# [[home](http://tinyurl.com/swa2025)] SWA-Lab03. Język C

**na 3.0:** Większość z Was wykonała już to zadanie w wersji “strcmp”, teraz czas na realizację rozwojową.

CLI dla systemu wbudowanego z obsługą komend z argumentami - uart0 lub stdio. Komendy wprowadzane z lokalnego terminala przez konwerter USB-UART wbudowany w programator. Załóż zworki DEBUG UART!

Argumenty mogą być liczbami całkowitymi DEC, liczbami HEX lub napisami.

**help** //wyświetla listę dostępnych komend

**reboot** //restart zestawu

**uptime** //wyświetlenie czasu w sekundach od momentu resetu zestawu

**na 3.5:**

Sterowanie LED0,1,2 za pomocą komend.

**led N state** //N=0,1,2 state=0,1

przykład: led 2 1 //zapal LED2

**leds bitmask** //bitmask 0b000 .. 0b111

przykład: leds 111 //zapala wszystkie 3 diody

Użycie zegara czasu rzeczywistego. W Pico2W służy do tego AON.

**settime YYYY MM DD hh mm ss** //ustaw czas

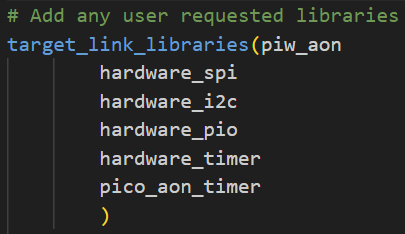
**time** //wyświetl bieżący czas

[~~https://github.com/raspberrypi/pico-examples/tree/master/rtc~~](https://github.com/raspberrypi/pico-examples/tree/master/rtc)

Najwygodniej to zadanie wykonać używając biblioteki aon\_timer zamiast funkcji niskiego poziomu.

Używam UART zamiast stdio, dlatego uart\_puts a nie printf.

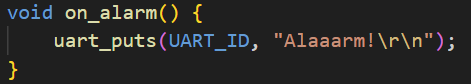
[1] CMake - dodaj pico\_aon\_timer



[2] Plik nagłówkowy biblioteki aon\_timer:

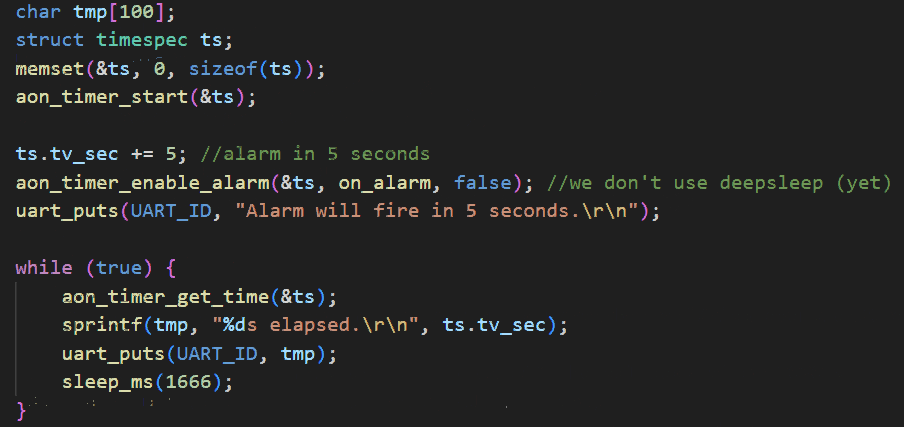


[3] Definicja handlera (funkcji obsługi) alarmu:

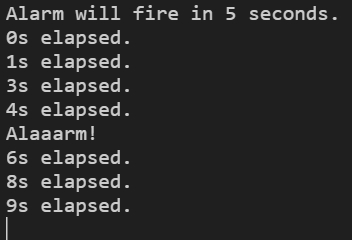


[4] Przykładowy kod (bez funkcji przeliczającej sekundy⇐⇒ YYYYMMDDhhmmss)

Startujemy od zera, alarm uruchomi się po 5 sekundach.

sleep\_ms(1666) celowo, żeby się nie wydawało że odliczanie czasu jest w pętli while(...)

Wynik działania powyższego kodu.



**na 4.0:**

PWM - sprzętowo lub programowo

**pwm N value** //N=0,1,2 value 0..100 (%)

przykład: pwm 0 50 //LED0 połowa jasności

Sprzętowy PWM: <https://github.com/raspberrypi/pico-examples/tree/master/pwm/hello_pwm>

Obsługa OLED i wyświetlanie tekstów przesłanych z CLI

**oled systemy|wbudowane|dla|automatyki** //wyświetli 4 linijki tekstu na wyświetlaczu OLED

Wskazówka: kontroler OLED to SSD1306 128x32 na I2C adres 0x3C  
<https://github.com/raspberrypi/pico-examples/tree/master/i2c/ssd1306_i2c>

**na 4.5:**

Obsługa UART1-485,115200,8N1 w trybie RO - kierunek transmisji "na sztywno" do GND

Odbieranie i dekodowanie komend rozgłaszanych przeze mnie - z listy powyżej

**na 5.0:**

Obsługa dancefloor (16 x WS2812B)

**ws N RRGGBB** //N:0..15, RGB jako hex 00..FF

**df r0r0b0 r1g1b1 ... rFgFbF** //dane dla 16 ledów RGB

**na 5.5:**

Obsługa DMX512

**dmx 60 255** //zapis 255 do rejestru 60

Implementacja Modbus na przekaźnikach <https://www.waveshare.com/wiki/Modbus_RTU_Relay>

**mb05 slave hr value** //adres slave, adres holding register, wartość 16-bitowa

Wskazówki:

* Użyj jakiejś gotowej biblioteki — implementacja Modbus od podstaw zajmie zbyt dużo czasu.
* Zwróć uwagę na domyślną konfigurację modułu.
* https://www.modbus.org/docs/Modbus\_Application\_Protocol\_V1\_1b.pdf