Konkurs STEM

Skrócony Opis Środowiska Arduino

Wersja z 24.03.2023

Środowisko Arduino

Arduino jest to gotowym zestawem uruchomieniowym z popularnym ośmiobitowym mikrokontrolerem AVR. Stworzony według następujących założeń:

- Nie wymaga zewnętrznego programatora – programowanie jest możliwe dzięki wbudowanemu bootloaderowi
- Współpracuje z dedykowanym kompilatorem języka C/C++
- Można dokupić do niego dużą liczbę płytek rozszerzających (np.: moduły czujników, sterowniki silników, wyświetlacze, moduły wykonawcze)

Uproszczona budowa mikrokontrolera

Mikrokontroler może spełniać swoje zadanie dzięki odpowiedniej budowie wewnętrznej. Zawiera w sobie procesor (ang. Central Processing Unit, w skrócie CPU), który jest odpowiedzialny za realizację programu. Niezbędne do jego działania są również pamięci, różniące się pojemnością, szybkością dostępu czy trwałością danych.

Innymi niezbędnymi elementami są urządzenia peryferyjne, które służą do komunikacji z otoczeniem, na przykład równoległe porty wejściowe lub wyjściowe.

Procesor (CPU)

- Jest układem cyfrowym, sekwencyjnym i synchronicznym. Rozróżnia tylko stan niski lub wysoki napięcia, co odpowiada dwóm stanom logicznym 0 i 1, lub fałsz i prawda. Jest układem sekwencyjnym, gdyż każdy następny stan zależy od aktualnego stanu wejść oraz poprzedniego stanu, co dzieje się w rytm sygnału zegarowego (układ synchroniczny).

- Zadaniem procesora jest wykonywanie rozkazów umieszczonych w pamięci programu. Program składa się z rozkazów, które kolejno są odczytywane z pamięci i przesyłane do CPU i wykonywane. W pojedynczym rozkazie zawarte są informacje o rodzaju operacji oraz o argumentach, na jakich ma ona zostać wykonana.

Pamięć Flash

- W systemie Arduino jest to pamięć programu, o stosunkowo największej pojemności, służąca przede wszystkim do przechowywania rozkazów. Po odłączeniu zasilania informacje w niej zawarte nie są kasowane.

- Oprócz kodu programu mogą się na niej znajdować również tablice stałych oraz sekcja bootloadera, umożliwiająca wgrywanie do pamięcu Flash nowego programu bez użycia zewnętrznego programatora.

- Pamięć FLASH ma ograniczoną żywotność, zwłaszcza jeśli chodzi o liczbę zapisów

Pamięć RAM

- Pamięć RAM, przechowuje dane jedynie wtedy, kiedy jest do niej podłączone napięcie zasilające. Po każdym wyłączeniu zasilana jej zawartość jest tracona. W zamian za to, dostęp do niej jest znacznie szybszy i nie ma dla niej limitu liczby zapisów. Bardzo dobrze nadaje się do przechowywania zmiennych.

- Dodatkowo może być użyta do przechowywania stosu. Zasada działania stosu jest prosta - można na niego dokładać dane, a następnie zdejmować, czyli odczytywać. Stos zwykle jest umieszczony na końcu pamięci i rośnie w stronę niższych adresów.

Płytka Arduino UNO R3

Arduino jest platformą typu Open Hardware. Oznacza to, że udostępnione publicznie są wszelkie informacje konieczne do stworzenia własnego zestawu rozwojowego działającego w tym standardzie. Z tego powodu znaleźć można wiele różnych płytek zgodnych z Arduino.

Na potrzeby konkursu wybrano najpopularniejszą płytkę - Arduino UNO R3.

Płytka Arduino UNO R3

Sercem układu jest popularny, 8 bitowy mikrokontroler firmy Atmel, AVR ATmega328 taktowany z częstotliwością 16 MHz.

Specjalne złącza, umieszczone na bokach płytki, to wyprowadzenia najważniejszych sygnałów. Możemy tam znaleźć 14 programowalnych cyfrowych wejść lub wyjść. Sześć z nich można używać jako wyjścia PWM, a kolejne 6 jako analogowe wejścia. Znajdziemy tam również sygnał resetu oraz zasilanie. Arduino może być zasilane na kilka sposobów. Najpopularniejsze metody to:

- Zasilanie przez przewód USB

- Zasilanie przez zasilacz wtyczkowy o napięciu z zakresu 7V - 12V

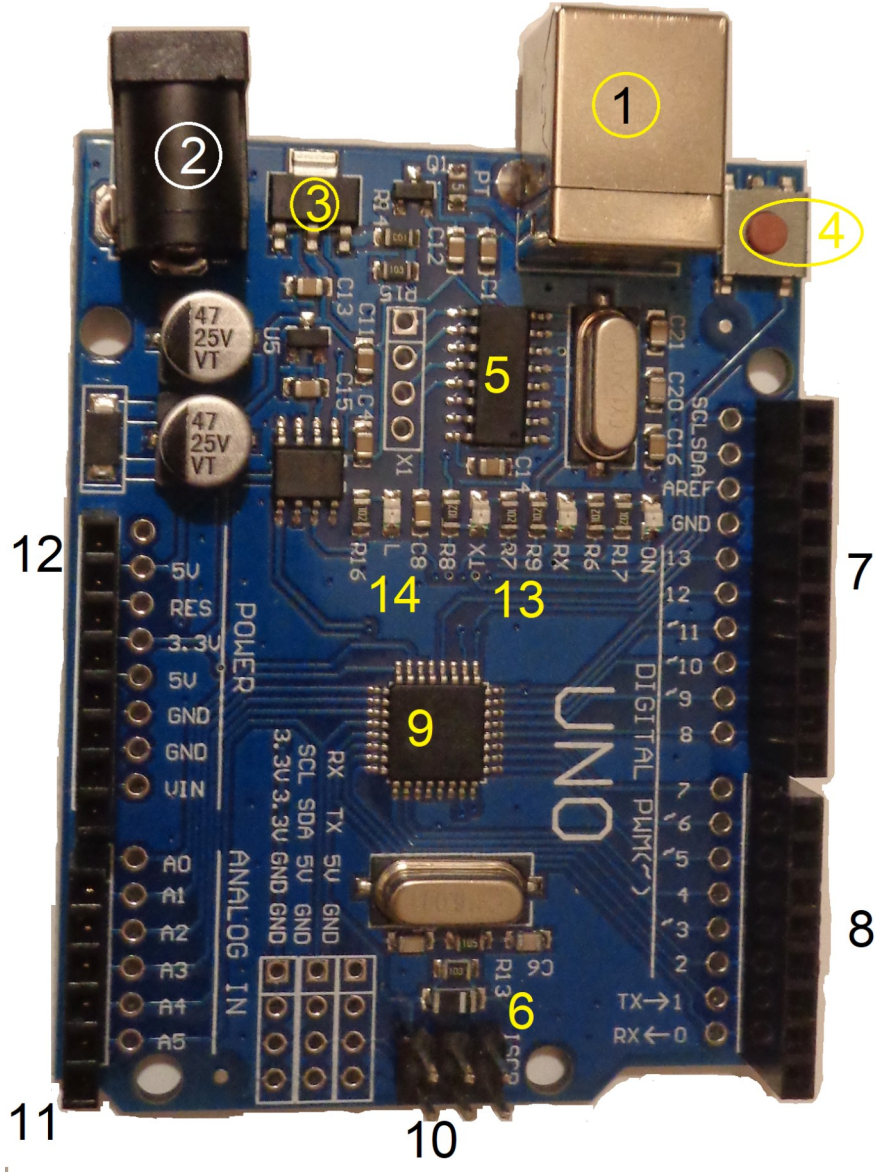
Płytka Arduino

1. Złącze USB - wykorzystywane do zasilania, programowania oraz komunikacji z komputerem
2. Złącze zasilania (optymalnie 7V - 12V)
3. Stabilizator napięcia
4. Przycisk resetu - resetuje płytkę Arduino
5. Mikrokontroler odpowiedzialny za komunikację z komputerem przez USB
- 6. Dioda LED sygnalizująca podłączenie napięcia do Arduino.

7, 8, 11 - Złącze sygnałowe

9. Główny mikrokontroler AVR ATmega328
10. Wyjście programatora dla mikrokontrolera z punktu 9.
12. Złącze zasilania
13. Diody LED sygnalizujące transmisję do/z komputera
14. Dioda LED do dyspozycji użytkownika

Źródło: https://kurs.forbot.pl/arduino

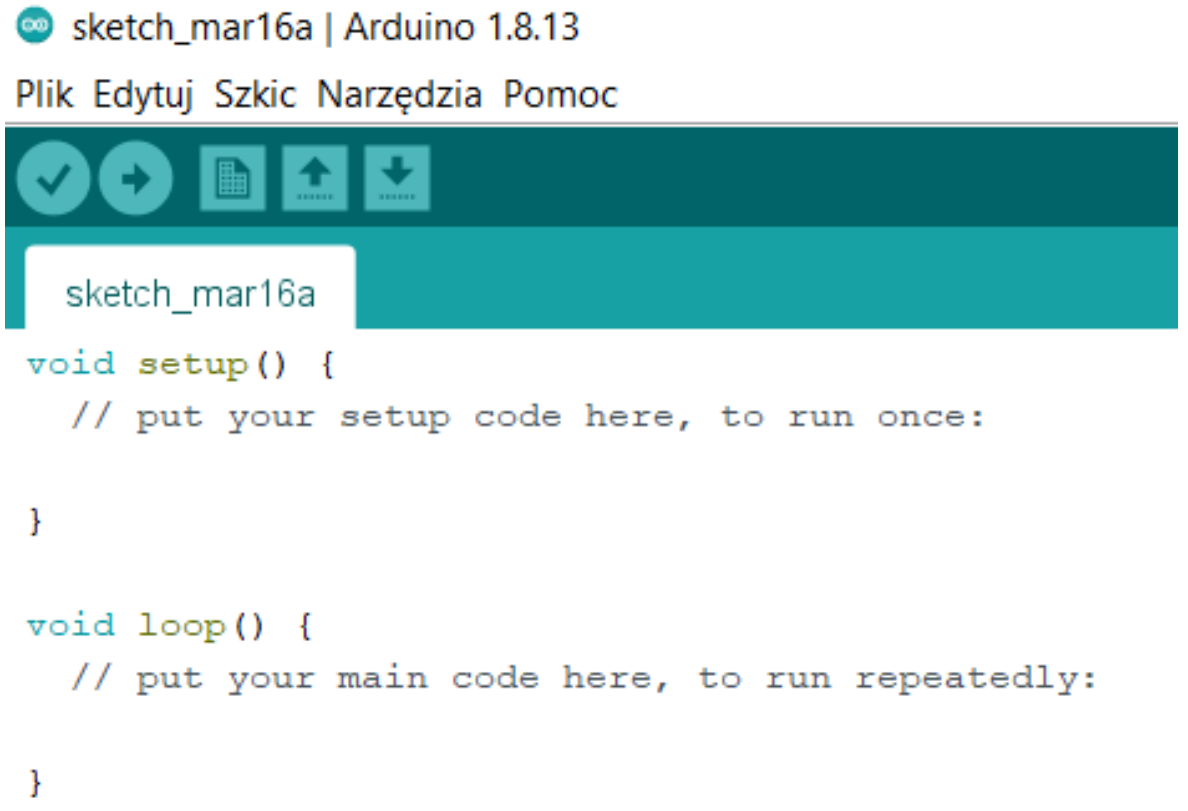


Szkielet programu

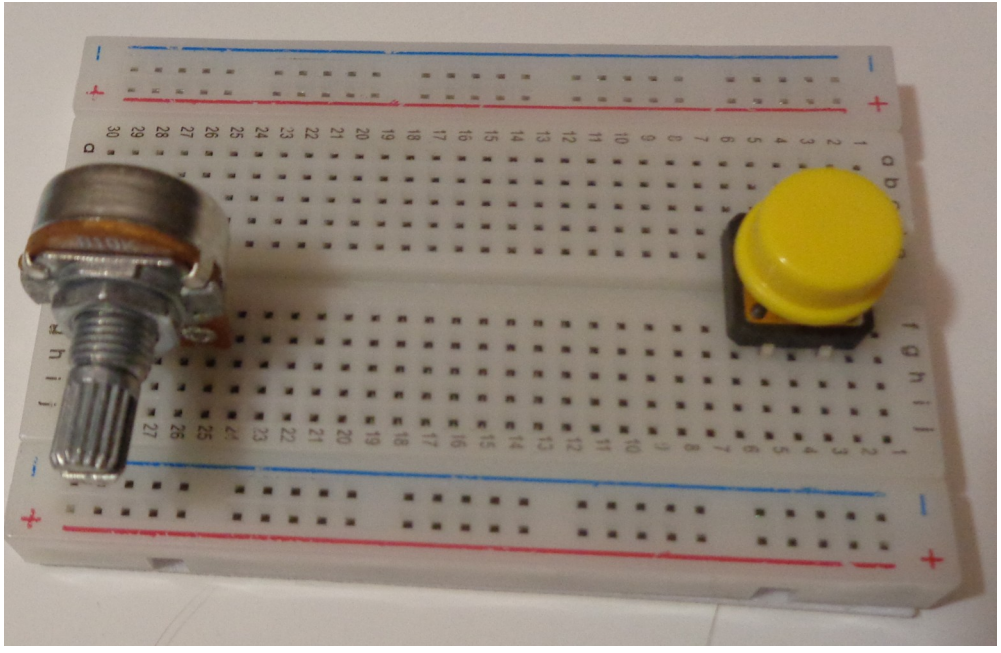
W Arduino, w każdym programie najpierw niektóre instrukcje wykonują się jednorazowo, a następnie wykonują się w nieskończonej pętli. Symbolami "//" oznaczamy komentarze. Przykładowy kod minimalnego programu pokazano poniżej.

```
void setup() {
 //Instrukcje, które wykonają się jeden raz
}

void loop() {
 //Instrukcje, które będą wykonywały się w koło (w pętli)
}
```

Wygląd środowiska po uruchomieniu



Płytka stykowa

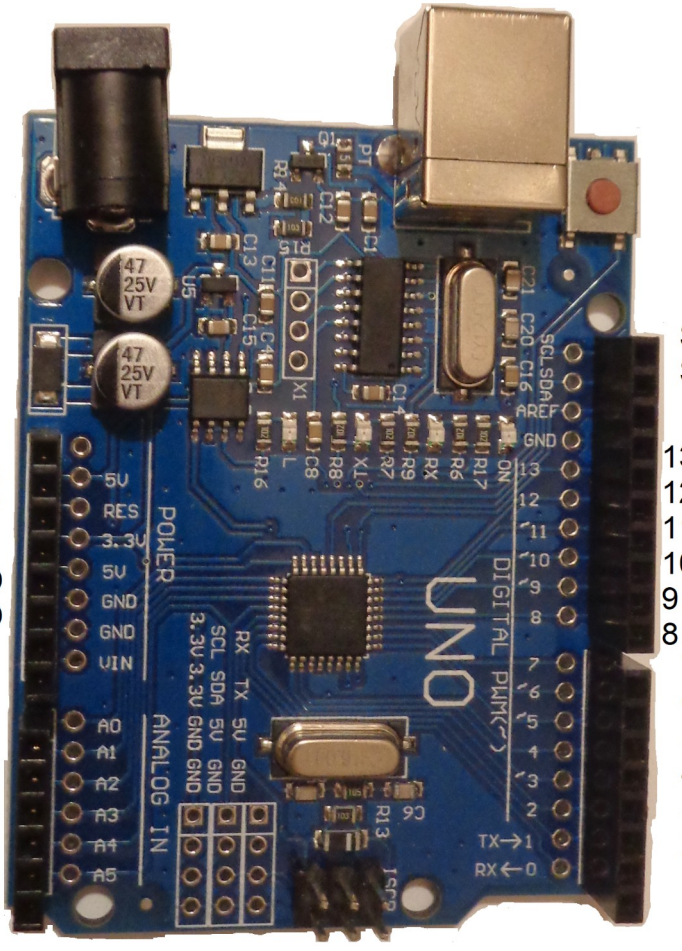


Szkielet programu

Funkcja setup() ma za zadanie wykonać jednorazowo blok instrukcji, które się w niej znajdują. Jak wskazuje jej nazwa przeznaczona jest głównie do ustawień. W praktyce funkcja setup() będzie zawierała najczęściej inicjalizacje i ustawienia. Dzięki nim dane piny (wyprowadzenia) mikrokontrolera będą ustawione jako wejścia lub wyjścia. Można także uruchomić bardziej zaawansowane peryferia oraz wykonać akcje, które mają dziać się tylko raz, po włączeniu zasilania.

W funkcji loop() umieszcza się właściwy kod aplikacji, który będzie wykonywał się cały czas (w pętli). Funkcja (procedura) loop jest pętlą nieskończoną. Znajdują się w niej instrukcje, które powinny być zapętlone (wykonywać się cały czas).

Wyprowadzenia Arduino



zasilanie

wejścia analogowe

we/wy cyfrowe

we/wy cyfrowe

Wyprowadzenia Arduino

Na wyjściach cyfrowych można ustawić 0V (logiczne 0, stan niski) lub 5V (logiczne 1, stan wysoki). Gdy zostaną skonfigurowane w roli wejść będą mogły wykrywać połączenie pinu z 0V lub 5V.

Wejścia analogowe (A0-A5) pozwalają na pomiar napięcia (w zakresie 0-5V). Praca w trybie analogowym to dodatkowa funkcja tych pinów, które można też wykorzystać jako wejścia/wyjścia cyfrowe.

Wyprowadzenia Arduino

Niektóre wyprowadzenia mogą mieć alternatywne funkcje. Oznacza to, że oprócz bycia standardowym wejściem lub wyjściem mogą one pełnić bardziej skomplikowane funkcje.

SDA, SCL - wyprowadzenia magistrali I^2C wykorzystywanej np.: do komunikacji z bardziej zaawansowanymi czujnikami, wyprowadzenia tych pinów są zdublowane (znajdują się w lewym dolnym i prawym górnym rogu płytki - to są dokładnie te same sygnały),

TX, RX – piny interfejsu UART, wykorzystywanego głównie do komunikacji z komputerem,
PWM - wyprowadzenia, na których możliwe jest generowanie sygnału prostokątnego o zmiennym wypełnieniu.

LED - dioda świecąca, wbudowana na stałe w Arduino, która połączona jest z pinem nr 13.

Wyprowadzenia Arduino

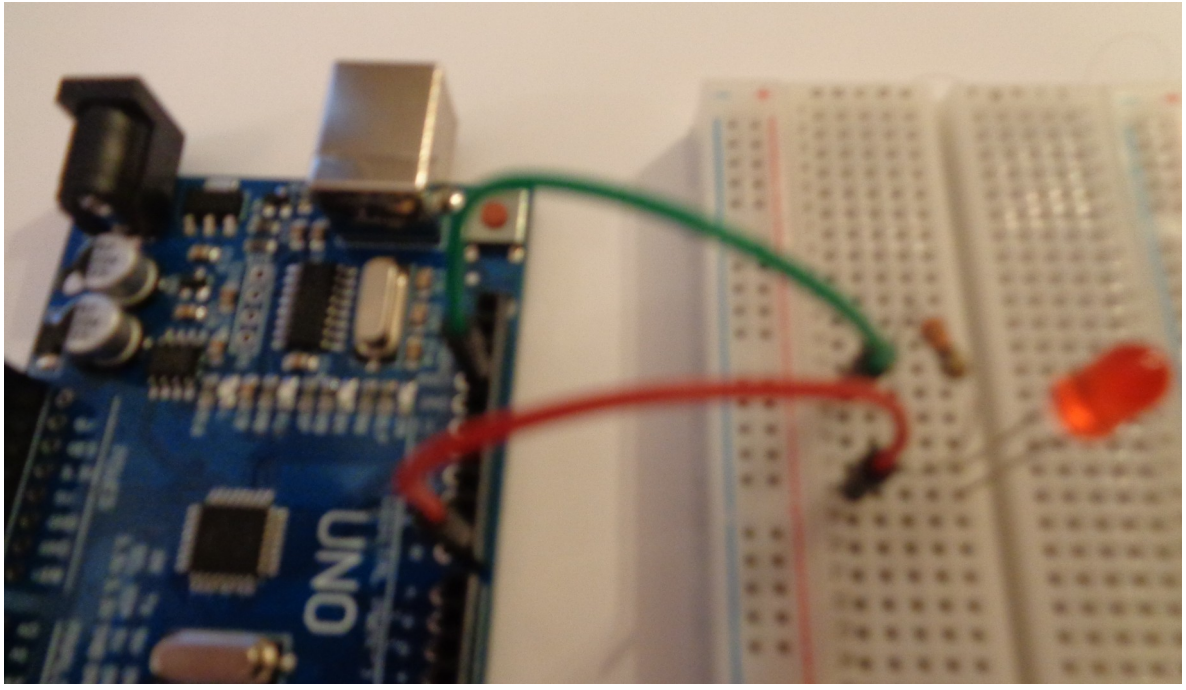
Niektóre wyprowadzenia nie są programowalne i odpowiadają głównie za zasilanie układu.

Włączanie diody LED.

Zgodnie z powyższym opisem możemy do tego wykorzystać dowolny pin I/O. Na początek wybierzmy wyjście cyfrowe nr 8. Wyjście cyfrowe - jest to wyjście, które możemy ustawić w jeden z dwóch stanów - niskim (0 V) lub wysokim (5 V).

Podłączenie LED

Układ należy podłączyć następująco: diodę łączymy szeregowo z rezystorem (330R). Następnie dłuższą nóżkę diody (anodę) łączymy z wyprowadzeniem nr 8. Drugą, przez rezystor z masą, którą znajdziemy w złączu zasilania (opisaną jako GND). Na płytce znajdują się trzy wyprowadzenia opisane jako GND - można wybrać dowolne.



Konfiguracja pinu

```
void setup() {
  pinMode(8, OUTPUT);
  digitalWrite(8, HIGH);
}

void loop() {
}
```

Funkcja pinMode(Pin, Tryb) umożliwia wybranie, czy dany pin jest wejściem, czy wyjściem. Pin może być liczbą całkowitą z zakresu od 0 do 13, zaś Tryb to:

INPUT,
OUTPUT,
INPUT_PULLUP.
Jeżeli chcemy sterować wyjściem, to zawsze używamy trybu Output

Sterowanie diodą LED

Dzięki takiej konfiguracji możemy ustawić stan logiczny 1 na wyjściu i dzięki temu włączyć diodę. Do tego celu służy funkcja digitalWrite(Pin, Stan). Stan jest stanem logicznym, który może być HIGH bądź LOW (wysoki bądź niski).

W naszym przykładzie katoda diody została już podłączona do masy, dlatego Arduino musi doprowadzić do jej anody stan wysoki stąd digitalWrite(8, HIGH);

Po jednorazowym ustawieniu pinu w stan wysoki jego wartość nie zmieni się do momentu, gdy w programie nie będzie instrukcji ustawienia innej wartości. W związku z tym, program taki jak powyższy sprawi, że dioda będzie świeciła się cały czas.

Miganie diodą LED

Do migotania diodą potrzebna jest nowa funkcja, której zadaniem będzie wprowadzanie opóźnienia – delay. Schemat połączeń jest dokładnie taki sam jak w pierwszym przypadku. Natomiast kod będzie wyglądał jak poniżej:

Arduino

```
void setup() {
  pinMode(8, OUTPUT); // Konfiguracja pinu 8 jako wyjście
}

void loop() {
  digitalWrite(8, HIGH); // Włączenie diody
  delay(1000); //Odczekanie 1000 ms = 1 sekundy
  digitalWrite(8, LOW); // Wyłączenie diody
  Delay(1000); // Odczekanie jednej sekundy
}
```

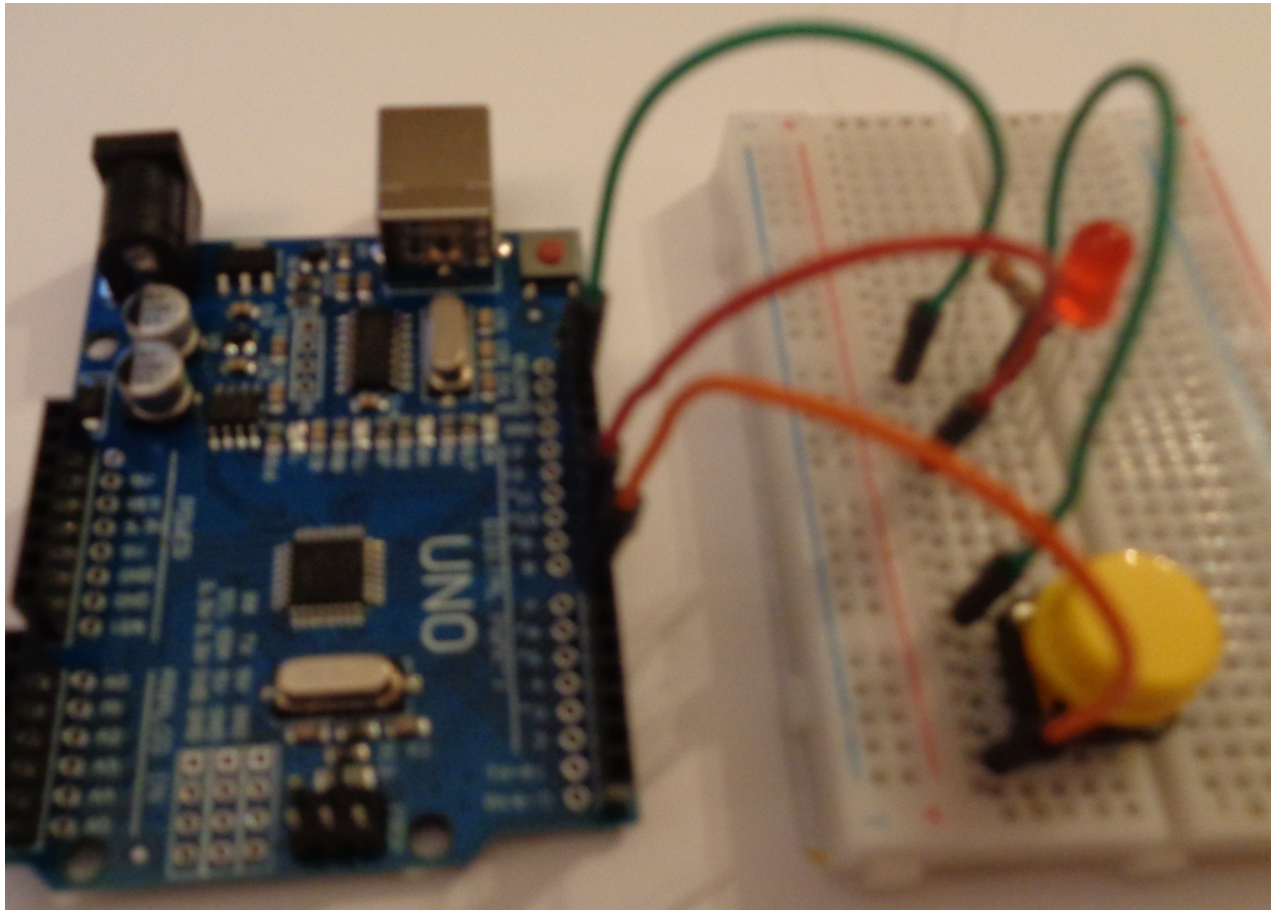
Miganie diodą LED i podłączenie przycisku

Stan wyjścia numer 8 zmienia się tu cały czas w pętli nieskończonej. W programie zostały dodane opóźnienia z pomocą funkcji delay(Czas) (dzięki temu miganie jest widoczne). Funkcja ta jako argument przyjmuje liczbę milisekund, jakie mają zostać odczekane przez procesor.

Często chcemy, aby zaprogramowany układ mógł reagować na sygnały z zewnątrz, na przykład na wciśnięcie przycisku.

Można zrobić to zgodnie z poniższym przykładem. Jedna strona przycisku została podłączona do masy (minusa), druga do wyprowadzenia nr 7.

Podłączanie przycisku



Konfiguracja pinu

W celu stwierdzenia, czy przycisk jest wciśnięty konieczne jest odczytanie stanu logicznego, który występuje na wejściu z przyciskiem.

Pin wejściowy musi być ustawiony jako INPUT_PULLUP. W tym trybie pin jest wejściem z włączonym wewnętrznym rezystorem podciągającym napięcie wejściowego pinu do dodatniego napięcia zasilania.

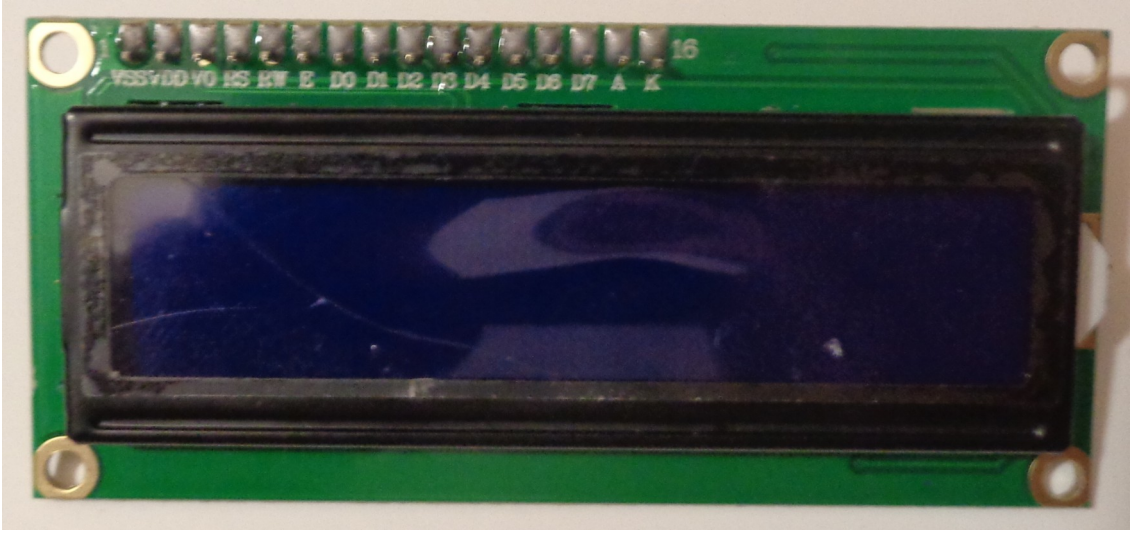
Do odczytania stanu na pinie wejściowym potrzebna jest funkcja digitalRead(pin), która zwraca wartość HIGH bądź LOW, zależnie od stanu. W naszym układzie przycisk zwiera wejście Arduino z masą (LOW).

Przykładowy program

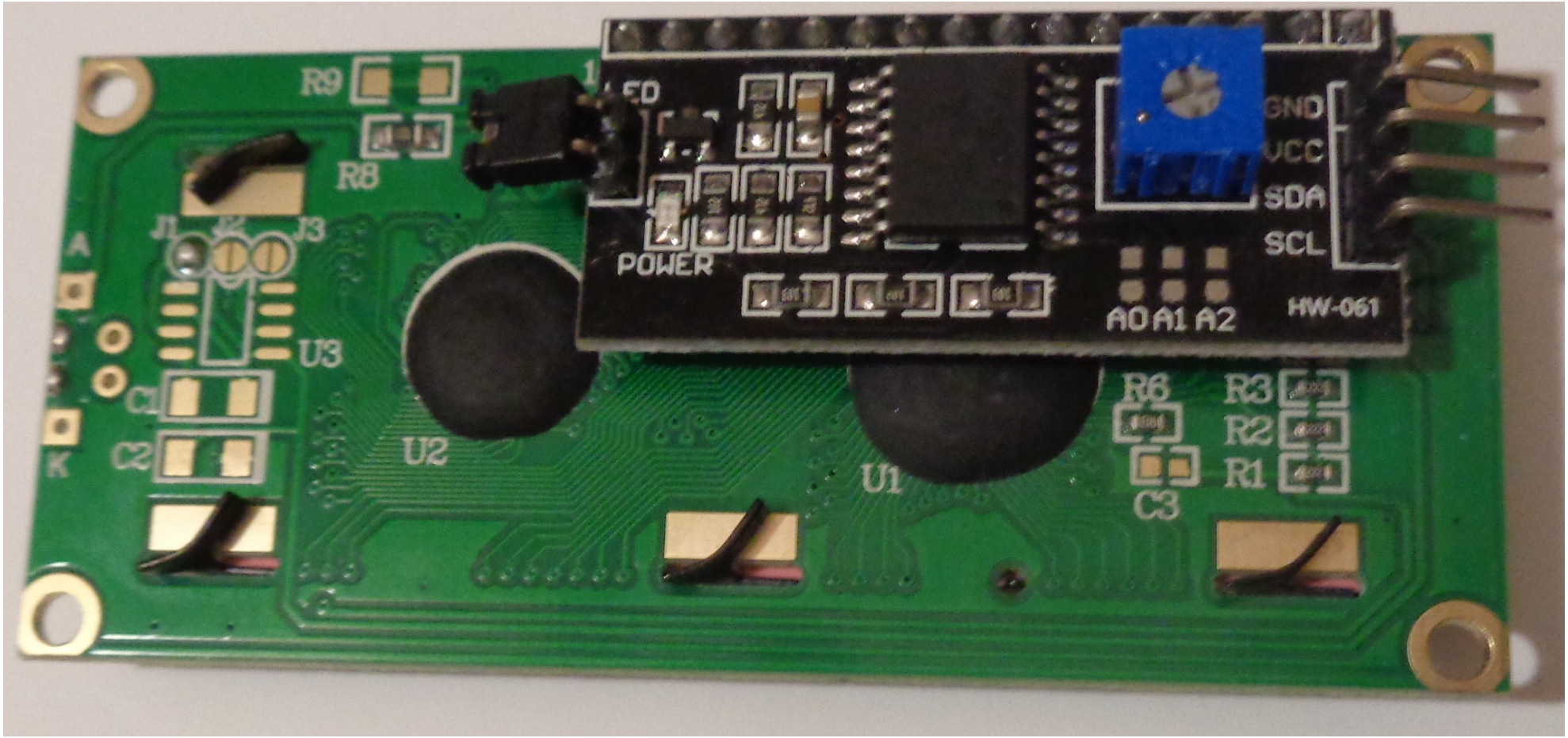
```
void setup() {
  pinMode(8, OUTPUT); // Dioda jako wyjście
  pinMode(7, INPUT_PULLUP); //Przycisk jako wejście
  digitalWrite(8, LOW); // Wyłączenie diody
}

void loop()
{
  if (digitalRead(7) == LOW) { // Jeśli przycisk wciśnięty
    digitalWrite(8, HIGH); // Włącz diodę
  } else { // Jeśli warunek nie został spełniony (przycisk nie jest wciśnięty)
    digitalWrite(8, LOW); // Wyłącz diodę
  }
}
```

Wyświetlacz LCD

Alfanumeryczny wyświetlacz LCD może wyświetlać znaki w dwóch rzędach po szesnaście kolumn. W zestawie znajduje się wlutowany po drugiej stronie konwerter I2C, dzięki któremu do obsługi wyświetlacza potrzebne są tylko dwie linie - SDA i SCL. Poprzez magistralę I2C można sterować zarówno wyświetlanym tekstem jak i podświetleniem. Na płytce znajduje się także potencjometr do ewentualnej regulacji kontrastu



Wyświetlacz LCD



Wyświetlacz LCD

W celu sterowania wyświetlaczem LCD z układem I2C należy skorzystać z funkcji zawartych w bibliotece LiquidCrystal_I2C, którą można pobrać z serwisu:
https://www.arduinolibraries.info/libraries/liquid-crystal-i2-c
Ściągnięte pliki należy rozpakować, a następnie cały folder LiquidCrystal umieścić w katalogu bibliotek Arduino (libraries).

Pobrana biblioteka ma taką samą nazwę jak standardowa, wbudowana w Arduino. Dlatego też po jej instalacji w środowisku Arduino widoczna będzie tylko ta nowa.

Podłączenie LCD

Po instalacji biblioteki, za pomocą przewodów, łączymy moduł z Arduino w następujący sposób:

Wyświetlacz	Arduino Uno
VCC	5 V
GND	GND
SDA	A4
SCL	A5

Instrukcje LCD

Bibliteki używamy w sposób podobny do tej wbudowanej w środowisko Arduino

lcd.setCursor(x,x) - ustawia kursor na określoną w nawiasie pozycję
lcd.print("xxxxx") - wyświetla na ekranie tekst xxxxx
lcd.clear(); - czyści ekran
Dodatkowo
lcd.backlight() - załącza podświetlenie
lcd.noBacklight() - wyłącza podświetlenie

Przykładowy program

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
	lcd.begin();		// inicjalizacja wyświetlacza

	lcd.backlight();	// włączenie podświetlenia
	lcd.print("Hello, world!");	// wypisanie tekstu
}
```

UART

UART jest podzespołem mikrokontrolera, który służy do przesyłania i odbioru danych szeregowych. UART w Arduino wykorzystuje dwa piny:
Tx do wysyłania danych (pin 1 w Arduino),
Rx do odbierania danych (pin 0 w Arduino).
Aby transmisja przebiegała prawidłowo, w obu układach musi być ustawiona ta sama prędkość przesyłu danych - zwana jako baud-rate. Określa ilość transmitowanych bitów na sekundę. Często spotykane wartości to: 9600 oraz 115200.
Do transmisji do komputera z wykorzystaniem układu UART należy skorzystać z konwertera USB-RS232. Konwerter taki został już wbudowany w płytkę Arduino.

UART

Zadaniem poniższego programu jest proste, regularne wysyłanie tekstu do komputera:

Arduino

```
void setup(){
  Serial.begin(9600); //Ustawienie prędkości transmisji
  Serial.println("Tekst"); //Jednorazowe wysłanie tekstu
}
void loop() {
  delay(2000);
  Serial.println("Kolejne 2 sekundy"); //Wysyłanie w pętli
}
```

Aby zaobserwować działanie programu należy z menu Arduino wybrać: Narzędzia->Monitor Portu Szeregowego. Dzięki temu otworzy się nowe okno, tak zwany terminal. Na nim można obserwować, to co jest przesyłane do i z Arduino przez port COM, czyli UART.

UART – transmisja dwukierunkowa

Zadaniem programu będzie odczyt tekstu wpisanego na klawiaturze komputera. Gdy go prześlemy, Arduino powinno wypisać komunikat "Witaj " + tekst.

```
String  odebrane = ""; //Pusty ciąg odebranych danych

void setup() {
  Serial.begin(9600); //Uruchomienie komunikacji z prędkością 9600 bitów/s
}

void loop() {
  if(Serial.available() > 0) { // Czy Arduino odebrało dane
    odebrane = Serial.readStringUntil('\n'); // Jeśli tak, to czyta je do znaku końca linii
    Serial.println("Witaj " + odebraneDane + "!"); // Wyświetla komunikat
  }
}
```

UART – transmisja dwukierunkowa

Na początku programu deklarujemy zmienną odebrane, do których kopiowany będzie ciąg odebranych znaków. Funkcja Serial.available() zwraca ilość bajtów, które zostały odebrane i czekają w buforze na obsługę przez Arduino.

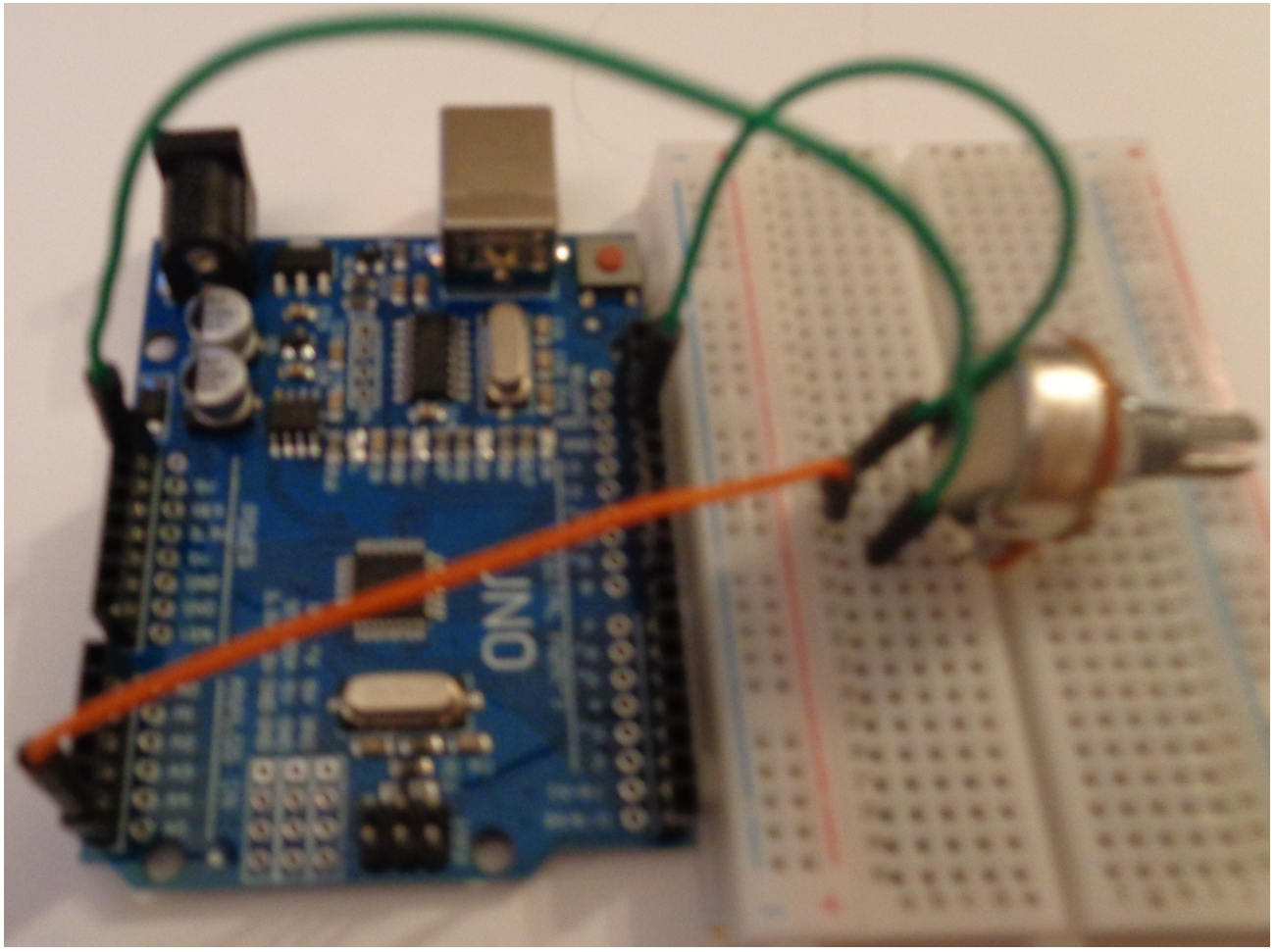
W wypadku kiedy dane będą dostępne (będzie ich więcej niż 0) - rozpoczyna się ich zapisywanie do zmiennej odebrane. Dzieje się to za pomocą funkcji Serial.readStringUntil(znak_konca), która kopiuje dane z bufora do momentu napotkania znaku końca - w tym przypadku "\n" - czyli znaku nowej linii.

Przetwornik analogowo-cyfrowy

Do przetwarzania sygnału analogowego na postać cyfrową służy 10-bitowy przetwornik analogowo-cyfrowy (Analog-Digital Converter - ADC). W Arduino UNO mamy do dyspozycji sześć wejść analogowych (A0-A5) i dzięki temu możliwy jest pomiar napięcia na maksymalnie sześciu kanałach.

Programowe wykorzystanie przetwornika ADC ogranicza się do wykorzystania jednej funkcji analogRead(kanał), gdzie za kanał podstawiamy wybrany pin (A0-A5).

Przetwornik ADC



Program z ADC

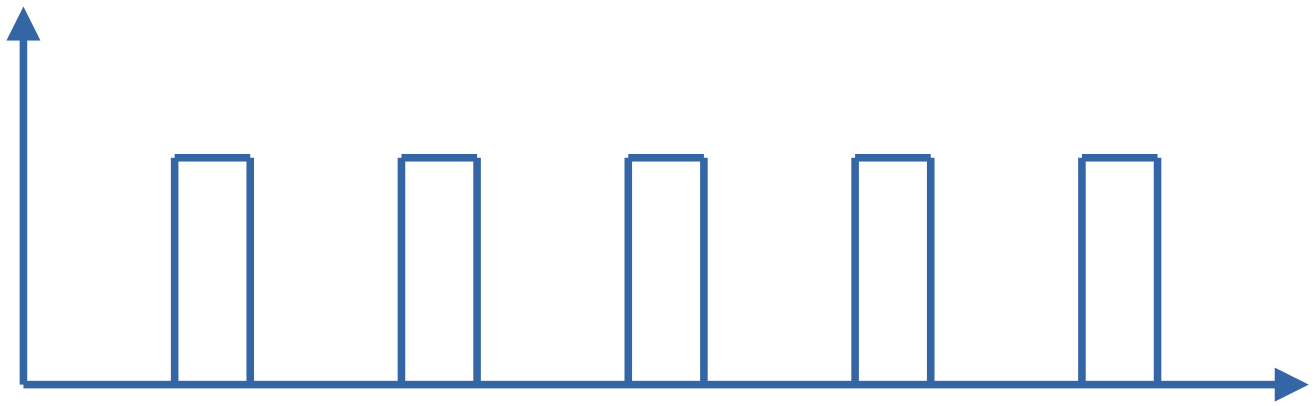
Poniższy program służy do przesyłania odczytanej wartości do komputera przy pomocy UART.

```
int napiecie = 0;

void setup() {
  Serial.begin(9600);//Uruchomienie komunikacji przez UART
}

void loop() {
  napiecie = analogRead(A5);// Odczytujemy wartość napięcia
  Serial.println(napiecie);// Wysyłamy ją do terminala
  Delay(500); //Czekamy pół sekundy
}
```

Podczas kręcenia potencjometrem, na ekranie otrzymujemy wartości od 0 do 1023, gdyż przetwornik ADC jest 10-bitowy.

PWM

PWM - modulacja szerokości impulsu.

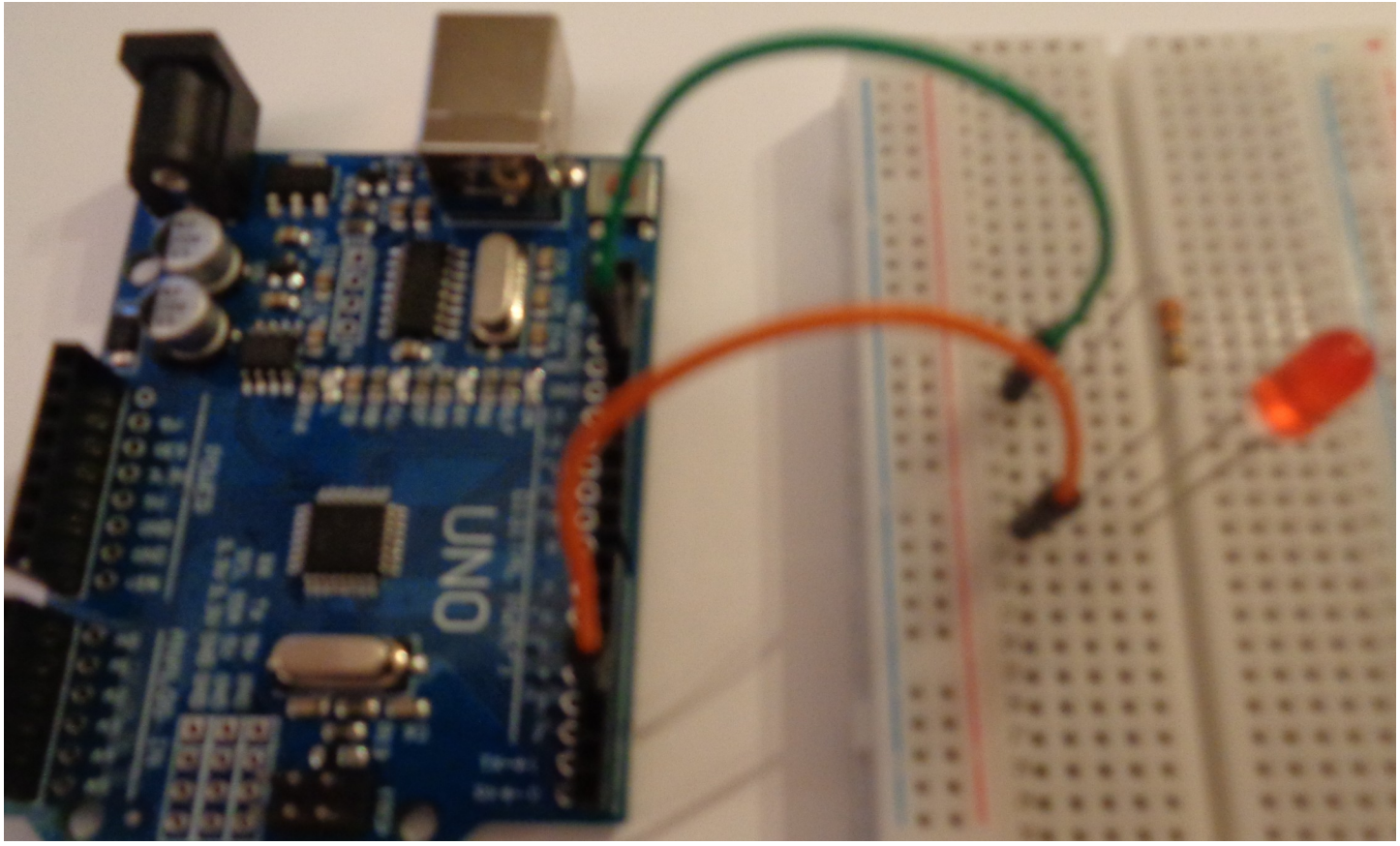


PWM

Sprzętowo generowany PWM oznacza, że wytwarzanie tego sygnału nie wpływa ujemnie na pracę programu.

Każdy kanał PWM dostępny w Arduino jest 8-bitowy. Oznacza to, że wypełnienie sygnału można określić liczbą od 0 do 255, gdzie 255 oznacza wypełnienie 100%.

PWM

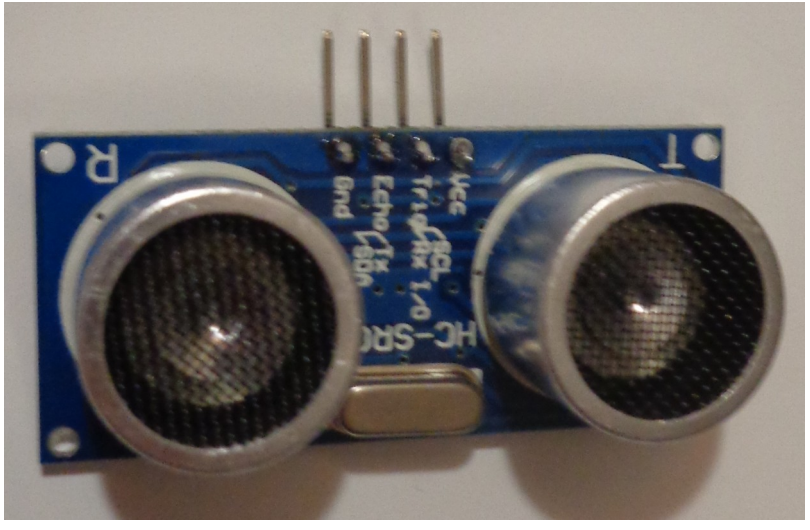


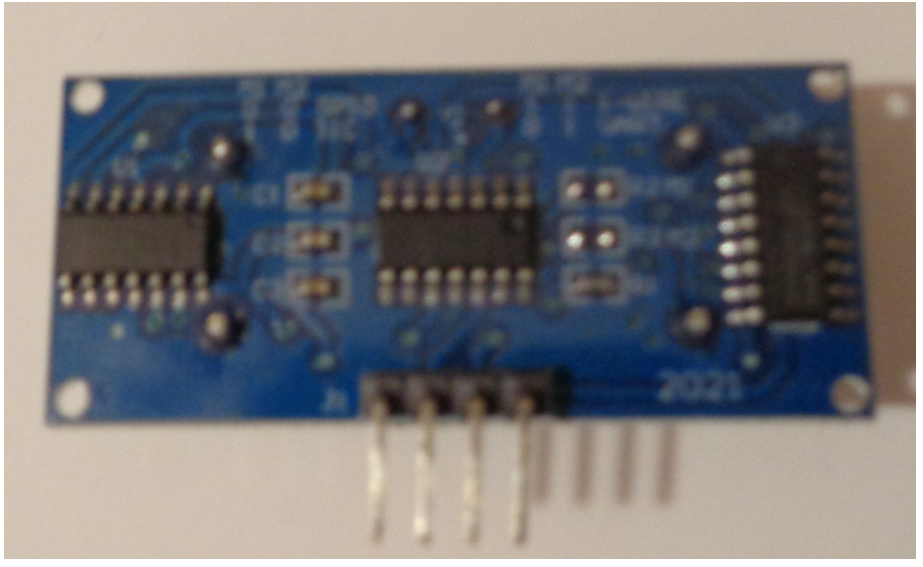
Program z PWM

```
int wypelnienie = 0;
int zmiana = 4;
void setup() {
	pinMode(3, OUTPUT);// Konfiguracja pinu 3 jako wyjścia
}
void loop() {
	analogWrite(3, wypelnienie); //Generujemy sygnał o zadanym wypełnieniu

	if (wypelnienie < 255) { // Jeśli wypełnienie mniejsze od 100%, czyli 255
		wypelnienie = wypelnienie + zmiana; // Zwiększamy wypełnienie
	} else {
		wypelnienie = 0; //Jeśli wypełnienie większe od 100%, to je zerujemy
	}
	delay(40); //Opóźnienie 40 ms
}
```

Ultradźwiękowy czujnik odległości HC-SR04

Czujnik HC-SR04 składa się z nadajnika i odbiornika ultradźwięków. Na podstawie nasłuchiwania, po jakim czasie powróci odbity sygnał można określić odległość czujnika od przeszkody.



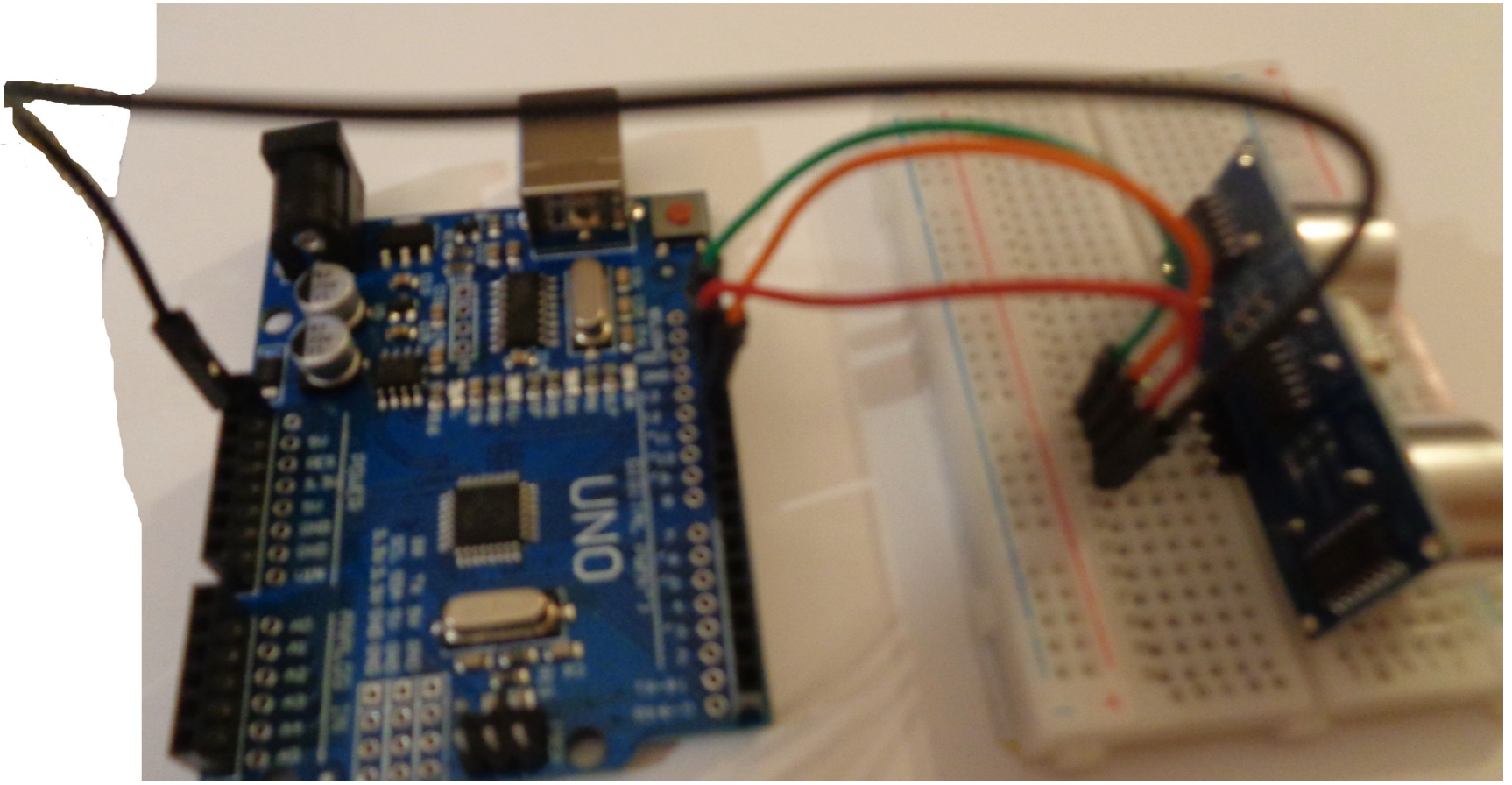


Czujnik odległości

Ma on cztery piny sygnałowe. Dwa z nich służą do zasilania układu (Vcc, GND), a drugie dwa (trigger i echo) do wykonywania pomiarów.

Trigger, to wejście wyzwalające pomiar. Gdy podamy na nie stan wysoki (przez min. 10 mikrosekund), to rozpocznie się pomiar odległości. Natomiast z wyjścia echo odczytamy zmierzoną odległość mierząc czas trwania stanu wysokiego.

Podłączenie czujnika



Pomiar odległości

Zmierzona odległość reprezentowana jest przez impuls (stan wysoki) na pinie echo. Jego długość jest proporcjonalna do odległości. Im jest on dłuższy, tym zmierzona odległość jest większa. Do pomiaru czasu trwania pinu służy funkcja pulseIn(numer pinu, stan pinu), która przyjmuje dwa argumenty. Numer pinu, który ma zostać sprawdzony oraz poziom logiczny (niski/wysoki), który ma być mierzony.
Podłączamy sygnał Trig do pinu 12, zaś sygnał Echo do pinu 11.

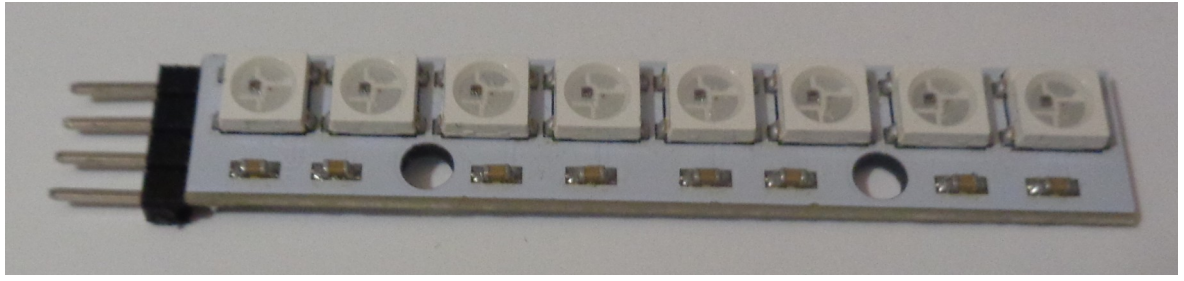
Kod programu

```
void setup() {
  Serial.begin (9600);
  pinMode(12, OUTPUT); // Pin wyjściowy, do którego podłączymy Trig
  pinMode(11, INPUT); // pin wejściowy do podłączenia sygnału Echo
}
void loop() {
  long czas, dystans;
  digitalWrite(12, LOW);
  delayMicroseconds(2);
  digitalWrite(12, HIGH);
  delayMicroseconds(10);
  digitalWrite(12, LOW);
  czas = pulseIn(11, HIGH);
  dystans = czas / 58;
  Serial.print(dystans);
  Serial.println(" cm");
  delay(400);
}
```

Listwa LED

Listwa LED RGB WS2812 5050 x 8 diod - 53mm

Listwa złożona jest z ośmiu indywidualnie adresowanych diod LED RGB ze zintegrowanym sterownikiem. Do obsługi modułu wystarczy jeden pin wyjściowy mikrokontrolera. Listwę LED można zasilać napięciem od 4 V do 7 V.



Listwa LED

Listwa LED RGB WS2812 5050 x 8 diod – 53mm

Podłączenie listwy LED
Urządzenie posiada cztery wyprowadzenia:

GND - masa modułu
4-7VDC - napięcie zasilania, 20 mA prądu na jedną diodę
DIN - cyfrowy sygnał sterujący z mikrokontrolera
DOUT - wyjście do ewentualnego podłączenia następnego odcinka listwy

Kod programu

```
#include <Adafruit_NeoPixel.h>
#include <avr/power.h>
#define PIN 0
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(8, PIN);
uint32_t color  = 0xFF0000; // kolor czerwony
void setup() {
  pixels.begin();
  pixels.setBrightness(127); // połowa jasności
}
void loop() {
  uint8_t  i;
    i = random(32);
    pixels.setPixelColor(i, color);
    pixels.show();
    delay(10);
    pixels.setPixelColor(i, 0);
}
```

Pomiar czasu

Do odczytu i pomiaru czasu służy funkcja millis(), która zwraca liczbę milisekund od startu programu.
Przykładowy kod programu zamieszczono poniżej:

```
unsigned long Czas = 0;

void setup(){
  Serial.begin(9600);
}

void loop(){
  Czas = millis(); // Odczytaj liczbe milisekund od startu
  Serial.println(Czas); // Wyslij do komputera
  delay(1000); // Odczekaj sekundę
}
```

Literatura:
https://forbot.pl/blog/kurs-arduino-podstawy-programowania-spis-tresci-kursu-id5290
https://botland.com.pl/lancuchy-i-matryce-led/
https://botland.com.pl/