# SWA-Lab2. Zadania

Niezależnie od użytego języka programowania i środowiska w którym działamy, taki terminal działa bardzo podobnie (chociaż w C/C++ będzie miał większe możliwości ze względu na lepszą responsywność i użycie przerwań).

## Wskazówki

* Nie generuj nieskończonego BREAK na wspólnej magistrali lub innej formy blokady (stall) - zanim podłączysz zasilanie do BeamKit, połącz DE/nRE z GND (wymuś stan niski, czyli 0, czyli tryb odbiornika). Dopiero gdy zaprogramujesz sterowanie kierunkiem transmisji, przełącz ten sygnał z GND np. na Pin3.
* W ISR nie rób nic co zajmuje czas, w szczególności nie używaj printf.
* Jeśli używasz magistrali, to wyłącz funkcję ECHO terminala (nie odsyłaj kopii znaków). Jeśli 2 lub więcej waszych programów zaczną kopiować każdy odebrany znak, to zastanów się, co się stanie kiedy ktoś wyśle pierwszy znak w eter.
* Z powodów praktycznych po pokazaniu że potraficie wysłać komunikat do terminala RS485 który wyświetlę na projektorze, wykonajcie lokalne połączenia w grupach 2-3 osobowych i w ten sposób testujcie wasze kody. Jeszcze lepiej będzie zrezygnować z stdio na uart0, lub początkowo połączyć uart1 z pinami DEBUG UART, nie będzie to RS485, ale można łatwo testować działanie oprogramowania terminala. Kiedy będziesz w stanie poprawnie dekodować i wykonywać komendy tekstowe, wrócisz do połączeń RS485 z koleżanką lub kolegą. Wciśnięcie przycisków BTN0..2 powinno zapalać LED0..2 u sąsiada i na odwrót.

Zadania do wykonania na zajęciach (niezależnie od języka programowania)

1. komenda help ma wyświetlić listę dostępnych komend wraz z ich opisem.
2. komendy mają mieć możliwość podawania argumentów (np. blink 3 100 1000 - mrugnij diodą 3 razy, 100ms ON, 1000ms OFF) itp.
3. terminal powinien być przygotowany do obsługi poleceń LoRa, NRF24, UDP, HTTP w przyszłości - ten sam dekoder komend bez modyfikacji kodu powinien dać radę współpracować z innymi fizycznymi interfejsami. Jakby nie było, różnica polega tylko na tym że w inny sposób zostanie dostarczony zbiór znaków tekstowych, czy to będzie UART czy pakiet UDP to nie ma znaczenia.
4. Mając kod działającego terminala UART, przerób go na obsługę RS485 (sygnał DE/nRE).
5. Można wykonać test łącząc 2 BeamKit (zgodnie z wiedzą z wykładu) - co sekundę Hello, world z licznikiem i odbieranie danych, ale można też kopiować na RS485 dane z UART-USB (C, MicroPython) lub z konsoli REPL (MicroPython).
6. Dekoder [zdań NMEA](https://aprs.gids.nl/nmea/) - odbiornik GPS jeden w sali, dane rozsyłane będą magistralą RS485. Wydobądź aktualny czas i wyświetl na OLED lub w terminalu.
7. Możesz dodać obsługę WS2812B, jest ich 16 sztuk, wystarczy 1 pin, ale uwaga! sygnał sterujący jest sprzętowo zanegowany (0→1,1→0), więc gotowce z neta nie zadziałają! :) Potrzeba kilku drobnych poprawek kodu, na wykładzie pokażę, jak to można zrobić. W połączeniu z RS485 można z jednej konsoli DJ sterować wszystkimi LEDami w sali - coś jak projekt PIWO. Na kolejne zajęcia (lub jako zadanie domowe) spróbujemy opanować [DMX512](https://en.wikipedia.org/wiki/DMX512) - też RS485, ale już nie UART.

Zasada działania serwera terminala jest następująca:

1. sprawdzaj, czy czasem nie nadszedł nowy znak z zewnątrz (np. z UART). Takie sprawdzanie może być rozwiązane sprzętowo (moduł UART poinformuje CPU o tym zdarzeniu zgłaszając przerwanie), ale też można programowo ciągle “pytać” (ang. poll) moduł UART czy też czasem nie ma dla nas jakichś danych.
2. jeśli odebrany znak to “zwykły” symbol, to dopisz go do wcześniej odebranej kolekcji
3. jeśli odebrany znak to ENTER (CR, LF, CRLF, …) to przekaż do dekodera komend to, co do tej pory zgromadziłeś, po czym opróżnij kolekcję wcześniej zgromadzonych znaków.
4. Dekoder komend: rozmontuj otrzymaną tekstową komendę. Pierwszy element to pewnie nazwa komendy, kolejne elementy to argumenty.
5. Na podstawie pierwszego elementu (treści komendy) wywołaj funkcję która jest z nią powiązana, i przekaż do niej wstępnie zdekodowane argumenty.

## Dancefloor WS2812B

Aby nie wypaliło wam oczu, przed włączeniem LED koniecznie zamontujcie dyfuzor:

Tutaj maksymalna jasność jakiej użyłem wynosiła 10%.

## 

## OLED w MicroPython

Podłączyć trzeba tylko sygnały SDA i SCL - dowolny I2C na Pico, np. GPIO8, GPIO9

import ssd1306

from machine import I2C, Pin

i2c = I2C(0, sda=Pin(8), scl=Pin(9))

display = ssd1306.SSD1306\_I2C(128, 32, i2c)

display.contrast(0)

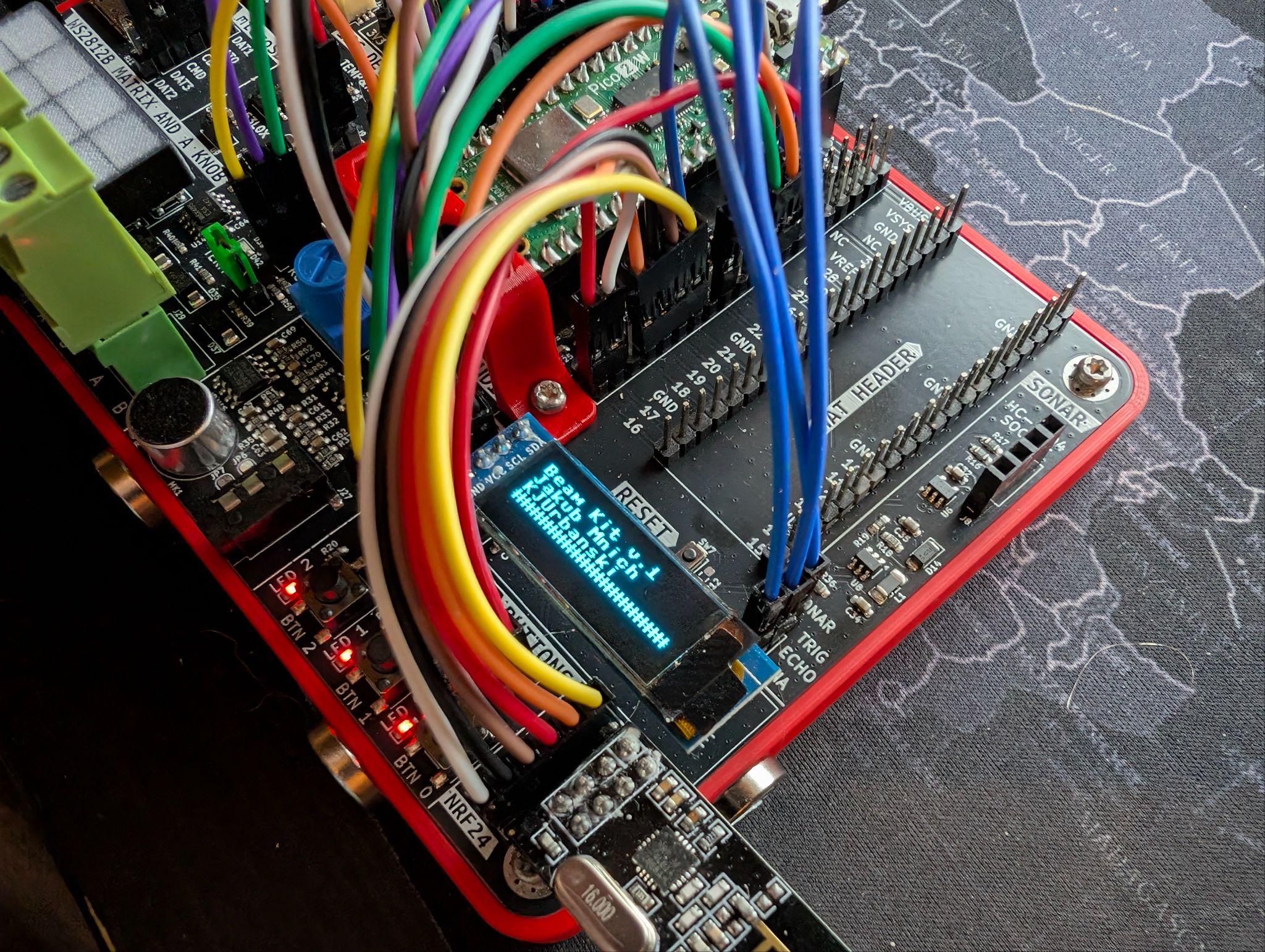
display.text('Beam Kit v.1', 0, 0, 1)

display.text('Jakub Mnich', 0, 8, 1)

display.text('KJUrbanski', 0, 16, 1)

display.text('################', 0, 24, 1)

display.show()



Oto zajawka jak może wyglądać taki terminal z dekoderem komend w języku MicroPython:

import utime

from machine import UART, Pin

uart\_bytes = bytearray()

led = Pin("LED", Pin.OUT)

uart = UART(1, baudrate=115200, tx=Pin(4), rx=Pin(5))

uart.init(bits=8, parity=None, stop=1)

def process\_command(line):

if len(line) == 0: return

args = line.split()

while True:

if uart.any():

data = uart.read()

for c in data:

if c>=32:

uart\_bytes.append(c)

elif c==13:

try:

uart\_str = uart\_bytes.decode('UTF-8')

process\_command(uart\_str)

except: pass

uart\_bytes = bytearray()

else:

led.toggle()

utime.sleep(0.1)

Inne rozwiązanie to sprzętowe przerwania UART-RX. Gdy wykryty zostanie odebrany znak, moduł UART zgłosi przerwanie, co spowoduje tymczasowe wstrzymanie działania bieżącego kodu wykonywanego przez MCU i przejście do funkcji ISR która zajmie się odebranym znakiem.

Ten przykład był przygotowany pod RS485, ale usunąłem fragmenty specyficzne dla RS485 (kod działa jako UART lub RS232 null modem).

void led\_toggle() {

static int toggle = 0;

cyw43\_arch\_gpio\_put(CYW43\_WL\_GPIO\_LED\_PIN, toggle);

toggle = !toggle;

}

Wskazówka: static przed definicją zmiennej lokalnej!

#define N\_TERM\_BUF 32 // Intentionally that short

char term\_buf[N\_TERM\_BUF]; // Here we collect the received data

uint32\_t term\_idx = 0; // term\_buf cursor - current position

void process\_command() {

/// tutaj wasze zadanie - dekodowanie zawartości term\_buf

}

void rs485\_send\_str(const char \*src) {

size\_t len = strlen(src);

uart\_write\_blocking(RS485\_UART\_ID, (const uint8\_t \*)src, len);

uart\_tx\_wait\_blocking(RS485\_UART\_ID); //wait for fifo empty (even though we don’t use fifo)

}

// RX interrupt handler

void on\_uart\_rx() {

while (uart\_is\_readable(RS485\_UART\_ID)) {

led\_toggle(); // Let’s have some fun

uint8\_t ch = uart\_getc(RS485\_UART\_ID);

if(ch == '\r') {

process\_command();

term\_idx = 0; // Prepare for another command reception

term\_buf[term\_idx] = '\0'; // even empty C string must end with zero (\0)

} else {

if(term\_idx<N\_TERM\_BUF-1) {

term\_buf[term\_idx++] = ch;

term\_buf[term\_idx] = '\0'; // ASCIIZ - C string must end with zero (\0)

} else {

ch = '~'; // Notify user: term buffer is full

}

if (term\_echo && uart\_is\_writable(RS485\_UART\_ID)) {

uart\_write\_blocking(RS485\_UART\_ID, (const uint8\_t \*) &ch, 1); // Send it back

uart\_tx\_wait\_blocking(RS485\_UART\_ID); //wait fifo empty (even we don’t use it)

}

}

}

}

// Set up UART

void rs485\_init(uart\_inst\_t \*uart, uint baudrate,

uint tx\_pin, uint rx\_pin,

uint data\_bits, uint stop\_bits, uart\_parity\_t parity) {

uart\_init(uart, baudrate);

gpio\_set\_function(tx\_pin, GPIO\_FUNC\_UART);

gpio\_set\_function(rx\_pin, GPIO\_FUNC\_UART);

uart\_set\_hw\_flow(uart, false, false); // flow control CTS/RTS, we don't want these

uart\_set\_format(uart, data\_bits, stop\_bits, parity); // Set our data format

uart\_set\_fifo\_enabled(uart, false); // Turn off FIFO's - we want to do this char by char

int UARTx\_IRQ = uart == uart0 ? UART0\_IRQ : UART1\_IRQ; // Select interrupt for the UART

irq\_set\_exclusive\_handler(UARTx\_IRQ, on\_uart\_rx); // Set up and enable the interrupt handlers

irq\_set\_enabled(UARTx\_IRQ, true);

uart\_set\_irq\_enables(uart, true, false); // Now enable the UART to send interrupts - RX only

}

int main() {

stdio\_init\_all();

if (cyw43\_arch\_init()) { // Initialise the Wi-Fi chip

printf("Wi-Fi init failed\n");

return -1;

}

rs485\_init(RS485\_UART\_ID, RS485\_BAUD\_RATE,   
 RS485\_TX\_PIN, RS485\_RX\_PIN,   
 RS485\_DATA\_BITS, RS485\_STOP\_BITS, RS485\_PARITY);

int32\_t cnt = 0;

while (true) {

static char buf[100];

sprintf(buf, "%d Hi, embedded!\r\n", cnt++);

rs485\_send\_str(buf);

sleep\_ms(1000);

}

}

W main nie ma śladu odbierania znaków, jest tylko pętla while wysyłająca co sekundę napis “Hello, embedded!” poprzedzony wartością licznika.

## Podłączenie GPS | BeamKit

**GPS | BeamKit**

**=============**

GND | GND

Vcc | 3V3 (NIE 5V!)

TX | dowolny pin UART RX, np. GPIO5

Konfiguracja UART odbiornika GPS: 9600,8N1

W tym zadaniu wystarczy że podłączysz na kilka minut moduł GPS i zbierzesz kilkanaście zdań NMEA w terminalu. Takie zdania (każdy będzie miał inne) wczytane z pliku lub w formie napisów możesz na spokojnie przetwarzać w C lub MP.

