

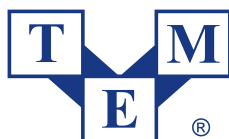


Dokumentacja zestawu edukacyjnego

Arduino based TME Educational Board

TME-EDU-ARD-2

Styczeń 2019



Electronic Components

WWW.TME.EU

www.TMEeducation.com

Spis treści

Wstęp	3
Czym jest Arduino?	3
Ogólny opis zestawu	3
Lista dostępnych peryferiów	4
Instrukcja bezpieczeństwa	4
Zasilanie	6
Niezbędne oprogramowanie	6
Programowanie zestawu edukacyjnego	7
Komentarze	7
Wgrywanie programu na płytke	8
Obsługa peryferiów w praktyce	8
Zwykła, jednokolorowa dioda świecąca (LED1 - D13)	8
Wykorzystywanie definicji	9
Opóźnienia	10
Buzzer z generatorem (D2)	11
Zwykła, trójkolorowa (RGB) dioda świecąca (LED2 - D9, D10, D11)	11
Sterowanie diody RGB sygnałem PWM	13
Klawiatura składająca się z 5 przycisków (D4, D5, D6, D7, D8)	14
Komunikacja z komputerem przez UART	16
Wysyłanie informacji do Arduino	17
Wyświetlacz tekstowy LCD 2x16 znaków	18
Obsługa czujników analogowych	20
Czujniki analogowe - potencjometr (A1)	20
Czujniki analogowe - czujnik światła (A3)	22
Czujniki analogowe - czujnik temperatury (A2)	23
Czujniki analogowe - mikrofon (A0)	24
Ekspander i wyświetlacz 7-segmentowy	25
Ekspander i wyświetlacz 7-segmentowy (cyfry)	27
Pętla <i>for</i>	28
Pętla <i>while</i>	28
Wyświetlacz graficzny OLED	29
Odbiornik podczerwieni (D3)	30
Obsługa diod RGB sterowanych cyfrowo (D12)	31
Moduł Bluetooth	33
Zworki konfiguracyjne	34
Złącza do shieldów Arduino	35
Opis funkcji Arduino	36
Lista bibliotek wraz z licencjami	40

Wstęp

Zestaw edukacyjny TME do nauki programowania Arduino jest częścią programu TME Education, więcej informacji na jego temat znaleźć można na stronach:

- <https://www.tmeeducation.com>
- <https://fb.com/tmeeducation>

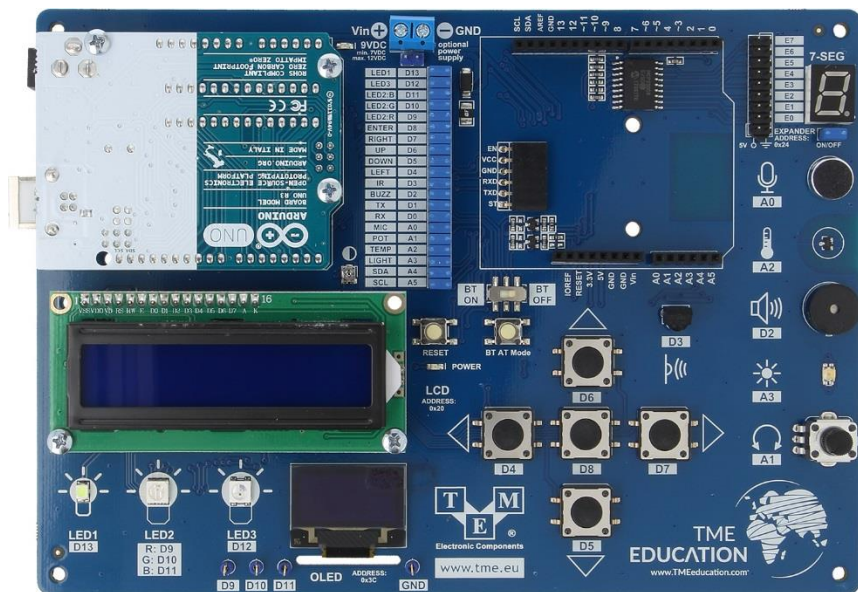
Niniejsza dokumentacja omawia wykorzystanie peryferiów znajdujących się na wyposażeniu zestawu edukacyjnego **TME-EDU-ARD-2**, który współpracuje z popularnym sterownikiem **Arduino UNO**. Na płycie znajduje się ponad 20 peryferiów wykorzystujących wszystkie dostępne wyprowadzenia użytego kontrolera. Elementy są połączone, zachowano jednak możliwość zmian tych połączeń za pomocą systemu zworek (o czym więcej w dalszej części).

Czym jest Arduino?

Platforma Arduino jest jednym z najpopularniejszych na świecie rozwiązań dedykowanych dla początkujących elektroników. Integruje popularne mikrokontrolery AVR oraz dedykowany, przyjazdy dla początkujących język programowania (oparty na C/C++) w spójne i łatwe w użyciu narzędzie mające nieskończoną liczbę zastosowań.

Ogólny opis zestawu

W lewym górnym narożniku zestawu znajduje się miejsce na sterownik Arduino UNO. W lewym dolnym rogu znajduje się sekcja z wyświetlaczami (LCD i OLED) oraz z diodami świecącymi (LED). Na prawo od wyświetlacza znajduje się 5 dużych przycisków monostabilnych.



Po prawej stronie znajduje się odbiornik IR, potencjometr, czujnik światła, buzzer, czujnik temperatury, mikrofon oraz wyświetlacz 7-segmentowy z ekspanderem pinów. W górnej, środkowej części płytki znaleźć można złącze zewnętrznego zasilania, komplet zworek do rekonfiguracji połączeń oraz miejsce na moduł Bluetooth oraz ewentualny, dodatkowy shield z innymi akcesoriami.

Lista dostępnych peryferiów

Do Arduino UNO podłączone są następujące peryferia:

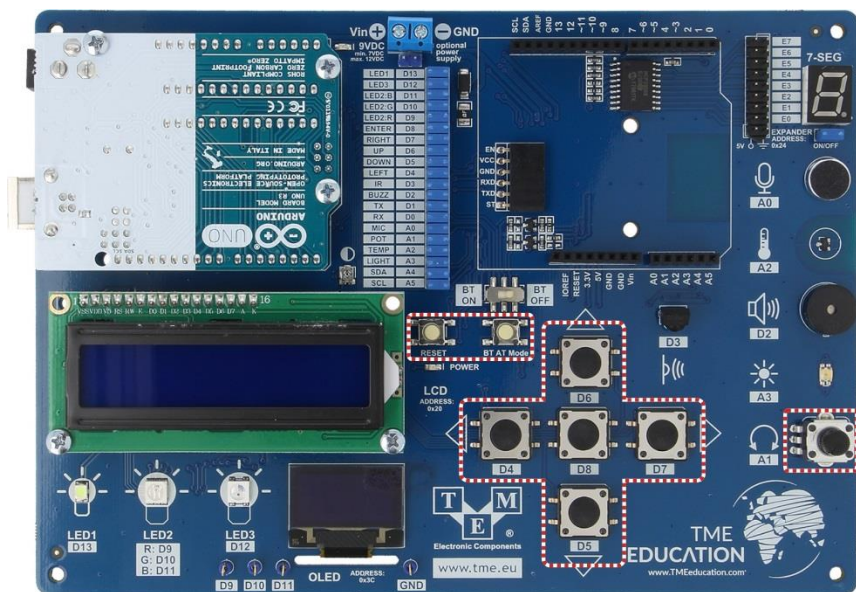
- Zwykła, jednokolorowa dioda świecąca (LED1 - D13).
- Zwykła, trójkolorowa (RGB) dioda świecąca (LED2 - D9, D10, D11).
- Sterowna cyfrowo, trójkolorowa (RGB) dioda świecąca WS2812 (LED3 - D12), do której szeregowo podłączono kolejne 4, identyczne diody umieszczone na spodzie płytki.
- Klawiatura składająca się z 5 przycisków monostabilnych rozmieszczonych w układzie przód (D6), tył (D5), lewo (D4), prawo (D7) oraz środek (D8).
- Czujniki analogowe:
 - Potencjometr (A1),
 - Czujnik światła (A3),
 - Termometr (A2),
 - Mikrofon (A0).
- Odbiornik podczerwieni (D3).
- Buzzer z generatorem (D2).
- Wyświetlacz tekstowy LCD 2x16 znaków, podłączony przez ekspander I2C (adres: 0x20).
- Wyświetlacz graficzny OLED z interfejsem I2C (adres: 0x3C).
- Wyświetlacz 7-segmentowy, podłączony przez ekspander I2C (adres: 0x24).

Oprócz tego na płytce znajduje się:

- Dioda świecąca (POWER) oznaczająca poprawne zasilanie układu.
- Mały przycisk monostabilny (RESET) pozwalający na zresetowanie sterownika Arduino.
- Mały przycisk monostabilny (BT AT Mode) pozwalający wejść w tryb komend AT modułu BT.
- Suwakowy przełącznik dwupozycyjny (BT ON/BT OFF) pozwalający odłączyć opcjonalny moduł BT.
- 4 punkty pomiarowe w formie oczek (podłączone do D9, D10, D11, GND)

Instrukcja bezpieczeństwa

Podczas pracy urządzenia należy unikać bezpośredniego kontaktu z płytką (dotykania elementów elektronicznych oraz ścieżek), ponieważ w skrajnych sytuacjach może to doprowadzić do uszkodzenia zestawu. Wyjątek stanowi 7 przycisków oraz potencjometr, które mogą być używane w programach wgrywanych na płytkę.

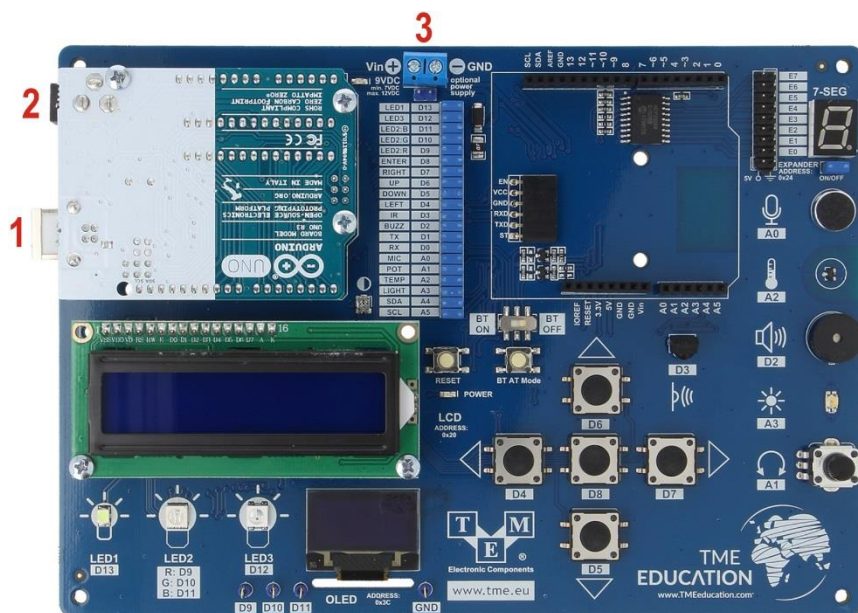


Ewentualne zmiany konfiguracji zwrotek oraz podłączanie dodatkowych modułów powinny odbywać się przy wyłączonym zasilaniu zestawu!

Przed uruchomieniem zestawu warto również upewnić się, że płytka nie została przypadkiem położona na metalowych przedmiotach (np. na śrubokręcie), które mogłyby zewrzeć sygnały płynące pod płytka.

Zasilanie

Zestaw edukacyjny można zasilać na jeden z trzech poniższych sposobów:



- Złącze USB (1) umieszczone na płytce Arduino (zasilanie z portu USB komputera lub z zasilacza 5V pozwalającego na podłączenie przewodu z końcówką USB typu B).
- Złącze zasilania zewnętrznego (2) na płytce Arduino (od 7V do 12V).
- Złącze zasilania zewnętrznego (3) umieszczone na płytce rozszerzającej (od 7V do 12V).

W zdecydowanej większości przypadków zestaw edukacyjny najlepiej zasilać za pomocą przewodu USB, który używany jest do programowania Arduino.

Niezbędne oprogramowanie

Do programowania zestawu potrzebne jest środowisko Arduino IDE, które można bezpłatnie pobrać ze strony projektu, tj.: <https://arduino.cc>. W menu nawigacyjnym strony wybieramy zakładkę „SOFTWARE”. Na nowo otwartej stronie należy przejść do sekcji „Download the Arduino IDE” i wybrać z listy po prawej stronie odpowiednią dla danego systemu wersję.

Następnie wyświetli się zapytanie o dobrowolne wsparcie finansowe dla projektu. Można w tym kroku przekazać datkę lub pobrać oprogramowanie za darmo wybierając przycisk „JUST DOWNLOAD”. W tym momencie rozpocznie się pobieranie instalatora.

Programowanie zestawu edukacyjnego

Programy pisane na Arduino nazywane są szkicami. Wszystkie programy powinny zawierać funkcję `setup` oraz `loop`. Instrukcje zawarte w funkcji `setup` wykonują się tylko raz (podczas włączania Arduino). W tym miejscu należy umieszczać wszystkie ustawienia oraz instrukcje, które mają być wykonane na początku pracy urządzenia (zaraz po jego włączeniu).

Funkcja `loop` pełni rolę nieskończonej pętli. Instrukcje zawarte w jej wnętrzu będą się wykonywały cały czas. Pętla ta zaczyna działać po zakończeniu instrukcji wykonywanych we wnętrzu funkcji `setup`.

Pusty szkic, który powinien być bazą wyjściową każdego programu:

```
// instrukcje z funkcji setup wykonują się tylko raz (po starcie Arduino)
void setup() {

}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {

}
```

Komentarze

Wewnątrz programu możliwe jest umieszczanie komentarzy, czyli informacji, które są pomijane podczas przesyłania szkicu do Arduino i mają one za zadanie jedynie ułatwić programistom rozumienie kodu. Komentarze mogą mieścić się w jednej linii (wtedy należy poprzedzić je znakami `"/"/`) lub w wielu liniach (wtedy tekst komentarza należy umieścić między znakami `"/*"` oraz `"*/"`).

```
void setup() {
    //Komentarz w jednej linii

    /*
    Komentarz
    w
    wielu
    liniach
    */
}
```

Komentarze mogą być umieszczane w dowolnej części programu. Często używa się ich również w celu chwilowego "wyłączenia" fragmentu programu np. na czas testów.

Wgrywanie programu na płytke

W celu wgrania programu na Arduino należy najpierw (jednorazowo):

1. podłączyć zestaw przewodem USB do komputera,
2. w pasku narzędzi Arduino IDE wybrać: Narzędzia > Płytką > Arduino/Genuino UNO,
3. w pasku narzędzi Arduino IDE wybrać: Narzędzia > Port i zaznaczyć tam numer portu COM, obok którego pojawi się nazwa Arduino UNO.

Następnie w celu skompilowania programu (czyli przetłumaczeniu go na język zrozumiały dla Arduino) i przesłania go na płytkę należy kliknąć opcję Wgraj, która widoczna jest w menu Arduino IDE w formie okrągłej ikonki ze strzałką w prawo. Po poprawnym załadowaniu programu na dole środowiska pojawi się stosowny komunikat np. "Ładowanie zakończone", a po chwili kod zacznie działać na płytce.

Jeśli do układu podłączony jest opcjonalny moduł Bluetooth to podczas wgrywania programu na płytkę odpowiedni przełącznik suwakowy powinien być ustawiony w pozycji *BT OFF*.

Obsługa peryferiów w praktyce

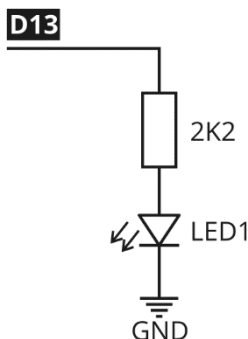
W tej części instrukcji opisane zostały najważniejsze informacje dotyczące uruchomienia wszystkich peryferiów, które umiejscowione są na płytce. Elementy zostały opisane w kolejności od najłatwiejszego do najtrudniejszego, dzięki czemu możliwe jest stopniowe poznanie kolejnych aspektów języka Arduino w praktyce.

Zwykła, jednokolorowa dioda świecąca (LED1 - D13)

Zestaw edukacyjny został wyposażony w jedną, zwykłą diodę świecącą, która znajduje się w lewym dolnym rogu płytki. Obok diody znajduje się napis D13, co oznacza, element ten można obsługiwać w sposób cyfrowy (D od *digital*). Numer "13" to numer pinu Arduino, do którego podłączony jest dany element przy domyślnej konfiguracji zworek. Analogiczne opisy zostały umieszczone przy wszystkich peryferiach.

Dioda została podłączona anodą do wyprowadzenia Arduino (przez rezystor). Oznacza to, że w celu włączenia diody należy na wyjściu cyfrowym 13 ustawić logiczny stan wysoki (ang. *high*), czyli "1".

Program uruchamiający diodę powinien rozpoczynać się od ustawienia pinu w roli wyjścia. W tym celu należy wykorzystać funkcję `pinMode(nr_pinu, OUTPUT)`, w której za `nr_pinu` podstawiamy numer pinu, który ma działać w trybie wyjściowym (ang. *output*).



Od tego momentu, w dowolnym miejscu programu możliwe będzie włączanie lub wyłączanie diody za pomocą funkcji `digitalWrite(nr_pinu, stan)`. Gdzie, analogicznie w miejscu `nr_pinu` wstawiamy numer pinu, do którego podłączona została dioda. Natomiast w miejsce opisane jako stan podajemy jeden, z dwóch możliwych wariantów *LOW*, czyli stan niski (logiczne 0) lub *HIGH*, czyli stan wysoki (logiczne 1).

Ustawienie stanu niskiego jest równoznaczne z wyprowadzeniem na wyjście potencjału masy, a ustawienie stanu wysokiego jest równoznaczne z wyprowadzeniem na wyjście dodatniej szyny zasilania, czyli w tym przypadku 5V.

Program uruchamiający diodę będzie więc wyglądał następująco:

```
//instrukcje z funkcji setup wykonują się tylko raz (po starcie Arduino)
void setup() {
    // Konfiguracja pinu jako wyjście
    pinMode(13, OUTPUT);

    // włączenie diody LED
    digitalWrite(13, HIGH);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
}
```

Obie instrukcje zostały umieszczone w funkcji `setup`, ponieważ po włączeniu Arduino diodę włączamy tylko raz i nie musimy już nic z nią robić.

Wykorzystywanie definicji

W powyższym programie numer pinu, do którego podłączona jest dioda należało wpisać w dwóch miejscach (w konfiguracji wyjścia oraz w instrukcji włączającej diodę). Dla ułatwienia późniejszych zmian informacje na temat numeru pinu można zapisać w formie definicji np.:

```
#define LED1 13
```

Od tej pory w każdym miejscu programu, w którym umieszczone zostanie `"LED1"` kompilator podstawy automatycznie numer `"13"`. Jest to szczególnie wygodne, podczas wprowadzania zmian pinów, do których podłączone są peryferia w bardziej rozbudowanych programach.

Program włączający diodę z użyciem definicji, może wyglądać następująco:

```
#define LED1 13

// instrukcje z funkcji setup wykonują się tylko raz (po starcie Arduino)
void setup() {
    // Konfiguracja pinu jako wyjście
    pinMode(LED1, OUTPUT);

    // włączenie diody LED
    digitalWrite(LED1, HIGH);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
}
```

Opóźnienia

Podczas tworzenia wielu programów przydaje się możliwość wprowadzenia opóźnienia. Popularnym przykładem jest program migający diodą. W praktyce oznacza to konieczność włączenia diody, odczekania określonego czasu, wyłączenia diody, odczekania określonego czasu i rozpoczęcie cyklu od początku. W tym celu podczas programowania Arduino wykorzystuje się funkcję `delay(czas)`, gdzie za `czas` podstawiamy czas trwania opóźnienia w milisekundach (1 sekunda = 1000 milisekund).

Podczas trwania opóźnienia cały program jest zatrzymywany
(i stoi w miejscu)!

Przykład programu migającego diodą:

```
#define LED1 13

// instrukcje z funkcji setup wykonują się tylko raz (po starcie Arduino)
void setup() {
    // Konfiguracja pinu jako wyjście
    pinMode(LED1, OUTPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
    // włączenie diody LED
    digitalWrite(LED1, HIGH);
    delay(500); // odczekaj 500 ms

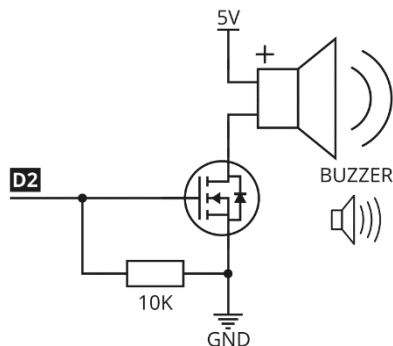
    // wyłączenie diody LED
    digitalWrite(LED1, LOW);
    delay(200); // odczekaj 200 ms
}
```

Tym razem instrukcje sterujące diodą zostały przeniesione do funkcji *loop*, ponieważ mają one być wykonywane cały czas (w pętli). Oczywiście instrukcja konfiguracyjna pin (do którego podłączona jest dioda) jako wyjście pozostała wewnątrz funkcji *setup* (ponieważ konfigurację wykonujemy tylko raz).

Buzzer z generatorem (D2)

Do pinu D2 podłączony jest buzzer z generatorem, czyli element wydający dźwięk. Sterowanie tym elementem przebiega w sposób identyczny do powyższego przykładu z diodą. Logiczna "1" na wyjściu Arduino sprawi, że buzzer zacznie wydawać dźwięk.

Przykładowy program wykorzystujący generator dźwięku:



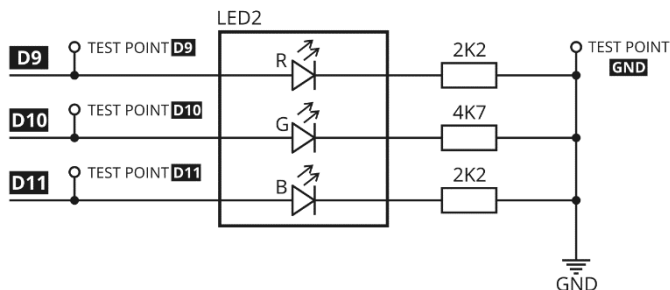
```
#define BUZ 2

// instrukcje z funkcji setup wykonują się tylko raz (po starcie Arduino)
void setup() {
    // konfiguracja pinu jako wyjście
    pinMode(BUZ, OUTPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
    // włączenie buzzera
    digitalWrite(BUZ, HIGH);
    delay(1000); // odczekaj 1000 ms
    // wyłączenie buzzera
    digitalWrite(BUZ, LOW);
    delay(200); // odczekaj 200 ms
}
```

Zwykła, trójkolorowa (RGB) dioda świecąca (LED2 - D9, D10, D11)

Na płytce znajduje się dioda LED2, która jest diodą typu RGB. Oznacza to, że w jej obudowie znajdują się 3 struktury świecące: czerwona (R), zielona (G) oraz niebieska (B).



Korzystając z opisów umieszczonych na płytce można utworzyć odpowiednie definicje:

```
#define LED2R 9
#define LED2G 10
#define LED2B 11
```

Sterowanie poszczególnymi diodami odbywa się w sposób analogiczny do poprzednio opisanej diody LED1. Zaczynamy od konfiguracji pinów jako wejścia, a następnie sterujemy każdym kolorem osobno:

```
#define LED2R 9
#define LED2G 10
#define LED2B 11

// instrukcje z funkcji setup wykonują się tylko raz (po starcie Arduino)
void setup() {
  // konfiguracja pinów
  pinMode(LED2R, OUTPUT);
  pinMode(LED2G, OUTPUT);
  pinMode(LED2B, OUTPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  // włącz i wyłącz niektóre kolory LED
  digitalWrite(LED2R, HIGH);
  digitalWrite(LED2G, LOW);
  digitalWrite(LED2B, LOW);
  delay(500); // odczekaj 500 ms

  // włącz i wyłącz niektóre kolory LED
  digitalWrite(LED2R, LOW);
  digitalWrite(LED2G, HIGH);
  digitalWrite(LED2B, LOW);
  delay(500); // odczekaj 500 ms

  // włącz i wyłącz niektóre kolory LED
  digitalWrite(LED2R, LOW);
  digitalWrite(LED2G, LOW);
  digitalWrite(LED2B, HIGH);
  delay(500); // odczekaj 500 ms
}
```

Oczywiście możliwe jest również świecenie dwoma/trzema kolorami jednocześnie. Dzięki temu możliwe jest mieszanie barw i uzyskiwanie różnych, pośrednich kolorów. Przykład takiego programu:

```
#define LED2R 9
#define LED2G 10
#define LED2B 11
```

```
// instrukcje z funkcji setup wykonują się tylko raz (po starcie Arduino)
void setup() {

  // konfiguracja pinów
  pinMode(LED2R, OUTPUT);
  pinMode(LED2G, OUTPUT);
  pinMode(LED2B, OUTPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  // ustawienie kolorów LED RGD
  digitalWrite(LED2R, HIGH);
  digitalWrite(LED2G, HIGH);
  digitalWrite(LED2B, LOW);
  delay(500); // odczekaj 500 ms

  // ustawienie kolorów LED RGD
  digitalWrite(LED2R, LOW);
  digitalWrite(LED2G, HIGH);
  digitalWrite(LED2B, HIGH);
  delay(500); // odczekaj 500 ms

  // ustawienie kolorów LED RGD
  digitalWrite(LED2R, HIGH);
  digitalWrite(LED2G, LOW);
  digitalWrite(LED2B, HIGH);
  delay(500); // odczekaj 500 ms
}
```

Sterowanie diody RGB sygnałem PWM

Na niektórych wyprowadzeniach Arduino (oznaczonych znakiem "~") możliwe jest uzyskanie sygnału prostokątnego o zmiennym wypełnieniu (PWM). Na zestawie edukacyjnym TME-EDU-ARD-2 wszystkie kolory diody RGB zostały podłączone właśnie do takich wyprowadzeń. Dzięki temu możliwe jest dodatkowe sterowanie jasnością każdego z nich.

W celu wygenerowania sygnału PWM należy wykorzystać funkcję *analogWrite(nr_pinu, wypełnienie)*, gdzie za *nr_pinu* podstawiamy numer wyprowadzenia, do którego podłączona jest dioda, a za *wypełnienie* podajemy liczbę z zakresu od 0 do 255.

Ustawienie wartości 0 będzie równoznaczne ze świeceniem diody z jasnością 0%, a 255, będzie oznaczało świecenie z jasnością równą 100%.

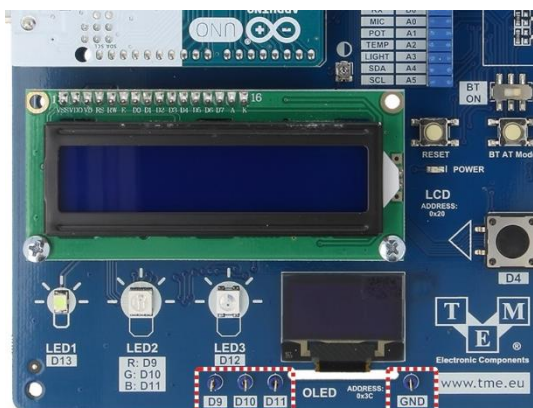
Poniższy przykład demonstruje w jaki sposób, zmienia się jasność diody czerwonej:

```
#define LED2R 9
#define LED2G 10
#define LED2B 11
```

```
void setup() {  
    // konfiguracja pinów  
    pinMode(LED2R, OUTPUT);  
}  
  
void loop() {  
    // ustaw jasność diody  
    analogWrite(LED2R, 0);  
    delay(500); // odczekaj 500 ms  
  
    // ustaw jasność diody  
    analogWrite(LED2R, 50);  
    delay(500); // odczekaj 500 ms  
  
    // ustaw jasność diody  
    analogWrite(LED2R, 150);  
    delay(500); // odczekaj 500 ms  
  
    // ustaw jasność diody  
    analogWrite(LED2R, 255);  
    delay(500); // odczekaj 500 ms  
}
```

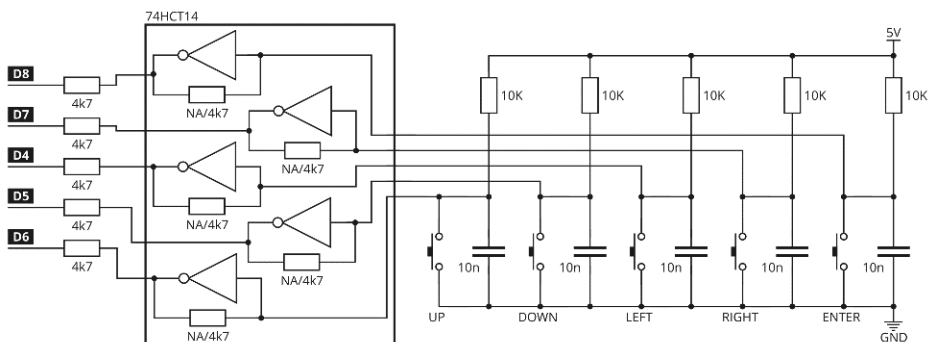
Za pomocą oczek testowych opisanych jako D9, D10, D11 oraz GND możliwe jest podłączenie płytki do oscyloskopu, na którym widoczne będą zmiany wypełnienia sygnałów, którymi sterowana jest dioda.

W przypadku braku oscyloskopu można w to miejsce podłączyć woltomierz (między dowolny punkt pomiarowy, a GND). Wraz ze wzrostem wypełnienia sygnału PWM woltomierz powinien wskazywać coraz wyższe napięcie (od 0 do 5V).



Klawiatura składająca się z 5 przycisków (D4, D5, D6, D7, D8)

Na płycie znajduje się 5 monostabilnych przycisków, które są podłączone do pinów cyfrowych D4, D5, D6, D7 oraz D8. Wszystkie przyciski zostały wyposażone w filtry RC, dzięki czemu odczyty są stabilne i nie ma potrzeby programowego filtrowania wejść w celu uniknięcia tzw. drgań styków.



Przyciski podłączone są przez bufony odwracające, więc gdy przycisk nie jest wciśnięty to na wejściu Arduino widoczny jest stan niski (logiczne "0"). Wciśnięcie przycisku spowoduje, że na wejściu będzie stan wysoki (logiczna "1").

W zależności od wersji płytki, rezystory oznaczone "NA" mogą nie być wlotowane, nie ma to jednak wpływu na działania układu.

Aby wykorzystać przyciski należy skonfigurować pin jako wejście za pomocą funkcji `pinMode(nr_pinu, INPUT)`, w której za `nr_pinu` podstawiamy numer pinu, który ma działać w trybie wejścia (ang. *input*). W dalszej części programu stan wejścia odczytujemy za pomocą funkcji `digitalRead(nr_pinu)`.

Program, który po wciśnięciu przycisku podłączonego do pinu D4 (w lewo) włączy diodę LED1:

```
#define LED1 13
#define SW_LEFT 4

// instrukcje z funkcji setup wykonują się tylko raz (po starcie Arduino)
void setup() {
    // konfiguracja pinów
    pinMode(LED1, OUTPUT);
    pinMode(SW_LEFT, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
    if (digitalRead(SW_LEFT) == HIGH) { // jeśli przycisk wciśnięty
        digitalWrite(LED1, HIGH);
    } else {
        digitalWrite(LED1, LOW); // jeśli nie
    }
}
```

Komunikacja z komputerem przez UART

Przewód USB podłączony do Arduino może zasilać płytke, służy do jej programowania, ale może być też używany do komunikacji zestawu z komputerem. W tym celu konieczne jest wykorzystanie interfejsu UART, który należy na początku jednorazowo uruchomić za pomocą instrukcji:

```
Serial.begin(115200);
```

Wartość podana w nawiasie tj. "115200" oznacza wybraną prędkość transmisji, która w tym przypadku została ustawiona na jedną z popularniejszych, standardowych wartości.

Następnie za pomocą poniższego polecenia możliwe jest wysyłanie różnych wartości do komputera. W tym przypadku jest to słowo "TEST":

```
Serial.println("TEST");
```

Aby sprawdzić działanie tego mechanizmu w praktyce można rozbudować wcześniejszy program, który wykorzystywał przycisk i dodać do niego wysyłanie do komputera informacji dotyczących wciśniętego przycisku.

Dodatkowe opóźnienie wprowadzone na końcu programu sprawia,
że pętla wykonuje się tylko co 200 ms, dzięki temu informacje
nie będą wysyłane do komputera zbyt często.

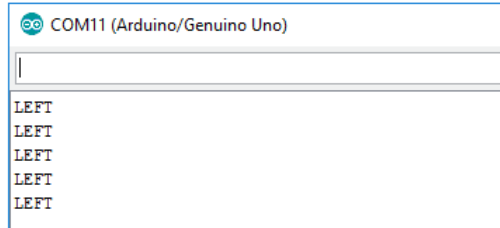
```
#define LED1 13
#define SW_LEFT 4

void setup() {
  // konfiguracja pinów
  pinMode(LED1, OUTPUT);
  pinMode(SW_LEFT, INPUT);
  Serial.begin(115200);
}

void loop() {
  if (digitalRead(SW_LEFT) == HIGH) { // jeśli przycisk wciśnięty
    digitalWrite(LED1, HIGH);
    Serial.println("LEFT");
  } else { // jeśli nie
    digitalWrite(LED1, LOW);
  }

  delay(200);
}
```

Po wgraniu tego programu należy uruchomić *Monitor Portu Szeregowego*, który znaleźć można w menu Arduino IDE pod opcją *Narzędzia*. Od teraz, po wciśnięciu przycisku na płytce (D4) w nowym oknie powinien pojawiać się tekst "LEFT":



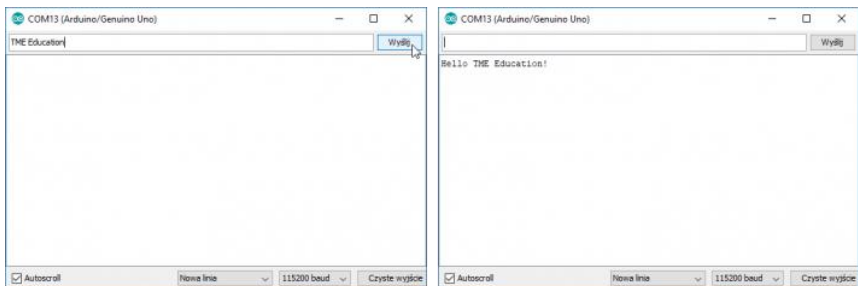
Jeśli zamiast tekstu "LEFT" wyświetlane są losowe, nieczytelne znaki należy się upewnić, że wybrana jest odpowiednia prędkość transmisji w dolnej części okna (w menu rozwijanym) tj.: 115200.

Wysyłanie informacji do Arduino

W podobny sposób można również wysłać informacje do Arduino. Przykładowy program tego typu widoczny jest na poniższym przykładzie. Program sprawdza cały czas, czy Arduino odebrało jakieś dane wysłane z komputera. Jeśli tak jest, to zapisuje je do zmiennej *text* i wyświetla powitanie.

```
String text = "";
void setup() {
  Serial.begin(115200);
}
void loop() {
  if(Serial.available() > 0) { //Czy odebrano jakiś tekst?
    text = Serial.readStringUntil('\n');
    //Odczytaj tekst aż do znaku końca wiersza
    Serial.println("Hello " + text + "!"); //Wypisz odebrany tekst
  }
}
```

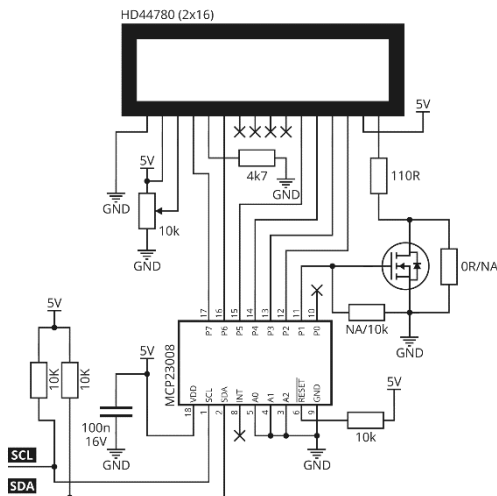
Działanie programu w praktyce:



Wyświetlacz tekstowy LCD 2x16 znaków

Jednym z głównych elementów zestawu edukacyjnego jest tekstowy wyświetlacz LCD, na którym można wyświetlić 2 linie tekstu (po 16 znaków w każdej). Wyświetlacz podłączony jest przez dodatkowy układ pośredniczący, który komunikuje się z Arduino przez cyfrowy interfejs I2C. Dzięki temu do obsługi tego wyświetlacza wykorzystywane są jedynie 2 linie sygnałowe.

W celu uruchomienia wyświetlacza konieczne jest dodanie odpowiednich bibliotek. W tym celu można skorzystać z Menedżera bibliotek wbudowanego w Arduino IDE, który znajduje się w menu: *Szkic > Dołącz bibliotekę > Zarządzaj bibliotekami...*



Do działania Menedżera bibliotek potrzebny jest dostęp do Internetu! Alternatywnie, biblioteki można zaimportować z przygotowanych wcześniej archiwów ZIP. W tym celu należy wykorzystać opcję: *Szkic > Dołącz bibliotekę > Dodaj bibliotekę .ZIP...*

Biblioteki dodajemy w programie za pomocą dyrektywy `#include`:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
```

Następnie, konieczne jest podanie informacji o sposobie połączenia wyświetlacza LCD z ekspanderem. Pierwsza informacja podana do wywoływanej funkcji (`0x20`) to adres ekspandera obsługującego ten wyświetlacz.

```
// konfiguracja połączenia LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008, 7, 6, 5, 4, 3, 2, 1, HIGH);
```

Kolejny krok to konfiguracja wyświetlacza, w której ustawiamy dozwoloną liczbę kolumn (16) oraz wierszy (2). Informacja ta jest ściśle związana z wykorzystanym wyświetlaczem i zawsze powinna być ustawiona w taki sam sposób:

```
void setup() {
    // konfiguracja wyświetlacza LCD
    lcd.begin(16, 2);
}
```

W celu wypisania informacji na wyświetlaczu należy wykorzystać konstrukcję `lcd.setCursor(x, y)`, która ustawi kursor w odpowiedniej lokalizacji. Linie i pozycje w linii liczone są od zera (a nie od 1), więc przykładowo:

- `lcd.setCursor(0, 0)` - zacznij od pierwszego znaku w pierwszej linii,
- `lcd.setCursor(0, 1)` - zacznij od pierwszego znaku w drugiej linii,
- `lcd.setCursor(5, 0)` - zacznij od szóstego znaku w pierwszej linii,
- `lcd.setCursor(5, 1)` - zacznij od szóstego znaku w drugiej linii.

Po ustawieniu kursora można już skorzystać z funkcji `lcd.print("TEXT")`, która wyświetli na LCD wskazany tekst. W celu usunięcia zawartości ekranu można wypisać na nim ciąg spacji lub skorzystać z specjalnej funkcji `lcd.clear()`.

Demo przykładu, który wyświetla tekst na zmianę w dwóch liniach:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja połączenia LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008, 7, 6, 5, 4, 3, 2, 1, HIGH);

void setup() {
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  // ustaw pozycję kursora
  lcd.setCursor(0, 0);
  // wypisz tekst na wyświetlaczu
  lcd.print("TME");
  // ustaw pozycję kursora
  lcd.setCursor(0, 1);
  // wypisz tekst na wyświetlaczu
  lcd.print("TESTER - 2");
  delay(1000);
  lcd.clear();

  // ustaw pozycję kursora
  lcd.setCursor(0, 0);
  // wypisz tekst na wyświetlaczu
  lcd.print("TESTER - 1");
  // ustaw pozycję kursora
  lcd.setCursor(0, 1);
  // wypisz tekst na wyświetlaczu
  lcd.print("TME");

  delay(1000);
  lcd.clear();
}
```

Jeśli na wyświetlaczu mimo wgrania programu nie widać tekstu to konieczna może być regulacja kontrastu, którą można wykonać za pomocą małego, srebrnego potencjometru umieszczonego nad prawym, górnym rogami wyświetlacza. Wystarczy przy wgranym programie i uruchomionym układzie delikatnie kręcić potencjometrem np. za pomocą śrubokręta z płaską końcówką.

Obsługa czujników analogowych

Na płytce znajduje się kilka czujników analogowych, a sposób obsługi wszystkich jest bardzo podobny. Czujniki analogowe **mierzą wielkość fizyczną** (np. temperaturę), a wynik przedstawiają w formie odpowiedniej wartości napięcia np. im cieplej tym na wyjściu czujnika otrzymujemy wyższą wartość napięcia z zakresu zasilania czujnika (czyli w tym przypadku od 0 do 5V).

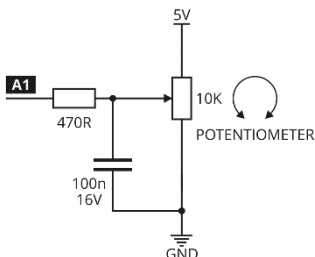
Aby korzystać z wejścia analogowego nie trzeba ich w żaden sposób deklarować. Wystarczy odczytać wartość napięcia, która jest reprezentowana przez liczbę z zakresu od 0 do 1023. Gdzie 0 oznacza ~0V, a 1023 oznacza ~5V. Odczytanie informacji odbywa się za pomocą funkcji `analogRead(nr_pinu)`.

Zakres 0-1023 wynika z tego, że w Arduino znajduje się przetwornik analogowo-cyfrowy (ADC) o rozdzielczości 10 bitów.

Czujniki analogowe - potencjometr (A1)

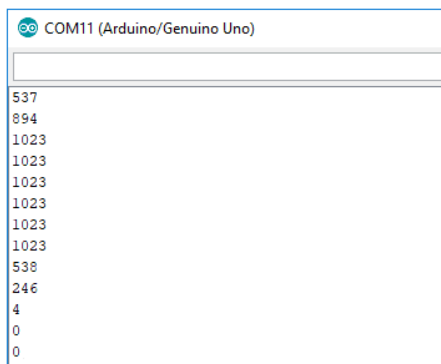
Najprostszym czujnikiem analogowym umieszczonym na płytce jest potencjometr podłączony do pinu A1 (A od analogowy). W tym wypadku wykorzystywana jest możliwość użycia tego elementu w roli dzielnika napięcia.

Poniższy program będzie odczytywał informacje z tego czujnika i wysyłał je do komputera przez UART. Nowe wartości będą wysyłane co 150 ms:



```
void setup() {  
  // konfiguracja UART  
  Serial.begin(115200);  
}  
  
void loop() {  
  // odczytaj wartość czujnika  
  int pot = analogRead(A1);  
  
  // wyślij informację do komputera  
  Serial.println(pot);  
  
  // odczekaj 150 ms  
  delay(150);  
}
```

Po wgraniu programu i uruchomieniu *Monitora Portu Szeregowego* powinien być widoczny ciąg liczb, których wartości zmieniają się podczas ręcznej regulacji potencjometru.



Wygodniejszą formą wyświetlania wartości odczytanych z czujnika analogowego może być LCD. W tym przypadku zawartość górnej linii wypisywana jest tylko raz, na początku programu, a w pętli podmieniana jest wyłącznie zawartość wyświetlana w drugiej linii:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja połączenia LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008, 7, 6, 5, 4, 3, 2, 1, HIGH);

void setup() {
    // konfiguracja LCD
    lcd.begin(16, 2);

    // ustaw pozycję kursora
    lcd.setCursor(0, 0);
    // wypisz tekst na wyświetlaczu
    lcd.print("POT:");
}

void loop() {
    // odczytaj wartość czujnika
    int pot = analogRead(A1);

    // ustaw pozycję kursora
    lcd.setCursor(0, 1);
    // wypisz tekst na wyświetlaczu
    lcd.print(pot);

    // odczekaj 150 ms
    delay(150);
}
```

```
// wyczyść drugi wiersz na wyświetlaczu
lcd.setCursor(0, 1);
lcd.print("   ");
}
```

Po wgraniu tego programu każda zmiana położenia potencjometru powinna być natychmiast widoczna na wyświetlaczu w formie zmieniającej się liczby z zakresu od 0 do 1023.

Czujniki analogowe - czujnik światła (A3)

Kolejnym czujnikiem analogowym umieszczonym na płytce jest **sensor światła KPS-3227**, który domyślnie podłączony jest do wejścia A3. Badając napięcie na wyjściu tego czujnika można zmierzyć ilość światła padającego na płytkę – im jaśniej, tym napięcie powinno być wyższe.

Przykład wyświetlający wartość na wyświetlaczu:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008, 7, 6, 5, 4, 3, 2, 1, HIGH);

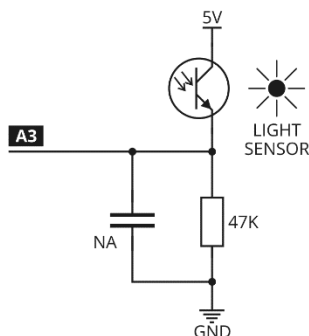
void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("LIGHT:");
}

void loop() {
  int light = analogRead(A3);  // odczytaj wartość czujnika

  // ustaw pozycję kursora
  lcd.setCursor(0, 1);
  // wypisz tekst na wyświetlaczu
  lcd.print(light);

  delay(150);  // odczekaj 150 ms

  // wyczyść drugi wiersz wyświetlacza
  lcd.setCursor(0, 1);
  lcd.print("   ");
}
```



Wskazania czujnika mogą być niestabilne przy niektórych źródłach sztucznego światła (np. przy świetłówkach).

Dla łatwiejszej interpretacji wyników można skorzystać z funkcji `map`, która pozwala na przeskalowanie wartości. Poniższy kod sprawi, że zamiast wartości z zakresu 0-1023 otrzymamy wartość mieszczącą się w zakresie 0-100, bo będzie można interpretować np. jako procentowy poziom oświetlenia:

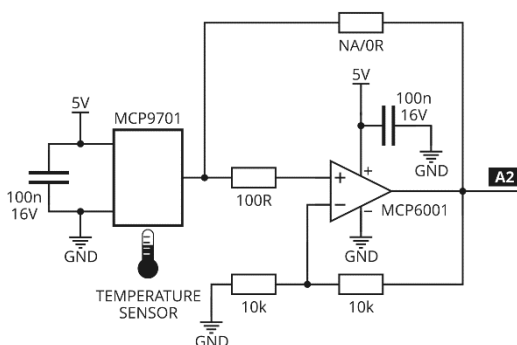
```
lcd.print(map(light, 0, 1023, 0, 100));
```

Lista wszystkich funkcji języka Arduino dostępna jest na stronie:
<https://www.arduino.cc/reference/en/>

Czujniki analogowe - czujnik temperatury (A2)

Na płytce znajduje się analogowy **czujnik temperatury MCP9701** (domyślnie podłączony do A2). Sensor ten może być wykorzystywany identycznie jak czujnik światła. W tym przypadku wzrost napięcia na jego wyjściu będzie oznaczał wzrost temperatury.

Odczyt napięcia na wyjściu czujnika można również przeliczyć na temperaturę w stopniach Celsjusza. W tym celu należy wykorzystać wzór:
 $temp * 0.125 - 22.0$.



Przykładowy program termometru:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008, 7, 6, 5, 4, 3, 2, 1, HIGH);

void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("TEMP [C]:");
}

void loop() {
  int temp = analogRead(A2);           // odczytaj wartość czujnika
  temp = (int)(temp*0.125 - 22.0);     // przelicz wartość na stopnie Celsjusza

  lcd.setCursor(0, 1);
  lcd.print(temp);
}
```

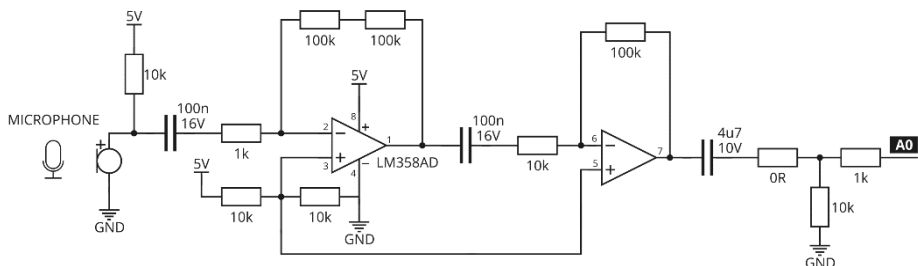
```
delay(200);

lcd.setCursor(0, 1);
lcd.print("    ");
}
```

W polu magnetycznym czujnik temperatury może działać nieprawidłowo.

Czujniki analogowe - mikrofon (A0)

Na płytce znajduje się mikrofon, który został podłączony do Arduino w sposób pozwalający na użycie go w roli czujnika poziomu hałasu. Mikrofon ten można więc wykorzystać np. do włączania diod za pomocą kłaśnięcia lub sygnalizacji zbyt wysokiego poziomu dźwięku w pomieszczeniu.



Poniższy kod to najprostsza forma wykorzystania tego czujnika. Arduino regularnie sprawdza poziom napięcia na wejściu, do którego podłączony jest układ z mikrofonem. Informacja ta jest pokazywana na wyświetlaczu. Gdy zmierzona wartość przekroczy 250 to warunek zostanie spełniony i dioda LED1 zostanie włączona na sekundę. Wartość wyzwalająca włączenie diody została **dobrana doświadczalnie** w taki sposób, aby układ reagował na kłaśnięcie.

```
#define LED1 13

#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008, 7, 6, 5, 4, 3, 2, 1, HIGH);

void setup() {
  pinMode(LED1, OUTPUT);
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("MIC:");
}

void loop() {
  // odczytaj wartość czujnika
  int mic = analogRead(A0);

  lcd.setCursor(0, 1);
```



```

lcd.print(mic);
if (mic > 250) { // jeśli wykryto kłaśnięcie
  // włącz diodę LED
  digitalWrite(LED1, HIGH);
  delay(1000);
} else { // jeśli nie
  // wyłącz diodę LED
  digitalWrite(LED1, LOW);
}

delay(150);

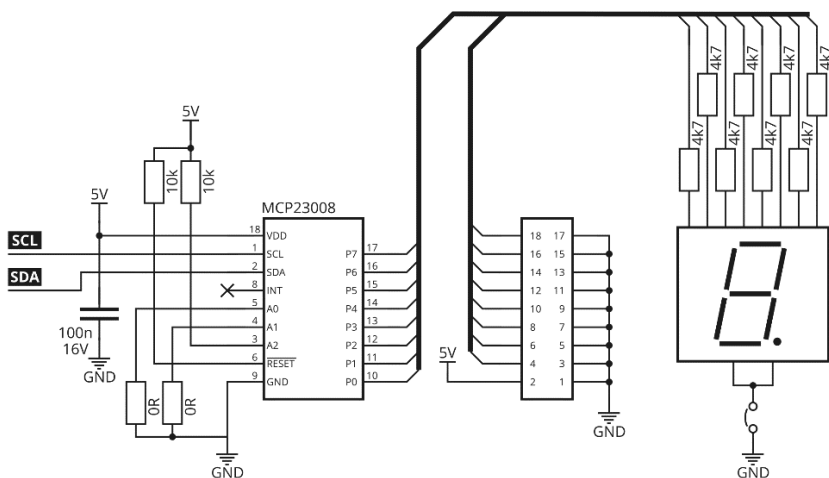
lcd.setCursor(0, 1);
lcd.print("  ");
}

```

Ekspander i wyświetlacz 7-segmentowy

W prawym górnym rogu płytki znajduje się wyświetlacz 7-segmentowy ze wspólną anodą, który został podłączony do Arduino za pomocą ekspandera I2C (adres 0x24). Wykorzystanie różnych adresów pozwala na równoczesne korzystanie z wielu podzespołów komunikujących się przez te same linie danych. Dzięki temu możliwe jest równoczesne używanie np. LCD tekstowego.

Pod wyświetlaczem znajduje się zworka opisana jako "ON/OFF". Za jej pomocą możliwe jest odłączanie wspólnej anody od masy układu co doprowadzi do całkowitego wyłączenia wyświetlacza. Zworka ta jest przydatna w sytuacjach, gdy chcemy wykorzystać piny ekspandera w inny sposób. Można wtedy usunąć zworkę, wyłączyć wyświetlacz i używać piny ekspandera wyprowadzone na podwójne złącze znajdujące się obok wyświetlacza.



Dokumentacja zestawu edukacyjnego

Wykorzystanie ekspandera należy zacząć od dodania biblioteki oraz zadeklarowania nowego obiektu (tutaj nazwanego jako `seg7`), który będzie pozwalał sterować poszczególnymi pinami:

```
#include "Adafruit_MCP23008.h"
Adafruit_MCP23008 seg7;
```

Inicjalizujemy ekspander i ustawiamy wszystkie jego piny w roli wejść. Nazwy funkcji oraz sposób ich wykorzystania są zbliżone do funkcji używanych do sterowania pinami wbudowanych w Arduino:

```
seg7.begin(0x4);

seg7.pinMode(0, OUTPUT);
seg7.pinMode(1, OUTPUT);
seg7.pinMode(2, OUTPUT);
seg7.pinMode(3, OUTPUT);
seg7.pinMode(4, OUTPUT);
seg7.pinMode(5, OUTPUT);
seg7.pinMode(6, OUTPUT);
seg7.pinMode(7, OUTPUT);
```

Adres ekspandera odpowiadającego za wyświetlacz 7-segmentowy to 0x24, jednak użyta tu biblioteka wymaga jedynie podania w roli adresu wartości offsetu ponad adres 0x20. W związku z tym do funkcji podajemy jedynie adres w **formie 0x4**. (Metoda ta jest jednak specyficzna wyłącznie dla tej biblioteki, więcej informacji na ten temat można znaleźć w dokumentacji biblioteki `Adafruit_MCP23008`.)

Od teraz, po konfiguracji wyświetlacza, możliwe będzie sterowanie wszystkimi diodami, które są jego częścią. Włączenie diody wygląda następująco: `seg7.digitalWrite(nr_pinu, HIGH)`. Przykład programu migającego dwoma elementami wyświetlacza:

```
#include "Adafruit_MCP23008.h"
Adafruit_MCP23008 seg7;

void setup() {
  // konfiguracja pinów ekspandera
  seg7.begin(0x4);

  seg7.pinMode(0, OUTPUT);
  seg7.pinMode(1, OUTPUT);
  seg7.pinMode(2, OUTPUT);
  seg7.pinMode(3, OUTPUT);
  seg7.pinMode(4, OUTPUT);
  seg7.pinMode(5, OUTPUT);
  seg7.pinMode(6, OUTPUT);
  seg7.pinMode(7, OUTPUT);
}

void loop() {
  seg7.digitalWrite(5, HIGH);
  seg7.digitalWrite(2, LOW);
  delay(250);
```

```

seg7.digitalWrite(5, LOW);
seg7.digitalWrite(2, HIGH);
delay(250);
}

```

Ekspander i wyświetlacz 7-segmentowy (cyfry)

Najczęściej na wyświetlaczach tego typu wyświetla się cyfry od 0 do 9. Aby ułatwić sobie ustawianie takich wartości warto do programu dodać tablice z 10 elementami. Każdy element tablicy to liczba zapisana binarnie, która reprezentuje informacje na temat włączenia (1) lub wyłączenia (0) każdej diody.

```

uint8_t digit[10] = {
  B00111111, // "0"
  B00000110, // "1"
  B01011011, // "2"
  B01001111, // "3"
  B01100110, // "4"
  B01101101, // "5"
  B01111101, // "6"
  B00000111, // "7"
  B01111111, // "8"
  B01101111, // "9"
};

```

Dzięki temu korzystając z nowej funkcji `seg7.writeGPIO()` będzie możliwe wyświetlenie dowolnej cyfry za pomocą jednej linii kodu. Poniższy program demonstracyjny wyświetla wszystkie wartości (0-9) w pętli:

```

#include "Adafruit_MCP23008.h"
Adafruit_MCP23008 seg7;

uint8_t digit[10] = {
  B00111111, // "0"
  B00000110, // "1"
  B01011011, // "2"
  B01001111, // "3"
  B01100110, // "4"
  B01101101, // "5"
  B01111101, // "6"
  B00000111, // "7"
  B01111111, // "8"
  B01101111, // "9"
};

void setup() {
  // konfiguracja pinów ekspandera
  seg7.begin(0x4);
  seg7.pinMode(0, OUTPUT);
  seg7.pinMode(1, OUTPUT);
  seg7.pinMode(2, OUTPUT);
}

```

```
seg7.pinMode(3, OUTPUT);
seg7.pinMode(4, OUTPUT);
seg7.pinMode(5, OUTPUT);
seg7.pinMode(6, OUTPUT);
seg7.pinMode(7, OUTPUT);
}

void loop() {
  for (int i=0; i<10; i++) {
    seg7.writeGPIO(digit[i]);
    delay(250);
  }
}
```

Pętla *for*

W powyższym programie została wykorzystana pętla *for*. Dzięki użyciu pętli możliwe jest wielokrotne wykonanie tych samych instrukcji (bez konieczności ich przepisywania) oraz skrócenie zapisu. W tym przykładzie pętla wykona się 10 razy (od 0 do 9). Za każdym razem na wyświetlaczu zostanie pokazana wartość z tablicy *digit*, która będzie odpowiadała numerowi aktualnego obiegu pętli. Jest to możliwe, ponieważ wartość zmiennej *i* będzie inkrementowana (zwiększana) za każdym razem, gdy wykonane zostaną wszystkie instrukcje znajdujące się w obrębie pętli *for*. Gdy pętla wykona się określoną liczbę razy to program przejdzie do kolejnych instrukcji (w tym wypadku wróci na początek programu i od nowa rozpocznie wykonywanie instrukcji w pętli *for*).

Pętla *while*

Drugą, również często wykorzystywaną pętlą w Arduino jest pętla *while*, która wykonuje się tak długo, jak długo spełniony będzie warunek podany w nawiasie. Wcześniejszą pętlę *for* można zastąpić nową pętlą *while* w następujący sposób:

```
int i = 0;
while(i < 10){
  seg7.writeGPIO(digit[i]);
  delay(250);
  i++;
}
```

Na początku tworzona jest zmienna licznikowa (tutaj jest to *i*). W nawiasie zaraz za *while* podany jest warunek, który sprawia, że pętla działa tak długo, jak długo wartość zmiennej *i* jest mniejsza od 10. Wewnątrz pętli znajdują się instrukcje, które mają wykonywać się wielokrotnie. Na końcu pętli dodana jest inkrementacja (*i++*), czyli zwiększanie wartości zmiennej o 1. Dzięki temu po 10 razach warunek w pętli *while* nie będzie spełniony i program wyjdzie z pętli.

Pętle tego typu wykorzystuje się również do stworzenia pętli nieskończonej – wtedy nie ma konieczności wpisywania tam żadnego warunku, wystarczy wpisać *1*, co zawsze będzie oznaczało prawdę:

```
while(1){
  //pętla nieskończona
}
```

Wyświetlacz graficzny OLED

Kolejnym elementem podłączonym do interfejsu I2C jest wyświetlacz graficzny OLED (adres 0x3C), który znajduje się obok diod. Użyty na płycie wyświetlacz posiada **popularny sterownik SSD1306**.

W celu wykorzystania wyświetlacza należy dodać do projektu potrzebną bibliotekę:

```
#include <Adafruit_SSD1306.h>
```

I utworzyć obiekt wyświetlacza:

```
Adafruit_SSD1306 display(NULL);
```

Następnie wewnątrz funkcji setup konieczna jest konfiguracja OLEDa, która polega na jego inicjalizacji (z podaniem adresu 0x3C), odczekaniu krótkiego czasu na inicjalizację sterownika, wyczyszczeniu zawartość wyświetlacza i ustawieniu koloru na biały (jedyń możliwy):

```
display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
delay(500);
display.clearDisplay();
display.setTextColor(WHITE);
```

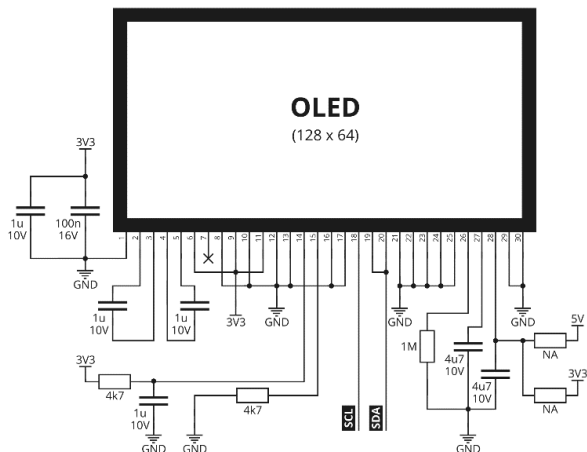
Wyświetlenia tekstu na ekranie polega na wybraniu pozycji kursora (ustawianej w pikselach), wpisaniu tekstu, a następnie "wysłaniu" wszystkich informacji na ekran.

Poniższy program najpierw wyświetla dwie testowe linie (jedna pod drugą z przesunięciem o 50 pikseli w prawo). Następnie, w pętli, po ekranie przesuwają po skosie napis "TEST LINE 12345":

```
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(NULL);

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);
  display.clearDisplay();
```



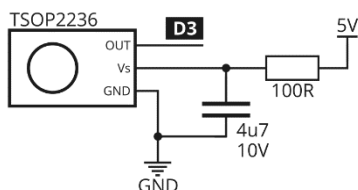
```
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("TEST LINE 1");
display.setCursor(50,10);
display.println("TEST LINE 2");
display.display();

delay(2000);
}

void loop() {
  for (int i = 0; i < 8; i++) {
    display.clearDisplay();
    display.setCursor(5*i,8*i);
    display.println("TEST LINE 12345");
    display.display();
    delay(250);
  }
}
```

Odbiornik podczerwieni (D3)

Płytką wyposażoną jest w **scalony odbiornik podczerwieni TSOP2236**, który może być używany do odbierania informacji wysyłanych przez piloty IR działające w standardzie **RC5**. Taki sposób komunikacji został opracowany przez firmę Philips ponad 30 lat temu do zdalnego kontrolowania sprzętu RTV i cały czas cieszy się dużą popularnością.



Pilot nadający w standardzie RC5 wysyła wiązkę podczerwieni o częstotliwości 36 kHz. Za każdym razem przesyłane jest **14 bitów danych**, które tworzą ramkę danych. Jeśli przycisk na pilocie jest wciśnięty, to ramki danych przesyłane są cały czas (co około 114 ms).

Wykorzystanie czujnika podłączonego do pinu D3 należy rozpocząć od dodania biblioteki, utworzenia obiektu oraz zadeklarowania nowej struktury do przechowywania odebranych informacji:

```
#include <IRremote.h>

IRrecv irrecv(3);
decode_results results;
```

Przykładowy program demonstracyjny, który dekoduje sygnały z pilota i wysyła je do komputera przez UART widoczny jest poniżej:

```
#include <IRremote.h>

IRrecv irrecv(3);
decode_results results;
```

```

void setup() {
  Serial.begin(115200);
  irrecv.enableIRIn(); // uruchom odbiornik IR
}

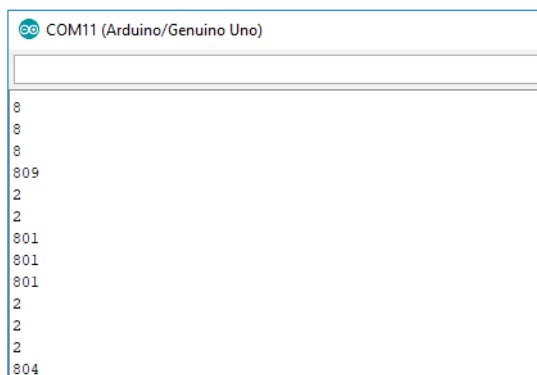
void loop() {
  if (irrecv.decode(&results)) { // jeśli wykryto dane
    // wyślij dane przez UART do komputera w formacie HEX
    Serial.println(results.value, HEX);

    irrecv.resume(); // wznów odbieranie
  }

  delay(100);
}

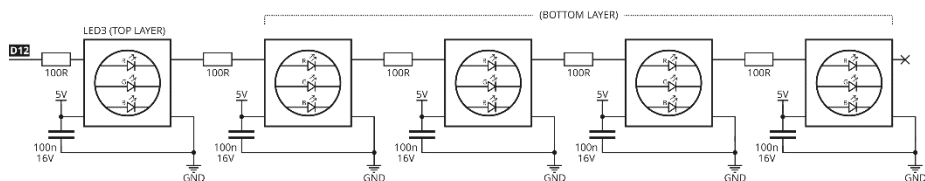
```

W efekcie jego działania w *Monitorze Portu Szeregowego* powinny pojawiać się zdekodowane kody, które wysyłane są przez pilota IR. Kody pilotów mogą różnić się w zależności od konkretnego modelu. Niektóre z pilotów wysyłają również inne kody w zależności od tego czy dany przycisk wciskany jest pierwszy lub kolejny raz.



Obsługa diod RGB sterowanych cyfrowo (D12)

Na płytce znajduje się 5 diod RGB sterowanych cyfrowo. Wszystkie te diody połączone są szeregowo do jednego wyprowadzenia Arduino (D12). Diody **WS2812B** posiadają wbudowany kontroler, dzięki czemu każdej diodzie można przypisać inny kolor.



Diody numerowane są od 0 do 4. Dioda numer 0 znajduje się obok wyświetlacza OLED, pozostałe 4 LEDy zostały umieszczone po drugiej stronie płytki (w narożnikach), dzięki czemu mogą oświetlać płytkę od spodu. Uruchomienie diod należy rozpocząć od dodania biblioteki i skonfigurowania obiektu, który zawiera informacje na ich temat.

Poniższy zapis oznacza deklarację 5 diod podłączonych do pinu nr 12:

```
#include <Adafruit_NeoPixel.h>
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(5, 12, NEO_GRB + NEO_KHZ800);
```

W celu włączenia każdego koloru diody numer 0 należy wykorzystać funkcję *setPixelColor*:

```
pixels.begin();

pixels.setPixelColor(0, pixels.Color(10,20,30));
pixels.show();
```

Zapis ten oznacza, że na pierwszej diodzie w szeregu (nr 0) ustawiony zostanie kolor czerwony z intensywnością 10/255, kolor zielony z intensywnością 20/255 oraz kolor niebieski z intensywnością 30/255. Po przypisaniu kolorów wystarczy wywołać funkcję *show*, która prześle informacje do diod.

Poniższy program testowy uruchamia wszystkie kolory diody znajdującej się obok wyświetlacza OLED, a następnie miga naprzemiennie 4 diodami na spodzie płytki (na niebiesko i czerwono):

```
#include <Adafruit_NeoPixel.h>
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(5, 12, NEO_GRB + NEO_KHZ800);

void setup() {
  pixels.begin();
  pixels.setPixelColor(0, pixels.Color(10,20,30));
  pixels.show();
}

void loop() {
  pixels.setPixelColor(1, pixels.Color(0,0,255));
  pixels.setPixelColor(2, pixels.Color(255,0,0));
  pixels.setPixelColor(3, pixels.Color(0,0,255));
  pixels.setPixelColor(4, pixels.Color(255,0,0));
  pixels.show();
  delay(250);
  pixels.setPixelColor(1, pixels.Color(255,0,0));
  pixels.setPixelColor(2, pixels.Color(0,0,255));
  pixels.setPixelColor(3, pixels.Color(255,0,0));
  pixels.setPixelColor(4, pixels.Color(0,0,255));
  pixels.show();
  delay(250);
}
```

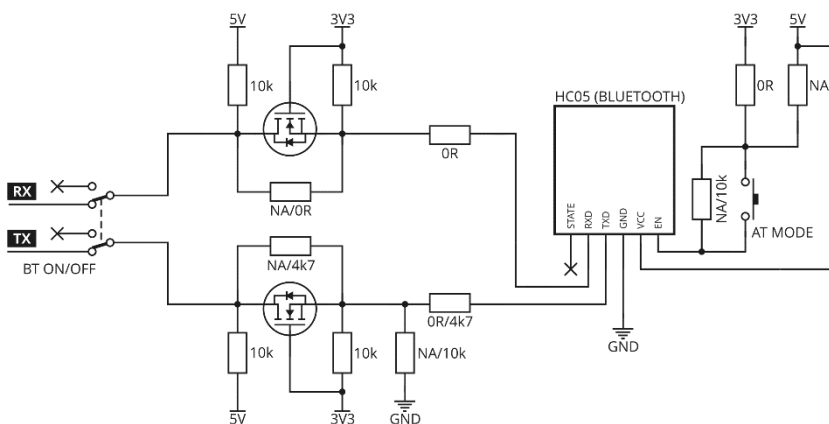
W przypadku problemu z uruchomieniem diod należy się upewnić, że po ustawieniu kolorów wywołana została funkcja *show*, która odpowiada za przesłanie informacji do sterowników.

Moduł Bluetooth

Płytkę została wyposażone w miejsce do wpięcia opcjonalnego modułu Bluetooth np. HC-05. Kątowe, 6 pinowe złącze do jego podłączenia znajduje się na prawo od listwy zworek. Wyprowadzone zostały tam następujące sygnały:

- zasilanie dla modułu BT,
- 2 piny kontrolujące pracę modułu,
- 2 piny do transmisji danych (UART).

Połączenie zostało zrealizowane przez konwerter poziomów logicznych:



Moduł Bluetooth można wykorzystywać np. do komunikacji zestawu z telefonami pracującymi pod kontrolą systemu Android. Komunikacja odbywa się przez interfejs UART - ten sam, który używany jest do komunikacji z komputerem PC. W związku z tym możliwe jest jednoczesne używanie tylko jednej z tych dwóch form komunikacji.

W celu sprawdzenia działania modułu można wykorzystać poniższy, testowy kod. Jego działanie polega na wysłaniu tekstu "TME BT" (co 250 ms). Równocześnie układ odbiera wszystkie dane wysłane do niego z telefonu. Jeśli odebrany zostanie ciąg znaków "TEST" to układ odpowie na to odsyłając "TME BT TEST OK" i wstrzyma wysyłanie nowych informacji na 1 sekundę.

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

String data = "";
void loop() {
    if(Serial.available() > 0) { // jeśli odebrano dane

        // odczytaj tekst aż do znaku końca linii
        data = Serial.readStringUntil('\n');
```

```
// jeśli odczytano "TEST"
if (data == "TEST") {
    // odeślij tekst "TME BT TEST OK"
    Serial.println("TME BT TEST OK");
    delay(1000);
}

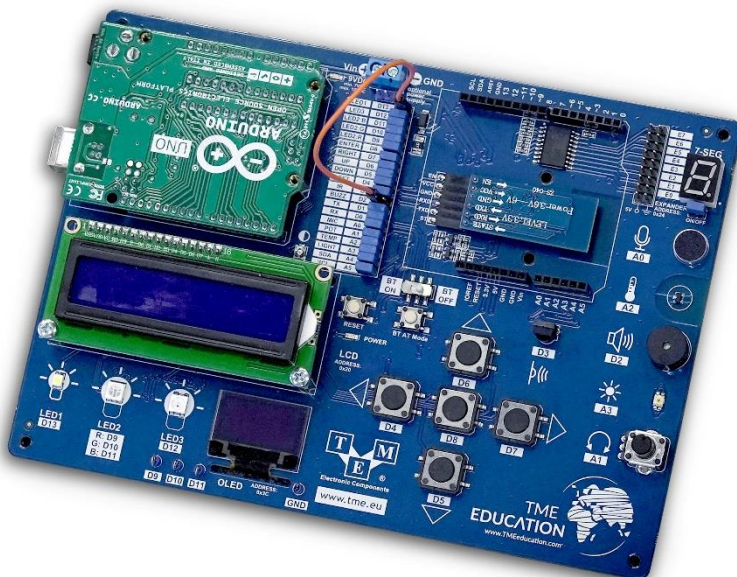
Serial.println("TME BT");
delay(250);
}
```

Po wgraniu programu, przełącznik suwakowy należy przełączyć w pozycję BT ON. Zestaw z modulem BT powinien być wykrywany przez telefon jako nowe urządzenie Bluetooth, z którym można się sparować używając domyślnego kodu pin: "1234". Po sparowaniu w aplikacji na telefonie powinny być widoczne komunikaty wysyłane przez płytę.

Zworki konfiguracyjne

Wszystkie peryferia znajdujące się na płycie są domyślnie połączone z Arduino. Możliwa jest jednak bardzo łatwa zmiana konfiguracji. Zdejmując zworki można **odłączyć peryferia** od sterownika. Jeśli zajdzie taka potrzeba to za pomocą dodatkowego przewodu połączeniowego (brak w zestawie) można **zmienić przypisanie peryferiów** do konkretnych pinów.

Poniższy przykład pokazuje w jaki sposób można odłączyć buzzer i LED1 oraz zmienić konfigurację w taki sposób, aby buzzer sterowany był przez pin D13 (zamiast D2).



Zmiana konfiguracji połączeń za pomocą zwerek może być szczególnie wygodna, gdy zajdzie potrzeba podłączenia danego peryferium zestawu pod inny pin np.: podłączenie diody LED1 do pinu, na którym możliwe jest sprzętowe generowanie sygnału PWM.

Złącza do shieldów Arduino

W prawym górnym rogu płytki znajdują się gniazda pozwalające na podłączenie płytek rozszerzających w standardzie Arduino Shield. Należy jednak pamiętać, aby **przed podłączeniem dodatkowego modułu rozłączyć za pomocą zwerek wszystkie piny, które są używane przez nakładkę!**

W skrajnej sytuacji, konflikt używanych sygnałów może doprowadzić do uszkodzenia Arduino, zestawu edukacyjnego lub dodatkowego modułu.

Opis funkcji Arduino

Poniżej zamieszczona została skrócona lista funkcji wraz z krótkim opisem, pełną listę funkcji i ich dokładny opis znajdziesz na stronie: <https://www.arduino.cc/reference/en/>.

digitalRead(*pin*) – sprawdzenie stanu wejścia cyfrowego.

- *pin* – wyprowadzenie, na którym ma być sprawdzony stan
- wartość zwracana: HIGH (dla stanu wysokiego) lub LOW (dla stanu niskiego)

digitalWrite(*pin*, *state*) – ustawienie stanu wyjścia cyfrowego.

- *pin* – wyprowadzenie, na którym ma być ustawiony stan
- *state* – stan, który ma być ustawiony (LOW lub HIGH)
- wartość zwracana: brak

pinMode(*pin*, *mode*) – konfiguracja trybu pracy pinu cyfrowego.

- *pin* – wyprowadzenie, które ma zostać skonfigurowane
- *mode* – typ pracy wyprowadzenia:
 - INPUT – wejście cyfrowe
 - INPUT_PULLUP – wejście z włączonym rezystorem podciągającym
 - OUTPUT – wyjście cyfrowe
- wartość zwracana: brak

analogRead(*pin*) – pomiar napięcia za pomocą przetwornika ADC.

- *pin* – wyprowadzenie analogowego (A0-A5), na którym ma być dokonany pomiar
- wartość zwracana: liczba z zakresu od 0 do 1023 (im wyższe napięcie na wejściu, tym wyższa wartość zwrócona przez funkcję)

analogWrite(*pin*, *value*) – ustawienie wypełnienia sygnału PWM na określonym wyprowadzeniu.

- *pin* – wyprowadzenie, na którym ma zostać wygenerowany sygnał PWM
- *value* – wartość wypełnienia z zakresu od 0 do 255, gdzie 0 oznacza 0% wypełnienia sygnału PWM, a 255 oznacza 100% wypełnienia sygnału PWM.
- wartość zwracana: brak

delay(*ms*) – funkcja wstrzymująca działanie programu na określną liczbę milisekund (1 sekunda to 1000 milisekund). Działanie funkcji całkowicie wstrzymuje działanie programu!

- *ms* – czas trwania opóźnienia/wstrzymania programu w milisekundach, maksymalna wartość to 4 294 967 295 (rozmiar zmiennej typu unsigned long).
- wartość zwracana: brak

delayMicroseconds(*us*) – funkcja wstrzymująca działanie programu na określną liczbę mikrosekund (1 milisekunda to 1000 mikrosekund). Działanie funkcji całkowicie wstrzymuje działanie programu!

- *us* – czas trwania opóźnienia/wstrzymania programu w mikrosekundach, maksymalna wartość to 16 383.
- wartość zwracana: brak

abs(x) – funkcja zwracająca wartość bezwzględną, dla poprawnego działania funkcji powinno unikać się wykorzystywania innych funkcji wewnątrz argumentów.

- x – liczba, której wartość bezwzględna ma zostać policzona
- wartość zwracana: wartość bezwzględna z liczby x

constrain(x, a, b) – funkcja sprawdzająca, czy liczba znajduje się w podanym zakresie.

- x – testowana liczba (dowolnego typu)
- a – dolna wartość zakresu
- b – górna wartość zakresu
- wartość zwracana:
 - x – jeśli liczba jest większa od a i mniejsza od b .
 - a – jeśli liczba x jest mniejsza od a
 - b – jeśli liczba x jest większa od b

map(value, fromLow, fromHigh, toLow, toHigh) – funkcja skalująca liczbę z jednego zakresu na drugi zakres. Przydatna np. do przeskalowania wartości zmierzonej przez przetwornik ADC (0...1023) na wartość procentową (0...100%).

- $value$ – wartość, która ma zostać przeskalowana
- $fromLow$ – dolny zakres aktualnego zakresu
- $fromHigh$ – górny zakres aktualnego zakresu
- $toLow$ – dolny zakres przyszłego zakresu
- $toHigh$ – górny zakres przyszłego zakresu
- wartość zwracana: przeskalowana liczba

max(x, y) – funkcja porównująca dwie liczby.

- x – pierwsza liczba
- y – druga liczba
- Wartość zwracana: większa z liczb

min(x, y) – funkcja porównująca dwie liczby.

- x – pierwsza liczba
- y – druga liczba
- wartość zwracana: mniejsza z liczb

pow(base, exponent) – funkcja obliczająca potęgę.

- $base$ – liczba, która ma być podniesiona do potęgi
- $exponent$ – wartość potęgi
- wartość zwracana: liczba $base$ podniesiona do potęgi $exponent$

sqrt(x) – funkcja obliczająca pierwiastek kwadratowy.

- x – liczba
- wartość zwracana: pierwiastek kwadratowy z liczby x

sq(x) – funkcja obliczająca potęgę kwadratową.

- x – liczba
- wartość zwracana: x podniesione do 2 potęgi

cos(rad) – funkcja obliczająca cosinus kąta.

- *rad* – kąt w Radianach
- wartość zwracana: cosinus podanego kąta

sin(rad) – funkcja obliczająca sinus kąta.

- *rad* – kąt w Radianach
- wartość zwracana: sinus podanego kąta

tan(rad) – funkcja obliczająca tangens kąta.

- *rad* – kąt w Radianach
- wartość zwracana: tangens podanego kąta

millis() – funkcja zwracająca liczbę milisekund od uruchomienia Arduino. Wartość zeruje się po około 50 dniach.

- argumenty funkcji: brak
- wartość zwracana: czas od uruchomienia w milisekundach (zmienna typu *unsigned long*)

pulseIn(*pin*, *value*, *timeout*) – funkcja obliczając czas trwania impulsu na danym wejściu. Np. pomiar startuje w momencie zmiany stanu niskiego na wysoki i zatrzymuje się po ponownym pojawieniu się stanu niskiego. Funkcja działa poprawnie dla impulsów o długości od 10 mikrosekund do 3 minut.

- *pin* – wyprowadzenie, na którym ma być mierzony impuls
- *value* – typ impulsu, który ma być mierzony (LOW lub HIGH)
- *timeout* – opcjonalna liczba mikrosekund, po której układ zaprzestanie pomiaru, jeśli stan na pinie nie ulegnie wcześniej zmianie
- wartość zwracana: *duration* – czas trwania impulsu w mikrosekundach lub 0 w przypadku braku pomiaru

isAlpha(*char*) – funkcja sprawdzająca czy podany znak jest literą.

- *char* – znak, który ma być sprawdzany
- wartość zwracana: prawda, jeśli znak jest literą

isAlphaNumeric(*char*) – funkcja sprawdzająca czy podany znak jest literą lub cyfrą.

- *char* – znak, który ma być sprawdzany
- wartość zwracana: prawda, jeśli znak jest literą lub cyfrą

isDigit(*char*) – funkcja sprawdzająca czy podany znak jest cyfrą.

- *char* – znak, który ma być sprawdzany
- wartość zwracana: prawda, jeśli znak jest cyfrą

random(*min*, *max*) – funkcja zwracająca wartość pseudolosową. Wykorzystanie funkcji powinno być poprzedzone jednorazowym wywołaniem funkcji `randomSeed()`.

- *min* – wartość początkowa zakresu
- *max* – wartość końcowa zakresu
- wartość zwracana: liczba pseudo-losowa z zakresu od *min* do (*max*-1)

randomSeed(seed) – funkcji inicjalizująca generator liczb pseudolosowych. Bez poprawnego wykorzystania tej funkcji wartości zwracane za pomocą funkcji random będą identyczne po każdym uruchomieniu Arduino.

- seed – wartość liczbowa, która inicjalizuje generator liczb pseudolosowych.
- wartość zwracana: brak
- Przykładowe wykorzystanie: *randomSeed(analogRead(0))* jako argument do funkcji zostaje podana wartość ADC odczytana z pinu A0, który może być wykorzystywany przez czujniki podłączone do układu lub może być niewykorzystywany. Losowy odczyt ADC sprawia, że po każdym uruchomieniu Arduino generator liczb pseudolosowych potrafi zwracać różne wartości w kolejnych losowaniach.

byte(x) – funkcja konwertująca zmienną x.

- x – zmienna, która ma zostać skonwertowana
- wartość zwracana: zmienna typu byte

char(x) – funkcja konwertująca zmienną x.

- x – zmienna, która ma zostać skonwertowana
- wartość zwracana: zmienna typu char

int(x) – funkcja konwertująca zmienną x.

- x – zmienna, która ma zostać skonwertowana
- wartość zwracana: zmienna typu int

long(x) – funkcja konwertująca zmienną x.

- x – zmienna, która ma zostać skonwertowana
- wartość zwracana: zmienna typu long

for (inicjalizacja; warunek; działanie) { } – pętla *for* jest wykorzystywana do wielokrotnego wykonywania instrukcji zawartych między nawiasami klamrowymi funkcji; inicjalizacja wykonywana jest tylko raz i najczęściej wykorzystuje się ją do tego, aby utworzyć zmienną licznikową używaną wewnątrz pętli (np. "*int i = 0*"); następnie umieszczany jest warunek określający jak długo pętla ma być wykonywana (np. "*i < 10*"); ostatni element to informacja mówiąca o tym co ma się dzieć ze zmienną licznikową po każdym obiegu pętli (np. "*i++*").

while(warunek) { } – instrukcje zawarte w pętli *while* będą wykonywane w nieskończoność do czasu, gdy warunek znajdujący się w nawiasach jest prawdziwy; testowanym warunkiem może być wartość liczbowa lub np. stan podłączonego przycisku lub czujnika.

Operatory:

- % dzielenie modulo
- * mnożenie
- + dodawanie
- - odejmowanie
- / dzielenie
- = przypisanie
- != różny
- < mniejszy
- <= mniejszy lub równy
- > większy
- >= większy lub równy
- == równy
- ! zaprzeczenie
- && logiczne *i* (logical AND)
- || logiczne *lub* (logical OR)

Lista bibliotek wraz z licencjami

Extensible hd44780 LCD library – biblioteka do obsługi wyświetlacza tekstowego

Strona projektu: <https://github.com/duinoWitchery/hd44780>

Autor: Bill Perry

Licencja: GPL

Adafruit_SSD1306 – biblioteka do obsługi wyświetlacza graficznego

Strona projektu: https://github.com/adafruit/Adafruit_SSD1306

Autor: firma Adafruit

Licencja: BSD

Adafruit_NeoPixel – biblioteka do obsługi cyfrowych diod RGB

Strona projektu: https://github.com/adafruit/Adafruit_NeoPixel

Autor: firma Adafruit

Licencja: GPL

Arduino RC5 remote control decoder library – biblioteka do obsługi kodów RC5

Strona projektu: <https://github.com/guyc/RC5>

Autor: Guy Carpenter

Licencja: BSD

Adafruit-MCP23008-library – biblioteka do obsługi ekspander MCP23008

Strona projektu: <https://github.com/adafruit/Adafruit-MCP23008-library>

Autor: firma Adafruit

Licencja: BSD