

Sprawozdanie z Zadania 4. Sterowanie przekaźnikiem

Adrian Lesniak nr.indeksu: 154256

1. Cel Ćwiczenia	3
2. Opis Elementów Wykorzystanych w Ćwiczeniu	3
Załączona Dokumentacja Techniczna - Podstawowe parametry.	4
Budowa Układu	6
3. Schemat Połączeń	6
4. Obliczenia Rezystancji dla Diod.	7
5. Kod Programu z Komentarzami	9
Projekt składa się z kilku kroków:	10
7. Uwagi i Wnioski	13
8. Wykonanie projektu na odpowiednich komponentach.	14

1. Cel Ćwiczenia

Celem ćwiczenia było zaprojektowanie i zbudowanie układu, który pozwala na sterowanie przekaźnikiem za pomocą platformy Arduino. Układ miał załączać diody LED w zależności od stanu przekaźnika oraz wyświetlać komunikaty o stanie przekaźnika na wyświetlaczu LCD, korzystając z polskich znaków.

2. Opis Elementów Wykorzystanych w Ćwiczeniu

Projekt wykorzystuje płytke Arduino UNO R3, charakteryzującą się prostotą i uniwersalnością, co czyni ją idealną do celów edukacyjnych oraz prototypowych. Płyta ta jest wyposażona w mikrokontroler ATmega328, który oferuje wystarczające zasoby i porty I/O do obsługi przekaźnika, dwóch diod LED oraz wyświetlacza LCD, przy stosunkowo niskim zużyciu energii. W projekcie wykorzystano także biblioteki Wire.h i LiquidCrystal_I2C.h do obsługi magistrali I2C oraz ekranu LCD, co umożliwia wygodną komunikację między urządzeniami za pomocą minimalnej liczby przewodów.

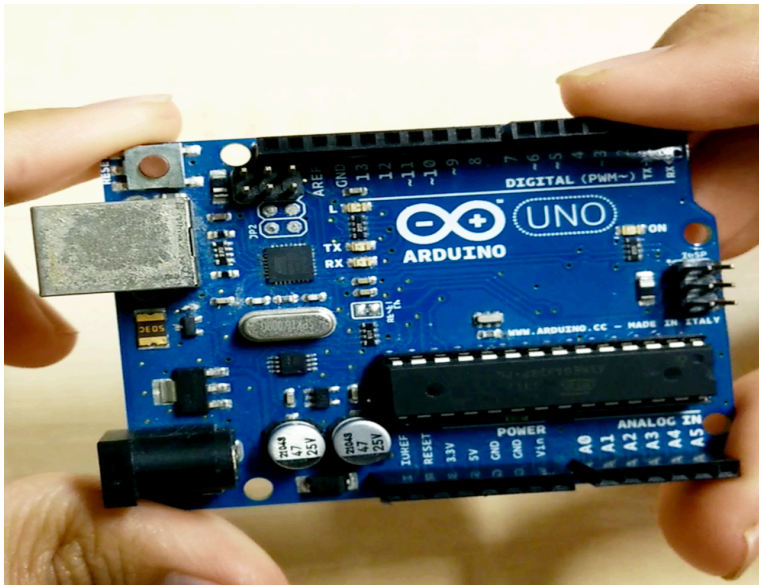
Rezystory o wartości 220Ω , które zostały użyte w projekcie, ograniczają prąd płynący przez diody LED, zapobiegając ich uszkodzeniu przez nadmierny prąd. Użycie jednolitych rezystorów upraszcza konstrukcję i pozwala na łatwe obliczenia oraz montaż układu. Przycisk monostabilny, użyty do sterowania przekaźnikiem, pozwala na manualne przełączanie między stanami przekaźnika, co jest odczytywane z osobnego wejścia cyfrowego na Arduino.

Wyświetlacz LCD 1602 z interfejsem I2C służy do wyświetlania komunikatów o stanie przekaźnika w języku polskim, z zastosowaniem polskich znaków. Płytko stykowa (breadboard) umożliwia szybkie i łatwe prototypowanie układu bez konieczności lutowania, a przewody połączeniowe zapewniają elastyczność w konfiguracji układu.

- **Arduino UNO:** Platforma mikrokontrolerów używana do sterowania przekaźnikiem i diodami.
- **Dioda LED:** Czerwona i zielona – każda dioda podłączona przez rezystor 220Ω , które ograniczają prąd płynący przez diody.
- **Przekaźnik:** Używany do załączania diod LED w zależności od stanu przycisku.

- **Wyświetlacz LCD 1602** z interfejsem I2C: Służy do wyświetlania statusu przekaźnika.
- **Rezystory 220Ω**: Ograniczają prąd płynący przez diody LED, zapobiegając ich uszkodzeniu.
- **Przycisk monostabilny**: Służy do sterowania przekaźnikiem.
- **Płytki stykowa**: Używana do prototypowania układu.

Rys 1. Zdjęcie Arduino UNO R3



Załączona Dokumentacja Techniczna - **Podstawowe parametry.**

Tabela 2: Parametry zastosowanego wyświetlacza LCD z konwerterem I2C

Parametr	Wielkość
Rodzaj wyświetlacza	LCD 2×16 znaków
Kolor znaków	Biały
Podświetlenie	Niebieskie
Rozmiar modułu	80 × 36 mm
Wymiary znaku	2,45 × 5,00 mm
Zakres temperatury pracy	-20°C do +70°C

Tabela 1: Specyfikacja płytki Arduino UNO R3

Parametr	Wielkość
Napięcie zasilania	Od 7 V do 12 V
Mikrokontroler	ATmega328
Maksymalna częstotliwość zegara	16 MHz
Pamięć SRAM	2 kB
Pamięć Flash	32 kB
Pamięć EEPROM	1 kB
Porty I/O	14
Wyjścia PWM	6
Wejścia Analogowe	6
Interfejsy szeregowe	UART, SPI, I2C
Wymiary	68,6 × 53,4 mm

Przełącznik sterowany elektrycznie



Nazwy pinów

Nazwa	Opis
Wirtualna karta kredytowa	Napięcie zasilania
GND	Grunt
W	Sygnał sterujący (np. z mikrokontrolera)
NC	Normalnie zamknięty
COM	Wspólny pin
NIE	Normalnie otwarty

Budowa Układu

Układ zawiera Arduino UNO połączone z przełącznikiem, który steruje dwoma diodami LED (czerwoną i zieloną) poprzez rezystory 220Ω. Te diody są połączone do przełącznika tak, że dioda czerwona jest aktywowana, gdy przełącznik jest załączony, a dioda zielona, gdy przełącznik jest wyłączony. Wyświetlacz LCD jest podłączony do Arduino za pomocą interfejsu I2C, gdzie pin SDA jest podłączony do A4, a SCL do A5. Przycisk monostabilny jest podłączony do pinu 3 i do masy (GND), wykorzystując wewnętrzny rezystor podciągający Arduino.

Połączenia Arduino UNO R3 z wyświetlaczem LCD za pomocą magistrali I2C dokonano w następujący sposób:

- **VCC wyświetlacza** do 5V Arduino,
- **GND wyświetlacza** do GND Arduino,
- **SDA wyświetlacza** do pinu A4 Arduino,
- **SCL wyświetlacza** do pinu A5 Arduino.

Dzięki wykorzystaniu szyny I2C, minimalizowana jest liczba potrzebnych przewodów, co upraszcza konstrukcję układu. Schemat połączeń ilustruje wszystkie połączenia, które są niezbędne do prawidłowego działania układu. Ponadto, wykorzystanie przełącznika umożliwia sterowanie wyższymi napięciami i prądami niż mogłoby być bezpośrednio obsługiwane przez Arduino, co zwiększa funkcjonalność i bezpieczeństwo układu.

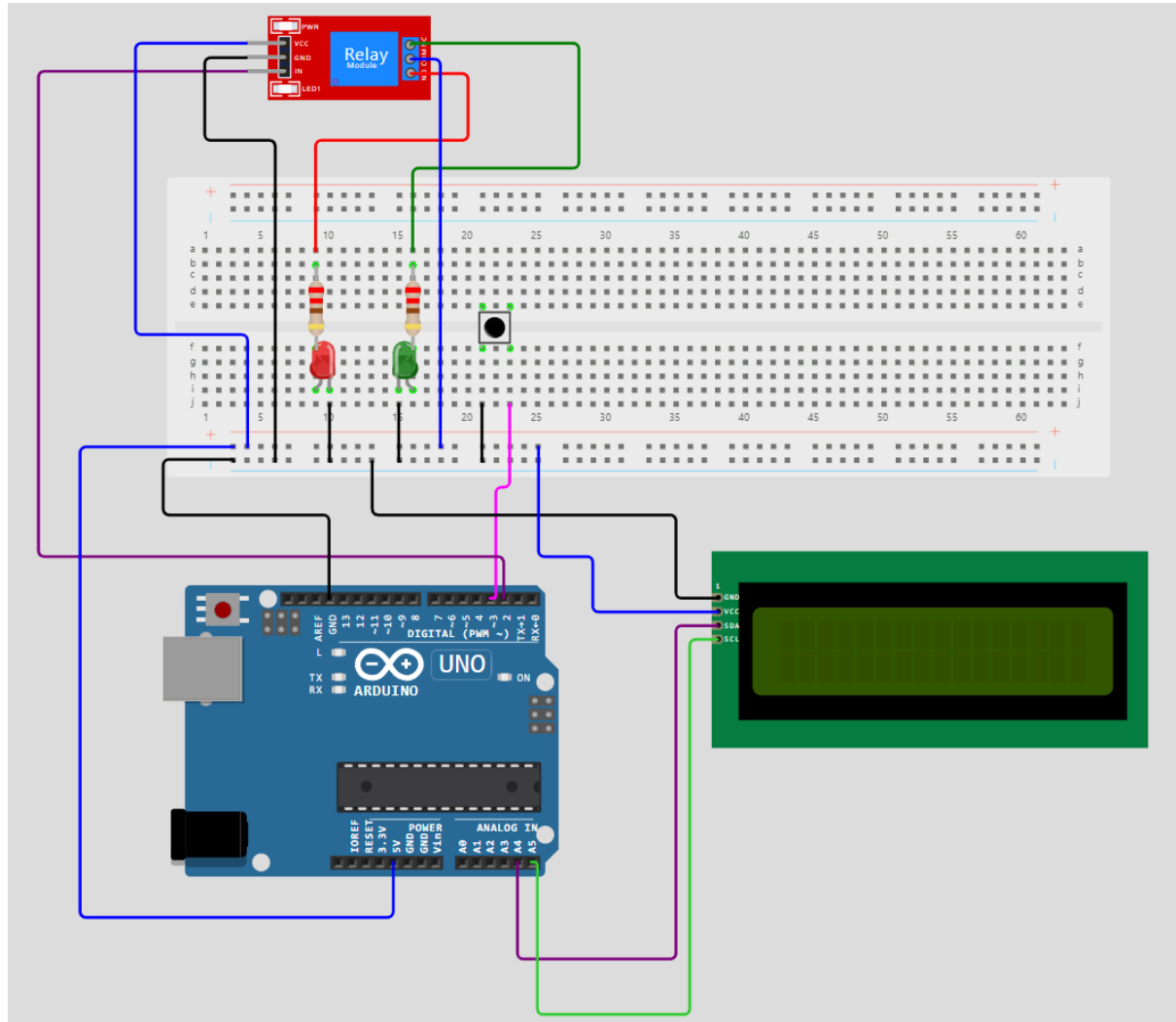
Tabela 3: Opis wyprowadzeń magistrali I2C dla wyświetlacza LCD

Nazwa	Opis	Podłączenie do Arduino
GND	Masa układu	GND
VCC	Zasilanie +5 V	5V
SDA	Linia danych magistrali I2C	A4
SCL	Linia zegarowa magistrali I2C	A5

3. Schemat Połączeń

Układ zawiera Arduino UNO połączone z przełącznikiem, który steruje dwoma diodami LED. Dioda czerwona zapala się, gdy przełącznik jest załączony, a zielona,

gdy jest wyłączony. Wyświetlacz LCD jest podłączony do Arduino za pomocą interfejsu I2C. Przycisk monostabilny jest podłączony do jednego z pinów cyfrowych, służąc do sterowania przekaźnikiem.



Rys 2. Na schemacie widoczne jest Arduino UNO, które jest połączone z modułem przekaźnika, dwoma diodami LED (czerwoną i zieloną) przez rezystory, oraz z wyświetlaczem LCD za pomocą interfejsu I2C.

4. Obliczenia Rezystancji dla Diod.

W zadaniu 4 użyto dwóch diod LED: czerwonej i zielonej. Zastosowano rezystory o wartości 220Ω , które ograniczają prąd płynący przez diody. Standardowe napięcie przewodzenia dla czerwonej diody wynosi około 2V, a dla zielonej 3V. Wykorzystanie

tych rezystorów umożliwia bezpieczne użytkowanie diod przy zalecanym prądzie około 20 mA.

Dla każdej diody LED obliczenia wartości rezystancji dokonano z wykorzystaniem prawa Ohma:

$$R = \frac{V_s - V_f}{I}$$

gdzie:

- **V_s** to napięcie źródła (5V),
- **V_f** to spadek napięcia na diodzie (2V dla czerwonej, 3V dla zielonej),
- **I** to prąd diody.

Czerwona dioda:

Przy założeniu, że spadek napięcia na czerwonej diodzie wynosi 2V, a zalecany prąd diody wynosi 20 mA, obliczenia dają:

$$R = \frac{5V - 2V}{20mA} = 150\Omega$$

Jednakże, użyto rezystora 220 Ω , co zwiększa bezpieczeństwo pracy diody.

Zielona dioda:

Przy spadku napięcia 3V i prądzie 20 mA, obliczenia dają:

$$R = \frac{5V - 3V}{20mA} = 100\Omega$$

Podobnie, zastosowanie rezystora 220 Ω jest odpowiednie dla bezpiecznej pracy. Te obliczenia pokazują, że wartości rezystorów 220 Ω są bezpieczne i odpowiednie dla obu diod, nawet jeśli nie są optymalne z matematycznego punktu widzenia. Pozwala to na uproszczenie układu i minimalizację ryzyka przepalenia diod LED.

5. Kod Programu z Komentarzami

Program na Arduino odpowiada za sterowanie przełącznikiem, który z kolei kontroluje diody LED. Na podstawie stanu przełącznika, program załącza diodę czerwoną (gdy przełącznik jest załączony) lub zieloną (gdy przełącznik jest wyłączony), jednocześnie wyświetlając odpowiedni status na wyświetlaczu LCD. Kod zawiera ochronę przed drganiem styków przycisku oraz implementuje logikę bistabilną, która pozwala na załączanie i wyłączanie przełącznika za każdym naciśnięciem przycisku.

Kod źródłowy został opracowany zgodnie z poniższym listingiem:

```
sketch.ino  diagram.json  libraries.txt  Library Manager  ▼

1  #include <LiquidCrystal_I2C.h> // Dołączenie biblioteki do obsługi wyświetlacza LCD z interfejsem I2C
2
3  // Definicje polskich znaków
4  byte customCharZ[8] = {000010,000100,011111,000010,000100,001000,011111,000000}; // Zdefiniowanie własnego znaku dla litery 'ż'
5  byte customCharI[8] = {001100, 000100, 000110, 000100, 001100, 000100, 001110, 000000}; // Zdefiniowanie własnego znaku dla litery 'i'
6  byte customCharA[8] = {000000,000000,001110,000001,001111,010001,001111,000001}; // Zdefiniowanie własnego znaku dla litery 'ą'
7
8  const int relayPin = 2; // Deklaracja pinu do sterowania przełącznikiem
9  const int buttonPin = 3; // Deklaracja pinu do odczytu stanu przycisku
10 bool relayState = LOW; // Początkowy stan przełącznika (wyłączony)
11 bool lastButtonState = HIGH; // Początkowy stan przycisku (nie naciśnięty, przy użyciu pullup)
12 bool buttonPressed = false; // Flaga wskazująca, czy przycisk został naciśnięty
13
14 LiquidCrystal_I2C lcd(0x27, 16, 2); // Inicjalizacja wyświetlacza LCD z adresem I2C 0x27 i rozmiarem 16x2
15
16 void setup() {
17   pinMode(relayPin, OUTPUT); // Ustawienie pinu przełącznika jako wyjście
18   pinMode(buttonPin, INPUT_PULLUP); // Ustawienie pinu przycisku jako wejście z wewnętrznym rezystorem podciągającym
19   lcd.init(); // Inicjalizacja LCD
20   lcd.backlight(); // Włączenie podświetlenia LCD
21   Serial.begin(9600); // Rozpoczęcie komunikacji szeregowej z prędkością 9600 bps
22
23   // Ustawienie początkowego stanu przełącznika
24   digitalWrite(relayPin, relayState);
25
26   // Utworzenie własnych znaków na LCD
27   lcd.createChar(0, customCharZ); // Utworzenie znaku dla 'ż'
28   lcd.createChar(1, customCharI); // Utworzenie znaku dla 'i'
29   lcd.createChar(2, customCharA); // Utworzenie znaku dla 'ą'
30
31   // Wyświetlenie początkowego stanu na LCD i konsoli
32   updateDisplayAndConsole();
33 }
34
35 void loop() {
36   bool buttonState = digitalRead(buttonPin); // Odczytanie stanu przycisku
37
38   // Sprawdzenie czy przycisk został naciśnięty
39   if (buttonState == LOW && lastButtonState == HIGH && !buttonPressed) {
40     relayState = !relayState; // Zmiana stanu przełącznika
41     digitalWrite(relayPin, relayState); // Zastosowanie zmiany stanu przełącznika
42     updateDisplayAndConsole(); // Aktualizacja wyświetlacza i konsoli
43     buttonPressed = true; // Ustawienie flagi naciśnięcia przycisku
44   } else if (buttonState == HIGH) {
45     buttonPressed = false; // Reset flagi naciśnięcia gdy przycisk zostanie zwolniony
46   }
47
48   lastButtonState = buttonState; // Aktualizacja ostatniego stanu przycisku
49   delay(100); // Krótkie opóźnienie dla debouncing
50 }
51
52 void updateDisplayAndConsole() {
53   lcd.clear(); // Wyczyszczenie wyświetlacza
54   lcd.setCursor(0, 0); // Ustawienie kursora na początku pierwszej linii
55   if (relayState) {
56     lcd.print("Przeka"); // Wyświetlenie tekstu na LCD
57     lcd.write(byte(2)); // Wstawienie znaku 'ż'
58     lcd.print("nik ");
59   } else {
60     lcd.print("Przeka");
61     lcd.write(byte(2)); // Wstawienie znaku 'i'
62     lcd.print("nik ");
63   }
```

```

64
65 lcd.setCursor(0, 1); // Ustawienie kursora na początku drugiej linii
66 if (relayState) {
67     lcd.print("Za");
68     lcd.write(byte(0)); // Wstawienie znaku 'i'
69     lcd.write(byte(1)); // Wstawienie znaku 'a'
70     lcd.print("czony");
71     Serial.println("Przełącznik załączony");
72 } else {
73     lcd.print("Wy");
74     lcd.write(byte(0)); // Wstawienie znaku 'i'
75     lcd.write(byte(1)); // Wstawienie znaku 'a'
76     lcd.print("czony");
77     Serial.println("Przełącznik wyłączony");
78 }
79 }
80

```

Szczegółowe omówienie dostarczonego kodu z uwzględnieniem komentarzy wyjaśniających funkcję każdej linii.

Projekt składa się z kilku kroków:

- **Inicjalizacja i konfiguracja płytki Arduino oraz wyświetlacza LCD:**

Na początku wykorzystano bibliotekę LiquidCrystal_I2C do komunikacji z wyświetlaczem LCD poprzez magistralę I2C. Adres I2C wyświetlacza to 0x27, a jego rozmiar to 16 znaków w dwóch wierszach. Płytke Arduino UNO skonfigurowano, aby sterować przełącznikiem oraz dwoma diodami LED (czerwoną i zieloną) za pośrednictwem wyjść cyfrowych, z przełącznikiem kontrolującym stan diod.

- **Konfiguracja pinów i początkowe ustawienia przełącznika oraz diod LED:**

Pin cyfrowy dla przełącznika oraz dla przycisku został skonfigurowany w funkcji setup() w kodzie Arduino. Na początku działania programu przełącznik jest ustawiony na wyłączony, co aktywuje diodę zieloną, a dioda czerwona jest wyłączona.

- **Debouncing i ochrona przed przetrzymaniem przycisku:**

Implementacja debouncingu przycisku zapewnia ignorowanie krótkich zmian stanu przycisku, które mogą wystąpić w momencie jego wciśnięcia lub zwolnienia. Ochrona przed przetrzymaniem przycisku umożliwia jednorazowe wykonanie akcji po naciśnięciu przycisku, niezależnie od długości jego trzymania.

- **Zaprogramowanie logiki sterowania przełącznikiem:**

Program dynamicznie przełącza stan przekaźnika na ON lub OFF za każdym naciśnięciem przycisku. Do zarządzania stanem przekaźnika wykorzystano funkcję `digitalWrite()`, a aktualny stan przekaźnika decyduje o załączeniu odpowiedniej diody LED.

- **Aktualizacja wyświetlacza LCD i komunikaty w monitorze szeregowym:**

Po każdej zmianie stanu przekaźnika, wyświetlacz LCD jest aktualizowany, aby pokazać komunikat "Przekaźnik załączony" lub "Przekaźnik wyłączony". Równocześnie, odpowiedni komunikat jest wysyłany do monitora portu szeregowego, co umożliwia śledzenie działania układu także przez port szeregowy Arduino.

- **Monitorowanie stanu przycisku i zarządzanie stanem przekaźnika:**

Program cały czas monitoruje stan przycisku i na tej podstawie decyduje o przełączeniu przekaźnika, co zapewnia precyzyjne i niezawodne działanie układu bez blokowania wykonywania innych zadań.

- **Testowanie i debugowanie programu:**

Po skonfigurowaniu układu i wgraniu programu, przeprowadzono szereg testów, aby upewnić się, że przekaźnik i diody działają prawidłowo oraz że informacje na LCD oraz w monitorze szeregowym są zgodne z oczekiwaniami.

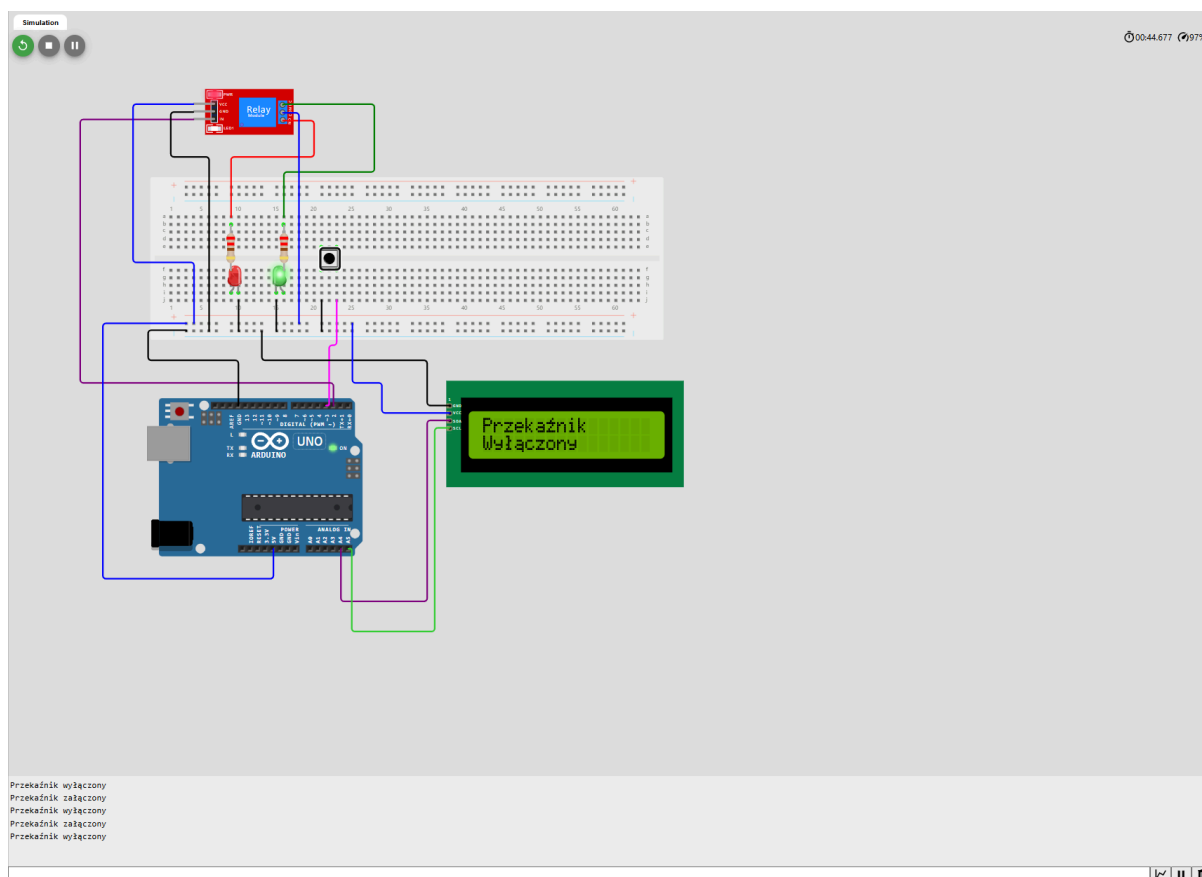
W ramach projektu zaimplementowano systematyczne podejście do budowy i programowania układu, co pozwoliło na głębsze zrozumienie zarówno hardware'u, jak i software'u używanego w projekcie. Użycie funkcji opartych o czas zamiast blokującej funkcji `delay()` znacząco poprawia wydajność i reaktywność systemu.

6. Testowanie i Uruchomienie Układu.

Układ był testowany w środowisku Wokwi, gdzie można było wizualnie zweryfikować poprawność działania przekaźnika, diod LED oraz wyświetlacza. Każde naciśnięcie przycisku monostabilnego zmienia stan przekaźnika, co z kolei załącza odpowiednią diodę (czerwoną gdy przekaźnik jest załączony, zieloną gdy jest wyłączony) oraz aktualizuje wyświetlacz LCD, pokazujący aktualny stan przekaźnika.

- Działanie programu dla zadania 4, które obejmuje sterowanie przekaźnikiem za pomocą Arduino, jest zorganizowane w sposób zapewniający jego funkcjonalność oraz niezawodność. Po uruchomieniu programu, rozpoczyna się od konfiguracji wyświetlacza LCD i portu szeregowego, które są niezbędne do komunikacji i wizualizacji stanu układu. Równocześnie, piny cyfrowe odpowiedzialne za sterowanie przekaźnikiem i diodami są ustawione jako wyjścia, a przekaźnik jest początkowo wyłączony.
- Program korzysta z funkcji *millis()* do śledzenia upływu czasu, co umożliwia kontrolowanie czasu załączenia i wyłączenia przekaźnika bez zatrzymywania działania kodu funkcją *delay()*. Po każdym naciśnięciu przycisku, stan przekaźnika jest zmieniany, co jest rejestrowane przez program, który aktualizuje informacje na wyświetlaczu LCD i wysyła odpowiednie komunikaty do monitora portu szeregowego, informując o stanie przekaźnika, takie jak "Przekaźnik załączony" czy "Przekaźnik wyłączony".
- Cały proces jest powtarzany w nieskończoność, dopóki układ jest zasilany. Program ciągle monitoruje stan przycisku i odpowiednio reaguje, co pozwala na efektywną pracę układu bez niepotrzebnego obciążania procesora czekaniem.
- W trakcie rozwoju programu regularnie przeprowadzane były testy, aby upewnić się, że wszystkie komponenty działają zgodnie z oczekiwaniami. Debugowanie było konieczne, gdy pojawiły się problemy z sekwencją przełączania przekaźnika lub z komunikacją między Arduino a wyświetlaczem, co umożliwiło skuteczne rozwiązanie tych problemów i optymalizację działania układu.

Rys 3. Zdjęcie przedstawiające uruchomiony program z wyświetlaczem LCD, przekaźnikiem, diodami, płytką stykową i przyciskiem monostabilnym.



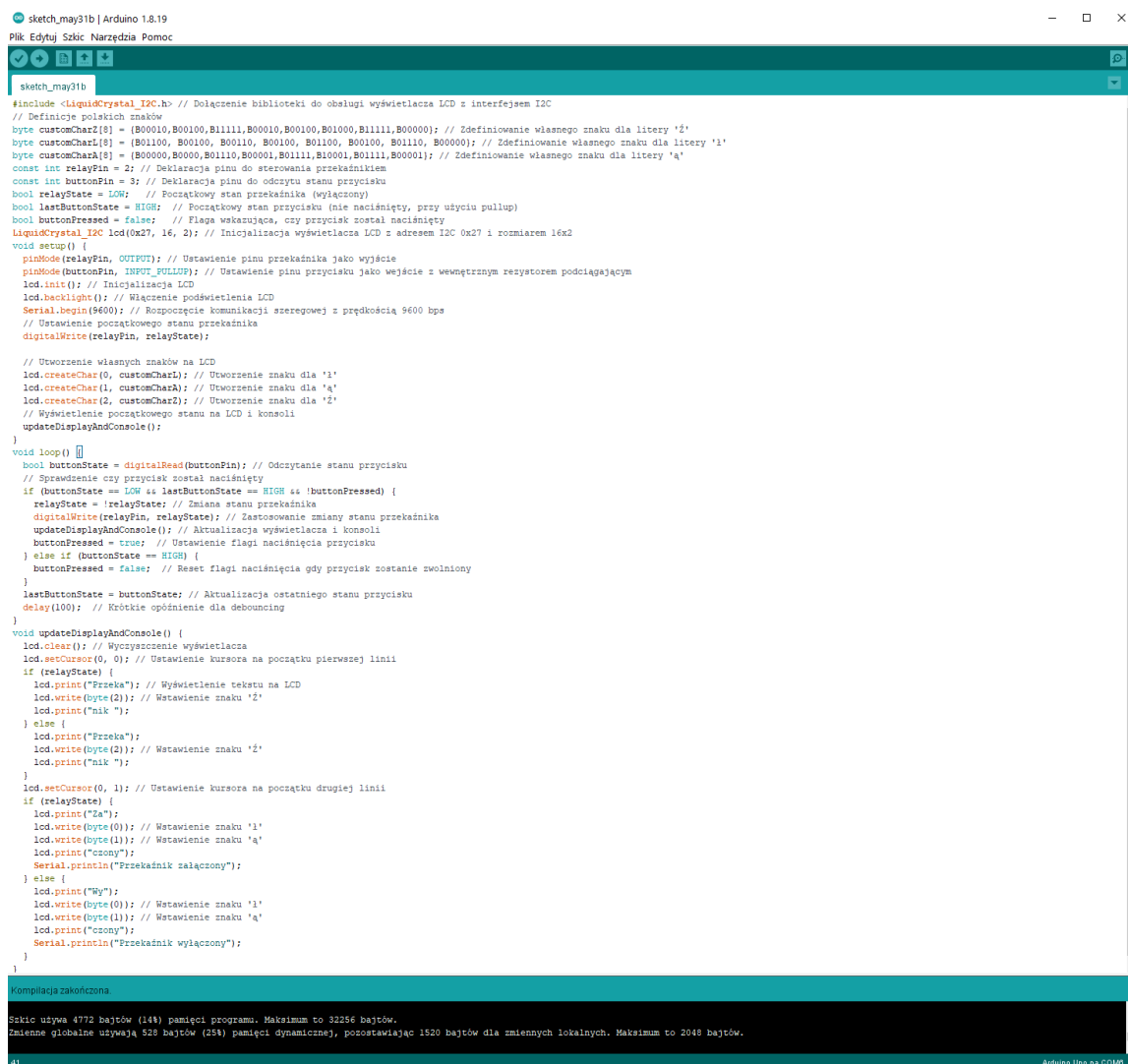
7. Uwagi i Wnioski

1. Projekt ten pozwolił na zrozumienie implementacji debouncingu przycisku oraz ochrony przed jego przetrzymaniem, co znacznie zwiększa niezawodność działania układu. Te techniki zapobiegają przypadkowemu załączaniu i wyłączaniu przełącznika, co jest kluczowe dla stabilnego działania systemu.
2. Dzięki użyciu wyświetlacza LCD z interfejsem I2C, komunikacja z wyświetlaczem została uproszczona, co znacząco zmniejszyło liczbę wymaganych połączeń. To nie tylko upraszcza konstrukcję, ale również ogranicza potencjalne punkty awarii.
3. Programowanie mikrokontrolera do zarządzania stanem przełącznika i diod LED bez użycia blokującej funkcji `delay()` umożliwiło lepszą responsywność i wydajność układu. Zastosowanie funkcji `millis()` do zarządzania czasem pozwala na płynne przełączanie stanów bez przerywania innych zadań wykonywanych przez mikrokontroler.

4. Projekt oferuje również potencjał dalszej rozbudowy. Można by dodać zdalne sterowanie przy użyciu Bluetooth lub WiFi, co umożliwiłoby kontrolę systemu z poziomu aplikacji mobilnej lub przez internet. Istnieje również możliwość rozbudowy o dodatkowe sensory, które mogłyby automatycznie kontrolować przełącznik na podstawie określonych warunków, np. sensor temperatury decydujący o załączeniu wentylatora.

Ten projekt nie tylko zilustrował podstawowe koncepty elektroniki i programowania, ale także pokazał, jak można skalować i modyfikować proste układy do bardziej złożonych aplikacji, oferując solidne podstawy do dalszego rozwoju i eksperymentowania.

8. Wykonanie projektu na odpowiednich komponentach.



```
sketch_may31b | Arduino 1.8.19
Plik Edytuj Szkic Narzędzia Pomoc

sketch_may31b
#include <LiquidCrystal_I2C.h> // Dołączenie biblioteki do obsługi wyświetlacza LCD z interfejsem I2C
// Definicje polskich znaków
byte customCharZ[8] = {B00010,B00100,B11111,B00010,B00100,B01000,B11111,B00000}; // Zdefiniowanie własnego znaku dla litery 'Z'
byte customCharI[8] = {B01100,B00100,B00110,B00100,B01100,B00100,B01110,B00000}; // Zdefiniowanie własnego znaku dla litery 'I'
byte customCharA[8] = {B00000,B0000,B01110,B00001,B01111,B10001,B01111,B00001}; // Zdefiniowanie własnego znaku dla litery 'a'
const int relayPin = 2; // Deklaracja pinu do sterowania przełącznikiem
const int buttonPin = 3; // Deklaracja pinu do odczytu stanu przycisku
bool relayState = LOW; // Początkowy stan przełącznika (wyłączony)
bool lastButtonState = HIGH; // Początkowy stan przycisku (nie naciśnięty, przy użyciu pullup)
bool buttonPressed = false; // Flaga wskazująca, czy przycisk został naciśnięty
LiquidCrystal_I2C lcd(0x27, 16, 2); // Inicjalizacja wyświetlacza LCD z adresem I2C 0x27 i rozmiarem 16x2
void setup() {
  pinMode(relayPin, OUTPUT); // Ustawienie pinu przełącznika jako wyjście
  pinMode(buttonPin, INPUT_PULLUP); // Ustawienie pinu przycisku jako wejście z wewnętrznym rezystorem podciągającym
  lcd.init(); // Inicjalizacja LCD
  lcd.backlight(); // Włączenie podświetlenia LCD
  Serial.begin(9600); // Rozpoczęcie komunikacji szeregowej z prędkością 9600 bps
  // Ustawienie początkowego stanu przełącznika
  digitalWrite(relayPin, relayState);
  // Utworzenie własnych znaków na LCD
  lcd.createChar(0, customCharZ); // Utworzenie znaku dla 'Z'
  lcd.createChar(1, customCharA); // Utworzenie znaku dla 'a'
  lcd.createChar(2, customCharI); // Utworzenie znaku dla 'I'
  // Wyświetlenie początkowego stanu na LCD i konsoli
  updateDisplayAndConsole();
}
void loop() {
  bool buttonState = digitalRead(buttonPin); // Odczytanie stanu przycisku
  // Sprawdzenie czy przycisk został naciśnięty
  if (buttonState == LOW && lastButtonState == HIGH && !buttonPressed) {
    relayState = !relayState; // Zmiana stanu przełącznika
    digitalWrite(relayPin, relayState); // Zastosowanie zmiany stanu przełącznika
    updateDisplayAndConsole(); // Aktualizacja wyświetlacza i konsoli
    buttonPressed = true; // Ustawienie flagi naciśnięcia przycisku
  } else if (buttonState == HIGH) {
    buttonPressed = false; // Reset flagi naciśnięcia gdy przycisk zostanie zwolniony
  }
  lastButtonState = buttonState; // Aktualizacja ostatniego stanu przycisku
  delay(100); // Krótkie opóźnienie dla debouncing
}
void updateDisplayAndConsole() {
  lcd.clear(); // Wyczyszczenie wyświetlacza
  lcd.setCursor(0, 0); // Ustawienie kursora na początku pierwszej linii
  if (relayState) {
    lcd.print("PrzeKa"); // Wyświetlenie tekstu na LCD
    lcd.write(byte(2)); // Wstawienie znaku 'Z'
    lcd.print("nik ");
  } else {
    lcd.print("PrzeKa");
    lcd.write(byte(2)); // Wstawienie znaku 'Z'
    lcd.print("nik ");
  }
  lcd.setCursor(0, 1); // Ustawienie kursora na początku drugiej linii
  if (relayState) {
    lcd.print("Za");
    lcd.write(byte(0)); // Wstawienie znaku 'I'
    lcd.write(byte(1)); // Wstawienie znaku 'a'
    lcd.print("conony");
    Serial.println("Przełącznik załączony");
  } else {
    lcd.print("Wy");
    lcd.write(byte(0)); // Wstawienie znaku 'I'
    lcd.write(byte(1)); // Wstawienie znaku 'a'
    lcd.print("conony");
    Serial.println("Przełącznik wyłączony");
  }
}
}
```

Kompilacja zakończona

Szkic używa 4772 bajtów (14%) pamięci programu. Maksimum to 32256 bajtów.
Zmienne globalne używają 528 bajtów (25%) pamięci dynamicznej, pozostawiając 1520 bajtów dla zmiennych lokalnych. Maksimum to 2048 bajtów.

41 Arduino Uno na COM8

Wgranie programu (sketchu) do płytki Arduino UNO R3 za pomocą Arduino IDE obejmuje kilka kroków:

1. Podłączenie Arduino do komputera:

- Użycie kabla USB do podłączenia Arduino UNO R3 do portu USB komputera zapewnia fizyczne połączenie potrzebne do programowania płytki.

2. Otwarcie Arduino IDE:

- Uruchomienie Arduino IDE na komputerze pozwala na dostęp do narzędzi niezbędnych do skompilowania i wgrania kodu do mikrokontrolera.

3. Wybranie Płytki:

- W narzędziach Arduino IDE wybranie opcji "Arduino/Genuino UNO" jako docelowej płytki, na której zostanie uruchomiony program.

4. Wybranie Portu:

- Wybranie portu, do którego podłączone jest Arduino (COM na Windows lub /dev/tty na macOS i Linux), jest kluczowe dla komunikacji między komputerem a mikrokontrolerem.

5. Wgranie Programu:

- Skopiowanie programu (sketchu) do nowego okna w Arduino IDE.
- Naciśnięcie przycisku "Weryfikuj" (ikona z symbolem kreski zakończonej znakiem "✓") w górnym lewym rogu pozwoli skompilować kod i sprawdzić, czy nie ma błędów.
- Naciśnięcie przycisku "Wgraj" (ikona strzałki wskazującej w prawo), aby wgrać skompilowany program na płytkę Arduino.

6. Czekanie na Zakończenie:

- Arduino IDE pokaże postęp wgrywania na dole okna. Po zakończeniu powinien pojawić się komunikat "Wgrano".

7. Testowanie:

- **Uruchomienie:** Po pomyślnym wgraniu programu, Arduino automatycznie zrestartuje się i zacznie wykonywać wgrany program.
- **Weryfikacja działania:** Sprawdzenie, czy wyświetlacz LCD pokazuje odpowiednie komunikaty ("Przełącznik załączony" lub "Przełącznik wyłączony"), a diody LED odpowiednio reagują na zmiany stanu przełącznika.

