

Sprawozdanie z Zadania 3.

Sterowanie diod LED

Adrian Lesniak nr.indeksu: 154256

| | |
|--|-----------|
| 1. Cel Ćwiczenia | 3 |
| 2. Opis Elementów Wykorzystanych w Ćwiczeniu | 3 |
| Załączona Dokumentacja Techniczna - Podstawowe parametry. | 4 |
| Budowa Układu | 5 |
| 3. Schemat Połączeń | 6 |
| 4. Obliczenia Rezystancji dla Diod. | 7 |
| 5. Kod Programu z Komentarzami | 8 |
| Projekt składa się z kilku kroków: | 9 |
| 7. Uwagi i Wnioski | 12 |
| 8. Wykonanie projektu na odpowiednich komponentach. | 13 |

1. Cel Ćwiczenia

Celem ćwiczenia było zaprojektowanie układu, który pozwala na kontrolę trzech diod LED (czerwonej, żółtej, zielonej) za pomocą pojedynczego przycisku monostabilnego. Układ wykorzystuje Arduino do cyklicznego przełączania diod oraz wyświetlania odpowiednich komunikatów na ekranie LCD z użyciem polskich znaków.

2. Opis Elementów Wykorzystanych w Ćwiczeniu

Projekt wykorzystuje płytkę 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 trzech 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 każdą diodę LED, co zapobiega ich uszkodzeniu przez nadmierny prąd. Użycie jednolitych rezystorów upraszcza konstrukcję i pozwala na łatwe obliczenia oraz montaż układu. Przycisk monostabilny, użyty do sterowania sekwencją diod, pozwala na manualne przełączanie między stanami diod, co jest odczytywane z osobnego wejścia cyfrowego na Arduino.

Wyświetlacz LCD 1602 z interfejsem I2C służy do wyświetlania komunikatów o stanie diod w języku polskim, z zastosowaniem polskich znaków. Płytkę stykową (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 diodami.
- Dioda LED: Czerwona, żółta, zielona – każda dioda podłączona przez rezystor 220Ω, które ograniczają prąd płynący przez diody.
- Wyświetlacz LCD 1602 z interfejsem I2C: Służy do wyświetlania statusu diod.
- Rezystory 220Ω: Ograniczają prąd diod LED, zapobiegając ich uszkodzeniu.
- Przycisk monostabilny: Służy do sterowania zmianą stanów diod.
- Płytkę stykową: Używana do prototypowania układu.

Rys 1. Zdjęcie Arduino UNO R3



Załączona Dokumentacja Techniczna - **Podstawowe parametry.**

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 |

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 |

Budowa Układu

Układ zawiera Arduino UNO połączone z trzema diodami LED poprzez rezystory 220Ω do pinów cyfrowych 6, 9 i 5. 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 7 i do masy (GND), wykorzystując wewnętrzny rezystor podciągający Arduino.

Połączenia Arduino UNO R3 i wyświetlacza LCD podłączonych do siebie 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.

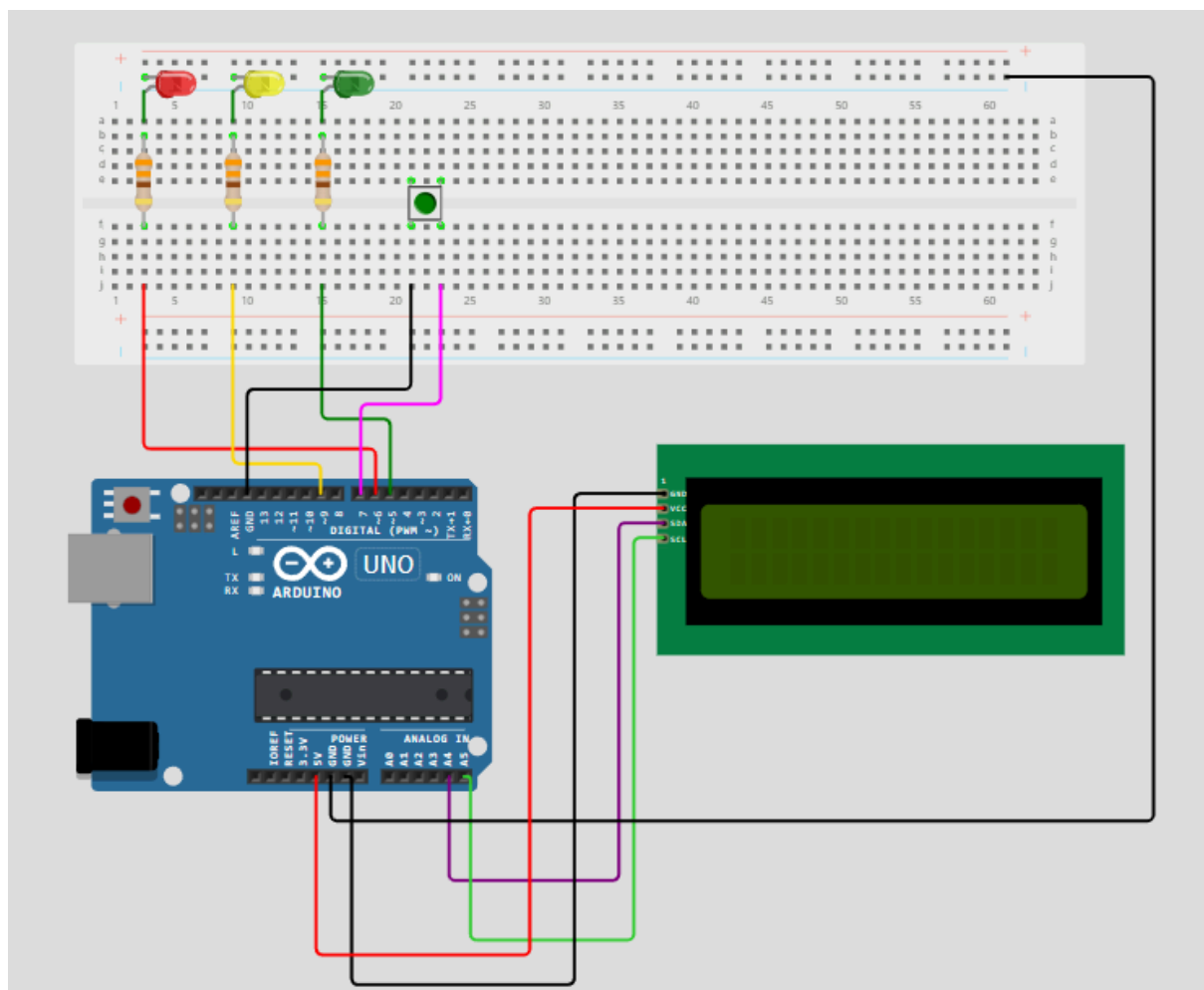
Dzięki wykorzystaniu szyny I2C, minimalizowana jest liczba potrzebnych przewodów, co upraszcza konstrukcję układu. Schemat połączeń ilustruje wszystkie połączenia, które są niezbędne do prawidłowego działania układu.

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

Schemat połączeń ilustruje wszystkie połączenia, które są niezbędne do prawidłowego działania układu oraz sposób połączenia Arduino UNO R3 z wyświetlaczem LCD za pomocą magistrali I2C oraz trzema diodami (czerwona, żółta i zielona), przełącznikiem monostabilnym i opornikami 220Ω.



Rys 2. Schemat połączeń zawiera Arduino UNO podłączone do diod LED przez rezystory oraz do wyświetlacza LCD za pomocą interfejsu I2C. Diodom przypisane są piny cyfrowe odpowiednio 6, 9, i 5, a przycisk monostabilny do pinu 7.

4. Obliczenia Rezystancji dla Diod.

Dla diod LED zastosowano rezystory o wartości 220Ω , co przy standardowym napięciu przewodzenia dla diod czerwonej i żółtej wynoszącym około 2V, oraz zielonej około 3V, pozwala na bezpieczne użytkowanie przy prądzie około 10-20 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 (typowo 2V dla czerwonej i żółtej, 3V dla zielonej), a I to prąd diody.

- Czerwona i Żółta: Przy założeniu, że spadek napięcia na diodzie wynosi 2V, a zalecany prąd diody wynosi 20 mA:

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

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

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

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

Podobnie, zastosowanie rezystora 220Ω jest odpowiednie dla bezpiecznej pracy.

Te obliczenia pokazują, że wartości rezystorów 220Ω są bezpieczne i odpowiednie dla wszystkich trzech 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 cykliczne załączanie i wyłączanie diod na przycisk, przy jednoczesnym wyświetlaniu statusu na LCD. Zawiera ochronę przed drganiem styków oraz przetrzymaniem przycisku. Kod jest szczegółowo skomentowany w języku polskim, co ułatwia zrozumienie działania programu.

Każda linijka kodu została skomentowana tak, aby wyjaśnić jej funkcję i rolę w programie.

Kod źródłowy został opracowany zgodnie z poniższym listingiem:

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

1  #include <Wire.h> // Dołącz bibliotekę Wire, używaną do komunikacji I2C
2  #include <LiquidCrystal_I2C.h> // Dołącz bibliotekę dla wyświetlacza LCD z interfejsem I2C
3
4  // Ustawienie adresu I2C dla LCD
5  LiquidCrystal_I2C lcd(0x27, 16, 2);
6
7  // Definicja własnych znaków dla polskich liter
8  byte customCharZ[8] = {B00100, B11111, B00001, B00010, B00100, B01000, B11111}; // 'Z'
9  byte customCharO[8] = {B00100, B00000, B01110, B10001, B10001, B01110, B00000}; // 'O'
10 byte customCharI[8] = {B01100, B00100, B00110, B00100, B01100, B00100, B01110, B00000}; // 'I'
11
12 const int redPin = 6; // Pin dla czerwonej diody LED
13 const int yellowPin = 9; // Pin dla żółtej diody LED
14 const int greenPin = 5; // Pin dla zielonej diody LED
15 const int buttonPin = 7; // Pin cyfrowy dla przycisku
16 bool lastButtonState = HIGH; // Zakładamy, że przycisk jest aktywowany do LOW
17 bool currentButtonState = HIGH; // Aktualny stan przycisku
18 unsigned long lastDebounceTime = 0; // Ostatni czas dla debouncingu
19 unsigned long debounceDelay = 50; // Opóźnienie dla debouncingu
20
21 int ledPins[] = {redPin, yellowPin, greenPin}; // Tablica pinów dla diod LED
22 String ledNames[] = {"Czerwona", "Żółta", "Zielona"}; // Nazwy diod LED
23 int currentLED = 0; // Aktualnie kontrolowana dioda LED
24 bool ledState = LOW; // Stan diody LED, LOW oznacza wyłączone
25 bool buttonPressed = false; // Dodana flaga do śledzenia, czy przycisk został już obsługiwany
26
27 void setup() {
28     lcd.init(); // Inicjalizacja LCD
29     lcd.backlight(); // Włączenie podświetlenia
30
31     // Utworzenie własnych znaków
32     lcd.createChar(0, customCharZ); // 'Z'
33     lcd.createChar(1, customCharO); // 'O'
34     lcd.createChar(2, customCharZ); // 'Z'
35
36     pinMode(buttonPin, INPUT_PULLUP); // Ustawienie pinu przycisku jako wejście z wewnętrznym rezystorem podciągającym
37     for (int i = 0; i < 3; i++) {
38         pinMode(ledPins[i], OUTPUT); // Ustawienie pinów LED jako wyjścia
39     }
40     Serial.begin(9600); // Rozpoczęcie komunikacji szeregowej
41 }
42
43 void loop() {
44     int reading = digitalRead(buttonPin); // Odczyt stanu przycisku
45
46     if (reading != lastButtonState) { // Jeśli stan przycisku się zmienił
47         lastDebounceTime = millis(); // Zaktualizuj czas debouncingu
48     }
49
50     if ((millis() - lastDebounceTime) > debounceDelay) { // Jeśli minęło wystarczająco dużo czasu od debouncingu
51         if (reading != currentButtonState) { // I stan przycisku się zmienił
52             currentButtonState = reading; // Zaktualizuj aktualny stan przycisku
```



```

53     if (currentButtonState == LOW && !buttonPressed) { // Jeśli przycisk jest wciśnięty i nie był wcześniej obsłużony
54         updateLEDs(); // Zaktualizuj stany diod LED
55         buttonPressed = true; // Ustaw flagę, że przycisk został obsłużony
56     }
57     else if (currentButtonState == HIGH) {
58         buttonPressed = false; // Reset flagi, gdy przycisk jest zwolniony
59     }
60 }
61
62 }
63
64 lastButtonState = reading; // Zaktualizuj ostatni odczytany stan przycisku
65 }
66
67 void updateLEDs() {
68     ledState = !ledState; // Zmiana stanu diody LED na przeciwny
69
70     // Wyłączenie wszystkich diod LED
71     for (int i = 0; i < 3; i++) {
72         digitalWrite(ledPins[i], LOW);
73     }
74
75     // Włączenie bieżącej diody LED
76     digitalWrite(ledPins[currentLED], ledState ? HIGH : LOW);
77
78     // Wysłanie informacji do Serial i LCD
79     lcd.clear();
80     lcd.setCursor(0, 0);
81
82     // Sprawdź czy bieżąca dioda LED to żółta i wyświetl "Żółta"
83     if (currentLED == 1) { // Zakładając, że żółta ma indeks 1
84         lcd.write(byte(2)); // '2'
85         lcd.write(byte(1)); // 'd'
86         lcd.write(byte(0)); // '1'
87         lcd.print("ta");
88         Serial.println("Żółta " + String(ledState ? "ON" : "OFF"));
89     } else {
90         lcd.print(ledNames[currentLED]); // Wypisz nazwę diody na LCD
91         Serial.print(ledNames[currentLED]); // Wypisz nazwę diody na port szeregowy
92         Serial.println(ledState ? " ON" : " OFF");
93     }
94
95     lcd.print(ledState ? " - ON" : " - OFF"); // Dodaj informację o stanie diody na LCD
96
97     if (!ledState) {
98         currentLED = (currentLED + 1) % 3; // Przejdź do następnej diody tylko podczas wyłączania
99     }
100 }
101

```

Szczegółowe omówienie dostarczonego kodu z uwzględnieniem komentarzy wyjaśniających funkcję każdej linii. Komentarze powinny wyjaśniać logikę przełączania diod i wyświetlania komunikatów.

Projekt składa się z kilku kroków:

1. 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 skontrolować diody LED poprzez cyfrowe piny wyjściowe, z których każdy steruje jedną diodą LED przez szeregowo połączony rezystor 220Ω.

2. Konfiguracja pinów i początkowe ustawienia diod LED:

Piny cyfrowe dla diod (6, 9, 5 odpowiednio dla czerwonej, żółtej i zielonej) zostały skonfigurowane jako wyjścia w funkcji `setup()` w kodzie Arduino. Na początku działania programu ustawiono wszystkie diody w stanie wyłączonym (OFF).

3. 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.

4. Zaprogramowanie cyklicznego przełączania diod:

Program dynamicznie przełącza stany diod z ON na OFF i odwrotnie, przesuwając cykl między czerwoną, żółtą i zieloną diodą po każdym naciśnięciu przycisku. Do zarządzania stanami diod wykorzystano funkcję `millis()`, co pozwala na nieliniowy sposób zarządzania czasem bez użycia funkcji `delay()`.

5. Aktualizacja wyświetlacza LCD i komunikaty w monitorze szeregowym:

W każdym cyklu, kiedy dioda zmienia stan, wyświetlacz LCD jest aktualizowany, aby pokazywać nazwę i stan aktualnie aktywnej diody (np. "Żółta - ON"). Równocześnie, odpowiedni komunikat jest wysyłany do monitora portu szeregowego, co umożliwia śledzenie działania układu także przez port szeregowy Arduino.

6. Monitorowanie stanu przycisku i zarządzanie stanem diod:

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

7. Testowanie i debugowanie programu:

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

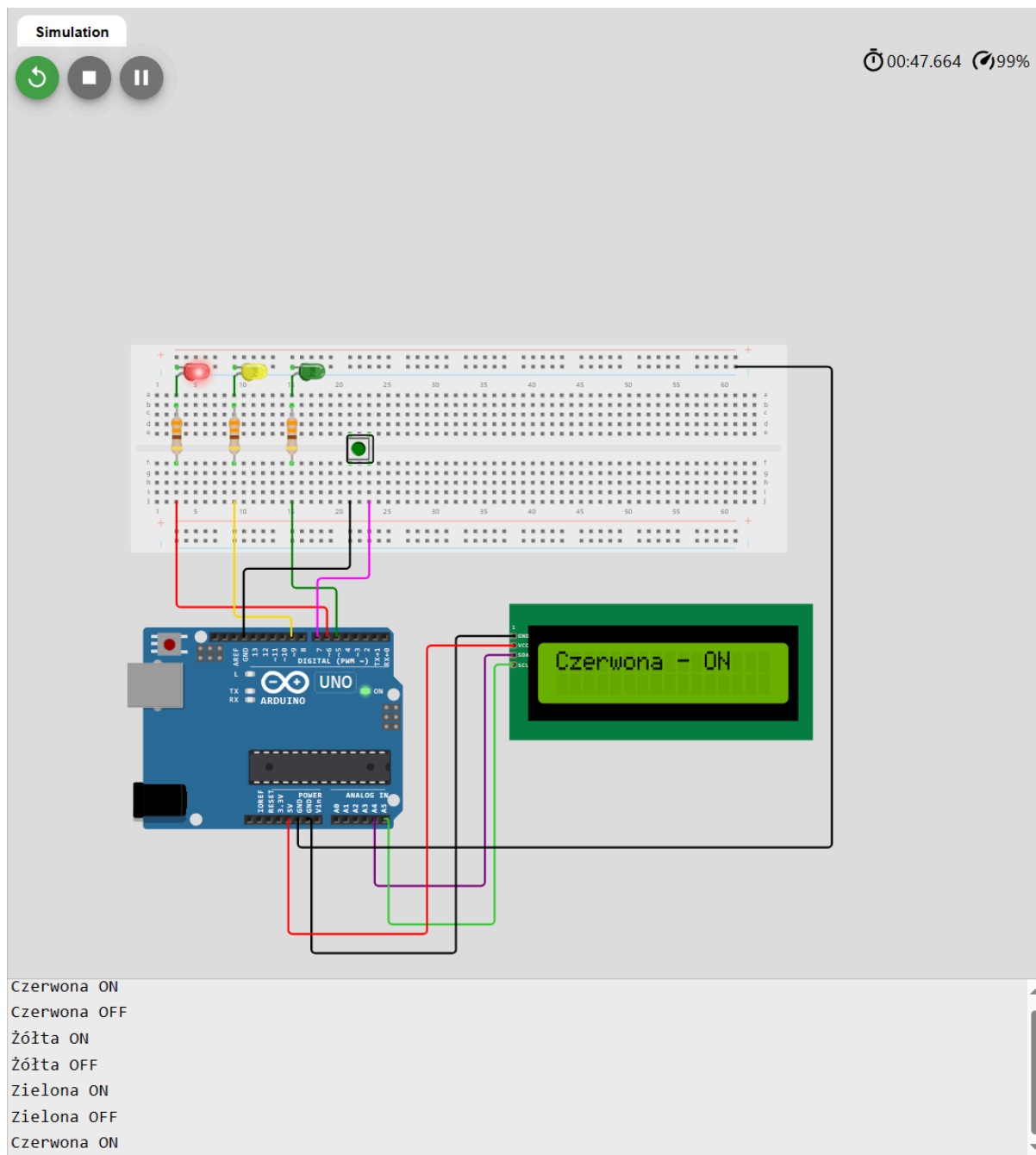
W ramach projektu zaimplementowano systematyczne podejście do budowy i programowania układu, co pozwoliło na głębsze zrozumienie zarówno hardware'u, jak i software'u używanego w projekcie. Użycie funkcji opartych o czas zamiast blokującej funkcji `delay()` znacząco poprawia wydajność i reaktywność systemu.

6. Testowanie i Uruchomienie Układu.

Układ był testowany w środowisku TinkerCad, gdzie można było wizualnie zweryfikować poprawność działania diod oraz wyświetlacza. Każde naciśnięcie przycisku monostabilnego zmienia stan aktualnie aktywnej diody oraz aktualizuje wyświetlacz LCD, pokazujący nazwę i stan diody.

- Działanie programu dla zadania 3, które obejmuje sterowanie diodami LED 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 diodami są ustawione jako wyjścia, a wszystkie diody są początkowo wyłączone.
- Program korzysta z funkcji `millis()` do śledzenia upływu czasu, co umożliwia kontrolowanie czasu przełączania diod bez zatrzymywania działania kodu funkcją `delay()`. Po każdym naciśnięciu przycisku, aktualnie świecąca dioda jest wyłączana, a następnie kolejna dioda w sekwencji (czerwona, żółta, zielona) jest załączana. Każda zmiana stanu diody jest rejestrowana przez program, który aktualizuje informacje na wyświetlaczu LCD i wysyła odpowiednie komunikaty do monitora portu szeregowego, informując o stanie diod, takie jak "Żółta - ON" czy "Zielona - OFF".
- 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 diod 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 , diodami , płytka stykowa, przełącznikiem monostabilnym i rezystorami.

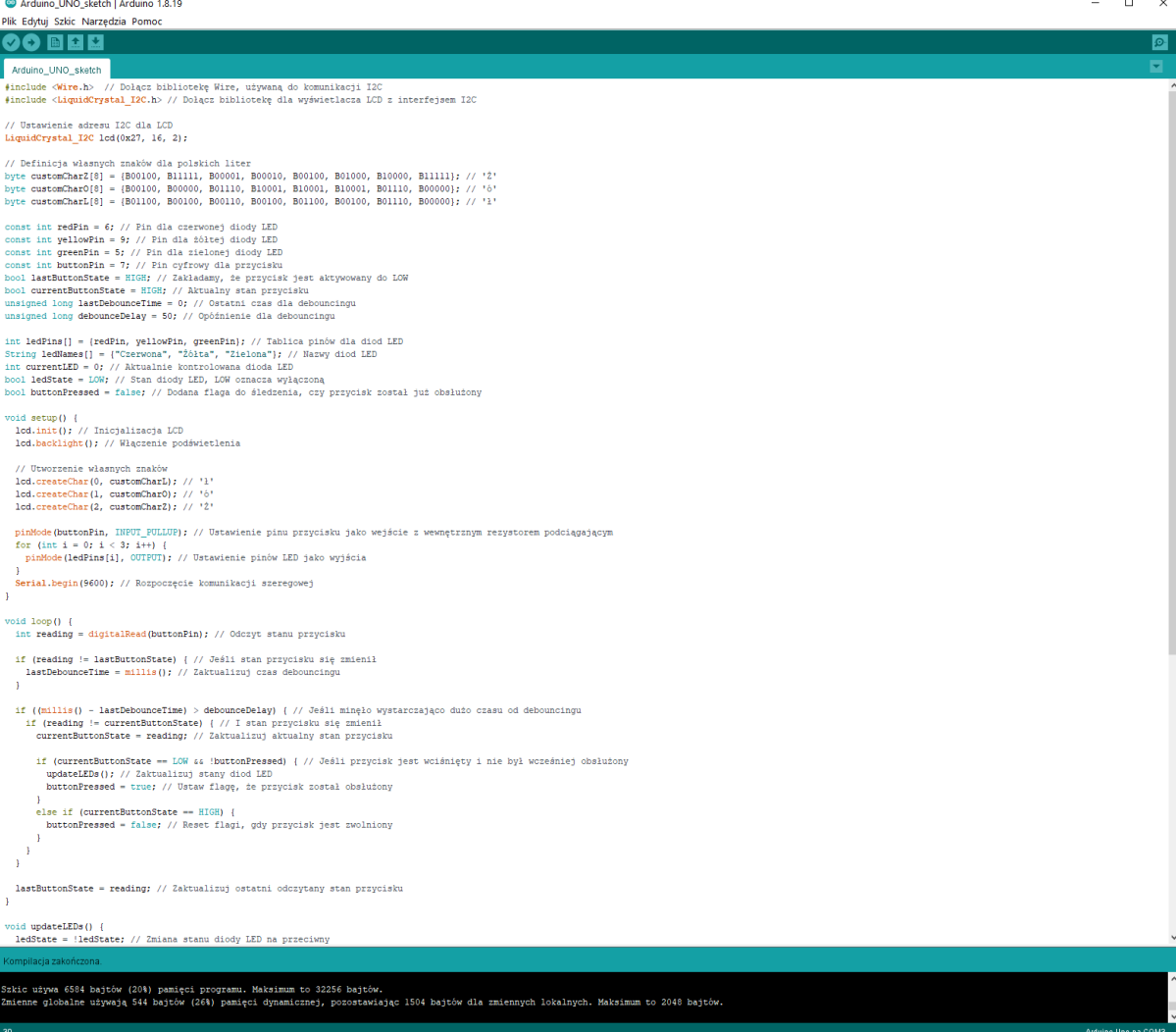


7. Uwagi i Wnioski

1. Projekt umożliwił zrozumienie implementacji debouncingu oraz ochrony przed przetrzymaniem przycisku, co znacznie zwiększa niezawodność działania układu.

2. Dzięki użyciu wyświetlacza LCD z interfejsem I2C, komunikacja z wyświetlaczem była uproszczona, a liczba przewodów zmniejszona.
3. Programowanie mikrokontrolera do realizacji złożonych zadań czasowych bez użycia funkcji delay() pozwoliło na lepszą responsywność i efektywność układu.
4. Projekt oferuje potencjał dalszej rozbudowy, np. o zdalne sterowanie lub dodatkowe czujniki.

8. Wykonanie projektu na odpowiednich komponentach.



```
Arduino_UNO_sketch | Arduino 1.8.19
Plik Edytuj Szukaj Narzędzia Pomoc

Arduino_UNO_sketch

#include <Wire.h> // Dołącz bibliotekę Wire, używaną do komunikacji I2C
#include <LiquidCrystal_I2C.h> // Dołącz bibliotekę dla wyświetlacza LCD z interfejsem I2C

// Ustawienie adresu I2C dla LCD
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Definicja własnych znaków dla polskich liter
byte customChar[8] = {B00100, B11111, B00001, B00010, B00100, B01000, B10000, B11111}; // '2'
byte customChar[8] = {B00100, B00000, B01110, B10001, B10001, B01110, B00000}; // 'ó'
byte customChar[8] = {B01100, B00100, B00110, B00100, B01100, B00100, B01110, B00000}; // '1'

const int redPin = 6; // Pin dla czerwonej diody LED
const int yellowPin = 9; // Pin dla żółtej diody LED
const int greenPin = 5; // Pin dla zielonej diody LED
const int buttonPin = 7; // Pin cyfrowy dla przycisku
bool lastButtonState = HIGH; // Zakładamy, że przycisk jest aktywowany do LOW
bool currentButtonState = HIGH; // Aktualny stan przycisku
unsigned long lastDebounceTime = 0; // Ostatni czas dla debouncingu
unsigned long debounceDelay = 50; // Opóźnienie dla debouncingu

int ledPins[] = {redPin, yellowPin, greenPin}; // Tablica pinów dla diod LED
String ledNames[] = {"Czerwona", "Żółta", "Zielona"}; // Nazwy diod LED
int currentLED = 0; // Aktualnie kontrolowana dioda LED
bool ledState = LOW; // Stan diody LED, LOW oznacza wyłączone
bool buttonPressed = false; // Dodana flaga do śledzenia, czy przycisk został już obsłużony

void setup() {
  lcd.init(); // Inicjalizacja LCD
  lcd.backlight(); // Włączenie podświetlenia

  // Utworzenie własnych znaków
  lcd.createChar(0, customChar); // '1'
  lcd.createChar(1, customChar); // 'ó'
  lcd.createChar(2, customChar); // '2'

  pinMode(buttonPin, INPUT_PULLUP); // Ustawienie pinu przycisku jako wejście z wewnętrznym rezystorem podciągającym
  for (int i = 0; i < 3; i++) {
    pinMode(ledPins[i], OUTPUT); // Ustawienie pinów LED jako wyjścia
  }
  Serial.begin(9600); // Rozpoczęcie komunikacji szeregowej
}

void loop() {
  int reading = digitalRead(buttonPin); // Odczyt stanu przycisku

  if (reading != lastButtonState) { // Jeśli stan przycisku się zmienił
    lastDebounceTime = millis(); // Zaktualizuj czas debouncingu
  }

  if ((millis() - lastDebounceTime) > debounceDelay) { // Jeśli minęło wystarczająco dużo czasu od debouncingu
    if (reading != currentButtonState) { // I stan przycisku się zmienił
      currentButtonState = reading; // Zaktualizuj aktualny stan przycisku

      if (currentButtonState == LOW && !buttonPressed) { // Jeśli przycisk jest wciśnięty i nie był wcześniej obsłużony
        updateLEDs(); // Zaktualizuj stany diod LED
        buttonPressed = true; // Ustaw flagę, że przycisk został obsłużony
      }
      else if (currentButtonState == HIGH) {
        buttonPressed = false; // Reset flagi, gdy przycisk jest zwolniony
      }
    }
  }

  lastButtonState = reading; // Zaktualizuj ostatni odczytany stan przycisku
}

void updateLEDs() {
  ledState = !ledState; // Zmiana stanu diody LED na przeciwny
}
```

Kompilacja zakończona

Skic używa 6584 bajtów (20%) pamięci programu. Maksimum to 32256 bajtów.
Zmienne globalne używają 544 bajtów (24%) pamięci dynamicznej, pozostawiając 1504 bajtów dla zmiennych lokalnych. Maksimum to 2048 bajtów.

30 Arduino Uno na COM5

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.

2. Otwarcie Arduino IDE:

- Uruchomienie Arduino IDE na komputerze.

3. Wybranie Płytki:

- W narzędziach Arduino IDE wybranie "Arduino/Genuino UNO".

4. Wybranie Portu:

- W narzędziach Arduino IDE wybranie portu, do którego podłączone jest Arduino (COM na Windows lub /dev/tty na macOS i Linux).

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:

- Po pomyślnym wgraniu programu, Arduino automatycznie zrestartuje się i zacznie wykonywać wgrany program.
- Sprawdzenie, czy wyświetlacz LCD pokazuje odpowiedni tekst.

