Mariusz Miziołek
Wydział Technologii i Edukacji
Politechnika Koszalińska

Algorytm i implementacja protokołu komunikacyjnego MODBUS w środowisku sterownika PLC firmy IDEC oraz języku programowania C#

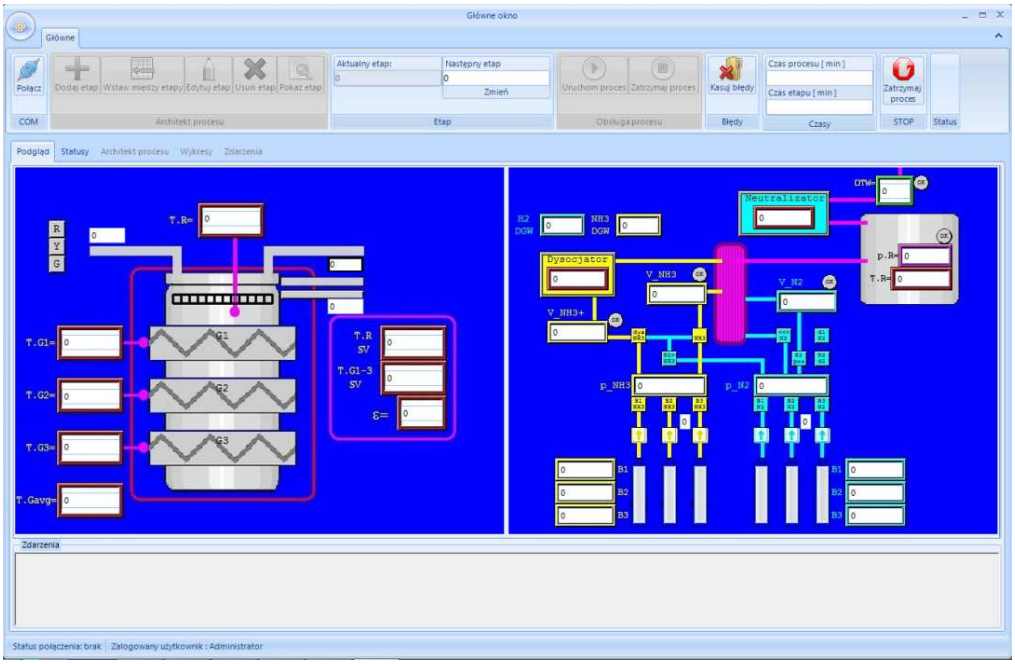
Słowa kluczowe: mikrokontroler, PLC, C#, modbus, protokół komunikacyjny, automatyka przemysłowa, IDEC

1. Wstęp

W przemyśle oraz w życiu codziennym coraz więcej urządzeń komunikuję się ze sobą w celu uzyskania dodatkowych informacji o swoim działaniu lub uzyskania kolejnych poleceń do wykonania. W automatyce przemysłowej istnieje wiele standardów komunikacji ale jednym z najstarszych i najczęściej implementowanych jest MODBUS [1], który posiada dwie najpopularniejsze wersje dla portu szeregowego MODBUS RTU oraz dla sieci Ethernet MODBUS TCP.

Przedstawiony w artykule projekt dotyczy istniejącego rozwiązania protokołu komunikacyjnego MODBUS RTU zaimplementowanego w sterowniku PLC firmy IDEC [2]. Prezentowana praca porusza temat realizacji części algorytmu do budowy zapytania oraz analizy odpowiedzi uzyskanej od sterownika PLC firmy IDEC, przy użyciu języka programowania C# [3]. Zapotrzebowanie na wyżej wymieniony algorytm pojawiło się w trakcie realizacji projektu aplikacji do harmonogramowania procesu azotowania w piecu firmy REMIX znajdującym się w Instytucie Technologii Eksploatacji - PIB w Radomiu. Aplikacja (Rys. 1) wymagała umożliwienia połączenia ze sterownikiem PLC w celu nadzorowania procesu azotowania oraz akwizycji danych procesowych uzyskiwanych w jego trakcie.

W kolejnych punktach określono podstawowe warunki i założenia odnośnie algorytmu dla takiego systemu.



Rys. 1. Interfejs użytkownika aplikacji do harmonogramowania procesu azotowania

2. Główne założenia i zastosowanie

W istniejącym systemie automatyki sterującym pracą pieca do azotowania firmy REMIX został zaimplementowany protokół komunikacji MODBUS RTU. Dlatego oczywistym wyborem okazało się opracowanie algorytmu oraz kodu umożliwiającego wykonanie aplikacji do wygenerowania harmonogramu oraz przesłanie wszystkich informacji niezbędnych do sterownika z jego zastosowaniem.

05h	**	**	30h	52h	**	**	**	**	**	**	**	**	**	0Dh
(1)	(2)		(3)	(4)	(5)	(6)					(7)		(8)		(9)

1	Znak kontrolny komunikacji	1 bajt	ENQ (05h)	Zapytanie
2	Numer urządzenia	2 bajty	00 - 1F FF	Urządzenie od 0 do 31 Wszystkie urządzenia
3	Kontynuacja	1 bajt	30h	Brak kontynuacji
4	Komenda - odczyt	1 bajt	52h	Odczyt danych
5	Typ danych	1 bajt	44h	Dla naszego przykładu odczyt z rejestrów danych
6	Numer operandu	4 bajty	Np. 0000-1999	
7	Długość danych	2 bajty	00 - F0h	F0h - 240 wartość maksymalna
8	BCC	2 bajty	00-7Fh	Suma kontrolna
9	Terminator	1 bajt	0Dh	Zakończenie wiadomości

Rys. 2. Ramka protokołu komunikacyjnego MODBUS

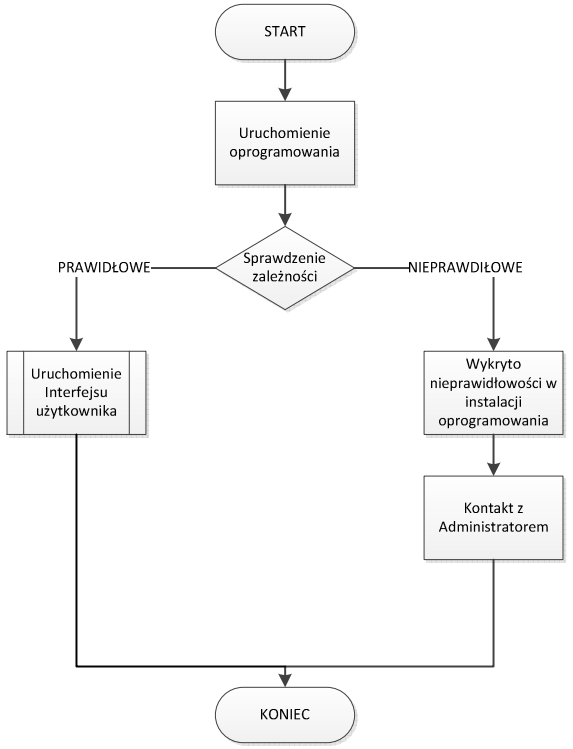
MODBUS RTU jest najczęściej używanym protokołem w komunikacji szeregowej. Wszystkie informacje przesyłane są w postaci ramek, które budowane są na podstawie schematu (Rys. 2). Założeniem jest możliwość przesyłania zapytania i odbierania odpowiedzi ze sterownika w pełnym zastosowaniu protokołu przy pomocy opracowanego algorytmu oraz późniejszego kodu w C#.

3. Algorytm komunikacji

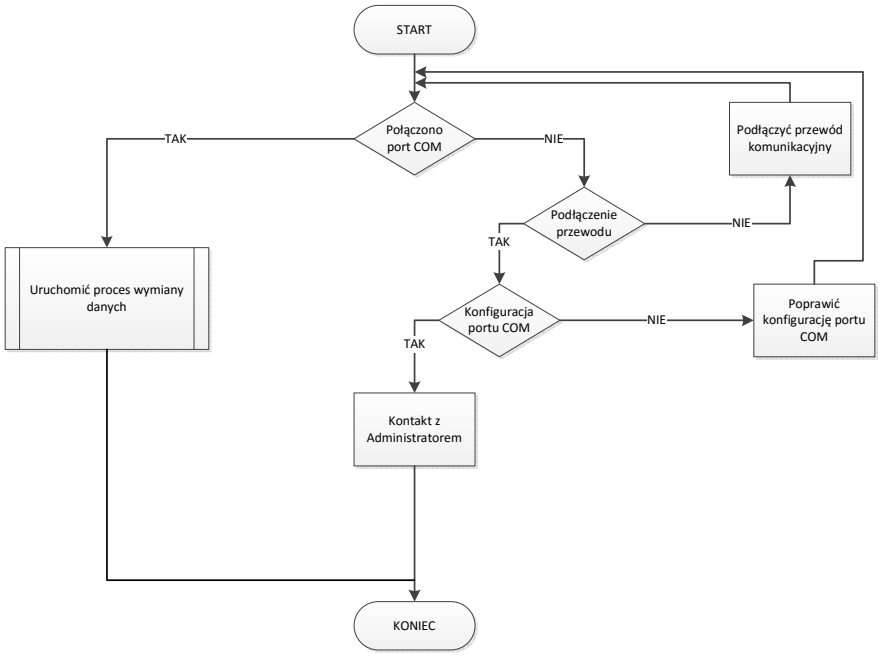
Do zadań algorytmu oraz opracowanego kod C# należy sprawdzenie poprawność uruchomienia interfejsu użytkownika oraz konfiguracji połączenia za pomocą portu szeregowego COM. Następnie w odpowiedniej kolejności wysłanie zapytania do sterownika o parametry procesu oraz ewentualną ich korektę.

3.1. Faza początkowa

Uruchomienie aplikacji i wywołanie fazy początkowej algorytmu sprawdza poprawność wyświetlenia interfejsu użytkownika, konfiguracji portu szeregowego COM.



Rys. 3. Sprawdzenie uruchomienia interfejsu użytkownika



Rys. 4. Sprawdzenie poprawności konfiguracji portu szeregowego.

Funkcja umożliwiająca nawiązanie połączenia ze sterownikiem PLC została przedstawiona na listingu 1. Do prawidłowego działania wymaga przekazania parametrów konfiguracyjnych pracy portu szeregowego COM umieszczonych w tablicy o nazwie "comParams".

W tablicy "comParams" znajdują się tu następujące informacje:

- numer portu COM,
- prędkość transmisji,
- bit parzystości,
- ilość bitów informacji,
- ilość bitów stopu,
- opóźnienie odbioru.

```
public void connect( string[] comParams)
{
    try
    {
        var _with1 = serialport1;
        _with1.PortName = comParams[0];
        _with1.BaudRate = Convert.ToInt32(comParams[1]);
        _with1.Parity = (Parity)Enum.Parse(typeof(Parity), comParams[3] );
        _with1.DataBits = Convert.ToInt32(comParams[2]);
        _with1.StopBits = (StopBits)Enum.Parse(typeof(StopBits), comParams[4]);
        _with1.Handshake = Handshake.None;
        _with1.RtsEnable = false;
        _with1.ReceivedBytesThreshold = Convert.ToInt32(comParams[5]);
        this._threshold = _with1.ReceivedBytesThreshold;
        _with1.ReadTimeout = 5000;
        _with1.ReadBufferSize = 4096;
        _with1.WriteBufferSize = 2048;
        _with1.Encoding = System.Text.Encoding.ASCII;
        this._receiveDelay = Convert.ToInt32(comParams[6]);
        serialport1.Open();
    }
    catch (IOException ex)
    {
        showmessage(ex.Message + " ComOpen IO");

    }
    catch (Exception ex)
    {
        showmessage(ex.Message + " ComOpen EX");

    }
    finally
    {
        isconnected = serialport1.IsOpen;
        if (connection != null)
        {
            connection(isconnected);
        }
    }
}
```

Listing 1. Zaproponowania funkcja połączenia ze sterownikiem za pomocą portu szeregowego COM

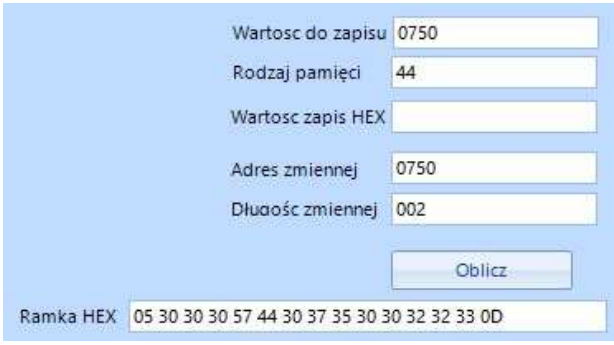
3.2. Wymiana danych

Po uzyskaniu połączenia ze sterownikiem PLC oraz poprawnym wyświetleniu interfejsu użytkownika kolejnym etapem jest wymiana danych. Aby tego dokonać należy zbudować odpowiednio ramkę z zapytaniem MODBUS zgodnie ze schematem (Rys.2). Oprócz ułożenia danych w odpowiedniej kolejności należy przed wysłaniem zapytania do sterownika dokonać kilku konwersji powstałej komendy. Całość zapytania należy rozbić na poszczególne cyfry, następnie zamienić je na kody ASCII a na końcu wszystkie cyfry po zamienianie ponownie złożyć w komendę, którą przesyłamy do sterownika. Na rysunku 5 przedstawiono przykładowe zapytanie.

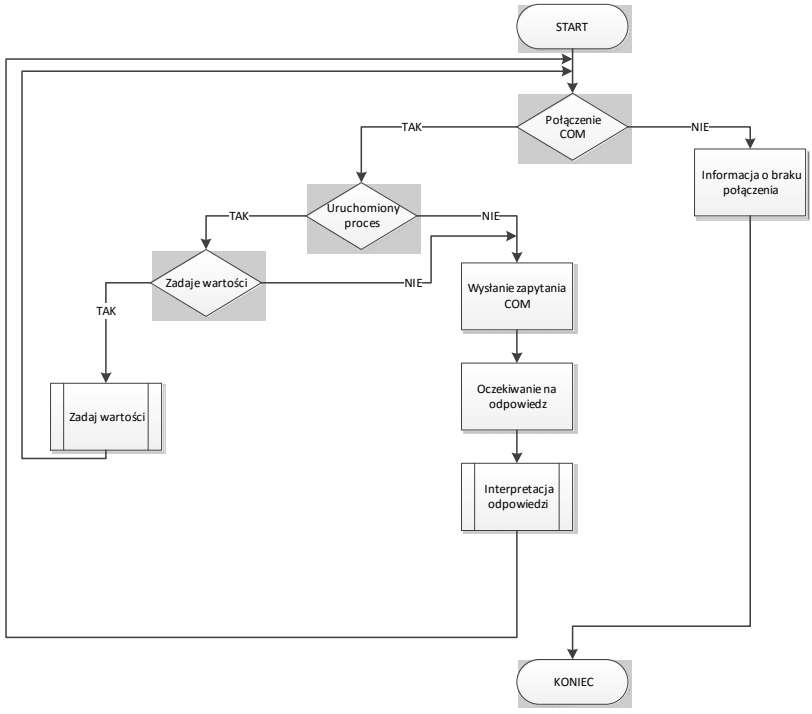
Po weryfikacji poprawności komendy oraz sumy kontrolnej odsyłana jest odpowiedź, która może przybrać kilka form.

Możliwe odpowiedzi na zapytanie:

- potwierdzenie przyjęcia wartości do zapisu,
- potwierdzenie przyjęcia zakresu wartości do zapisu,
- przesłanie wartości komórki pamięci zgodnie ze wskazaniem z zapytania,
- przesłanie zakresu wartości komórek pamięci zgodnie ze wskazaniem z zapytania.



Rys. 5. Przykładowa ramka z zapytaniem



Rys. 6. Wymiana danych przy użyciu protokołu MODBUS

```
private void Oblicz_Click(object sender, EventArgs e)
{

    adres0 = Convert.ToChar(TadresZmiennej.Text.Substring(0, 1));
    adres1 = Convert.ToChar(TadresZmiennej.Text.Substring(1, 1));
    adres2 = Convert.ToChar(TadresZmiennej.Text.Substring(2, 1));
    adres3 = Convert.ToChar(TadresZmiennej.Text.Substring(3, 1));

    dlugosc0 = Convert.ToInt32(TdlugoscZmiennej.Text);
    string dlugosc = "";
    string HexA0 = ((int)adres0).ToString("X");
    string HexA1 = ((int)adres1).ToString("X");
    string HexA2 = ((int)adres2).ToString("X");
    string HexA3 = ((int)adres3).ToString("X");
    string HexD1 = "";
    string HexD2 = "";
    string HexXOR1 = "";
    string HexXOR2 = "";
    if (dlugosc0 < 15)
    {
        char czlon1 = '0';
        char czlon2 = Convert.ToChar(dlugosc0.ToString("X"));
        HexD1 = ((int)czlon1).ToString("X");
        HexD2 = ((int)czlon2).ToString("X");
    }
    else
    {
        string HexDlugosc = dlugosc0.ToString("X");
        char czlon1 = Convert.ToChar(HexDlugosc.Substring(0, 1));
        char czlon2 = Convert.ToChar(HexDlugosc.Substring(1, 1));
        HexD1 = ((int)czlon1).ToString("X");
        HexD2 = ((int)czlon2).ToString("X");
    }
    int czesc1 = Convert.ToInt32("05", 16) ^ Convert.ToInt32("30", 16) ^ Convert.ToInt32("30", 16) ^ Convert.ToInt32("30", 16);
    int czesc2 = czesc1 ^ Convert.ToInt32("52", 16) ^ Convert.ToInt32(TrodzajPamieci.Text, 16) ^ Convert.ToInt32(HexA0, 16) ^
Convert.ToInt32(HexA1, 16);
    int czesc3 = czesc2 ^ Convert.ToInt32(HexA2, 16) ^ Convert.ToInt32(HexA3, 16) ^ Convert.ToInt32(HexD1, 16) ^
Convert.ToInt32(HexD2, 16);
    string hexResult = czesc3.ToString("X");
    if (czesc3 > 15)
    {
        char Xor1 = Convert.ToChar(hexResult.Substring(0, 1));
        char Xor2 = Convert.ToChar(hexResult.Substring(1, 1));
        HexXOR1 = ((int)Xor1).ToString("X");
        HexXOR2 = ((int)Xor2).ToString("X");
    }
    else
    {
        char Xor1 = Convert.ToChar('0');
        char Xor2 = Convert.ToChar(hexResult.Substring(0, 1));
        HexXOR1 = ((int)Xor1).ToString("X");
        HexXOR2 = ((int)Xor2).ToString("X");
    }
    TramkaHEX.Text = "05 30 30 30 57 " + TrodzajPamieci.Text + " " + HexA0 + " " + HexA1 + " "
    + HexA2 + " " + HexA3 + " " + HexD1 + " " + HexD2 + " " + HexXOR1 + " " + HexXOR2 + " 0D";

}
```

Listing 2. Zaproponowana funkcja do budowy ramki zapytania MODBUS

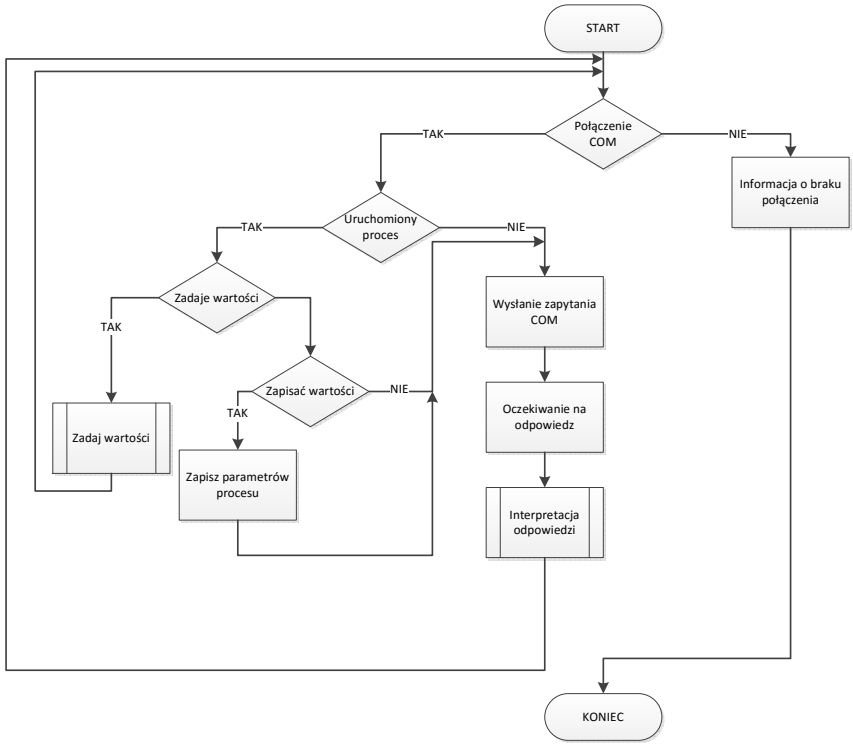
Listing 2 przedstawia funkcję zrealizowaną w języku C# za pomocą, której realizowana jest budowa ramki do wysłania zapytania MODBUS. Do wyliczenia poprawnej wartości pobierane są dane odnośnie rodzaju pamięci, adresu zmiennej, długości zmiennej oraz funkcji realizowanej (odczyt pojedynczy, odczyt wielu zmiennych, zapis pojedynczy, zapis wielu zmiennych). Wynik działania funkcji przedstawiony został na rysunku 5 w polu "Ramka HEX". W ostatnim etapie działania funkcji do zmiennej "TramkaHEX" zapisywany jest wynik działania poszczególnych przekształceń jako wynik całego działania funkcji (Listing 3).

```
TramkaHEX.Text = "05 30 30 30 57 " + TrodzajPamieci.Text + " " + HexA0 + " "
                 + HexA1 + " " + HexA2 + " " + HexA3 + " " + HexD1 + " "
                 + HexD2 + " " + HexXOR1 + " " + HexXOR2 + " 0D";
```

Listing 3. Złożenie wartości składających się na ramkę MODBUS

3.3. Wymiana i akwizycja danych

Algorytm przewiduję również dodatkowy tryb pracy z możliwością zapisu uzyskanych odpowiedzi od sterownika do bazy danych (Rys. 7.).



Rys. 7. Wymiana i akwizycja danych przy użyciu protokołu MODBUS

Poniższy listing nr 4 przedstawia realizacje funkcji, której zadaniem jest przetworzenie otrzymanej odpowiedzi ze sterownika na wartości liczbowe. Następnie są prezentowane na panelu użytkownika oraz dodatkowo mogą zostać zapisane do bazy danych.

```
private void radTextBox2_TextChanged(object sender, EventArgs e)
    {
        if (komorka > 0)
        {
            proba++;
            try
            {
                int dl = radTextBox2.Text.Length;
                bool znalazlemKoniec = false;
                try
                {
                    string kon = radTextBox2.Text.Substring(radTextBox2.Text.Length - 2, 2).ToString();
                    if (kon.Equals("0D"))
                        znalazlemKoniec = true;
                }
                catch { }
                if (znalazlemKoniec)
                {
                    if (pamiecM)
                    {
                        int int1 = 0;
                        int int2 = 0;
                        int int3 = 0;
                        int int4 = 0;
                        string bezPoczatku = radTextBox2.Text.Remove(0, 8);
                        string wartosc = bezPoczatku.Remove(bezPoczatku.Length - 6, 6);
                        int i = 1;
                        int j = 0;
                        int x = 1;
                        string[] hex = new string[komorka];
                        int v = 0;
                        string str = wartosc;
                        int b = (wartosc.Length / 8);
                        int z = 0;
                        int y = 0;
                        for (int c = 0; c < b; c++)
                        {
                            try
                            {
                                v++;
                                int1 = Convert.ToInt32(str.Substring(0, 8).Substring(0, 2), 16);
                                int2 = Convert.ToInt32(str.Substring(0, 8).Substring(2, 2), 16);
                                int3 = Convert.ToInt32(str.Substring(0, 8).Substring(4, 2), 16);
                                int4 = Convert.ToInt32(str.Substring(0, 8).Substring(6, 2), 16);
                                str = str.Remove(0, 8);
                                v++;
                                char char1 = (char)int1;
                                char char2 = (char)int2;
                                char char3 = (char)int3;
                                char char4 = (char)int4;
                                string wartoscHex1 = char3.ToString() + char4.ToString();
                                string wartoscHex2 = char1.ToString() + char2.ToString();
                                try
                                {
                                    string[] bit1 = rozszerzBityM(ToBinary(Convert.ToInt32(wartoscHex1, 16)));
                                    string[] bit = rozszerzBityM(ToBinary(Convert.ToInt32(wartoscHex2, 16)));
                                    tablicaM[adress] = bit[0];
                                    adress++;
                                    tablicaM[adress] = bit[1];
                                    adress++;
                                    tablicaM[adress] = bit[2];
                                    adress++;
                                    tablicaM[adress] = bit[3];
                                    adress++;
                                    tablicaM[adress] = bit[4];
                                    adress++;
                                    tablicaM[adress] = bit[5];
                                    adress++;
                                    tablicaM[adress] = bit[6];
                                    adress++;
                                    tablicaM[adress] = bit[7];
                                    adress = adress + 3;
                                    tablicaM[adress] = bit1[0];
                                    adress++;
                                    tablicaM[adress] = bit1[1];
                                    adress++;
```

```
                    tablicaM[adress] = bit1[2];
                    adress++;
                    tablicaM[adress] = bit1[3];
                    adress++;
                    tablicaM[adress] = bit1[4];
                    adress++;
                    tablicaM[adress] = bit1[5];
                    adress++;
                    tablicaM[adress] = bit1[6];
                    adress++;
                    tablicaM[adress] = bit1[7];
                    adress++;
                }
                catch { }
                wolnyPort = true;
                pamiecM = false;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
                wolnyPort = true;
                pamiecM = false;
            }
        }
        pamiecM = false;
        wolnyPort = true;
    }
    else
    {
        if (dl == ((komorka + 10) * 8) + 14)
        {
            int int1 = 0;
            int int2 = 0;
            int int3 = 0;
            int int4 = 0;
            string bezPoczatku = radTextBox2.Text.Remove(0, 8);
            string wartosc = bezPoczatku.Remove(bezPoczatku.Length - 6, 6);
            int i = 1;
            int j = 0;
            int x = 1;
            string[] hex = new string[komorka];
            int v = 0;
            string str = wartosc;
            int b = (wartosc.Length / 8);
            int z = 0;
            int y = 0;
            for (int c = 0; c < b; c++)
            {
                try
                {
                    v++;
                    int1 = Convert.ToInt32(str.Substring(0, 8).Substring(0, 2), 16);
                    int2 = Convert.ToInt32(str.Substring(0, 8).Substring(2, 2), 16);
                    int3 = Convert.ToInt32(str.Substring(0, 8).Substring(4, 2), 16);
                    int4 = Convert.ToInt32(str.Substring(0, 8).Substring(6, 2), 16);
                    str = str.Remove(0, 8);
                    v++;
                    char char1 = (char)int1;
                    char char2 = (char)int2;
                    char char3 = (char)int3;
                    char char4 = (char)int4;
                    string wartoscHex = char1.ToString() + char2.ToString() + char3.ToString()
                                        + char4.ToString();
                    tablica[adress] = wartoscHex;
                    adress++;
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.ToString());
                    wolnyPort = true;
                }
            }
        }
        wolnyPort = true;
    }
}
```

```
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
                wolnyPort = true;
            }
        }
    }
```

Listing 4. Zaproponowana funkcja interpretująca odpowiedź sterownika.

Funkcja przetwarzająca odpowiedź sterownika przedstawiona na listingu 4 w wyniku swojego działania i przekształceń prowadzonych na odpowiedzi zwraca tablicę "tablica", w której indeks odpowiada adresowi zmiennej, a zawartość jest odczytaną wartością zmiennej zaprezentowaną w formie dziesiętnej.

4. Podsumowanie i możliwości rozwoju

Opracowanie algorytmu do obsługi protokołu MODBAS RTU przy użyciu języka programowania C# powstało w celu zaimplementowania tego rozwiązania w aplikacji do harmonogramowania procesu azotowania (Rys. 1), która powstała dla Instytutu Technologii Eksploatacji - PIB w Radomiu. Poprawność algorytmu została zweryfikowania poprzez przeprowadzenie procesu stabilizacji retorty oraz azotowania które trwały odpowiednio 24 godziny i 8 godzin. Algorytm można rozwijać poprzez optymalizacje czasu przetwarzania odpowiedź, dla zadania w którym został użyty nie zaistniała dodatkowa potrzeba zwiększenia prędkości działania.

Bibliografia

1. MODBUS [online], http://www.modbus.org/
2. IDEC [online], http://www.idec.com/
3. C# [online], https://msdn.microsoft.com/pl-pl/library/kx37x362.aspx
4. Kwaśniewski J.: *Sterowniki PLC w praktyce inżynierskiej*, ISBN 9788360233351 Wydawnictwo BTC, 2008.
5. Praca grupowa: *Programowanie równoległe i asynchroniczne w C# 5.0*, ISBN 9788324666980, Wydawnictwo Helion , 2013.
6. Templeman J., Vitter D.: *Visual Studio .NET: .NET Framework. Czarna księga*, ISBN 8371977336, Wydawnictwo Helion , 2003
7. Daniluk A.: *RS 232C - praktyczne programowanie. Od Pascala i C++ do Delphi i Buildera. Wydanie III*, ISBN 8324607781, Wydawnictwo Helion, 2007.

Abstract

The article concerns the construction and use of an algorithm for communication via MODBUS RTU communication protocol used in IDEC PLCs. The main task of this algorithm is to build frames in MODBUS RTU standard transmission for their assistance requests to the controller and the interpretation of the replies sent by the driver. The practical part of the work brought to the development of C # code for developed algorithm and necessary validation activities for later use in other applications.

Keywords: microcontroller, PLC, C#, Modbus communication protocol, industrial automation, IDEC

Streszczenie

Artykuł dotyczy budowy oraz zastosowania algorytmu do komunikacji przez protokół komunikacyjny MODBUS RTU wykorzystywany w sterownikach PLC firmy IDEC. Głównym zadaniem tego algorytmu jest budowanie ramek w standardzie MODBUS RTU przesyłanie za ich pomocą zapytania do sterownika oraz interpretacja odpowiedzi przesyłanych przez sterownik. Część praktyczna pracy sprowadziła się do opracowania kodu C# dla opracowanego algorytmu oraz wykonanie niezbędnych weryfikacji poprawności działania w celu późniejszego wykorzystania w inne aplikacji.