

# SPRAWOZDANIE Z ZAJĘĆ PBL1

*Komunikacja przewodowa*

*Kinga Konieczna*

*Jan Czechowski*

*16.01.2025*

# 1. Zadanie 1

## 1.1. Cel

Zwizualizować i odkodować na oscyloskopie przykładowe dane przesyłane przez protokoły UART, I2C i SPI. Aby zdekodować dane z I2C i SPI należy zastanowić się, w jaki sposób odbywa się komunikacja pomiędzy dwoma urządzeniami wykorzystującymi wspomniane protokoły.

## 1.2. Wykorzystane narzędzia i komponenty

- Mikrokontroler Arduino UNO
- Oprogramowanie: Arduino IDE
- Oscyloskop cyfrowy Keysight DSOX1204G
- Wyświetlacz LCD 2x16 oraz konwerter I2C LCM1602

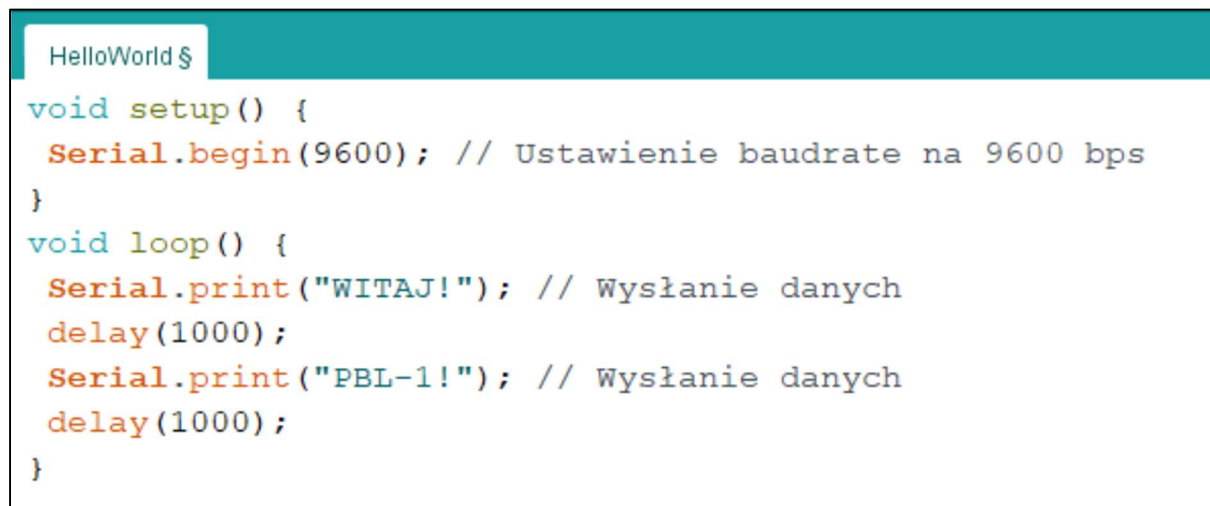
## 1.3. Dekodowanie danych przesyłanych przez protokół UART

### 1.3.1. Podłączenie sprzętu

Sonda 1 – pin Tx (D1) oraz do GND na Arduino.

### 1.3.2. Kod w Arduino IDE

W programie Arduino IDE zaprogramowaliśmy płytkę zgodnie z kodem na **Rys. 1.1**:

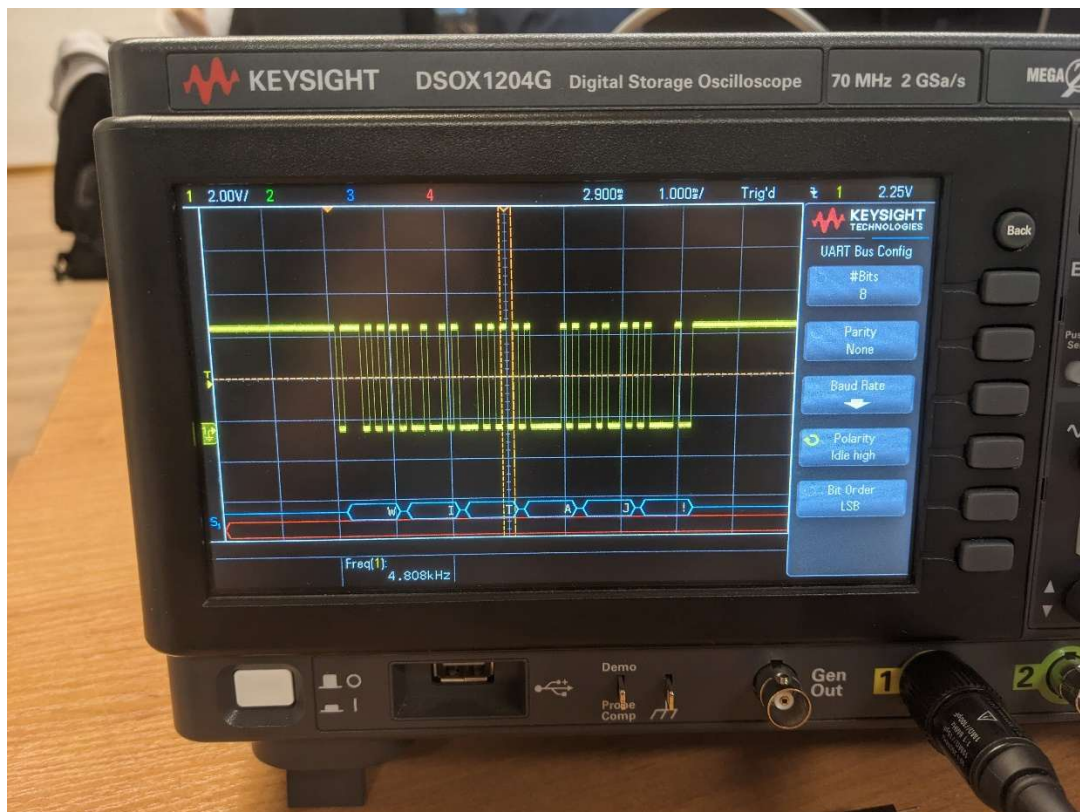


```
HelloWorld$  
void setup() {  
  Serial.begin(9600); // Ustawienie baudrate na 9600 bps  
}  
void loop() {  
  Serial.print("WITAJ!"); // Wysłanie danych  
  delay(1000);  
  Serial.print("PBL-1!"); // Wysłanie danych  
  delay(1000);  
}
```

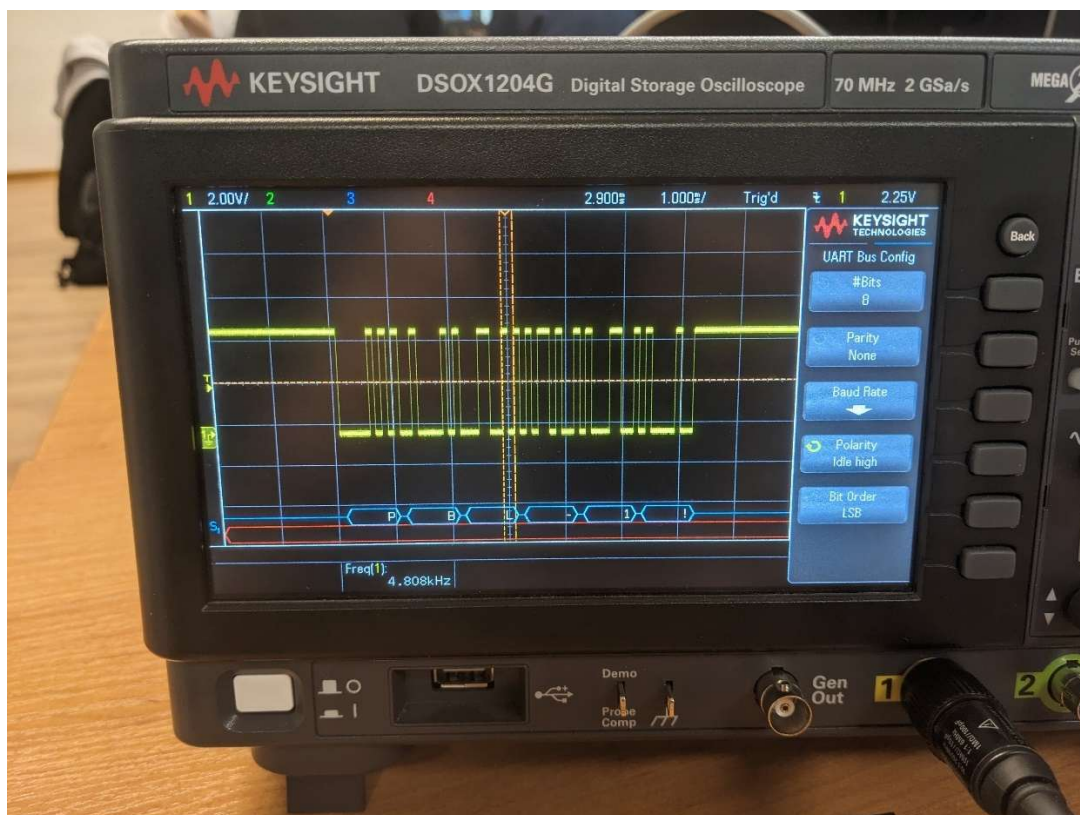
*Rys. 1.1. Kod do zadania 1. – UART.*

### 1.3.3. Sygnał na oscyloskopie

Z oscyloskopu możemy odczytać dane przesyłane w czasie rzeczywistym z Arduino w czytelny sposób w formacie ASCII, co widać na **Rys. 1.2** oraz **Rys. 1.3**.



Rys. 1.2. Zdekodowany napis „WITAJ!”. [opracowanie własne]



Rys. 1.3. Zdekodowany napis „PBL-1!”. [opracowanie własne]

## 1.4. Dekodowanie danych przesyłanych przez protokół I2C

### 1.4.1. Podłączenie sprzętu

Podłączyliśmy wyświetlacz LCD do modułu Arduino, dzięki czemu dekodowanie danych jest możliwe poprzez podsłuchiwanie pinów z LCD.

- VCC (LCD) → 5V (Arduino)
- GND (LCD) → GND (Arduino)
- SDA (LCD) → A4 (Arduino Uno)
- SCL (LCD) → A5 (Arduino Uno)
- Sonda kanału 1 do linii SDA (linia danych)
- Sonda kanału 2 do linii SCL (linia zegara)
- GND sond oscyloskopu do GND układu

### 1.4.2. Kod w Arduino IDE

Zaprogramowaliśmy płytkę zgodnie z kodem na **Rys. 1.4**:

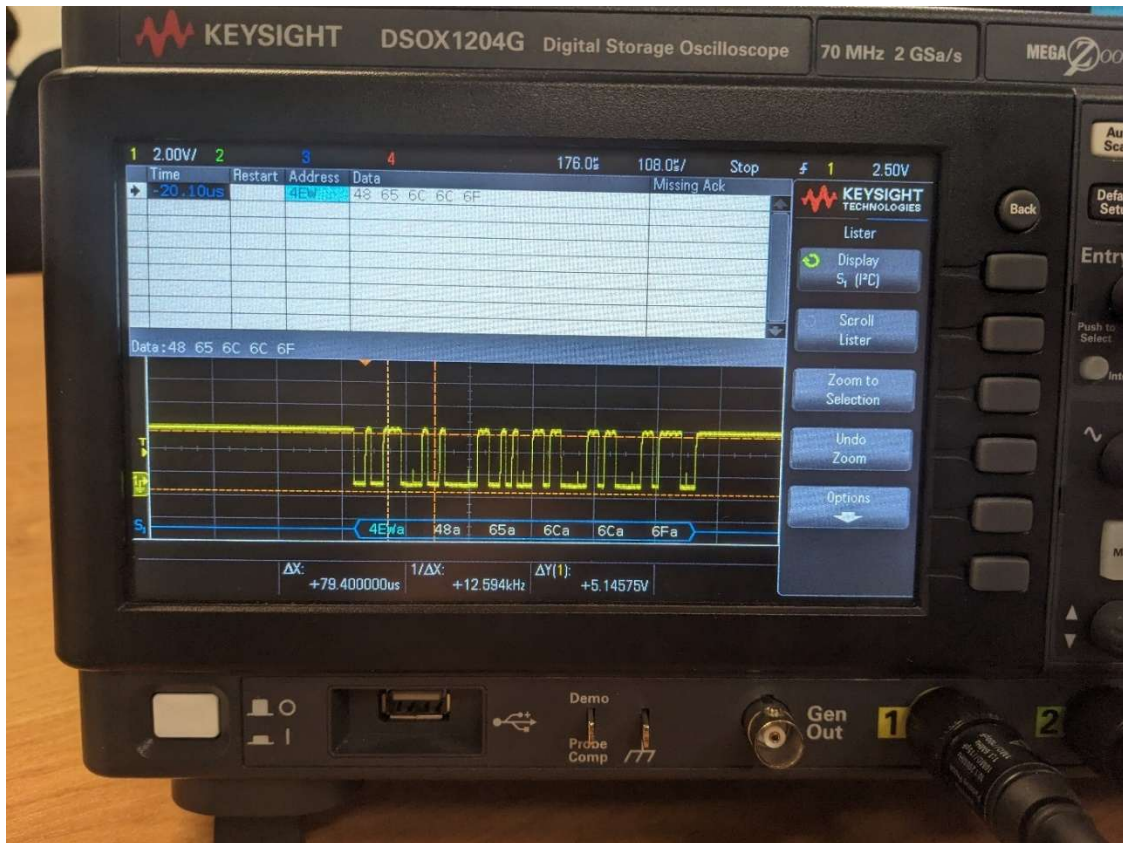
```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup() {
  Wire.begin();
  lcd.begin(16, 2);
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Hello, World!");
}
void loop() {
  Wire.beginTransmission(0x27);
  Wire.write("Hello");
  Wire.endTransmission();
  delay(1000);
}
```

*Rys. 1.4. Kod do zadania 1. – I2C. [opracowanie własne]*

### 1.4.3. Odczyt z oscyloskopu

Z oscyloskopu [**Rys. 1.5**] możemy odczytać dane wysłane przez protokół I2C. Napis „Hello” jest przedstawiony w systemie szesnastkowym.





Rys. 1.5. Odczyt sygnału i napisu na oscyloskopie przez protokół I2C. [opracowanie własne]

#### 1.4.4. Przebieg komunikacji w I2C

1. **Start:** Master generuje warunek startu, polegający na zmianie stanu linii SDA z wysokiego na niski, gdy linia SCL jest wysoka.
2. **Adresowanie:** Master wysyła adres slave'a (7-bitowy lub 10-bitowy) przez linię SDA, zsynchronizowany z sygnałem zegara na linii SCL.
3. **Odczyt/Zapis:** Po adresie następuje przesyłanie danych (bajtów), zsynchronizowane przez sygnał zegarowy na linii SCL.
4. **Stop:** Master generuje warunek stopu, zmieniając stan linii SDA z niskiego na wysoki, gdy SCL jest wysoka.

### 1.5. Dekodowanie danych przesyłanych przez protokół SPI

#### 1.5.1. Podłączenie sprzętu

- VCC (LCD) do 5V
- GND (LCD) do GND
- MOSI (LCD) do 11 pinu
- SCK (LCD) do 13 pinu
- Sonda kanału 1 do MOSI oraz do GND
- Sonda kanału 2 do linii SCK oraz do GND

#### 1.5.2. Kod w Arduino IDE

Zaprogramowaliśmy płytkę Arduino Uno zgodnie z kodem na **Rys. 1.6:**

```

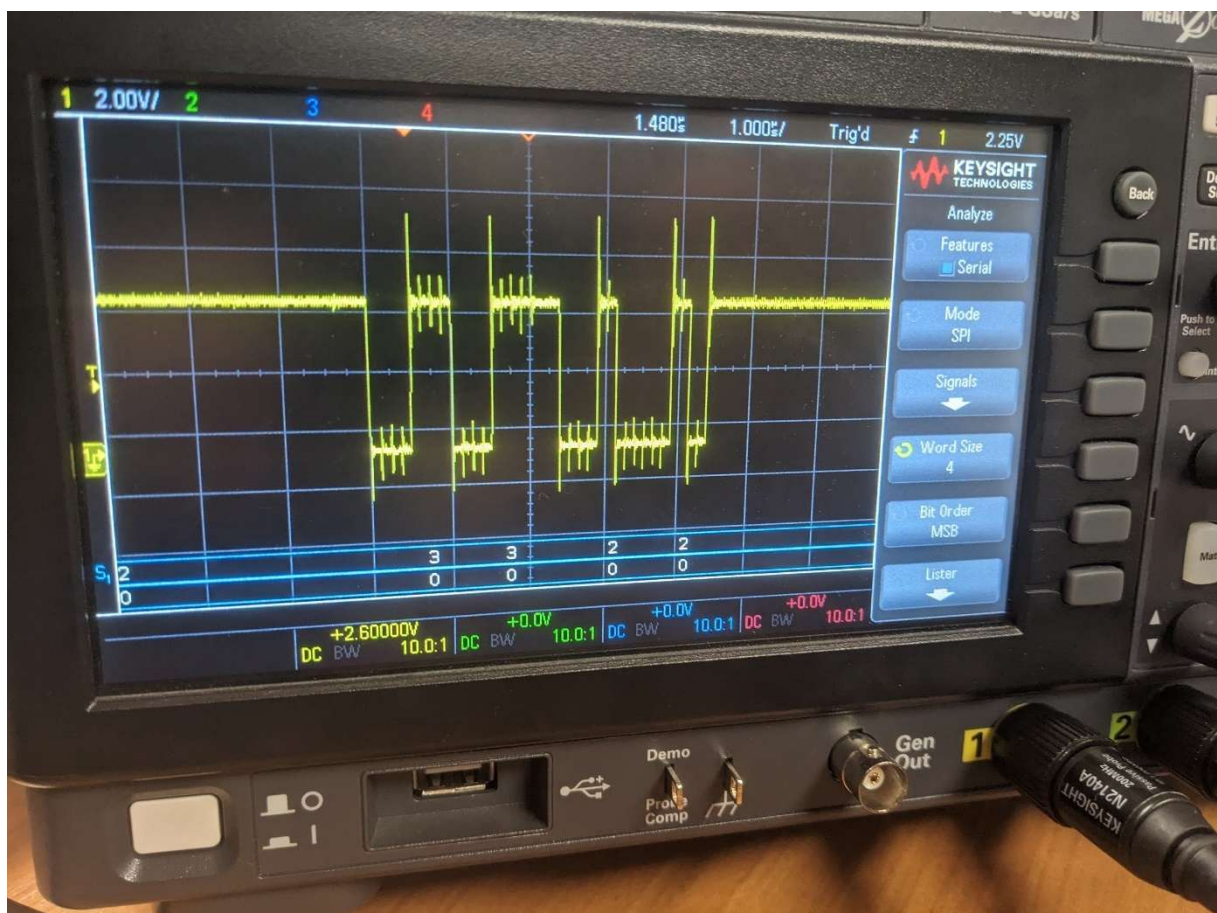
#include <SPI.h>
void setup() {
  SPI.begin(); // Inicjalizacja SPI
}
void loop() {
  digitalWrite(SS, LOW); // Aktywacja linii Slave Select
  SPI.transfer(0x33); // Przesłanie danych
  SPI.transfer(0x22); // Przesłanie danych
  digitalWrite(SS, HIGH); // Dezaktywacja linii Slave Select
  delay(1000);
}

```

Rys. 1.6. Kod do zadania 1. – SPI

### 1.5.3. Odczyt na oscyloskopie

Na Rys 1.7 widzimy zdekodowane dane z SPI, czyli bajt 0x33 oraz bajt 0x22:



Rys. 1.7. Odczyt bajtów na oscyloskopie przesłanych przez SPI. [opracowanie własne]

## 2. Zadanie 2

### 2.1. Cel

Wykonać komunikację pomiędzy dwoma modułami Arduino z wykorzystaniem protokołów UART i I2C.

### 2.2. Wykorzystane narzędzia i komponenty

- Dwa mikrokontrolery Arduino UNO
- Oprogramowanie: Arduino IDE

### 2.3. Komunikacja pomiędzy dwoma modułami Arduino z wykorzystaniem protokołu UART

W komunikacji UART jedno Arduino działa jako nadawca, a drugie jako odbiorca. Nadawca przesyła dane, a odbiorca odczytuje dane i wyświetla je na monitorze szeregowym. Komunikacja odbywa się przez linie TX (transmisja) i RX (odbior).

#### 2.3.1. Podłączenie sprzętu

- **Tx** Arduino 1 – podłączony do **Rx** Arduino 2
- **Rx** Arduino 1 – podłączony do **Tx** Arduino 2
- **GND** Arduino 1 – podłączony do **GND** Arduino 2

#### 2.3.2. Kod w Arduino IDE

W programie Arduino IDE zaprogramowaliśmy płytki zgodnie z poniższym kodem [Rys. 2.1] – Arduino 1 i [Rys. 2.2] – Arduino 2:

```
void setup() {  
  
    Serial.begin(9600);  
  
}  
  
void loop() {  
  
    Serial.write('H');  
    delay(1000);  
  
}
```

Rys. 2.1. Kod do Arduino 1 – wysłanie wiadomości poprzez UART. [opracowanie własne]

```

void setup() {
  // Rozpocznij komunikację UART na prędkości 9600 bps
  Serial.begin(9600);
}

void loop() {
  // Sprawdź, czy dostępne są dane do odczytu
  if (Serial.available() > 0) {
    // Odczytaj dane z UART
    char receivedChar = Serial.read();

    // Wyświetl odebrany znak na monitorze szeregowym
    Serial.print("Odebrano: ");
    Serial.println(receivedChar);
  }
}

```

Rys. 2.2. Kod do Arduino 2 – odbiór wiadomości poprzez UART. [opracowanie własne]

### 2.3.3. Odczyt z Serial Monitora – Arduino 2

Z Serial Monitora na Rys. 2.3 w Arduino 2 możemy odczytać wiadomość przesłaną w czasie rzeczywistym z Arduino 1 poprzez protokół UART.



Rys. 2.3. Serial Monitor Arduino 2 – odbiór wiadomości poprzez UART z Arduino 1. [opracowanie własne]

## 2.4. Komunikacja pomiędzy dwoma modułami Arduino z wykorzystaniem protokołu I2C

W komunikacji I2C jedno Arduino działa jako master, a drugie jako slave. Master inicjuje transmisję, wysyła adres slave'a i przesyła dane. Slave odbiera te dane i odczytuje je a następnie wyświetla w Serial Monitorze. Komunikacja odbywa się przez linie SDA (dane) i SCL (zegara).

### 2.4.1. Podłączenie sprzętu

- **Pin A4** Arduino nr 1 – podłączony do **Pinu A4** Arduino nr 2
- **Pin A5** Arduino nr 1 – podłączony do **Pinu A5** Arduino nr 2
- **GND** Arduino nr 1 – podłączony do **GND** Arduino nr 2



### 2.4.2. Kod w Arduino IDE

W programie Arduino IDE zaprogramowaliśmy płytki zgodnie z poniższym kodem [Rys. 2.4]

– Arduino nr 1 i [Rys. 2.5] – Arduino nr 2:

```
#include <Wire.h>

void setup() {
  Wire.begin();
  Serial.begin(9600);
}

void loop() {
  Wire.beginTransmission(8);

  Wire.write(420);

  Wire.endTransmission();

  Serial.println("Dane wysłane do slave: 420");
  delay(1000);
}
```

Rys. 2.4. Kod do Arduino nr 1 – wysłanie wiadomości poprzez I2C. [opracowanie własne]

```
#include <Wire.h>

void setup() {
  Wire.begin(8);
  Wire.onReceive(receiveData);
  Serial.begin(9600);
}

void loop() {
}

// Funkcja odbierania danych
void receiveData(int byteCount) {
  while (Wire.available()) {
    int receivedByte = Wire.read();
    Serial.print("Odebrano: ");
    Serial.println(receivedByte);
  }
}
```

Rys. 2.5. Kod do Arduino nr 2 – odbiór wiadomości poprzez I2C. [opracowanie własne]

### 2.4.3. Odczyt z Serial Monitora – Arduino nr 2

Z Serial Monitora na **Rys. 2.6** w Arduino nr 2 możemy odczytać wiadomość przesłaną w czasie rzeczywistym z Arduino nr 1 poprzez protokół I2C.



*Rys. 2.6. Serial Monitor Arduina nr 2 – odbiór wiadomości poprzez I2C z Arduino nr 1. [opracowanie własne]*

## 3. Zadanie 3

### 3.1. Cel

Wykonać sterowanie silnikiem krokowym poprzez PWM z Arduino. Dane sygnału PWM odczytać na oscyloskopie.

### 3.2. Wykorzystane narzędzia i komponenty

- Dwa mikrokontrolery Arduino UNO
- Oprogramowanie: Arduino IDE
- Serwo FS90
- Oscyloskop cyfrowy Keysight DSOX1204G

### 3.3. Podłączenie sprzętu

- **Czerwony** przewód serwa do **pinu 5V** Arduino (zasilanie).
- **Brazowy** przewód serwa do **GND** Arduino (masa).
- **Pomarańczowy** przewód serwa (sygnałowy) do **pinu 9** Arduino.
- **Sonda** do **pinu 9** Arduino (sygnał PWM).
- **Masa** sondy do **GND** Arduino.

### 3.4. Kod w Arduino IDE

W programie Arduino IDE zaprogramowaliśmy płytkę zgodnie z kodem na **Rys. 3.1**.

```
#include <Servo.h>

Servo myservo;

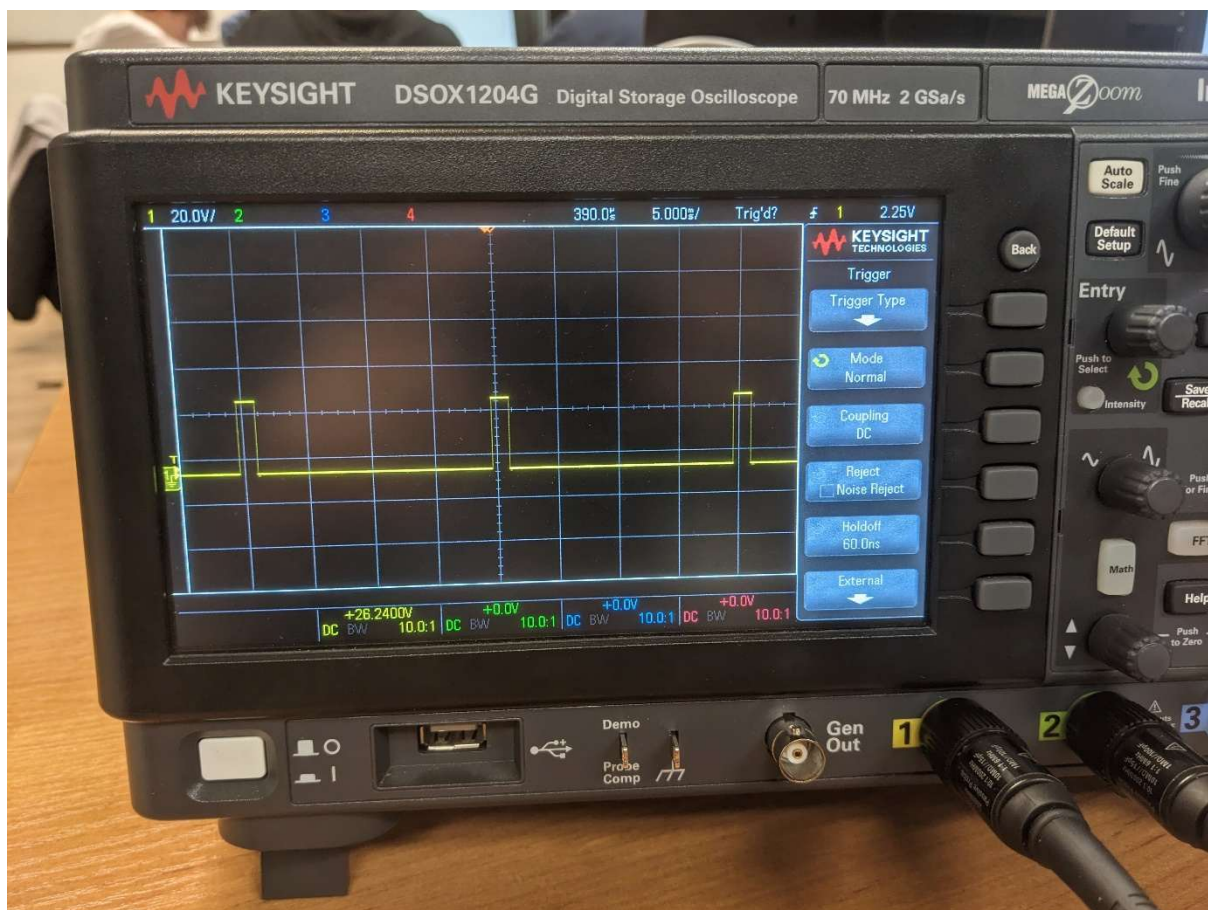
void setup() {
  myservo.attach(9);
}

void loop() {
  myservo.write(90);
}
```

*Rys. 3.1. Kod do Arduino – ustawienie serwomechanizmu w pozycji 90 stopni. [opracowanie własne]*

### 3.5. Odczyt na oscyloskopie

Na **Rys. 3.2** widzimy wykres przedstawiający przebieg sygnału PWM odpowiadający pozycji serwomechanizmu ustawionej na 90 stopni.



Rys. 3.2. Odczyt sygnału PWM na oscyloskopie. [opracowanie własne]



## Spis ilustracji

Rys. 1.1. Kod do zadania 1. – UART. ....	2
Rys. 1.2. Zdekodowany napis „WITAJ!”.....	3
Rys. 1.3. Zdekodowany napis „PBL-1!”.....	3
Rys. 1.4. Kod do zadania 1. – I2C.....	4
Rys. 1.5. Odczyt sygnału i napisu na oscyloskopie przez protokół I2C. ....	5
Rys. 1.6. Kod do zadania 1. – SPI.....	6
Rys. 1.7. Odczyt bajtów na oscyloskopie przesłanych przez SPI.....	6
Rys. 2.1. Kod do Arduino 1 – wysłanie wiadomości poprzez UART. [opracowanie własne].....	7
Rys. 2.2. Kod do Arduino 2 – odbiór wiadomości poprzez UART. [opracowanie własne].....	8
Rys. 2.3. Serial Monitor Arduina 2 – odbiór wiadomości poprzez UART z Arduino 1. [opracowanie własne].....	8
Rys. 2.4. Kod do Arduino nr 1 – wysłanie wiadomości poprzez I2C. [opracowanie własne].....	9
Rys. 2.5. Kod do Arduino nr 2 – odbiór wiadomości poprzez I2C. [opracowanie własne].....	9
Rys. 2.6. Serial Monitor Arduina nr 2 – odbiór wiadomości poprzez I2C z Arduino nr 1. [opracowanie własne].....	10
Rys. 3.1. Kod do Arduino – ustawienie serwomechanizmu w pozycji 90 stopni. [opracowanie własne].....	11
Rys. 3.2. Odczyt sygnału PWM na oscyloskopie. [opracowanie własne].....	12