Projekt Musikbox

Wir bauen und programmieren eine Musikbox mit Hilfe eines Raspberry Pico 2 W.

Hierbei durchlaufen wir folgende Schritte:

- Einleitung/Vorstellung der Musikbox, der Begriffe, des Simulators
- Installation Thonny und Flashen der Firmware
- Zusammenbau

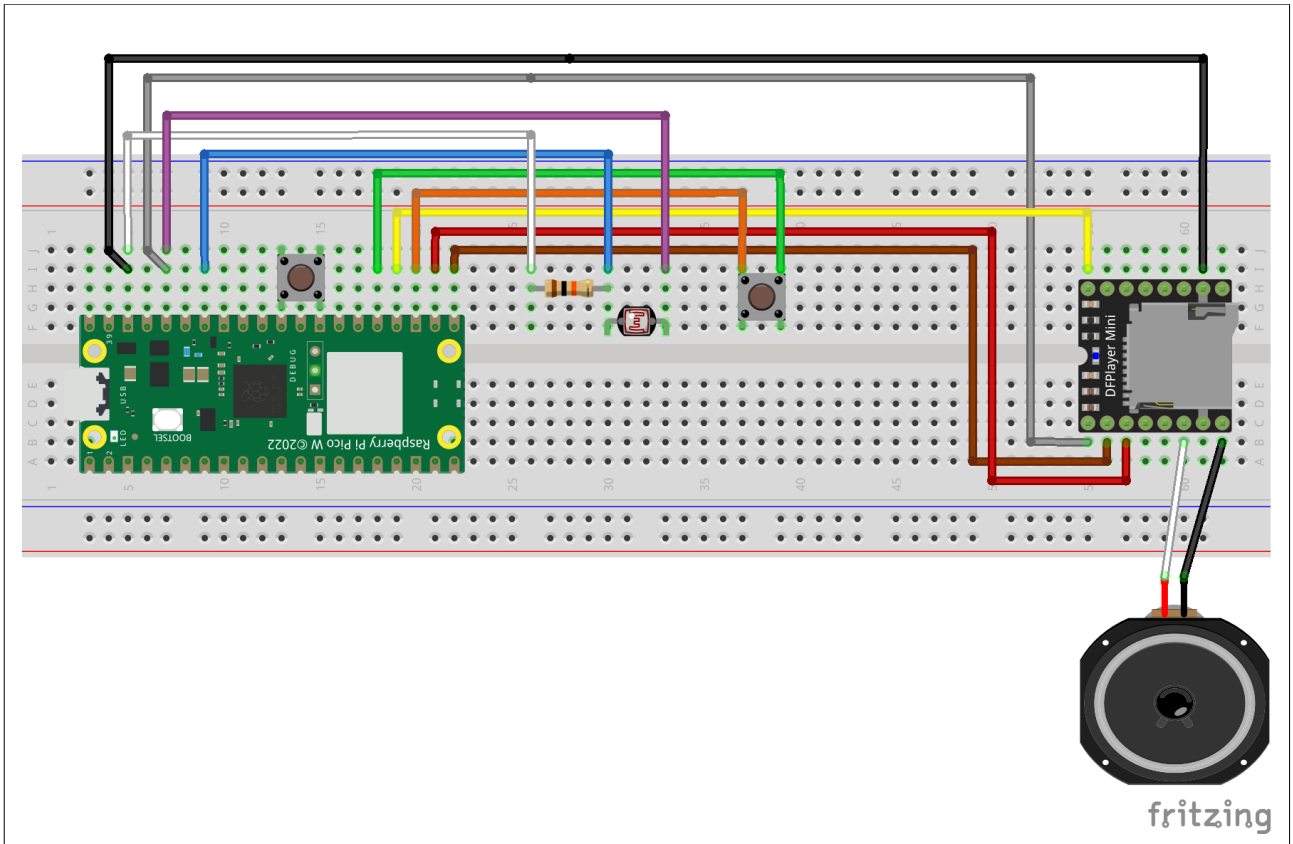
Einleitung

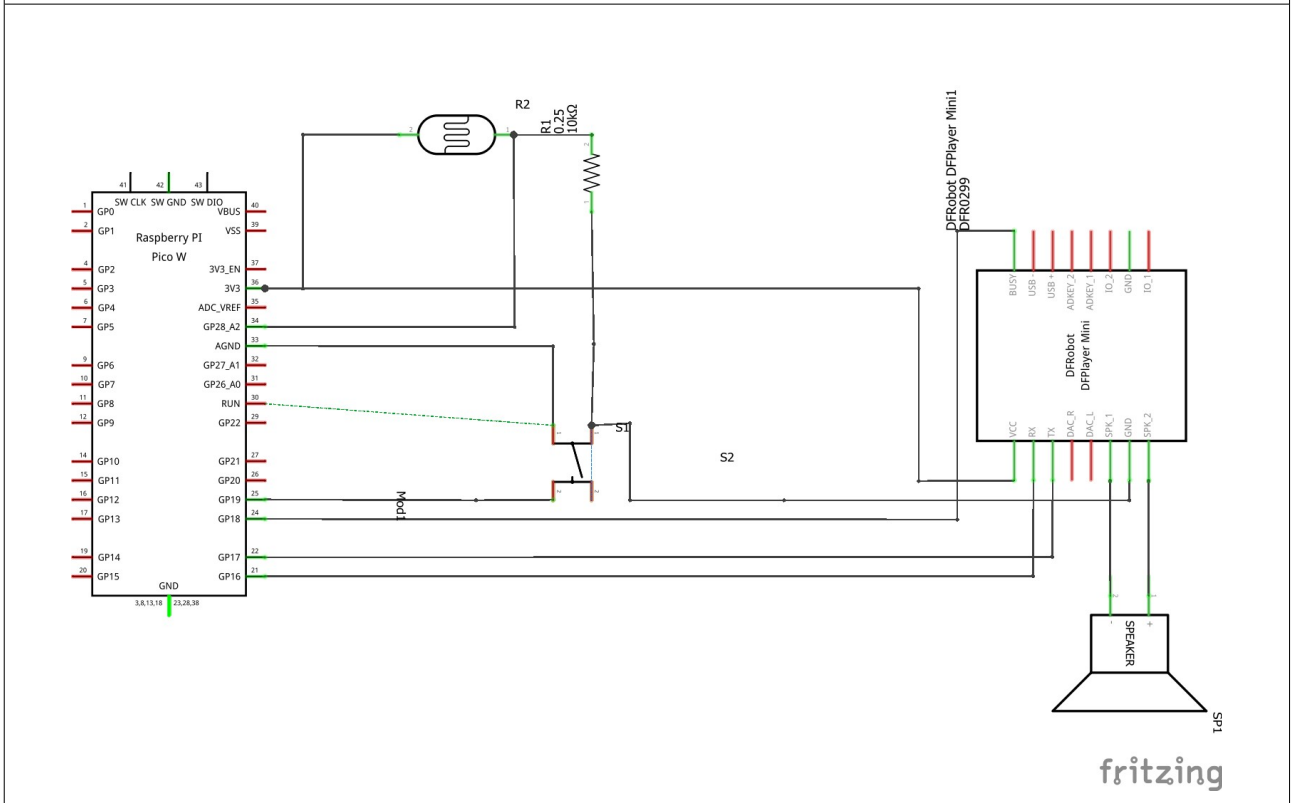
https://www.raspberrypi.com/ -> Documentation -> Microcontrollers -> Pico-series Microcontrollers -> Raspberry Pi Pico 2 W	Pico 2 W Dokumentation
https://magazine.raspberrypi.com/	Maker Magazin
https://pico2.pinout.xyz/	Pinout (Achtung Pico 2 nicht Pico 2 W)
https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299	DFPlayer Dokumentation
https://wokwi.com/	Pico Simulator
https://ttsmp3.com/ai	Text-2-Speech

Installation Thonny und Flashen der Firmware

https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico	Getting Started with Pico mit Bildern
https://magazine.raspberrypi.com/	Maker Magazin
https://www.raspberrypi.com/news/new-book-get-started-with-micropython-on-raspberry-pi-pico/	Book: Get Started with MicroPython on Raspberry Pi Pico

Zusammenbau






```
#DFPlayer mp3 player Driver using UART for Raspberry Pi Pico.
#File picodfplayer.py

from machine import UART, Pin
from utime import sleep_ms, sleep
```

```
#Constants

class DFPlayer():
    UART_BAUD_RATE=9600
    UART_BITS=8
    UART_PARITY=None
    UART_STOP=1

    START_BYTE = 0x7E
    VERSION_BYTE = 0xFF
    COMMAND_LENGTH = 0x06
    ACKNOWLEDGE = 0x01
    END_BYTE = 0xEF
    COMMAND_LATENCY =   500


    def __init__(self, uartInstance, txPin, rxPin, busyPin):
        self.playerBusy=Pin(busyPin, Pin.IN, Pin.PULL_UP)
        self.uart = UART(uartInstance, baudrate=self.UART_BAUD_RATE, tx=Pin(txPin),
rx=Pin(rxPin), bits=self.UART_BITS, parity=self.UART_PARITY, stop=self.UART_STOP)

    def split(self, num):
        return num >> 8, num & 0xFF

    def sendcmd(self, command, parameter1, parameter2):
        checksum = -(self.VERSION_BYTE + self.COMMAND_LENGTH + command +
self.ACKNOWLEDGE + parameter1 + parameter2)
        highByte, lowByte = self.split(checksum)
        toSend = bytes([b & 0xFF for b in [self.START_BYTE, self.VERSION_BYTE,
self.COMMAND_LENGTH, command, self.ACKNOWLEDGE,parameter1, parameter2, highByte,
lowByte, self.END_BYTE]])

        self.uart.write(toSend)
        sleep_ms(self.COMMAND_LATENCY)
        return self.uart.read()

    def queryBusy(self):
        return not self.playerBusy.value()

    #Common DFPlayer control commands
    def nextTrack(self):
        self.sendcmd(0x01, 0x00, 0x00)

    def prevTrack(self):
        self.sendcmd(0x02, 0x00, 0x00)

    def increaseVolume(self):
        self.sendcmd(0x04, 0x00, 0x00)

    def decreaseVolume(self):
        self.sendcmd(0x05, 0x00, 0x00)

    def setVolume(self, volume):
        #Volume can be between 0-30
        self.sendcmd(0x06, 0x00, volume)

    def setEQ(self, eq):
        #eq can be o-5
        #0=Normal
        #1=Pop
        #2=Rock
        #3=Jazz
        #4=Classic
        #5=Base

        self.sendcmd(0x07, 0x00, eq)

    def setPlaybackMode(self, mode):
        #Mode can be 0-3
```

```
        #0=Repeat
        #1=Folder Repeat
        #2=Single Repeat
        #3=Random
        self.sendcmd(0x08, 0x00, mode)

    def setPlaybackSource(self, source):
        #Source can be 0-4
        #0=U
        #1=TF
        #2=AUX
        #3=SLEEP
        #4=FLASH
        self.sendcmd(0x09, 0x00, source)

    def standby(self):
        self.sendcmd(0x0A, 0x00, 0x00)

    def normalWorking(self):
        self.sendcmd(0x0B, 0x00, 0x00)

    def reset(self):
        self.sendcmd(0x0C, 0x00, 0x00)

    def resume(self):
        self.sendcmd(0x0D, 0x00, 0x00)

    def pause(self):
        self.sendcmd(0x0E, 0x00, 0x00)

    def playTrack(self, folder, file):
        self.sendcmd(0x0F, folder, file)

    def playMP3(self, filenum):
        a = (filenum >> 8) & 0xff
        b = filenum & 0xff
        return self.sendcmd(0x12, a, b)#a, b)

    #Query System Parameters
    def init(self, params):
        self.sendcmd(0x3F, 0x00, params)
```

```
#File main.py

from picodfplayer import DFPlayer
from time import sleep, ticks ms
from machine import Pin, ADC, Timer
from sys import exit
from random import randint

#
#C:\...\DATA                    SD Card
#├──dice                        ├──02
#│       dice 1 de.mp3           │     001.mp3
#│       dice 2 de.mp3           │     002.mp3
#│       dice 3 de.mp3           │     003.mp3
#│       dice_4_de.mp3           │     004.mp3
#│       dice_5_de.mp3           │     005.mp3
#│       dice_6_de.mp3           │     006.mp3
#│                               │
#└──motivation                   └──01
#       motivation_1_de.mp3           001.mp3
#       motivation_2_de.mp3           002.mp3
#       motivation_3_de.mp3           003.mp3
#       motivation_4_de.mp3           004.mp3
#       motivation_5_de.mp3           005.mp3
#       motivation_6_de.mp3           006.mp3
#       motivation_7_de.mp3           007.mp3

ldr = ADC(2)
button pin = Pin(19, Pin.IN, Pin.PULL UP)


player = DFPlayer(0, 16, 17, 18)
busy pin = Pin(18, Pin.IN)
led = Pin("LED", Pin.OUT)

flag=0
debounce=500
delta=0
button pin = Pin(19, Pin.IN, Pin.PULL UP)
count=0

def callback(pin):
    global flag, delta
    if (ticks_ms()-delta) > debounce:
        flag= 1
        delta=ticks_ms()

button_pin.irq(trigger=Pin.IRQ_FALLING, handler=callback)

def measure_light(timer):
    global flag
    read = ldr.read_u16()
    if read < 15000:
        led.on()
        flag = 1
    else:
        led.off()
        flag = 0

timer = Timer(period=250, mode=Timer.PERIODIC, callback=measure light)

dice mode = not button pin.value()

while True:
    if flag == 1:
        if dice_mode:
            player.playTrack(2,randint(1,6))
```

```
        else:
            player.playTrack(1,randint(1,7))
        sleep(1)
        while not bool(busy pin.value()):
            sleep(0.1)
        #bug on some picos
        button pin.irq(trigger=Pin.IRQ FALLING, handler=callback)
        flag = 0
```