be seen in Table 1. The index refers to the list of pin objects. This list, named layer, is used to address the LED layers by looping, as we'll see later. The connections are distributed in such a way that an even-numbered index is always followed by the same-layer LED layer on board B. Of course, any changes are possible at any time.

Bäumchen	A1	B1	A2	B2	A3	B3
GPIO	32	26	33	27	25	12
Index	0	1	2	3	4	5

Tabelle 1: Verbindungen zwischen Bäumchen und ESP32

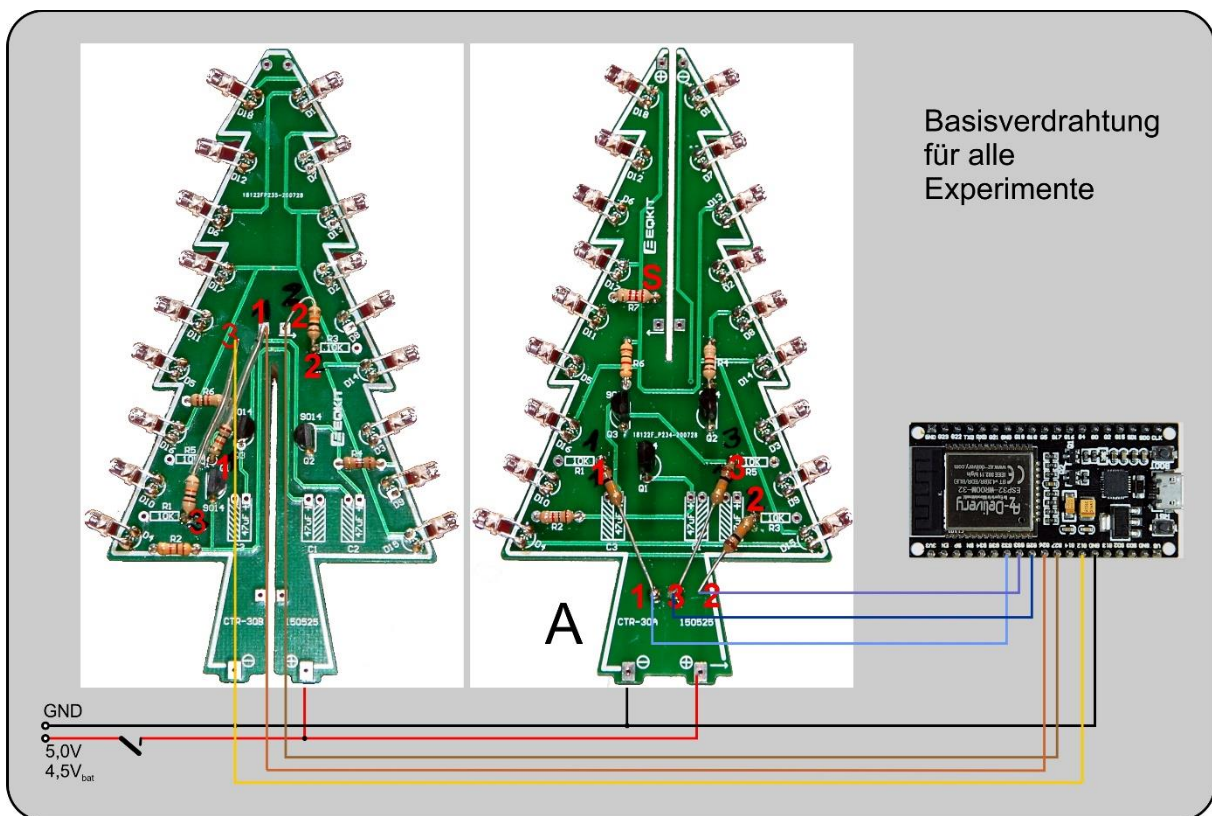

Abbildung 5: Basisverdrahtung

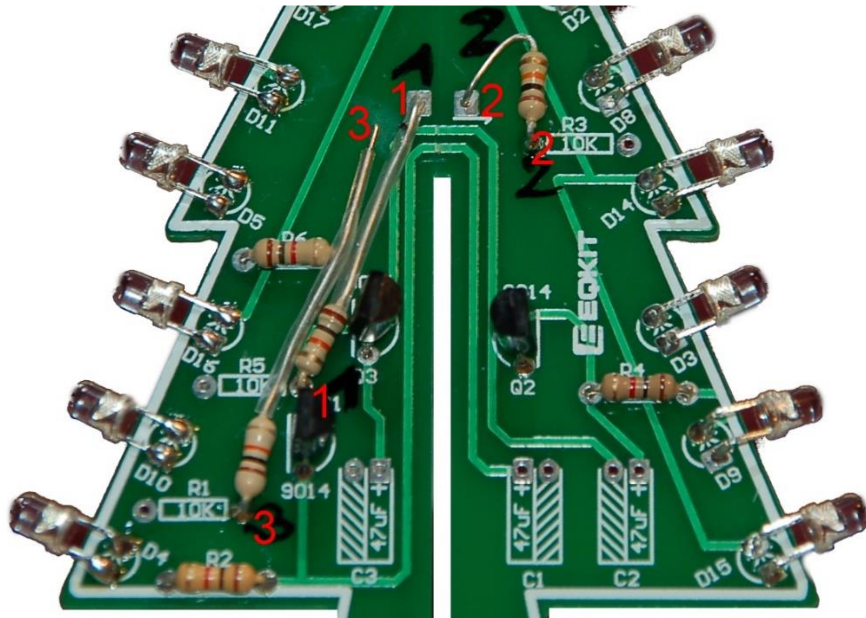

Abbildung 6: Basiswiderstände an Teil B – Detail, freie Enden liegen oben

Abbildung 7: Verkabelt an Teil A

4. Targeted illumination

Is the wiring done? Then we want to light the LEDs on the tree. We supply the tree either with batteries or with the supplied cable from a USB port. After switching on it stays dark. Sure, because the base connections of the Transtoren are exposed, so no base current can flow and because then no collector current flows, the LEDs remain dark.

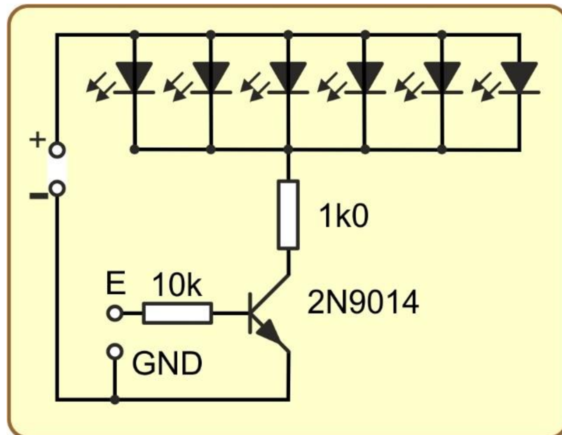

Abbildung 8: Eine von 6 Transistorstufen

This changes when the GPIOs are programmed as outputs and the level of GND potential is raised to 3.3V. We do this by assigning a 1 as the value. We enter the following lines in Thonny's terminal.

```
>>> from machine import pin
>>> a1 = Pin (32, Pin.OUT, value = 0)
>>> a1.value (1)
```

If the wiring is correct, the LEDs of level A1 will start to light up, after entering

```
>>> a1.value (0)
```

to go out. In contrast to the previous version of the tree kit, the new version is equipped with flicker LEDs. Before that, it was simply colored LEDs. This has a certain disadvantage because it is no longer possible to dim the "Flashing LEDs". Still, it's fun to experiment with. Using the six transistors, we are now able to start or switch off all 6 levels exactly as we wish. This also influences the overall brightness.

The arrangement of the lights is shown in Fig. 9. It applies to both part A and part B..

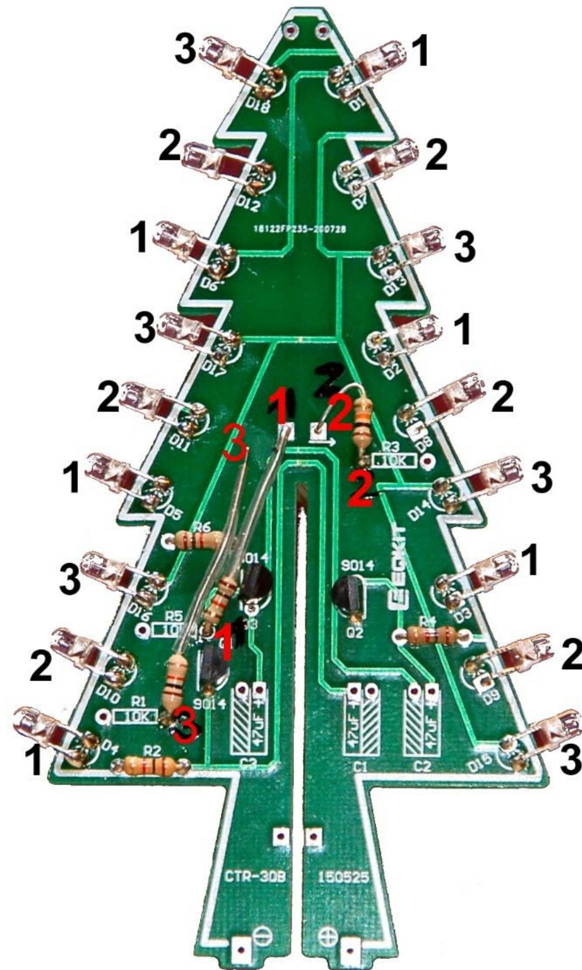

Abbildung 9: Verteilung der LEDs – erfolgt reihum

The arrangement of the LEDs, the wiring to and the connections on the ESP32 are assumed for all further experiments. They therefore no longer appear explicitly in the descriptions and circuit diagrams of the subcircuits.

5. The OLED display

The OLED display can provide us with plain text information, but it can also display graphics in black and white. Programming is easy when we use the associated MicroPython software modules. The hardware driver SH1106 is directly responsible for the 1.3 '' display and can only be used for this. The module framebuf integrated in the MicroPython core provides simple graphic and text commands and the module oled.py gives us convenient commands for text output.

The display is only controlled via the two lines of the I2C bus. We create an I2C object and pass it to the constructor of the OLED class. The hardware device address of the display is fixed and anchored in OLED as a constant. Nevertheless, we first look to see what is on the bus. Then we clear the screen and print a few lines.

The graphic in Fig. 10 and the following program demonstrate the handling. We enter the program in the Thonny editor, save it and then start it with the function key F5.

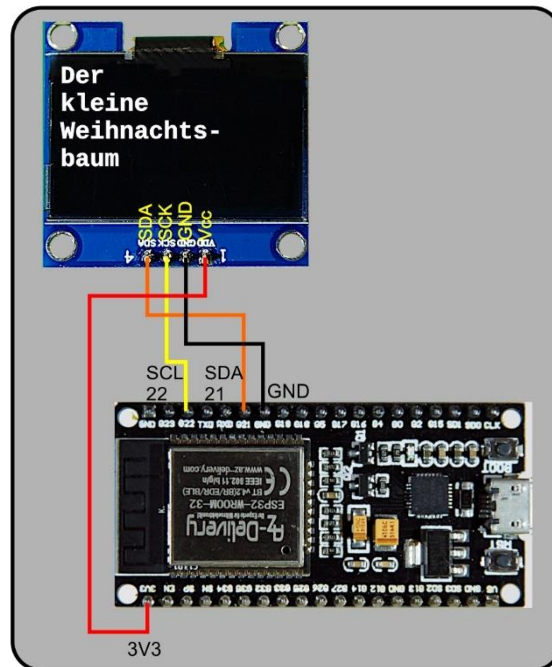

Abbildung 10: Das OLED am ESP32

[oledtest.py]

```
# OLED-Display-Demo
#
from machine import Pin, I2C
from time import sleep
from oled import OLED

# Initialisieren der Schnittstelle **********************
i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
print(i2c.scan())
d=OLED(i2c)

d.clearAll()
d.writeAt("Der",0,0,False)
d.writeAt("kleine",0,1,False)
d.writeAt("Weihnachts-",0,2,False)
d.writeAt("baum",0,3)
sleep(4)
d.clearFT(0,2,15,2,False)
d.writeAt("Christ-",0,2)
```

Output:
[60]
this is the constructor of the OLED class
Size: 128x64

The device address of the display is decimal 60 or 0x3C hexadecimal. The OLED class constructor also knows that the display has 128 x 64 pixels. After the four lines have been output, "Christmas-" is replaced by "Chist-" 4 seconds later. Before doing this, of course, we have to delete this line. You can also try out the individual commands individually via REPL, Thonny's terminal console.

6. Graduated glow

Since we can control the individual levels of the LEDs on the tree separately, we take advantage of this to go up and down from the lowest level - OFF - to maximum brightness. We don't need to change anything in the circuit. The display informs us about the currently active level.

steigerung.py

```
# steigerung.py
#
import sys
from machine import Pin, I2C
from oled import OLED
from time import sleep, sleep_ms

# Initialisieren der Schnittstellen *********************
i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
d=OLED(i2c)

# LED-Schichten einrichten *****************************
#schichtPin = [32,33,25,27,26,12]  # sortiert
schichtPin = [32,26,33,27,25,12]   # verteilt
schicht=[0]*6
for i in range(6): # Ausgaenge erzeugen und auf 0
    schicht[i]=Pin(schichtPin[i],Pin.OUT)
    schicht[i].value(0)

# Funktionen defnieren *********************************
def  switch(n,val):  # Ebene n ein-/ausschalten
    schicht[n].value(val)

def stop():  # alle LED-Ebenen aus
    d.writeAt("GOOD BYE",4,3)
    for i in range(6):
        switch(i,0)

def alle():  # alle LED-Ebenen ein
    for i in range(6):
        sleep_ms(300)
        switch(i,1)

# Hauptprogramm ******************************************
d.clearAll()
d.rect(4,16,123,40,1)  # Rechteck in Pixelwerten
for j in range(3):
    for i in range(6):
        d.writeAt("Ebene: {} ein".format(i),2,3)
        switch(i,1)
        sleep_ms(3000)
    for i in range(5,-1,-1):
        d.writeAt("Ebene: {} aus".format(i),2,3)
```

```
        switch(i,0)
        sleep_ms(3000)
d.clearFT(2,3,14,3,False)
stop()
```

We define the order of the layers in the layerPin list. The pin objects are generated according to this pattern in the following for loop. The functions switch (), stop () all of us () help us to make the program clearer. We will also use them several times in the following chapters.

In the main program we clear the screen and draw a frame. 4 and 16 are the pixel coordinates of the top left corner, 123 and 40 are the width and height in pixels and 1 is the color white, there are no more colors. The outer for loop counts the total number of passes. The first inner for loop counts i up at intervals of 3 seconds and switches the levels on. The second loop counts down and turns off the LEDs again.

The last output is removed and the stop () function reliably extinguishes all LEDs and says goodbye with a friendly "GOOD BYE".

We can fully specify the behavior of the LEDs ourselves via the interval length and the number of runs.

7. The enchanted tree

Anyone could come along and want to turn on our little tree. But nothing there, that can only be done by our magic hands. Of course we are not saying that we have hidden a small neodymium magnetic stick in each hand. Why do we need it? Just for "magic". Because we have now rebuilt our circuit. A reed contact to ground is now connected to GPIO pin 13. In the glass tube there is a switching contact that closes when a magnet approaches.

Attention:
The glass is very brittle and the wires are very stiff. Do not bend on it, otherwise the glass will splinter and you can bury the component.

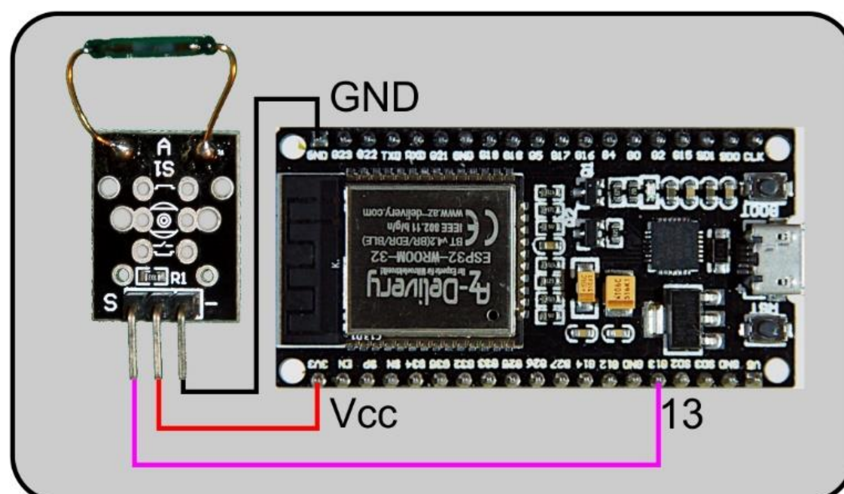

Abbildung 11: Reedkontakt hilft zaubern

We must not forget something else, namely that the OLED and the little tree remain connected as described above.

verzaubert.py

```
from os import uname
import sys
from machine import  Pin, I2C
from oled import OLED
from time import sleep_ms

# Initialisieren der Schnittstellen *********************
i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
d=OLED(i2c)

taste=Pin(0,Pin.IN,Pin.PULL_UP)
reed=Pin(13,Pin.IN,Pin.PULL_UP)

# LED-Schichten einrichten *****************************
schichtPin = [32,33,25,26,27,12]
schicht=[0]*6
for i in range(6):
    schicht[i]=Pin(schichtPin[i],Pin.OUT)
    schicht[i].value(0)

# Funktionen defnieren ***********************************
def  switch(n,val):
    schicht[n].value(val)

def stop():
    d.writeAt("   MUGGLE     ",1,3)
    for i in range(6):
        switch(i,0)

def alle():
    d.writeAt("  DUMBLEDOR   ",1,3)
    for i in range(6):
        sleep_ms(300)
        switch(i,1)

# Hauptprogramm *****************************************
d.clearAll()
d.writeAt("Kannst du ...",0,0)
d.writeAt("ZAUBERN???",0,1)
while 1:
    if reed()==0:
        alle()
    else:
        stop()
```

The reed contact must be attached in such a way that we can get close to it with our magnet when we place a tree or breadboard on the palm of the hand. A thin black

glove helps us to keep the magnet invisible. Most likely everyone around you is Muggles.

The program is very simple. We already know everything up to the main program. The while loop runs endlessly until the power is switched off. If the contact in the vicinity of the magnet is closed, then GPIO13 is at GND potential and all lights go on. Otherwise the resistor built into the module will pull the GPIO13 to Vcc = 3.3V and the lights will go out.

So that the magic works better, the tree with breadboard should be powered by the battery. The plus connection of the battery must be connected to pin Vin / 5V of the ESP32. Furthermore, the program must then be uploaded to the ESP32 as boot.py so that the controller starts up autonomously after being switched on. How this works is described in detail in Chapter 2 - Autostart.

8. Advent and Christmas, the "staade" time.

"Staad" translated into general German usage means something like "calm", "contemplative". However, everyday life teaches us that things can get tough even in the run-up to Christmas. If it gets too turbulent, the tree warns you to switch back a few decibels. How does it do that? Now there is a sound module that picks up sound and delivers the digitized signal to the ESP32.

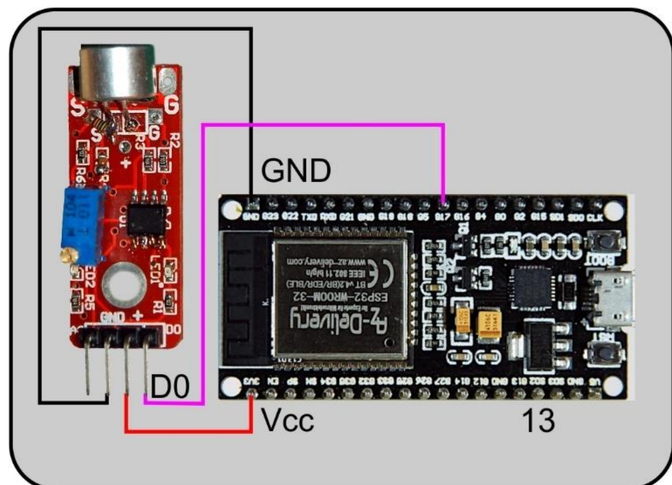

Abbildung 12: Soundmachine am ESP32

noisy.py

```
# noisy.py
import esp32
from os import uname
from machine import Timer, Pin, I2C
from oled import OLED
from time import time, sleep,


i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
d=OLED(i2c)

# IRQ-Steuerung durch Soundmodul
ST=Timer(1)
sound=Pin(17,Pin.IN)
#
# LED-Schichten einrichten *******************************
schichtPin = [32,33,25,27,26,12]
L=len(schichtPin)
schicht=[0]*L
for i in range(L):
    schicht[i]=Pin(schichtPin[i],Pin.OUT)
    schicht[i].value(0)

# Funktionen defnieren ***********************************
def  switch(n,val):
    schicht[n].value(val)

def stop():
    d.clearAll()
    d.writeAt("ALLES GUT",4,2)
    for i in range(L):
        switch(i,0)

def alle():
    d.clearAll()
    d.writeAt("ZU LAUT!!",0,0)
    for i in range(L):
        sleep(0.5)
        switch(i,1)

def soundDetected(pin):
    global n
    if pin==Pin(17):
        sound.irq(handler=None)
        if n:
            return
        n=True
        ST.init(period=15000, mode=Timer.ONE_SHOT,\
                callback=soundDone)
        print("begin",time())
        alle()
```

```
def soundDone(t):
    global n
    n=False
    print("ende",time())
    stop()
    sound.irq(handler=soundDetected, trigger=Pin.IRQ_FALLING)

# Hauptprogramm ******************************************
n=False
sound.irq(handler=soundDetected, trigger=Pin.IRQ_FALLING)
```

Sound is transmitted through rapid pressure fluctuations in the air. A sound signal propagates at approx. 340 m / s. In a room with practically no noticeable delay. The microphone in the sound module converts the pressure fluctuations into an electrical signal. In contrast to the reed contact, these oscillations can no longer be recognized by querying the GPIO port; this process is too slow. That's why we're using a different technique here, interrupt programming. An interrupt is the interruption of a program by a certain event. We are going to use two different sources of interruption. One triggers an interruption when the level on a GPIO pin changes from 0 to 1 or vice versa. The other IRQ source is a hardware timer of the ESP32. It triggers the IRQ when the alarm goes off.

Both of them now alternately pass the ball to each other. The GPIO17 waits for a signal from the sound module. If a falling edge occurs, the function soundDetected () starts and first checks whether it is meant based on the transferred parameter pin. If n is True, then a cycle is already running and there is nothing more to be done. If, on the other hand, n is False, then it is a fresh job. The Pin-Change-IRQ is switched off and n is set to True in order to suppress immediately following pulses at GPIO17. Then the timer is started, which specifies the runtime of the tree lighting. The lighting is switched on by calling up all ().

If the timer has expired, the associated interrupt is triggered, which starts the soundDone () function. n is set to false, the lights go out, and the pin change IRQ is armed again.

The main program consists of just two lines. n is set to false so that the subsequently activated pin change IRQ can be triggered.

The interesting thing is that the IRQs are still active even after the main program has ended. To switch this off, the ESP32 must be reset with the STOP / RESTART button.

9. On the trail of the tree stealer

There are supposed to be people who steal their Christmas tree - from the forest. Well, dear foresters, do as we do and build a guardian like the one we are about to describe in your little trees.

OK, admittedly that will be just as difficult as monitoring other bans when there is no staff. Then why do you forbid something if you cannot control it? So be it.

Our sapling gets a supervisor - namely itself! He is helped by a sensor that comes from a completely different corner. The GY-521 module used with the MPU6050 component is an accelerometer with a gyroscope. This can be used to measure accelerations, forces and rotations. Yes, and if you want to take something away, you have to lift it up and set it in motion. In both cases the object is accelerated. Even very small changes in location generate forces and thus our sensor to respond. The rest is simple, after the triggering, the tree is illuminated and the potential thief hopefully runs away. A timer interrupt is responsible for the duration of the alarm.

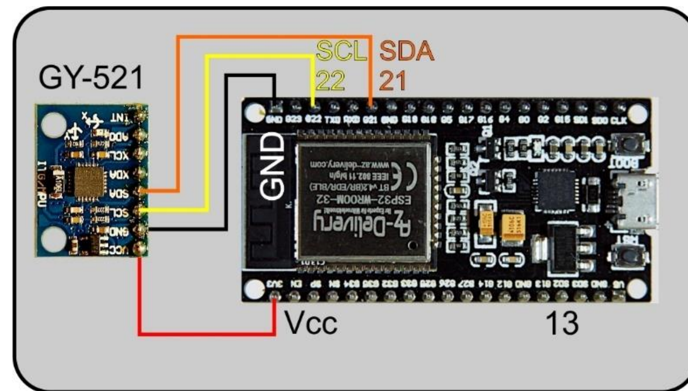

Abbildung 13: Antiklau-Einheit

alarm.py

```
# alarm.py
# RED-ALLERT by movement
import esp32
from os import uname
from machine import Timer, Pin, I2C
from oled import OLED
from time import sleep,ticks_ms, ticks_us, sleep_ms
from gy521rc import GY521

# Initialisieren der Schnittstellen **********************
i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
d=OLED(i2c)

AT=Timer(0)
acc=GY521(i2c)
limit=36
dauer=5000

schichtPin = [32,33,25,27,26,12]
L=len(schichtPin)
schicht=[0]*L
```

```
for i in range(L):
    schicht[i]=Pin(schichtPin[i],Pin.OUT)
    schicht[i].value(0)

# Funktionen defnieren ***********************************
def TimeOut(t):
    start=ticks_ms()
    def compare():
        return int(ticks_ms()-start) >= t
    return compare

def  switch(n,val):
    schicht[n].value(val)

def stop():
    d.clearAll()
    d.writeAt("ALLES GUT",4,2)
    for i in range(L):
        switch(i,0)

def alle():
    d.clearAll()
    d.writeAt("DIEBSTAHL",0,0)
    for i in range(L):
        sleep(0.5)
        switch(i,1)

def hasMoved(delay):
    xs,ys,zs=0,0,0
    for i in range(100):
        x,y,z=acc.getXYZ()
        xs+=x
        ys+=y
        zs+=z
    xs//=100
    ys//=100
    zs//=100
    #print(xs,ys,zs)
    n=0
    while 1:
        x,y,z=acc.getXYZ()
        x=abs(xs-x)
        y=abs(ys-y)
        z=abs(zs-z)
        #print(x,xs//limit)
        if x > abs(xs//limit) :
            print("*******",n)
            n+=1
            alle()
            # Optional Nachricht via UDP
            AT.init(period=delay, mode=Timer.ONE_SHOT,\
                    callback=alertDone)
```

```
        sleep(0.3)

def alertDone(t):
    stop()

print("Diebstahlschutz gestartet")
hasMoved(dauer)
```

Again there are good friends in the program. What is new, however, is the initialization of the GY521. For the block we have to upload another module to the ESP32, gy521rc.py. The class it contains has the same name as the module.

Like the OLED display, the GY521 is also operated via the I2C bus. We pass the same I2C object to the constructor, define the threshold for triggering the alarm and its duration in milliseconds.

The threshold is the absolute amount of deviation of the measured value from the mean value of the acceleration measurement in the x-direction. The sensor is aligned so that the positive x-axis points vertically upwards. The measured value is around 16000 counts and in this case corresponds to the acceleration due to gravity g = 9.81m / s².

The hasMoved () function represents the main loop here. When entering, the mean value is determined by 100 measurements. It goes without saying that the sensor must not move.

Then it goes into the main loop. the current acceleration is measured and the deviations from the mean values are calculated. If the difference exceeds the specified limit, an alarm is triggered and the timer is activated. The alarm means that the tree goes to full brightness.

The service routine of the timer IRQ extinguishes the lights. The solution via the IRQ ensures that the circuit is armed again immediately after the alarm is triggered. If the alarm duration were specified by a sleep command in the main loop, the circuit would be dead for this duration.

The vibrating contacts mentioned in the parts list would be connected to the ESP32 in a similar way to the reed contact, but would not allow the sensitivity to be set.

10. ESP32 and DHT22 wish you a pleasant stay

A pleasant room climate is part of the festive mood. Now the ESP32 cannot change the room climate in this simple application, but it can report about it. The exact values for temperature and humidity are shown on the OLED display; the tree tells us the rough values. In 2-degree steps, it reports the values of the room temperature through a different number of switched-on LED levels.

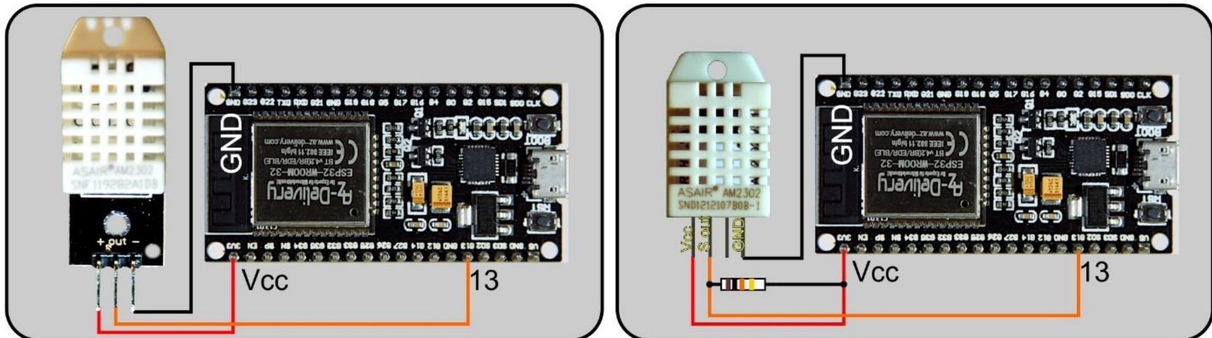

Abbildung 14: Temperatur- und Luftfeuchtemessung in einem Modul

There are 2 variants of DHT22, alias AM2302. The module in the illustration on the left already contains the necessary pull-up resistor for the one-wire bus, which, by the way, must not be confused with the system of the Dallas module DS18B20. The Dallas bus has a completely different timing. For the bare version in the right figure, a 4.7kΩ to 10kΩ resistor to Vcc must be built in.

The operation in the program is very easy. The dht module, which is already integrated in MicroPython, provides the three necessary commands.

roomclimate.py

```
# roomclimate.py
import esp32, dht
from os import uname
import sys
from machine import Pin, I2C
from oled import OLED
from time import sleep

# Initialisieren der Schnittstellen **********************
i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
d=OLED(i2c)

dhtPin=Pin(13)
dht22=dht.DHT22(dhtPin)

# LED-Schichten einrichten ******************************
schichtPin = [32,33,25,26,27,12]
schicht=[0]*6
for i in range(6):
    schicht[i]=Pin(schichtPin[i],Pin.OUT)
    schicht[i].value(0)

# Funktionen defnieren ***********************************
```

```
def  switch(n,val):
    schicht[n].value(val)

def stop():
    d.writeAt("TEMP TO LOW",4,2)
    for i in range(6):
        switch(i,0)

def alle():
    for i in range(6):
        sleep_ms(300)
        switch(i,1)

def tree(n):
    for i in range(6):
        if i <=n:
            switch(i,1)
        else:
            switch(i,0)
# Hauptprogramm *****************************************
d.clearAll()
d.writeAt("***RAUMKLIMA***",0,0)
while True:
    sleep(0.3)
    dht22.measure()
    t=dht22.temperature()
    h=dht22.humidity()
    d.rect(0,10,126,38,1)
    d.clearFT(1,2,14,3)
    d.writeAt("TEMP: {:.1f} *C".format(t),1,2)
    d.writeAt("HUM : {:.1f} %".format(h),1,3)
    tree(int(((t-15)//2)%6))
    sleep(2.7)
```

In addition to the usual suspects, the program only offers the import of the dht module, the instantiation of the dht22 object and the main loop with the measurement job dht22.measure () and the reading in of temperature and humidity values. We already know the output on the display and the tree display. The conversion of the temperature from ° C to the index of the lighting level is perhaps interesting and inconspicuous. by the term int (((t-15) // 2)% 6). From the quotient value of the integer division of the deviation of the temperature from 15 ° C upwards and 2, the 6-part remainder is determined and, to be on the safe side, represented as an integer. Again very slowly.

Example: t = 18 ° C
18-15 = 3
3 // 2 = 1
1% 6 = 1 i.e. level index 1

for 28 ° C the result would be: 28-15 = 13; 13 // 2 = 6; 6% 6 = 0; The last step is necessary because there is no stage number 6.

11. The (slightly different) Christmas raffle

I know that from my school days. Everyone brought a parcel with them during Advent and the week before the outdoors the raffle started - every ticket wins.

I have chosen neutral RFID cards as the recyclable lots. The ESP32 and the RFID kit take care of the drawing. You only have to take care of the profits yourself. Of course, the tree is also there. With its luminosity, it announces its win to the respective player. So that there are no doubts about the interpretation, the display clearly names the location of each drawing: Freiburg, Berlin, Hamburg…. Six lottery tickets and one master card are required.

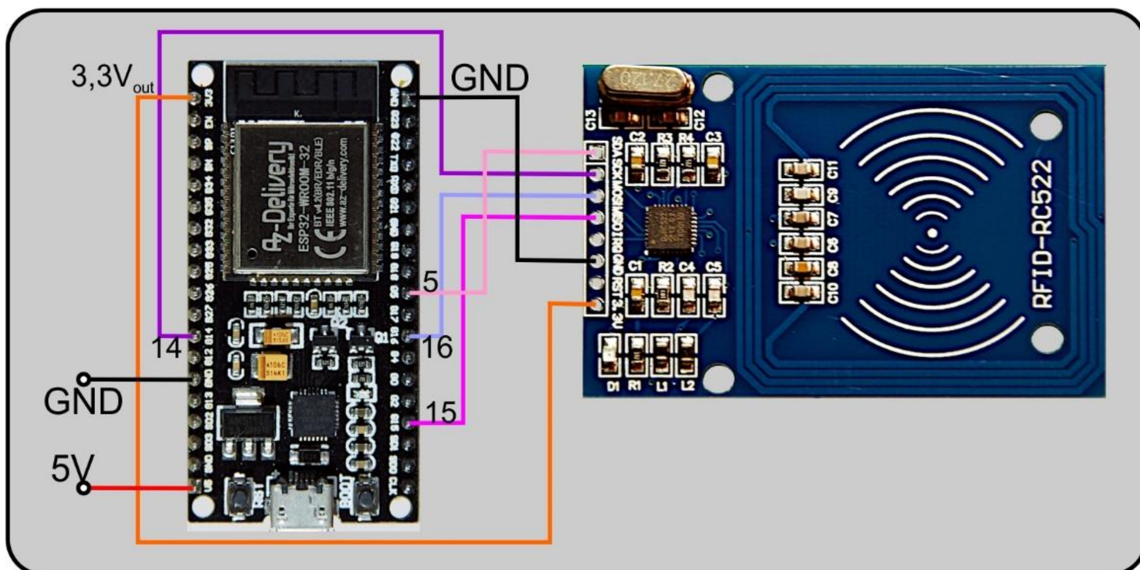

Abbildung 15: RFID-Karten-Leser

With the SPI bus, the wiring is a little more complex than with the I2C bus with its 2 lines. SPI bus devices do not have a hardware device address, instead they have a chip select connection that must be LOW if the device is to be addressed. The data transfer is also a little different, it is always sent and received at the same time. It is not necessary to clarify the more detailed procedure here, because the MFRC522 class does it for us. We only inform the constructor of the pin assignments and the transmission speed. The transfer works at a brisk 3.2MHz. For comparison, I2C works at 400kHz.

The readUID () function reads out the unique ID of the card and returns it as a hexadecimal value and as a decimal number. The cards are requested via the OLED display. A timeout ensures an orderly withdrawal so that the function does not block the entire process. In this case, the value None is returned instead of the card ID.

Abbildung 16: RFID-Karten und Chip

In order for the ticket cards to come into play, we need a master card. To do this, we take any card or chip from the stack, read the ID and assign the decimal value to the variable at the beginning of the program:
MasterID = 4217116188.

When the ESP32 is started for the first time, it detects that there is not yet a file with the batch card data and requests the master card. After this has been recognized, a ticket will be requested. After reading out the ID, it is written to the file and the master card is requested again. Reading is continued until the last ticket card. If no ticket is offered for 10 seconds after the master card has been requested, the system restarts itself. The prerequisite for this is that the rfid.py program has been sent to the ESP32 as boot.py. Chapter 2 - Autostart explains exactly how to do this. To start from scratch, we can delete the slavecards.txt file with the ticket IDs via the Thonny console. After a reset, the ticket cards can then be read in again.

rfid.py

```
# rfid.py
# workes with RC522 13,2MHz
import mfrc522
import esp32, dht
from os import uname
from machine import Timer, Pin, I2C, ADC, reset
from oled import OLED
from time import sleep,ticks_ms, ticks_us, sleep_ms
from gy521 import GY521

# Initialisieren der Schnittstellen **********************
if uname()[0] == 'esp32':
    #                     sck, mosi, miso, cs=sda
    rdr = mfrc522.MFRC522(14,  16,   15,   5,
baudrate=3200000)
elif uname()[0] == 'esp8266':
    #                     sck, mosi, miso, cs=sda
    #                     D3   D4    D2    D5
```

```
    rdr = mfrc522.MFRC522(0,   2,    4,    14,
baudrate=100000)
else:
    raise RuntimeError("Unsupported platform")
MasterID=4217116188  #  0XFB5C161C

i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
d=OLED(i2c)

schichtPin = [32,33,25,27,26,12]
schicht=[0]*6
for i in range(6):
    schicht[i]=Pin(schichtPin[i],Pin.OUT)
    schicht[i].value(0)
gewinn=[
        "Freiburg",
        "Berlin",
        "Hamburg",
        "Augsburg",
        "Ratzeburg",
        "Erfurt",
        "Essen",
        "Bonn",
       ]
# Funktionen defnieren ***********************************
def TimeOut(t):
    start=ticks_ms()
    def compare():
        return int(ticks_ms()-start) >= t
    return compare

def readUID(display,kartentyp,timeout):
    display.clearFT(0,1,15,show=False)
    display.writeAt("Put on "+kartentyp,0,1)
    readTimeOut=TimeOut(timeout)
    while not readTimeOut():
        (stat, tag_type) = rdr.request(rdr.REQIDL)
        if stat == rdr.OK:
            (stat, raw_uid) = rdr.anticoll()
            if stat == rdr.OK:
                display.clearFT(0,2,15,show=False)
                display.writeAt("Card OK",0,2)
                sleep(1)
                userID=0
                for i in range(4):
                    userID=(userID<<8) | raw_uid[i]
                userIDS="{:#X}".format(userID)
                print(userIDS)
                return userID,userIDS
    return None

def addUID(display):
```

```
    display.clearAll()
    m=readUID(display,"Master",3000)
    if m is not None:
        mid,_= m
        if mid==MasterID:
            sleep(3)
            u=readUID(display,"Slavecard",3000)
            if u is not None:
                uid,uids=u
                if uid is not None and uid != MasterID:
                    with open("slavecards.txt","a") as f:
                        f.write("{}\n".format(uids))
                        display.writeAt("New slave
written",0,3)
                        sleep(3)
                        return True
            else:
                display.writeAt("ERROR!!!",0,3)
                display.writeAt("Card not added!",0,4)
                return False
        else:
            display.writeAt("ERROR!!!",0,3)
            display.writeAt("Not mastercard",0,4)
            sleep(3)
    return False

def  switch(n,val):
    schicht[n].value(val)

def stop():
    d.writeAt("GOOD BYE",4,2)
    for i in range(6):
        switch(i,0)

def alle():
    for i in range(6):
        sleep_ms(300)
        switch(i,1)

def tree(n):
    for i in range(6):
        if i <=n:
            switch(i,1)
        else:
            switch(i,0)

# ****************** Hauptprogramm ********************
d.clearAll()
d.writeAt("*XMAS  LOTTERIE*",0,0)
d.rect(0,20,127,28,1)
cards=[]
try:
```

```
    with open("slavecards.txt","r") as f:
        for line in f:
            cards.append(line.strip("\n"))
    closed=TimeOut(60000)
    while not closed():
        u=readUID(d,"LOSKARTE",5000)
        d.clearFT(1,3,14,4,False)
        if u is not None:
            uid,uids=u
            try:
                n=cards.index(uids)
                d.writeAt("TREFFER {}".format(n),1,3, False)
                d.writeAt(gewinn[n],1,4)
            except ValueError as e:
                d.writeAt("TROSTPREIS",1,3)
                n=-1
            tree(n)
            closed=TimeOut(60000)
            sleep(10)
            stop()
except OSError as e:
    print("keine Datei, keine Daten!")
    allRead=TimeOut(10000)
    while not allRead():
        if addUID(d):
            allRead=TimeOut(10000)
    print("Alle Karten eingelesen und gespeichert")
    d.clearFT(0,3,15,4,False)
    d.writeAt(" ALL CARDS READ",0,3)
    d.writeAt("**R E B O O T**",0,4)
    reset()
d.clearFT(0,1,15,3,False)
d.writeAt("Lotterie neu",0,2)
d.writeAt("starten",0,3)
```

For a game cycle, 6 wins are determined, the 6 cards are shuffled and distributed and the new round is started with the PROG button on the ESP32.

12. The tree in the LAN / WLAN

Let's fill up the dozen and connect the sapling to the network. Because if an ESP32 is already being used for control, then LAN or WLAN access to the control must also be provided. I decided to implement a web server on the ESP32 because the levels of the tree can then be controlled with almost any browser. An alternative would be a UDP server on the controller and a mobile phone app. But that would have gone beyond the scope of this blog and that's why I refrained from it. For those interested, I have already described this type of control in other posts, for example here and here.

For the circuit, the structure of Chapter 5 is needed, which we are expanding with an RGB LED and three 1.0 kΩ resistors.

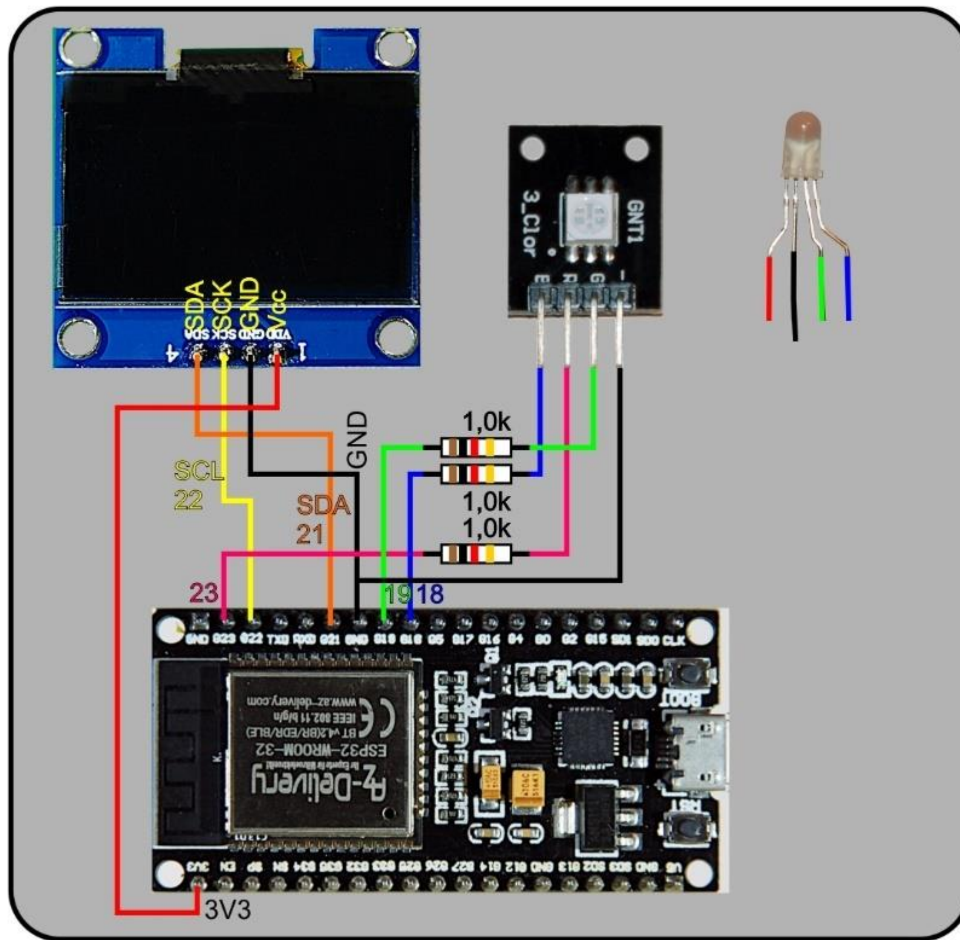

Abbildung 17: Web-Aufbau

After importing the necessary modules, we define the pins for the RGB LED, which visibly tells us the network status from a long distance. This is followed by the selection of the network operating mode, WLAN or ESP32's own access point. WLAN is preset by default. In order to access the WLAN router, the access data must then also be entered here. The layer definition is extended by three lists, plain text for on / off, background color for the table in the website and the switching states of the levels.

The blocking functions have been removed, swell (), swell (), wave () and tree (). HexMac (), blink (), ledsOff () and web_page () have been added. hexMac outputs the MAC address of the ESP32 in station mode, blink () signals the network and server status. With ledsOff () the RGB-LED is switched off and web_page () picks up requests from the browser, executes the orders and returns a response as website text.

The browser's request is sent to the server as a query string. The string has the form? A,? P or? E = x & v = y. Here x stands for the level number and y for the switching state, 0 or 1.

web_page () converts the query to capital letters and first checks for "A" and "P". If the request contains more than 2 characters, an attempt is made to determine the level and the switching status. If an error occurs, no action is triggered and the bare start page is called. This also happens if no query string has been specified. The website is then set up as a string and returned to the main loop.

After the function definitions, the network connection is established, either as a separate access point or as a connection to the WLAN router. This is controlled by the two variables ownAP and WLANconnect. In both cases, a fixed IP address (10.0.1.181) is assigned because it is a server. Dynamic addresses from the WLAN router are unsuitable as they can change from time to time. The connection establishment to the router is indicated by the blinking of the blue LED. The display informs us when the connection is established and the connection socket s is also ready to accept requests.

In the main loop, the receive loop of the accept () method waits for a request. If nothing arrives by the timeout, accept () throws an exception, which we catch with the preceding try.

If there is a request, accept () returns a communication socket c and the address of the requesting machine. c is used to handle the data exchange between client and server, while s becomes free again to accept further incoming requests. The method c.recv () returns the text of the request, of which we are only interested in the first few characters. In the development phase you can enter queries by hand to test the web_page () parser. ownAP and WLANconnect must then both be set to False.

The byte object request of the received text is now decoded into a string r, which is easier to handle. We are looking for a "GET /" at the very beginning of the string r and for the position followed by "HTTP". If both are found, we isolate the text after the "/" of "GET" up to the space in "HTML" and send it as a query string to the parser web_page (). We receive its response in the variable response. Then we send the HTML header and the text of the HTML page with the contained answer back to the caller. The following two else and the except are used to catch and handle possible errors. The final c.close (), which closes the communication socket c, is important.

After a button query to abort the program, the green LED flashes briefly as a heartbeat to indicate that the system is still alive.

webcontrol.py

```
# webcontrol.py
# Fernsteuerung vom Browser via TCP
# (C) 2021 Jürgen Grzesina
# released under MIT-License (MIT)
# http://www.grzesina.de/az/weihnachtsbaum/MIT-License.txt
#
from machine import Pin, I2C
from oled import OLED

# ******************* Network stuff *******************
from time import sleep,ticks_ms, sleep_ms
try:
  import usocket as socket
except:
  import socket
import ubinascii
import network
```

```
statusLed=Pin(18,Pin.OUT,value=0) # blau=2
onairLed=Pin(19,Pin.OUT,value=0)  # gruen=1
errorLed=Pin(23,Pin.OUT,value=0)  # rot=0
led=[errorLed,onairLed,statusLed ]
red,green,blue=0,1,2
request = bytearray(50)
response=""
taste=Pin(0,Pin.IN,Pin.PULL_UP)

# Auswahl der Betriebsart Netzwerk oder Tastatur:
# ---------------------------------------------------------
# Netzwerk: Setzen Sie genau !_EINE_! Variable auf True
WLANconnect=True  # Netzanbindung ueber lokales WLAN
ownAP=False       # Netzanbindung ueber eigenen Accessppoint
# beide False ->> Befehlseingabe ueber PC + USB in Testphase
# Falls WLANconnect=True:
# Geben Sie hier die Credentials Ihres WLAN-Accesspoints an
mySid = 'beteigeuze'; myPass = "4u2getACCess2theweb"
#mySid = 'YOUR_SSID'; myPass = "YOUR_PASSWORD"
myIP="10.0.1.181"
myPort=9002

# Initialisieren der Schnittstellen **********************
i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
d=OLED(i2c)

#schichtPin = [32,33,25,27,26,12]  # sortiert
schichtPin = [32,26,33,27,25,12]   # verteilt
schicht=[0]*6
for i in range(6):
    schicht[i]=Pin(schichtPin[i],Pin.OUT)
    schicht[i].value(0)
zustand=["aus","an "]
color=["red","lightgreen"]
eState=[0,0,0,0,0,0]

connectStatus = {
    1000: "STAT_IDLE",
    1001: "STAT_CONNECTING",
    1010: "STAT_GOT_IP",
    202:  "STAT_WRONG_PASSWORD",
    201:  "NO AP FOUND",
    5:    "GOT_IP"
    }

# Funktionen defnieren ***********************************
def TimeOut(t):
    start=ticks_ms()
    def compare():
        return int(ticks_ms()-start) >= t
    return compare
```

```
def  switch(n,val):
    schicht[n].value(val)

def stop():
    d.writeAt("ALL LEDS OFF",2,5)
    for i in range(6):
        switch(i,0)

def alle():
    d.writeAt("ALL LEDS ON ",2,5)
    for i in range(6):
        sleep_ms(300)
        switch(i,1)

def tree(n):
    d.writeAt("TREE PROGR. ",2,5)
    for i in range(6):
        if i <=n:
            switch(i,1)
        else:
            switch(i,0)

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)):     # Fuer alle Bytewerte
    macString += hex(byteMac[i])[2:]  # ab Position 2 bis Ende
    if i <len(byteMac)-1 :            # Trennzeichen
      macString +="-"
  return macString

def blink(pulse,wait,col,inverted=False):
    if inverted:
        led[col].off()
        sleep(pulse)
        led[col].on()
        sleep(wait)
    else:
        led[col].on()
        sleep(pulse)
        led[col].off()
        sleep(wait)

def ledsOff():
    for i in range(3):
        led[i].value(0)

def web_page(q):
    global eState
```

```
    q=q.upper()
    print("Anfrage: ",q)
    if q=="?A":
        alle()
        for i in range(6):
            eState[i]=1
    elif q=="?P":
        stop()
        for i in range(6):
            eState[i]=0
    elif len(q)>2:
        try:
            ebene,state=q[1:].split("&")
            _,ebene= ebene.split("=")
            _,state= state.split("=")
            ebene=(int(ebene) if 0<=int(ebene)<=5 else 0)
            state=(int(state) if 0<=int(state)<=1 else 0)
            switch(ebene,state)
            eState[ebene]=state
        except:
            pass
    else:
        pass
    antwort="<tr>"
    for i in range(6):
        h="<td bgcolor={}><H3>E{}
{}.</H3></td>".format(color[eState[i]],i, zustand[eState[i]])
        antwort=antwort+h
    antwort=antwort+"</tr>"
    html1 = """<html>
    <head>
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    </head>
    <body>
    <h2>Hallo, <br>ich bin dein
Weihnachtsb&auml;umchen</h2>"""
    html2="""<table border=2 cellspacing=2>
    """
    html3="""
    <tr>
    <td>
    <a href='http://10.0.1.181:9002/?e=0&v=1'><H3>E0 An </H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=1&v=1'><H3>E1 An </H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=2&v=1'><H3>E2 An </H3>
</a>
```

```
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=3&v=1'><H3>E3 An </H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=4&v=1'><H3>E4 An </H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=5&v=1'><H3>E5 An </H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?a'><H3>ALLE AN </H3> </a>
    </td>
    </tr>
    <tr>
    <td>
    <a href='http://10.0.1.181:9002/?e=0&v=0'><H3>E0 Aus</H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=1&v=0'><H3>E1 Aus</H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=2&v=0'><H3>E2 Aus</H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=3&v=0'><H3>E3 Aus</H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=4&v=0'><H3>E4 Aus</H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?e=5&v=0'><H3>E5 Aus</H3>
</a>
    </td>
    <td>
    <a href='http://10.0.1.181:9002/?p'><H3>ALLE AUS</H3> </a>
    </td>
    </tr>
    """
    html9 = "</table> </body> </html>"
    html=html1+html2+antwort+html3+html9
    return html

if taste.value()==0:
```

```
    print("Mit Flashtaste abgebrochen")
    ledsOff()
    d.writeAt("Abbruch d. User ",0,5)
    sys.exit()

# ********************************************************
# Netzwerk einrichten
# ********************************************************
# Eigener ACCESSPOINT
# ********************************************************
if ownAP and (not WLANconnect):
    #
    nic = network.WLAN(network.AP_IF)
    nic.active(True)
    ssid="christbaum"
    passwd="don't_care"

    # Start als Accesspoint
    nic.ifconfig((myIP,"255.255.255.0",myIP,\
                  myIP))

    print(nic.ifconfig())

    # Authentifizierungsmodi ausser 0 werden nicht
unterstuetzt
    nic.config(authmode=0)

    MAC=nic.config("mac") # liefert ein Bytes-Objekt
    # umwandeln in zweistellige Hexzahlen
    MAC=ubinascii.hexlify(MAC,"-").decode("utf-8")
    print(MAC)
    nic.config(essid=ssid, password=passwd)

    while not nic.active():
      print(".",end="")
      sleep(0.5)

    print("Unit1 listening")
# ***************  Setup accesspoint end ****************

# ********************************************************
# WLAN-Connection
# ********************************************************
if WLANconnect and (not ownAP):
    nic = network.WLAN(network.STA_IF) # erzeuge WiFi-Objekt
    nic.active(True)  # Objekt nic einschalten
    #
    MAC = nic.config('mac')  # binaere MAC-Adresse abrufen +
    myMac=hexMac(MAC)        # in Hexziffernfolge umwandeln
    print("STATION MAC: \t"+myMac+"\n") # ausgeben
    # Verbindung mit AP im lokalen Netzwerk aufnehmen,
    # falls noch nicht verbunden, dann
```

```
    # connect to LAN-AP
    if not nic.isconnected():
      nic.connect(mySid, myPass)
      # warten bis die Verbindung zum Accesspoint steht
      print("connection status: ", nic.isconnected())
      while not nic.isconnected():
        blink(0.8,0.2,0)
        print("{}.".format(nic.status()),end='')
        sleep(1)
    # zeige Verbindungsstatus & Config-Daten
    print("\nconnected: ",nic.isconnected())
    print("\nVerbindungsstatus: ",connectStatus[nic.status()])
    print("Weise neue IP zu:",myIP)
    nic.ifconfig((myIP,"255.255.255.0",myIP, \
                  myIP))
    STAconf = nic.ifconfig()
    print("STA-IP:\t\t",STAconf[0],"\nSTA-NETMASK:\t",\
          STAconf[1],"\nSTA-GATEWAY:\t",STAconf[2] ,sep='')

# ***********  Setup Router connection end ***************

# **********************************************************
# TCP-Web-Server
# **********************************************************
# ----------------- Server starten --------------------------
if WLANconnect or ownAP:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    s.bind(('', myPort))
    print("Socket established, waiting on port",myPort)
    d.clearAll()
    #          0123456789012345
    d.writeAt("SOCK ESTABLISHED",0,0)
    d.writeAt("LISTENING AT",0,1)
    d.writeAt(myIP+":"+str(myPort),0,2)
    s.settimeout(0.9)
    s.listen(2)

if taste.value()==0:
    print("Mit Flashtaste abgebrochen")
    ledsOff()
    d.writeAt("Abbruch d. User ",0,5)
    sys.exit()

# ------------------- Serverschleife ----------------------
while True:
    try:  # wegen timeout
        r=""
        if WLANconnect or ownAP:
            c, addr = s.accept()
            print('Got a connection from {}:{}\n'.\
                  format(addr[0],addr[1]))
```

```
        request=c.recv(1024)
    else:
        request=input("Kommando:")
        addr="999.999.999.999:99999"
    try:  # decodieren und parsen
        r=request.decode("utf8")
        getPos=r.find("GET /")
        if r.find("favicon")==-1:
            print("***********************************")
            print("Position:",getPos)
            print("Request:")
            print(r)
            print("***********************************")
            pos=r.find(" HTTP")
            if getPos == 0 and pos != -1:
                query=r[5:pos] # nach ? bis HTTP
                print("*********QUERY:{}*********\n\n".\
                      format(query))
                response = web_page(query)
                print("---------------\n",response,\
                      "\n----------------")
                c.send('HTTP/1.1 200 OK\n'.encode())
                c.send('Content-Type: text/html\n'\
                       .encode())
                c.send('Connection: close\n\n'.encode())
                c.sendall(response.encode())
            else:
                print("##########\nNOT HTTP\n###########")
                c.send('HTTP/1.1 400 bad request\n'\
                       .encode())
        else:
            print("favicon request found")
            c.send('HTTP/1.1 200 OK\n'.encode())
    except:  # decodieren und parsen
        request = rawRequest
        c.send('HTTP/1.1 200 OK\n'.encode())
    c.close()
except:  # wegen timeout
    pass

if taste.value()==0:
    print("Mit Flashtaste abgebrochen")
    ledsOff()
    d.writeAt("Abbruch d. User ",0,5)
    sys.exit()
blink(0.05,0.05,1)
```

Abbildung 18: Live aus dem Browser

This is what the website looks like in reality on Google Chrome. Opera offers a similar picture after entering the URL 10.0.1.181:9002. Firefox makes bitches because the makers have made it into their head to have to tame users by making their browser only accept https addresses. But there are alternatives. If things get really bad, you could even write your own front end for the PC with CPython.

Well, I think by Christmas you will have enough to do with the sapling projects. There is sure to be one or the other for everyone. It is important that you enjoy the implementation and that I was able to arouse your interest. In any case, I wish you a nice Advent season.