Systemy wbudowane dla automatyki W04

dr inż. Krzysztof Urbański

Dodatkowe materiały do kursu

Źródła: materiały własne, publicznie dostępne dokumenty w dostępie otwartym oraz noty katalogowe i oficjalna dokumentacja prezentowanych rozwiązań.

Modbus (uzupełnienie)

1 bit startu, 8 bitów danych, 1 bit parzystości (typowo E), 1 bit stopu.

Ramka RTU (z ang. *Remote Terminal Unit*) oraz ASCII (ang. *American Standard Code for Information Interchange*)

Modbus RTU Frame Format

Start	Address	Function	Data	CRC	End
≥3.5 char	8 bit	8 bit	N * 8 bit	16 bits	≥3.5 char

Modbus ASCII Frame Format

Start	Address	Function	Data	LRC	End
;	2chars	2chars	N * 1 chars	2chars	CR, LF

https://www.wevolver.com/article/modbus-rtu-a-comprehensive-guide-to-understanding-and-implementing-the-protocol

Adres 0 jest rozgłoszeniowy.

Dozwolone adresy urządzeń slave: 1 do 247.

Kodowanie porządku bajtów: big-endian (bardziej znaczący bajt jest przesyłany wcześniej).

Przykład: wartość 0x1234 jest przesyłana jako bajt 0x12, a następnie 0x34.

W praktyce używa się niewielu funkcji, najczęściej tych związanych z PLC:

- Read Coils (0x01): odczyt stanu cewek (**DQ**) R/W
- Read Discrete Inputs (0x02): odczyt wejść cyfrowych (**DI**) RO
- Read Holding Registers (0x03): odczyt wyjść analogowych (**AQ**) R/W
- Read Input Registers (0x04): odczyt wejść analogowych (**AI**) RO
- Write Single Coil (0x05)
- Write Single Register (0x06)
- Write Multiple Coils (0x0F)
- Write Multiple Registers (0x10)

Uwaga na pułapkę w adresach logicznych!

Data block	Prefix
Coils	0
Discrete inputs	1
Input registers	3
Holding registers	4

PDU = Protocol Data Unit = adres przesyłany protokołem
Logical Address = Register Address + Offset
Register Address = Logical Address – Offset

Przykład:

- Niech adres PDU = 13
- Adres logiczny: 4014 (offset 4001)
- Adres logiczny: 40014 (offset 40001)
- Adres logiczny: 400014 (offset 400001) – a zatem więcej niż 16 bitów dopuszczalnie dla PDU!

Przykładowo mamy offset 40001 (prefix 4, więc dotyczy HR). Chcemy odczytać 2 rejestry HR od 40108 do 40109.
Adres PDU początku bloku tych rejestrów to 40108-40001 = 107 = 0x006B.
Kod funkcji to 0x03 (bo czytamy HR)
Załóżmy, że urządzenie slave ma adres 0x96.

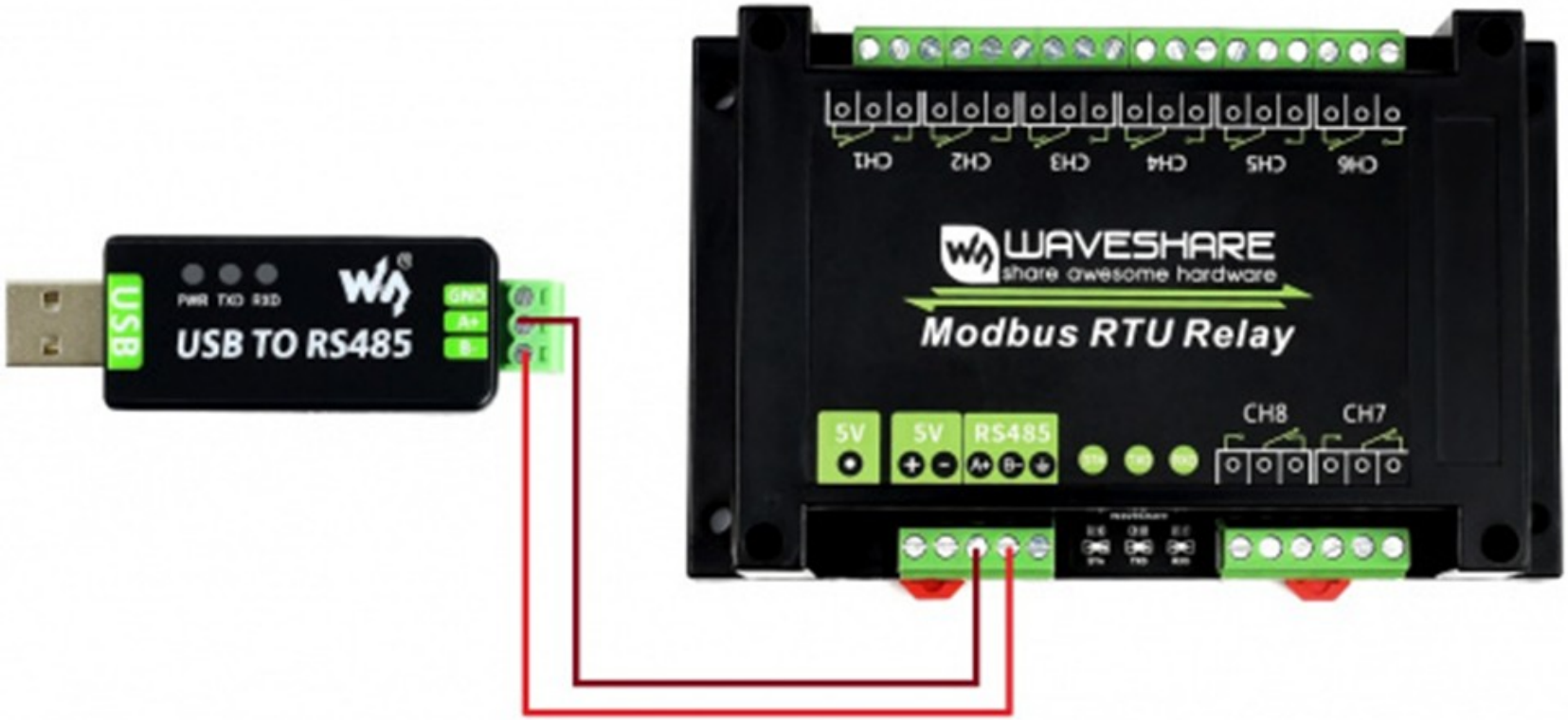
Ramka RTU wygląda tak (gdzie xxxx to wyliczone CRC):
Wysyłamy 00 6B bo 00 to hi-byte, 6B to lo-byte → big endian!

`[start≥3.5] 96 03 006B 0002 xxxx [stop≥3.5]`

Niebieski fragment to **PDU**, a cała ramka to **ADU** (ang. Application Data Unit).

https://www.waveshare.com/wiki/Modbus_RTU_Relay

Sprzęt dostępny na zajęciach – kilka modułów przekaźnikowych w domyślnej konfiguracji.



Przykład 1.

Control Single Relay

Send code: 01 05 00 00 FF 00 8C 3A

Byte	Meaning	Description
01	Device address	0x00: the broadcast address; 0x01-0xFF: device addresses
05	05 Command	Command for controlling Relay
00 00	Address	The register address of controlled Relay, 0x0000 - 0x0007
FF 00	Command	0xFF00: open Relay; 0x0000: close Relay; 0x5500: flip Relay
8C 3A	CRC16	The CRC16 checksum of the first six bytes

Return code: 01 05 00 00 FF 00 8C 3A

Przykład 2.

Read States of Relays

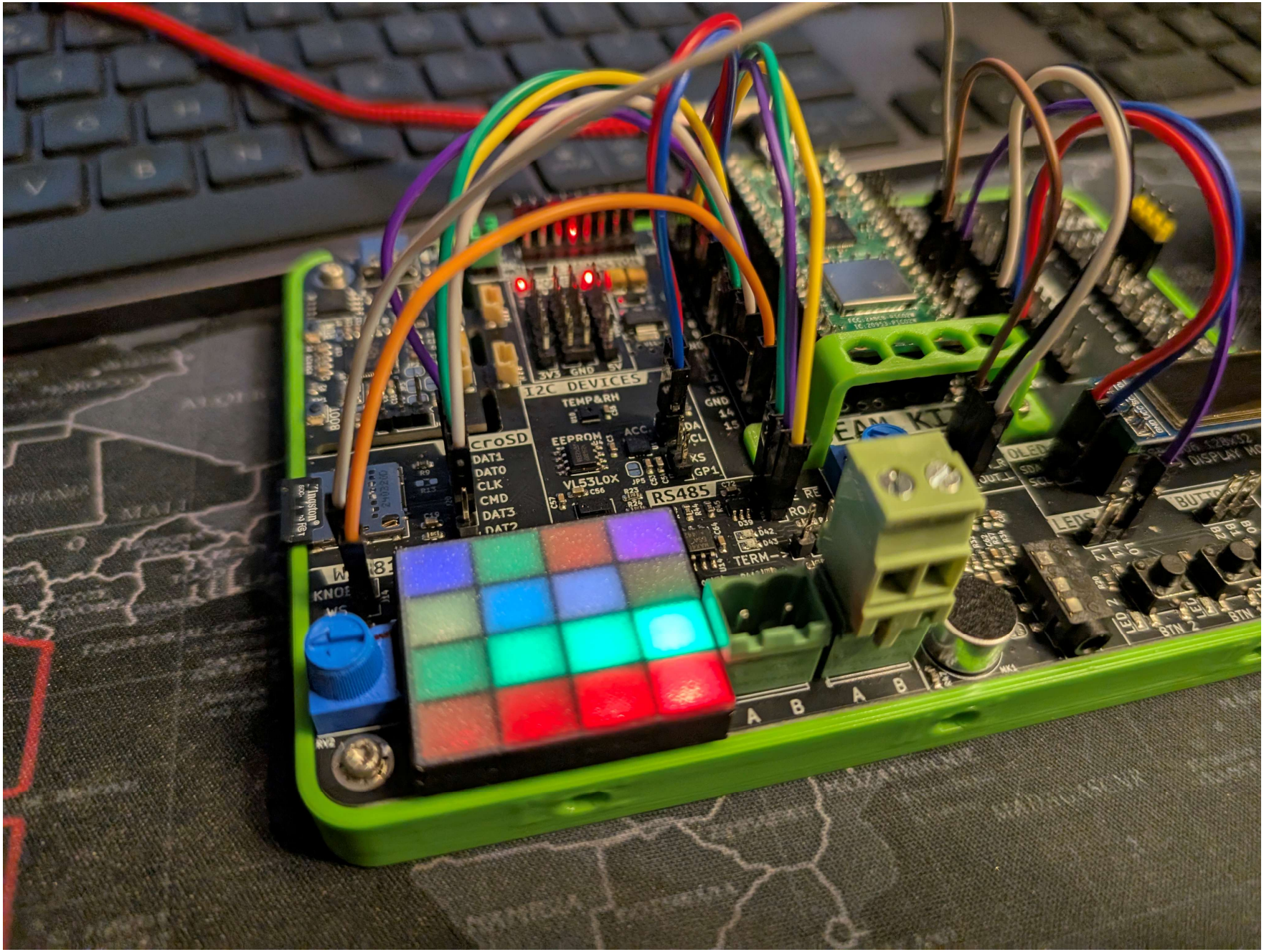
Send code: 01 01 00 00 00 08 3D CC

Bytes	Meaning	Description
01	Device address	0x00: the broadcast address; 0x01-0xFF: device addresses
01	01 Command	Command for checking states of Relay
00 00	Relay start address	The register address for the relays,0x0000-0x0007
00 08	Quantity	The quantity of relays to be read, should not exceed the maximum number of relays.
3D CC	CRC16	The CRC16 checksum of the first six bytes

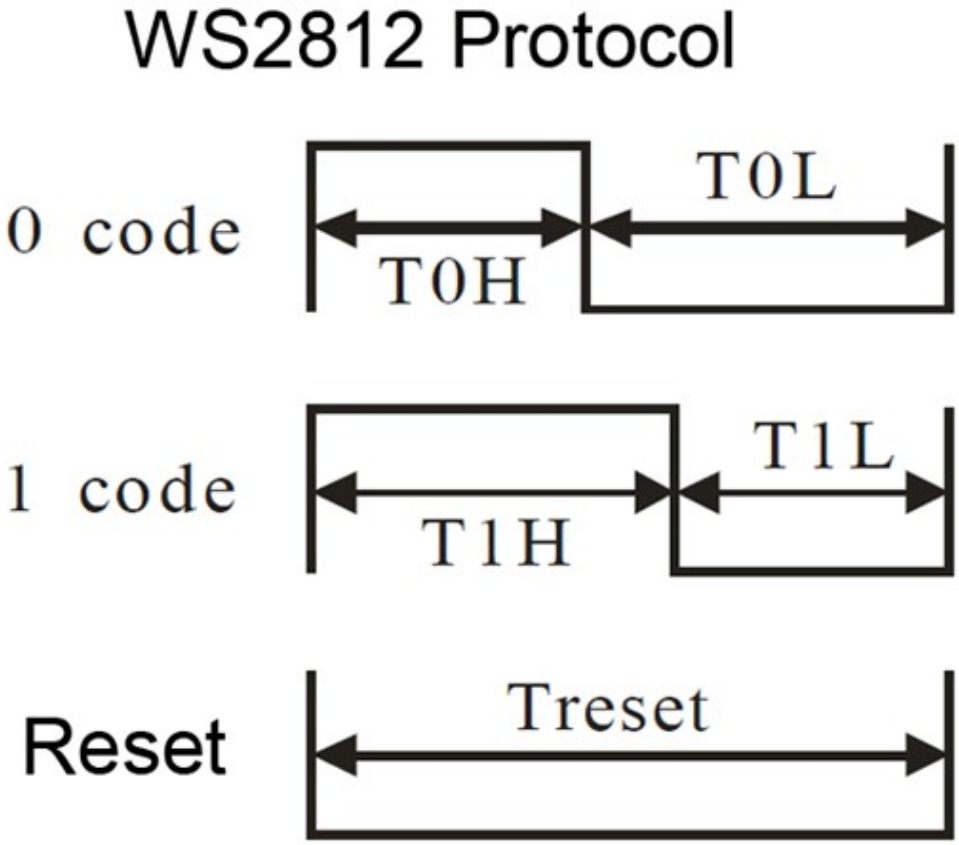
Return code: 01 01 01 00 51 88

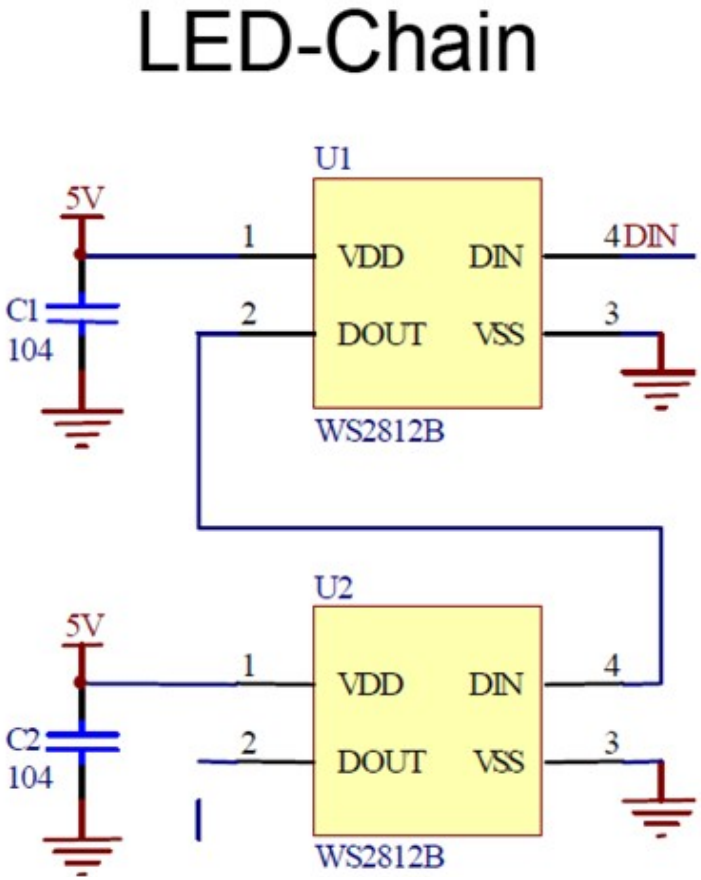
Byte	Meaning	Description
01	Device address	0x00: the broadcast address; 0x01-0xFF: device addresses
01	01 Command	Command for checking states of Relay
01	Number	The number of bytes returned
00	State	The state of Relay Bit0: The state of the first Relay; Bit1: The state of the second Relay; Bit2: The state of the third Relay; ..... Bit7: The state of the eighth Relay
51 88	CRC16	The CRC16 checksum of the first six bytes

Dancefloor WS2812B



Ten interfejs też jest asynchroniczny i szeregowy – ale tm razem inaczej kodowane są bity – zarówno 0 jak i 1 to kombinacja wysokiej i niskiej wartości napięcia:





→patrz: datasheet

Nie będzie łatwo, ale...

```
from machine import Pin
import rp2, array
NUM_LEDS = const(16)
PIN_NUM = const(15)
# dane pikseli po 32 bity (używane 24:GRB) na każdy LED
ar = array.array("I", [0 for _ in range(NUM_LEDS)])
# PIO w wersji zanegowanej
@rp2.asm_pio(sideset_init=rp2.PIO.OUT_HIGH,
out_shiftdir=rp2.PIO.SHIFT_LEFT, autopull=True, pull_thresh=24)
def ws2812():
    T1 = 2
    T2 = 5
    T3 = 3
    wrap_target()
    label("bitloop")
    out(x, 1)               .side(1)    [T3 - 1]
    jmp(not_x, "do_zero")   .side(0)    [T1 - 1]
    jmp("bitloop")          .side(0)    [T2 - 1]
    label("do_zero")
    nop()                   .side(1)    [T2 - 1]
    wrap()
```

```
# włączenie maszyny PIO
sm = rp2.StateMachine(0, ws2812, freq=8_000_000,
sideset_base=Pin(PIN_NUM))
sm.active(1)

# pompowanie danych z podmianą RGB na GRB
def ws_show():
    dimmer_ar = array.array("I", [0 for _ in range(NUM_LEDS)])
    for i,c in enumerate(ar):
        r = int(((c >> 8) & 0xFF))
        g = int(((c >> 16) & 0xFF))
        b = int((c & 0xFF))
        dimmer_ar[i] = (g<<16) + (r<<8) + b
    sm.put(dimmer_ar, 8) ← tutaj rozpoczęcie transmisji

# ustaw kolor jednego piksela
def ws_set(i, color):
    ar[i] = (color[1]<<16) + (color[0]<<8) + color[2]

# ustaw kolor wszystkich pikseli
def ws_fill(color):
    for i in range(len(ar)):
        ws_set(i, color)
```

WS2812: plik .pio

```
.pio_version 0

.program ws2812
.side_set 1

.define public T1 3
.define public T2 3
.define public T3 4

.wrap_target
bitloop:
    out x, 1       side 1 [T3 - 1]
    jmp !x do_zero side 0 [T1 - 1]
do_one:
    jmp  bitloop   side 0 [T2 - 1]
do_zero:
    nop            side 1 [T2 - 1]
.wrap
```

WS2812: CMake

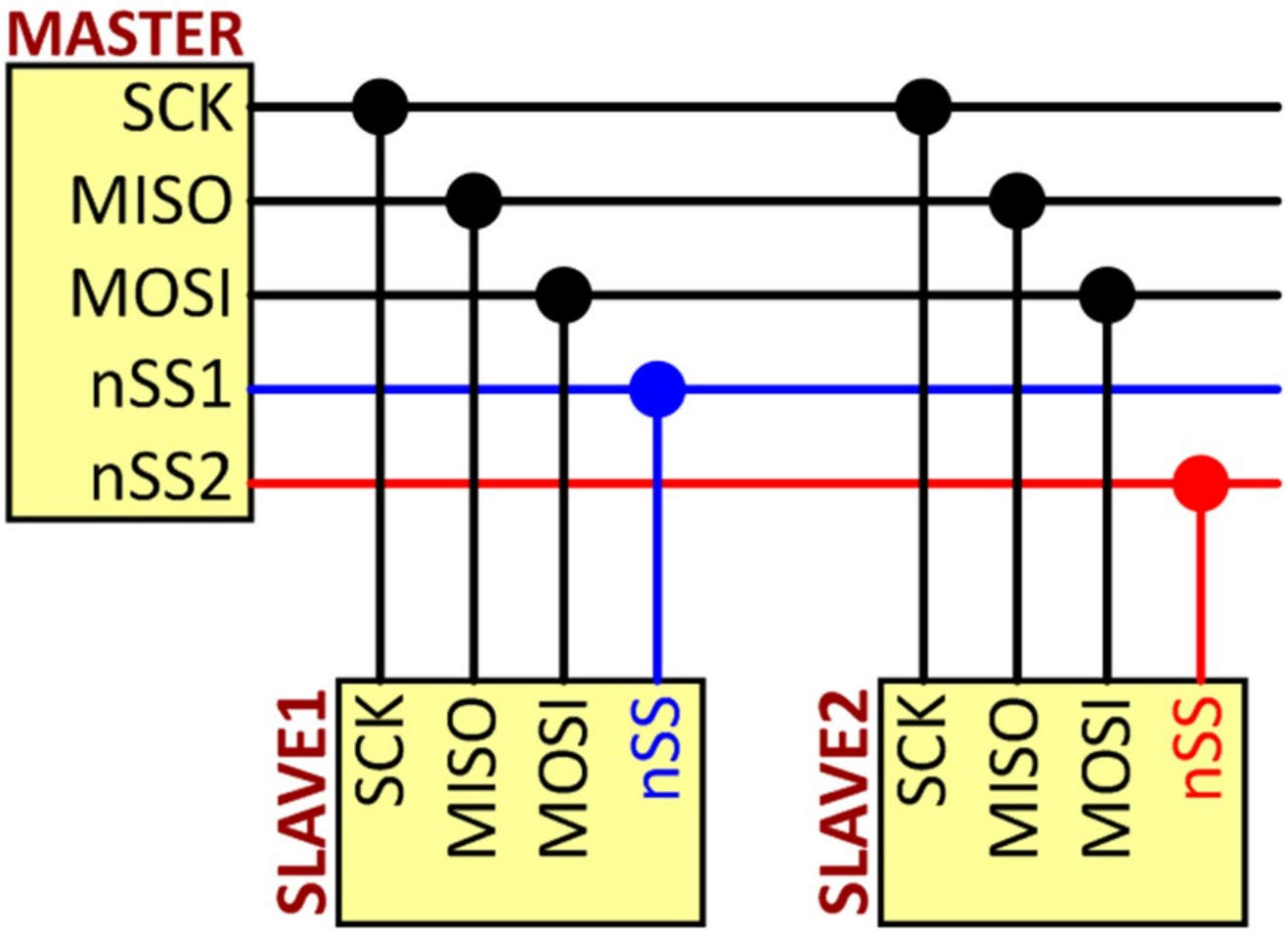
```
# Add the standard library to the build
target_link_libraries(ws2812inv_v1 pico_stdlib hardware_pio)

file(MAKE_DIRECTORY ${CMAKE_CURRENT_LIST_DIR}/generated)

# generate the header file into the source tree as it is included in the
RP2040 datasheet
pico_generate_pio_header(ws2812inv_v1
${CMAKE_CURRENT_LIST_DIR}/ws2812.pio
OUTPUT_DIR ${CMAKE_CURRENT_LIST_DIR}/generated)
```

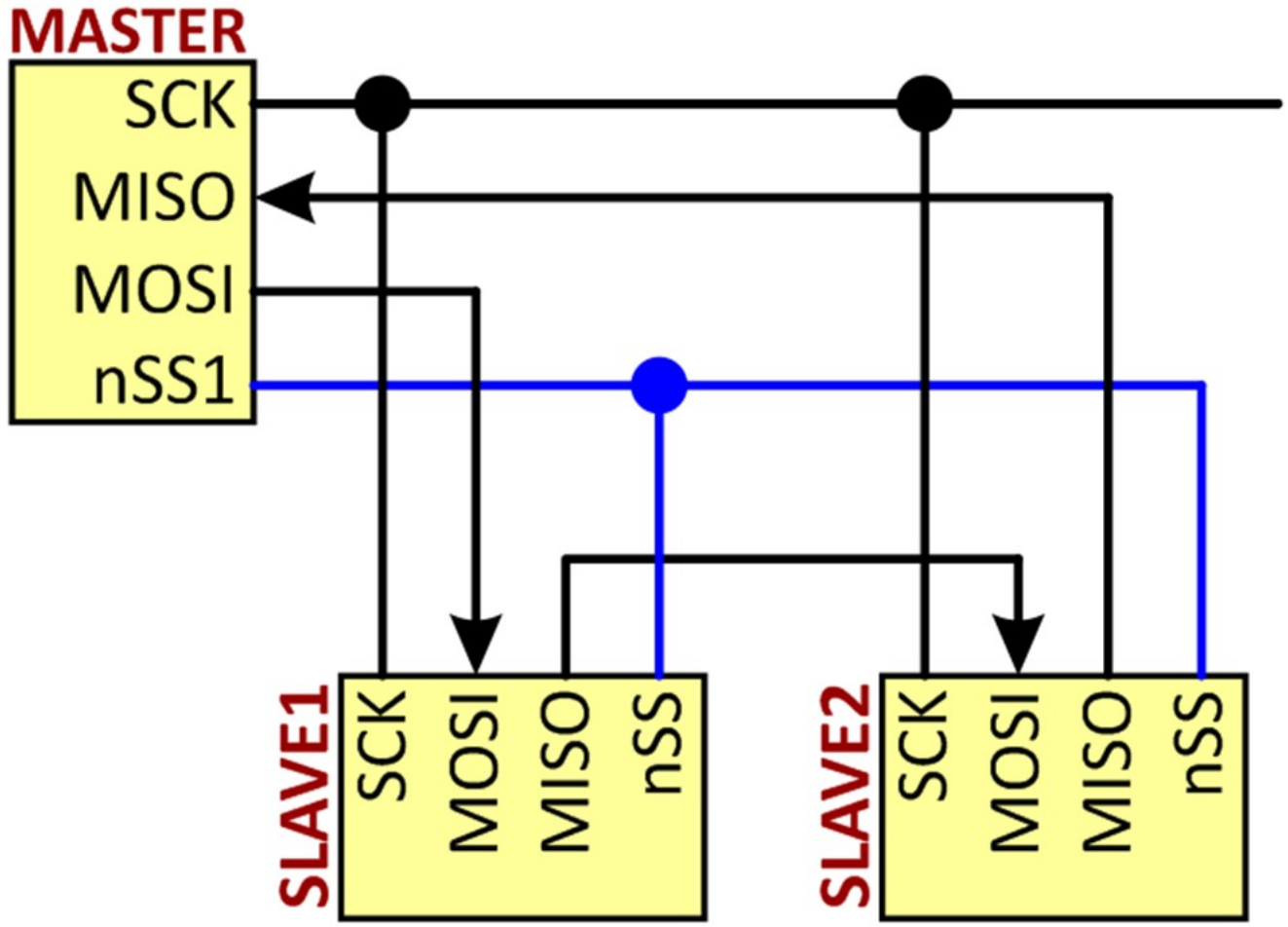
WS2812: C/C++

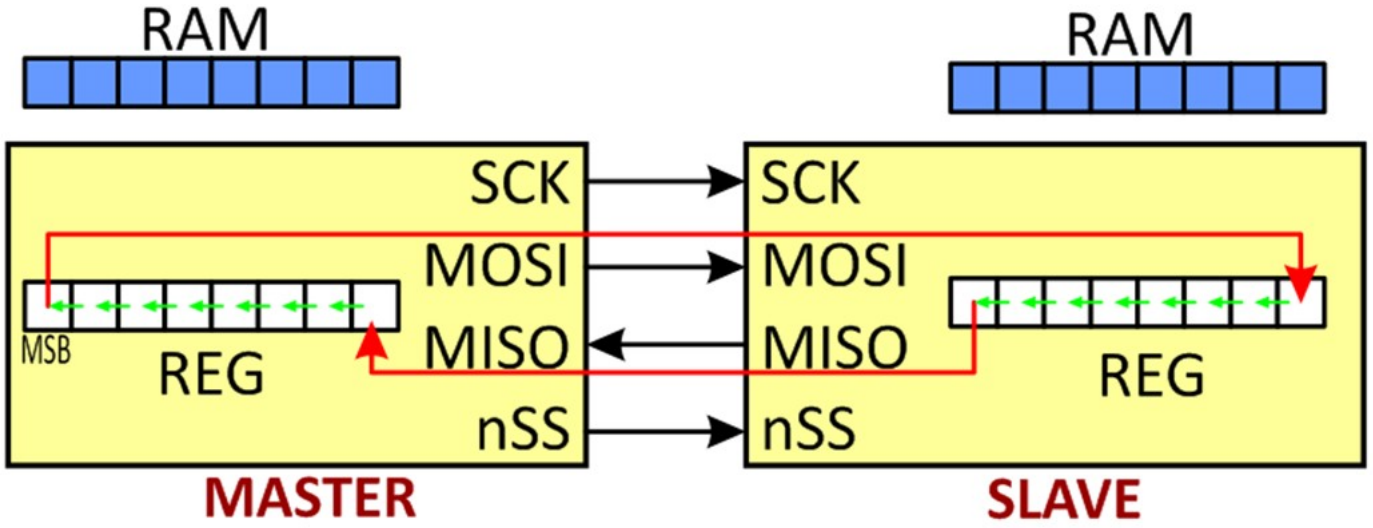
```
static inline void put_pixel(
    PIO pio, uint sm, uint32_t pixel_grb) {
    pio_sm_put_blocking(pio, sm, pixel_grb << 8u);
}
void pattern_uni(PIO pio, uint sm, uint len, uint32_t grb) {
    for (uint i = 0; i < len; ++i) {
        put_pixel(pio, sm, grb);
    }
}

PIO pio;
uint sm;
uint offset;
bool success = pio_claim_free_sm_and_add_program_for_gpio_range(
    &ws2812_program, &pio, &sm, &offset, WS2812_PIN, 1, true);
hard_assert(success);
ws2812_program_init(pio, sm, offset, WS2812_PIN, 800000);
```

SPI: Serial Peripheral Interface

- interfejs **synchroniczny**, szeregowy
- układ nadrzędny (*master*) + układy podrzędne (*slave*)
- topologia magistrali (ang. bus)
- wysoka przepustowość (nawet >100 Mbit/s)
- sygnały MISO, MOSI, SCLK (SCK, CLK) i dodatkowo nSS (nCS), spotykane więcej (2, 4) linii danych, np. zewnęrzna pamięć Flash Pi Pico
- 4 tryby pracy (kombinacje bitów CPHA i CPOL)
- możliwość połączenia układów *slave* w łańcuch (*daisy-chain*) lub równolegle: →

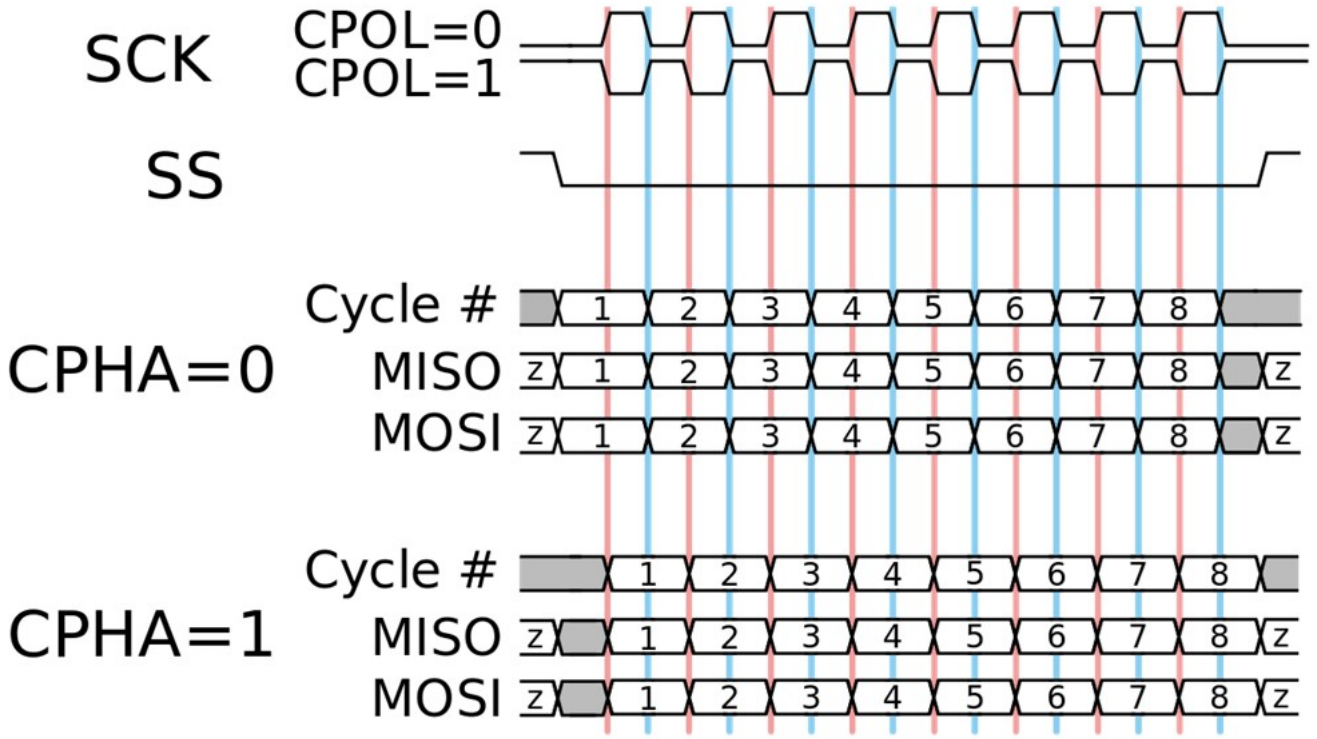

MASTER
SCK
MISO
MOSI
nSS1
nSS2
SLAVE1
SCK
MISO
MOSI
nSS
SLAVE2
SCK
MISO
MOSI
nSS


MASTER
SCK
MISO
MOSI
nSS1
SLAVE1
SCK
MOSI
MISO
nSS
SLAVE2
SCK
MOSI
MISO
nSS



Schemat transmisji 8-bitowego znaku, jako pierwszy wysyłany jest najbardziej znaczący bit (*MSB first*). Efekt końcowy to "zamienione miejscami" rejestry danych układów *Master* i *Slave*.

Transmisja wyzwalana jest instrukcją zapisania dowolnej wartości do rejestru danych układu *Master*, po której można odczytać to, co zostało przekazane od *Slave*. Odczytanie rejestru nie aktywuje transmisji SPI! (patrz też: *dummy write*).



[źródło: en.wikipedia.org]

Przykład transmisji znaku w SPI (bez przerwań, bez DMA):

```
byte spi(byte d) {
  while(!SPISR_SPTEF); //Transmitter Empty
  SPIDR = d;
  while(!SPISR_SPIF); //Interrupt Flag
  return SPIDR;
}
```

…i dla porównania w UART:

```
void TERMIO_PutChar(char c) {
  while(!SCI0SR1_TC); //Transmission Complete
  SCI0DRL = (byte)c; //Data Register Low
}
```

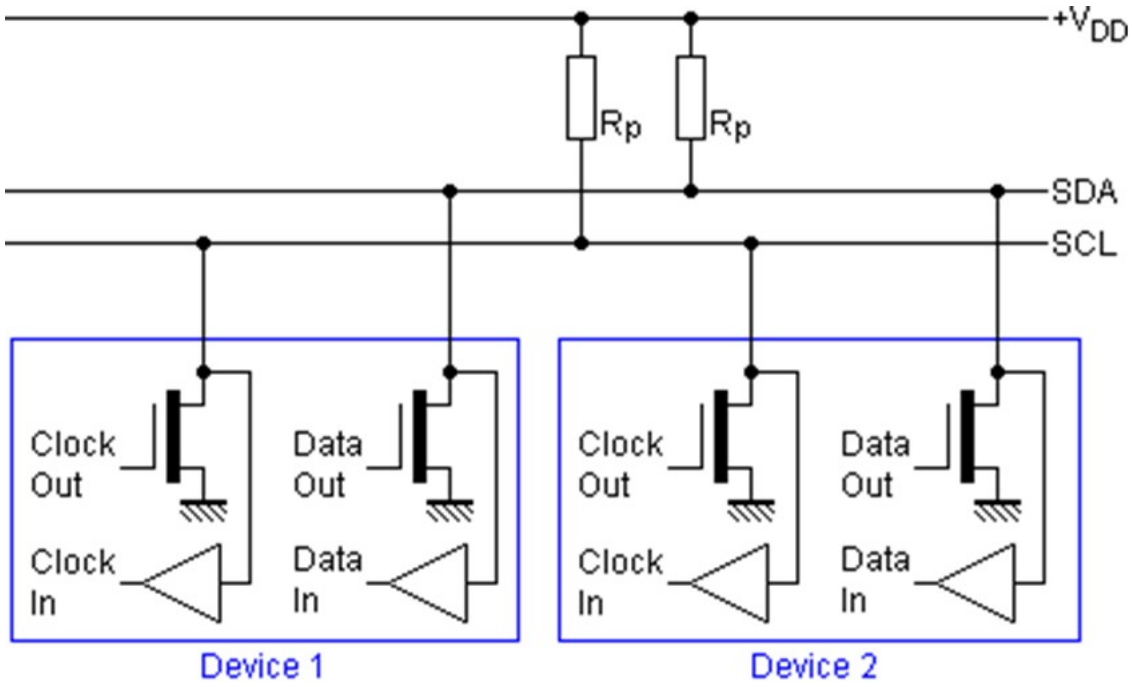
Użyto mikrokontrolera z rodziny HCS12 (16-bitowy), ale w innych MCU zasada działania analogiczna.

IIC, I2C – Inter-Integrated Circuit

- Wersja podstawowa: 100 kbit/s, 7-bitowe adresy
- 1992: v.1.0 – fast mode (400 kbit/s), 10-bitowe adresy
- 1998: v.2.0 – high speed (3,4 Mbit/s) i szerszy zakres napięć (2,3 .. 5 V)

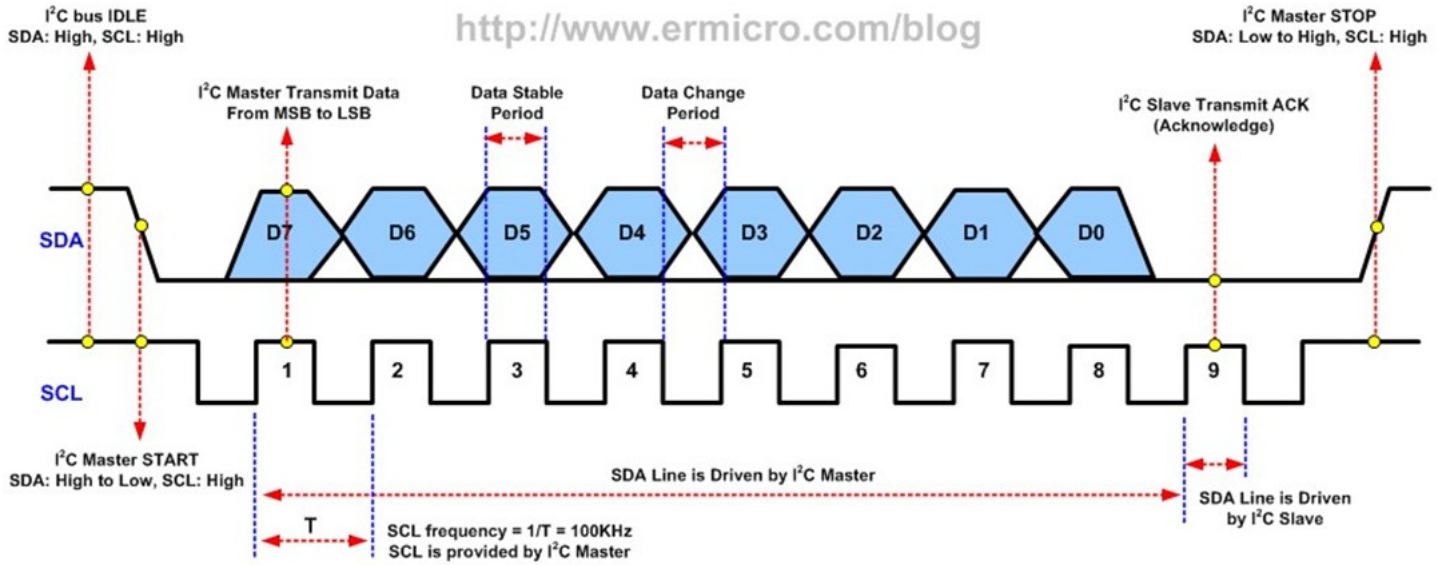
Kluczowe cechy:

- 2-przewodowa, szeregowa magistrala: SCL, SDA
- Rezystory podciągające (*pull-up resistors*)
- 1 *master* ***, wiele układów *slave*
- 7-bitowe adresy
- Niewielka prędkość i zasięg
- IDLE, START, STOP, RSTA, ACK, NAK (n-ACK)



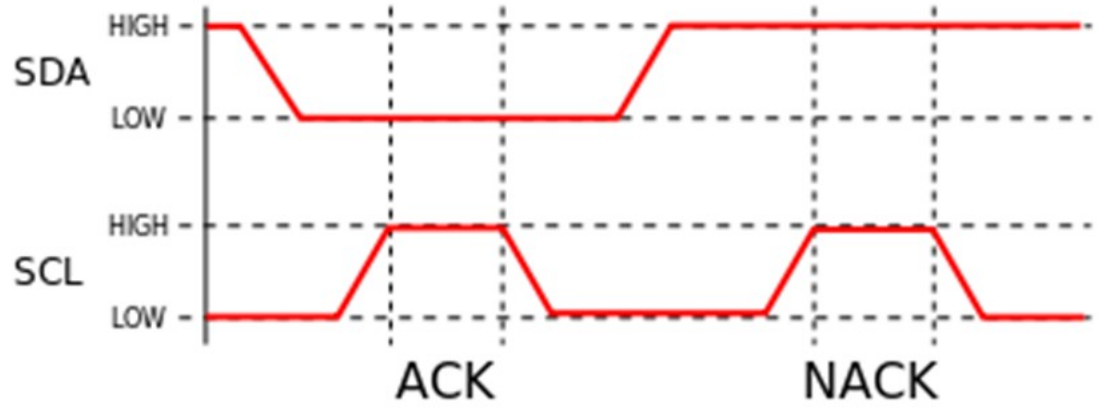
Symboliczne przedstawienia stopni wejściowych i wyjściowych magistrali I2C, przy czym Device 1 oraz Device 2 mogą być skonfigurowane zarówno jako master jak i slave (typowe w mikrokontrolerach)

1. Co znaczą te symbole?
2. Porównaj z 2-kierunkowymi stopniami we/wy w RS-485.



I²C Bus Master and Slave Timing Diagram

Typowy początek transmisji w I2C. Zwróć uwagę na zmianę kierunku linii SDA w 9. cyklu!



Istnieje możliwość implementacji master soft-I2C poprzez tzw. bit-banging, wystarczy zbiór kilku podstawowych operacji na 2 liniach GPIO.