Wydział
Budowy Maszyn
i Informatyki
Uniwersytet Bielsko-Bialski

PROGRAMOWANIE MIKROKONTOLERÓW

(ćwiczenia laboratoryjne)

Sprawozdanie z zajęć laboratoryjnych

Wykonali:
1. Adam Kubliński

Katedra Informatyki i Automatyki
Bielsko – Biała 2024

Spis treści

1. Wprowadzenie teoretyczne: Mikrokontrolery Arduino

Mikrokontrolery Arduino to uniwersalne platformy mikroprocesorowe oparte na otwartym oprogramowaniu i sprzęcie. Są one szeroko stosowane w edukacji, prototypowaniu oraz projektach inżynierskich dzięki swojej prostocie, elastyczności i przystępności. W ramach zajęć laboratoryjnych korzystaliśmy z modeli Arduino Nano i Arduino Uno, które umożliwiają realizację różnorodnych zadań związanych z wejściami/wyjściami cyfrowymi, analogowymi oraz komunikacją bezprzewodową.

1.1. Architektura mikrokontrolerów Arduino

Arduino opiera się na mikrokontrolerach z rodziny ATmega (np. ATmega328P). Główne cechy charakterystyczne:

- Taktowanie procesora: 16 MHz.
- Pamięć Flash: 32 KB do przechowywania kodu programu.
- Pamięć SRAM: 2 KB do przechowywania danych zmiennych w trakcie działania programu.
- Pamięć EEPROM: 1 KB do trwałego przechowywania danych (np. konfiguracji).
- Napięcie zasilania: 5V (czasami 3.3V w zależności od modelu).
- Interfejsy komunikacyjne: UART, I2C, SPI.

1.2. Opis pinów Arduino

Mikrokontrolery Arduino wyposażone są w piny, które pełnią różnorodne funkcje. W przypadku modelu Arduino Nano mamy:

Piny cyfrowe (D0-D13):

- Wykorzystywane do operacji wejścia/wyjścia cyfrowego.
- Wybrane piny (np. D3, D5, D6, D9, D10, D11) obsługują PWM (Pulse Width Modulation), co umożliwia generowanie sygnałów analogowych.

Piny analogowe (A0-A7):

- Służą do odczytu sygnałów analogowych (0-1023), które można wykorzystać np. do odczytu z czujników.
- Piny te mogą być również używane jako dodatkowe wejścia/wyjścia cyfrowe.

Pin zasilania:

- Vin: Zasilanie zewnętrzne dla mikrokontrolera.
- 5V i 3.3V: Wyjścia zasilania do podłączenia komponentów peryferyjnych.
- GND: Masa układu.

Piny komunikacyjne:

- RX (D0) i TX (D1): Służą do komunikacji szeregowej (UART), np. z komputerem lub modułami Bluetooth.
- SPI (D10-D13): Piny dedykowane do komunikacji szeregowej z urządzeniami peryferyjnymi, takimi jak wyświetlacze lub pamięci.
- I2C (A4 - SDA, A5 - SCL): Magistrala komunikacyjna wykorzystywana np. do obsługi czujników.

1.3. Zastosowania i zalety Arduino

Arduino jest idealną platformą do nauki podstaw elektroniki oraz tworzenia zaawansowanych projektów prototypowych. Do jego głównych zalet należą:

- Intuicyjne środowisko programistyczne (Arduino IDE).
- Ogromna społeczność użytkowników i dostępność bibliotek.
- Łatwość integracji z szeroką gamą czujników i modułów, takich jak Bluetooth, Wi-Fi, RF czy GPS.
- Możliwość zasilania zarówno z portu USB, jak i zewnętrznego źródła.

W tej części sprawozdania przedstawiono ogólne informacje teoretyczne dotyczące mikrokontrolerów Arduino, w szczególności opis ich parametrów oraz funkcji poszczególnych pinów. W kolejnych sekcjach omówiono praktyczne zastosowania Arduino w realizacji zadań laboratoryjnych.

2. Laboratorium 1 – Porty wejść / wyjść GPIO

2.1. Cel ćwiczenia

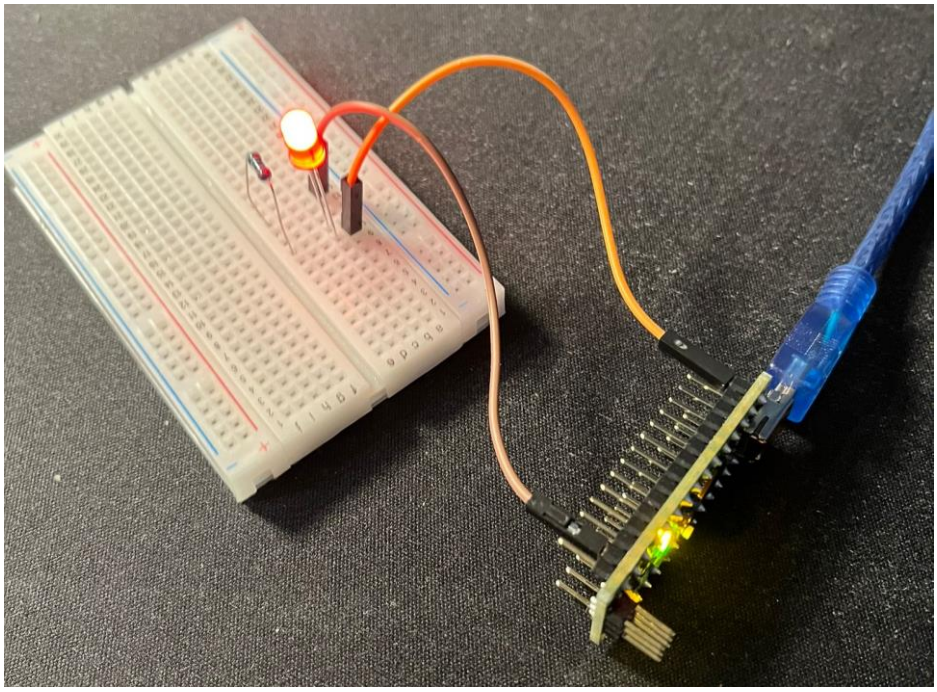
Celem ćwiczenia jest zapoznanie się z mikrokontrolerem Arduino Nano, jego portami wejścia / wyjścia tzw. GPIO (General Port Input Output) oraz roli poszczególnych elementów takich jak LED-y, przyciski. Zdobycie umiejętności przesyłania kodu do mikrokontrolera i testowania programu.

2.2. Zadanie 1: Miganie diodą LED

Twoim zadaniem jest napisanie programu, który sprawi, że dioda LED będzie migać w regularnych odstępach czasu (0,5 sekundy).

Program 1: Zadanie 1 Laboratorium 1

```
// Definicja pinu LED
const int ledPin = 13;
void setup() {
// Ustawienie pinu LED jako wyjście
pinMode(ledPin, OUTPUT);
}
void loop() {
// Włączenie LED
digitalWrite(ledPin, HIGH);
delay(500); // Opóźnienie 500 ms
// Wyłączenie LED
digitalWrite(ledPin, LOW);
delay(500); // Opóźnienie 500 ms
}
```



Rysunek 1: Podgląd układu do Zadania 1

Podsumowanie:

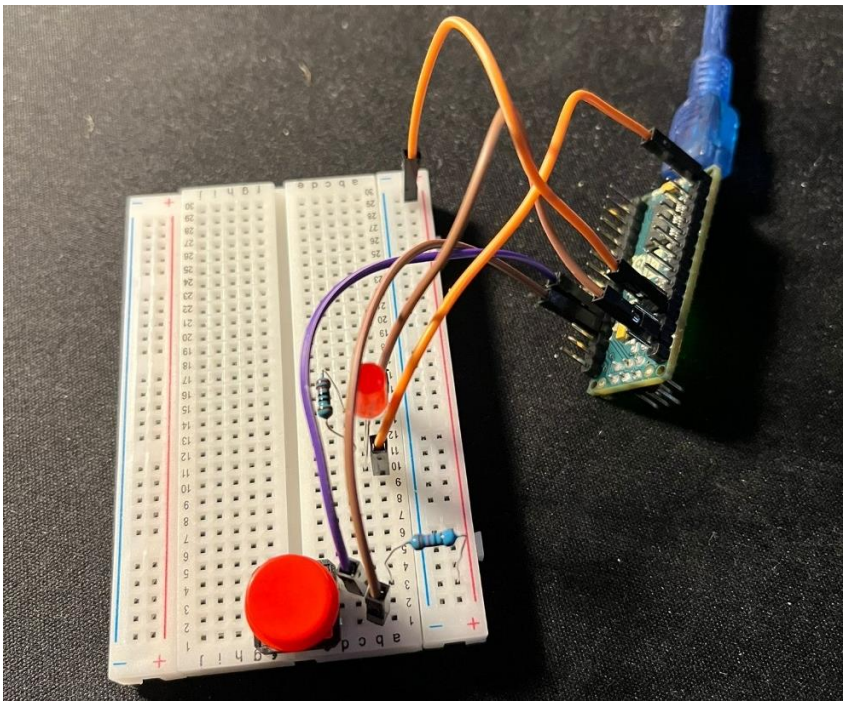
W tym zadaniu zaprogramowano diodę LED, aby migała w regularnych odstępach czasu co 0,5 sekundy. Zadanie pozwoliło zrozumieć podstawowe operacje na portach GPIO, takie jak ustawianie pinu jako wyjścia i manipulacja jego stanem logicznym. Zdobyto podstawowe umiejętności sterowania diodą LED i opóźniania operacji w mikrokontrolerze.

2.3. Zadanie 2: Sterowanie diodą za pomocą przycisku

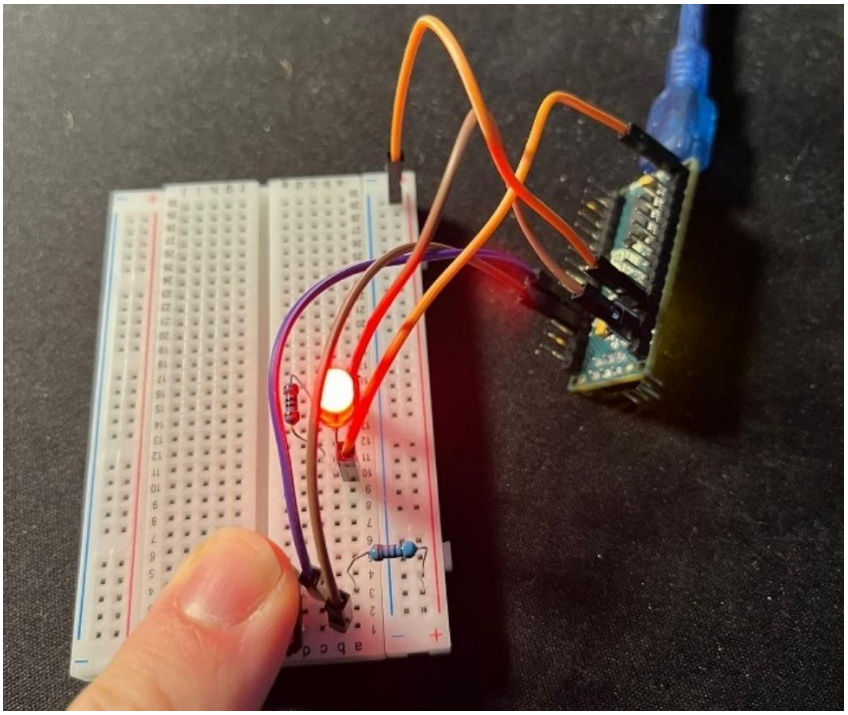
Napisz program, który umożliwi zapalanie diody LED po wciśnięciu przycisku.

Program 2: Zadanie 2 Laboratorium 1

```
// Definicja pinów
const int ledPin = 13;
const int buttonPin = 2;
void setup() {
// Ustawienie pinu LED jako wyjście
pinMode(ledPin, OUTPUT);
// Ustawienie pinu przycisku jako wejście
pinMode(buttonPin, INPUT);
}
void loop() {
// Odczyt stanu przycisku
int buttonState = digitalRead(buttonPin);
// Sprawdzenie stanu przycisku
if (buttonState == LOW) {
// Włączenie LED
digitalWrite(ledPin, HIGH);
} else {
// Wyłączenie LED
digitalWrite(ledPin, LOW);
}
}
```



Rysunek 2: Podgląd układu stan spoczynku



Rysunek 3: Podgląd układu stan uruchomiony

Podsumowanie:

W tym zadaniu zaprogramowano diodę LED, która zapala się po wciśnięciu przycisku. Zadanie pokazało, jak odczytywać stany wejść GPIO. Napotkany problem z odwrotnym zachowaniem diody (światło włączało się w stanie HIGH) został rozwiązany poprzez zmianę kodu na sprawdzanie stanu LOW. Zdobyto wiedzę, jak obsługiwać wejścia cyfrowe, odczytać stan przycisku i reagować na jego wciśnięcie.

2.4. Zadanie 3: Tryby migania diody

Zadaniem jest zaprogramowanie kontrolera tak, aby pojedyncze wciśnięcie przycisku zmieniało tryb działania diody LED.

- Tryb 1: Dioda stale wyłączona.
- Tryb 2: Dioda miga co 1 sekundę.
- Tryb 3: Dioda miga szybko (0,2 sekundy).

Program 3: Zadanie 3 Laboratorium 1

```
// Definicja pinów
const int ledPin = 13;       // Pin dla diody LED
const int buttonPin = 2;     // Pin dla przycisku
int mode = 0;                // Aktualny tryb (0, 1, 2)
bool buttonPressed = false;  // Stan przycisku
unsigned long lastTime = 0;  // Do kontroli migania diody
unsigned long interval = 1000;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

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

  // Sprawdzanie, czy przycisk został wciśnięty
  if (buttonState == LOW && !buttonPressed) {
    buttonPressed = true;
    mode = (mode + 1) % 3; // Przechodzenie między trybami 0, 1, 2
  } else if (buttonState == HIGH) {
    buttonPressed = false;
  }

  switch (mode) {
    case 0: // Tryb 1: Dioda stale wyłączona
      digitalWrite(ledPin, LOW);
      break;

    case 1: // Tryb 2: Dioda miga co 1 sekundę
      interval = 1000; // Czas migania 1 sekunda
      blinkLED();
      break;

    case 2: // Tryb 3: Dioda miga szybko (0,2 sekundy)
      interval = 200; // Czas migania 0,2 sekundy
      blinkLED();
      break;
```

```
  }
}

void blinkLED() {
  unsigned long currentTime = millis();
  if (currentTime - lastTime >= interval) {
    lastTime = currentTime;
    int ledState = digitalRead(ledPin);
    digitalWrite(ledPin, !ledState);
  }
}
```

Podsumowanie:

Rozwiązanie wykorzystało mechanizm pętli loop, odczytywania stanu przycisku oraz przełączania trybów za pomocą zmiennej sterującej. Wprowadzono funkcję pomocniczą do migania diodą w regularnych odstępach czasu. Nauczono się implementować różne tryby działania oraz obsługiwać dynamiczne przełączanie stanów

2.5. Laboratorium 1 – Podsumowanie

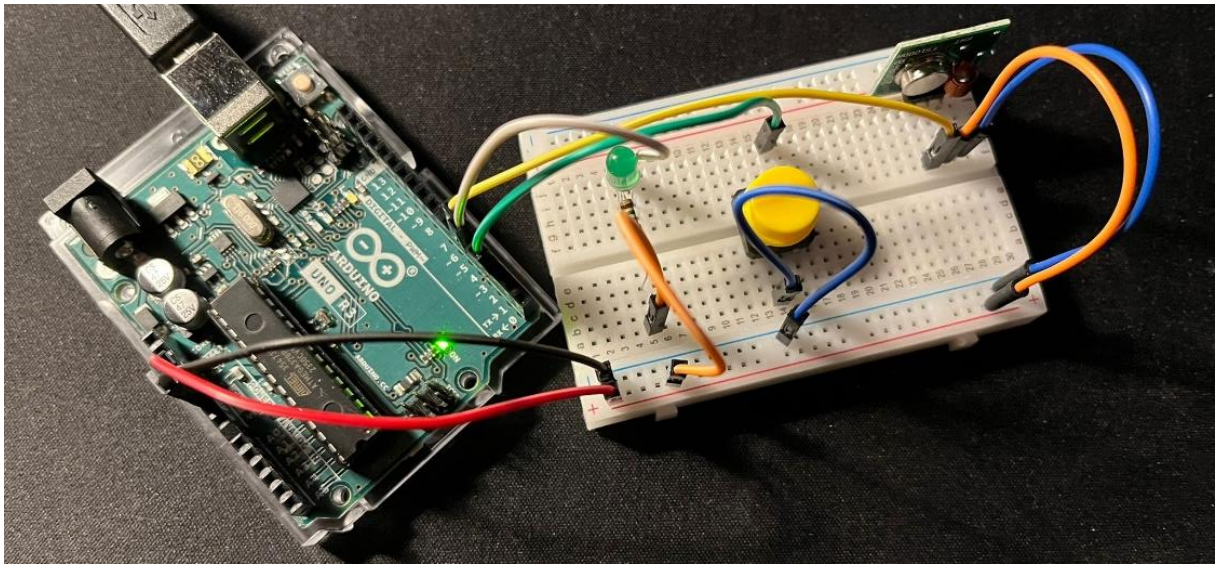
Laboratorium pozwoliło opanować podstawowe operacje na portach GPIO, obsługę wejść i wyjść oraz manipulację stanami logicznymi. Zrozumiano podstawy pracy z przyciskami, diodami LED oraz podstawowe zasady programowania dla mikrokontrolerów.

3. Laboratorium 2 - Moduł komunikacji radiowej RF - FS100A

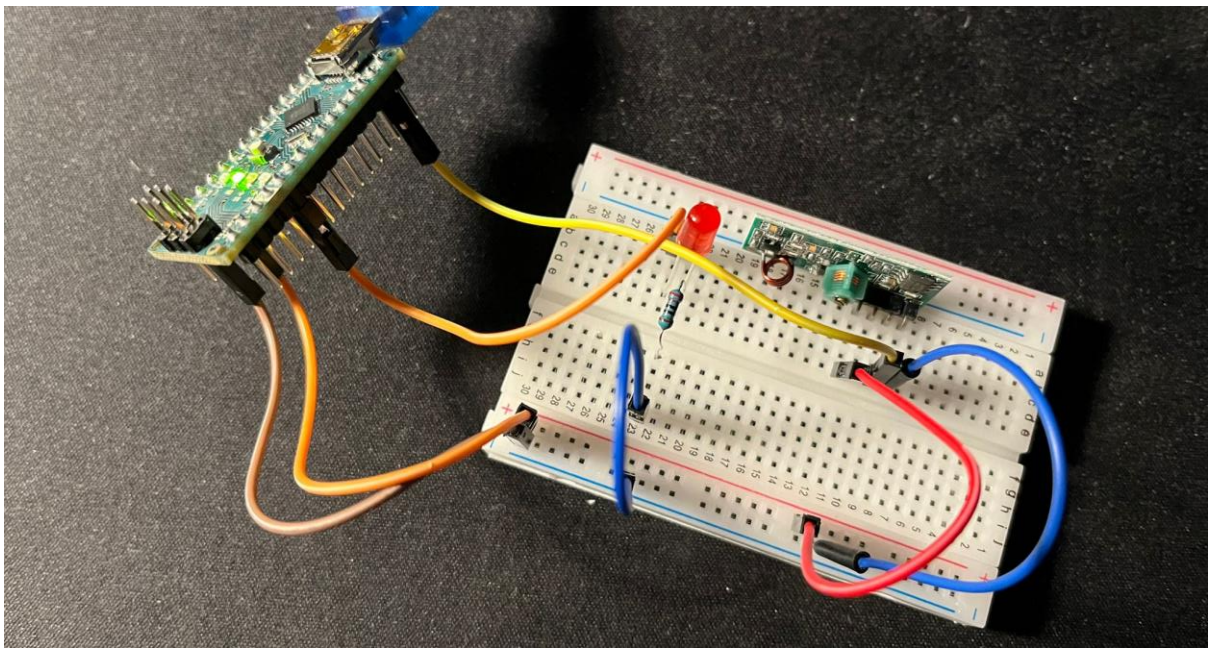
3.1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z zasadą działania, konfiguracją, podłączeniem oraz obsługą programową modułu odbiornika/nadajnika RF (Radio Frequency) w paśmie 433MHz umożliwiającego bezprzewodową transmisje danych. Moduł radiowy to nadajnik FS100A + odbiornik 433 MHz.

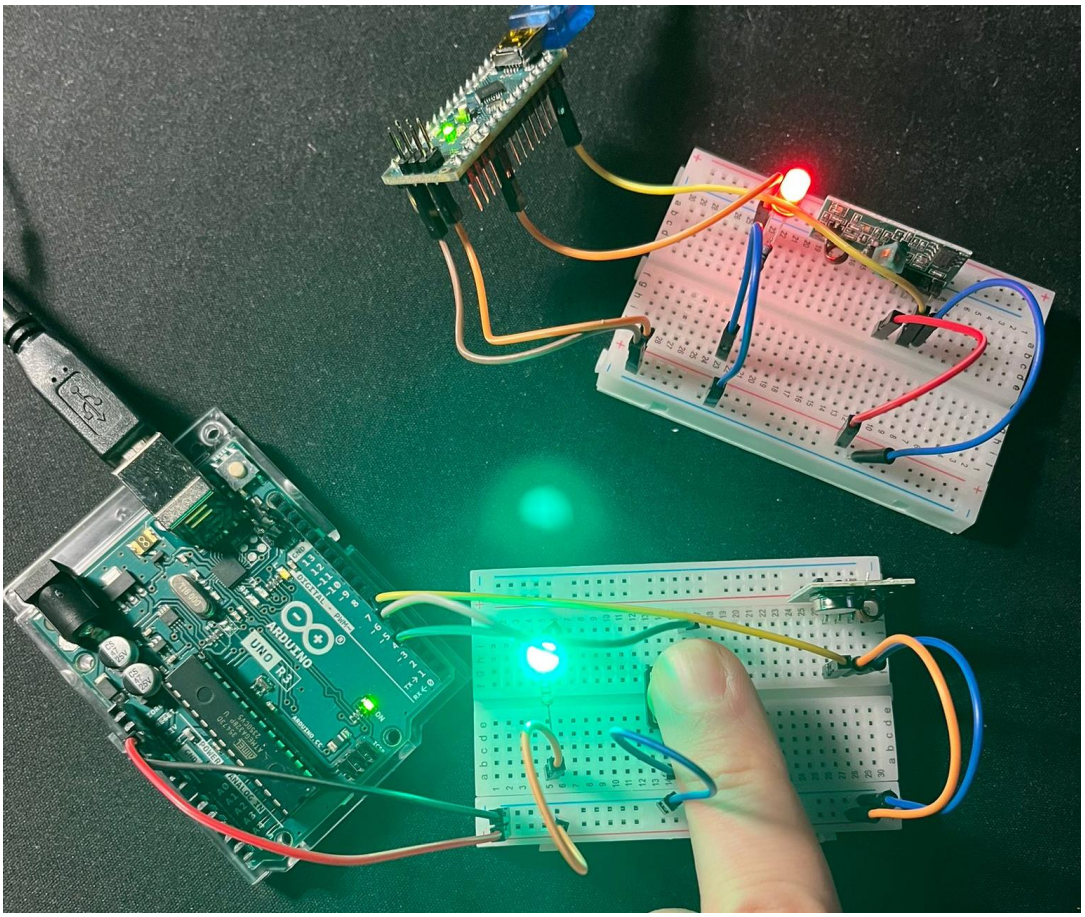
3.2. Zadanie 1 – Zbudowanie nadajnika i odbiornika do komunikacji radiowej



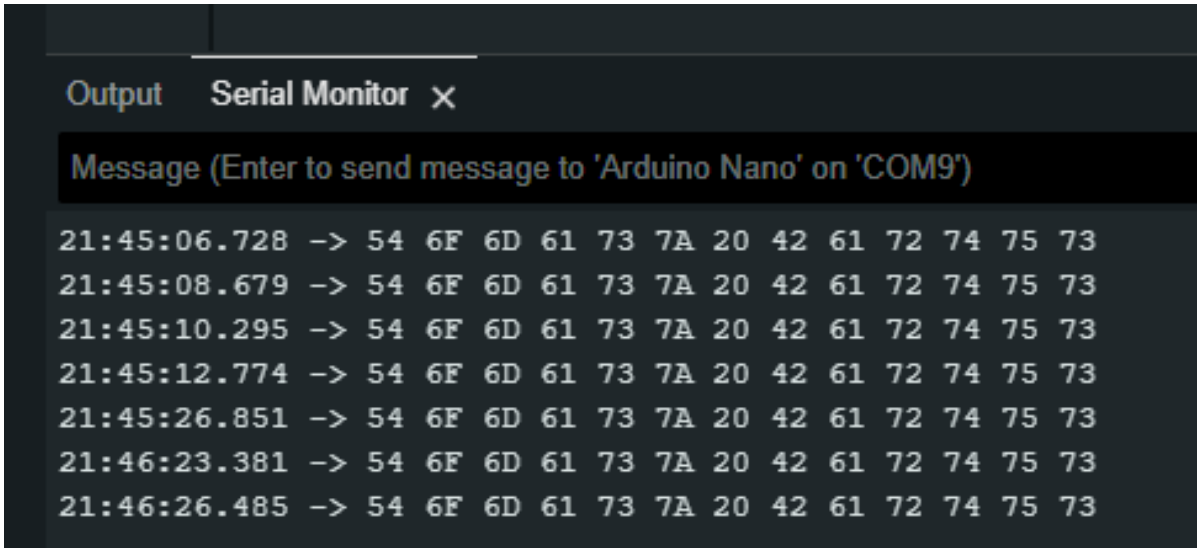
Rysunek 4: Podgląd układu nadajnika



Rysunek 5: Podgląd układu odbiornika



Rysunek 6: Układ w trakcie pracy



Rysunek 7: Odpowiedzi z Serial Monitor

Podsumowanie:

W tym zadaniu zapoznano się z podstawową konfiguracją modułów RF (FS100A i odbiornika 433 MHz). Zbudowano układ pozwalający na przesyłanie danych pomiędzy nadajnikiem a odbiornikiem. Nauczyłem się używać biblioteki VirtualWire do obsługi transmisji radiowej. Zdobyto umiejętność przesyłania danych pomiędzy dwoma mikrokontrolerami za pomocą sygnałów radiowych.

3.3. Zadanie 2 – Przesyłanie sygnału Live

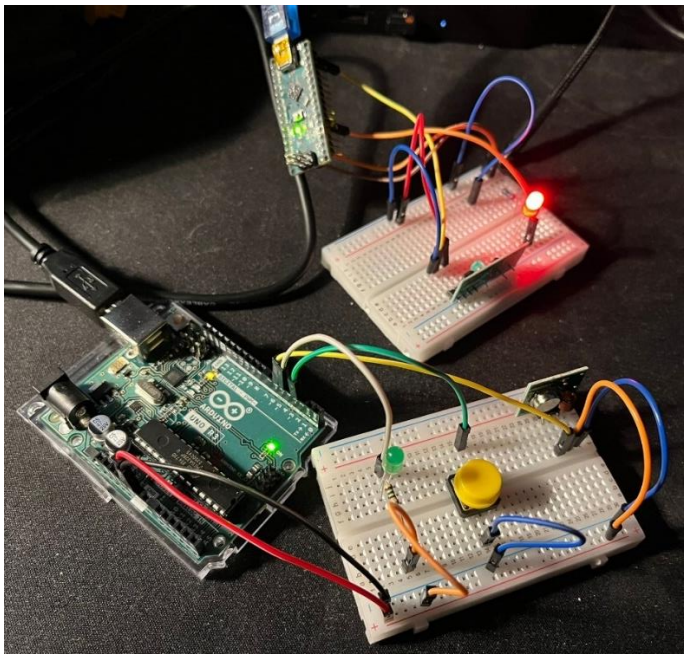
Zmodyfikować program tak, aby między nadajnikiem a odbiornikiem przesyłany był sygnał „Live bit” sygnalizujący ciągłość wykonywanej komunikacji.

Program 4: Kod programu nadajnika

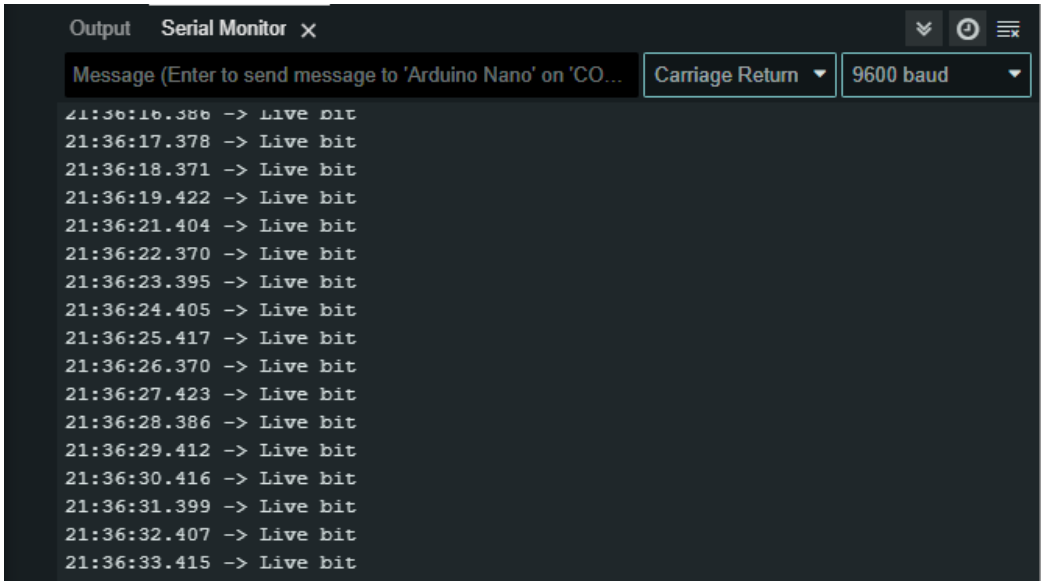
```
#include <VirtualWire.h> // Załącz bibliotekę VirtualWire
#define LED_PIN 5         // Pin diody LED
#define OK_PIN 3          // Pin przycisku
#define TRANSMIT_PIN 6    // Pin sygnału RF

unsigned long lastSendTime = 0;  // Zmienna do śledzenia czasu
const unsigned long interval = 1000; // Czas w ms między wysyłaniem Live
bit

void setup() {
  pinMode(OK_PIN, INPUT_PULLUP);  // Ustawienie przycisku jako wejście
  pinMode(LED_PIN, OUTPUT);       // Ustawienie diody jako wyjście
  vw_set_tx_pin(TRANSMIT_PIN);    // Ustawienie pinu RF
  vw_setup(2000);                 // Ustawienie prędkości transmisji
  digitalWrite(LED_PIN, HIGH);    // Wyłącz diodę na start
}

void loop() {
  unsigned long currentTime = millis(); // Pobranie bieżącego czasu

  // Wysyłanie Live bit co określony czas
  if (currentTime - lastSendTime >= interval) {
    String liveMessage = "Live bit";  // Sygnał Live bit
    char liveMsg[20];
    liveMessage.toCharArray(liveMsg, liveMessage.length() + 1);
    vw_send((uint8_t *)liveMsg, strlen(liveMsg));  // Wyślij wiadomość
    vw_wait_tx(); // Zaczekaj na zakończenie transmisji
    lastSendTime = currentTime;  // Aktualizacja czasu wysyłki
  }
  // Wysyłanie wiadomości po wciśnięciu przycisku
  if (digitalRead(OK_PIN) == LOW) {
    delay(20); // Debounce
    String message = "Adamadacho";  // Główna wiadomość
    char msg[20];
    message.toCharArray(msg, message.length() + 1);
    digitalWrite(LED_PIN, LOW); // Włącz diodę LED
    vw_send((uint8_t *)msg, strlen(msg));  // Wyślij wiadomość
    vw_wait_tx(); // Zaczekaj na zakończenie transmisji
    while (digitalRead(OK_PIN) == LOW) {
      delay(20); // Czekaj na zwolnienie przycisku
    }
    digitalWrite(LED_PIN, HIGH); // Wyłącz diodę LED
  }
}
```

Program 5: Kod programu odbiornika

```
#include <VirtualWire.h> // Załącz bibliotekę VirtualWire

#define LED_PIN 2         // Pin diody LED
#define RECEIVE_PIN 11    // Pin sygnału RF
unsigned long lastLiveBitTime = 0;  // Czas ostatniego Live bit
const unsigned long timeout = 3000; // Czas w ms do uznania utraty
komunikacji

void setup() {
  pinMode(LED_PIN, OUTPUT);     // Ustawienie diody jako wyjście
  Serial.begin(9600);           // Uruchomienie monitora portu szeregowego
  vw_set_rx_pin(RECEIVE_PIN);   // Ustawienie pinu RF
  vw_setup(2000);               // Ustawienie prędkości transmisji
  vw_rx_start();                // Start odbiornika
}

void loop() {
  uint8_t buf[VW_MAX_MESSAGE_LEN];    // Bufor na wiadomość
  uint8_t buflen = VW_MAX_MESSAGE_LEN; // Długość bufora

  if (vw_get_message(buf, &buflen)) { // Jeśli odebrano wiadomość
    digitalWrite(LED_PIN, HIGH);      // Włącz diodę LED
    String receivedMessage = "";
    for (int i = 0; i < buflen; i++) {
      receivedMessage += (char)buf[i]; // Konwersja do tekstu
    }
    Serial.println(receivedMessage);  // Wyświetlenie wiadomości

    if (receivedMessage == "Live bit") {
       lastLiveBitTime = millis();  // Aktualizacja czasu ostatniego Live
bit
    }
    delay(500);  // Krótkie opóźnienie dla czytelności sygnału
  }

  // Sprawdzanie utraty komunikacji
  if (millis() - lastLiveBitTime > timeout) {
    digitalWrite(LED_PIN, LOW);  // Wyłącz diodę LED
    Serial.println("Brak komunikacji!");
  }
}
```



Rysunek 8: Podgląd układu w trakcie pracy



Rysunek 9: Odpowiedzi z Serial Monitor

Podsumowanie:

Zadanie polegało na implementacji sygnału "Live bit", który okresowo potwierdza ciągłość komunikacji między nadajnikiem a odbiornikiem. Opracowano program, który wysyła sygnał co 1 sekundę, a odbiornik monitoruje jego odbiór, zapalając diodę w przypadku poprawnej transmisji i gasząc ją w przypadku braku sygnału. Zapoznano się z implementacją mechanizmów monitorowania komunikacji w transmisji radiowej.

3.4. Laboratorium 2 – Podsumowanie

Laboratorium umożliwiło zdobycie wiedzy na temat podstaw działania komunikacji radiowej w systemach mikroprocesorowych. Dzięki praktycznym zadaniom zrozumiałem, jak konfiguracja i obsługa modułów RF wpływa na niezawodność transmisji.

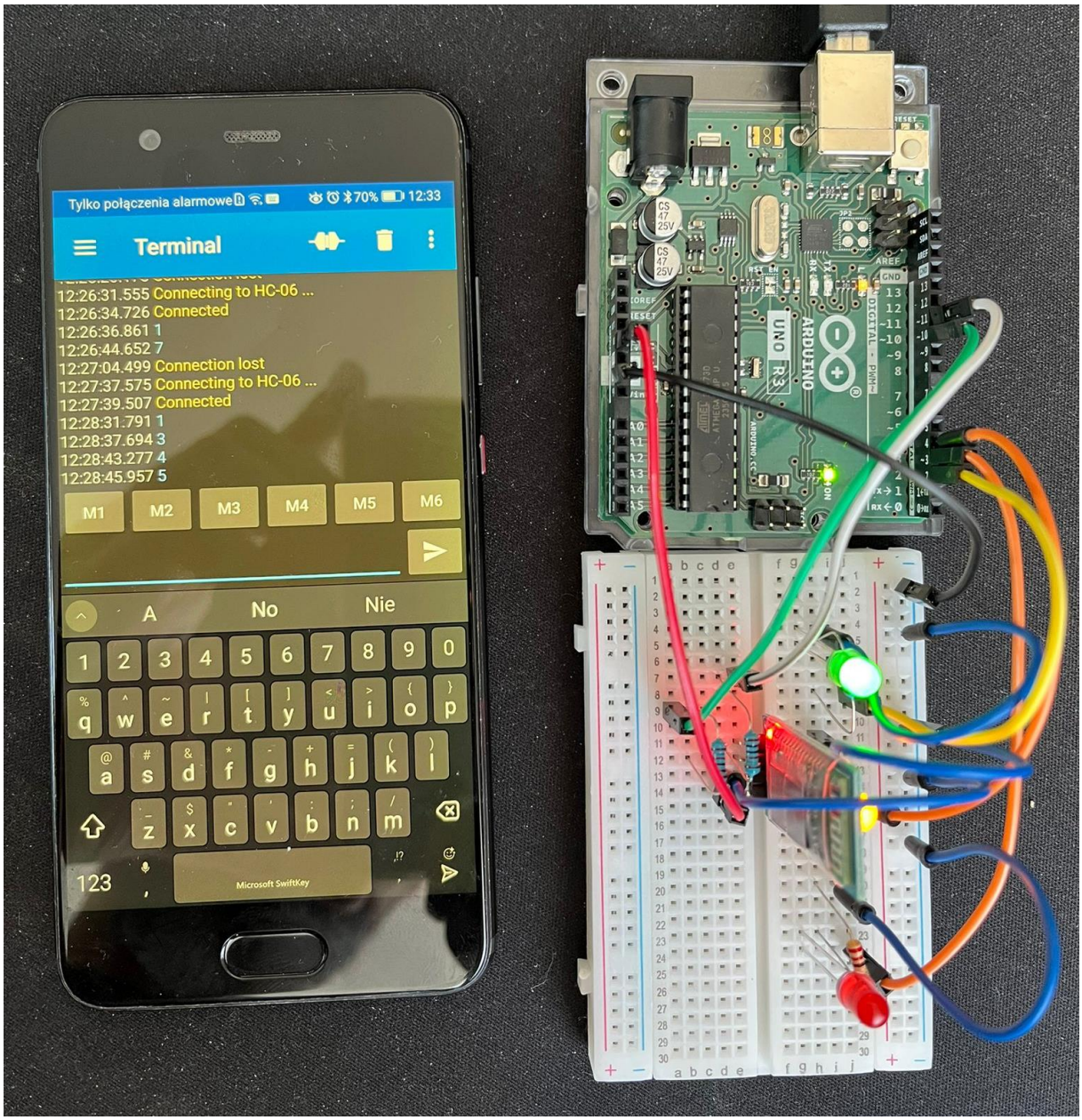
4. Laboratorium 3 – Moduł komunikacji Bluetooth

4.1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z zasadą działania, konfiguracją, podłączeniem oraz obsługą programową modułu do komunikacji Bluetooth HC-06 umożliwiającego bezprzewodową transmisje danych. Dalej zdobycie umiejętności podłączenia modułu z interfejsem Bluetooth do płytki Arduino oraz zaprojektowanie sterowanego bezprzewodowo układu mikroprocesorowego.

4.2. Zadanie 1 – Połączenie Arduino z Serial Bluetooth Terminal

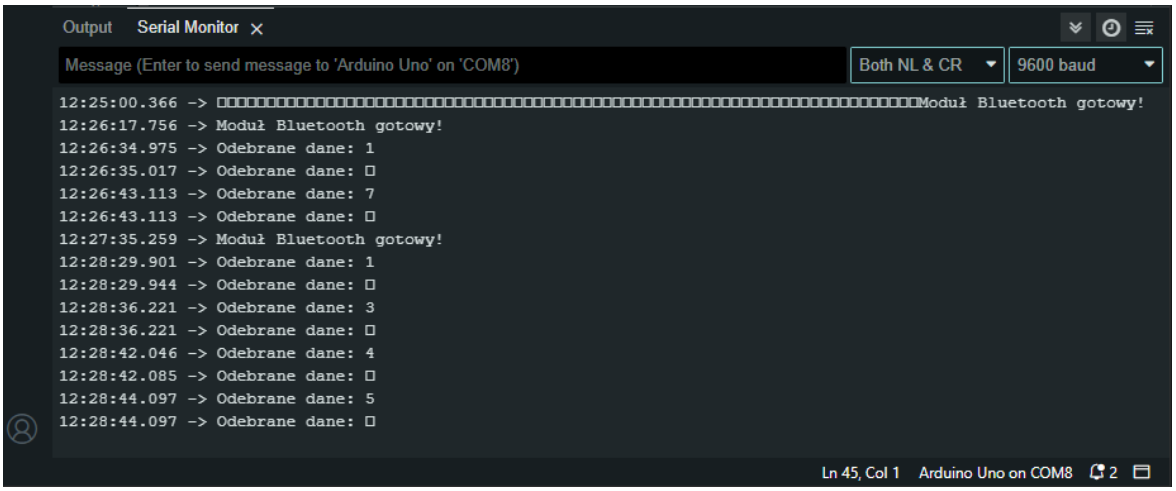
W trakcie testowania kodu podanego w instrukcji do laboratorium zauważono pewne problemy z konfliktem wejść pomiędzy TX i RX, a podłączeniem USB do komputera. By uniknąć konfliktu skorzystano z biblioteki SoftwareSerial i zmieniono docelowe porty z RX i TX na odpowiednio 10 i 11.



Rysunek 10: Podgląd układu wraz z aplikacją do testów

Program 6: Kod programu Zadania 1 Laboratorium 3

```
#include <SoftwareSerial.h>
SoftwareSerial bluetooth(10, 11); // RX, TX

#define LEDYellow 4
#define LEDRed 3
#define LEDBlue 2

char odebraneDane;

void setup() {
  Serial.begin(9600);          // Komunikacja z monitorem szeregowym
  bluetooth.begin(9600);       // Komunikacja z HC-06
  pinMode(LEDYellow, OUTPUT);
  pinMode(LEDRed, OUTPUT);
  pinMode(LEDBlue, OUTPUT);
  Serial.println("Moduł Bluetooth gotowy!");
}

void loop() {
   if (bluetooth.available() > 0) {  // Sprawdź, czy dostępne są dane z
Bluetooth
    odebraneDane = bluetooth.read(); // Odczytaj dane z HC-06
    Serial.print("Odebrane dane: ");
     Serial.println(odebraneDane); // Wyświetl odebrane dane w monitorze
szeregowym
  }

  // Sterowanie diodami
  if (odebraneDane == '1') {
    digitalWrite(LEDYellow, HIGH);
  } else if (odebraneDane == '2') {
    digitalWrite(LEDYellow, LOW);
  }

  if (odebraneDane == '3') {
    digitalWrite(LEDRed, HIGH);
  } else if (odebraneDane == '4') {
    digitalWrite(LEDRed, LOW);
  }

  if (odebraneDane == '5') {
    digitalWrite(LEDBlue, HIGH);
  } else if (odebraneDane == '6') {
    digitalWrite(LEDBlue, LOW);
  }
}
```



Rysunek 11: Odpowiedzi z Serial Monitor

Podsumowanie:

Zaprojektowano układ, który umożliwia sterowanie diodami LED za pomocą komend wysyłanych z aplikacji mobilnej (Serial Bluetooth Terminal). Kod obsługiwał włączanie i wyłączanie poszczególnych diod na podstawie odebranych danych. Zapoznano się z tym, jak nawiązać połączenie Bluetooth pomiędzy Arduino, a smartfonem oraz sterować urządzeniami za pomocą danych przesyłanych bezprzewodowo.

4.3. Zadanie 2 – Sekwencyjne włączanie diod

Opis działania:

- Komenda 1: Zapala zieloną diodę (wyłącza pozostałe).
- Komenda 2: Zapala czerwoną diodę, a po 3 sekundach także żółtą.
- Komenda 3: Uruchamia sekwencję jak na skrzyżowaniu: Zielone → Żółte → Czerwone.
- Komenda 4: Wyłącza wszystkie diody (zieloną, żółtą i czerwoną).

Program 7: Kod programu do Zadania 2 Laboratorium 3:

```
#include <SoftwareSerial.h>
SoftwareSerial bluetooth(10, 11); // RX, TX

#define LEDGreen 2  // Dioda zielona (Blue w oryginalnym kodzie)
#define LEDYellow 4 // Dioda żółta
#define LEDRed 3    // Dioda czerwona

char odebraneDane;

void setup() {
  Serial.begin(9600);          // Komunikacja z monitorem szeregowym
  bluetooth.begin(9600);       // Komunikacja z HC-06
  pinMode(LEDGreen, OUTPUT);
  pinMode(LEDYellow, OUTPUT);
  pinMode(LEDRed, OUTPUT);

  // Wyłącz wszystkie diody na start
```

```
  digitalWrite(LEDGreen, LOW);
  digitalWrite(LEDYellow, LOW);
  digitalWrite(LEDRed, LOW);

  Serial.println("Bluetooth gotowy!");
}

void loop() {
  if (bluetooth.available() > 0) {   // Sprawdź, czy dostępne są dane z
Bluetooth
    odebraneDane = bluetooth.read(); // Odczytaj dane z HC-06
    Serial.print("Odebrane dane: ");
    Serial.println(odebraneDane); // Wyświetl odebrane dane w monitorze
szeregowym
  }

  // Obsługa diod na podstawie odebranych danych
  if (odebraneDane == '1') {
    zapalZielona(); // Zapal zieloną diodę
  } else if (odebraneDane == '2') {
    zapalCzerwonaZoltyPo3Sek(); // Zapal czerwoną, a po 3 sek żółtą
  } else if (odebraneDane == '3') {
    swiatlaSkrzyzowanie(); // Sekwencja świateł jak na skrzyżowaniu
  } else if (odebraneDane == '4') {
    wylaczWszystkie(); // Wyłącz wszystkie diody
  }
}

// Funkcja zapalająca zieloną diodę
void zapalZielona() {
  digitalWrite(LEDGreen, HIGH);  // Zapal zieloną
  digitalWrite(LEDYellow, LOW); // Wyłącz pozostałe
  digitalWrite(LEDRed, LOW);
}

// Funkcja zapalająca czerwoną diodę, a po 3 sekundach żółtą
void zapalCzerwonaZoltyPo3Sek() {
  digitalWrite(LEDRed, HIGH);   // Zapal czerwoną diodę
  digitalWrite(LEDGreen, LOW); // Wyłącz pozostałe
  digitalWrite(LEDYellow, LOW);
  delay(3000);                  // Czekaj 3 sekundy
  digitalWrite(LEDYellow, HIGH); // Zapal żółtą
}

// Funkcja symulująca światła na skrzyżowaniu
void swiatlaSkrzyzowanie() {
  // Zielone światło
  digitalWrite(LEDGreen, HIGH);
  digitalWrite(LEDYellow, LOW);
  digitalWrite(LEDRed, LOW);
  delay(3000); // Świeci 3 sekundy
```

```
  // Żółte światło
  digitalWrite(LEDGreen, LOW);
  digitalWrite(LEDYellow, HIGH);
  digitalWrite(LEDRed, LOW);
  delay(1000); // Świeci 1 sekundę

  // Czerwone światło
  digitalWrite(LEDGreen, LOW);
  digitalWrite(LEDYellow, LOW);
  digitalWrite(LEDRed, HIGH);
  delay(3000); // Świeci 3 sekundy
}

// Funkcja wyłączająca wszystkie diody
void wylaczWszystkie() {
  digitalWrite(LEDGreen, LOW);  // Wyłącz zieloną
  digitalWrite(LEDYellow, LOW); // Wyłącz żółtą
  digitalWrite(LEDRed, LOW);    // Wyłącz czerwoną
}
```

Podsumowanie:

Zaprogramowano sekwencję włączania diod LED przypominającą działanie sygnalizacji świetlnej. Diodami sterowano na podstawie komend przesyłanych przez Bluetooth. Wykorzystano opóźnienia do odzwierciedlenia kolejnych stanów świateł (zielone → żółte → czerwone). Zapoznano się, jak implementować złożone sekwencje zdarzeń w systemie mikroprocesorowym i sterować nimi bezprzewodowo.

4.4. Zadanie 3 – Regulacja jasności diody poprzez smartfon.

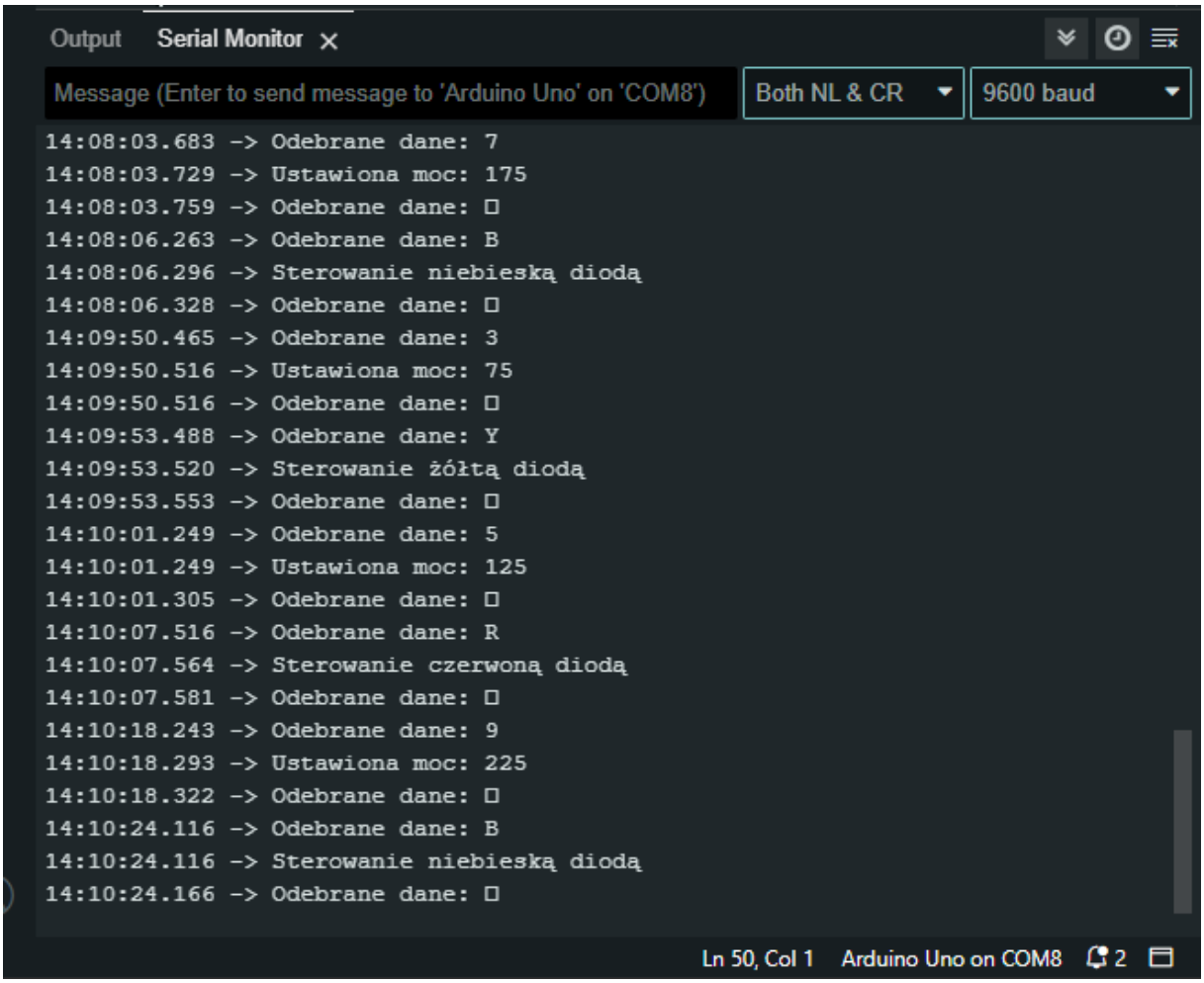
Opis działania:

Jasność jest ustawiana poprzez wpisanie cyfry od 1-9, gdzie 1 to wyłączenie diody, a 9 to ustawienie pełnej mocy. Następnie należy podać, które diody mają się świecić z zadaną mocą poprzez wpisanie: R – czerwona, B – niebieska lub Y- żółta. By zmienić ponownie jasność należy wpisać odpowiednio cyfrę oraz literę odpowiadającą danej diodzie. jasność.

Program 8: Kod programu Zadanie 3 Laboratorium 3

```
#include <SoftwareSerial.h>
SoftwareSerial bluetooth(10, 11); // RX, TX

#define LEDYellow 3
#define LEDRed 5
#define LEDBlue 6

char odebraneDane;
int moc = 0;  // Jasność (domyślnie 0)

void setup() {
```

```
  Serial.begin(9600);          // Komunikacja z monitorem szeregowym
  bluetooth.begin(9600);       // Komunikacja z HC-06
  pinMode(LEDYellow, OUTPUT);
  pinMode(LEDRed, OUTPUT);
  pinMode(LEDBlue, OUTPUT);
  Serial.println("Moduł Bluetooth gotowy!");
}

void loop() {
  if (bluetooth.available() > 0) {  // Sprawdź, czy dostępne są dane z
Bluetooth
    odebraneDane = bluetooth.read(); // Odczytaj dane z HC-06
    Serial.print("Odebrane dane: ");
     Serial.println(odebraneDane); // Wyświetl odebrane dane w monitorze
szeregowym

    // Obsługa sterowania jasnością
    if (odebraneDane >= '0' && odebraneDane <= '9') {
      moc = (odebraneDane - '0') * 25;  // Konwersja '0'-'9' na jasność (0,
25, ..., 255)
      Serial.print("Ustawiona moc: ");
      Serial.println(moc);
    }
  }

  // Sterowanie diodami
  if (odebraneDane == 'Y') {  // Sterowanie żółtą diodą
    analogWrite(LEDYellow, moc);
    Serial.println("Sterowanie żółtą diodą");
  }

  if (odebraneDane == 'R') {  // Sterowanie czerwoną diodą
    analogWrite(LEDRed, moc);
    Serial.println("Sterowanie czerwoną diodą");
  }

  if (odebraneDane == 'B') {  // Sterowanie niebieską diodą
    analogWrite(LEDBlue, moc);
    Serial.println("Sterowanie niebieską diodą");
  }
}
```



Rysunek 12: Odpowiedzi z Serial Monitor

Podsumowanie:

Wprowadzono możliwość sterowania jasnością diod LED poprzez Bluetooth za pomocą sygnałów PWM. Przesyłane komendy pozwalały regulować jasność każdej z diod (żółtej, czerwonej i niebieskiej). Nauczono się jak używać sygnałów PWM do regulacji jasności diod oraz jak implementować bardziej zaawansowane sterowanie w połączeniu z komunikacją Bluetooth.

4.5. Laboratorium 3 – Podsumowanie

Laboratorium pozwoliło zapoznać się z zasadami komunikacji Bluetooth oraz jej zastosowaniem w sterowaniu urządzeniami. Zrozumiałem, jak implementować różne schematy sterowania oraz jak wykorzystać sygnały PWM do regulacji jasności diod.

5. Wnioski

Dzięki wykonanym ćwiczeniom zdobyto szeroką wiedzę na temat obsługi mikrokontrolerów Arduino, komunikacji bezprzewodowej oraz sterowania urządzeniami zewnętrznymi. Każde laboratorium pozwoliło na rozwinięcie umiejętności programistycznych i praktycznych związanych z projektowaniem układów mikroprocesorowych. Zdobyta wiedza jest solidną podstawą do realizacji bardziej zaawansowanych projektów w przyszłości.