

## Komunikacja łączem szeregowym Przetwornik analogowo-cyfrowy

Opracował opiekun przedmiotu dr inż. Krzysztof Chudzik

Wydział Informatyki i Telekomunikacji  
Politechnika Wrocławskiego

Wrocław, 2021.08.26 21:30:56

---

### Spis treści

<b>1 Komunikacja z wykorzystaniem łącza szeregowego</b>	<b>1</b>
1.1 Działanie łącza szeregowego w Arduino UNO . . . . .	1
1.2 Wysyłanie danych do komputera PC przez łącze szeregowe . . . . .	3
1.3 Odczyt danych wysyłanych przez łącze szeregowe Arduino po stronie komputera PC . . . . .	3
1.4 Sterowanie pracą Arduino za pomocą komend wysyłanych przez łącze szeregowe z PC . . . . .	5
<b>2 Przetwornik analogowo-cyfrowy</b>	<b>6</b>
2.1 Czym jest przetwornik analogowo-cyfrowy (ADC) . . . . .	6
2.2 Potencjometr - Zasada działania potencjometru . . . . .	6
2.3 Odczytu wartości napięcia przez ADC . . . . .	8
2.4 Przeskalowywanie odczytu ADC . . . . .	9

---

### Lista zadań

1 Wysyłanie danych do komputera PC przez łącze szeregowe . . . . .	4
2 Sterowanie pracą Arduino za pomocą komend wysyłanych przez łącze szeregowe z PC . . . . .	5
3 Odczyt wartości napięcia zadanego potencjometrem . . . . .	9
4 Monitorowanie napięcia wejściowego portu za pomocą narzędzia Serial Plotter z Arduino IDE . . . . .	9

---

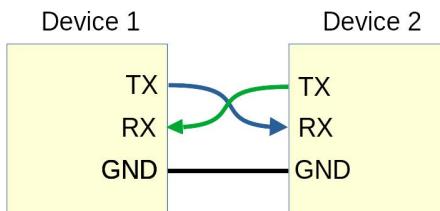
## 1 Komunikacja z wykorzystaniem łącza szeregowego

### 1.1 Działanie łącza szeregowego w Arduino UNO

Standardów łącz cyfrowych jest wiele. Na laboratorium z zestawem Arduino będziemy wykorzystywać transmisję z wykorzystaniem UART - Universal Asynchronous Receiver/Transmitter (uniwersalny odbiornik/nadajnik asynchroniczny). Transmisja asynchroniczna oznacza, że nie wykorzystuje sygnału zegarowego, a urządzenia synchronizują się wykorzystując własne precyzyjne zegary i oraz zmiany stanów napięciowych na łączu, to znaczy rozpoznają kiedy, na przykład, rozpoczęła się lub zakończyła transmisja ramki danych, bitu, itp.

Do komunikacji z wykorzystaniem UART przewidziano na płytce Arduino dwa piny D0/RX oraz D1/TX. Komunikacja z wykorzystaniem łącza szeregowego odbywa się poprzez wysyłanie i/lub odbieranie jednego bitu w danej chwili czasowej. Wyprowadzenie RX jest wejściem, natomiast TX jest wyjściem sygnałowym.

Jeśli obie linie RX oraz TX są podłączone możliwa jest transmisja dwukierunkowa w tym samym czasie (*full duplex*). Podłączenie dwóch urządzeń wygląda jak na rysunku 1. Strzałki oznaczają kierunek przepływu danych. Piny (linie) RX oraz TX w urządzeniu (mikrokontrolerze) nazywa się łącznie portem UART.



Rysunek 1: Połączenie wyprowadzeń RX i TX w transmisji szeregowej.

Na płytce Arduino UNO, są to oczywiście wyprowadzenia mikrokontrolera ATmega328P, który wykorzystuje je do połączenia z komputerem PC. Dzieje się to za pośrednictwem dodatkowego mikrokontrolera zamontowanego na płytce Arduino w tym celu. Ten dodatkowy mikrokontroler odpowiedzialny jest za konwersję sygnału UART do formatu umożliwiającego przesyłanie go przez łącze USB do komputera. Zachodzi tam, między innymi, proces enkapsulacji połączenia (komunikacji, danych) UART w połączenie po łączu USB. Po stronie komputera zachodzi działanie odwrotne i połączenie UART jest wyodrębniane (dekapsulacja) z danych transmitowanych po USB. Następnie, za pomocą odpowiednich sterowników, to wyodrębnione połączenie jest udostępniane jako wirtualne urządzenie łącza szeregowego. W systemie Windows widoczne jest ono pod nazwą COM $x$ , gdzie  $x$  jest numerem przydzielonym temu urządzeniu przez system. Arduino w systemie Linux jest widoczne, zazwyczaj, jako urządzenie /dev/ttym $x$ , gdzie  $x$  jest numerem przydzielonym temu urządzeniu przez system. W przypadku systemu Linux nie jest to reguła ścisła.

Uwaga: Arduino Uno piny D0/RX oraz D1/TX są używane do komunikacji z komputerem również podczas wgrywania programów do mikrokontrolera. Należy rozważyć podłączanie do nich inne elementy, ponieważ może to zakłócić komunikację, w tym spowodować, że program nie zostanie wgrany do Arduino lub zostanie wgrany nieprawidłowo.

Podstawowe łącze szeregowe Arduino UNO jest łączem realizowanym sprzętowo, to znaczy w mikrokontrolerze jest system sprzętowy odpowiedzialny za obsługę komunikacji z wykorzystaniem tego łącza. Nie musimy samodzielnie ustawać poziomów logicznych na pinie TX i interpretować tego co dzieje się na RX, dbać o czasy tych operacji, itd. Robi to mikrokontroler. Co więcej, w tym samym czasie mikrokontroler niezależnie wykonuje swój program.

Niemniej jednak, w Arduino UNO jest to pojedynczy port UART sprzętowy. Jeśli zachodzi potrzeba, kolejne można emulować programowo. Jest to, oczywiście, mocno obciążające dla mikrokontrolera, ponieważ wszystkie działania portu muszą zostać zrealizowane programowo. Ograniczona jest też funkcjonalność portu programowego, gdy porównamy go ze sprzętowym. Do realizacji programowego portu stworzono, między innymi, bibliotekę **SoftwareSerial**. Więcej informacji w dokumentacji biblioteki **SoftwareSerial** ([link](#)).

Dane przesyłane przez UART podzielone są na ramki. Parametry ramki mogą być konfigurowalne. Ramka składa się z następujących bitów:

- Bit startu - Informuje o rozpoczęciu transmisji ramki. Na łączu jest to zawsze stan logiczny napięciowy niski (0),
- Bity danych - Reprezentują dane do przesłania. Ich liczba jest konfigurowalna i przyjmuje wartości od 5 do 9.
- Bit parzystości - zazwyczaj jest ustawiany jeden z trzech stanów (inne pominiemy):
  - None (N) - oznacza, że nie jest wysyłany żaden bit parzystości.
  - Odd (O) - oznacza, że bit parzystości jest ustawiony tak, że liczba "logicznych jedynek" musi być nieparzysta.
  - Even (E) - oznacza, że bit parzystości jest ustawiony tak, że liczba "logicznych jedynek" musi być parzysta.
- Bity stopu - pozwalają sprzętowi odbierającemu sygnał wykryć koniec ramki i ponownie zsynchronizować się ze strumieniem znaków. Na łączu jest to zawsze stan logiczny napięciowy wysoki (1). Ilość bitów jest konfigurowalna i może wynosić 1 lub 1,5 (to nie błąd, stan stopu zajmuje okres równy jeden i pół okresu transmisji jednego bita) lub 2 bity.

Podstawowy parametr, który należy skonfigurować, to szybkość (prędkość) transmisji wyrażona w bitach na sekundę. W praktyce, najczęściej wykorzystywane prędkości to 9600, 19200, 57600 i 115200 bps (bitów na sekundę). Przy definiowaniu prędkości stwierdzamy, jaka będzie szybkość transmisji bitów na łączu. Jeśli transmitujemy bity danych w liczbie osiem, a skonfigurujemy brak bitu parzystości, jeden bit stopu i dodamy wymagany bit startu, to ramka liczy sobie 10 bitów. Czyli same dane zajmują 80% czasu transmisji ramki. W ten sposób, przy prędkości 9600 bps mamy maksymalną teoretyczną prędkość transmisji danych na poziomie 7680 bps. Do tego dochodzą różne problemy i błędy w transmisji, które trzeba obsłużyć w dużej mierze programowo, co dodatkowo obniża szybkość transferu danych.

Gdy uwzględnimy wszystkie parametry transmisji, to zazwyczaj zapisujemy je w łącznej notacji, której przykładem może być 115200-8-N-1, oznaczający szybkość 115,2 kbps bez parzystości z jednym bitem stopu.

## 1.2 Wysyłanie danych do komputera PC przez łącze szeregowe

Szkic z kodem (plik z rozszerzeniem „ino”) w Arduino IDE, czy w VS Code, jest tylko fragmentem większej kompilowanej całości, która zawiera wszystko co określiliśmy na nasz użytek mianem języka Arduino. Zawiera on zbiór funkcji, stałych, itp., ułatwiających programowanie i ukrywających pewne złożone operacje, które trzeba zaprogramować programując bezpośrednio mikrokontroler. Obsługa komunikacji przez łącze szeregowe do komputera z wykorzystaniem łącza USB nie jest tu wyjątkiem.

Wykorzystanie łącza szeregowego w Arduino UNO odbywa się poprzez predefiniowany obiekt reprezentujący to łącze o identyfikatorze `Serial`. Proszę zapoznać się z dokumentacją obiektu `Serial` ([link](#)).

Pracę łącza rozpoczynamy od wywołania metody `Serial.begin()`. Przyjmuje ona dwa argumenty. Pierwszy jest obligatoryjny i jest to szybkość transmisji. Drugi jest opcjonalny i zawiera pozostałe opcje konfiguracyjne: ilość bitów danych w ramce, parzystość i bit stopu. Są one opisane identyfikatorami, więc łatwo je wykorzystać. Konfiguracja domyślna to `SERIAL_8N1`. Więcej informacji w dokumentacji metody `Serial.begin()` ([link](#)).

Następnie warto sprawdzić, czy połączenie zostało otwarte. Można zrobić to z użyciem instrukcji warunkowej wywołanej bezpośrednio na obiekcie `Serial`, na przykład, `if(Serial){...}` lub `while(!Serial){...}`, jak opisano to na stronie dokumentacji o tytule „`if(Serial)`” ([link](#)).

Teraz można rozpoczęć wysyłanie danych. Dane binarne można wysyłać za pomocą funkcji `Serial.write()` ([link](#)). Dane tekstowe i liczby wypisane jako tekst wysyłamy przy pomocy funkcji `Serial.print()` ([link](#)) i `Serial.println()` ([link](#)). Dwie ostatnie można wywołać ze specyfikatorami, wskazującymi jak konwertować liczby nałańcuchy tekstowe.

Więcej informacji na temat manipulowaniałańcuchami testowymi można uzyskać w dokumentacji klasy `String` ([link](#)).

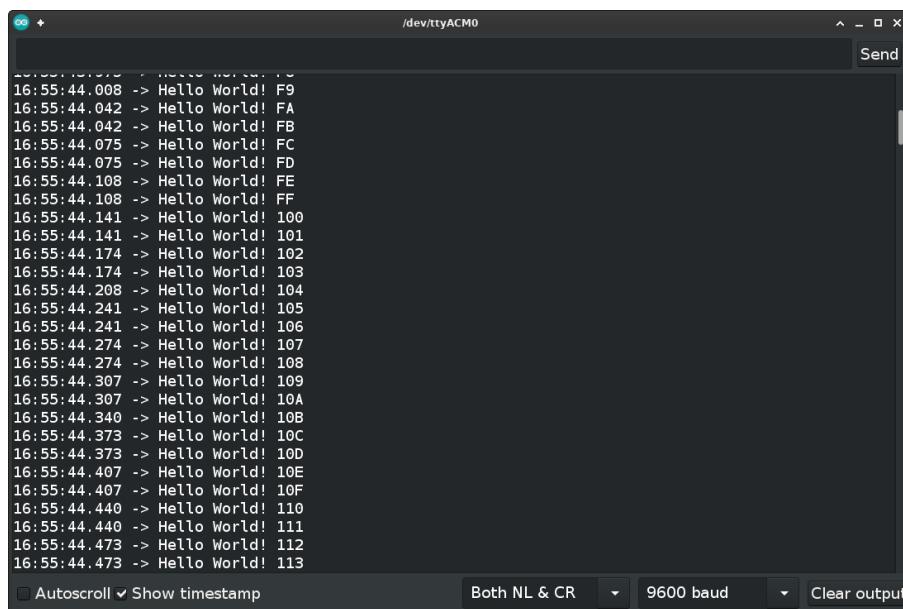
Przykładowy, bardzo prosty program wysyłający dane tekstowe i liczbę w postaci szesnastkowej zawiera kod 1;

Kod 1: Prosty program typu "Hello World!" ilustrujący wysyłanie danych do komputera PC przez łącze szeregowe.

```
1 void setup()
2 {
3     Serial.begin(9600);
4     while (!Serial)
5     { /* just wait */
6     }
7 }
8
9 int i = 0;
10 void loop()
11 {
12     Serial.print("Hello World! ");
13     Serial.println(i++, HEX);
14 }
```

## 1.3 Odczyt danych wysyłanych przez łącze szeregowe Arduino po stronie komputera PC

Wyniki działania programu można obserwować przy pomocy narzędzia z Arduino IDE, dostępnego przez Menu: Tools > Serial Monitor lub skrót klawiszowy Ctrl+Shift+M. Widok działania tego narzędzia dla kodu 1 widoczny jest na rysunku 2.



Rysunek 2: Okno narzędzia Serial Monitor z Arduino IDE.

Po włączeniu tego okna należy skonfigurować przede wszystkim prędkość transmisji. Miejsce gdzie na rysunku 2 wpisana jest wartość „9600 baud”. Opcja „Autoscroll” decyduje, czy okno samodzielnie przewija się w miarę napływu nowych linii tekstu. „Show timestamp” na początku każdej linii tekstu dodaje znacznik czasowy. W liście rozwijanej, w której na rysunku 2 widnieje napis „Both NL & CR” wybieramy jak mają być traktowane znaki nowej linii, ponieważ systemy Windows i Linux posługują się inną konwencją.

Serial Monitor nie jest jedynym programem, który można wykorzystać do komunikacji szeregowej z Arduino. Proponuję zapoznać się z artykułem „[4 alternatywy dla monitora portu szeregowego Arduino](#)” ([link](#)).

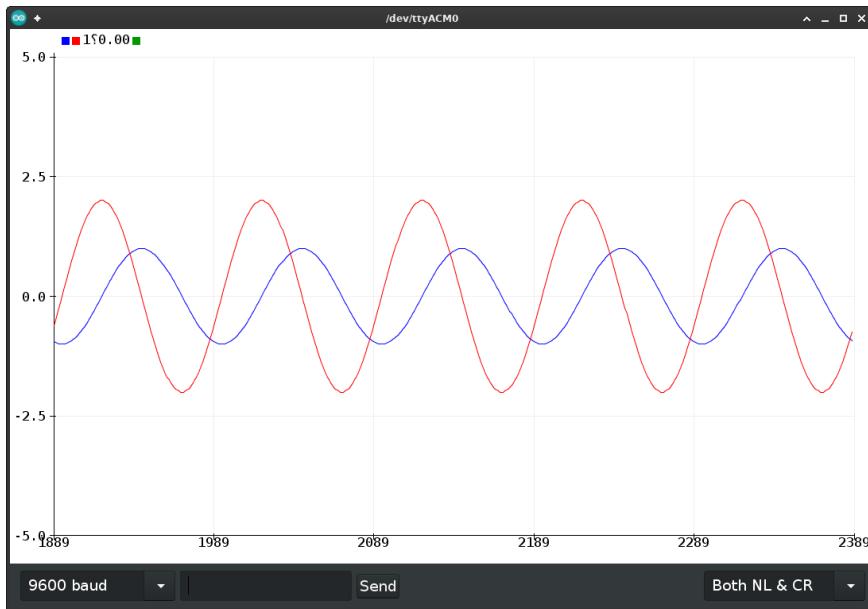
Jeśli przesyłane są dane liczbowe, to można je zwizualizować poprzez narzędzie dostępnego przez menu **Tools > Serial Plotter** lub skrót klawiszowy **Ctrl+Shift+L**. Narzędzie rysuje linie wykresu odpowiadające kolejnym danym wysłanym w jednej linii. Kod 2 przesyła wartości funkcji  $\sin(x)$  oraz  $2 \cos(x)$ .

Kod 2: Prosty program ilustrujący wysyłanie danych do komputera PC przez łącze szeregowe.

```

1 void setup()
2 {
3     Serial.begin(9600);
4     while (!Serial)
5     { /* just wait */
6     }
7 }
8
9 int i = 0;
10 void loop()
11 {
12     double x = TWO_PI * i / 100;
13     String str = String(sin(x)) + " " + String(2 * cos(x));
14     Serial.println(str);
15     i++;
16 }
```

Wynik działania kodu 2 w oknie narzędzi Serial Monitor jest widoczny na rysunku 3.



Rysunek 3: Narzędzie Serial Plotter z Arduino IDE.

## Zadanie 1: Wysyłanie danych do komputera PC przez łącze szeregowe

Napisz program, który będzie wysyłał informację o stanie przycisków czerwonego i zielonego przez łącze szeregowe do komputera PC. Sprawdź jaką maksymalną szybkość transmisji danych możesz skonfigurować. Spróbuj zwizualizować stany przycisków na wykresie w Serial Plotter

Jest to zadanie przykładowe. Prowadzący może zmodyfikować lub zmienić treść zadania.

Do wykresów z wartościami zmierzonymi przetwornikiem analogowo-cyfrowym, wróćmy w zadaniu 4.

Zainteresowanym bardziej złożonym wyświetlaniem danych przesyłanych przez różne urządzenia polecam przyjrzenie się programowi [Telemetry Viewer](#) ([link](#)). Na witrynie programu znajduje się instruktażowe wideo. Jego wykorzystanie jest też opisane w artykule „[Arduino: jak rysować rozbudowane wykresy na żywo?](#)” ([link](#)).

## 1.4 Sterowanie pracą Arduino za pomocą komend wysyłanych przez łącze szeregowe z PC

Przyjrzelismy się jak wygląda przesyłanie i wizualizacja danych przesyłanych z Arduino do komputera. Komunikacja może zachodzić również w drugą stronę. Możemy wpisywać dane w górne okienko dialogowe narzędzia Serial Monitor i to co wpiszymy, będzie wysyłane jako tekst do Arduino przez łącze szeregowe. Zobaczmy, jak można skonfigurować taką komunikację.

Przed kontynuowaniem dalszej lektury, proszę zapoznać się z funkcjami, których dotąd nie analizowaliśmy, do których odwołania znajdują się w dokumentach „Serial” ([link](#)) oraz „String()” ([link](#)). Proszę nie uczyć się niczego na pamięć, ale nabyc orientację, jakie funkcje mamy do dyspozycji w klasach Serial i String, oraz co można dzięki nim osiągnąć i umieć odnaleźć opisy, gdy zajdzie potrzeba analizy istniejącego kodu, lub pisania własnego kodu.

Postawmy następujące zadanie. Po wydaniu polecenia „on” wbudowana dioda LED zaświeci się. Po wydaniu polecenia „off” dioda zgaśnie. Program ma być niewrażliwy na duże i małe litery w poleceniach, to znaczy powinien zaakceptować też „On”, „ON”, a nawet „oN”. Przed analizą komendy powinien usunąć zbędne spacje, tabulacje, itp., na początku i końcu wprowadzonej komendy. W przypadku nieroznalezienia polecenia, powinien to sygnalizować przez łącze szeregowe do narzędzia Serial Monitor odpowiednim komunikatem.

Przykład rozwiązania ilustruje kod 3. Konfiguracja łącza szeregowego jest taka sama jak w poprzednich przykładach. Metoda `Serial.available()` dostarcza informacji czy oczekują dane do wczytania, i jeśli tak, to wczytujemy je po jednym wierszu metodą `Serial.readStringUntil()`, ze znakiem nowej linii jako argumentem. Kolejne kroki to pozbycie się zbędnych znaków i zamiana wszystkich znaków na małe litery. Gdy to zostało zrobione, wystarczy zidentyfikować komendę, albo poinformować, że wydana została nieprawidłowa komenda.

Kod 3: Prosty program ilustrujący sterowanie pracą Arduino z komputera PC.

```
1 void setup()
2 {
3     pinMode(LED_BUILTIN, OUTPUT);
4     digitalWrite(LED_BUILTIN, LOW);
5     Serial.begin(9600);
6     while (!Serial)
7     { /* just wait */
8     }
9 }
10
11 void loop()
12 {
13     if (Serial.available() > 0)
14     {
15         String command = Serial.readStringUntil('\n');
16         command.trim();
17         command.toLowerCase();
18
19         if (command == "on")
20         {
21             digitalWrite(LED_BUILTIN, HIGH);
22             Serial.println("LED ON");
23         }
24         else if (command == "off")
25         {
26             digitalWrite(LED_BUILTIN, LOW);
27             Serial.println("LED OFF");
28         }
29         else
30         {
31             Serial.println(String("Unknown command '") + command + "'");
32         }
33     }
34 }
```

Aby program działał sprawniej, Serial Monitor konfigurujemy tak, aby korzystał tylko ze znaku nowej linii (lista rozwijana na dole okna narzędzia).

### Zadanie 2: Sterowanie pracą Arduino za pomocą komend wysyłanych przez łącze szeregowe z PC

Napisz program, który będzie reagował na komendy wysyłane z narzędzia Serial Monitor do zestawu Arduino. Program powinien zapalać wszystkie diody LED zadaną każdej osobno jasnością w reakcji na komendę. Postać komendy ustal samodzielnie.

Jest to zadanie przykładowe. Prowadzący może zmodyfikować lub zmienić treść zadania.

## 2 Przetwornik analogowo-cyfrowy

### 2.1 Czym jest przetwornik analogowo-cyfrowy (ADC)

Przetwornik analogowo-cyfrowy ADC (Analog to Digital Converter) jest interfejsem pomiędzy światem analogowym i cyfrowym. Przetwornik umożliwia przetworzenie sygnału analogowego doprowadzonego do wejścia mikrokontrolera na liczbę, która odpowiada napięciu wejściowemu sygnału. Dzięki temu możemy podłączyć do mikrokontrolera różnego rodzaju źródła i sensory analogowe (takie, które zamieniają mierzoną wielkość fizyczną na napięcie). Możliwe jest wtedy mierzenie napięć, prądów, temperatury, ciśnienia, odległości, itp.

Podstawowe parametry ADC:

- Rozdzielcość (Resolution) - liczba bitów.
- Precyzja (Accuracy) - monotoniczność, liniowość, stabilność stałoprądowa.
- Napięcie odniesienia (Reference voltage -  $V_{REF}$ ) - wewnętrzne lub zewnętrzne.
- Skalowanie wejścia (Input scaling) - jednobieguność lub dwubieguność, zakres napięcia.
- Prędkość (Speed) - czas konwersji i opóźnienie.

Na zajęciach laboratoryjnych najistotniejsze będą dla nas rozdzielcość i napięcie odniesienia.

Rozdzielcość rozumiana jest jako liczba bitów służąca do zapisu wartości mierzonej i dla Arduino UNO wynosi 10 bitów. Zatem, możemy zmierzyć  $2^{10} = 1024$  poziomy, czyli będą to liczby w zakresie od 0 do 1023.

Napięcie odniesienia (referencyjne), które jest domyślnie ustawione to 5V (jest to napięcie zasilania mikrokontrolera ATmega328P na płytce Arduino). Względem tego napięcia dokonywany jest pomiar. Napięcie mierzone nie powinno przekraczać referencyjnego. Jeśli przekroczy, to zawsze odczytamy wartość maksymalną. Należy też mieć na uwadze, że napięcie referencyjne wewnętrzne mikrokontrolera nie jest zbyt dokładne, a jeśli jest nim napięcie zasilania mikrokontrolera, to musi być dokładnie stabilizowane i filtrowane, i też nie jest uważane za dokładne.

Możemy zmienić napięcie referencyjne w Arduino UNO. Możliwe jest skonfigurowanie napięcia referencyjnego wewnętrznego (1,1V) lub zewnętrznego z zakresu do 5V (nie może przekroczyć napięcia zasilania), które podłączamy do pinu oznaczonego na schemacie pinów (wyprowadzeń) Arduino jako AREF. Służy do tego funkcja `analogReference()`. Zapoznaj się z [jej dokumentacją](#).

**Uwaga:** Mierzone napięcie analogowe nie może przekroczyć napięcia zasilania mikrokontrolera (w Arduino UNO to 5V), ani nie może być napięciem ujemnym względem masy płytki Arduino (pin GND), ponieważ dojdzie do uszkodzenia mikrokontrolera. Dlatego nie próbuj mierzyć bezpośrednio napięcia, na przykład, baterii 9V, akumulatorów samochodowych, których napięcie w zależności od naładowania i trybu pracy może osiągać prawie 15V, zasilaczy urządzeń elektronicznych, ponieważ często mają podaną wartość średnią napięcia pod obciążeniem, a tężnienia napięcia potrafią mieć znacznie większe wartości, szczególnie w zasilaczach gorszej jakości, itp. Natomiast, dla napięć do 5V, zwracaj uwagę na bieguność, aby zawsze mierzyć napięcie dodatnie.

Warto też mieć świadomość, jaka jest zmiana wartości wejściowej przy jednostkowej zmianie odczytu ADC. Uwaga, to nie to samo co dokładność pomiaru! (więcej na temat dokładności na wykładzie). Zmiana, o której mowa, jest wyznaczana przez wartość  $1LSB$  - Least Significant Bit, czyli najmniej znaczącego bitu.  $1LSB$  możemy zdefiniować jako:

$$1LSB = \frac{V_{REF}}{2^n}$$

gdzie  $n$  to rozdzielcość ADC w bitach, a  $V_{REF}$  to użyte napięcie referencyjne.

W przypadku Arduino UNO, cały 10 bitowy zakres pomiarowy ( $n = 10$ ) przekłada się na zakres napięciowy od 0V do  $V_{REF} = 5V$ . Zatem, w Arduino UNO wartość  $1LSB$  to

$$1LSB = \frac{V_{REF}}{2^n} = \frac{5V}{2^{10}} = \frac{5V}{1024} = 0.0048828125V$$

Zatem, aby obliczyć wartość mierzonego napięcia  $V_{IN}$  należy skorzystać z zależności:

$$V_{IN} = \frac{V_{REF}}{2^n} * N_{ADC}$$

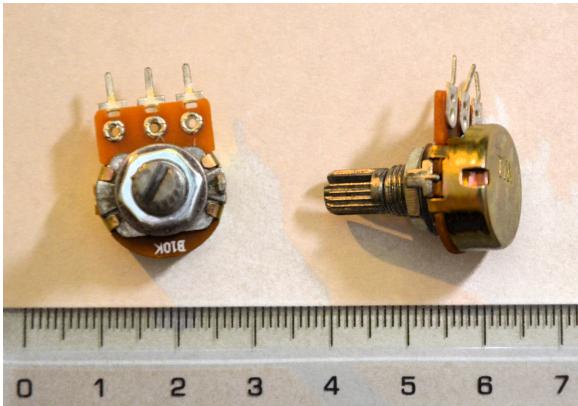
gdzie  $n = 10$ , natomiast  $N_{ADC}$ , to wartość odczytana przez przetwornik analogowo-cyfrowy.

Przykładowo, jeśli przetwornik odczytał wartość  $N_{ADC} = 500$ , to napięcie wejściowe jest równe:

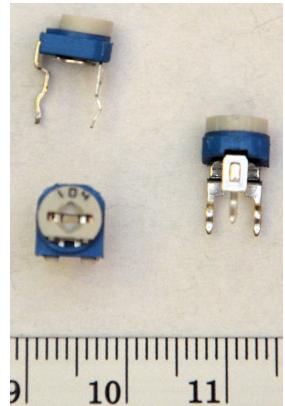
$$V_{IN} = \frac{V_{REF}}{2^n} * N_{ADC} = \frac{5V}{1024} * 500 \approx 2,44V$$

### 2.2 Potencjometr - Zasada działania potencjometru

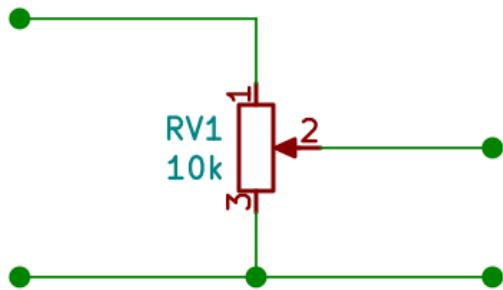
Potencjometr (fotografie 1 i 2) jest trójwyprowadzeniowym rezystorem (schemat 1).



Fotografia 1: Potencjometry.

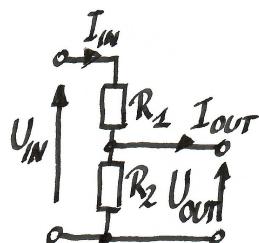


Fotografia 2: Potencjometry montażowe (trymery).



Schemat 1: Symbol (i schemat) elektryczny potencjometru.

Dwa wyprowadzenia funkcjonują jak w zwykłym rezystorze (numery 1 i 3 na schemacie 1). Pomiędzy nimi znajduje się warstwa lub drut o zadanej rezystancji. Trzecie wyprowadzenie (numer 2 na schemacie 1) przyłączone jest do elektrody, która ślizga się wzdłuż warstwy lub drutu rezystancyjnego pomiędzy pierwszymi dwoma wyprowadzeniami. Położenie trzeciej elektrody reguluje się przez obrót osi lub przesunięcie suwaka. Potencjometr w swej istocie jest dzielnikiem napięcia (rysunek 4).



Rysunek 4: Dzielnik napięcia.

Zasada działania potencjometru: Zgodnie z prawem Ohma:

$$I = \frac{U}{R}$$

Jeśli założymy, że  $I_{OUT} = 0$  i  $R = R_1 + R_2$ , to:

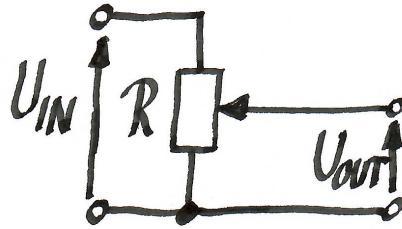
$$\frac{U_{IN}}{R} = \frac{U_{IN}}{R_1 + R_2} = I_{IN} = \frac{U_{OUT}}{R_2}$$

ponieważ przez oba rezystory płynie ten sam prąd. Zatem:

$$U_{OUT} = \frac{R_2}{R_1 + R_2} U_{IN} = \frac{R_2}{R} U_{IN}$$

Wniosek: Przy  $I_{OUT} = 0$  lub pomijalnie małym  $I_{OUT} \ll I_{IN}$ , napięcie wyjściowe dzielnika jest proporcjonalne do  $R_2$ .

W przypadku potencjometru rezystancja  $R_2$  jest proporcjonalna do położenia pokrętła lub suwaka (ślizgacza) potencjometru (rysunek 5).



Rysunek 5: Działanie potencjometru jako dzielnika napięcia.

Oznaczmy przez  $\alpha$  położenie suwaka potencjometru, przy czym  $0 \leq \alpha \leq 1$ , gdzie  $\alpha = 0$  oznacza położenie pokrętła dla  $R_2 = 0$ , natomiast  $\alpha = 1$  oznacza położenie pokrętła dla  $R_2 = R$ . Zakładamy liniową proporcjonalność  $R_2 = \alpha R$ . Uwaga: Istnieją potencjometry, dla których ta zależność nie jest liniowa. Podstawiając to do wzoru na  $U_{OUT}$  otrzymujemy:

$$U_{OUT} = \frac{R_2}{R} U_{IN} = \frac{\alpha R}{R} U_{IN} = \alpha U_{IN}$$

Przypomnienie: Wszystko to dla  $I_{OUT} = 0$  lub pomijalnie małego.

W praktyce, ślizgacz potencjometru podłączony jest do układu, który przetwarza napięcie. W zastosowaniach, które rozpatrujemy, możemy przyjąć, że mierzy napięcie przy pomocy przetwornika analogowo cyfrowego. Wejście tego układu musi mieć dużą rezystancję, aby nie obciążał potencjometru, to znaczy, aby  $I_{OUT}$  pozostał zaniedbywalnie mały w stosunku do  $I_{IN}$ , to znaczy:  $I_{OUT} \ll I_{IN}$ .

Potencjometr to nie tylko pomoc na naszym laboratorium. Jest on wykorzystywany w praktyce.

Potencjometr może służyć do ustawiania parametrów pracy programu w urządzeniu. Odczytujemy za pomocą ADC wartość napięcia ustawioną na potencjometrze i jej postać liczbową jest parametrem działania programu. Potencjometr w niezwykle łatwy umożliwia ustawianie tych parametrów, ponieważ od użytkownika wymaga się nastawienia odpowiedniej wartości za pomocą pokrętła lub śrubokrętem.

Popatrzmy na przykład. Typowy włącznik oświetlenia z czujnikiem ruchu wyposażony jest zazwyczaj w trzy potencjometry, które wyznaczają: czułość na ruch, okres czasu świecenia po zaniku ruchu, oraz poziom oświetlenia, od którego powinien zadziałać (ma działać gdy jest ciemno, a nie jasno - potencjometr pozwala wyznaczyć tą granicę w konkretnym otoczeniu włącznika oświetlenia i dostosować do preferencji użytkownika).

### 2.3 Odczytu wartości napięcia przez ADC

Przetwornik analogowo-cyfrowy (ADC) jest to stosunkowo skomplikowany i kosztowny układ. Zazwyczaj nie ma ich wiele w mikrokontrolerach i obsługują wiele pinów skonfigurowanych jako analogowe wejścia. Arduino UNO, a w zasadzie wykorzystywany mikrokontroler ATmega328P, ma jeden przetwornik analogowo-cyfrowy, który obsługuje sześć wejść. Na schemacie pinów (wyprowadzeń) Arduino UNO, i na samej płytce Arduino, zaznaczonych jest 6 pinów, które mogą pełnić funkcję analogowego wejścia i noszą oznaczenia od A0 do A5.

Konfiguracja ADC i odczyt nie jest banalna. Niemniej jednak, jeśli korzystamy z języka Arduino (bibliotek dostarczonych z Arduino IDE), to jest zadanie wręcz trywialne. Wszystkie niuanse konfiguracji i odczytu zostały ukryte za jedną funkcją. Wraz z Arduino IDE dostarczony został przykład dostępny przez Menu: Files > Examples > 03. Analog > AnalogInput. Program ten bez żadnych zmian można uruchomić na zestawie laboratoryjnym. Jest on przytoczony tutaj jako kod 4. Częstotliwość migania wbudowanej diody LED, zależy od nastawy potencjometru.

Kod 4: Odczyt wartości napięcia na wejściu za pomocą ADC - Przykład z Arduino IDE AnalogInput.ino.

```

1 int sensorPin = A0; // select the input pin for the potentiometer
2 int ledPin = 13; // select the pin for the LED
3 int sensorValue = 0; // variable to store the value coming from the sensor
4
5 void setup()
6 {
7     // declare the ledPin as an OUTPUT:
8     pinMode(ledPin, OUTPUT);
9 }
10
11 void loop()
12 {
13     // read the value from the sensor:
14     sensorValue = analogRead(sensorPin);
15     // turn the ledPin on
16     digitalWrite(ledPin, HIGH);
17     // stop the program for <sensorValue> milliseconds:
18     delay(sensorValue);
19     // turn the ledPin off:
20     digitalWrite(ledPin, LOW);
21     // stop the program for for <sensorValue> milliseconds:

```

```
22|     delay(sensorValue);  
23| }
```

Na podstawie kodu 4 widać, że nie jest wymagana żadna wstępna konfiguracja, a odczyt następuje poprzez proste wywołanie funkcji `analogRead()`. Zapoznaj się z jej dokumentacją ([link](#)). Przeczytaj też dokument [Analog Input Pins \(link\)](#).

## 2.4 Przeskalowywanie odczytu ADC

W wielu praktycznych zastosowaniach trzeba przeskalać (przeliczać) wartości z ADC do innych wartości. Można napisać własną funkcję, ale można użyć funkcji `map()` z języka Arduino. Zapoznaj się z [dokumentacją funkcji map\(\) \(link\)](#). Kod 5 prezentuje przykład sterowania jasnością diod czerwonej i niebieskiej w analogowej diodzie RGB. Przy zmianie położenia potencjometru dioda łagodnie zmienia kolor. Odczyt z ADC o wartościach w zakresie [0, 1023] trzeba przeskalać liniowo do [0, 255]. Można to zrobić właśnie funkcją `map()`.

Pamiętaj, że rzeczywisty potencjometr nie jest elementem idealnym i może nie być możliwe uzyskanie czystych kolorów przy krańcach zakresu regulacyjnego, bo nie zawsze zostaną osiągnięte wartości 0 i 1023. Takie mankamenty też trzeba uwzględnić, przy pisaniu kodu. Dlatego, w linii 16 kodu następuje obcięcie wartości na końcach zakresu pomiarowego (regulacyjnego) dla potencjometru. Wartości w zakresie [0,10] są uznawane za 0, a wartości [1013, 1023] za 1003. Stąd w linii 17 skalowanie z zakresu [0, 1003]. Nie w każdym programie zachodzi konieczność takich działań, ale tu i w tym układzie jest to przydatne.

Kod 5: Odczyt wartości napięcia na wejściu za pomocą ADC z przeskalowaniem odczytu.

```
1 #define LED_RED 6  
2 #define LED_BLUE 3  
3 #define POTENTIOMETER A0  
4  
5 void setup()  
6 {  
7     pinMode(LED_RED, OUTPUT);  
8     analogWrite(LED_RED, 0);  
9     pinMode(LED_BLUE, OUTPUT);  
10    analogWrite(LED_BLUE, 0);  
11}  
12  
13 void loop()  
14 {  
15     int value = analogRead(A0);  
16     value = min(max(0, value - 10), 1003); //1003 = 1023 - 10 - 10  
17     int lightRed = map(value, 0, 1003, 0, 255);  
18     int lightBlue = 255 - lightRed;  
19     analogWrite(LED_RED, lightRed);  
20     analogWrite(LED_BLUE, lightBlue);  
21}
```

### Zadanie 3: Odczyt wartości napięcia zadanego potencjometrem

Napisz program, który dla napięcia zadanego potencjometrem, na wyświetlaczu LCD wyświetli wartość, którą odczytał ADC i rzeczywistą wartość napięcia zadanego potencjometrem. Sprawdź, czy wartości ustawiane potencjometrem osiągają skrajne wartości odczytu (0 i 1023) i czy są one stabilne. Skonsultuj się z innymi uczestnikami laboratorium, jak to wyglądało w ich przypadku.

Proszę wykonać dokładnie to zadanie i nie zmieniać go.

### Zadanie 4: Monitorowanie napięcia wejściowego portu za pomocą narzędzia Serial Plotter z Arduino IDE

Napisz program, który odczytuje wartość z przetwornika analogowo-cyfrowego, a następnie wysyła przez łącze szeregowe do komputera. Zobacz wynik jego działania przy pomocy narzędzi Serial Monitor i Serial Plotter z Arduino IDE.

Jest to zadanie przykładowe. Prowadzący może zmodyfikować lub zmienić treść zadania.