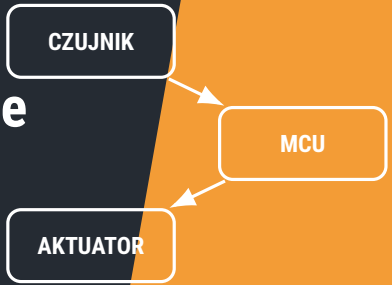
Od czujnika do aktuatora
Arduino UNO R4 WiFi w praktyce

ADC, I²C, SPI, 1-Wire, serwo i czujnik odległości



Podstawy Systemów Wbudowanych • Wykład 2
dr inż. Angelika Tefelska

Do czego przygotowuje Wykład 2?

Laboratorium 3

Czujniki i akwizycja danych

- ADC i czujnik analogowy,
- czujnik środowiskowy I^2C,
- SPI i czytnik RFID.

Laboratorium 4

Aktuatory i układ sterowania

- serwomechanizm,
- joystick analogowy,
- HC-SR04,
- automatyczna bariera parkingowa.



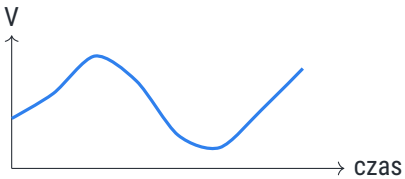
1. ADC — od napięcia do liczby

Czujnik analogowy nie wysyła „wilgotności 63%”. Najpierw ustawia napięcie.

Sygnał analogowy i cyfrowy — dwa różne światy

Analogowy

- napięcie może przyjmować wiele wartości,
- zmienia się w sposób ciągły,
- przykład: potencjometr, fotorezystor, sonda wilgotności czy czujnik temperatury.

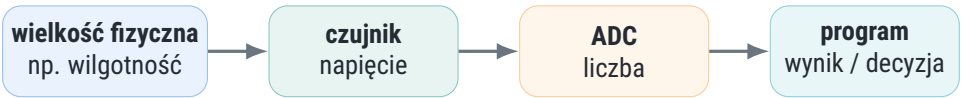


Cyfrowy

- informacja jest zakodowana w liczbach / bitach,
- urządzenie komunikuje się według protokołu,
- przykład: BME280, RFID, GPS.



Łańcuch pomiarowy — gdzie pojawia się ADC?



ADC = Analog-to-Digital Converter

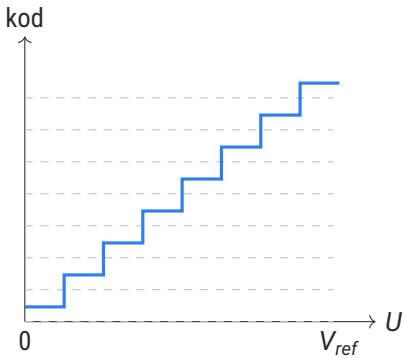
Przetwornik analogowo-cyfrowy zamienia **napięcie** na **liczbę**, z którą może pracować program.

ADC nie zna wszystkich napięć — musi je skwantować

- zakres napięcia dzielimy na skończoną liczbę przedziałów,
- każdemu przedziałowi odpowiada jeden **kod cyfrowy**,
- liczba dostępnych kodów zależy od liczby bitów ADC,
- proces przypisania napięcia do poziomu nazywamy **kwantyzacją**.

Co robi ADC?

Dla każdego pomiaru ustala, **do którego przedziału napięcia należy wartość wejściowa**, a następnie zwraca numer tego przedziału jako kod cyfrowy.



Przykład liczbowy — jak 2.73 V zamienia się na kod ADC?

Założenia

ADC 10-bitowy, $V_{ref} = 5\,V$, napięcie wejściowe $U_{in} = 2.73\,V$.

1. Liczba przedziałów: $2^{10} = 1024$.
2. Szerokość jednego przedziału:

$$\Delta U = \frac{5\,V}{1024} = 4.8828\,mV.$$

3. Numer przedziału:

$$k = \left\lfloor \frac{2.73}{0.0048828} \right\rfloor = 559.$$

Co oznacza kod 559?

Kod **559** odpowiada napięciom w przybliżeniu od

$$559\Delta U = 2.7295\,V$$

do

$$560\Delta U = 2.7344\,V.$$

Napięcie $2.73\,V$ wpada właśnie do tego przedziału.

$$559 = 1000101111_2$$

N bitów — ile możliwych wyników?

Najważniejsza zależność

liczba kodów $= 2^N$, zakres kodów: $0 \ldots 2^N - 1$

Rozdzielczość	Liczba kodów	Zakres
8 bitów	$2^8 = 256$	0–255
10 bitów	$2^{10} = 1024$	0–1023
12 bitów	$2^{12} = 4096$	0–4095
14 bitów	$2^{14} = 16384$	0–16383

Więcej bitów = **drobniejszy podział** zakresu napięcia.

Arduino UNO R4 WiFi — ADC może pracować z większą rozdzielczością

- mikrokontroler RA4M1 ma ADC o maksymalnej rozdzielczości **14 bitów**,
- dla zgodności ze starszym Arduino `analogRead()` domyślnie zwraca wynik **10-bitowy**,
- możemy jawnie ustawić 12 lub 14 bitów przez `analogReadResolution()`.

Domyślnie

`0 ... 1023`

10 bitów

Po ustawieniu 14 bitów

`0 ... 16383`

14 bitów

Źródło: https://docs.arduino.cc/tutorials/uno-r4-wifi/adc-resolution/

Jak mały krok napięcia potrafi rozróżnić idealny ADC?

Idealny krok kwantyzacji

$$\Delta U = \frac{V_{ref}}{2^N}$$

10 bitów, zakres 0–5 V

$$\Delta U \approx \frac{5\,V}{1024} \approx 4.88\,mV$$

14 bitów, zakres 0–5 V

$$\Delta U \approx \frac{5\,V}{16384} \approx 0.305\,mV$$

14 bitów daje około **16 razy mniejszy krok** niż 10 bitów.

Rozdzielczość to nie to samo co dokładność

Rozdzielczość

Jak małą zmianę sygnału ADC **potrafi zakodować**.

Dokładność

Jak blisko wyniku pomiaru jesteśmy **wartości rzeczywistej**.

- wynik może zawierać szum,
- czujnik ma własny błąd i nieliniowość,
- napięcie odniesienia / zasilanie nie jest idealne,
- błędna kalibracja może dać bardzo „precyzyjną”, ale nieprawdziwą liczbę.

Więcej cyfr nie gwarantuje lepszego pomiaru.

Pierwszy odczyt analogowy i przeliczenie na napięcie

```arduino
const int SENSOR_PIN = A0;

void setup() {
  Serial.begin(115200);
  analogReadResolution(14);
}

void loop() {
  int raw = analogRead(SENSOR_PIN);
  Serial.println(raw);
  delay(200);
}
```

Co robi kod?

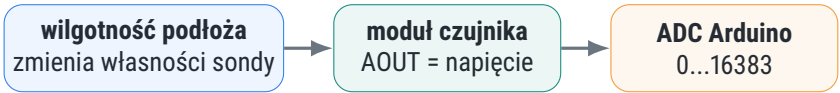
- `analogRead(A0)` pobiera próbkę z wejścia A0,
- `analogReadResolution(14)` ustawia rozdzielczość 14 bitów,
- wtedy spodziewamy się kodów z zakresu 0–16383.

Przeliczenie na napięcie

Dla zakresu 0–5 V: $U \approx \texttt{raw} \cdot 5.0/16383$.
Dla `raw = 8192` otrzymujemy około 2.50 V.

Gdyby `analogReadResolution(14)` pominąć, domyślnie korzystalibyśmy z odczytu 10-bitowego.

Czujnik wilgotności gleby — od napięcia do umownych procentów



Co widzi Arduino?

- surowy wynik ADC opisuje napięcie,
- sam wynik nie jest jeszcze „wilgotnością w procentach”,
- aby nadać mu znaczenie, wyznaczamy **punkty kalibracyjne** w rzeczywistych warunkach.

Przykład kalibracji

Suche podłoże: `4200`, bardzo wilgotne: `13500`.



map() i constrain() — dwie bardzo praktyczne funkcje

```
int moisture = map(raw, dryValue, wetValue, 0, 100);
moisture = constrain(moisture, 0, 100);
```

- `map()` wykonuje liniowe przeskalowanie z jednego zakresu na drugi,
- wynik może wyjść poza 0–100, jeśli pomiar przekroczy wartości kalibracyjne,
- `constrain()` ogranicza wynik do zadanego przedziału.

Uwaga

`map()` nie „wie”, czym jest wilgotność. To **my** zdecydowaliśmy, że dwa punkty pomiarowe odpowiadają 0% i 100%.

Pełny przykład: surowy ADC → umowna wilgotność 0–100%

```
const int SENSOR_PIN = A0;
const int dryValue = 4200;
const int wetValue = 13500;

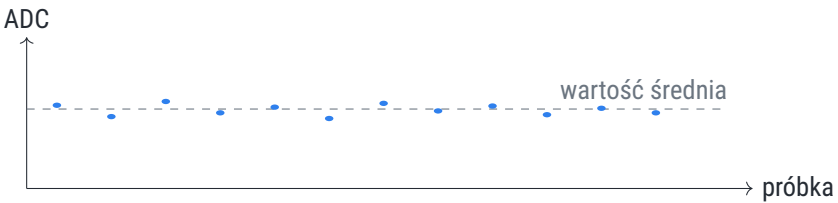
void setup() {
  Serial.begin(115200);
  analogReadResolution(14);
}

void loop() {
  int raw = analogRead(SENSOR_PIN);
  int moisture = map(raw, dryValue, wetValue, 0, 100);
  moisture = constrain(moisture, 0, 100);

  Serial.print("Wilgotnosc: ");
  Serial.print(moisture);
  Serial.println(" %");
  delay(200);
}
```

Dlaczego kilka kolejnych pomiarów nie daje identycznego wyniku?

- szum elektryczny w torze pomiarowym,
- drobne wahania zasilania i napięcia odniesienia,
- zakłócenia elektromagnetyczne,
- ADC kwantuje sygnał do skończonej liczby poziomów.



Proste uśrednianie kilku próbek

```
const int SENSOR_PIN = A0;
const int N = 10;

long sum = 0;

for (int i = 0; i < N; i++) {
  sum += analogRead(SENSOR_PIN);
  delay(5);
}

float average = (float)sum / N;
```

- średnia zmniejsza wpływ losowych wahań,
- nie naprawi błędnej kalibracji ani złego czujnika,
- większe `N` daje stabilniejszy wynik, ale pomiar trwa dłużej.

Co zmieni się po przejściu z ADC 10-bitowego na 14-bitowy?

1. odczytana wartość napięcia będzie na pewno dokładniejsza,
2. zakres kodów wzrośnie z 0–1023 do 0–16383,
3. czujnik zacznie przesyłać dane cyfrowe,
4. maksymalne napięcie wejściowe wzrośnie na pewno 16 razy.

?

Najpierw spróbuj odpowiedzieć bez zaglądania do kodu.

Co zmieni się po przejściu z ADC 10-bitowego na 14-bitowy?

1. odczytana wartość napięcia będzie na pewno dokładniejsza,
2. **zakres kodów wzrośnie z 0–1023 do 0–16383,**
3. czujnik zacznie przesyłać dane cyfrowe,
4. maksymalne napięcie wejściowe wzrośnie na pewno 16 razy.

?

Najpierw spróbuj odpowiedzieć bez zaglądania do kodu.

2. Czujnik cyfrowy I^2C i biblioteki

W poprzednim wykładzie poznaliśmy magistralę. Teraz użyjemy jej jak użytkownik urządzenia.

Czujnik analogowy a cyfrowy — co naprawdę odbiera Arduino?

	Czujnik analogowy	Czujnik cyfrowy I^2C
Co wychodzi z czujnika?	napięcie	bajty danych
Identyfikacja	pin analogowy	adres urządzenia
Przetwarzanie	ADC mikrokontrolera	ADC/algorytm wewnątrz czujnika
Odczyt wartości	`analogRead()`	funkcje biblioteki / rejestry
Przykład	wilgotność gleby	BME280 / ENS160

„Cyfrowy” nie znaczy „bez pomiaru analogowego” — często ADC znajduje się po prostu **wewnątrz czujnika**.

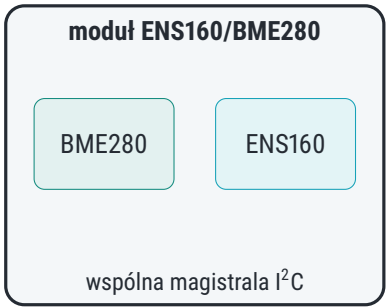
Jeden moduł może zawierać kilka niezależnych urządzeń

BME280

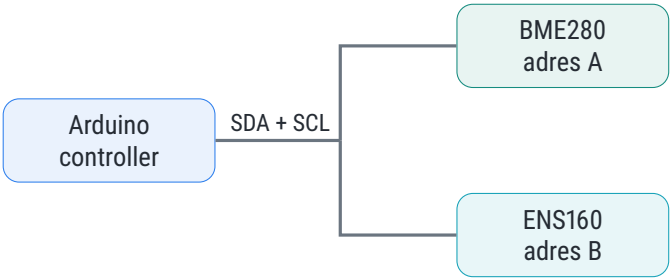
- temperatura,
- wilgotność,
- ciśnienie.

ENS160

- jakość powietrza,
- wskaźniki gazowe / eCO_2 zależnie od biblioteki.



Dwa urządzenia na jednej magistrali — rozróżnia je adres



- controller wysyła adres urządzenia, z którym chce rozmawiać,
- tylko właściwie zaadresowany układ odpowiada,
- dlatego skaner I^2C może znaleźć **więcej niż jedno urządzenie** na tym samym module.

Qwiic — wygodne I²C, ale pamiętaj o napięciu

- złącze Qwiic przenosi **SDA, SCL, 3.3 V i GND**,
- ogranicza ryzyko pomylenia przewodów,
- pozwala łączyć moduły I²C bez płytki stykowej.

Arduino UNO R4 WiFi

Złącze Qwiic na płytce pracuje z **3.3 V**. Nie podłączamy do niego urządzeń wymagających 5 V.



Po co biblioteka urządzenia?

Bez biblioteki

Musimy znać:

- adresy rejestrów,
- format danych,
- kolejność inicjalizacji,
- maski bitowe,
- szczegóły kalibracji producenta.

Z biblioteką

Korzystamy z funkcji typu:

- `begin()`,
- `readTempC()`,
- `readFloatHumidity()`,
- `getAQI()`.

Biblioteka jest **warstwą abstrakcji**: ukrywa szczegóły protokołu, ale nie zmienia zasady działania urządzenia.

Typowy schemat pracy z biblioteką czujnika

```
#include <Wire.h>
#include <SparkFunBME280.h>

BME280 sensor;

void setup() {
  Serial.begin(115200);
  Wire1.begin();

  if (!sensor.beginI2C(Wire1)) {
    Serial.println("Brak odpowiedzi BME280");
    while (true) { }
  }
}
```

Krok po kroku

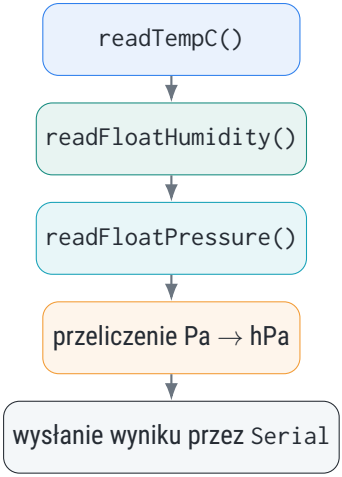
- `#include` dołącza biblioteki,
- `BME280` to **nazwa klasy**,
- `sensor` to obiekt reprezentujący konkretny czujnik,
- `Wire1.begin()` uruchamia magistralę Qwiic/I²C,
- dalej operujemy na obiekcie `Wire1`, a nie na `Wire`,
- `beginI2C(Wire1)` inicjalizuje czujnik i sprawdza odpowiedź.

Odczyt wielkości fizycznych — biblioteka zwraca już sensowne jednostki

```ino
void loop() {
  float temperature = sensor.readTempC();
  float humidity = sensor.readFloatHumidity();
  float pressurePa = sensor.readFloatPressure();
  float pressureHpa = pressurePa / 100.0;

  Serial.print("T = ");
  Serial.print(temperature);
  Serial.print(" C, RH = ");
  Serial.print(humidity);
  Serial.print(" %, p = ");
  Serial.print(pressureHpa);
  Serial.println(" hPa");

  delay(1000);
}
```



Jak czytać taki kod krok po kroku?

- program wywołuje funkcję biblioteki i dostaje gotowy wynik w odpowiednich jednostkach,
- dla ciśnienia producent zwraca wynik w paskalach, dlatego dzielimy przez 100, aby uzyskać hPa,
- potem formatujemy tekst do monitora portu szeregowego.

Wciąż trzeba myśleć

- sprawdź jednostki zwracanej wartości,
- oceń, czy wynik mieści się w fizycznie sensownym zakresie,
- pamiętaj, że dwa układy na module nadal są **dwoma urządzeniami I^2C**,
- biblioteka upraszcza kod, ale nie naprawi złego zasilania ani błędnego połączenia.

Dobra praktyka

Najpierw uruchom **przykład producenta**, potem uprość go i dopasuj do swojego zadania.

Źródło: Laboratorium PSW, labe.engined.eu

3. SPI — szybka magistrala z linią wyboru urządzenia

I^2C wybiera urządzenie adresem. SPI robi to osobną linią CS.

Po co kolejne SPI, skoro znamy już I^2C?

I^2C

- tylko SDA + SCL,
- adresy urządzeń,
- wygodne dla wielu czujników,
- mniejsza liczba przewodów.

SPI

- osobna linia zegara i danych,
- osobna linia CS dla urządzenia,
- zwykle wyższa przepustowość,
- brak obowiązkowego adresowania.

Dobieramy interfejs do urządzenia i wymagań systemu — nie ma jednej „najlepszej” magistrali.

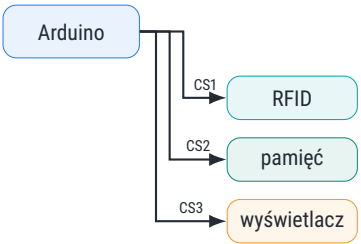
Cztery podstawowe linie SPI

SCK	zegar generowany przez controller
COPI / MOSI	dane od controller do peripheral
CIPO / MISO	dane od peripheral do controller
CS / SS	wybór konkretnego urządzenia

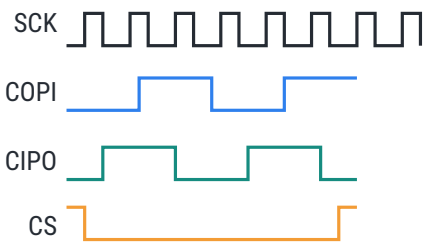


Kilka urządzeń SPI

SCK, COPI i CIPO mogą być wspólne. Każde urządzenie dostaje natomiast własną linię **CS**.



Transmisja SPI — krok po kroku



1. Controller ustawia **CS = LOW** dla wybranego urządzenia.
2. Generuje kolejne impulsy zegara **SCK**.
3. Na linii **COPI** wysyła bity do peripheral.
4. Jednocześnie może odbierać bity na **CIPO**.
5. Po zakończeniu transmisji ustawia **CS = HIGH**.

Ważne

SCK, COPI i CIPO mogą być współdzielone. W typowym SPI tylko urządzenie z aktywnym CS uczestniczy w transmisji.

Dokładny moment próbkowania zależy od trybu SPI (CPOL/CPHA); w tym kursie biblioteka konfiguruje go za nas.

SPI na Arduino UNO R4 WiFi

Sygnał	Nazwa alternatywna	Pin
SCK	clock	D13
CIPO	MISO	D12
COPI	MOSI	D11
CS / SS	chip select	D10

Warto zapamiętać

CIPO/COPI to nowsze, neutralne nazwy. W wielu bibliotekach i materiałach nadal zobaczycie **MISO/MOSI**.

I^2C czy SPI? — porównanie praktyczne

	I^2C	SPI
Przewody	SDA + SCL	SCK + COPI + CIPO + CS
Wybór urządzenia	adres	stan niski na konkretnej linii CS
Wiele urządzeń	łatwo, jeśli adresy się nie konfliktują	więcej urządzeń $\rightarrow$ więcej linii CS
Prędkość	zwykle mniejsza	zwykle większa
Kierunek danych	jedna linia SDA w obie strony	osobne linie COPI/CIPO
Typowe użycie	czujniki, RTC, małe peryferia	RFID, pamięci, wyświetlacze, szybkie ADC

Nie wybieramy magistrali „z przyzwyczajenia” — zwykle urządzenie już narzuca interfejs.

RC522 — czytnik RFID jako przykład urządzenia SPI

- czytnik komunikuje się z kartą bezprzewodowo,
- z Arduino komunikuje się przez **SPI**,
- biblioteka odczytuje m.in. identyfikator UID karty,
- w laboratorium używamy linii CS na D10 i RST na D9,
- czujnik zasilany jest z 3.3V.



Źródło: Botland — moduł MF-RC522

Biblioteka MFRC522 — znowu ten sam wzorzec

```
#include <SPI.h>
#include <MFRC522.h>

const int SS_PIN = 10;
const int RST_PIN = 9;

MFRC522 rfid(SS_PIN, RST_PIN);

void setup() {
  Serial.begin(115200);
  SPI.begin();
  rfid.PCD_Init();
}
```

- `SPI.begin()` uruchamia magistralę,
- `rfid.PCD_Init()` inicjalizuje konkretny czytnik.

UID karty to kilka bajtów — nie jedna „magiczna liczba”

Przykładowy UID

```
A3  17  4C  92
```

- każdy element jest bajtem zapisanym szesnastkowo,
- biblioteka podaje długość UID i tablicę kolejnych bajtów,
- do porównania karty najlepiej porównać **wszystkie bajty**, a nie tylko pierwszy.

Bezpieczeństwo

UID prostego tagu RFID nie powinien być traktowany jako silne uwierzytelnienie. Na laboratorium służy do identyfikacji i nauki komunikacji.

Odczyt UID — fragment, który warto umieć przeczytać

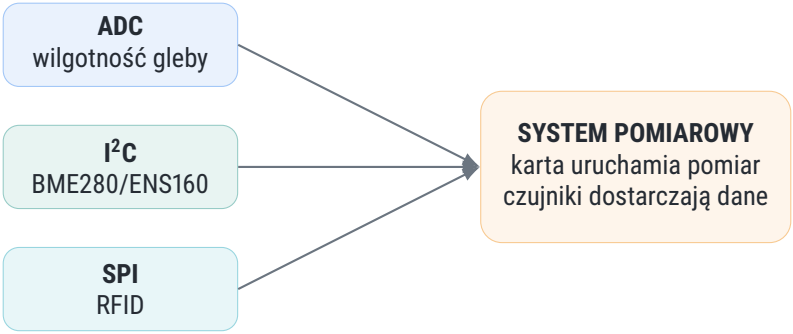
```
if (!rfid.PICC_IsNewCardPresent()) {
  return;
}

if (!rfid.PICC_ReadCardSerial()) {
  return;
}

Serial.print("UID: ");
for (byte i = 0; i < rfid.uid.size; i++) {
  Serial.print(rfid.uid.uidByte[i], HEX);
  Serial.print(" ");
}
Serial.println();
```

`return;` kończy bieżące wykonanie funkcji `loop()` i program wróci do jej początku przy następnym obiegu.

Laboratorium 3 składa trzy sposoby pozyskiwania informacji



Ten sam mikrokontroler może jednocześnie obsługiwać **różne sposoby komunikacji z otoczeniem**.

Które zdanie najlepiej opisuje różnicę między I^2C i SPI?

1. SPI zawsze używa tylko dwóch przewodów,
2. I^2C wybiera urządzenie adresem, a SPI typowo linią CS,
3. I^2C nie może obsługiwać więcej niż jednego urządzenia,
4. w SPI linia zegara nie jest potrzebna.

?

Najpierw spróbuj odpowiedzieć bez zaglądania do kodu.

Które zdanie najlepiej opisuje różnicę między I^2C i SPI?

1. SPI zawsze używa tylko dwóch przewodów,
2. **I^2C wybiera urządzenie adresem, a SPI typowo linią CS,**
3. I^2C nie może obsługiwać więcej niż jednego urządzenia,
4. w SPI linia zegara nie jest potrzebna.

?

Najpierw spróbuj odpowiedzieć bez zaglądania do kodu.

4. 1-Wire — wiele czujników na jednej linii danych

Minimalna liczba przewodów, unikalne adresy urządzeń i ścisłe zależności czasowe.

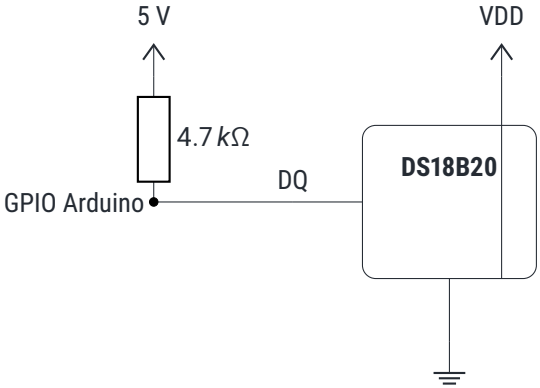
Po co jeszcze 1-Wire?

- magistrala została zaprojektowana do prostych, wolniejszych urządzeń,
- komunikacja odbywa się po **jednej linii danych** + GND,
- wiele urządzeń może współdzielić tę samą linię,
- każde urządzenie ma własny **unikalny identyfikator**,
- klasyczny przykład: cyfrowy termometr **DS18B20**.

Nie mylić

„1-Wire” opisuje protokół. W praktycznym układzie zwykle nadal mamy także **zasilanie VDD i GND**.

Podstawowe połączenie 1-Wire — linia potrzebuje pull-up



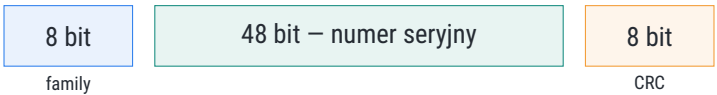
- urządzenia mają wyjścia typu open-drain / trójstanowe,
- rezystor podciągający ustala stan HIGH, gdy nikt nie wymusza LOW.

Skąd wiadomo, który termometr odpowiada? — 64-bitowy ROM

Każdy DS18B20 ma unikalny 64-bitowy kod

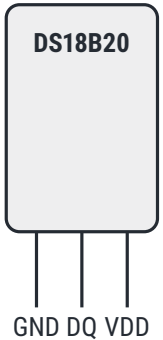
```
family code | serial number | CRC
```

- dzięki temu wiele czujników może być połączonych równolegle,
- controller może adresować konkretny egzemplarz,
- identyfikator jest zapisany w pamięci ROM urządzenia.



DS18B20 — czujnik cyfrowy, który sam wykonuje pomiar temperatury

- mierzy temperaturę od -55°C do +125°C,
- programowalna rozdzielczość 9–12 bitów,
- wynik przechowuje w pamięci scratchpad,
- komunikacja 1-Wire i unikalny kod 64-bitowy,
- może pracować z normalnym zasilaniem lub w trybie parasite power.

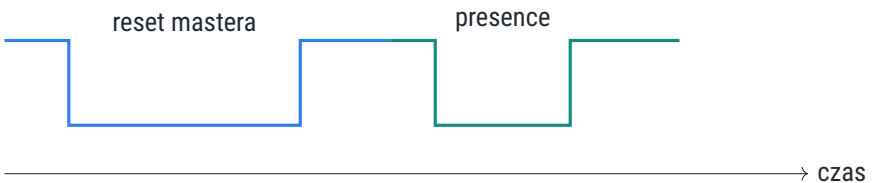


Parasite power

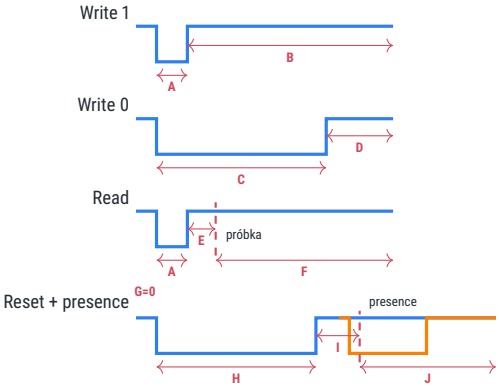
Bez osobnego zasilania VDD czujnik pobiera energię z linii DQ. To upraszcza okablowanie, ale daje mniejszy zapas energetyczny.

1-Wire komunikuje się przez precyzyjne „okna czasowe”

- controller rozpoczyna cykl impulsem **reset**,
- urządzenie odpowiada impulsem **presence**,
- kolejne bity są przesyłane w odpowiednio zdefiniowanych szczelinach czasowych,
- dlatego ręczna implementacja protokołu jest bardziej wymagająca niż zwykłe `digitalRead()`.



1-Wire — podstawowe operacje i zalecane czasy



Parametr	[µs]	Znaczenie
A	6	niski impuls: write 1 / read
B	64	dopełnienie write 1
C	60	niski impuls: write 0
D	10	dopełnienie write 0
E	9	opóźnienie do próbki
F	55	dopełnienie read
G	0	opóźnienie przed resetem
H	480	reset mastera
I	70	czas do próbki presence
J	410	recovery po resecie

Najważniejsza idea

Litery A–J opisują kolejne **odcinki czasowe** protokołu, a nie poziomy logiczne.

Na podstawie: Maxim Application Note 126, Fig. 1 i Table 2.

Na szczęście korzystamy z bibliotek — dwa obiekty, dwie role

```
#include <OneWire.h>
#include <DallasTemperature.h>

const int ONE_WIRE_PIN = 4;

OneWire oneWire(ONE_WIRE_PIN);
DallasTemperature sensors(&oneWire);
```

- `oneWire` reprezentuje samą **magistralę 1-Wire**,
- `sensors` dodaje obsługę czujników temperatury Dallas/Maxim,
- zapis `&oneWire` przekazuje bibliotece informację, z której magistrali ma korzystać.

Pełny przykład: odczyt temperatury DS18B20

```
#include <OneWire.h>
#include <DallasTemperature.h>

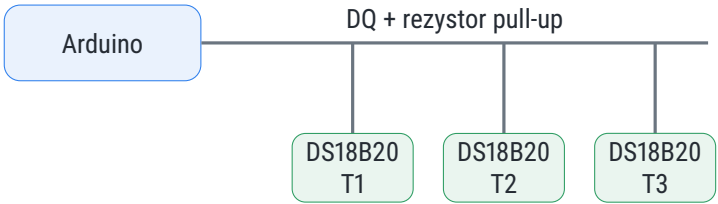
const int ONE_WIRE_PIN = 4;
OneWire oneWire(ONE_WIRE_PIN);
DallasTemperature sensors(&oneWire);

void setup() {
  Serial.begin(115200);
  sensors.begin();
}

void loop() {
  sensors.requestTemperatures();
  float tempC = sensors.getTempCByIndex(0);

  Serial.print("T = ");
  Serial.print(tempC);
  Serial.println(" C");
  delay(1000);
}
```

Wiele DS18B20 na jednej linii — główna zaleta 1-Wire



- wszystkie czujniki współdzielą fizyczną linię,
- unikalne 64-bitowe adresy pozwalają odczytać konkretny egzemplarz,
- `getTempCByIndex(0)` jest wygodne w prostym układzie, ale w większym systemie lepiej korzystać z adresów urządzeń.

I²C, SPI i 1-Wire — trzy różne kompromisy

	I²C	SPI	1-Wire
Linie danych	SDA + SCL	SCK, COPI, CIPO + CS	jedna DQ
Wybór urządzenia	adres	CS	unikalny 64-bit ROM
Szybkość	średnia	wysoka	mała
Pull-up	SDA/SCL	zwykle nie dla linii danych	DQ
Typowe urządzenia	czujniki, RTC	RFID, pamięci, ekrany	termometry, proste sensory

Wybór interfejsu to zawsze kompromis: **przewody, szybkość, liczba urządzeń i złożoność**.

Dlaczego kilka DS18B20 może pracować na tej samej linii danych?

1. każdy ma inny pin GPIO,
2. każdy ma unikalny 64-bitowy identyfikator,
3. magistrala nie potrzebuje adresowania,
4. wszystkie czujniki zawsze zwracają ten sam wynik.

?

Najpierw spróbuj odpowiedzieć bez zaglądania do kodu.

Dlaczego kilka DS18B20 może pracować na tej samej linii danych?

1. każdy ma inny pin GPIO,
2. **każdy ma unikalny 64-bitowy identyfikator,**
3. magistrala nie potrzebuje adresowania,
4. wszystkie czujniki zawsze zwracają ten sam wynik.

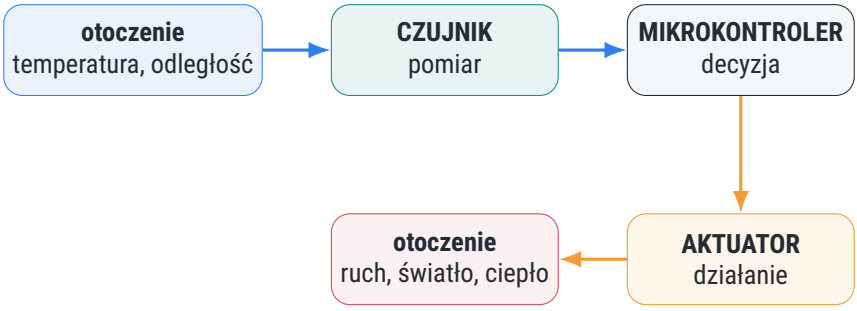
?

Najpierw spróbuj odpowiedzieć bez zaglądania do kodu.

5. Aktuator — system zaczyna działać na otoczenie

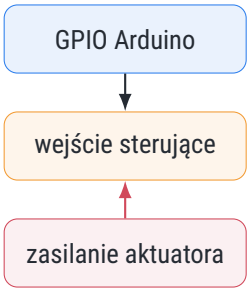
Czujnik dostarcza informacji. Aktuator zamienia decyzję programu na ruch, światło lub inną akcję.

Czujnik i aktuator — kierunek informacji jest odwrotny

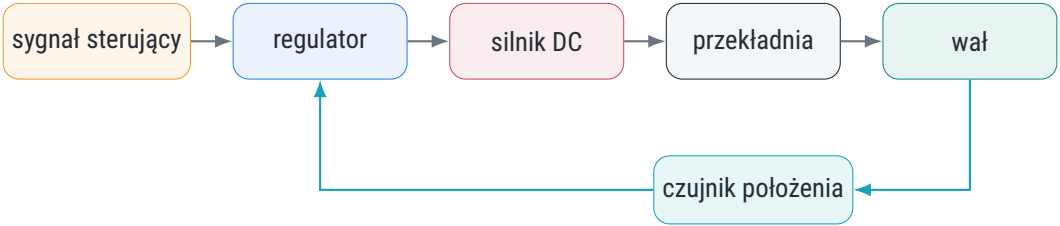


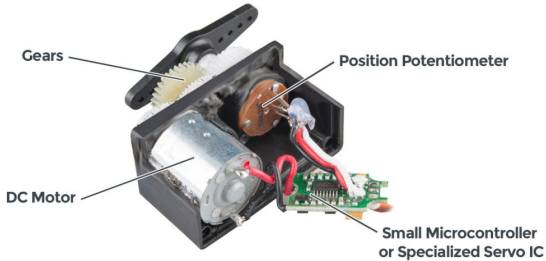
GPIO wysyła sygnał sterujący — nie zasila dużego obciążenia

- pin GPIO ma ograniczony prąd,
- silnik lub serwo może potrzebować znacznie więcej energii,
- GPIO przekazuje przede wszystkim **informację sterującą**,
- większe obciążenia wymagają odpowiedniego zasilania i często drivera / tranzystora.



Co jest w środku typowego serwomechanizmu?



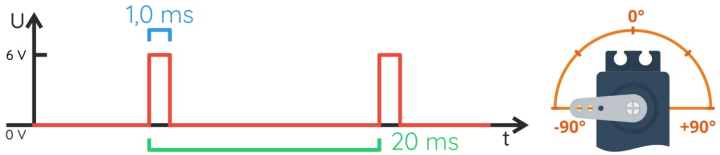


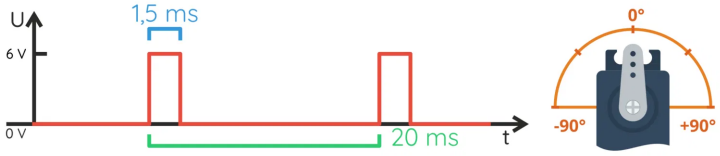
Źródło: https://www.sparkfun.com/servos?srsltid=AYCX9vJVXgBQmskY1K7tX9h8kJiWo1Ku28VKeuShaqdRCw0rwSQRpgZY

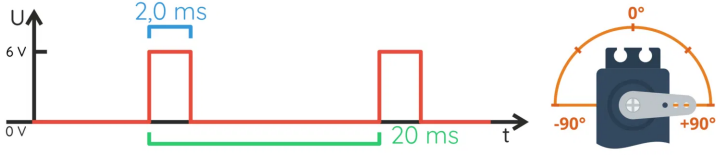
Mikrokontroler nie steruje bezpośrednio silnikiem — zadaje **położenie**, a regulator serwa realizuje je wewnętrznie.

Serwo — położenie jest zakodowane szerokością impulsu

- impulsy są powtarzane mniej więcej co **20 ms**,
- zmienia się głównie czas trwania stanu HIGH,
- dokładne zakresy czasu trwania impulsu zależą od konkretnego modelu serwa.







Źródło: https://forbot.pl/blog/leksykon/serwomechanizm

Pełny prosty przykład: $0^\circ \rightarrow 90^\circ \rightarrow 180^\circ$

```cpp
#include <Servo.h>

Servo myServo; //tworzenie obiektu
const int SERVO_PIN = 6;

void setup() {
  myServo.attach(SERVO_PIN); //konfiguruje, ktory pin bedzie uzywany do sterowania serwem
}

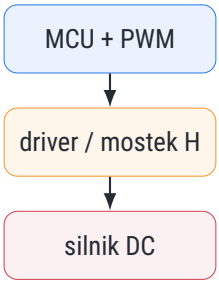
void loop() {
  myServo.write(0); //ustawienie polozenia serwa
  delay(1000);

  myServo.write(90);
  delay(1000);

  myServo.write(180);
  delay(1000);
}
```

Inny przykład aktuatora: silnik DC

- silnik DC nie ustawia konkretnego kąta — obraca się ciągle,
- mikrokontroler zwykle steruje nim przez **driver** lub mostek H,
- PWM pozwala regulować średnią moc, a więc w praktyce prędkość,
- aby zmieniać kierunek obrotów, potrzebujemy dodatkowego układu wykonawczego.



Przykład: sterowanie silnikiem DC przez mostek H

```arduino
const int PWM_PIN = 6;  // EN/PWM drivera
const int IN1 = 7;
const int IN2 = 8;

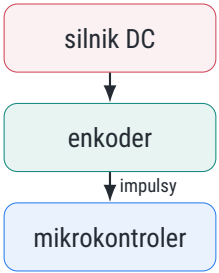
void setup() {
  pinMode(PWM_PIN, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
}

void loop() {
  digitalWrite(IN1, HIGH);   // kierunek 1
  digitalWrite(IN2, LOW);
  analogWrite(PWM_PIN, 160); // predkosc
  delay(2000);
  analogWrite(PWM_PIN, 0);   // stop
  delay(500);
  digitalWrite(IN1, LOW);    // kierunek 2
  digitalWrite(IN2, HIGH);
  analogWrite(PWM_PIN, 160);
  delay(2000);
}
```

GPIO steruje driverem; silnika nie zasilamy bezpośrednio z pinu Arduino. Driver i Arduino powinny mieć wspólną masę.

Aby znać położenie lub prędkość, dodajemy enkoder

- enkoder nie jest aktu-atorem, tylko **czujnikiem sprzężenia zwrotnego**,
- dostarcza impulsy związane z ruchem wału,
- z liczby impulsów możemy wyznaczyć położenie,
- z częstotliwości impulsów — prędkość obrotową.



Przykład: zliczanie impulsów enkodera inkrementalnego

```
const int ENC_A = 2;
const int ENC_B = 3;
volatile long encoderCount = 0;

void encoderISR() {
  if (digitalRead(ENC_B) == HIGH)
    encoderCount++;
  else
    encoderCount--;
}

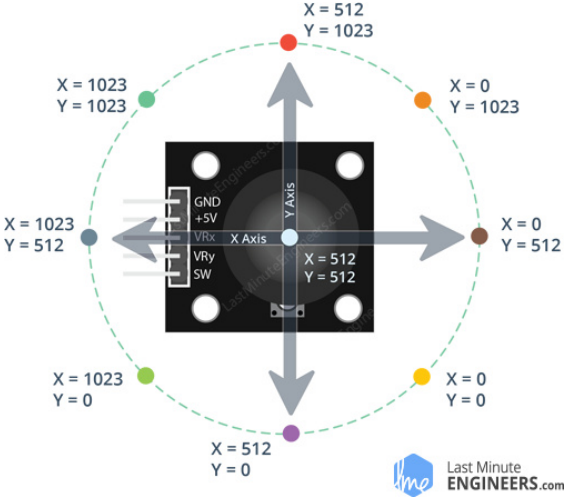
void setup() {
  Serial.begin(115200);
  pinMode(ENC_A, INPUT_PULLUP);
  pinMode(ENC_B, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(ENC_A),
                  encoderISR, RISING);
}

void loop() {
  Serial.println(encoderCount);
  delay(100);
}
```

Zliczamy jedno zbocze kanału A; stan kanału B rozróżnia kierunek. Bardziej rozbudowany dekoder może wykorzystywać oba zbocza obu kanałów.

Joystick analogowy — w środku są dwa potencjometry

- VRX zmienia napięcie wraz z ruchem w osi X,
- VRY zmienia napięcie wraz z ruchem w osi Y,
- dla Arduino są to po prostu dwa wejścia analogowe,
- środek joysticka daje wartości w okolicy środka zakresu ADC.



Źródło:
https://forum.arduino.cc/t/code-trigonometry-joystick-how-to-map-x-y-values/585018

Joystick → kąt serwa: ponownie map() i constrain()

```
int raw = analogRead(A0);

int angle = map(raw, rawMin, rawMax, 0, 180);
angle = constrain(angle, 0, 180);

myServo.write(angle);
```

- `rawMin` i `rawMax` warto najpierw **zmierzyć** dla konkretnego joysticka,
- dzięki temu cały rzeczywisty zakres ruchu mapujemy na 0–180°,
- to ten sam schemat co kalibracja czujnika wilgotności: **pomiar** → **przeskalowanie** → **ograniczenie**.

Pełny przykład: sterowanie serwem joystickiem

```ino
#include <Servo.h>

Servo myServo;
const int SERVO_PIN = 6;
const int JOY_X = A0;
const int rawMin = 100;
const int rawMax = 1000;

void setup() {
  myServo.attach(SERVO_PIN);
}

void loop() {
  int raw = analogRead(JOY_X);
  int angle = map(raw, rawMin, rawMax, 0, 180);
  angle = constrain(angle, 0, 180);
  myServo.write(angle);
  delay(20);
}
```

Wartości `rawMin/rawMax` są przykładowe — na laboratorium wyznaczycie je dla swojego egzemplarza.

Co przede wszystkim zmienia się w sygnale sterującym serwem, gdy zmieniamy zadany kąt?

1. napięcie zasilania serwa,
2. szerokość impulsu,
3. liczba przewodów,
4. adres I^2C serwa.

?

Najpierw spróbuj odpowiedzieć bez zaglądania do kodu.

Co przede wszystkim zmienia się w sygnale sterującym serwem, gdy zmieniamy zadany kąt?

1. napięcie zasilania serwa,
2. **szerokość impulsu**,
3. liczba przewodów,
4. adres I^2C serwa.

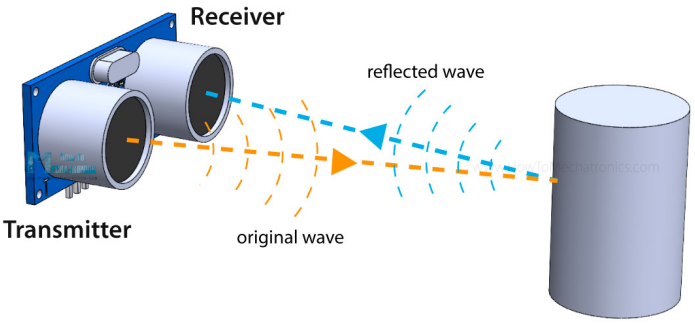
?

Najpierw spróbuj odpowiedzieć bez zaglądania do kodu.

6. HC-SR04 i automatyczny układ sterowania

Mierzymy czas przelotu fali, a potem wykorzystujemy wynik do sterowania serwem.

HC-SR04 — pomiar czasu przelotu fali ultradźwiękowej

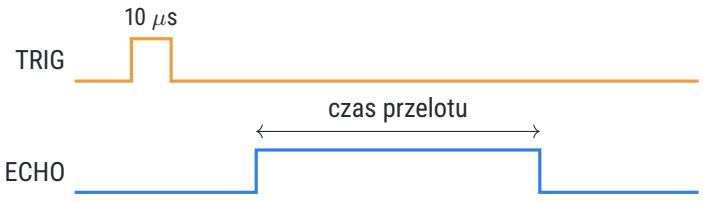


Źródło: https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/

- Czujnik mierzy czas podróży **tam i z powrotem**. Dlatego przy obliczaniu odległości dzielimy drogę przez 2

TRIG i ECHO — dwa sygnały o różnych rolach

- na **TRIG** wysyłamy impuls HIGH trwający co najmniej 10 μs,
- moduł emituje serię impulsów ultradźwiękowych około 40 kHz,
- **ECHO** pozostaje HIGH przez czas związany z drogą fali do obiektu i z powrotem.



Źródło: Laboratorium PSW, labe.engined.eu

Od czasu do odległości — skąd bierze się dzielenie przez 58?

Zależność fizyczna

$$d = \frac{v\,t}{2}$$

Dla prędkości dźwięku około $v = 0.0343\,\mathrm{cm}/\mu\mathrm{s}$:

$$d[\mathrm{cm}] \approx \frac{0.0343 \cdot t[\mu s]}{2}$$

$$\frac{0.0343}{2} \approx 0.01715 \quad \Longrightarrow \quad d[\mathrm{cm}] \approx 0.01715\, t[\mu s]$$

$$0.01715 \approx \frac{1}{58.3} \quad \Longrightarrow \quad d[\mathrm{cm}] \approx \frac{t[\mu s]}{58}$$

Przykład

Jeżeli ECHO jest HIGH przez 1166 μs, to $d \approx 1166/58 \approx 20.1\,\mathrm{cm}$.

Pełna funkcja pomiaru odległości

```
const int TRIG_PIN = 7;
const int ECHO_PIN = 8;

float readDistanceCm() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);

  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  unsigned long duration = pulseIn(ECHO_PIN, HIGH, 30000);

  if (duration == 0) {
    return -1.0;  // brak echa w zadanym czasie
  }

  return duration / 58.0; // przyblizenie z dokumentacji HC-SR04
}
```

Po co timeout w pulseIn()?

- bez echa program nie powinien czekać w nieskończoność,
- trzeci argument `pulseIn(..., 30000)` ogranicza maksymalny czas oczekiwania,
- zwrócone `0` możemy potraktować jako **błąd / brak poprawnego pomiaru**.

Dobra praktyka

Błąd pomiaru nie powinien automatycznie oznaczać „odległość = 0 cm”. To są dwie zupełnie różne sytuacje.

System wbudowany musi umieć obsłużyć także **brak danych**.

Dziękuję za uwagę!

Na laboratoriach: ADC i czujniki, SPI/RFID, serwomechanizm, joystick i HC-SR04.