



Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja
do pracowni specjalistycznej z przedmiotu

Programowanie C

Kod przedmiotu: **CP1S01005**

(studia stacjonarne)

JĘZYK C - PODSTAWY PROGRAMOWANIA Z WYKORZYSTANIEM PLATFORMY ARDUINO

Numer ćwiczenia

PRC_08

Autor:
dr inż. Jarosław Forenc

Białystok 2024

Spis treści

1. Opis stanowiska	3
1.1. Stosowana aparatura	3
1.2. Oprogramowanie.....	3
2. Wiadomości teoretyczne.....	3
2.1. Instalacja rozszerzenia PlatformIO w edytorze Visual Studio Code	3
2.2. Platforma Arduino.....	4
2.3. Stworzenie nowego projektu, kompilacja i uruchomienie programu	7
2.4. Sterowanie diodą LED.....	10
2.5. Zastosowanie przycisku do sterowania układem	11
3. Przebieg ćwiczenia.....	14
4. Literatura.....	15
5. Pytania kontrolne	15
6. Wymagania BHP.....	15

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

© Wydział Elektryczny, Politechnika Białostocka, 2024 (wersja 1.1)

Wszelkie prawa zastrzeżone. Żadna część tej publikacji nie może być kopiowana i odtwarzana w jakiegokolwiek formie i przy użyciu jakichkolwiek środków bez zgody posiadacza praw autorskich.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows 10/11 oraz platforma Arduino wraz z zestawem czujników.

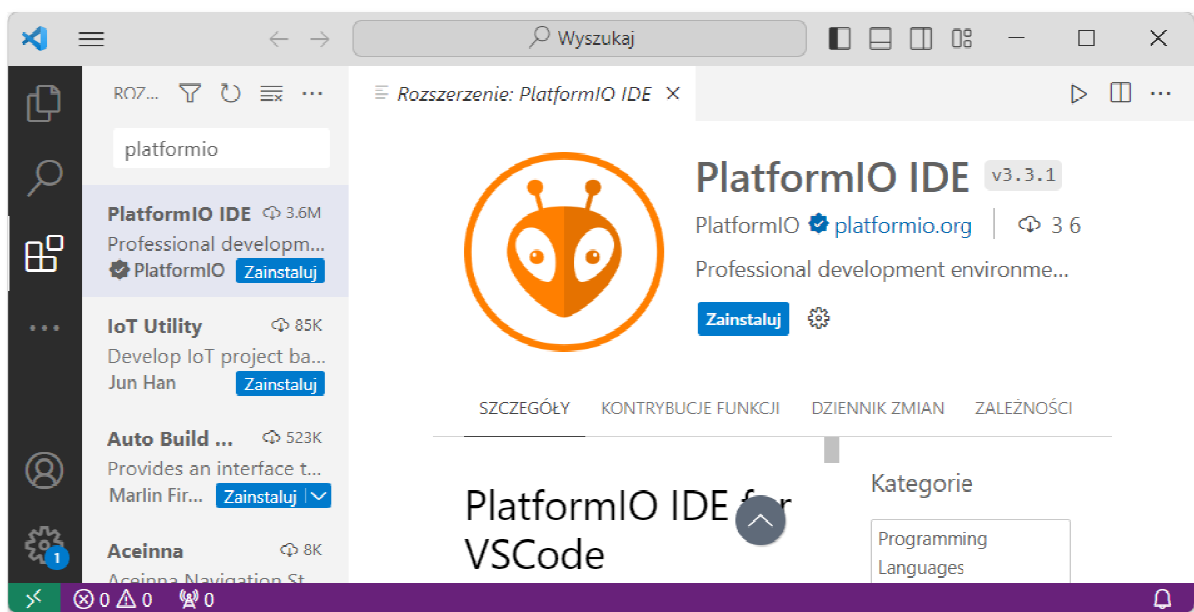
1.2. Oprogramowanie

Na komputerach zainstalowany jest edytor kodu źródłowego Visual Studio Code 1.92 (lub nowszy) wraz z rozszerzeniem (PlatformIO IDE for VSCode).

2. Wiadomości teoretyczne

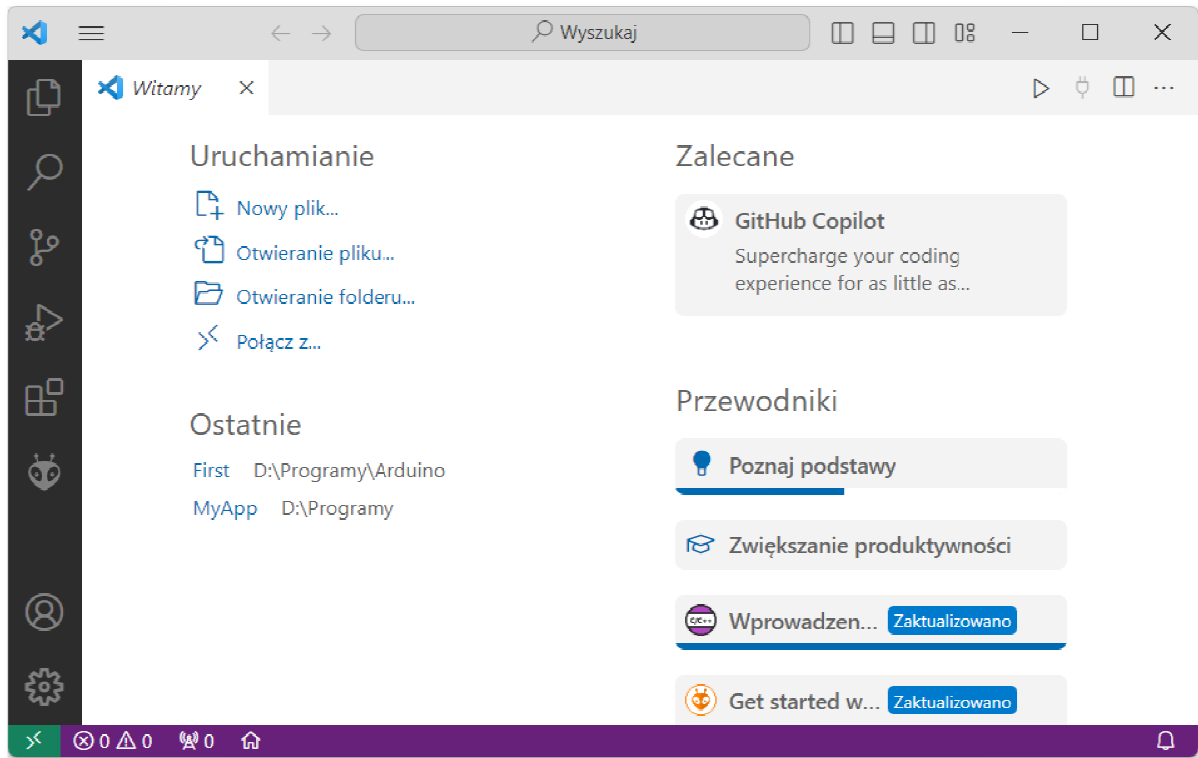
2.1. Instalacja rozszerzenia PlatformIO w edytorze Visual Studio Code

Zastosowanie edytora Visual Studio Code do pisania i uruchamiania programów w języku C na platformie Arduino wymaga zainstalowania rozszerzenia **PlatformIO**. W tym celu przechodzimy do rozszerzeń (klawisz skrót **Ctrl + Shift + X**), a następnie wyszukujemy i instalujemy rozszerzenie **PlatformIO IDE** (Rys. 1).



Rys. 1. Instalacja rozszerzenia PlatformIO

Po zainstalowaniu rozszerzenia **PlatformIO** należy ponownie uruchomić Visual Studio Code. Na podstawowym pasku bocznym (znajdującym się domyślnie przy lewej krawędzi okna programu) pojawi się dodatkowa ikonka: 🐼 (Rys. 2).

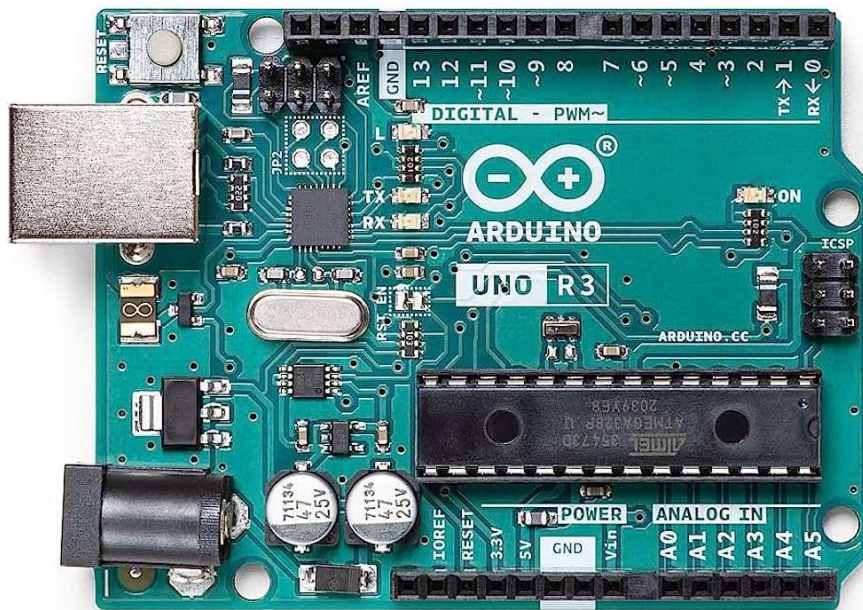


Rys. 2. Visual Studio Code z zainstalowanym rozszerzeniem **PlatformIO**

2.2. Platforma Arduino

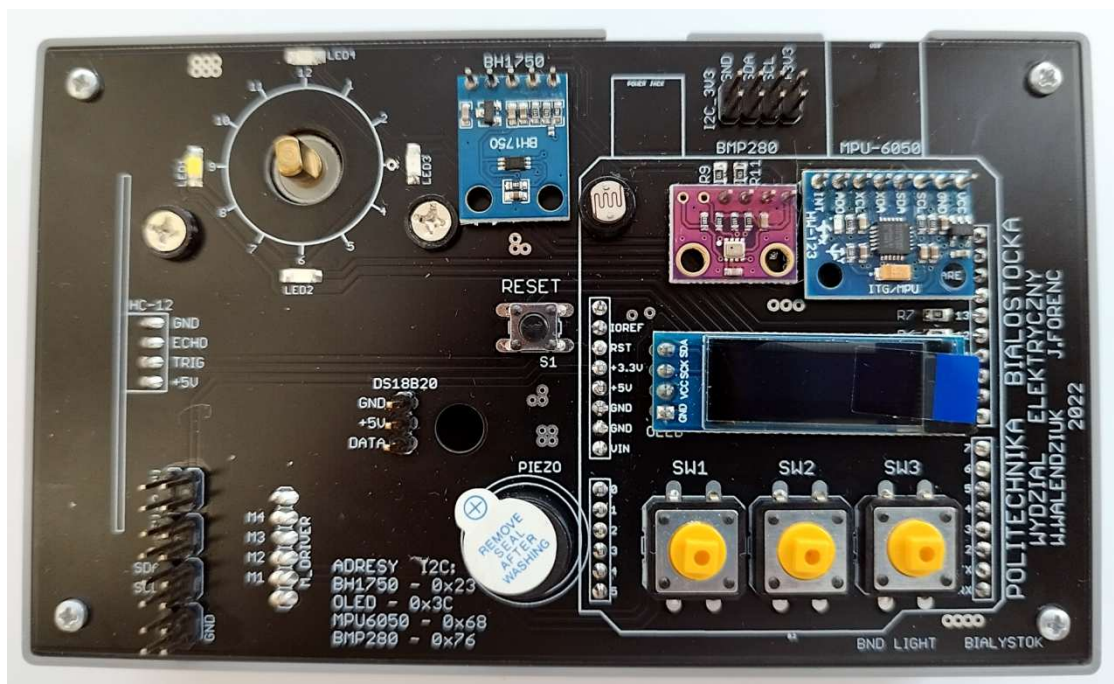
Arduino to projekt typu open-source (Open Hardware), którego głównym celem było opracowanie platformy programistycznej dla systemów wbudowanych opartych na mikrokontrolerach montowanych na pojedynczym obwodzie drukowanym. Język programowania Arduino oparty jest na środowisku Wiring i przypomina języki C/C++. Obecnie najbardziej popularną wersją jest **Arduino Uno R3** (Rys. 3). Moduł ten zawiera: mikrokontroler AVR ATmega328 taktowany sygnałem zegarowym o częstotliwości 16 MHz, 32 kB pamięci programu Flash, 1 kB pamięci EEPROM, 2 kB pamięci operacyjnej SRAM, 14 programowalnych cyfrowych wejść/wyjść (6 z nich może pracować jako kanały PWM, np. do sterowania silnikami lub regulowania jasności diod), 6 analogowych wejść wbudowanego przetwornika analogowo-cyfrowego o rozdzielczości 10-bitów, port USB, gniazdo zasilające, interfejs ICSP, przycisk Reset i diodę LED. Do zasilania

Arduino można wykorzystać dowolny zasilacz o napięciu od 7 V do 12 V. Moduł może być również zasilany bezpośrednio z komputera poprzez przewód USB.



Rys. 3. Arduino UNO R3

Na zajęciach będzie wykorzystywany moduł **Arduino UNO R3** wraz z dodatkowymi elementami i czujnikami, umieszczony w zamkniętej obudowie (Rys. 4).

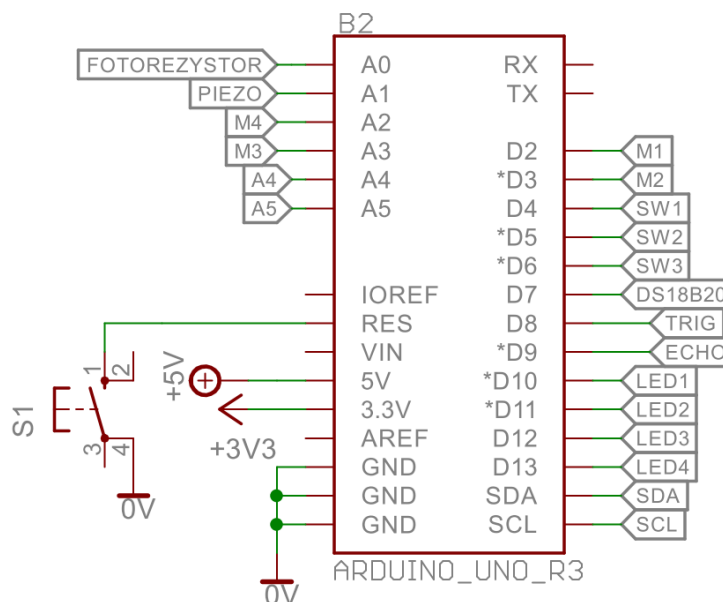


Rys. 4. Moduł Arduino wraz z dodatkowymi elementami i czujnikami

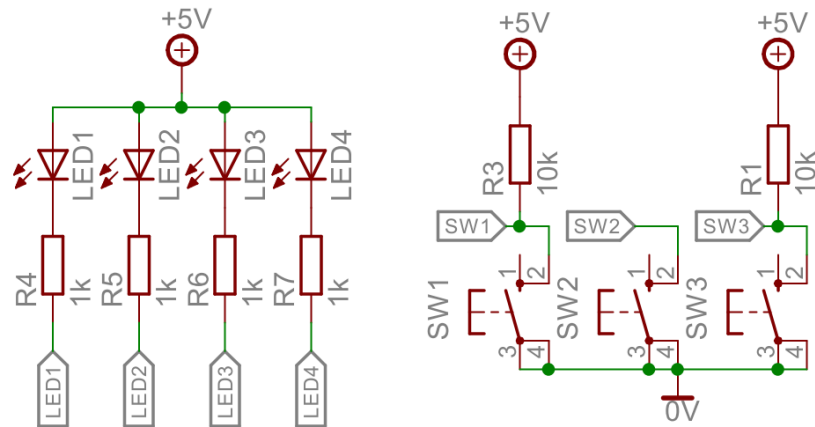
Lista dodatkowych elementów:

- **LED1, LED2, LED3, LED4** - diody LED w różnych kolorach;
- silnik krokowy;
- moduł **BH1750** - czujnik natężenia światła;
- fotorezystor;
- moduł **BMP280** - czujnik ciśnienia z wbudowanym termometrem;
- moduł **MPU-6050** - 3-osiowy akcelerometr i żyroskop;
- moduł wyświetlacza OLED;
- przycisk **RESET** - przycisk resetujący mikrokontroler;
- **PIEZO** - buzzer;
- **SW1, SW2, SW3** - przyciski;
- moduł **DS18B20** - czujnik temperatury;
- moduł **HC-SR04** - ultradźwiękowy czujnik odległości.

Schemat podłączenia poszczególnych elementów i czujników do modułu Arduino przedstawiono na Rys. 5 i Rys. 6.



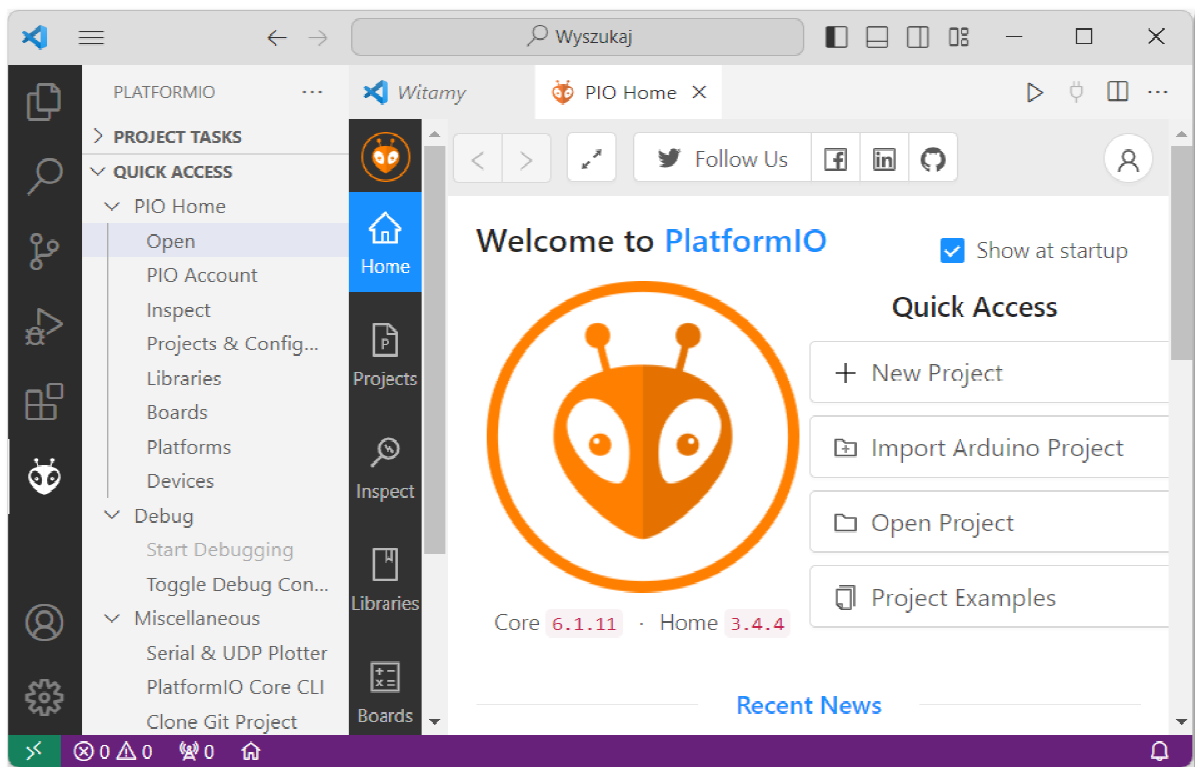
Rys. 5. Schemat podłączenia wyprowadzeń modułu Arduino



Rys. 6. Schemat podłączenia diod LED oraz włączników

2.3. Stworzenie nowego projektu, kompilacja i uruchomienie programu

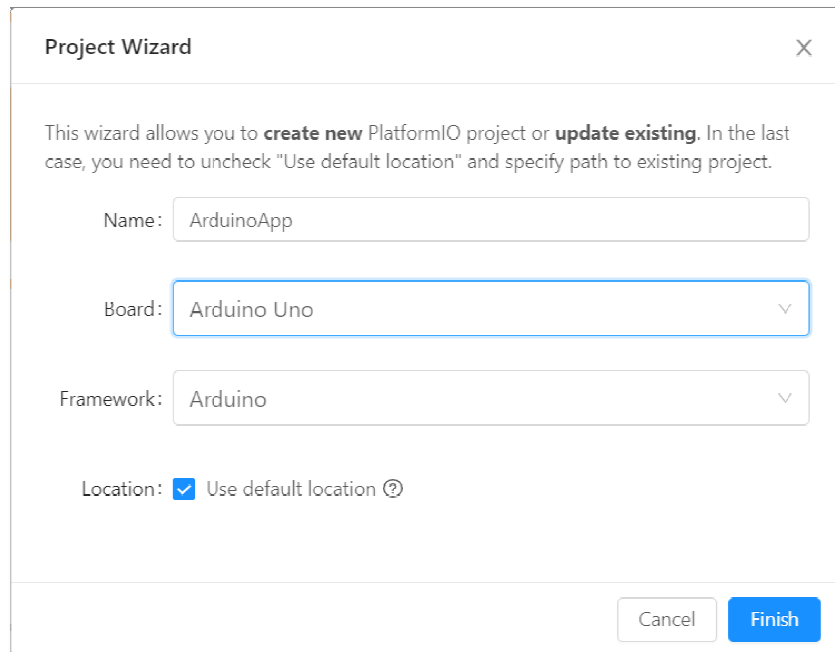
W celu napisania programu na platformę Arduino, musimy stworzyć nowy projekt. Klikamy ikonę rozszerzenia **PlatformIO**, a następnie w oknie, które się pojawi, wybieramy opcję **PIO Home** → **Open** (Rys. 7).



Rys. 7. Uruchomienie rozszerzenia PlatformIO

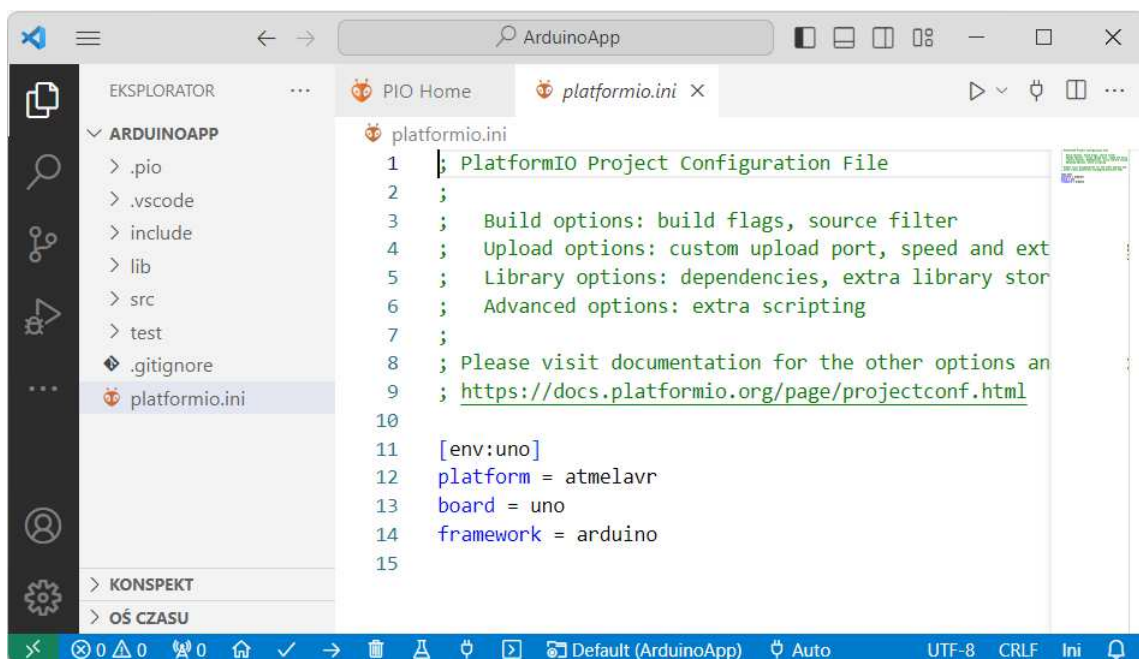
W oknie **Welcome to PlatformIO** wybieramy opcję **New Project**. W oknie **Project Wizard** (Rys. 8) wprowadzamy:

- **Name** - nazwę naszego projektu;
- **Board** - wyszukujemy i wybieramy Arduino Uno;
- **Framework** - nie mamy żadnego wyboru, domyślnie będzie to Arduino;
- **Location** - położenie folderu, w którym będą znajdowały się pliki projektu.



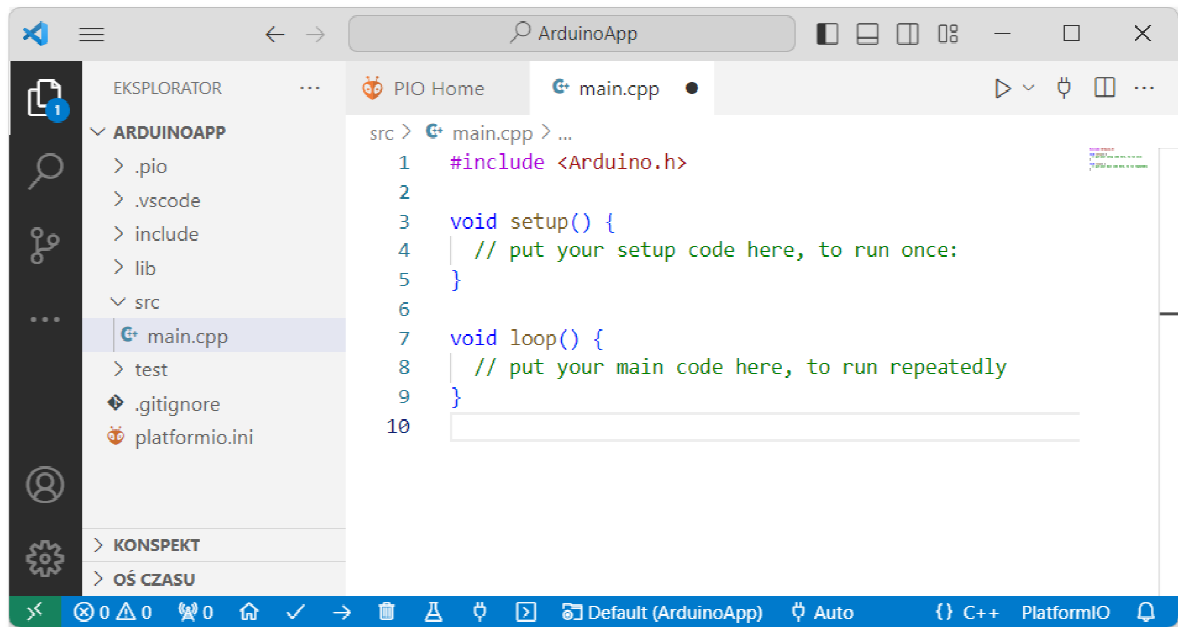
Rys. 8. Tworzenie nowego projektu

Po zatwierdzeniu danych (**Finish**) projekt zostanie utworzony (Rys. 9).



Rys. 9. Visual Studio Code po utworzeniu nowego projektu

Domyślnie w edytorze zostanie wyświetlona zawartość głównego pliku konfiguracyjnego projektu **PlatformIO**. W celu wyświetlenia pliku z kodem źródłowym, rozwijamy opcję **> src** i dwukrotnie klikamy myszką na nazwie pliku **main.cpp**. Plik zostanie otwarty w edytorze (Rys. 10).

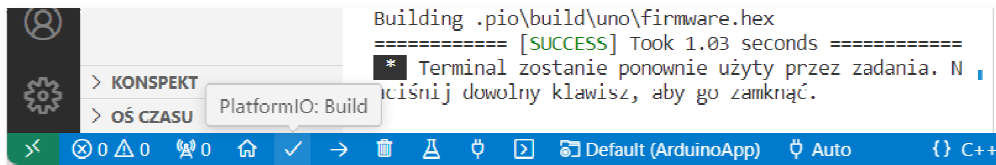


Rys. 10. Plik z kodem źródłowym programu

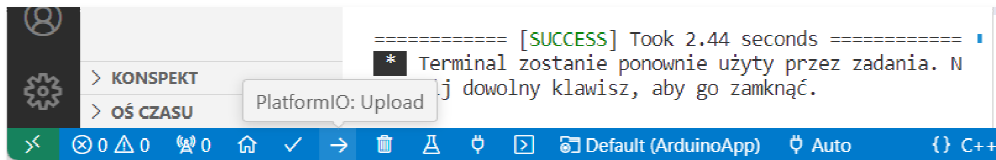
Jak można zauważyć w programie tym nie ma funkcji **main()** charakterystycznej dla programów napisanych w języku C. Najprostszy program dla platformy Arduino składa się z dwóch funkcji:

- **setup()** - funkcja wykonywana tylko jeden raz po włączeniu zasilania, przeznaczona jest głównie do ustawień początkowych;
- **loop()** - funkcja wykonywana w pętli nieskończonej, zawiera główną część programu.

Program pokazany na Rys. 10 nie wykonuje żadnych operacji, ale można go już skompilować, przesłać do modułu Arduino i uruchomić. Wszystkie te operacje wykonujemy klikając odpowiednie ikonki na niebieskim pasku znajdującym się na dole okna programu. Kliknięcie ikonki znacznika wyboru ✓ (**PlatformIO: Build**, Rys. 11) spowoduje kompilację programu, zaś kliknięcie ikonki ze strzałką → (**PlatformIO: Upload**, Rys. 12) spowoduje przesłanie programu do Arduino i jego uruchomienie.



Rys. 11. Kompilacja programu



Rys. 12. Przesłanie programu do modułu Arduino i uruchomienie

2.4. Sterowanie diodą LED

Zadaniem pierwszego programu będzie sterowanie diodą **LED1**. W programie tym dioda będzie naprzemiennie zapalała się i gasła. Kod programu pokazany jest poniżej.

Migająca dioda LED.

```
#include <Arduino.h>

#define LED1_PIN 10

void setup() {
  pinMode(LED1_PIN, OUTPUT);
  digitalWrite(LED1_PIN, HIGH);
}

void loop() {
  digitalWrite(LED1_PIN, LOW);
  delay(500);
  digitalWrite(LED1_PIN, HIGH);
  delay(500);
}
```

Na Rys. 5 możemy zauważyć, że dioda **LED1** podłączona jest do pinu Arduino o numerze **10**. W programie definiujemy stałą preprocesora **LED1_PIN**, która będzie przechowywać ten numer. Pin nr **10** musimy ustawić jako wyjście. Do tego

celu służy funkcja **pinMode()**, którą umieszczamy w funkcji **setup()**. Funkcja ta ma dwa argumenty. Pierwszym jest numer pinu (liczba od **0** do **13**), zaś drugim - określenie funkcji pinu (**INPUT**, **OUTPUT**, **INPUT_PULLUP**). Pin ustawiony jako wyjście może znajdować się w jednym z dwóch stanów logicznych: wysokim (**HIGH**) i niskim (**LOW**). Anoda diody **LED1** podłączona jest do napięcia **5 V** (Rys. 6), natomiast jej katoda, przez rezystor **R4**, do pinu nr **10**. Jeśli pin nr **10** będzie miał stan wysoki, to przez diodę nie popłynie prąd i nie będzie się świecić. Jeśli na pinie nr **10** ustawimy stan niski to prąd popłynie i dioda zacznie świecić się. Do ustawienia stanu logicznego pinu wyjściowego stosujemy funkcję **digitalWrite()**. Funkcja ta ma dwa argumenty. Pierwszym jest numer pinu, zaś drugim stan logiczny (**HIGH**, **LOW**). W funkcji **setup()** ustawiamy początkowy stan diody **LED1** na **HIGH**, czyli nie będzie ona świecić się. Następnie wykonywana jest, w pętli nieskończonej, funkcja **loop()**. W funkcji tej ustawiamy stan pinu nr **10** na niski (dioda **LED1** zaczyna świecić się). Następnie wywołujemy funkcję **delay()**, która wstrzymuje wykonanie programu na czas będący jej argumentem. Czas podajemy w milisekundach. Po 500 ms zmieniamy stan pinu nr **10** na wysoki (dioda gaśnie) i ponownie wstrzymujemy działania programu na 500 ms. Po zakończeniu funkcji **loop()** jest ona ponownie wykonywana.

2.5. Zastosowanie przycisku do sterowania układem

Do sterowania układami bardzo często stosuje się przyciski. W kolejnym programie, po wciśnięciu przycisku **SW1**, dioda **LED1** będzie świecić się przez 2 sekundy.

Przycisk zapalający diodę LED na 2 sekundy.

```
#include <Arduino.h>

#define LED1_PIN 10
#define SW1_PIN 4

void setup() {
  pinMode(LED1_PIN, OUTPUT);
  pinMode(SW1_PIN, INPUT);
}
```

```

    digitalWrite(LED1_PIN, HIGH);
}

void loop() {
    if (digitalRead(SW1_PIN) == LOW)
    {
        digitalWrite(LED1_PIN, LOW);
        delay(2000);
        digitalWrite(LED1_PIN, HIGH);
    }
}

```

Zgodnie z Rys. 5, przycisk **SW1** jest podłączony do pinu Arduino o numerze **4**. Definiujemy dwie stałe preprocesora, **LED1_PIN** oraz **SW1_PIN**, które będą przechowywać numery pinów. W funkcji **setup()** ustawiamy pin nr **10** jako wyjście (**OUTPUT**), a pin nr **4** jako wejście (**INPUT**). Początkowy stan diody **LED1** ustawiamy na **HIGH**, co oznacza, że nie będzie ona świecić się. Analizując Rys. 6 możemy zauważyć, że jeśli przycisk **SW1** nie jest wciśnięty, to na pinie nr **4** jest stan wysoki (**HIGH, +5V**), natomiast po wciśnięciu przycisku, na pinie nr **4** pojawia się stan niski (**LOW, 0V**). W funkcji **loop()** sprawdzamy stan pinu funkcją **digitalRead()**. Jeśli stan jest niski, to wtedy zapalamy diodę **LED1**, czekamy 2 sekundy, a następnie wyłączamy diodę.

W następnym przykładzie naciśnięcie przycisku **SW1** powoduje zapalenie diody **LED1**. Kolejne wciśnięcie tego przycisku powoduje jej zgaszenie.

Przycisk zapalający i gaszący diodę LED.

```

#include <Arduino.h>

#define LED1_PIN 10
#define SW1_PIN 4

bool led_on = false;

void setup() {
    pinMode(LED1_PIN, OUTPUT);
}

```

```

    pinMode(SW1_PIN, INPUT);
    digitalWrite(LED1_PIN, HIGH);
}

void loop() {
    if (digitalRead(SW1_PIN) == LOW)
    {
        delay(10);
        while (digitalRead(SW1_PIN) == LOW);
        delay(10);

        if (led_on == true)
        {
            led_on = false;
            digitalWrite(LED1_PIN, HIGH);
        }
        else
        {
            led_on = true;
            digitalWrite(LED1_PIN, LOW);
        }
    }
}

```

W programie zadeklarowano zmienną **led_on**, która przyjmuje jedną z dwóch wartości logicznych: **true** (dioda **LED1** zapalona), **false** (dioda **LED1** zgaszona). Wciśnięcie przycisku **SW1** powoduje zmianę wartości zmiennej **led_on** na przeciwną oraz zmianę stanu logicznego na pinie nr **10**, do którego podłączona jest dioda **LED1**. Dodatkowego wyjaśnienia wymaga obecność poniższego fragmentu kodu.

```

    delay(10);
    while (digitalRead(SW1_PIN) == LOW);
    delay(10);

```

W momencie naciśnięcia lub zwolnienia przycisku pojawia się zjawisko drgania styków. Polega ono na tym, że kiedy styki przycisku stykają się lub rozłączają, zwarcie styków nie następuje do razu, ale może dochodzić do wielokrotnych

krótkotrwałych połączeń i rozłączeń. Dodanie opóźnienia **delay(10)**; pozwala pominąć czas trwania tego zjawiska. Dodatkowo, może wystąpić sytuacja, w której przycisk **SW1** zostanie naciśnięty i nie zostanie zwolniony. Dodanie pętli **while (digitalRead(SW1_PIN) == LOW)**; spowoduje, że program będzie w niej czekał na zwolnienie przycisku. W rezultacie, w powyższym programie dioda **LED1** będzie zapalana i gaszona w momencie zwalniania przycisku.

3. Przebieg ćwiczenia

Na pracowni specjalistycznej należy wykonać wybrane zadania wskazane przez prowadzącego zajęcia. W różnych grupach mogą być wykonywane różne zadania.

1. Napisz program, w którym cztery diody LED zapalają się kolejno i gasną, tworząc efekt poruszającego się punktu świetlnego po okręgu (zgodnie z ruchem wskazówek zegara).
2. Napisz program, w którym cztery diody LED zapalają się kolejno i gasną tworząc efekt poruszającego się punktu świetlnego po okręgu (przeciwnie do ruchu wskazówek zegara).
3. Napisz program, w którym 4 diody LED zapalają się kolejno i gasną tworząc efekt poruszającego się punktu świetlnego po okręgu. Każdorazowe naciśnięciu przycisku **SW1** powinna spowodować zmianę kierunku ruchu punktu świetlnego.
4. Napisz program, w którym przyciski **SW1**, **SW2** i **SW3** sterują świeceniem i gaszeniem diod **LED1**, **LED2** i **LED3**. Po uruchomieniu programu diody nie powinny świecić się. Po naciśnięciu przycisku **SW1** dioda **LED1** powinna zaświecić się. Ponowne naciśnięcie tego przycisku powinno spowodować jej zgaszenie. Pozostałe przyciski i diody LED powinny działać na tej samej zasadzie.

4. Literatura

- [1] Prata S.: Język C. Szkoła programowania. Wydanie VI. Helion, Gliwice, 2016.
- [2] Wrotek W.: Arduino od podstaw. Helion, Gliwice, 2023.
- [3] Monk S.: Arduino dla początkujących. Podstawy i szkice. Helion, Gliwice, 2019.
- [4] Evans M., Noble J., Hochenbaum J.: Arduino w akcji. Helion, Gliwice, 2014.
- [5] Kernighan B.W., Ritchie D.M.: Język ANSI C. Programowanie. Wydanie II. Helion, Gliwice, 2010.
- [6] <https://code.visualstudio.com/> - Visual Studio Code
- [7] <https://www.arduino.cc/reference/en/> - Arduino Language Reference

5. Pytania kontrolne

1. Wyjaśnij, w jaki sposób w Visual Studio Code można utworzyć projekt, skompilować program, przesłać go i uruchomić na module Arduino.
2. Opisz ogólną strukturę programu działającego na platformie Arduino.
3. Wyjaśnij, w jaki sposób można sterować diodą LED przy wykorzystaniu przycisku.

6. Wymagania BHP

Warunkiem przystąpienia do praktycznej realizacji ćwiczenia jest zapoznanie się z instrukcją BHP i instrukcją przeciw pożarową oraz przestrzeganie zasad w nich zawartych.

W trakcie zajęć laboratoryjnych należy przestrzegać następujących zasad.

- Sprawdzić, czy urządzenia dostępne na stanowisku laboratoryjnym są w stanie kompletnym, nie wskazującym na fizyczne uszkodzenie.

- Jeżeli istnieje taka możliwość, należy dostosować warunki stanowiska do własnych potrzeb, ze względu na ergonomię. Monitor komputera ustawić w sposób zapewniający stałą i wygodną obserwację dla wszystkich członków zespołu.
- Sprawdzić prawidłowość połączeń urządzeń.
- Załączenie komputera może nastąpić po wyrażeniu zgody przez prowadzącego.
- W trakcie pracy z komputerem zabronione jest spożywanie posiłków i picie napojów.
- W przypadku zakończenia pracy należy zakończyć sesję przez wydanie polecenia wylogowania. Zamknięcie systemu operacyjnego może się odbywać tylko na wyraźne polecenie prowadzącego.
- Zabronione jest dokonywanie jakichkolwiek przełączeń oraz wymiana elementów składowych stanowiska.
- Zabroniona jest zmiana konfiguracji komputera, w tym systemu operacyjnego i programów użytkowych, która nie wynika z programu zajęć i nie jest wykonywana w porozumieniu z prowadzącym zajęcia.
- W przypadku zaniku napięcia zasilającego należy niezwłocznie wyłączyć wszystkie urządzenia.
- Stwierdzone wszelkie braki w wyposażeniu stanowiska oraz nieprawidłowości w funkcjonowaniu sprzętu należy przekazywać prowadzącemu zajęcia.
- Zabrania się samodzielnego włączania, manipulowania i korzystania z urządzeń nie należących do danego ćwiczenia.
- W przypadku wystąpienia porażenia prądem elektrycznym należy niezwłocznie wyłączyć zasilanie stanowiska. Przed odłączeniem napięcia nie dotykać porażonego.