Urządzenia peryferyjne

Silnik krokowy

Autor : Jakub Smolarczyk 272924

Gr. czw. np. 13:15 – 16:15

1. Zadania do wykonania

Podstawowym zadaniem do wykonania było zapoznanie się z właściwościami systemowymi urządzenia USB używanego do sterowania silnikiem, uruchomienie programu ft_prog w celu sprawdzenia kluczowych parametrów silnika oraz uruchomienie programu usbstep w celu sprawdzenia sposobu działania silnika krokowego. W kolejnym kroku należało napisać aplikację do sterowania silnikiem krokowym.

2. Wstęp teoretyczny

Silnik krokowy to przykład maszyny, która przekształca energię elektryczną w energię mechaniczną, powodując ruch obrotowy. Odbywa się to poprzez impulsowe zasilanie prądem elektrycznym, które wymusza na magnetycznym lub elektromagnetycznym wirniku silnika ruch o określony kąt, taki aby po wygenerowaniu na odpowiednich uzwojeniach pola magnetycznego, opór strumienia magnetycznego w układzie był jak najmniejszy.

Silniki krokowe można podzielić na dwa rodzaje: bipolarne oraz unipolarne. W silnikach bipolarnych każde z dwóch fazowych rdzeni posiada tylko jedno uzwojenie, co zwiększa moment obrotowy wykonywanych kroków, lecz komplikuje sterowanie. W takim silniku do zmiany biegunowości pola magnetycznego potrzebna jest zmiana kierunku przepływu prądu w uzwojeniach. W silnikach unipolarnych ten sam efekt może być osiągnięty bez zmiany kierunku przepływu prądu, gdyż każdy z dwóch fazowych rdzeni posiada dwa uzwojenia, które mogą być używane na przemian. Upraszcza to proces sterowania silnikiem kosztem mniejszego momentu obrotowego z powodu mniejszego użycia danego rdzenia.

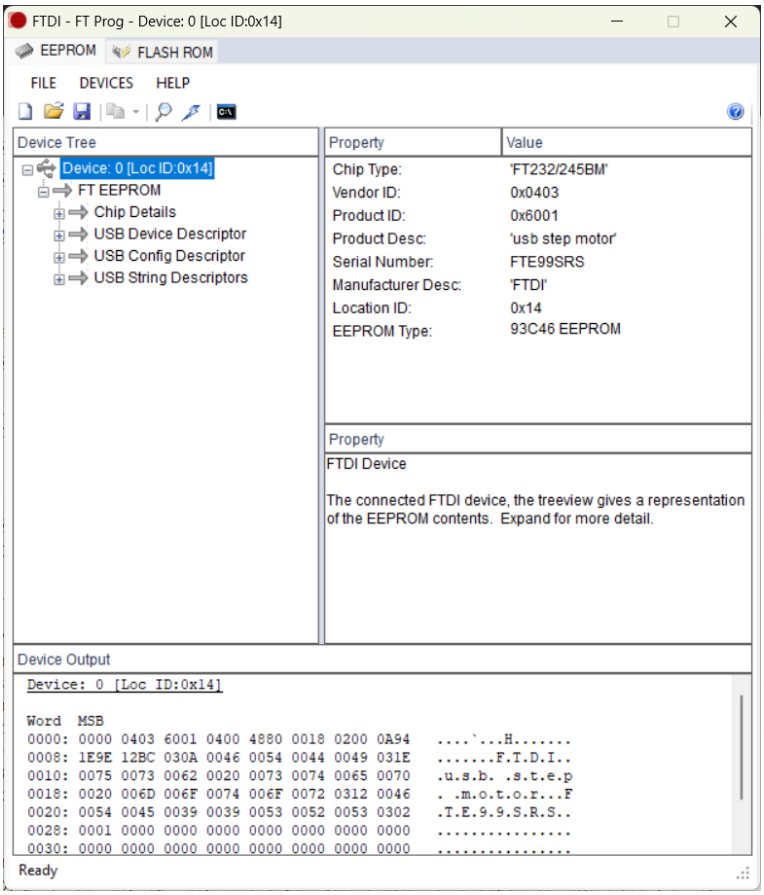
Istnieje kilka metod sterowania silnikiem, takie jak sterowanie falowe, pełnokrokowe oraz półkrokowe. W każdym z nich kontrolowany jest przepływ prądu przez uzwojenia w określonej sekwencji, która będzie wyglądać inaczej dla poszczególnych sterowań. Najprostsze jest sterowanie falowe, dla którego w danej chwili wykorzystywane jest tylko jedno uzwojenie, przez co sterowanie to charakteryzuje się minimalnym zużyciem energii. W sterowaniu pełnokrokowym aktywne są jednocześnie dwa uzwojenia po jednym na każdy rdzeń. Kroki w tym sterowaniu są tej samej długości co w sterowaniu falowym, jednak sterowanie to charakteryzuje stabilniejsza praca kosztem większego zużycia energii. Sterowanie półkrokowe łączy cechy sterowania falowego oraz pełnokrokowego. Na przemian wykorzystuje jedno lub dwa uzwojenia. Dzięki temu kroki wykonywane w tym sterowaniu są o połowę mniejsze niż w przypadku obu pozostałych sterowań, co zwiększa precyzję ruchu wirnika.

3. Realizacja ćwiczenia

Realizacja ćwiczenia została rozpoczęta od uruchomienia programu ft_prog w celu sprawdzenia poprawności danych parametrów silnika krokowego:

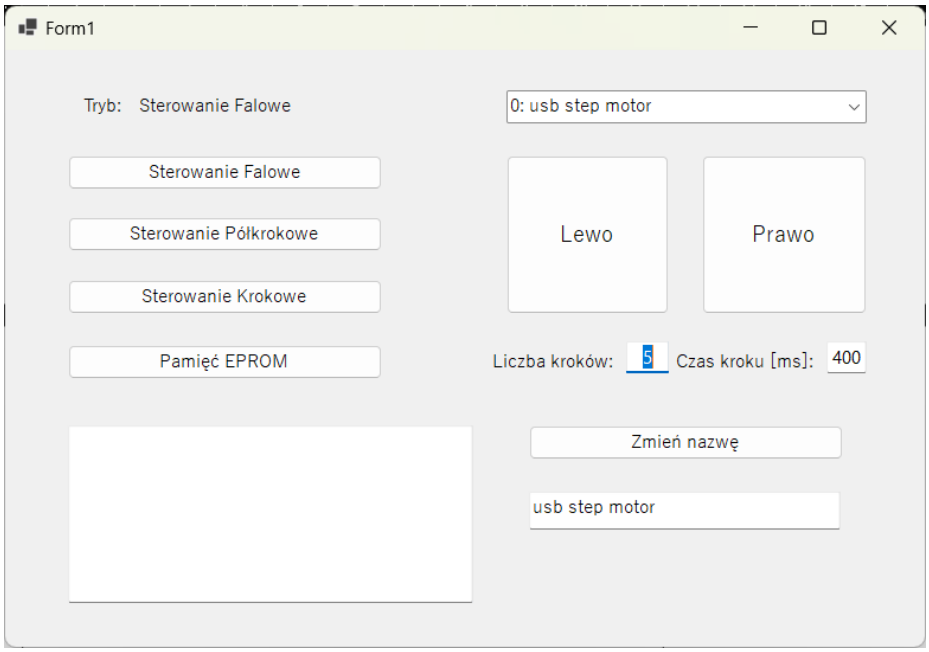
- Vendor ID = 0403 (VID)
- Product ID = 6001 (PID)
- Device Description ="usb step motor"

Na rysunku 1 przedstawiony został wynik programu, potwierdzający poprawność sprawdzanych parametrów. W kolejnym kroku uruchomiony został program usbstep.exe do przetestowania działania silnika krokowego, poprzez wykonanie różnych dostępnych z poziomu GUI operacji.



Rys. 1 Wynik działania programu ft_prog

Podczas testowania silnika krokowego nie wystąpiły żadne problemy w jego działaniu. Przystąpiliśmy więc do tworzenia aplikacji, której GUI z rysunku 2 zostało zaprojektowane tak aby oferowało użytkownikowi wszystkie funkcjonalności wymienione w instrukcji.



Rys. 2 GUI tworzonej aplikacji

- **Wybór i otwieranie urządzenia**

```
public Form1()
{
    InitializeComponent();
    ftdiDevice = new FTDI();
    PrepareDevices();
}

private void PrepareDevices()
{
    uint deviceCount = 0;

    // pobranie liczby urządzeń
    FTDI.FT_STATUS status = ftdiDevice.GetNumberOfDevices(ref deviceCount);
    if (status != FTDI.FT_STATUS.FT_OK || deviceCount == 0)
    {
        MessageBox.Show("Nie znaleziono urządzeń!", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }

    // pobranie listy urządzeń
    deviceList = new FTDI.FT_DEVICE_INFO_NODE[deviceCount];
    status = ftdiDevice.GetDeviceList(deviceList);
    if (status != FTDI.FT_STATUS.FT_OK)
    {
        MessageBox.Show("Nie udało się pobrać listy urządzeń!", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }

    // Wypełnienie ComboBox informacjami o urządzeniach
    comboBoxDevices.Items.Clear();
    for (uint i = 0; i < deviceCount; i++)
    {
        comboBoxDevices.Items.Add($"{i}: {deviceList[i].Description}");
        Console.WriteLine("Element: " + deviceList[i].ToString());
    }
    comboBoxDevices.SelectedIndex = 0; // domyślnie wybrane pierwsze urządzenie

}
```

Na początku wykonywania programu wywoływana jest metoda *PrepareDevices()*, która przygotowuje wewnątrz elementu ComboBox listę wszystkich podłączonych urządzeń. Pobierana jest najpierw liczba urządzeń poprzez *GetNumberOfDevices()*, której wartość jest wykorzystana w kolejnym kroku przy tworzeniu listy (new FTDI.FT_DEVICE_INDO_NODE[deviceCount]), po czym ta lista jest wypełniana i jej zawartość jest dodawana do ComboBox wewnątrz pętli for.

```
private void comboBoxDevices_SelectedIndexChanged(object sender, EventArgs e)
{
    if (comboBoxDevices.SelectedIndex < 0)
        return;

    // zakonczenie istniejacego połączenia
    if (ftdiDevice.IsOpen)
    {
        ftdiDevice.Close();
    }

    // otwarcie wybranego urządzenia
    int selectedIndex = (int)comboBoxDevices.SelectedIndex;
    FTDI.FT_STATUS status =
ftdiDevice.OpenBySerialNumber(deviceList[selectedIndex].SerialNumber);
    if (status != FTDI.FT_STATUS.FT_OK)
    {
        MessageBox.Show("Nie udało się otworzyć wybranego urządzenia!", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    else
    {
        ftdiDevice.SetBitMode(0xFF, 1);
        MessageBox.Show($"Urządzenie \"{comboBoxDevices.SelectedItem}\" otwarte
pomyślnie!", "Sukces", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

Po wypełnieniu ComboBox można wybrać potrzebne urządzenie. Jego otwieranie odbywa się wewnątrz metody *comboBoxDevices_SelectedIndexChanged()*, reagującej na zmianę wybranej pozycji w ComboBox. Najpierw zamykane jest poprzednie urządzenie, po czym otwierane jest nowe (o wybranym indeksie z ComboBox) po jego atrybucie SerialNumber. Dla otwartego urządzenia ustawiany jest odpowiedni tryb poprzez metodę *SetBitMode()* (asynchroniczny tryb big-bang).

- **Zamykanie urządzenia**

```
protected override void OnFormClosing(FormClosingEventArgs e)
{
    if (ftdiDevice != null)
    {
        ftdiDevice.Close();
    }
    base.OnFormClosing(e);
}
```

W reakcji na zamykanie aplikacji wywołana jest automatycznie metoda *OnFormClosing()*, wewnątrz której na otwartym urządzeniu jest wywoływana metoda *Close()*.

- **Ustawianie sterowania**

```
byte[] motorSequence = { 0x08, 0x02, 0x04, 0x01 };
byte[] waveSequence = { 0x08, 0x02, 0x04, 0x01 };           // sterowanie falowe
byte[] fullstepSequence = { 0x06, 0x0A, 0x09, 0x05 };       // sterowanie krokowe
byte[] halfstepSequence = { 0x08, 0x0A, 0x02, 0x06, 0x04, 0x05, 0x01, 0x09 };
// sterowanie półkrokowe

private void driveWave_Click(object sender, EventArgs e)
{
    motorSequence = waveSequence;
    labelModeText.Text = "Sterowanie Falowe";
}

private void driveFullstep_Click(object sender, EventArgs e)
{
    motorSequence = fullstepSequence;
    labelModeText.Text = "Sterowanie Krokowe";
}

private void driveHalfstep_Click(object sender, EventArgs e)
{
    motorSequence = halfstepSequence;
    labelModeText.Text = "Sterowanie Półkrokowe";
}
```

Dane sterowanie jest wybierane kliknięciem odpowiedniego przycisku z GUI na rysunku 2: Sterowanie Falowe, Sterowanie Krokowe lub Sterowanie Półkrokowe. Przypisywana jest wtedy odpowiednia sekwencja bajtów do tablicy motorSequence (która domyślnie zawiera sekwencję do sterowania falowego), po czym użytkownik jest informowany o dokonanym wyborze poprzez etykietę labelModeText umieszczoną nad przyciskami.

- **Wykonywanie kroków**

```
private void rotateLeft_Click(object sender, EventArgs e)
{
    motorDirection = -1;
    PerformSteps(motorSequence, motorDirection);
}

private void rotateRight_Click(object sender, EventArgs e)
{
    motorDirection = 1;
    PerformSteps(motorSequence, motorDirection);
}
```

Kroki są wykonane poprzez wybranie opcji „Lewo” lub „Prawo” z GUI na rysunku 2. Ustawiany jest wtedy odpowiedni kierunek (-1 dla obrotu w lewo, 1 dla obrotu w prawo), po czym wywołana zostaje metoda PerformSteps() z parametrami w postaci sekwencji bajtowej odpowiadającej wybranemu wcześniej sterowaniu oraz kierunku obrotu.

```
private void PerformSteps(byte[] sequence, int direction)
{
    int sequenceLength = sequence.Length;
    int number_of_steps = Convert.ToInt32(numberSteps.Text);
    int stepDelay = Convert.ToInt32(timeSteps.Text);

    for (int i = 0; i < number_of_steps; i++)
    {
        // Obliczanie indeksu w sekwencji
        int stepIndex;
        if (direction == 1)
        {
            stepIndex = i % sequenceLength; // W prawo
        }
        else
        {
            stepIndex = (sequenceLength - (i % sequenceLength) - 1); // W lewo
        }

        // Dane do wysłania
        byte[] data = { sequence[stepIndex] };

        // Wysyłanie danych do urządzenia
        uint bytesWritten = 0;
        FTDI.FT_STATUS status = ftdiDevice.Write(data, data.Length, ref bytesWritten);
        if (status != FTDI.FT_STATUS.FT_OK)
        {
            MessageBox.Show("Błąd wysyłania danych!", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

        // opoźnienie między krokami
        System.Threading.Thread.Sleep(stepDelay);
    }
}
```

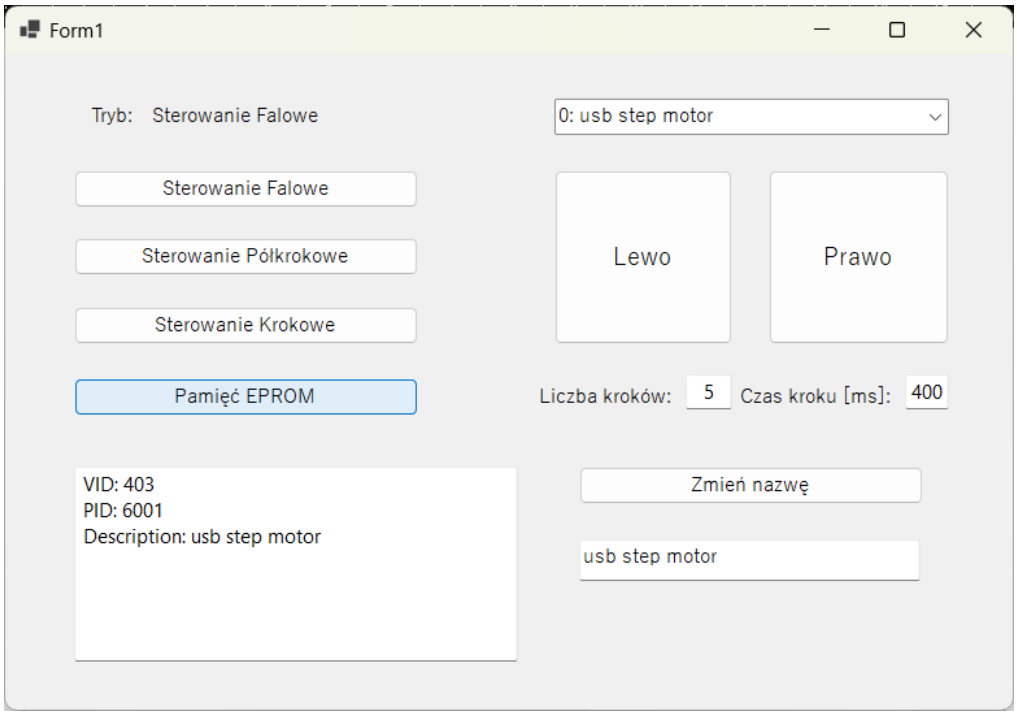
Wewnątrz metody inicjowane są zmienne przechowujące długość sekwencji, liczbę kroków do wykonania oraz przerwę pomiędzy kolejnymi krokami. Wartości ostatnich dwóch zmiennych są pobierane z pól tekstowych w GUI aplikacji.

Następnie wykonana zostaje pętla, wewnątrz której wyznaczany jest najpierw indeks kroku z sekwencji do wykonania. W przypadku obracania w prawo (direction == 1) kolejne bajty są pobierane począwszy od początku tablicy a dla obracania w lewo począwszy od końca tablicy. Potem przygotowywany jest odpowiedni bajt w tablicy data na podstawie wyznaczonego wcześniej indeksu. Bajt ten wysyła metoda *Write()*, czego wynikiem jest uruchomienie odpowiednich uzwojeń i wykonanie kroku przez wirnik silnika krokowego. Iteracja pętli kończy się przeczekaniem czasu, tak aby dostosować się do podanego w GUI czasu pojedynczego kroku.

- **Odczyt pamięci EEPROM**

```
private void EPROM_Click(object sender, EventArgs e)
{
    FTDI.FT232B_EEPROM_STRUCTURE structure = new FTDI.FT232B_EEPROM_STRUCTURE();
    FTDI.FT_STATUS status = ftdiDevice.ReadFT232BEEPROM(structure);
    if (status != FTDI.FT_STATUS.FT_OK)
    {
        MessageBox.Show("Błąd odczytu pamięci EPROM!", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }
    else
    {
        textEPROM.AppendText("VID: " + structure.VendorID.ToString("X"));
        textEPROM.AppendText(Environment.NewLine);
        textEPROM.AppendText("PID: " + structure.ProductID.ToString("X"));
        textEPROM.AppendText(Environment.NewLine);
        textEPROM.AppendText("Description: " + Convert.ToString(structure.Description));
        textEPROM.AppendText(Environment.NewLine);
    }
}
```

Kliknięcie opcji „Pamięć EPROM” powoduje wywołanie metody *EPROM_Click()*, wewnątrz której wywoływana jest na urządzeniu metoda *ReadFT232BEEPROM()* z parametrem w postaci utworzonego wcześniej obiektu „structure” do przechowania pozyskiwanych danych. Jeśli odczyt wykona się pomyślne to w textBox poniżej przycisku wypisywane są podstawowe dane urządzenia takie jak Vendor ID, Product ID oraz Device Description. Wynik działania odczytu przedstawia rysunek 3 poniżej.

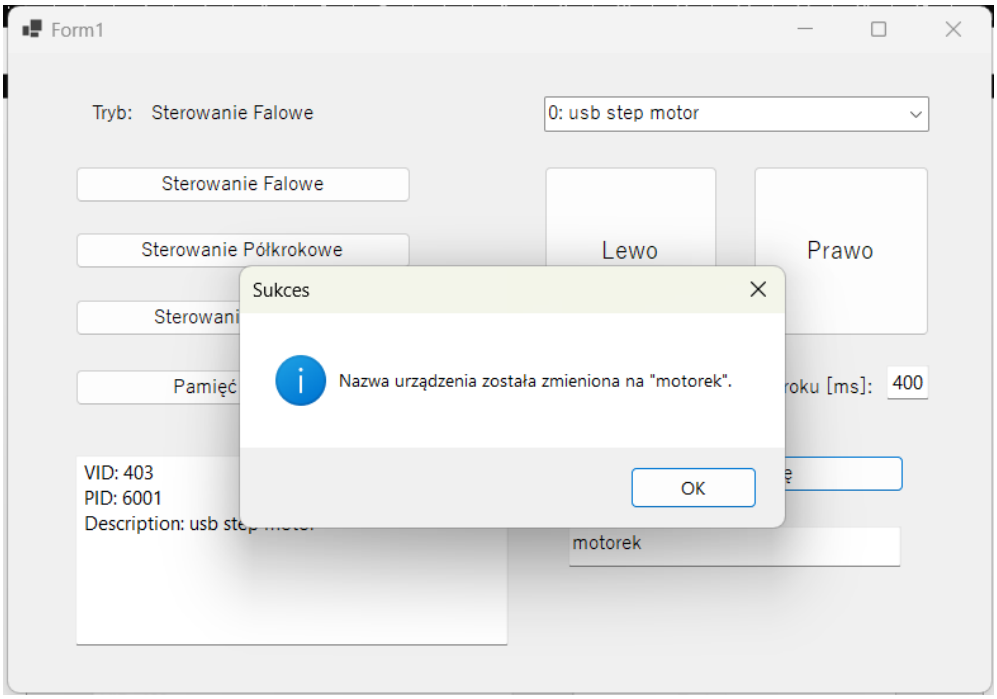


Rys. 3 Odczyt pamięci EEPROM w aplikacji

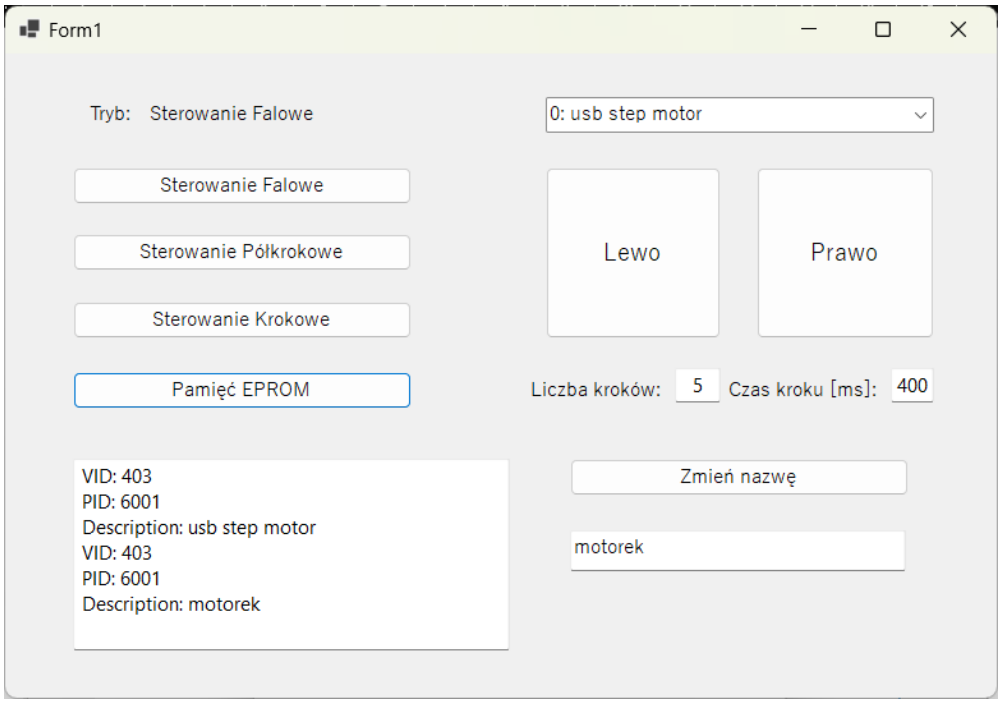
- **Zapis pamięci EEPROM**

```
// modyfikacja pamieci EEPROM
private void buttonName_Click(object sender, EventArgs e)
{
    // Odczyt aktualnej struktury EEPROM
    FTDI.FT232B_EEPROM_STRUCTURE structure = new FTDI.FT232B_EEPROM_STRUCTURE();
    FTDI.FT_STATUS status = ftdiDevice.ReadFT232BEEPROM(structure);
    if (status != FTDI.FT_STATUS.FT_OK)
    {
        MessageBox.Show("Błąd odczytu pamięci EPROM!", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }

    // Ustawienie nowej nazwy
    structure.Description = textBoxName.Text;

    // Zapis nowej struktury do EEPROM
    status = ftdiDevice.WriteFT232BEEPROM(structure);
    if (status != FTDI.FT_STATUS.FT_OK)
    {
        MessageBox.Show("Błąd zapisu do pamięci EPROM!", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
    else
    {
        MessageBox.Show($"Nazwa urządzenia została zmieniona na
\"{structure.Description}\".", "Sukces", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

Funkcjonalność tą zrealizowaliśmy jako możliwość zmiany pola Description. Podobnie jak w metodzie *EPROM_Click()*, odczytywana jest najpierw struktura EEPROM. Na tej strukturze dokonywana jest modyfikacja atrybutu Description poprzez ustawienie na nim zawartości z pola tekstowego. Zmodyfikowana struktura jest zapisywana przez wywołanie na urządzeniu metody *WriteFT232BEEPROM()* z parametrem w postaci zmodyfikowanej struktury. Poprzez MessageBox użytkownik jest informowany o pomyślności lub niepomyślności wykonywanej operacji. Na rysunkach 4 i 5 pokazany został przykład działania tej opcji.



Rys. 4 Modyfikacja pamięci EEPROM



Rys. 5 Wynik modyfikacji widoczny przy odczycie EEPROM (przed zamknięciem aplikacji przywrócona została domyślna wartość atrybutu Description „usb step motor”)

- **Rozwinięcie programu**

Pod koniec laboratorium podczas weryfikacji działania aplikacji w przypadku ustawienia tylko 1 kroku do wykonania zauważono, że kolejne wykonania tego kroku przywracały wirnik do pozycji sprzed jego wykonania po czym krok był wykonywany, co powodowało że wirnik nie przemieszczał się dalej niż o 1 krok od pozycji początkowej. Zaproponowane zostało więc naprawienie tej usterki.

Problemem okazało się nieprawidłowe wyznaczanie indeksu bajta z sekwencji nieuwzględniające przypadku gdy wykonana zostaje tylko część sekwencji. Przez to gdy ponawiane było wykonywanie tylko 1 kroku to do silnika za każdym razem był wysyłany pierwszy bajt danej sekwencji, co powodowało nieoczekiwane zachowanie wirnika. Poniżej przedstawione zostały kolorem pomarańczowym wprowadzone zmiany.

```
private int index = 0;

private void PerformSteps(byte[] sequence, int direction)
{
    int sequenceLength = sequence.Length;
    int number_of_steps = Convert.ToInt32(numberSteps.Text);
    int stepDelay = Convert.ToInt32(timeSteps.Text);

    for (int i = 0, j = index; i < number_of_steps; i++)
    {
        // Obliczanie indeksu w sekwencji
        int stepIndex;
        if (direction == 1)
        {
            stepIndex = ++j % sequenceLength; // W prawo
        }
        else
        {
            stepIndex = (sequenceLength - (++j % sequenceLength) - 1); // W lewo
        }
        index = stepIndex;

        // Dane do wysłania
        byte[] data = { sequence[stepIndex] };

        // Wysyłanie danych do urządzenia
        uint bytesWritten = 0;
        FTDI.FT_STATUS status = ftdiDevice.Write(data, data.Length, ref bytesWritten);
        if (status != FTDI.FT_STATUS.FT_OK)
        {
            MessageBox.Show("Błąd wysyłania danych!", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

        // opoźnienie między krokami
        System.Threading.Thread.Sleep(stepDelay);
    }
}
```

Zadeklarowana została globalna zmienna index do przechowywania indeksu ostatnio wykonanego kroku, która jest wykorzystana w parametrach pętli for metody *PerformSteps()* w postaci parametru „j”. Teraz parametr „i” jest wykorzystywany tylko do określana liczby iteracji pętli, gdyż do obliczania indeksu kolejnego kroku podstawiony został parametr „j”, dzięki czemu nie występuje już błąd w postaci wysyłania cały czas tego samego bajta przy ponawianym wykonaniu pojedynczego kroku. Każdorazowo po wyznaczeniu indeksu kolejnego bajta do wysłania, wartość ta jest zapisywana w zmiennej globalnej index. Reszta metody *PerformSteps()* pozostała bez zmian.

4. Podsumowanie i wnioski

Podczas laboratorium pomyślnie została zweryfikowana poprawność wybranych atrybutów silnika krokowego oraz przetestowane zostało jego działanie za pomocą gotowego programu.

W kolejnym kroku napisana została aplikacja zawierająca wszystkie wymagane funkcjonalności wliczając w to otwieranie oraz zamykanie urządzenia, które uprzednio było wybierane z listy w ComboBox, wybór sposobu sterowania silnikiem, wprowadzenie liczby kroków oraz czasu wykonania pojedynczego kroku, wykonanie zadanej liczby kroków zarówno w prawo jak i lewo oraz odczyt i modyfikacja pamięci EEPROM.

Podczas testowania aplikacji napotkany został jednak problem ze sposobem wykonywania pojedynczego kroku, który wynikał z niewłaściwie wyznaczanego indeksu kolejnego kroku, nieuwzględniającego wyniku końcowego poprzednio wykonanej sekwencji. Problem ten został rozwiązany po zajęciach laboratoryjnych poprzez wprowadzenie nowej zmiennej oraz dokonanie drobnych modyfikacji w kodzie metody odpowiedzialnej za wykonywanie kroków. Ostateczna wersja programu działa zgodnie z wymaganiami.

5. Źródła

http://elportal.pl/pdf/k01/81_24.pdf

https://pl.wikipedia.org/wiki/Silnik_krokowy

https://pl.wikipedia.org/wiki/Silnik_elektryczny