Mateusz Krawczak 241318

Karol Jaskółka 241306

Grupa: Pon P 17:00

Data wykonania ćwiczenia: 13.11.2019

**Urządzenia Peryferyjne**

**Ćwiczenie 18 – Analizator parametrów sieci**

# **Wstęp**

Na drugich zajęciach laboratoryjnych otrzymaliśmy zadanie polegające na napisaniu programu pozwalającego na odczytanie różnych parametrów m.in. napięcia prądu dzięki analizatorowi parametrów sieci EMA-90N firmy Contrel. Zadania do wykonania:

 Połączyć urządzenie EMA-90N z komputerem za pomocą komunikacji Ethernet.

 Uruchomić aplikację demonstracyjną i połączyć się z urządzaniem odczytując napięcie i prąd na L1.

 Napisać aplikację w C#, która połączy się z urządzeniem i umożliwi odczytanie napięcia i prądu L1 z użyciem protokołu modbus.

1. **Zagadnienia**

* **Protokół Modbus:**

- protokół komunikacyjny opracowany przez firmę Modicon

- wykorzystuję regułę wymiany danych typu master-slave (nadrzędny-podrzędny)

- wykorzystywany do znakowej wymiany informacji pomiędzy urządzeniami

* **Modbus TCP/IP:**

- to protokół Modbus RTU z interfejsem TCP/IP

- wykorzystuję sieć Ethernet do transportu danych

- dla aplikacji wykorzystuje port systemowy 502

* **Modbus TCP/IP ramka:**

**-** z ramki Modbus RTS usuwa się pole adresu (indentyfikator slave’a) oraz pole sumy kontrolnej CRC, natomiast dalej wykorzystuję się pole kodu funkcji i danych

**-** dodatkowo ramka TCP/IP rozrasta się o 7 bajtowy nagłówek MBAP (Modbus Application Header)

Obraz zawierający zrzut ekranu

Opis wygenerowany automatycznie

* **Nagłówek MBAP :**

- identyfikator transakcji (2 bajty): do identyfikacji kolejnego zapytania w ramach jednego połączenia TCP

- identyfikator protokołu (2 bajty): obecnie niewykorzystywane ( ustawione na zero 0000)

- długość pola: liczba bajtów wiadomości (2 bajty)

- identyfikator jednostki (1 bajt) : identyfikuje podłączonego klienta

* **Wybrane kody funkcji:**

- 0x01 odczyt wyjść bitowych

- 0x02 odczyt wejść bitowych

- 0x03 odczyt n rejestrów wyjściowych

- 0x04 odczyt n rejestrów wejściowych

* **Numery rejestrów:**

- Discrete Output Coils: 1-9999

- Discrete Input Contacts: 10001-19999

- Analog Input Registers: 30001-39999

- Analog Output Holding Registers: 40001-49999

* **Przykładowa analiza ramek (według przykładu na simplymodbus.ca):**

**Request :**



- 0006 : identyfikator transakcji

- 0000 : identyfikator protokołu

- 0006 : długość wiadomości (6 kolejnych bajtów)

- 01: identyfikator jednostki 0x01

- 03: kod funkcji ( 0x03 read analog output holding registers)

- 0000: adres pierwszego odpytywanego rejestru (40001)

- 0014: ilość odpytywanych rejestrów heksadecymalnie 0x14

**Response**:



- 0006 : identyfikator transakcji

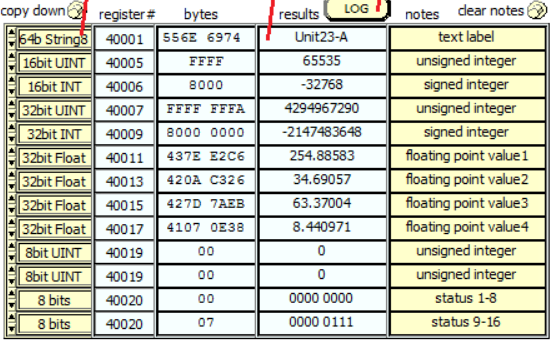
- 0000 : identyfikator protokołu

- 003B : długość wiadomości heksadecymalnie 0x3B

- 01: identyfikator jednostki 0x01

- 03: kod funkcji ( 0x03 read analog output holding registers)

- pozostałe bajty to zawartości rejestrów jak na zdjęciu poniżej



Rysunek 1 Przykład ze strony http://www.simplymodbus.ca/

1. **Przebieg zajęć i kod programu**

* Na początku zajęć udało nam się połączyć urządzenie EMA-90N z komputerem za pomocą komunikacji Ethernet i uruchomić aplikację demonstracyjną
* Następnie przystąpiliśmy do napisania aplikacji w C# wykorzystując bibliotekę EasyModbus
* Zgodnie z zaleceniem ze strony z zadaniem pole UnitIdentifier (identyfikator jednostki) zostało ustawione na wartość 0x01
* Poniższy kod powstał w oparciu o dokumentację biblioteki EasyModbus

Cały program znajduję się pod tym linkiem <https://github.com/matson19/UP/tree/master/Lab%202%20-%20analizator_sieci>

public partial class Form1 : Form

{

//obiekt klasy modbus potrzebny do odczytania parametrow

ModbusClient modbusClient;

public Form1()

{

InitializeComponent();

}

private void buttonConnect\_Click(object sender, EventArgs e)

{

try

{

//inicjalizacja ip podanego przez użytkownika oran numer portu

modbusClient = new ModbusClient(textBoxIP.Text, 502);

modbusClient.UnitIdentifier = 0x01;

modbusClient.ConnectionTimeout = 350;

modbusClient.Connect();

labelStatus.Text = "Connected";

timer.Start();

} catch(Exception ex)

{

labelStatus.Text = ex.ToString();

}

}

private void buttonDisconnect\_Click(object sender, EventArgs e)

{

modbusClient.Disconnect();

labelStatus.Text = "Offline";

}

private void timer\_Tick(object sender, EventArgs e)

{

modbusClient.WriteMultipleCoils(4, new bool[] { true, true, true, true, true, true, true, true, true, true }); //Write Coils starting with Address 5

bool[] readCoils = modbusClient.ReadCoils(9, 10); //Read 10 Coils from Server, starting with address 10

int[] readHoldingRegisters = modbusClient.ReadHoldingRegisters(0, 10); //Read 10 Holding Registers from Server, starting with Address 1

for (int i = 0; i < readCoils.Length; i++)

{

textBox1.AppendText("Value of Coil " + (9 + i + 1) + " " + readCoils[i].ToString());

}

for (int i = 0; i < readHoldingRegisters.Length; i++)

{

textBox2.AppendText("Value of HoldingRegister " + (i + 1) + " " + readHoldingRegisters[i].ToString());

modbusClient.Disconnect();

}

timer.Stop();

}

Próby uruchomienia kody zakończyły się niepowodzeniem oraz uzyskaniem błędu EasyModbus.Exception.FunctionCodeNotSupportedException: ‘Function code not supported by master’

1. **Podsumowanie**

Program nie działał całkowicie poprawnie, ponieważ pomimo faktu, że pogram wykrywał urządzenie to przy próbie odczytania zawartości rejestrów otrzymywaliśmy wyżej wymieniony błąd. W związku z czym nie byliśmy w stanie odczytać parametrów prądu. Kod programu został wykonany zgodnie z dokumentacją EasyModbus, a mimo tu to podczas kompilacji wystąpił błąd. Był on prawdopodobnie spowodowany problemem programu z określeniem które urządzenie funkcjonuję jako master, a które jako slave. Działanie protokołu modbus TCP/IP oraz analiza ramek została dokładnie przestawiona w punkcie numer 2 (Zagadnienia).