Podstawy Systemów Mikroprocesorowych Laboratorium

Laboratorium 4: UART i GPS — komunikacja szeregowa

Autorzy: Dariusz Tefelski, Angelika Tefelska

Spis treści

1 Wprowadzenie **1**

2 Wyposażenie stanowiska **1**

3 Zadanie 1: Pierwsza transmisja UART **2**
3.1 Połączenie RPi Pico A z RPi Pico B 2

4 Zadanie 2: Co naprawdę jest przesyłane linią TX? **5**

5 Zadanie 3: Komunikacja dwukierunkowa i prosty protokół **6**

6 Zadanie 4: Moduł GPS jako urządzenie UART **8**
6.1 Komunikaty NMEA 10

7 Zadanie końcowe: Most GPS–UART–RPi Pico B **12**

8 Podsumowanie **13**

1 Wprowadzenie

Cel ćwiczenia

Celem ćwiczenia jest poznanie asynchronicznej komunikacji szeregowej UART oraz wykorzystanie jej do wymiany danych pomiędzy dwoma mikrokontrolerami i do odbioru informacji z modułu GPS.

W poprzednich laboratoriach Raspberry Pi Pico 2 W pracowało głównie samodzielnie: odczytywało wejścia, wykonywało obliczenia i sterowało wyjściami. W praktycznym systemie mikroprocesorowym mikrokontroler bardzo często wymienia dane z innymi mikrokontrolerami, czujnikami, modułami komunikacyjnymi lub komputerem.

Na tych zajęciach wykorzystamy fakt, że na płytce dydaktycznej znajdują się **dwa moduły Raspberry Pi Pico 2 W**. Najpierw połączymy je ze sobą, a następnie jeden z mikrokontrolerów będzie odbierał dane z modułu GPS.

2 Wyposażenie stanowiska

Do wykonania ćwiczenia potrzebne są:

- płytka dydaktyczna z dwoma modułami Raspberry Pi Pico 2 W,
- dwa przewody USB,
- przewody połączeniowe,
- moduł GPS Waveshare LC76G GNSS,
- jedna z diod LED oraz potencjometr POT1 na płytce dydaktycznej,
- Analog Discovery II z programem WaveForms.

Uwaga

Przed rozpoczęciem zajęć podłącz moduł GPS: VCC → 5V, GND → GND oraz TXD → GP5, ponieważ GPS potrzebuje czasu by złapać sygnał.

Dwa moduły RPi Pico na jednej płytce

W dalszej części instrukcji moduły będą oznaczane:

- **RPi Pico A** — pierwszy mikrokontroler, pełniący początkowo rolę nadajnika,
- **RPi Pico B** — drugi mikrokontroler, pełniący początkowo rolę odbiornika.

Każdy moduł może być programowany niezależnie przez własne złącze USB.

3 Zadanie 1: Pierwsza transmisja UART

UART (*Universal Asynchronous Receiver/Transmitter*) jest interfejsem szeregowym. Dane są przesyłane kolejno bit po bicie. Podstawowe linie to:

- `TX` (*transmit*) — linia nadawcza,
- `RX` (*receive*) — linia odbiorcza.

Ponieważ UART jest interfejsem asynchronicznym, urządzenia nie przesyłają osobnego sygnału zegarowego. Nadajnik i odbiornik muszą więc zostać skonfigurowane z taką samą szybkością transmisji.

W zadaniach 1–3, pomiędzy dwoma Pico, przyjmiemy konfigurację (moduł LC76G w zadaniu 4 będzie pracował z szybkością 115200 bit/s):

9600 bit/s, 8 bitów danych, brak parzystości, 1 bit stopu

Taki zapis często skraca się do **9600 8N1**.

Co oznacza *baud rate*?

Baud rate określa liczbę symboli przesyłanych w ciągu jednej sekundy. W typowej transmisji UART każdy symbol odpowiada jednemu bitowi, dlatego dla UART wartości podawane w bodach i w bitach na sekundę są liczbowo takie same. Oznacza to, że ustawienie `baudrate=9600` odpowiada transmisji 9600 bitów na sekundę.
Nie oznacza to jednak przesyłania 9600 bajtów na sekundę. W konfiguracji 8N1 jeden znak składa się z 1 bitu startu, 8 bitów danych i 1 bitu stopu, czyli łącznie z 10 bitów. Maksymalna szybkość przesyłania znaków wynosi więc w przybliżeniu:

$$\frac{9600}{10} \approx 960 \text{ znaków/s}. \tag{1}$$

Jak wygląda pojedynczy znak UART?

W stanie spoczynku linia TX znajduje się w stanie wysokim. Transmisja jednego znaku obejmuje:

1. bit startu — stan niski,
2. osiem bitów danych,
3. bit stopu — stan wysoki.

Przy szybkości 9600 bit/s czas pojedynczego bitu wynosi około:

$$t_b = \frac{1}{9600} \approx 104\,\mu\text{s}. \tag{2}$$

3.1 Połączenie RPi Pico A z RPi Pico B

Na początek wykorzystamy tylko jeden kierunek transmisji:

RPi Pico A	RPi Pico B
GP0 — UART0 TX	GP1 — UART0 RX
GND	GND

Uwaga

Linie nadajnika i odbiornika łączymy **na krzyż**: TX jednego urządzenia musi trafić do RX drugiego. Nie łącz ze sobą dwóch wyjść TX.

Zadanie 1.1 — RPi Pico A wysyła, RPi Pico B odbiera

Najpierw zaprogramuj **RPi Pico A**. **Zapisz poniższy program na mikrokontrolerze jako main.py, aby uruchamiał się automatycznie po restarcie.**

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

uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1), bits
   =8, parity=None, stop=1)

counter = 0

while True:
    message = "CNT:{}\n".format(counter)
    uart.write(message.encode())
    counter += 1
    sleep(0.5)
```

Jak działa linia tworząca komunikat?

Napis `CNT:{}\n` jest szablonem tekstu. Para nawiasów klamrowych {} oznacza miejsce, w które metoda `format()` wstawi przekazaną wartość.
Dla przykładu, gdy:

```
counter = 12
```

wyrażenie:

```
''CNT:{}\n''.format(counter)
```

utworzy tekst odpowiadający `CNT:12` zakończony znakiem nowej linii `\n`. Dzięki temu w każdej iteracji pętli wysyłana jest aktualna wartość licznika.

Następnie przełącz Thonny na **RPi Pico B** i uruchom:

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

```

```
uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1), bits
   =8, parity=None, stop=1,
            timeout=100, timeout_char=20)

while True:
    if uart.any():
        line = uart.readline()
        if line is not None:
            try:
                text = line.decode("ascii").strip()
                print(text)
            except UnicodeError:
                print("Blad dekodowania: sprawdz baudrate"
                   )

    sleep_ms(10)
```

Sprawdź:

1. czy licznik zwiększa się kolejno,
2. co stanie się po odłączeniu przewodu TX–RX,
3. co stanie się, jeżeli na RPi Pico B ustawisz inną szybkość transmisji, np. 4800 bit/s.

Podstawowe metody klasy `UART`

W dalszej części laboratorium wielokrotnie wykorzystujemy trzy podstawowe metody:

- `uart.write(data)` — wysyła podane bajty przez linię TX; w przypadku tekstu najpierw używamy zwykle `encode()`,
- `uart.any()` — informuje, czy w buforze odbiorczym znajdują się dane; wartość różna od zera oznacza, że można próbować je odczytać,
- `uart.readline()` — odczytuje dane do znaku końca linii `\n`; po upływie czasu oczekiwania może zwrócić także fragment linii, a przy braku danych — `None`.

W prostym przykładzie ustawiono czasy oczekiwania, aby ograniczyć dzielenie krótkich komunikatów. Nie gwarantuje to jednak odbioru całej linii. W zadaniach 3–5 stosuj buforowanie do znaku końca linii, pokazane w zadaniu 4.1. Metoda `read()` pobiera dostępne bajty (lub zwraca `None`); nie rozpoznaje granic komunikatów.

W przykładzie schemat odbioru ma postać: najpierw sprawdzamy `uart.any()`, a następnie próbujemy pobrać komunikat za pomocą `uart.readline()`.

`str`, `bytes`, `encode()` i `decode()`

Tekst używany w programie Python jest obiektem typu `str`. UART przesyła jednak **bajty**.
Dlatego przed wysłaniem tekst można zamienić na bajty:

```
message.encode()
```

Natomiast dane odebrane z UART mają postać `bytes`. Aby ponownie otrzymać tekst używamy:

```
line.decode()
```

Metoda `strip()` usuwa z początku i końca tekstu znaki białe, w tym znak końca linii `\n`.

Po co dodajemy `\n`?

UART przesyła jedynie kolejne bajty i sam nie wie, gdzie kończy się komunikat. W tym ćwiczeniu przyjmujemy prostą zasadę: **każdy komunikat kończy się znakiem nowej linii**.
Dzięki temu odbiornik może użyć funkcji `readline()`, która odczytuje dane do momentu wystąpienia `\n`.

4 Zadanie 2: Co naprawdę jest przesyłane linią TX?

Do tej pory obserwowaliśmy jedynie tekst pojawiający się po stronie odbiornika. Teraz sprawdzimy fizyczny przebieg transmisji.

Pomiar 2.1 — UART w Analog Discovery II

Pozostaw połączenie RPi Pico A → RPi Pico B. Podłącz wejście cyfrowe Analog Discovery II do linii `GP0` RPi Pico A oraz połącz masę analizatora z GND płytki. W programie WaveForms uruchom narzędzie **Logic**. Dodaj dekoder UART i ustaw parametry:

- szybkość: 9600 bit/s,
- 8 bitów danych,
- brak parzystości,
- 1 bit stopu.

Zaobserwuj przesyłane komunikaty i porównaj je z tym, co pojawia się w Thonny po stronie RPi Pico B.

Zadanie 2.1 — Pomiar czasu pojedynczego bitu

Na RPi Pico A uruchom na chwilę program wysyłający regularnie pojedynczy znak `U` (poprzedni program zakomentuj gdyż będzie potrzebny w dalszej części laboratorium):

```
1 from machine import UART, Pin
```

```
from utime import sleep_ms

uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))

while True:
    uart.write(b"U")
    sleep_ms(20)
```

Znak `U` ma kod ASCII `0x55`, dzięki czemu w jego bitach danych występują naprzemiennie zera i jedynki.
Za pomocą kursorów w WaveForms:

1. zmierz czas pojedynczego bitu,
2. oblicz z niego szybkość transmisji,
3. porównaj wynik z wartością 9600 bit/s,
4. wskaż bit startu i bit stopu.

Po wykonaniu pomiaru przywróć na RPi Pico A program z zadania 1.1.

5 Zadanie 3: Komunikacja dwukierunkowa i prosty protokół

UART może jednocześnie nadawać i odbierać dane. Dodaj drugie połączenie:

RPi Pico A	RPi Pico B
GP0 TX	GP1 RX
GP1 RX	GP0 TX
GND	GND

Oznacza to, że wykonujemy dwa połączenia **krzyżowe**: `GP0` (TX) modułu RPi Pico A łączymy z `GP1` (RX) modułu RPi Pico B, natomiast `GP0` (TX) modułu RPi Pico B łączymy z `GP1` (RX) modułu RPi Pico A. Dodatkowo łączymy masy obu mikrokontrolerów.

Zamiast wysyłać dowolne teksty, zdefiniujemy prosty **protokół aplikacyjny**. RPi Pico A będzie wysyłało polecenia, a RPi Pico B będzie wykonywało odpowiednią operację i odsyłało odpowiedź.

Polecenie	Działanie RPi Pico B	Odpowiedź
`LED:ON`	włączenie LED	`OK`
`LED:OFF`	wyłączenie LED	`OK`
`ADC?`	odczyt POT1 przez ADC	`ADC:xxxxx`

Zadanie 3.1 — Urządzenie wykonawcze RPi Pico B

Na RPi Pico B:

1. podłącz diodę LED do `GP15`,
2. połącz POT1 z `GP26`,
3. odbieraj kolejne polecenia zakończone znakiem `\n`,
4. dla `LED:ON` i `LED:OFF` steruj diodą,
5. dla `ADC?` wykonaj pomiar i odeślij wynik,
6. dla nieznanego polecenia odeślij `ERR`.

Pamiętaj, że dioda LED na płytce jest aktywna stanem niskim.

Podpowiedź

Odebrany komunikat możesz porównywać za pomocą konstrukcji:

```
if command == "LED:ON":
    # ...
elif command == "LED:OFF":
    # ...
elif command == "ADC?":
    # ...
else:
    # ...
```

Odpowiedź również zakończ znakiem `\n`.

Zadanie 3.2 — Sterownik RPi Pico A

Napisz program dla RPi Pico A, który kolejno:

1. wysyła `LED:ON` i czeka na odpowiedź,
2. po 1 s wysyła `ADC?` i wyświetla otrzymany wynik,
3. po kolejnej 1 s wysyła `LED:OFF` i czeka na odpowiedź,
4. powtarza cały cykl.

Sprawdź, co stanie się, gdy celowo wyślesz nieobsługiwane polecenie, np. `TEST`.

Po co tworzyć protokół?

UART definiuje sposób przesyłania kolejnych bitów i bajtów, ale nie określa znaczenia danych. To programista ustala, czy ciąg bajtów oznacza temperaturę, polecenie włączenia diody, współrzędne GPS czy coś zupełnie innego.
Zbiór reguł opisujących format i znaczenie komunikatów nazywamy **protokołem komunikacyjnym**.

6 Zadanie 4: Moduł GPS jako urządzenie UART

Moduły GPS używane na stanowiskach wysyłają dane przez UART. Domyślna szybkość transmisji modułów Waveshare LC76G GNSS wynosi 115200 bit/s.

W tej części wykorzystamy drugi interfejs UART mikrokontrolera:

GPS	RPi Pico A
VCC	5 V
GND	GND
TXD	GP5 — UART1 RX
RXD	nie podłączaj

Reszty wyprowadzeń z modułu GPS nie podłączaj.

Do samego odbierania pozycji wystarczy linia TX modułu GPS połączona z wejściem RX mikrokontrolera.

Uwaga

Przed podłączeniem sprawdź oznaczenia wyprowadzeń konkretnego modułu GPS znajdującego się na stanowisku. W tej instrukcji wykorzystujemy moduły Waveshare LC76G GNSS.
Odbiór sygnału satelitarnego wewnątrz budynku może być słaby. Moduł może wysyłać poprawne ramki NMEA, mimo że nie ma jeszcze poprawnie wyznaczonej pozycji.

Zadanie 4.1 — Surowe dane z GPS

Podłącz moduł GPS do RPi Pico A i uruchom:

```
from machine import UART, Pin
import utime

gps = UART(1, baudrate = 115200, tx = Pin(4), rx=Pin(5),
   bits=8, parity = None, stop=1, rxbuf=4096)
bufor = b""

while True:
    if gps.any():
        dane = gps.read()

        if dane:
            bufor += dane

            # Odczyt moze zawierac fragment lub kilka
               ramek.
            while b"\n" in bufor:
                linia, bufor = bufor.split(b"\n", 1)
```

```
                    linia = linia.strip()
18
19                  # Pomijamy urwany poczatek po starcie
                        programu
20                  if linia.startswith(b"$"):
21                      try:
22                          text = linia.decode("ascii")
23                          print(text)
24                      except UnicodeError:
25                          pass
26
27              # Ochrona przed brakiem konca linii
28              if len(bufor) > 2048:
29                  bufor = b""
30
31      utime.sleep_ms(1)
```

Zaobserwuj:

- od jakiego znaku rozpoczynają się komunikaty,
- jakie różne identyfikatory ramek pojawiają się w danych,
- jak często pojawiają się nowe linie,
- czy pola współrzędnych są wypełnione.

Jak działają bufor, split() i strip()?

UART dostarcza strumień bajtów. Jeden odczyt może zawierać część ramki lub kilka ramek NMEA. Dlatego dopisujemy dane do zmiennej `bufor` i czekamy na znak końca linii: `b"\n"`. Ramki NMEA kończą się parą CR/LF, czyli `b"\r\n"`.

Metoda split(separator, maxsplit) dzieli dane i zwraca listę części. Separator musi mieć ten sam typ co dane: dla bajtów używamy `b"\n"`, dla tekstu np. `","`. Argument `1` ogranicza liczbę podziałów do jednego:

```
bufor = b"ABC\r\nDEF\r\nGHI"
linia, bufor = bufor.split(b"\n", 1)
# linia: b"ABC\r"
# bufor: b"DEF\r\nGHI"
```

Przypisanie rozpakowuje dwie części do dwóch zmiennych. Separator znika z wyniku, lecz reszta bufora pozostaje do dalszej analizy. Pętla `while` odbiera wszystkie kompletne linie; niepełna końcówka czeka na następny odczyt.

Metoda strip() zwraca kopię bez białych znaków na początku i końcu (np. spacji, CR i LF). Nie usuwa przecinków ani znaków wewnątrz danych:

```
linia = b"ABC\r"
```



```
linia = linia.strip()          # b"ABC"
tekst = "  A,B  \r\n".strip()  # "A,B"
```

Metody nie zmieniają oryginału w miejscu, dlatego przypisujemy ich wynik. Tutaj `strip()` usuwa przede wszystkim CR pozostałe po podziale po LF. Dopiero potem `decode(''ascii'')` zamienia bajty na tekst.

Drugi sposób użycia `split()` rozdziela pola NMEA:

```
fields = "A,,B,".split(",")
# Wynik: ["A", "", "B", ""]
```

Puste pola są zachowane i mają znaczenie. Nie usuwaj ich, ponieważ przesunęłoby to indeksy współrzędnych i statusu. Indeksowanie listy zaczyna się od zera.

6.1 Komunikaty NMEA

Popularne odbiorniki GPS przesyłają dane w postaci tekstowych zdań NMEA. Poszczególne pola są oddzielone przecinkami.

LC76G jest odbiornikiem GNSS: GPS to jeden z obsługiwanych systemów. Prefiks `GN` oznacza dane z wielu systemów, a `GP` — GPS. Typ zdania (np. GGA) określa układ pól; sam prefiks nie zmienia ich kolejności. W LC76G spotkasz m.in.:

- `$GPGGA` lub `$GNGGA` — dane dotyczące aktualnej pozycji,
- `$GPRMC` lub `$GNRMC` — podstawowe dane nawigacyjne.

W zadaniu wykorzystamy ramkę **GGA**. Jej najważniejsze pola mają postać:

Pole GGA	**Indeks `fields`**	**Znaczenie**
1	1	czas UTC
2	2	szerokość geograficzna w formacie NMEA
3	3	kierunek N/S
4	4	długość geograficzna w formacie NMEA
5	5	kierunek E/W
6	6	status wyznaczenia pozycji (*fix quality*); 0 oznacza brak poprawnego fixu
7	7	liczba satelitów używanych do wyznaczenia pozycji
8	8	HDOP — współczynnik geometrii satelitów w płaszczyźnie poziomej
9	9	wysokość nad poziomem morza

Co oznaczają pola 6 i 8?

Pole 6 nie opisuje jakości połączenia radiowego. Jest to **status sposobu wyznaczenia pozycji** (*fix quality*). Wartość 0 oznacza, że odbiornik nie ma poprawnie wyznaczonej pozycji; wartość większa od zera oznacza dostępny fix.
Pole 8 zawiera wartość **HDOP** (*Horizontal Dilution of Precision*). Opisuje ona wpływ geometrii widocznych satelitów na wyznaczenie pozycji w płaszczyźnie poziomej. Mniejsza wartość HDOP oznacza zazwyczaj korzystniejszą geometrię, ale nie jest to bezpośrednio błąd pozycji wyrażony w metrach.

Zadanie 4.2 — Odczyt danych z ramki GGA

Zmodyfikuj program tak, aby analizował tylko ramki GGA.
W zadaniu 4.1 zastąp instrukcję `print(text)` analizą tekstu, pozostawiając ją wewnątrz tego samego bloku `try`. Ramkę rozpoznasz przez:

```
if text.startswith(("$GNGGA,", "$GPGGA,")):
    fields = text.split(",")
```

Co znajduje się w zmiennej `fields`?

Metoda `split(",")` dzieli tekst w miejscach występowania przecinków. Zmienna `fields` nie jest więc pojedynczym napisem, lecz **listą napisów typu** `str`.
Dla przykładowej ramki GGA:

- `fields[0]` może zawierać `"$GNGGA"`,
- `fields[2]` może zawierać `"5213.5000"`,
- `fields[3]` może zawierać `Ń`",
- `fields[6]` może zawierać `"1"`.

Jeżeli chcesz wykonywać obliczenia, tekst trzeba jawnie zamienić na liczbę. Przykładowo:

```
latitude_raw = float(fields[2])
fix_quality = int(fields[6])
satellites = int(fields[7])
hdop = float(fields[8])
altitude = float(fields[9])
```

Przed odwołaniem do indeksów sprawdź, czy `len(fields) >= 10`. Obsłuż `ValueError` przy konwersji uszkodzonych danych. Przed użyciem `float()` lub `int()` warto sprawdzić, czy dane pole nie jest pustym napisem , ponieważ ramka bez poprawnego fixu może zawierać puste pola.

Jeżeli status fixu w polu 6 jest większy od zera, wyświetl:

- czas UTC,



- szerokość geograficzną,
- długość geograficzną,
- liczbę satelitów,
- wartość HDOP,
- wysokość.

Jeżeli moduł nie ma poprawnej pozycji, wyświetl komunikat `NO FIX` oraz — jeśli jest dostępna — liczbę satelitów.

Jeżeli w sali nie uda się uzyskać pozycji



Parser można sprawdzić na uproszczonym tekście treningowym (bez sumy kontrolnej; nie jest to kompletne zdanie NMEA):

```
$GNGGA,120000.00,5213.298718,N,02100.423738,E,1,12,0.94,112.549,M,...
```

7 Zadanie końcowe: Most GPS–UART–RPi Pico B

Na zakończenie połączymy oba zastosowania UART w jeden system. RPi Pico A będzie jednocześnie:

- odbierało surowe dane NMEA z GPS przez UART1,
- przetwarzało ramkę GGA,
- wysyłało uproszczone informacje do RPi Pico B przez UART0.

Zadanie 5.1 — Przekazywanie pozycji do drugiego mikrokontrolera

Zbuduj system o następującym działaniu:

RPi Pico A:

1. odbiera ramki GPS,
2. wybiera ramki GGA,
3. po uzyskaniu poprawnej pozycji przelicza współrzędne na stopnie dziesiętne,
4. wysyła do RPi Pico B komunikat:

```
GPS,latitude,longitude,satellites,altitude
```

Na przykład:

```
GPS,52.225000,21.008333,8,110.0
```

Jeżeli pozycja nie jest dostępna, wyślij:

```
NOFIX,satellites
```

RPi Pico B:

1. odbiera komunikat,
2. rozdziela pola za pomocą `split(",")`,
3. dla komunikatu `GPS` wyświetla współrzędne i pozostałe dane w Thonny,
4. dla `NOFIX` wyświetla informację o oczekiwaniu na pozycję,

5. zapala wybraną diodę LED po uzyskaniu poprawnego fixu.

Każdy komunikat zakończ znakiem `\n`. UART0 pozostaje ustawiony na 9600 bit/s, a UART1 na 115200 bit/s. Wysyłaj jedynie uproszczony komunikat, nie cały strumień NMEA. Po utracie fixu zgaś LED; brak nowych danych również nie powinien pozostawiać bezterminowo wskazania poprawnej pozycji.

Dwa niezależne interfejsy UART

W zadaniu końcowym RPi Pico A korzysta jednocześnie z:

- **UART1** — odbiór danych z GPS na `GP5`,
- **UART0** — wymiana danych z RPi Pico B przez `GP0`/`GP1`.

Jest to typowa sytuacja w systemie mikroprocesorowym: mikrokontroler odbiera informacje z jednego urządzenia, przetwarza je i przekazuje dalej innym interfejsem komunikacyjnym.

8 Podsumowanie

Po tych zajęciach powinieneś umieć

- poprawnie połączyć linie TX i RX dwóch urządzeń,
- skonfigurować UART w MicroPythonie,
- wyjaśnić znaczenie konfiguracji 9600 8N1,
- wysyłać tekst i odbierać dane typu `bytes`,
- zdekodować transmisję UART za pomocą Analog Discovery II,
- zaprojektować prosty protokół tekstowy,
- odczytać dane NMEA z modułu GPS,
- przeliczyć współrzędne NMEA na stopnie dziesiętne,
- wykorzystać dwa interfejsy UART jednocześnie.

Uwaga

Przed opuszczeniem stanowiska zatrzymaj programy na obu modułach Pico i usuń z ich pamięci pliki utworzone podczas tego laboratorium. Odłącz moduł GPS i uporządkuj przewody.

Dokumentacja

- Waveshare LC76G GNSS Module: https://www.waveshare.com/wiki/LC76G_GNSS_Module
- MicroPython, klasa UART: https://docs.micropython.org/en/latest/library/machine.UART.html