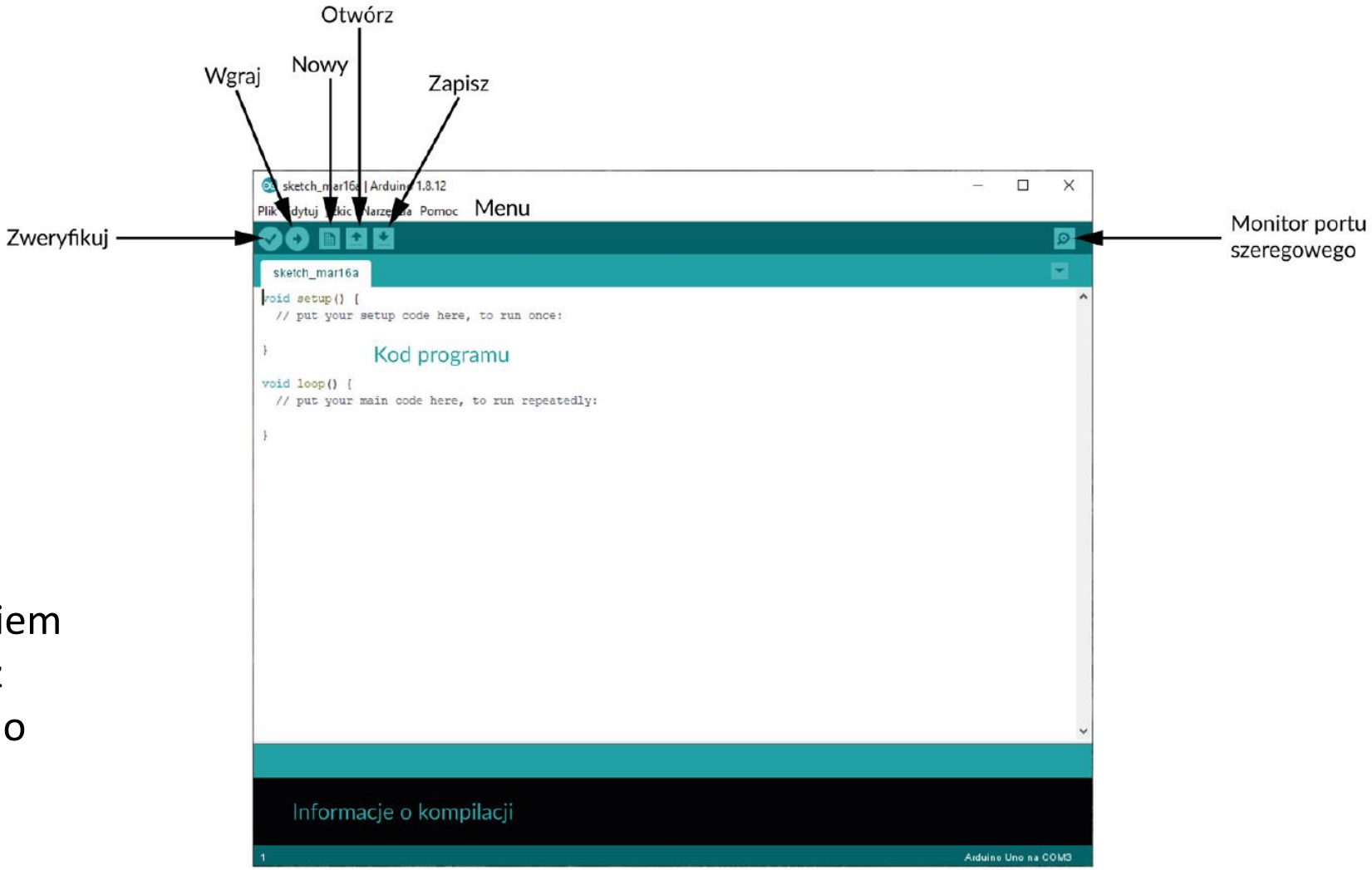
ELEMENTY POGRAMOWANIA MIKROKONTROLERÓW

Pisanie programów* realizowanych z wykorzystaniem Arduino

Istnieje Arduino IDE (Integrated Development Environment).
Jest to zintegrowane środowisko programistyczne, które upraszcza proces programowania komputerowego tj. pisania kodu i pozwala przesłać go na płytkę Arduino.

Arduino IDE jest oprogramowaniem (interfejsem) współpracującym z płytkami Arduino, niezbędnym do pisania szkiców (programów) wykonywanych później przez mikrokontroler.



w Arduino program nazywa się szkicem – nie jestem zwolennikiem tego nazewnictwa

Programy w Arduino IDE pisane są w edytorze tekstu i zapisywane w plikach z rozszerzeniem .ino

Edytor posiada funkcje wycinania/wklejania oraz wyszukiwania i zastępowania tekstu.

Obszar wiadomości zapewnia informacje zwrotne podczas zapisywania i eksportowania, a także wyświetla błędy i inne informacje.

Przyciski paska narzędzi umożliwiają weryfikację i przesyłanie programów, tworzenie, otwieranie i zapisywanie programów oraz otwieranie monitora szeregowego.

Do elementów interfejsu Arduino IDE zaliczamy:

1. Główne menu służące do obsługi programu.
2. Przycisk **Zweryfikuj**, służący do sprawdzenia poprawności kodu.
3. Przycisk **Wgraj**, służący do wysłania programu do płytki Arduino.
4. Kolejne przyciski: **Nowy**, **Otwórz**, **Zapisz**, służące odpowiednio do tworzenia nowego programu, otwierania wcześniej zapisanego programu oraz zapisywania bieżącego programu.
5. Przycisk **Monitor portu szeregowego**. Wykorzystuje się go bardzo często, a służy do otwarcia okna umożliwiającego komunikację między komputerem a płytką Arduino.

W środkowej części okna Arduino IDE znajduje się obszar, w którym wpisujemy kod programu (szkicu).
U dołu wyświetlane są informacje o przebiegu kompilacji, ewentualnych błędach lub komunikat o prawidłowym wysłaniu szkicu do płytki Arduino.

uwagi ogólne dotyczące pisania programów

W języku C stosujemy wolny format kodu.
Wszystko co chcemy zapisać może się znajdować w dowolnym miejscu linii, a nawet można zajmować treść kilku linii.
Związane jest to z tym, że koniec instrukcji jest określany przez średnik, który umiejscawiany na końcu.

Wewnątrz instrukcji może się znaleźć dowolna liczba tzw. białych znaków (znaki spacji, znaki tabulatora, znaki nowego wiersza). Są one ignorowane przez kompilator.

Stosowanie wcięć przy pisaniu kodu ułatwia jego zrozumienie. Wcięcia mogą być dowolne. Zalecane trzy naciśnięcia spacji.

uwagi ogólne dotyczące pisania programów cd.1

W zapisie programów stosowana konwencja typograficzna zwana **camel case**.

Nazwy zmiennych muszą być jednowyrazowe bez spacji.
Gdy nazwa jest dwuwyrazowa zaczyna się od wyrazu pierwszego pisanego z małej litery a następny wyraz pisany bez spacji zaczyna się z dużej litery.
Gdy zapis jest jednowyrazowy można zachowując konwencję wprowadzając dużą literę na kolejnej sylabie lub części wyrazu (jest to atrakcyjne dla polskich nazw, gdzie mamy na ogół dłuższe wyrazy).

Przykłady:
delayPeriod, czasOczekiwania, czeKam

uwagi ogólne dotyczące pisania programów cd.2

Funkcja jest fragmentem kodu który wykonuje określone zadanie.
Kod funkcji umieszczany jest w nawiasach klamrowych { … }
Kod umieszczany pomiędzy nawiasami klamrowymi nosi nazwę bloku kodu.

Po otwierającym nawiasie klamrowym { musi zawsze następować zamykający nawias klamrowy }. Oznacza to że musi mieć miejsce równowaga nawiasów klamrowych.

Arduino IDE zawiera wygodną funkcję sprawdzania równowagi nawiasów. Wystarczy wybrać nawias, a nawet kliknąć punkt wstawiania bezpośrednio za nawiasem, a jego logiczny odpowiednik zostanie podświetlony.

Pozostałe fragmenty kodu są poleceniami inaczej dyrektywami.
W nawiasach okrągłych poleceń występują argumenty, przy wielu argumentach oddzielane są one przecinkami.
Średnik **;** umieszczony na końcu sygnalizuje koniec polecenia.

Uwaga: istnieje dość powszechna niejednoznaczność nomenklaturowa - niektórzy zarówno funkcje jak i polecenia nazywają funkcjami

Dla przypomnienia:
nawiasy okrągłe (...)
nawiasy kwadratowe [...]
nawiasy klamrowe {...}

Pisanie programu (szkicu)

Każdy program musi się składać z dwóch części – jednej zawierającej funkcję **setup** i drugiej zawierającej funkcję **loop**. Słowo kluczowe **void** - używane w deklaracjach funkcji oznacza, że funkcja nie powinna zwracać żadnych informacji.

```
void setup() {
  // umieść tutaj swój kod konfiguracji, aby uruchomił się raz
}
void loop() {
  // umieść tutaj swój główny kod, aby uruchamiał się wielokrotnie
}
```

Symbol podwójnego ukośnika // oznacza ze za tym jest komentarz jednowierszowy i program pomija go. Komentarz wielowierszowy zaczyna się od symbolami ukośnika i gwiazdki /* a kończy się symbolami gwiazdki i ukośnika */ , czyli

…… // komentarz jednowierszowy

…….. /* komentarz
wielowierszowy */

Pomiędzy nawiasami okrągłym otwierającym (oraz okrągłym zamykającym) umieszczane są argumenty funkcji. W przypadku funkcji setup i loop nie ma argumentów, ale nawiasy muszą pozostać.

Pomiędzy nawiasami klamrowym otwierającym { oraz klamrowym zamykającym } umieszczany jest program, który ma być wykonany.

polecenie **pinMode**

konfiguruje określony pin,
aby funkcjonował jako wejście lub wyjście

składnia
pinMode (pin, tryb)

parametry
pin – numer pinu
tryb – ustawienie czy dany pin jest:
INPUT –wejściem
OUTPUT – wyjściem
INPUT_PULLUP – wejście z podłączonym wewnętrznym rezystorem podciągającym

przykład

```
void setup() {
  pinMode(8, OUTPUT);//konfiguracja pinu 8 jako wyjścia
}
 void loop() {
}
```

polecenie **digitalWrite**

ustawia wartość LOW lub HIGH na zadanym pinie

składnia
digitalWrite (pin, stan)

parametry
pin – numer pinu
stan – wybranie jaki stan logiczny ma być na wyjściu danego pinu:
HIGH – ustawienie stanu wysokiego (napięcie 5V)
LOW – ustawienie stanu niskiego (napięcie 0 V)

przykład

```
void setup() {
  pinMode(8, OUTPUT);//ustawienie pinu 8 jako wyjścia
  digitalWrite(8, HIGH);//ustawienie na pinie 8 wysokiego stanu logicznego
}
 void loop() {
}
```

polecenie **digitalRead**

odczytuje wartość z określonego pinu cyfrowego

składnia
digitalRead (pin)

parametry
pin – numer pinu

Zwraca wartości ustawioną na wybranym pinie: HIGH lub LOW

przykład

```
void setup()
{
  pinMode(13, OUTPUT);      // ustawia pin 13 jako wyjście
  pinMode(7, INPUT);      // ustawia pin 7 jako wejście
}

void loop()
{
  int warTosc = digitalRead(7);   // odczyt pinu 7, odczyt zapamiętany w parametrze całkowitoliczbowym warTosc
  digitalWrite(13, warTosc);    // ustawienie pinu 13 w zależności od stanu pinu 7
}
```

polecenie **delay**

wstrzymuje program na czas (w milisekundach) określony jako parametr

składnia
delay (czas)

parametry
czas - oznacza liczbę milisekund jaka trzeba odczekać, aby kontynuować realizację programu

```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000); // oczekiwanie 1000 milisekund - dioda swieci
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000); // oczekiwanie 1000 milisekund – dioda nie swieci
}
```

funkcje **if** oraz **else**

Uwaga:
znak = jest instrukcją przypisania np. x=10 znaczy, że zmiennej x przypisałem wartość 10
znak == jest operatorem porównania i np. x==10 oznacza że trzeba zbadać czy x ma wartość 10

składnia
if (warunek)

parametry
warunek – podanie warunku który musi być spełniony aby dalsza część programu była wykonywana

składnia
else

program załączony po else zostanie wykonany, jeśli warunek w instrukcji if będzie fałszem

przykład ogólny

```
…
void loop()  {
if (warunek) {
tu jest program wykonywany kiedy warunek jest spełniony
}
else {
tu jest kod programu który jest wykonywany gdy warunek nie jest spełniony
}
}
```

wykonanie instrukcji w zależności od warunku zewnętrznego

if

Linie kodu programu są wykonywane po kolei.
Istnieje jednak możliwość, aby pewne operacje zostały wykonane dopiero po ingerencji z zewnątrz np. wciśnięciu przez użytkownika przycisku podłączonego do Arduino.

Przykład kodu takiego programu:

```
void setup()
{
pinMode(5, INPUT_PULLUP); // ustawienie pinu 5 jako wejścia z podłączonym rezystorem podciągającym
pinMode(9, OUTPUT);//ustawienie pinu 9 jako wyjścia
}
void loop()
{
if (digitalRead(5) == LOW)//gdy podamy na pin 5 stan L - naciśniemy przycisk
{
digitalWrite(9, HIGH);//wtedy na pinie 9 będzie H
}
}
```

cykliczne wykonywanie instrukcji

funkcja **for** oraz **while**

Uwaga w parametrze „zmiana” funkcji if
i=i+i lub i++ to jest inkrementacja
i=i-1 lub i-- to jest dekrementacja

składnia
for (inicjalizacja; warunek; zmiana)

parametry
inicjalizacja – jednorazowe ustawienie początkowe
warunek – przy każdym przejściu pętli sprawdzany jest ten warunek
zmiana – zmiana ustawienia początkowego, która jest dokonywana przy każdym przejściu przez pętlę

składnia
while (warunek)

parametry
warunek – przy każdym przejściu pętli sprawdzany jest ten warunek,
gdy zostanie spełniony wykonywany jest kod programu następujący po funkcji while

funkcja **for** oraz **while**

przykład

```
void setup() {
  pinMode(13, OUTPUT); //Konfiguracja pinu 8 jako wyjście
}
void loop() {
for (int i = 1; i <= 5; i++) { //Wykonaj 5 razy
          digitalWrite(13, HIGH); //Włączenie diody poprzez podanie stanu wysokiego
           delay(1000); //Odczekanie jednej sekundy
          digitalWrite(13, LOW); //Wyłączenie diody poprzez podanie stanu niskiego
           delay(1000); //Odczekanie jednej sekundy

  //poniżej warunek zakończenia migania diody
  while (i==5);//kiedy i osiągnie wartość 5
  digitalWrite(13, LOW);//wyłączenie diody
          }
           }
```

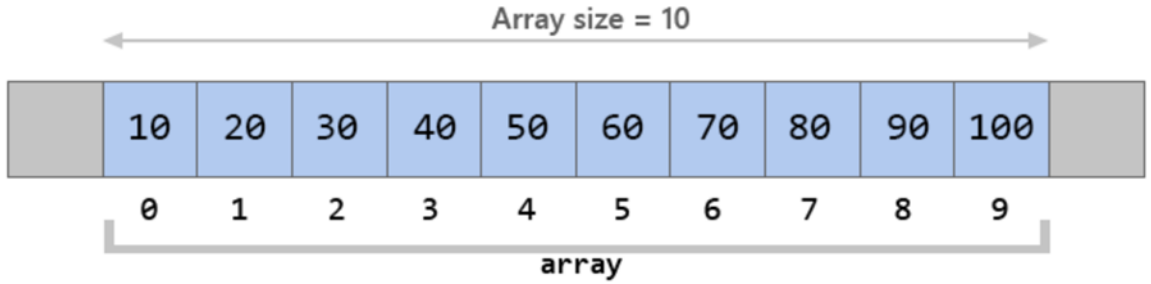
Uwagi
i++ to to samo co i=i+1 (inkrementacja)
wykorzystujemy diodę na płytce podłączoną do pinu 13

Tablica

Tablica przechowuje dane zawsze tego samego typu, posiada indeksy, których numeracja zaczyna się od zera. Aby stworzyć tablicę należy napisać jaki typ danych będzie przechowywać, nazwę jaką nadamy tablicy oraz otwierający i zamykający nawias kwadratowy.

Przykład

int wiekUzytkownikow [] = { 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 , 100};
int – deklaracja, że w tablicy będą zmienne typu integer (typ całkowitoliczbowy)
wiekUzytkownikow – nazwa własna tablicy nadana przez programistę
nawiasy kwadratowe [] - informacja że zmienna zawiera tablicę



Aby uzyskać dostęp do poszczególnych pozycji wpisanych w tablicę należy posługiwać się indeksem - cały czas pamiętając, że pierwszy element ma numer indeksu 0

Aby wyświetlić w monitorze portu szeregowego liczbę 10, która ma indeks 0 powinniśmy napisać:
Serial.println(wiekUzytkownikow[0]);

Przykład (zawiera także wywołanie monitora portu szeregowego)

```
int wiekUzytkownikow [] = { 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 , 100};

void setup() {
          Serial.begin(9600);//wywołania monitora portu szeregowego
          for (int i = 0; i < 10; i++){
          Serial.println( wiekUzytkownikow[i] );//treść wyświetlana na monitorze,..ln wyniki jeden pod drugim
}
}
void loop() {
}
```

na monitorze portu szeregowego otrzymujemy wydruk:

10
20
30
40
50
60
70
80
90
100

Aby zmodyfikować wybrane wartości w tablicy przykładowo zmienić wartość 40 na 45, powinniśmy zadysponować następujące polecenie:

wiekUzytkownikow[3] = 45;

definiowanie stałych

Polecenia const i umożliwia utworzenie zmiennej "tylko do odczytu" czyli stałej.
Zamiennikiem dla const jest define.
Twórcy ArduinoIDE zalecają stosowanie const.
Poniżej przykłady dla stałych zdefiniowanych przez użytkownika:

```
const float pi=3.14;    // definicja stałej pi, pi zadeklarowano jako zmiennoprzecinkowe float

#define ledPin 13       // definicja pinu 13 jako ledPin
//kompilator zastąpi wszelkie wzmianki o ledPin wartością 13 w czasie kompilacji
```

Jak widać const wskazuje na typ, natomiast define przyporządkowuje nazwie odpowiednią wartość bez wskazywania typu.
Brak średnika na końcu linii z define nie jest przeoczeniem, ale właściwą składnią zapożyczoną z C/C++.

Dobrą praktyką wydaje się stosowanie define do oznaczenia słownego poszczególnych wejść/wyjść programowanego układu, natomiast dla pozostałych stałych wykorzystywanych w programie zaleca się korzystać z dyrektywy const

Funkcje własne (bez argumentów)

Wiemy, że kod umieszczamy w funkcjach setup() {} lub loop() {}.

Pierwsza z nich dotyczy ustawień, a druga jest nieskończoną pętlą główną, która wykonuje się cały czas.

Aby uprościć zapis programu w przypadkach gdy pewien fragment kodu powtarzał by się można go wyróżnić jako osobną funkcje i wywoływać ja w odpowiednim miejscu programu.

Schemat umieszczenia funkcji własnej (pod nazwą, którą wymyśliłem tj. mojaFunkcja) w programie przedstawiono poniżej.

```
void setup() {
....
}

void loop() {
mojaFunkcja(); //wywołanie funkcji własnej
...
}

void mojaFunkcja() {
//tu napisać kod funkcji własnej
}
```

Przykład

Aby migać diodą podłączoną do pinu nr 13 należało napisać taki program:

```arduino
void setup() {
  pinMode(13, OUTPUT); //Konfiguracja pinu 13 jako wyjście
}

void loop() {
  digitalWrite(13, HIGH); //Włączenie diody
  delay(1000); //Odczekanie 1 sekundy
  digitalWrite(13, LOW); //Wyłączenie diody
  delay(1000); //Odczekanie 1 sekundy
}
```

Ten sam program przy wykorzystaniu funkcji własnej o nazwie zamigajLED (bez argumentów) przedstawiono poniżej

```c
void setup() {
  pinMode(13, OUTPUT); //Konfiguracja pinu 13 jako wyjście
}

void loop() {
  zamigajLED();
}

void zamigajLED() {
  digitalWrite(13, HIGH); //Włączenie diody
  delay(500); //Odczekanie 0,5 sekundy
  digitalWrite(13, LOW); //Wyłączenie diody
  delay(500); //Odczekanie 0,5 sekundy
}
```

Koniec przykładu

Funkcje własne z argumentami

Gdy podczas wywoływania funkcji własnej chcemy, aby możliwe było użycie argumentu który może ulegać zmianie trzeba w deklaracji funkcji dodać informację, że może ona przyjmować argument. Robimy przy definiowaniu nazwy funkcji.

Funkcja własna z jednym argumentem

```
void setup()
{
  ......
}
void loop()
{
  mojaFunkcja(x) /*wywołanie funkcji własnej z zadanym argumentem x
  który jest zmienną globalną */
  ......
}

void mojaFunkcja(int paraMetr) /* definiowanie funkcji własnej
o nazwie mojaFunkcja z argumentem całkowitoliczbowym paraMetr,
który jest zmienną lokalną działajacą tylko w ramach definicji funkcji*/
{
  // tu napisać kod funkcji własnej
}
```

Uwaga: jeśli chcemy aby funkcja nie zwracała wyniku to przed je nazwą piszemy void, gdy ma zwracać wynik to przed jej nazwą deklarujemy typ zmiennych jakie są zwracane np. int. W przykładzie jak obok funkcja własna nie zwraca wyniku.

Przykład

Aby migać diodą podłączoną do pinu nr 13 należało napisać taki program:

```arduino
void setup() {
  pinMode(13, OUTPUT); //Konfiguracja pinu 13 jako wyjście
}

void loop() {
  digitalWrite(13, HIGH); //Włączenie diody
  delay(1000); //Odczekanie 1 sekundy
  digitalWrite(13, LOW); //Wyłączenie diody
  delay(1000); //Odczekanie 1 sekundy
}
```

Poniżej zastosowano funkcję własną z jednym parametrem zamigajLED(int czas).
Parametrem jest liczba całkowitoliczbowa o nazwie czas tu liczba ta oznacza czas przez jaki dioda ma być włączona i wyłączona.

```arduino
void setup() {
  pinMode(13, OUTPUT); //Konfiguracja pinu 13 jako wyjście
}

void loop() {
  zamigajLED(1000);
}

void zamigajLED(int czas){
  digitalWrite(13, HIGH); //Włączenie diody
  delay(czas); //Odczekanie zadeklarowanego czasu
  digitalWrite(13, LOW); //Wyłączenie diody
  delay(czas); //Odczekanie zadeklarowanego czasu
}
```

Koniec przykładu

Funkcja własna z dwoma argumentami

```
void setup()
{
  ......
}
void loop()
{
  mojaFunkcja(x,y) /*wywołanie funkcji własnej z zadanymi argumentami x oraz y
  które są  zmiennymi globalnymi */
  ......
}

void mojaFunkcja(int paraMetr1, int paraMetr2) /* definiowanie funkcji własnej
o nazwie mojaFunkcja z argumentami całkowitoliczbowymi paraMetr1 oraz paraMetr2
które są zmiennymi lokalnymi działajacymi tylko w ramach definicji funkcji */
{
  // tu napisać kod funkcji własnej
}
```

Przykład

Poniżej przedstawiono program w którym dioda na płytce Arduino zaświeca się pięciokrotnie na czas jednej sekundy

```
1  void setup() {
2    pinMode(13, OUTPUT); //Konfiguracja pinu 13 jako wyjście
3   for (int i=0; i < 5; i++) {
4      digitalWrite(13, HIGH); //Włączenie diody
5      delay(1000); //Odczekanie jakiegoś czasu
6      digitalWrite(13, LOW); //Wyłączenie diody
7      delay(1000); //Odczekanie jakiegoś czasu
8  }
9  }
10 void loop() {
11 }
```

Realizacja tego samego programu przy wykorzystaniu funkcji własnej dwuargumentowej.

Funkcja własna jak i uprzednio program bez użycia funkcji własnej jest umieszczony w części konfiguracyjnej a nie w pętli. Spowodowane jest to tym, aby został wykonany jednorazowo (pięć mignięć diody umieszczonej na płytce).

```
1  void setup() {
2    pinMode(13, OUTPUT); //Konfiguracja pinu 13 jako wyjście
3    zamigajLED(1000, 5);
4  }
5
6  void loop() {
7  }
8
9  void zamigajLED(int czas, int ile){
10   for (int i=0; i < ile; i++) {
11     digitalWrite(13, HIGH); //Włączenie diody
12     delay(czas); //Odczekanie jakiegoś czasu
13     digitalWrite(13, LOW); //Wyłączenie diody
14     delay(czas); //Odczekanie jakiegoś czasu
15   }
16 }
```

Koniec przykładu

Modulacja szerokości impulsów

PWM (Pulse Width Modulation) – metoda regulacji sygnału prądowego lub napięciowego, o stałej amplitudzie i częstotliwości, polegająca na zmianie wypełnienia sygnału, używana m.in. w układach sterujących pracą silników elektrycznych.

Współczynnik wypełnienia impulsu – stosunek czasu trwania impulsu do okresu tego impulsu.
Wyrażany jest w postaci ułamka (z zakresu od 0 do 1) lub w procentach.

W Arduino wybrane piny mogą używać polecenia analogWrite() do generowania impulsów PWM.
Są to piny oznaczone znakiem tylda (wężyk) **~** o numerach 3, 5, 6, 9, 10 i 11.
Polecenie analgWrite() przyjmuje dwa argumenty:

analogWrite(numer pinu, wartość od 0 do 255);

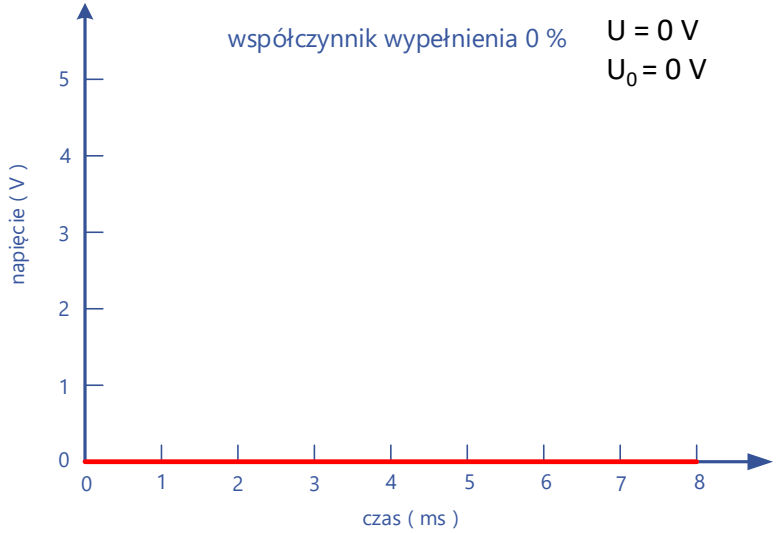
zakres wartości od 0 do 255 wynika z tego, modulacja szerokości impulsu jest 8-bitową wartością zawierającą się w przedziale od 0 do 2^8-1.

Im większa wartość tym większy współczynnik wypełnienia impulsu.

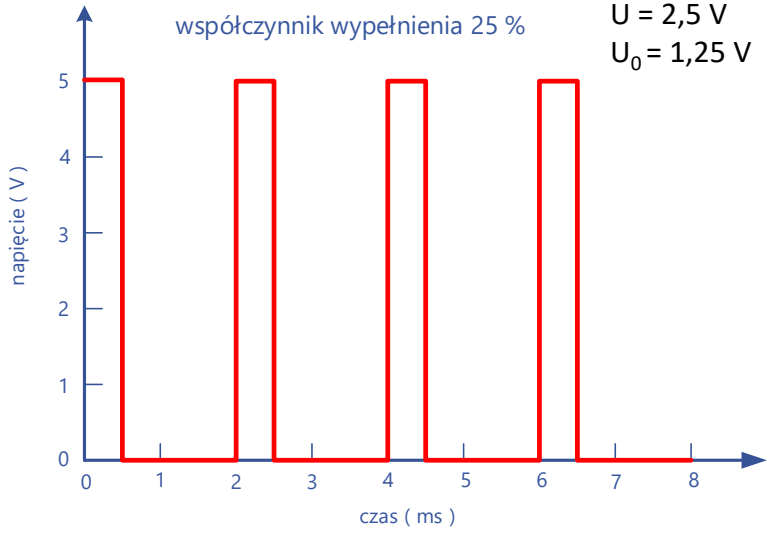
Modulacja szerokości impulsów dla różnych współczynników wypełnienia

U - wartość skuteczna napięcia
U_0 - wartość średnia napięcia (składowa stała)

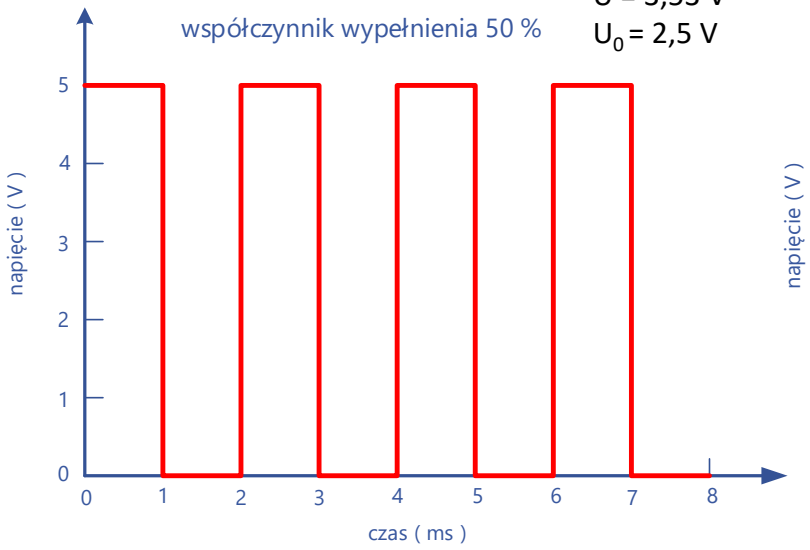
współczynnik wypełnienia 0 %
U = 0 V
U_0 = 0 V



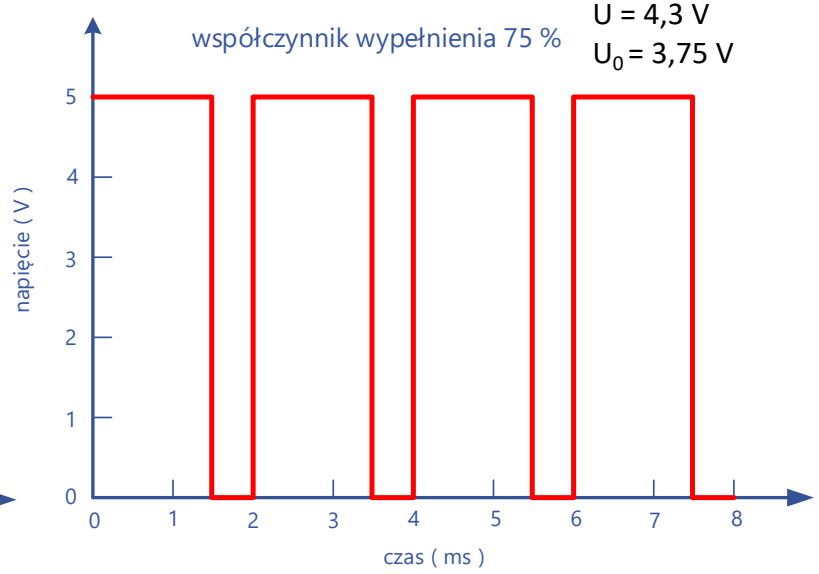
współczynnik wypełnienia 25 %
U = 2,5 V
U_0 = 1,25 V



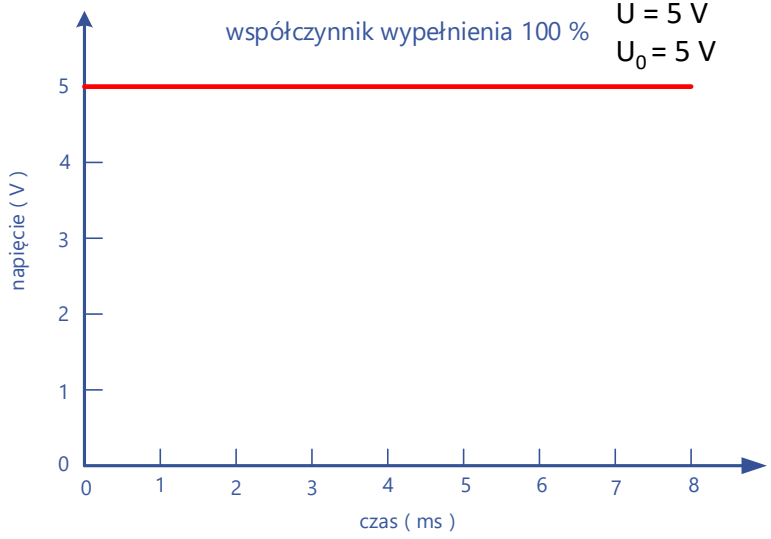
współczynnik wypełnienia 50 %
U = 3,53 V
U_0 = 2,5 V



współczynnik wypełnienia 75 %
U = 4,3 V
U_0 = 3,75 V



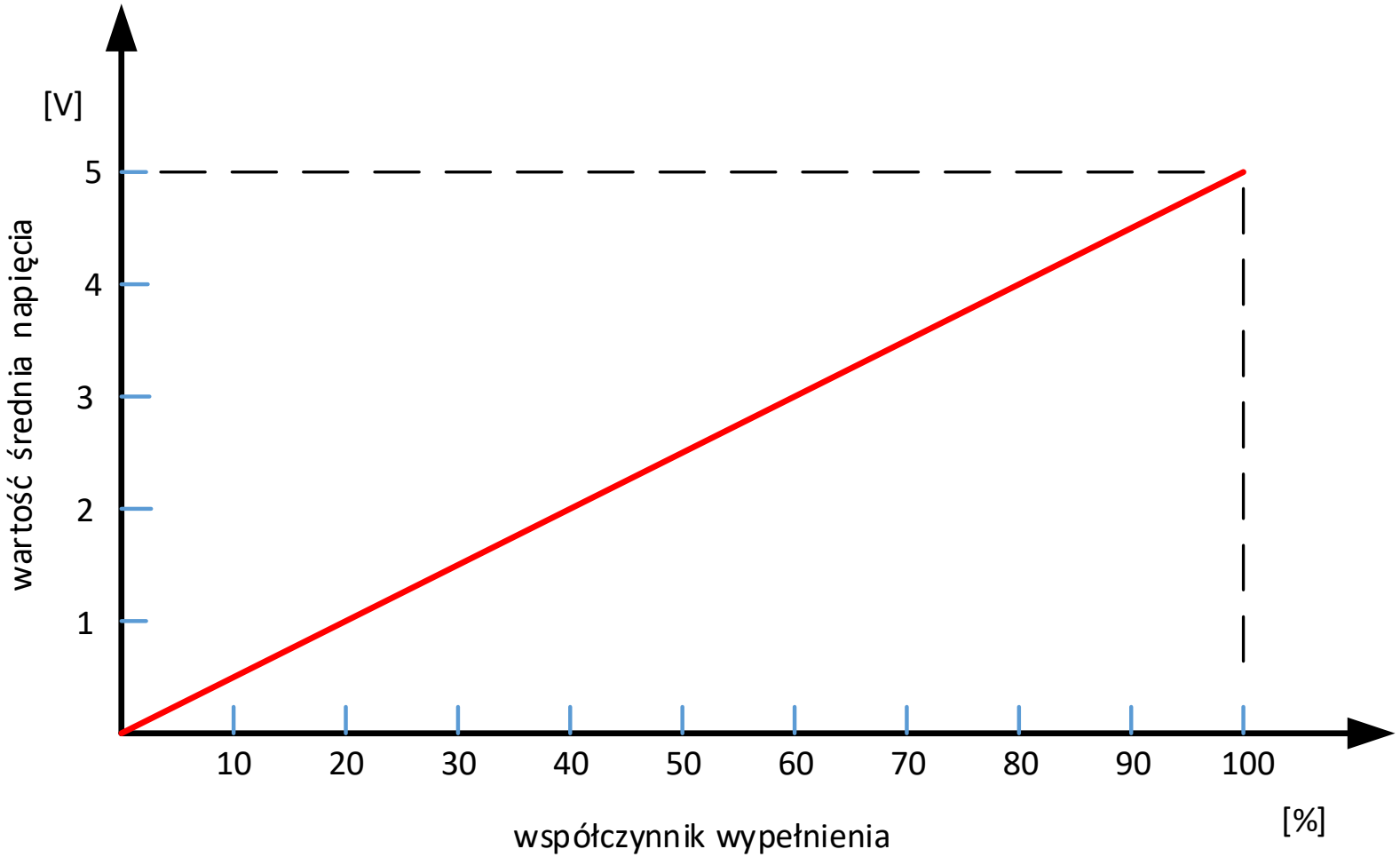
współczynnik wypełnienia 100 %
U = 5 V
U_0 = 5 V



przebieg zmian wartości średniej napięcia w zależności od wartości współczynnika wypełnienia impulsu

wartość średnia (składowa stała) napięcia

$$U_0 = \frac{1}{T}\int_0^T u\,dt$$

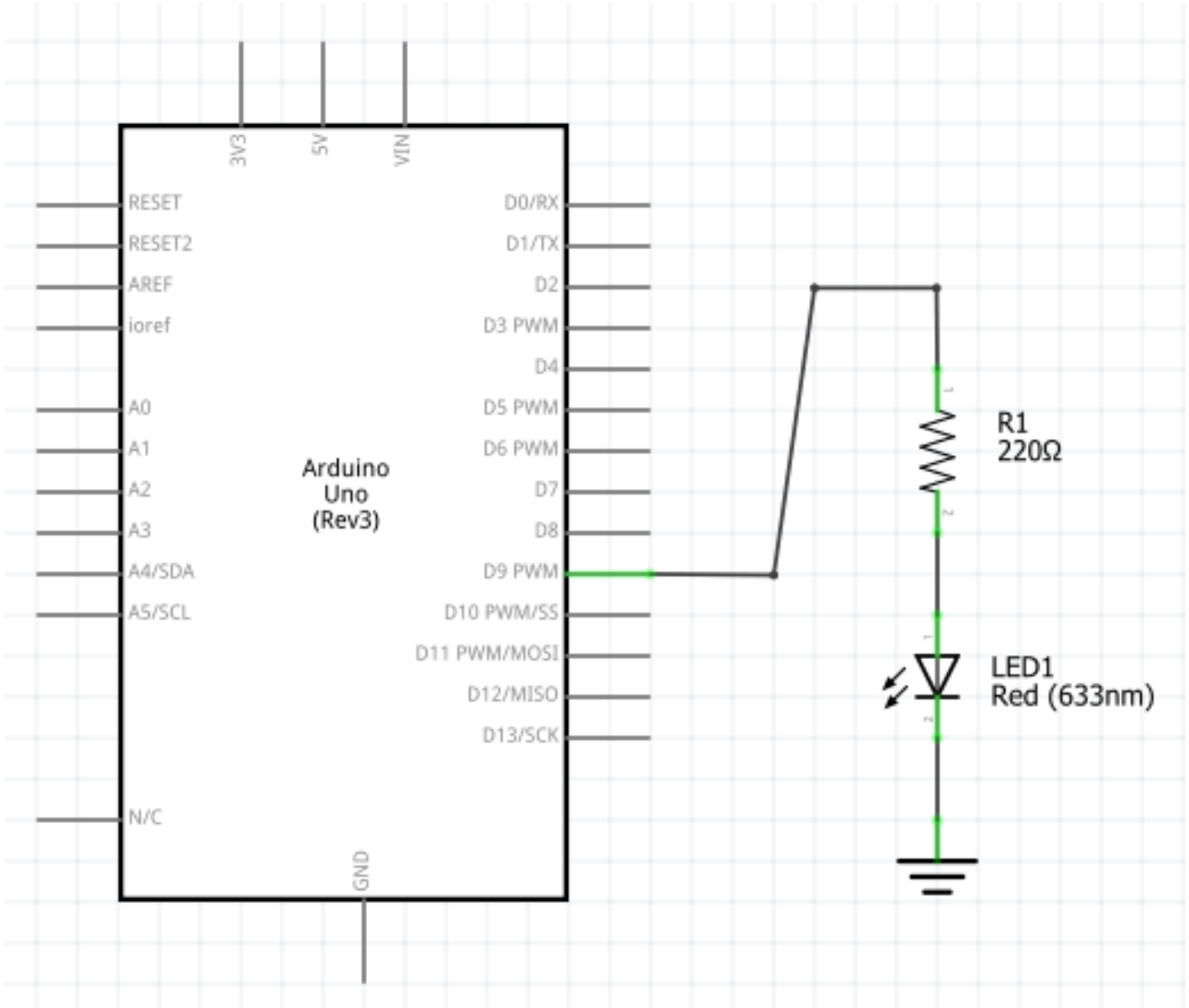


przykład zastosowania modulacji szerokości impulsów

zmiana jasności świecenia diody LED

```
/*
Modulacja szerokości impulsów
zmiana jasności diody LED
*/

const int LED=9;   // Przypisanie LED do pinu 9
void setup()
{
  pinMode (LED, OUTPUT); // Ustawienie pinu LED jako wyjście
}

void loop()
{
  for (int i=0; i<256; i++)
  {
    analogWrite(LED, i);//wpisywanie wartości i do pinu analogowego
    //wartości coraz wyższe od 0 do 256
    delay(10);//oczekiwanie 10 ms
  }
  for (int i=255; i>=0; i--)
  {
    analogWrite(LED, i);//wpisywanie wartości i do pinu analogowego
    //wartości coraz nizsze od 255 do 0
    delay(10);//oczekiwanie 10 ms
  }
}
```

schemat połączeń do przykładu zmiana jasności świecenia diody LED



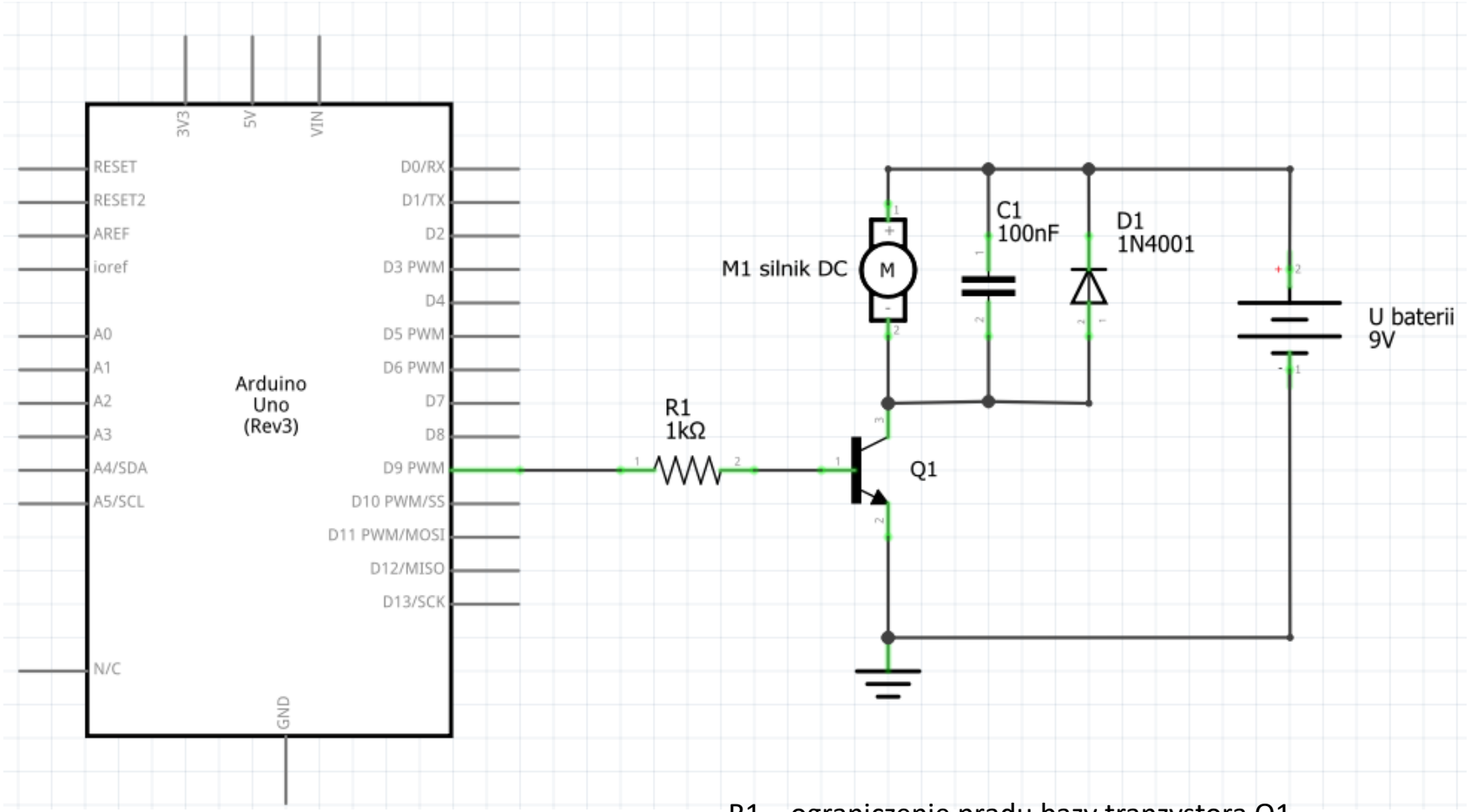
przykład zastosowania modulacji szerokości impulsów

zmiana prędkości obrotowej silnika

tranzystor_i_silnik

```
/*
tranzystor sterujący silnikiem
i regulujący jego prędkość wykorzystując PWM
*/

//program sterujący prędkością obrotowa slnika

const int SILNIK=9;
/* const kwantyfikator zmiennej powoduje, ze zmienna SILNIK wystapi tylko do odczytu
zadeklarowana jako całkowitoliczbowa (int) i ma wartość =9.
W ten sposób sygnalizujemy sobie że silnik podłączamy do pinu 9 */

void setup()
{
    pinMode (SILNIK, OUTPUT);// deklaracja pinu 9 (SILNIK) jako wyjścia
}

void loop()
{
  //poniżej zwiekszamy prędkość obrotową silnika wykorzystując PWM
    for (int i=0; i<256; i++)
    {
        analogWrite(SILNIK, i); //wpisywanie wartości i do pinu analogowego
        delay(60);//czekamy 60 ms
    }
    delay(2000);
     //poniżej zmniejszamy prędkość obrotową silnika wykorzystując PWM
        for (int i=255; i>=0; i--)
    {
        analogWrite(SILNIK, i);//wpisywanie wartości i do pinu analogowego
        delay(60);//czekamy 60 ms
    }
    delay(2000);
}
```

schemat połączeń do przykładu zmiana prędkości obrotowej silnika



R1 – ograniczenie prądu bazy tranzystora Q1
C1 – filtracja zakłóceń pochodzących z silnika
D1 – zabezpiecza przed napięciem wstecznym pochodzącym z silnika pracującego chwilowo jako prądnica

literatura

Margolis Michael, Jepson Brian, Weldin Nicholas Robert: Arduino. Przepisy na rozpoczęcie i udoskonalenie projektów. Wydanie III. Helion, Gliwice 2021

Blum Jeremy: Odkrywanie Adruino. Narzędzia i techniki inżynierii pełnej czaru. Helon, Gliwice 2020

Karaś Władysław: Mikrokontrolery AVR. Język C – podstawy programowania. Wydanie II poprawione. Wydawnictwo Atnel, Szczecin 2013

Baranowski Rafał: Mikrokontrolery AVR ATmega w praktyce. Wydawnictwo BTC, Warszawa 2005

ELEMENTY POGRAMOWANIA MIKROKONTROLERÓW

KONIEC