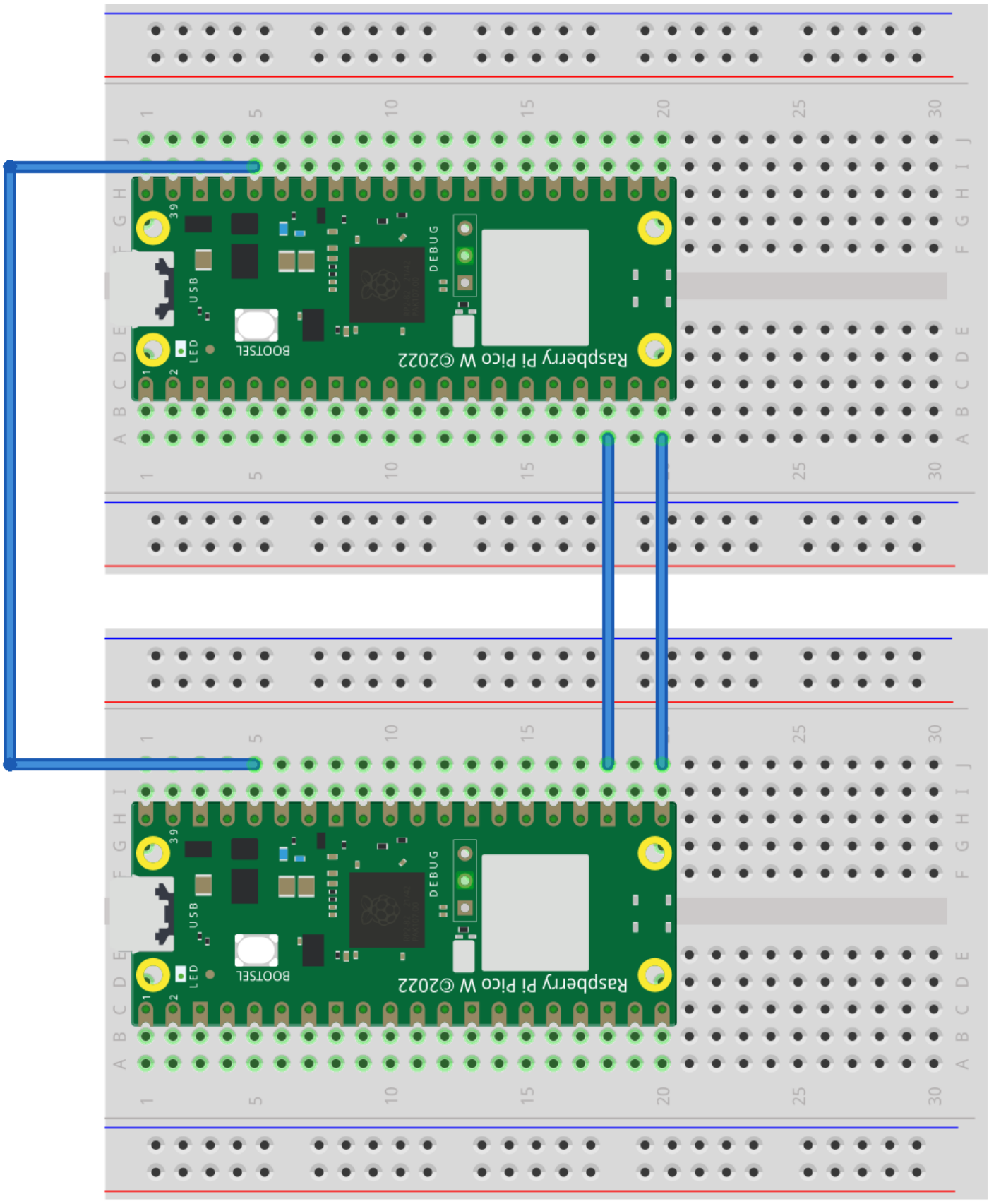
Ampelschaltung mit LEDs

Ampelschaltung

Verwende die LED-Ampel, um eine Ampelschaltung zu programmieren. Schaltet dann mehrere Ampeln zu einer Kreuzung zusammen, indem ihr die Picos miteinander kommunizieren lasst. Dazu müsst ihr einen Pin auf dem Pico, der Befehle erhält, als Eingangspin definieren.


Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
fritzing

Empfange Daten auf Pin 16 und blinke mit der internen LED

```
from machine import Pin


sensor=Pin(16, Pin.IN, Pin.PULL_UP)
led=Pin(25, Pin.OUT)
while True:
  while sensor.value():
    led.value(1)
  led.value(0)
```

Sende Daten mit Pin 15 und blinke mit der internen LED

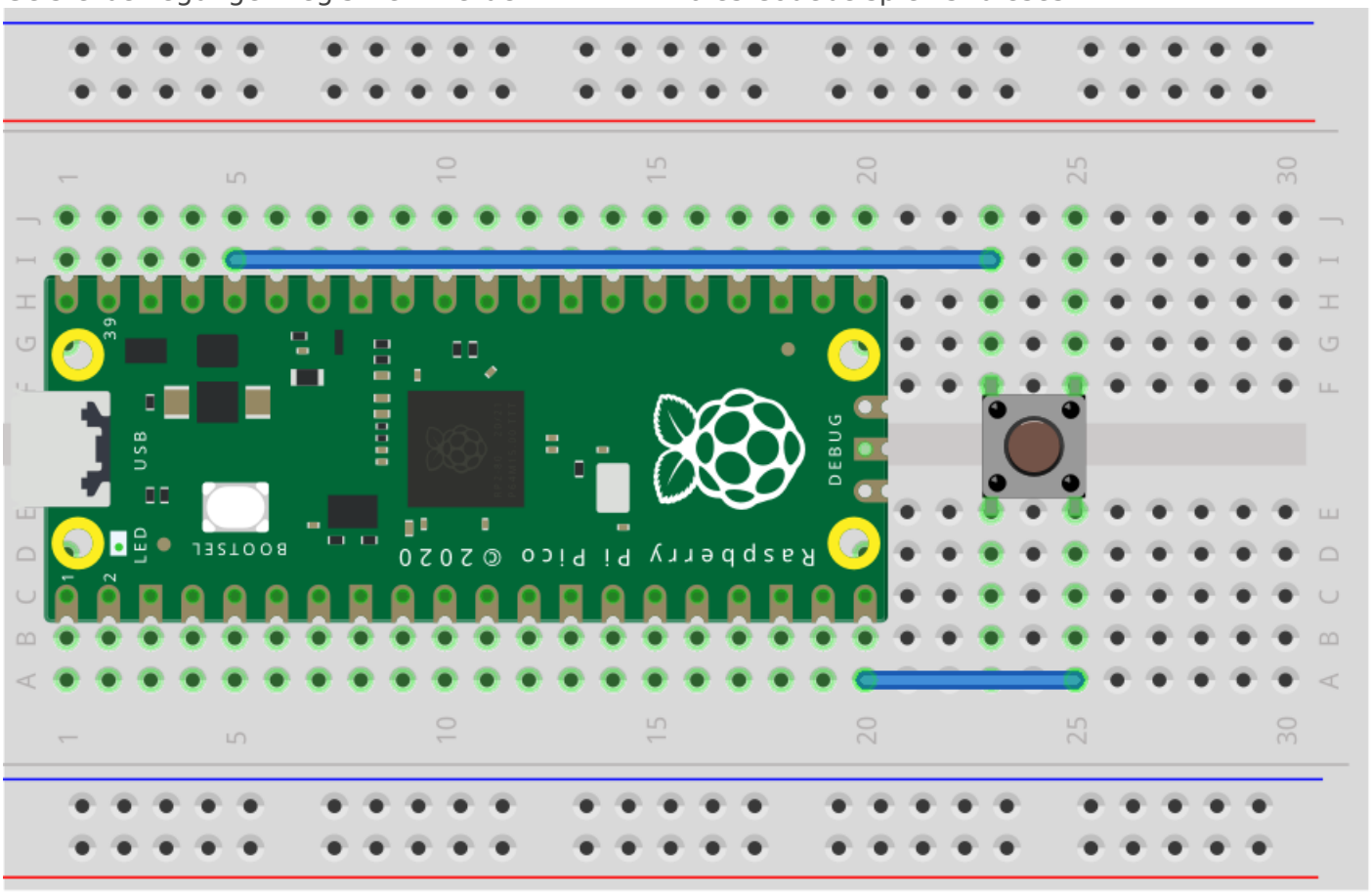
```
import utime
from machine import Pin


#led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
sender=Pin(15,Pin.OUT)
while True:
  led.value(1)
  sender.value(1)
  utime.sleep(1)
  led.value(0)
  sender.value(0)
  utime.sleep(1)
```

Nur einer der beiden Picos muss über USB an Strom angeschlossen werden. Beide Programm werden unter dem Namen "main.py" auf dem Pico abgespeichert, dann laufen die Programme automatisch, sobald die Picos Strom bekommen.

Knopfsteuerung der Ampel

Für die Programmierung eines Ampelknopfes, muss man den Knopf *entprellen*, damit keine Geisterbewegungen registriert werden. Ein minimales Codebeispiel ist dieses:




```
from machine import Pin
import utime
button = Pin(15, Pin.IN, Pin.PULL_DOWN)
pressed = False
num_pressed = 0
last_pressed = 0
DEBOUNCE_WAIT = 100
def button_handler(pin):
    global pressed, num_pressed, last_pressed #mit dem Befehl global teilt man Python mit, dass man die Variabel verwenden möchte, die außerhalb der Funktion initialisiert wurde.
    while utime.ticks_diff(utime.ticks_ms(), last_pressed) < DEBOUNCE_WAIT: # Hier wird verhindert,dass mehrere Auslösungen hintereinander registriert werden.
        pass
    last_pressed = utime.ticks_ms()
    if not pressed:
        while utime.ticks_diff(utime.ticks_ms(), last_pressed) < DEBOUNCE_WAIT:
            pass
```

```
        if pin.value() == 1:
            pressed=True #Damit kann im Programmablauf der Knopfdruck registriert werden.
            num_pressed +=1
            print(pin.value(), "number presses: ", num_pressed)
            last_pressed = utime.ticks_ms()
            pressed=False # Dies hier eher im weiteren Programmablauf verwenden.

button.irq(trigger=Pin.IRQ_RISING, handler=button_handler)

# Hier weiterer Programmablauf
while True:
    pass
```

Programmiergrundkurs in Python

Revision #2
Created 28 November 2023 07:45:27 by Marcus Jacobs
Updated 10 January 2024 07:26:44 by Marcus Jacobs