BOGDAN STĂNESE – ILIE

Tracker Project

An application

CONTENTS

Introduction

General description

Tracker device practically has a GPS module which keeps track of location by connecting itself to satellites around the globe and it returns data in a format similar to a vector, each field being separated by comma, and a GSM module, it receives the parsed data from GPS and by a simple algorithm of decisions, it selects each user who would try to access the device by sending a certain SMS to GSM module, the user can get approximately where the device is.

A practical idea

In some situations everyone wants to know where an important object is, for example, their phone, keys, wallet, laptop, bike, tablet, satchel, bag, purse, car, motorbike. This approach tries to fit in, but not for such small objects, like keys. It can be used inside cars or bags.

Aspects of improvement

The device size can be reduced to be used with many other gadgets. GSP and GSM modules are on separate, but much smaller, PCBs, an idea would have been to have each IC on the main board to decrease total size of device. Power supply part had not been implemented on PCB before the overall project to be done, which later would have come in handy to have all things in one place, and from this drawback, tracker device needs an external supply to power it through USB-B. Program efficiency can be increased by adding layers of error checking.

Theoretical backround

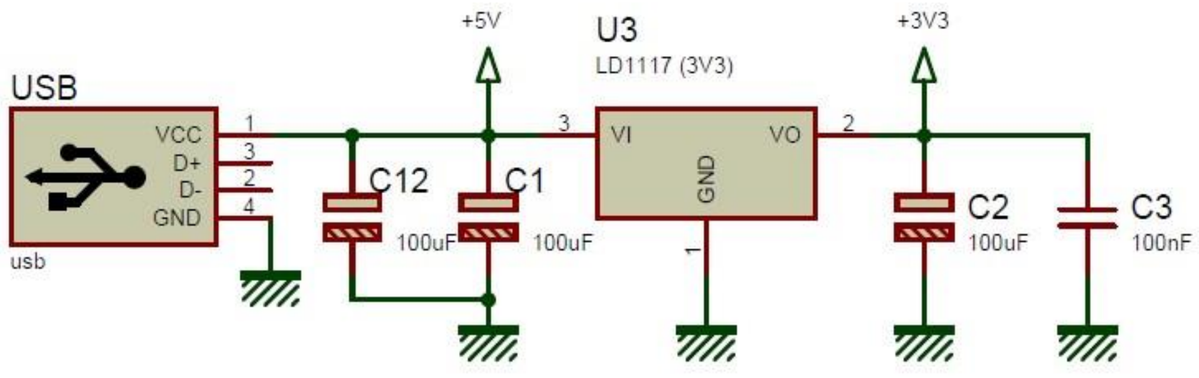
The project came with idea to have basically a working program which would have combined GPS module with a simple OLED display by using a development board. With the help of such boards we can create robust programs for prototyping, controlling and automating machine insides factories. Later on, all in one place circuits from a development board can bring up powerfull features, for example HDMI, audio codec, MIPI interfaces. This lead to the idea of having a personal and trackable device which brought in discussion the GSM module. It integrates with GSP seamlessly through a microcontroller from Raspberry Py, RP2040 which offers multi-threading features, offering a big bonus for the overall program.

Implementation

Hardware

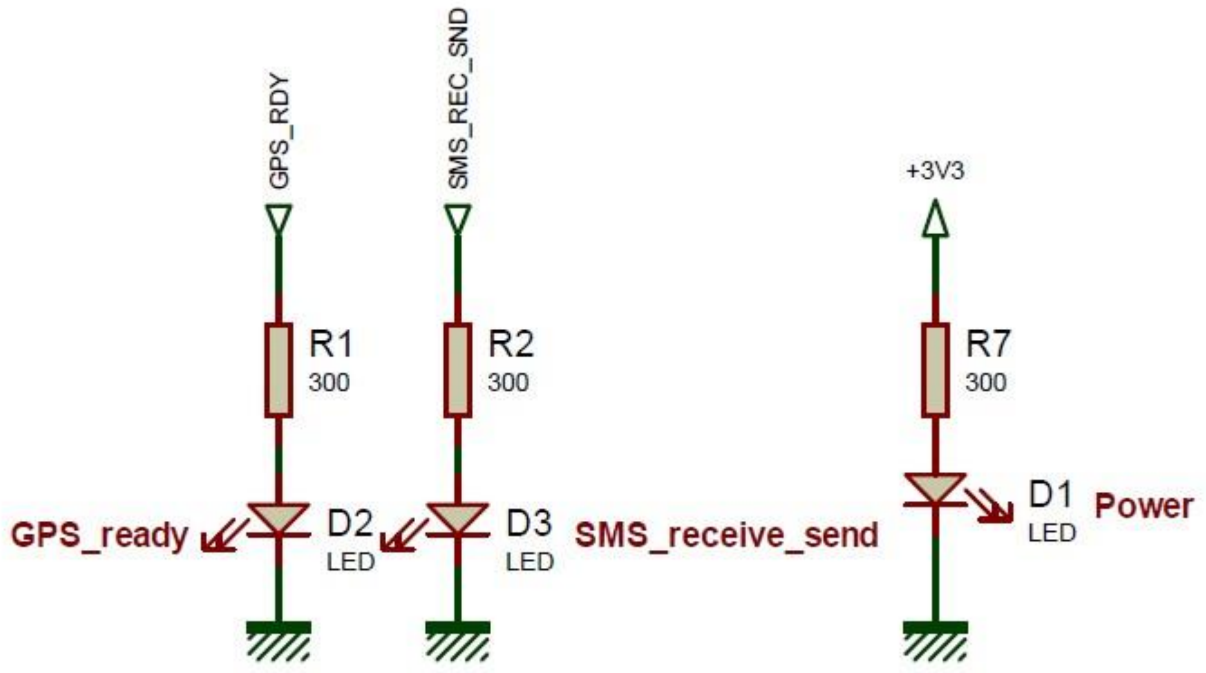
Electrical Schematic

Power supply



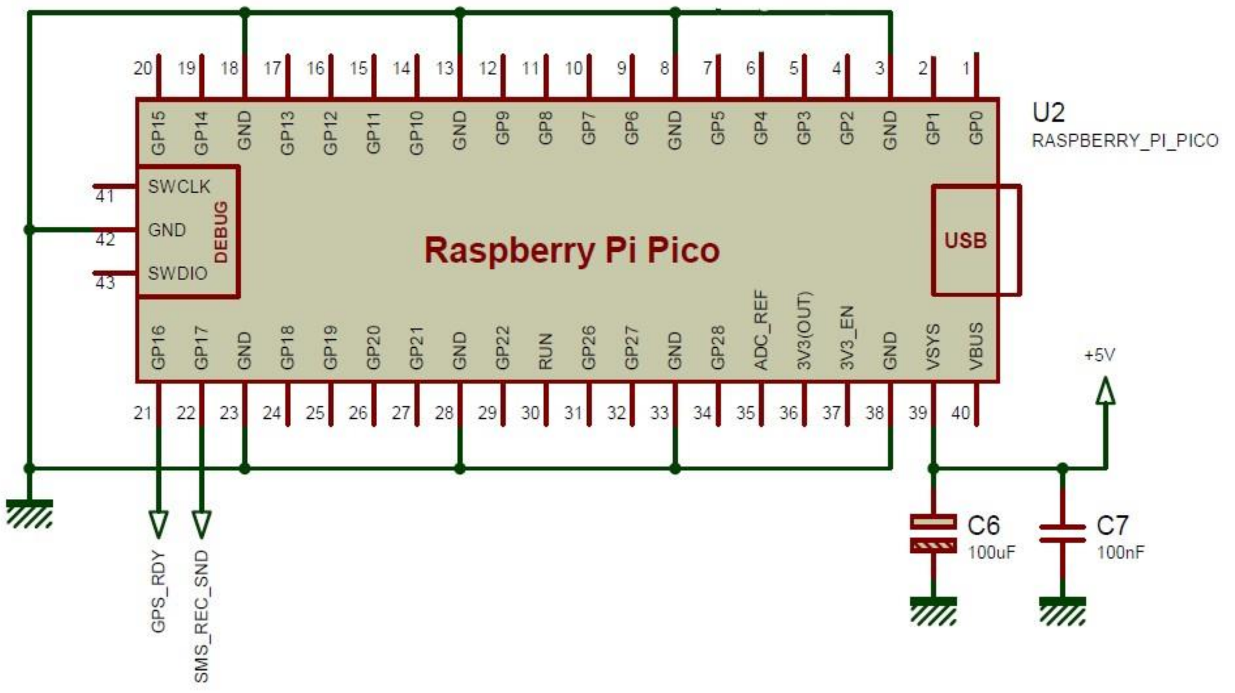
An LDO voltage regulator is used to decrease the voltage from 5V to 3V3 because of GPS module. LD1117 has through-hole pins to make it easier for soldering. Decoupling capacitors help in reducing ΔV_{ripple}.

LEDs



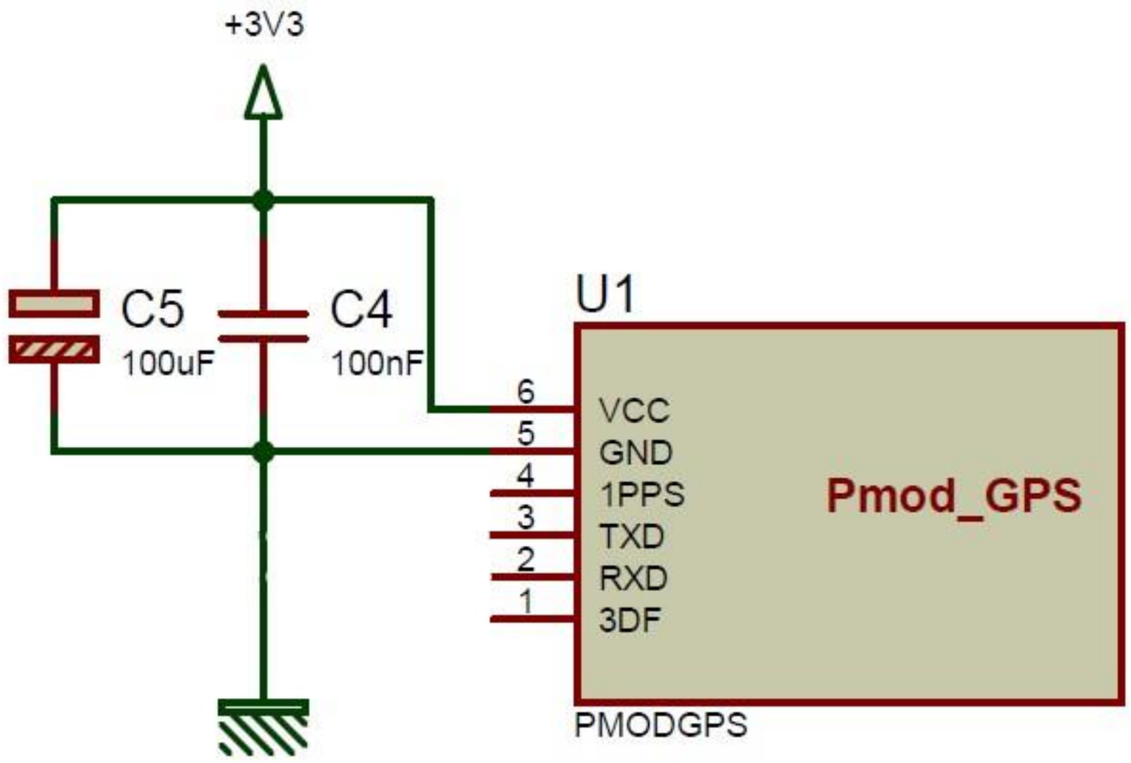
Leds were used to check on runtime of program what it does more precisely. RP2040 generates sufficient current to drive LEDs.

Pi Pico board



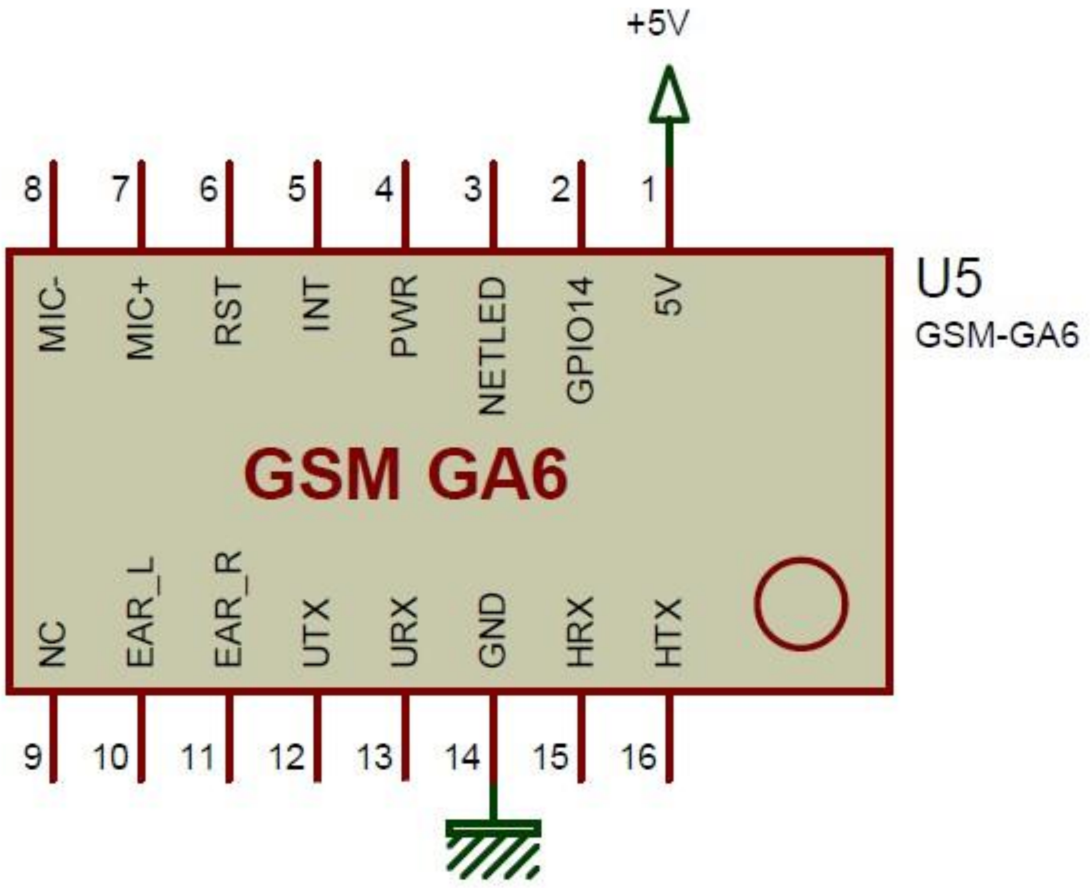
Decoupling capacitors are used for Vsys pin of Pi Pico board to maintain ΔV_{ripple} as low as possible. The board has a switching regulator to obtain 3V3 from 5V to supply RP2040 microcontroller.

GPS module



GPS IC is mounted on a separate PCB and its functionalities can be accessed through an external header placed on its PCB.

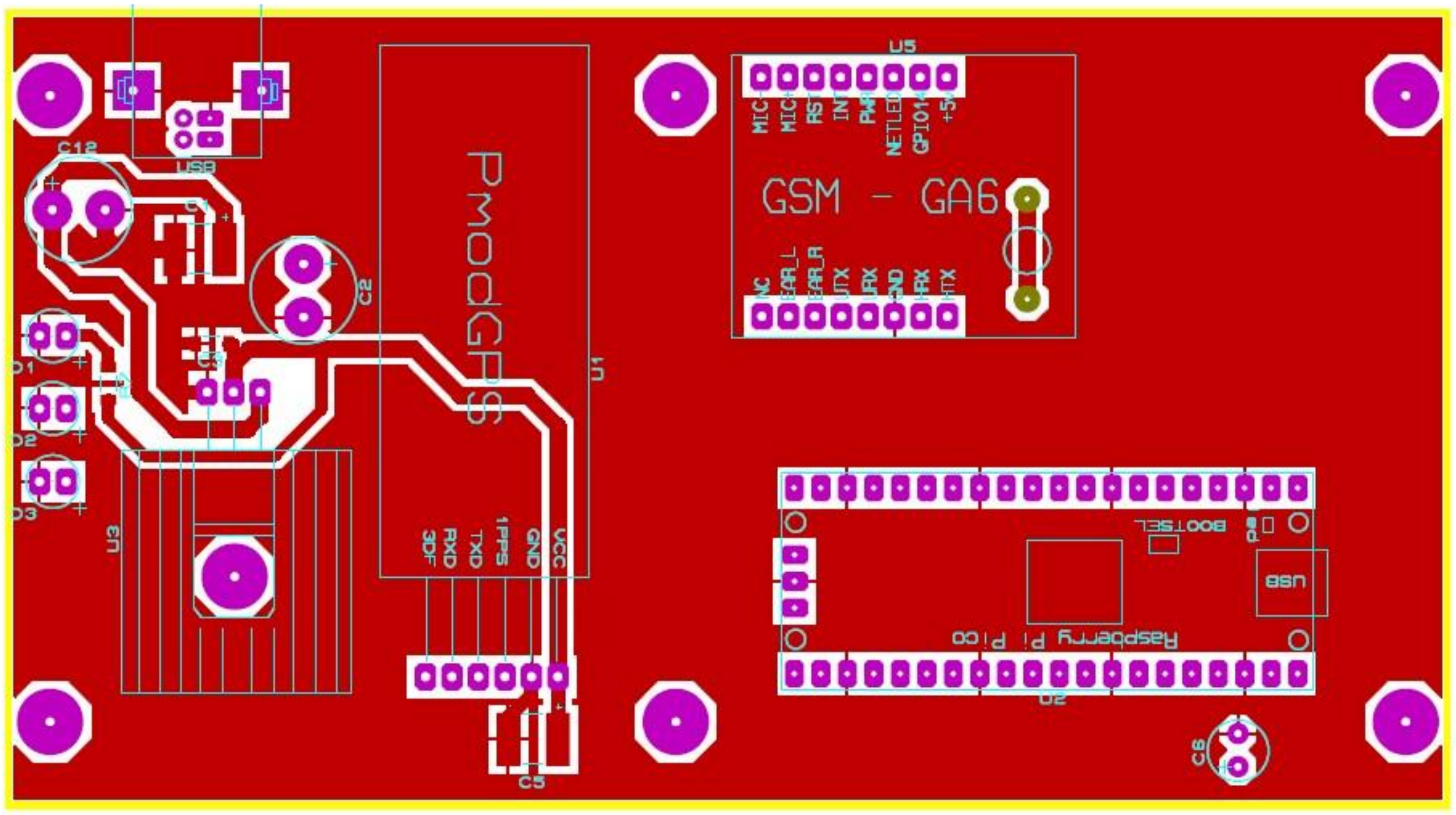
GSM module



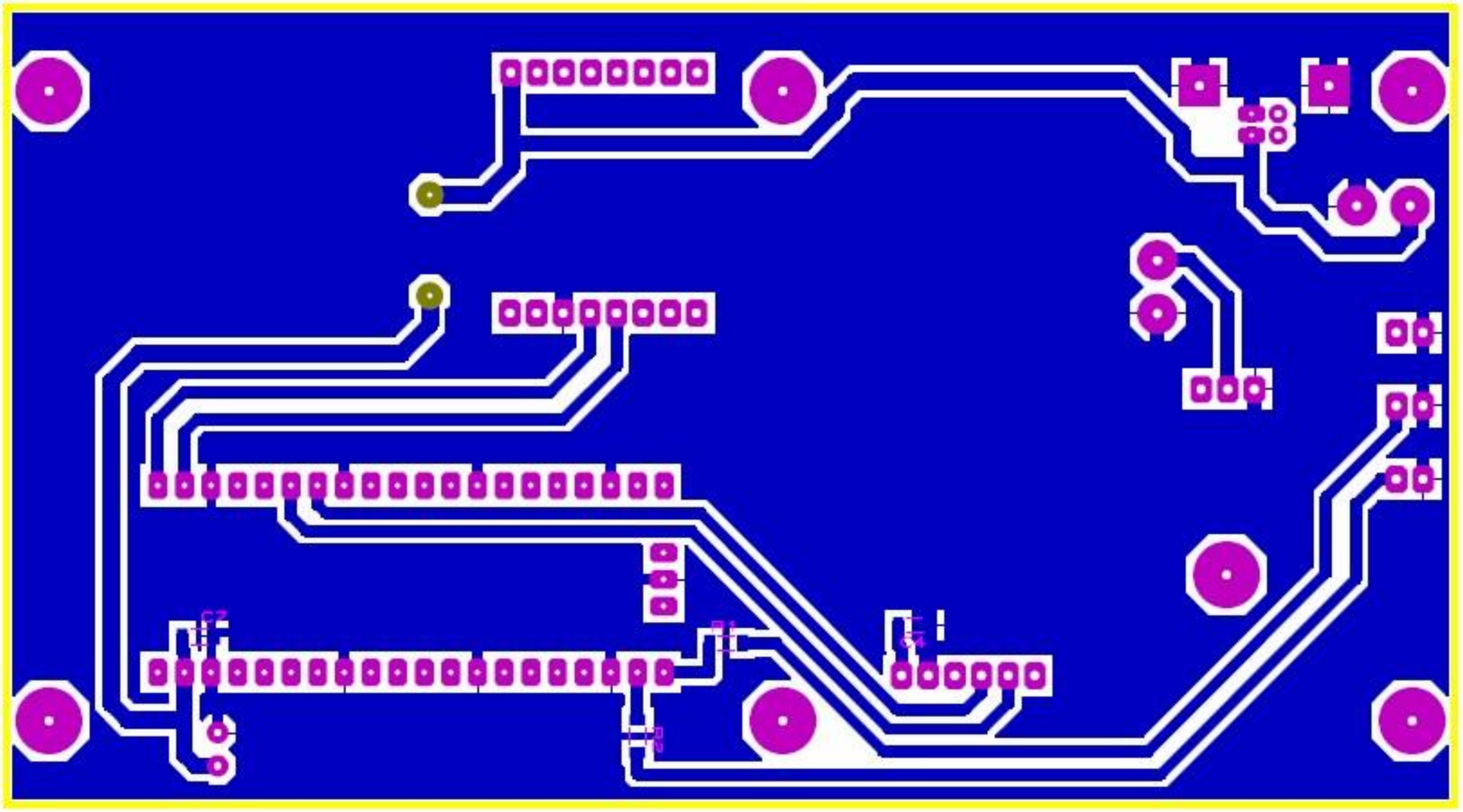
GSM IC as GPS IC is on its own board, it is powerd up by a switching regulator. It responses at AT commands

PCB & ECAD

In project development KiCad 6.0 and Proteus tools were used, but Proteus being a more powerful ECAD tool the final PCB design is implemented on the physical board.



Top part of PCB has almost all components, but there are on bottom part as well to route easily traces. D1 is the power LED, it take 3V3 from output of regulator to indicate the board is powerd up. Vcc of GPS module is routed directly on top to output of regulator. The USB Vcc pin is routed on bottom of PCB, regulator takes 5V from '+' pin of C12, electrolytic capacitor. Under GSM module two Vias connect Vcc of USB to power up Pi Pico board at VSYS pin.



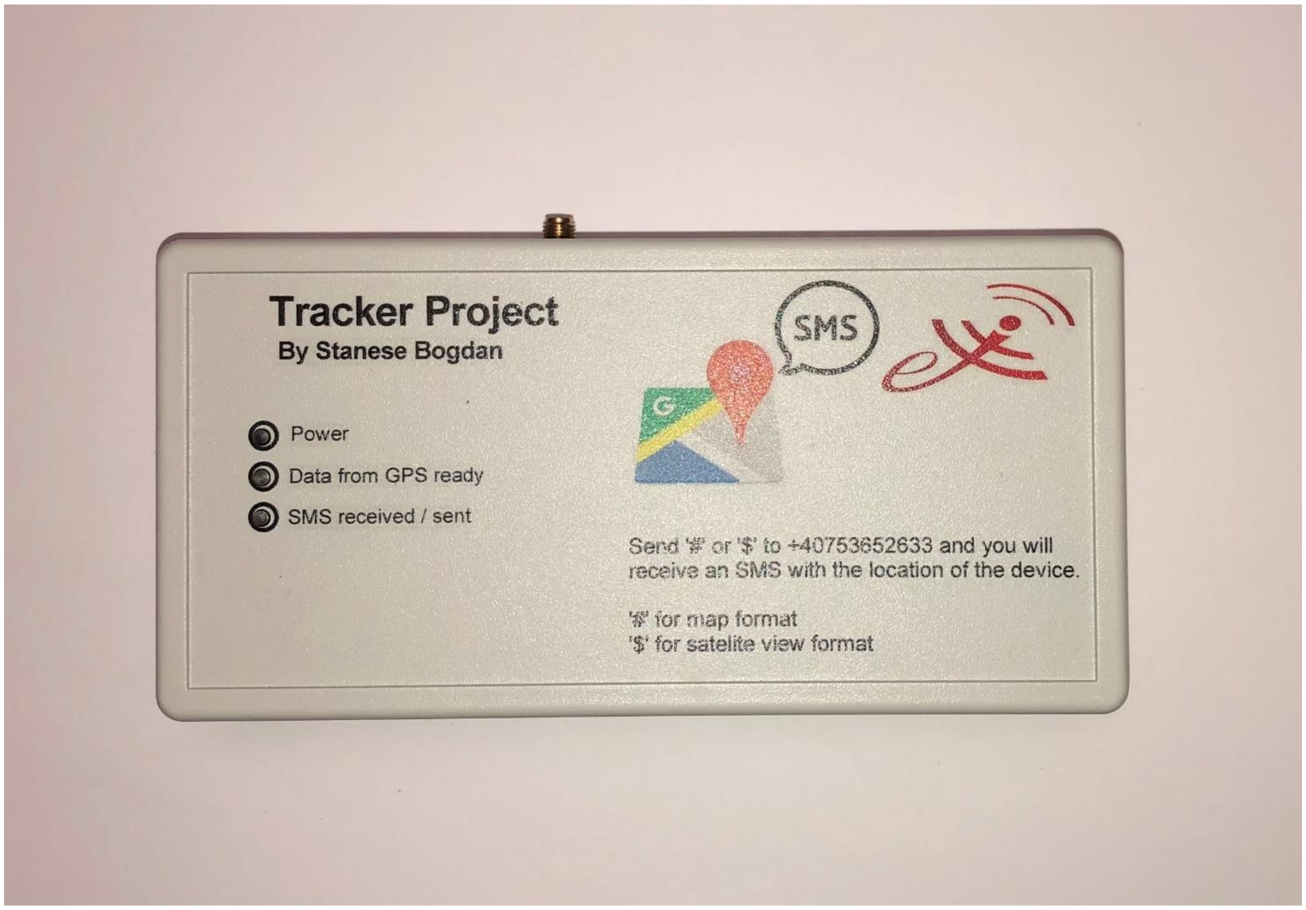
Bottom part of PCB has traces routed for D2 and D3 to GPIO 16, GPIO17 respectively. GPS module's Tx and Rx are routed to GPIO5 and GPIO4, GSM module's Tx and Rx are routed to GPIO1 and GPIO0, these are for the default UART from RP2040. The actual ground planes are created with solder masks for the GND pins.

BOM

Component Count:	19		
Ref	Qnty	Value	Cmp name
U2	1	SC0915	SC0915
C3, C4, C7	3	100nF	Tantalum type-D
C12, C1, C2, C6, C5	5	100uF	Tantalum type-D

D1	1	LED_power	LED
D2	1	LED_gps	LED
D3	1	LED_gsm	LED
USB	1	USB_B	USB_B
R1	1	300Ω	1206
R2, R3	2	300Ω	1206
U3	1	LD1117_TO-220	LD1117_TO-220
U2	1	Pmod_GPS	Pmod_GPS
U1	1	GSM_module	GSM_module

Chasis


Tracker Project
By Stanese Bogdan
Power
Data from GPS ready
SMS received / sent
SMS
Send '#' or '$' to +40753652633 and you will receive an SMS with the location of the device.
'#' for map format
'$' for satelite view format

Software

Programming language & IDE

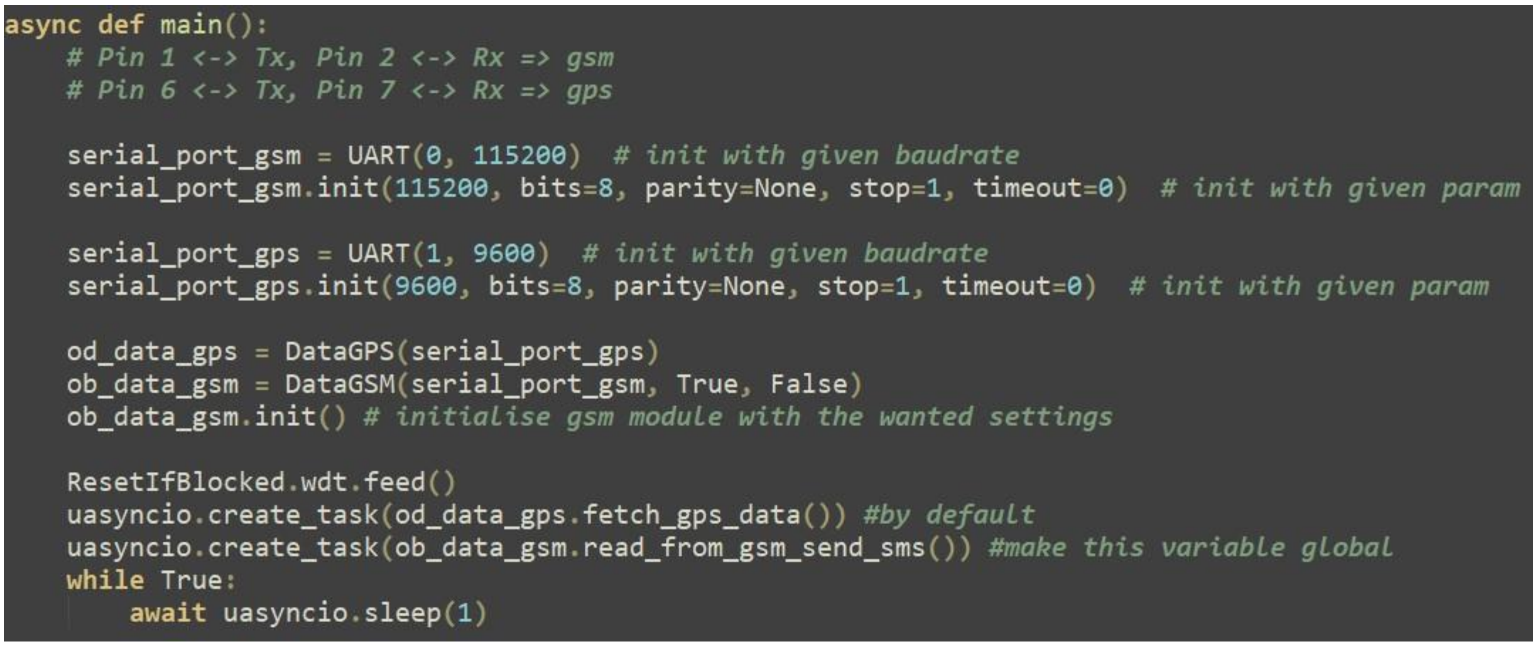
Python is used for the whole program, files with source code are placed on an external flash memory from Pi Pico board and inside the RP2040 microcontroller runs MicroPython interpreter. It is placed by pressing the bootloader button from board before connecting it to an laptop through USB port and realeasing it afterwards. RP2040 will be seen as a Mass Storage Device in which by simply ‘Drag & Drop’, the interpreter can be inserted. PyCharm IDE has MicroPython Plugin, but it does not work perfectly. Thonny IDE was developed for Raspberry Py microcontrollers which works better. It is used to place source code files in the flash memory.

Interpreter

MicroPython has a good documentation and online source code on GitHub. It can be used on various microcontrollers, for example esp8266, stm 32I0, nRF52840, mimxrt, cc3200, RA4M1.

Program

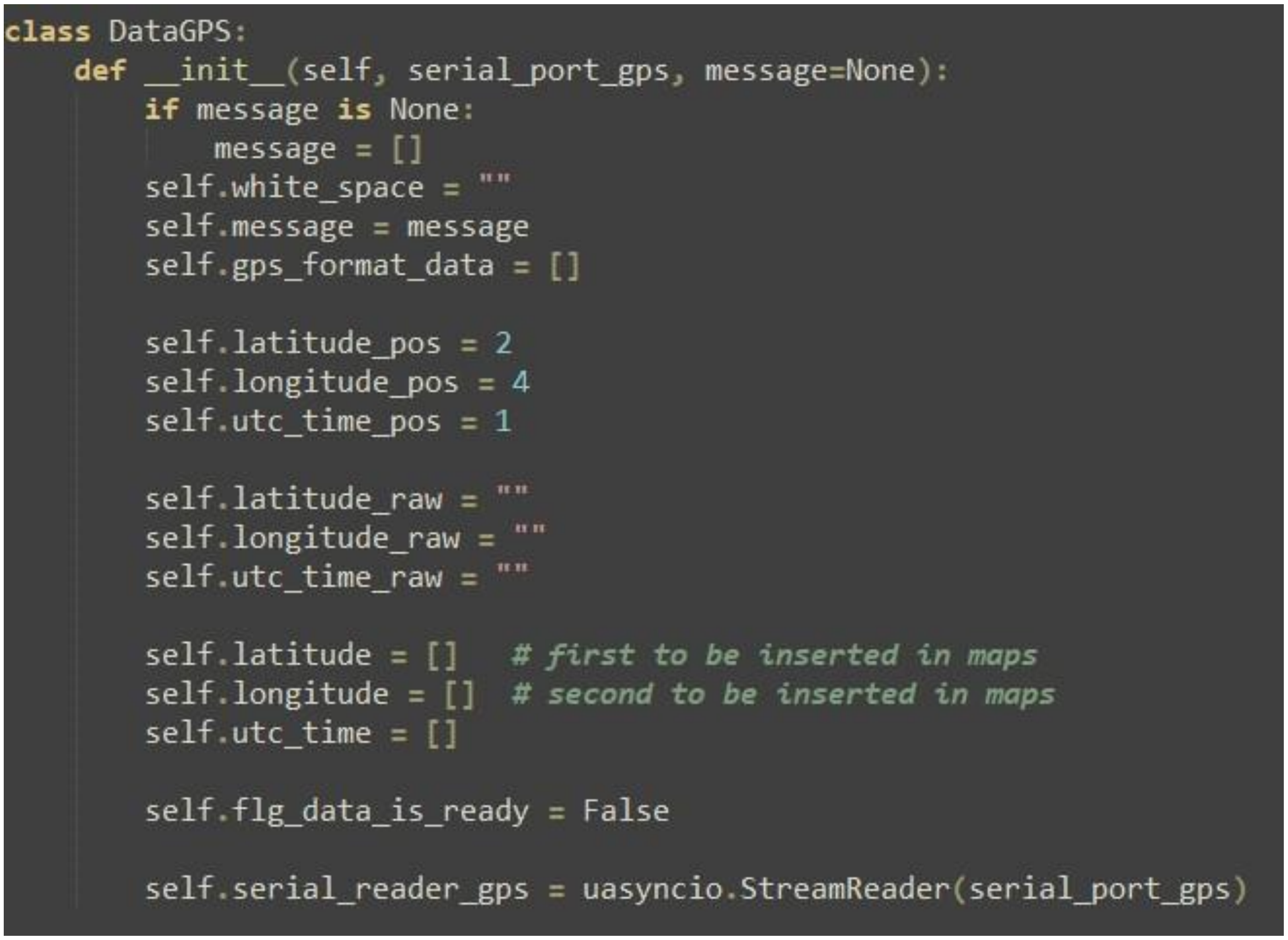
All files are linked through **main.py** file, **gps_linker.py** has a class specific for GPS module with methods which parse data coming from IC, **gsm_linker.py** has a class intended for GSM module with methods for initialization, sending messages/receiving messages. The file **reset_if_blocked.py** uses WatchDog timer from RP2040 if the program has a chance to block. A class for URL formats is located in **url_format.py.**



```
async def main():
    # Pin 1 <-> Tx, Pin 2 <-> Rx => gsm
    # Pin 6 <-> Tx, Pin 7 <-> Rx => gps

    serial_port_gsm = UART(0, 115200)  # init with given baudrate
    serial_port_gsm.init(115200, bits=8, parity=None, stop=1, timeout=0)  # init with given param

    serial_port_gps = UART(1, 9600)  # init with given baudrate
    serial_port_gps.init(9600, bits=8, parity=None, stop=1, timeout=0)  # init with given param

    od_data_gps = DataGPS(serial_port_gps)
    ob_data_gsm = DataGSM(serial_port_gsm, True, False)
    ob_data_gsm.init() # initialise gsm module with the wanted settings

    ResetIfBlocked.wdt.feed()
    uasyncio.create_task(od_data_gps.fetch_gps_data()) #by default
    uasyncio.create_task(ob_data_gsm.read_from_gsm_send_sms()) #make this variable global
    while True:
        await uasyncio.sleep(1)
```

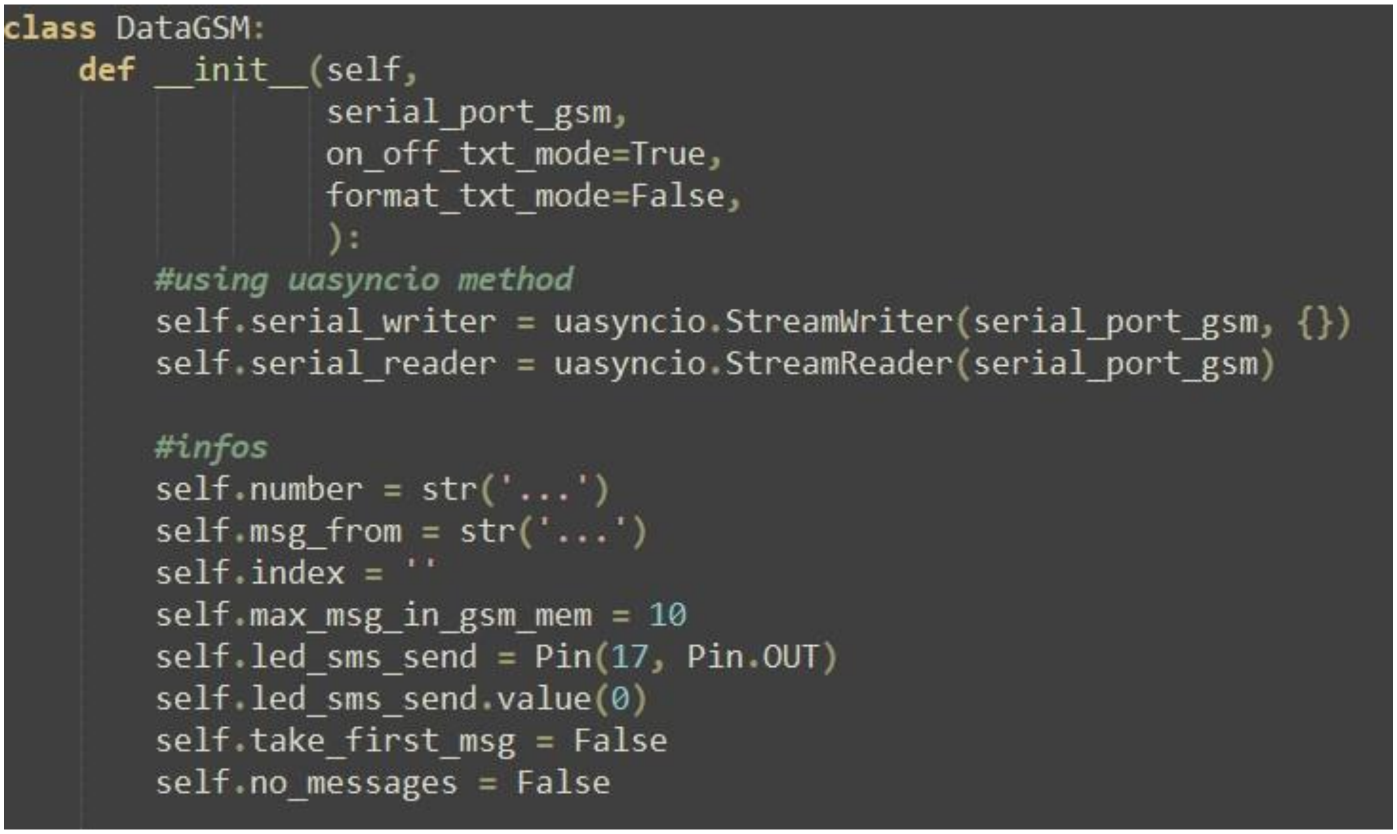
In main function UART module are instantiated with specific baudrate for gsm and gps. Objects for GPS class and GSM class are then created, afterwards GSM is initialized with predefined settings. By using uasyncio module we can create tasks which run independently, **await** keyword is used to transform each function with

async identifier into a generator which is iterated continuously by placing an endless loop at the end of main().



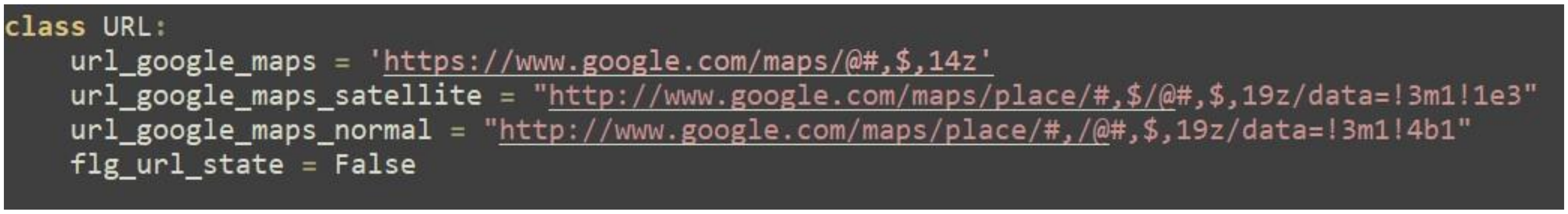
```python
class DataGPS:
    def __init__(self, serial_port_gps, message=None):
        if message is None:
            message = []
        self.white_space = ""
        self.message = message
        self.gps_format_data = []

        self.latitude_pos = 2
        self.longitude_pos = 4
        self.utc_time_pos = 1

        self.latitude_raw = ""
        self.longitude_raw = ""
        self.utc_time_raw = ""

        self.latitude = []   # first to be inserted in maps
        self.longitude = []  # second to be inserted in maps
        self.utc_time = []

        self.flg_data_is_ready = False

        self.serial_reader_gps = uasyncio.StreamReader(serial_port_gps)
```

In DataGPS class, contructor **__init__()** creates initialize attributes which are then used in internal methods. **White_space** is used to check for spaces when a string of data from gps is received to see if the gps has connected to satellites, if not it will put in string spaces where it did not placed real information. In **message** will be what GPS transmits on Tx and then parsed by **parse_message** method. Variables with pos extension represents where the specific items are place in string. **Flg_data_is_ready** will be true if data is ready to be segmented. **Serial_reader_gps** is an object to SteamReader class which lets us to access UART.



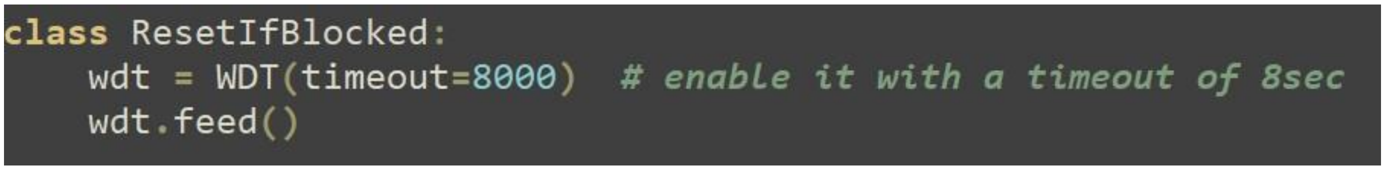
```python
class DataGSM:
    def __init__(self,
                 serial_port_gsm,
                 on_off_txt_mode=True,
                 format_txt_mode=False,
                 ):
        #using uasyncio method
        self.serial_writer = uasyncio.StreamWriter(serial_port_gsm, {})
        self.serial_reader = uasyncio.StreamReader(serial_port_gsm)

        #infos
        self.number = str('...')
        self.msg_from = str('...')
        self.index = ''
        self.max_msg_in_gsm_mem = 10
        self.led_sms_send = Pin(17, Pin.OUT)
        self.led_sms_send.value(0)
        self.take_first_msg = False
        self.no_messages = False
```

GSM class has objects for reading and writing to UART via StreamWriter and StreamReader class from uasyncio module. **Number** will have thelephone number extracted from GSM, **msg_from** will get what the user sent, **index** keeps the index of message the GSM sets it to. **Max_msg_in_gsm_mem** from initialization of GSM, all messages are store in its internal memory, not on SIM card, which are maximum ten. **Take_first_msg** flag is used to take the index just for first message from GSM when it receives the AT command for sending on Tx all its messages. **No_messages** flag signals if the GSM has not a single message.



```
class URL:
    url_google_maps = 'https://www.google.com/maps/@#,$,14z'
    url_google_maps_satellite = "http://www.google.com/maps/place/#,$/@#,$,19z/data=!3m1!1e3"
    url_google_maps_normal = "http://www.google.com/maps/place/#,/@#,$,19z/data=!3m1!4b1"
    flg_url_state = False
```

Attributes in URL class are static because they are not in **__init__()** function, **flg_url_state** signals if the url is ready.



```
class ResetIfBlocked:
    wdt = WDT(timeout=8000)  # enable it with a timeout of 8sec
    wdt.feed()
```

Wdt object connects to WatchDog timer to reset RP2040 if it blocks.

Conclusions

The project suffered various changes until it was finished, for example, because of lack of documenation for GSM module, it took some time to figure out the Tx and Rx pins work at 3V3 and 5V, and the development of program took more than expected. It is interesting that the whole program is saved by WhatchDog timer, it is not yet very clear if GSM blocks UART communication on Pi Pico board or if some signal on board disrupts the communication, when GSM receives a message or it sends one, which will lead the program to block, because source code files are not in RP2040, but in an external flash memory which RP2040 communicates through QSPI with, and the microcontroller cannot access anymore the files. SIM800L module is a better version of GSM IC.

Bibliography

Source code – Interpreter

1. MicroPython - Python for microcontrollers

2. micropython/micropython: MicroPython - a lean and efficient Python implementation for microcontrollers and constrained systems (github.com)

Documentation

1. Overview — MicroPython latest documentation
2. PCBDesignTutorialRevA (alternatezone.com)
3. Explore SnapEDA's Symbol & Footprint Libraries | SnapEDA
4. Basic AT Commands - ESP32 - — ESP-AT User Guide latest documentation (espressif.com)

Examples on “in action” aproaches

1. https://github.com/peterhinch/micropython-async/blob/master/v3/docs/TUTORIAL.md#63-using-the-stream-mechanism
2. Uaysncio v3 Stream Writer Drain Bug · Issue #6621 · micropython/micropython (github.com)
3. micropython-async/auart_hd.py at master · peterhinch/micropython-async (github.com)