

UNIwersYTET W BIAŁYMSTOKU
WYDZIAŁ MATEMATYKI I INFORMATYKI
INSTYTUT INFORMATYKI

Grzegorz Lewandowski

INTERFEJS W JĘZYKU C I PHP
DO KOMUNIKACJI KOMPUTERA
Z PROGRAMOWALNYM
STEROWNIKIEM LOGICZNYM

*Praca dyplomowa napisana
pod kierunkiem
dr Mariusza Żynela*

Białystok 2019

Spis treści

Wstęp	1
1 Sterowniki PLC	3
1.1 Historia	3
1.2 Rozwój	6
1.3 Zastosowania	7
1.4 Zasady działania	7
1.5 Urządzenia wejścia i wyjścia	8
1.6 Programowanie drabinkowe	9
2 Rozwiązania oparte na PLC	10
2.1 Dom jednorodzinny lub mieszkanie	10
2.2 Alarm antywłamaniowy	10
2.3 Parking samochodowy	12
2.4 Zbiornik z grzałką	14
3 Protokół komunikacyjny FACON	17
3.1 Urządzenia nadrzędne i podrzędne	17
3.2 Format komunikatu	18
3.3 Kody błędów	20
3.4 Polecenia	20
3.4.1 Zestawienie dostępnych rejestrów	20
3.4.2 Zestawienie dostępnych poleceń	21
3.4.3 Polecenie 0x40 – odczyt uproszczonego statusu sterownika	23
3.4.4 Polecenie 0x41 – uruchomienie i zatrzymanie sterownika	24
3.4.5 Polecenie 0x42 – zapis statusu rejestru 1-bitowego . . .	25
3.4.6 Polecenie 0x43 – odczyt statusu kolejnych rejestrów 1-bitowych	27
3.4.7 Polecenie 0x44 – odczyt wartości kolejnych rejestrów 1-bitowych	28
3.4.8 Polecenie 0x45 – zapis wartości kolejnych rejestrów 1-bitowych	29
3.4.9 Polecenie 0x46 – odczyt wartości kolejnych rejestrów 16 lub 32-bitowych	30

3.4.10	Polecenie 0x47 – zapis wartości kolejnych rejestrów 16 lub 32-bitowych	32
3.4.11	Polecenie 0x48 – odczyt wartości dowolnych rejestrów .	33
3.4.12	Polecenie 0x49 – zapis wartości dowolnych rejestrów . .	35
3.4.13	Polecenie 0x4E – testowa pętla zwrotna	36
3.4.14	Polecenie 0x4F – odczyt programu ze sterownika	38
3.4.15	Polecenie 0x50 – zapis programu do sterownika	39
3.4.16	Polecenie 0x53 – odczyt szczegółowego statusu sterownika	40
4	Biblioteki programistyczne	43
4.1	API w C	43
4.1.1	Stałe	43
4.1.2	Struktury	44
4.1.3	Zmienne globalne	45
4.1.4	Funkcje	46
4.2	API w PHP	50
4.2.1	Stałe	50
4.2.2	Klasy	50
	Spis tabel	55
	Spis rysunków	56
	Bibliografia	58

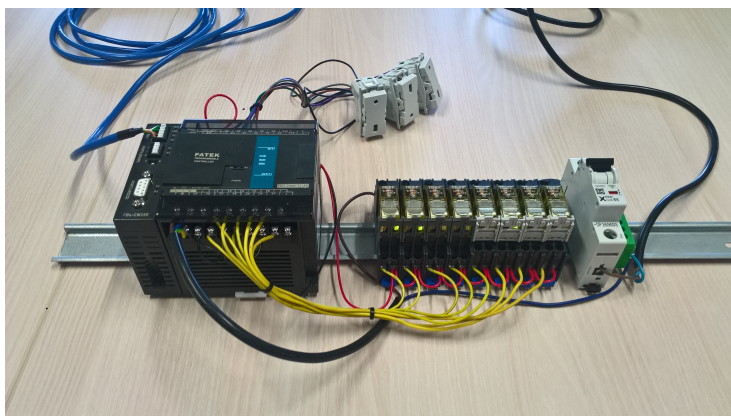
Wstęp

Programmable Logic Controller, w skrócie PLC, to urządzenie mikroprocesorowe pozwalające zautomatyzować pracę wielu różnych urządzeń począwszy od prostej żarówki czy elektrozaworu, a skończywszy na skomplikowanej obrabiarce laserowej albo platformie wiertniczej. Do tego należy dołożyć możliwość analizy wielu czujników, detektorów czy wskaźników takich jak termometr albo licznik przepływowy cieczy. Interakcja użytkownika z takim sterownikiem może odbywać się za pomocą zwykłych przełączników umieszczonych na tablicy rozdzielczej, w szafie elektrotechnicznej, albo na ścianie budynku. O wiele wygodniej byłoby obsługiwać PLC za pomocą ekranu dotykowego na smartfonie, tablecie albo komputerze. Poza ergonomią, mniejszymi rozmiarami panelu dochodzi mobilność i łatwość instalacji. Są oczywiście dostępne rozwiązania komercyjne realizujące ten pomysł, ale są one albo bardzo drogie, albo wyspecjalizowane do konkretnych zastosowań. Ich podstawą działania jest komunikacja z PLC.

Obecnie PLC stosowane jest nie tylko w przemyśle ale także do sterowania urządzeniami w domu. Taki dom z PLC, mnóstwem kabli w ścianach, podłogach i sufitach, naszpikowany czujnikami, termometrami i innymi detektorami nazywa się *inteligentnym*. Inspiracją do powstania interfejsu komunikacyjnego w ramach tej pracy było oprogramowanie do obsługi właśnie inteligentnego domu, które powstaje równoległe w pracy [1]. Dzisiaj każdy ma smartfona i zamiast wstawać by włączyć światło, albo podkręcić piec, chciałby to zrobić nie ruszając się z wygodnego fotela. Do tego przydałyby się z grubsza 3 rzeczy: PLC, serwer HTTP i aplikacja webowa. Pomędzy PLC i serwerem HTTP potrzebne jest z kolei coś, co zapewni wymianę danych. W ramach tej pracy wypełniamy tę właśnie lukę.

Aby w ogóle praca mogła powstać niezbędny sterownik PLC otrzymaliśmy do testów z firmy Elektrokomplex spod Białegostoku specjalizującej się w instalacjach i konserwacji inteligentnych budynków. Jest to urządzenie marki Fatek, model FBs-20MCT2-AC wraz z dodatkowym modułem komunikacyjnym FBs-CM25E. Moduł posiada dwa złącza RS-232 i najważniejsze dla nas złącze ethernetowe do komunikacji TCP/IP.

Na sterowniku działa program drabinkowy, który piszę się za pomocą aplikacji WinProLadder. Do testów, ale i docelowo aby nasz inteligentny dom zadział, musieliśmy sami opracować program drabinkowy. Ponieważ aplikacja



Rysunek 1: Zestaw urządzeń używanych w ramach pracy licencjackiej.

do obsługi idomu będzie na serwerze HTTP z modułem PHP, oczywistym było, że potrzebujemy interfejs do komunikacji z Fatek PLC napisany w PHP. Z drugiej strony, oprogramowanie drabinkowe zapewnia poprawną pracę PLC, ale nie przechowuje historii zmian rejestrów, co przydałoby się do celów statystycznych, tworzenia raportów i wykresów. Tutaj można stworzyć dość proste oprogramowanie działające bez przerwy na serwerze odczytujące dane z rejestrów i zapisujące w bazie danych. Taki program najlepiej napisać w C. Dlatego też ten sam interfejs do komunikacji z Fatek PLC napisaliśmy w C. Ponieważ składniowo PHP jest podobne do C (nie na odwrót), mieliśmy ułatwione zadanie. W PHP zdecydowaliśmy się jednak na podejście obiektowe.

Rozdział 1

Sterowniki PLC

PLC (PROGRAMMABLE LOGIC CONTROLLER) – mikroprocesorowe urządzenie używane do zarządzania systemem zależnych od siebie urządzeń w celu automatycznej współpracy i uniknięcia stałej kontroli manualnej. Jest to przełomowa myśl techniczna w przemyśle, która w znacznym stopniu ułatwiła zarządzanie zarówno prostymi, jak i złożonymi układami połączonych ze sobą urządzeń.[2] Pierwsze programowalne kontrolery były po prostu zamiennikami przekaźników. Ich podstawową funkcją było wykonywanie operacji sekwencyjnych. Operacje te obejmowały sterowanie: WŁĄCZANIE/WYŁĄCZANIE maszyn i procesów, które wymagały powtarzalnych operacji. Jednak sterowniki programowalne dają więcej możliwości niż proste przekaźniki. Łatwa instalacja, o wiele mniejsze zużycie miejsca i energii, możliwość rekonfiguracji i wielokrotnego użycia, obecność wskaźników diagnostycznych, które pomagają w rozwiązywaniu problemów przemawiają na korzyść sterowników programowalnych PLC.

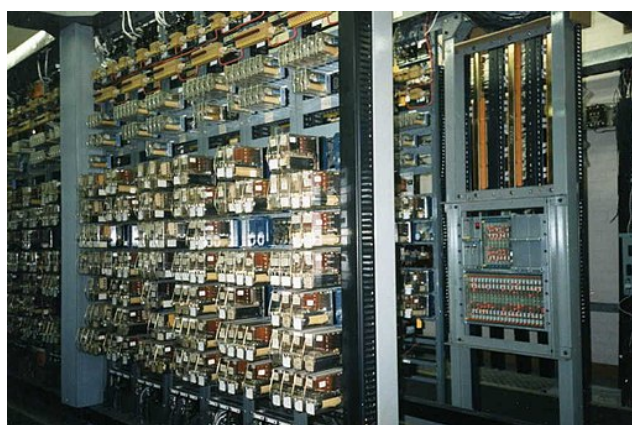
Głównymi zaletami sterowników PLC są:

- uniwersalność, która umożliwia zarządzanie różnymi typami urządzeń,
- niezawodność działania w każdych warunkach,
- zabezpieczenia przed błędnym oprogramowaniem,
- cykliczne działanie równoległe i
- prostota programowania.

1.1 Historia

Przed powstaniem PLC jedynym sposobem sterowania maszynami było użycie przekaźników. Przekaźniki wykorzystują cewkę, która po wzbudzeniu wytwarza siłę magnetyczną, aby skutecznie pociągnąć przełącznik do pozycji ON lub OFF. Gdy przekaźnik jest odłączony od zasilania, przełącznik zwalnia

i przywraca urządzenie do standardowej pozycji ON lub OFF. Było to proste rozwiązanie, umożliwiające włączanie i wyłączanie zasilania dla silników urządzeń na liniach produkcyjnych. Problem polegał na tym, że każdy silnik potrzebował oddzielnego przełącznika, a z uwagi na częstą zależność między urządzeniami przełączniki również musiały być kontrolowane przez przełączniki sterujące. Do tego należy dodać wiele różnych przełączników do obsługi możliwych przypadków zdarzeń na linii produkcyjnej. W efekcie w fabrykach stały wielkie szafy elektryczne pełne przełączników mocy (rysunek 1.1). W takiej płątanie kabli i przełączników dokonanie modyfikacji było bardzo trudne lub niemożliwe. Wymagane by było rozmontowanie całej instalacji, zaprojektowanie jej i zbudowanie na nowo.



Rysunek 1.1: System przełączników używany przed stworzeniem PLC [4].

W 1968 roku firma General Motors w USA uznała, że dotychczasowa technologia używana na liniach produkcyjnych jest zbyt przestarzała i ograniczały możliwości dalszego rozwoju w tej dziedzinie. Ich głównym celem było wyeliminowanie wysokich kosztów związanych z nieelastycznymi systemami, które po każdej modyfikacji trzeba było tworzyć na nowo. Potrzebne było nowe i bardziej wydajne rozwiązanie. Specyfikacje wymagały systemu półprzewodnikowego dostosowanego do komputera spełniającego poniższe warunki:

- sprawnie działać w trudnych warunkach,
- łatwy w programowaniu i obsłudze przez techników,
- być wielokrotnego użytku,
- interfejsy wejściowe i wyjściowe musiały być łatwo wymienne,
- urządzenie w formie modułowej (łatwość wymiany podzespołów),
- przekazywanie danych do systemu centralnego,
- konkurencyjny cenowo wobec poprzedniego rozwiązania.



Rysunek 1.2: Pierwszy PLC [5].

Pracami nad tym zajęła się firma Bedford Associates. Dzięki współpracy Dicka Morley'a oraz Josefa Struger'a powstał pierwszy na świecie PLC o roboczej nazwie model 84 (rysunek 1.2). Jednak dopiero model 184 (rysunek 1.3) dopracowany przez Michaela Greenberga odniósł sukces na światowym rynku. Był to początek trzeciej rewolucji przemysłowej. W odróżnieniu od modelu 84, model 184 mógł zostać użyty w różnych dziedzinach przemysłu. W latach 80-tych wyprodukowano tysiące egzemplarzy. Model 184 był używany przez 20 lat. Potem zastąpił go model 984.



Rysunek 1.3: Model 184 [5].

1.2 Rozwój

Pierwsze sterowniki PLC miały możliwość pracy z sygnałami wejściowymi i wyjściowymi, wewnętrzną logiką styków, zegarami i licznikami. PLC nadal ewoluował dzięki kolejnym ulepszeniom typów danych i nowym funkcjom. Wraz z rozwojem funkcjonalności PLC, programowanie urządzeń i ich komunikacji również się rozwinęło. Najpierw były to urządzenia programujące wielkości walizek, ale szybko powstało oprogramowanie do programowania PLC opartym na systemie Windows. Połączenie komputera PC ze sterownikiem PLC umożliwiło nie tylko programowanie, ale także ułatwiło testowanie i rozwiązywanie problemów. Komunikacja odbywała się przy użyciu złącza szeregowego RS-232. Zastosowano różne protokoły automatyzacji komunikujące się przez RS-485, DeviceNet, Profibus i inne architektury komunikacji szeregowej. Wykorzystanie komunikacji szeregowej i różnych protokołów PLC umożliwiło również połączenie sterowników PLC z innymi sterownikami PLC, interfejsami człowiek-maszyna (HMI) oraz komunikacji z internetem.



Rysunek 1.4: Współczesny sterownik PLC marki FATEK [6].

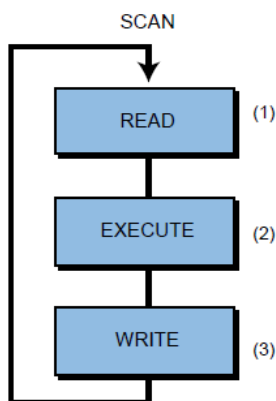
Sterowniki PLC przed użyciem trzeba było oprogramować. W tym celu powstało programowanie drabinkowe. Było oparte na bramkach logicznych oraz funkcjach arytmetycznych. Współczesne sterowniki programowalne można programować na wiele różnych sposobów, od logiki drabinowo-pochodnej po języki programowania takie jak BASIC i C. Programowanie drabinkowe jest jednak najbardziej popularnym rozwiązaniem dzięki graficznemu programowaniu i wizualnym działaniu.

1.3 Zastosowania

Sterowniki PCL stosowane są głównie na liniach produkcyjnych i montażowych oraz wszędzie tam, gdzie konieczna jest automatyzacja procesów technologicznych. Obecnie, coraz częściej sterowniki PLC używane są również niekomercyjnie w domach. Oto niektóre z przykładów zastosowania technologii PLC:

- fabryka,
- sygnalizacja świetlna na skrzyżowaniu,
- myjnia samochodowa,
- piec z boilerem,
- oświetlenie w budynku,
- pompa w fontannie,
- rolety okienne,
- winda,
- przejazd kolejowy.

1.4 Zasady działania



Rysunek 1.5: Tryb działania PLC [7].

Rysunek 1.5 ilustruje proces skanowania w sterowniku PLC. Skanowanie to proces sekwencyjnego wykonywania następujących czynności:

(1) skanowanie urządzeń wejściowych

odczyt i akceptacja danych z urządzeń wejścia,

(2) wykonanie programu

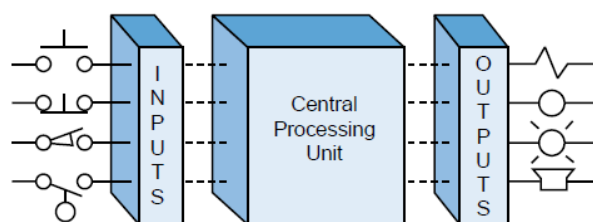
wykonanie programu sterującego przechowywanego w pamięci systemu,

(3) skanowanie urządzeń wyjściowych

zapisanie i aktualizacja urządzeń wyjścia.

1.5 Urządzenia wejścia i wyjścia

Sterownik podłączony jest do wielu różnych urządzeń. Urządzenia, z których można otrzymać dane, czyli na przykład różnego rodzaju czujniki (temperatury, ciśnienia, poziomu cieczy, procentowej zawartości gazu) zaliczają się do urządzeń wejścia, natomiast takie, które mają wykonać jakąś czynność (świecenie, poruszanie, obracanie, zamykanie, otwieranie) zaliczają się do urządzeń wyjścia.



Rysunek 1.6: Diagram blokowy PLC [7].

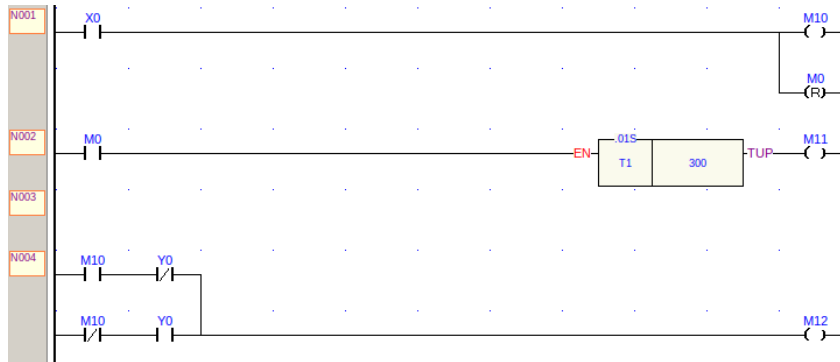
Urządzenia wejścia:

- Przyciski
- Przełączniki
- Zegary
- Sensory zbliżeniowe
- Komórki światłoczułe
- Czujniki nacisku
- Czujniki pływakowe
- Termometry
- Ciśnieniomierze

Urządzenia wyjścia:

- Silniki
- Przekładnie
- Rotory
- Głośniki
- Żarówki
- Pompy
- Wiatraki

1.6 Programowanie drabinkowe



Rysunek 1.7: Programowanie drabinkowe.

Programowanie drabinkowe jest dedykowane głównie do sterowników PLC. Składa się z bramek i działań logicznych, funkcji związanych z czasem oraz funkcji arytmetycznych. Ogólnym celem użycia programowania drabinkowego jest ustalenie jak stany na wejściach, wpływają na stany na wyjściach programu.

Rozdział 2

Rozwiązania oparte na PLC

2.1 Dom jednorodzinny lub mieszkanie

Państwo Kowalscy mieszkają w domu jednorodzinym. Oboje pracują od 8 do 16. Zimą, po powrocie do domu jest zimno, a na rozgrzanie domu potrzeba sporo czasu. Sterownik PLC może pomóc rozwiązać ten problem. Można go zaprogramować tak, by temperatura w domu była obniżana podczas nieobecności gospodarzy i podwyższana kiedy są w domu.

W rozwiązaniu praktycznym, urządzenia wejścia podłączone do PLC to kalendarz, zegar i termometr, natomiast urządzenie wyjścia to piec gazowy. Sterownik zaprogramowany jest tak by od 7 do 15 piec był wyłączony. Jeśli temperatura w domu w tym czasie spadnie poniżej 10 stopni system uruchomi piec by podgrzał dom. Po godzinie 15 system uruchamia piec, by podniósł temperaturę w domu do przyjemnych 21 stopni i utrzymywał ją do 7 rano. W programie należy uwzględnić weekendy oraz święta bo wtedy rodzina więcej czasu spędza w domu i lepiej aby piec był włączony cały czas..

Zastosowany sterownik PLC można rozbudować o nowe funkcjonalności i integrację z aplikacją internetową. Co jeśli okaże się, że pan Kowalski kończy pracę wcześniej. Jeśli PLC ma dodatkowe wejścia pozwalające na sterowanie za pomocą aplikacji internetowej, to można zdalnie za pomocą smartfona zmieniać temperaturę w domu i włączyć piec wcześniej.

2.2 Alarm antywłamaniowy

Alarm zostanie aktywowany, jeśli nieautoryzowana osoba zostanie wykryta przez czujnik okna lub czujnik ruchu.

Założenia:

1. Mamy dwa czujniki: jeden to detektor ruchu i drugi czujnik okienny.
2. Detektor ruchu jest zaprojektowany w taki sposób, że po wykryciu osoby wyjście czujnika staje się prawdą.

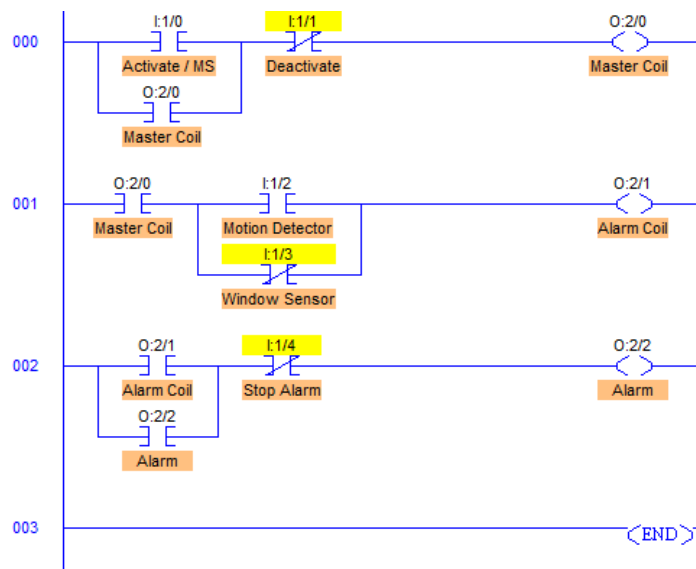
3. Czujnik okienny to nic innego jak pętla z drutu, czyli kawałek cienkiej metalowej folii, która otacza okno.
4. W czujniku okiennym płynie prąd dopóki okno nie zostanie uszkodzone.
5. Gdy system alarmowy jest aktywny i ktoś rozbija okno, prąd nie przepływa przez metalową folię powodując przerwanie obwodu.

Lista wejść:

- I:1/0 = Główny przełącznik
- I:1/1 = Wyłącznik systemu
- I:1/2 = Detektor ruchu
- I:1/3 = Czujnik okienny
- I:1/4 = Wyłącznik alarmu

Lista wyjść:

- O:2/0 = Główna cewka
- O:2/1 = Cewka alarmowa
- O:2/2 = Alarm



Rysunek 2.1: Program sterownika do obsługi alarmu antywłamaniowego [8].

Opis do rysunku 2.1:

- Linia 000: pokazuje zatrzaśnięcie Głównej cewki O:2/0 do aktywacji całego systemu bezpieczeństwa.
- Po naciśnięciu przycisku Główny przełącznik I:1/0 system bezpieczeństwa zostaje aktywowany.
- Jeśli system nie jest włączony, alarm nie wskazuje niczego, ponieważ czujniki nie będą oddziaływać na Cewkę alarmową O:2/1.
- Gdy system jest aktywny, a detektor ruchu wykryje osobę, cewka alarmowa aktywuje Alarm O:2/2, który pozostaje WŁĄCZONY, dopóki Wyłącznik alarmu I:1/4 nie zostanie naciśnięty ręcznie.
- Linia 001: Czujnik okna normalnie jest w stanie TRUE, gdy okno jest całe. Po wykryciu pęknięcia szyby przechodzi w stan FALSE włączając Cewkę alarmu na chwile, co aktywuje Alarm.
- Linia 002: Alarm dzwoni, bez względu czy detekcja przez czujniki jest chwilowa lub ciągła.

2.3 Parking samochodowy

Poprzedni program opierał się tylko na binarnym systemie bramek logicznych. Skupię się zatem bardziej na funkcjach zliczających.

Parking posiada ograniczoną liczbę miejsc na pojazdy. Duża liczba samochodów utrudnia znalezienie miejsca parkingowego. Dzięki PLC można to znacznie ułatwić.

Dodajmy do parkingu duży wyświetlacz z:

- liczbą pustych miejsc,
- dostępnymi miejscami wskazywanymi przez diody LED.

Założenia:

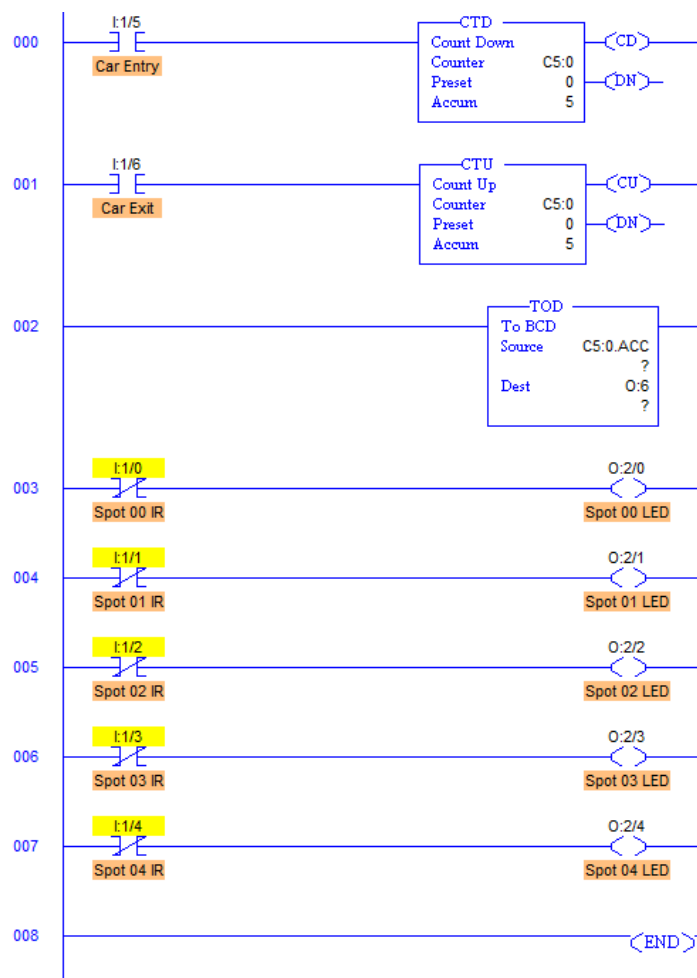
1. Licznik służy do zliczania pustych miejsc.
2. Czujniki zbliżeniowe służą do wykrywania obecności samochodu.
3. Wartość licznika jest wyświetlana na wyświetlaczu, który jest zamontowany przy wjeździe na parking.
4. Wartość licznika jest wyświetlana w postaci dziesiętnej.
5. Dodatkowo należy zwrócić uwagę na fakt, że liczba wolnych miejsc, a ilość zapalonych diod nie musi być równa, gdyż niektóre samochody będące na parkingu nie muszą być zaparkowane. Dlatego oba podzespoły muszą działać niezależnie.

Lista wejść:

- I:1/0-4 = Czujnik zbliżeniowe do wykrywania obecności samochodów na miejscach parkingowych
- I:1/5 = Czujnik przy wjeździe do parkingu
- I:1/6 = Czujnik przy wyjeździe z parkingu

Lista wyjść:

- O:2/0-4 = Diody LED sygnalizujące obecność miejsc na parkingu
- O:6 = Wyświetlacz z danymi



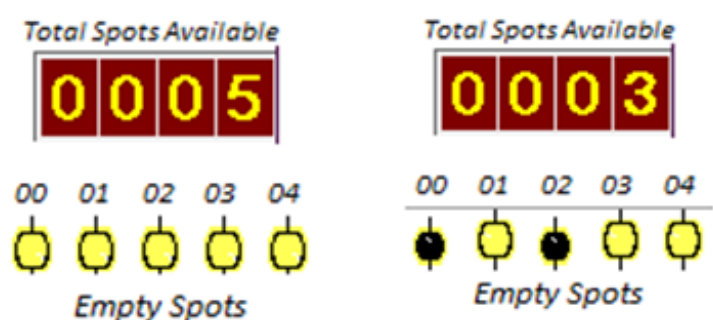
Rysunek 2.2: Program sterownika do obsługi parkingu [9].

Opis do rysunku 2.2:

- Liczniki CTU i CTD mają ten sam adres by zmieniały tą samą zmienną.

- Linia 000: Zwiększenie licznika, gdy samochód wjeżdża na parking
- Linia 001: Zmniejszenie licznika, gdy samochód wyjeżdża z parkingu
- Linia 002: Tu następuje konwersja liczby i wysłanie jej na wyświetlacz
- Linia 003-007: Jeśli czujnik nie wykrywa pojazdu na tablicy zapala się dioda

Rysunek 2.3 pokazuje przykładowy wygląd tablicy.



Rysunek 2.3: Przykładowy wygląd wyświetlacza [9].

2.4 Zbiornik z grzałką

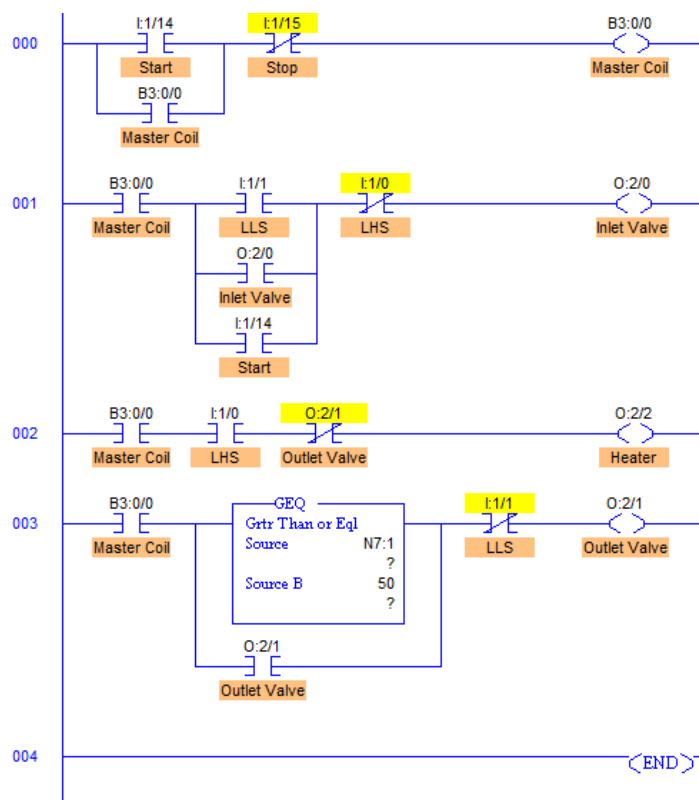
Na tym przykładzie chciałbym zwrócić uwagę na system który w przypadku niewłaściwego zaprojektowania mógłby być wadliwy lub nawet niebezpieczny. Bardzo ważną kwestią jest zaprojektowanie układu i wszystkich możliwych przypadków zanim zaczniesz programować PLC. Czy jeśli zbiornik jest w połowie pełny, to znaczy, że poziom wody jest niski czy wysoki? Co jeśli grzałka będzie podgrzewać zbiornik bez wody? W których przypadkach woda powinna wpływać, a w których wypływać ze zbiornika? Brak wnikliwej analizy może skutkować zalaniem pomieszczenia czy nawet eksplozją zbiornika.

Założenia:

1. Aby wykryć wysoki i niski poziom cieczy w zbiorniku, stosuje się dwa przełączniki poziomu, które dają wyjście w postaci cyfrowej, to znaczy, gdy odpowiednie poziomy zostaną wykryte, daje to wysoką moc wyjściową, w przeciwnym razie pozostanie niska.
2. Przełącznik niskiego poziomu jest zamontowany w dolnej części zbiornika, a przełącznik wysokiego poziomu jest zamontowany w górnej.
3. Ogrzewacz jest zainstalowany wewnątrz zbiornika, a czujnik temperatury jest używany do wykrywania temperatury cieczy w zbiorniku.

Lista wejść i wyjść:

- I:1/0 = Przełącznik wysokiego poziomu
- I:1/1 = Przełącznik niskiego poziomu
- O:2/0 = Zawór wlotowy
- O:2/1 = zawór wylotowy
- O:2/2 = Grzałka
- I:1/14 = Start
- I:1/15 = Stop
- N7:1 = rejestr z danymi temperatury



Rysunek 2.4: Program sterownika do obsługi zbiornika z grzałką [10].

Opis do rysunku 2.4:

- Linia 000: służy do zatrzymywania cewki i głównych przycisków start-stop.

- Linia 001: otwiera zawór wlotowy, gdy wykryty zostanie niski poziom cieczy w zbiorniku.
- Linia 002: dotyczy grzałki jest uruchomiona, gdy wykryty zostanie wysoki poziom. Odbywa się to do momentu, gdy temperatura osiągnie 50°C, grzałka zostaje wyłączona, a zawór wyjściowy otwiera się.
- Linia 003: ma sterować zaworem wylotowym. Gdy temperatura jest wyższa niż 50°C, zawór wylotowy zaczyna opróżniać ciecz. Po wykryciu niskiego poziomu cieczy w zbiorniku, następuje zamknięcie zaworu wylotowego.
- Gdy zbiornik jest tylko częściowo napełniony i nie wykryto ani niskiego, ani wysokiego poziomu, grzałka zostaje wyłączona, ale poziom cieczy się nie zmienia. Aby wyeliminować ten błąd, I:1/14 (Start) jest połączony równoległe z przełącznikiem niskiego poziomu. Sprawdza to, czy wysoki poziom jest wykryty, czy nie. Jeśli wysoki poziom nie zostanie wykryty, otwiera zawór wlotowy do momentu wykrycia i następnie rozpoczęcia ogrzewania.

Rozdział 3

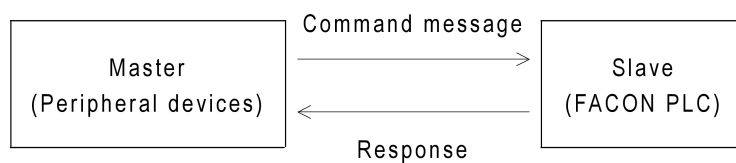
Protokół komunikacyjny FACON

Protokół komunikacyjny FACON ma zastosowanie do wszystkich portów komunikacyjnych urządzeń Fatek PLC działających w standardowym trybie. Urządzenie peryferyjne, które komunikuje się ze sterownikiem PLC, powinno być wyposażone w odpowiedni interfejs sprzętowy skonfigurowany zgodnie z wymaganymi parametrami, a przekazywane wiadomości powinny być formatowane z zastosowaniem opisywanego tutaj protokołu.

Niniejszy rozdział opracowano na podstawie [3] i stamtąd zaczerpnięto wszystkie grafiki zamieszczone w tym rozdziale.

3.1 Urządzenia nadrzędne i podrzędne

Podczas komunikacji sterownik PLC nazywany jest *urządzeniem podrzędnym*, natomiast urządzenie peryferyjne nazywane jest *urządzeniem nadrzędnym* (por. rys. 3.1). To urządzenie peryferyjne rozpoczyna komunikację i aktywnie wysyła komunikaty do sterownika PLC natomiast sterownik PLC wyłącznie pasywnie odpowiada na odebrane komunikaty.



Rysunek 3.1: Urządzenia nadrzędne i podrzędne.

Połączone ze sobą urządzenia nadrzędne i podrzędne tworzą sieć komunikacyjną. W takiej sieci występuje dokładnie jedno urządzenie nadrzędne oraz maksymalnie do 255 urządzeń podrzędnych, czyli Fatek PLC. Każdemu

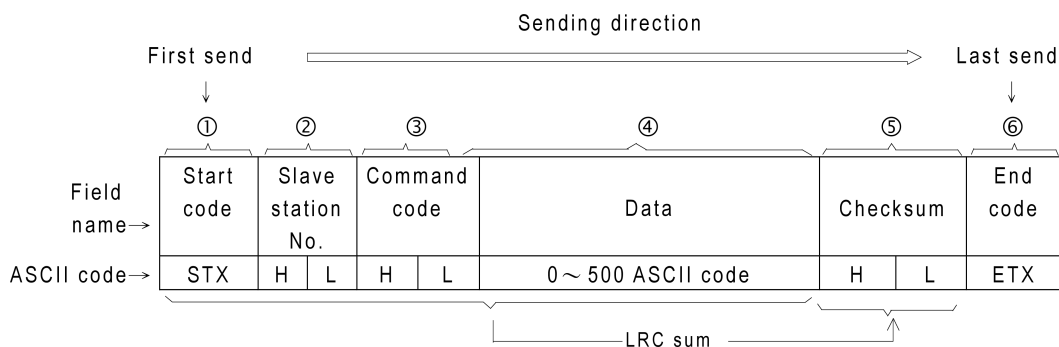
urządzeniu podrzędnemu w sieci przypisany jest jego unikalny identyfikator – *numer stacji* z zakresu od 0x01 do 0xFE.

Fabrycznie wszystkim urządzeniom Fatek PLC nadawany jest numer stacji o wartości 1. W protokole FACON nie przewidziano możliwości zmiany numeru stacji. Aby zmienić numer stacji należy użyć programu konfiguracyjnego lub WinProLadder.

Wszystkie wartości przesyłane są w protokole FACON jako ich szesnastkowa reprezentacja w kodzie ASCII. Na przykład numer stacji o dziesiętnej wartości 26, czyli szesnastkowo 0x1A, przesyłany jest jako dwa znaki 1A, natomiast wartość rejestru, dziesiętnie 171, czyli szesnastkowo 0xAB, przesyłana jest jako dwa znaki AB. Praktycznie całość transmisji w protokole FACON jest czytelna w surowej postaci bez potrzeby dekodowania przesyłanych danych. Czyni to protokół FACON przejrzystym i ułatwia diagnostykę, za pewną cenę jednak jaką jest narzut w ilości przesyłanych danych. Zgodnie z przyjętą konwencją bowiem, aby przesłać 1 bit zużywa się aż 1 bajt (1 znak ASCII), czyli 8 bitów, natomiast do przesłania 16-bitów zużywa się 4 bajty (4 znaki ASCII), czyli 32 bity.

3.2 Format komunikatu

Wszystkie komunikaty, zarówno *zapytania* wysyłane z urządzenia nadrzędnego jak i *odpowiedzi* wysyłane przez urządzenie podrzędne mają ten sam format (por. rys. 3.2).



Rysunek 3.2: Format komunikatu w protokole FACON.

Komunikat w protokole FACON zbudowany jest z sześciu następujących pól:

Znak początku Jeden znak (jeden bajt) STX (Start Of Text) w kodzie ASCII o wartości 0x02. Ten znak rozpoczyna zarówno zapytanie jak i odpowiedź. Gdy urządzenie oczekuje na komunikację i odbierze taki znak to oznacza początek komunikatu.

Numer stacji podrzędnej Dwa znaki (dwa bajty) reprezentujące numer stacji podrzędnej w kodzie szesnastkowym. Wartość 00 oznacza, że komunikat adresowany jest do wszystkich stacji podrzędnych. Gdy urządzenie nadrzędne chce wysłać żądanie zaadresowane do konkretnej stacji podrzędnej umieszcza jej numer w tym właśnie polu komunikatu. Stacja podrzędna zawsze umieszcza swój numer w tym polu w odsyłanej odpowiedzi.

Kod polecenia Dwa znaki (dwa bajty) reprezentujące kod polecenia w kodzie szesnastkowym. To jest kod polecenia jakie urządzenie nadrzędne chce aby zostało wykonane przez urządzenie podrzędne, czyli na przykład: zapis lub odczyt danych dyskretnych itp. Kod polecenia w odpowiedzi urządzenia podrzędnego jest taki sam jak kod polecenia w żądaniu jakie otrzymało.

Dane Od 0 do 500 znaków (bajtów), choć praktyka pokazuje, że może być nieco więcej. To pole zwykle zawiera adresy rejestrów do odczytu i ewentualnie ich wartości do zapisu w Fatek PLC. Pierwszy znak odpowiedzi w tym polu to kod błędu. W przypadku braku błędu przesyłany jest znak 0 o wartości 0x30 w tabeli ASCII. Po nim następują właściwie przesyłane w odpowiedzi urządzenia podrzędnego dane. Jeśli wystąpi błąd, przesyłany jest znak reprezentujący szesnastkową wartość jego kodu. Wówczas jest to jedyny znak w tym polu i nie są przesyłane żadne dodatkowe dane.

Suma kontrolna Dwa znaki (dwa bajty) reprezentujące wartość sumy kontrolnej w kodzie szesnastkowym. Suma kontrolna obliczana jest przy pomocy zmodyfikowanego algorytmu LRC. Dodawane są do siebie kolejno wartości znaków od znaku początku po ostatni znak w polu danych. Przy każdym sumowaniu obliczane jest modulo 256.

Urządzenie odbierające komunikat oblicza sumę kontrolną stosując ten sam algorytm i porównuje wynik z przesłaną sumą kontrolną w komunikacie. Jeśli wartości są równe to mamy gwarancję, że odebrany komunikat jest taki sam jak został nadany. W przeciwnym razie podczas transmisji wystąpiło przekłamanie.

Znak końca Jeden znak (jeden bajt) ETX (End Of Text) w kodzie ASCII o wartości 0x03. Ten znak kończy zarówno żądanie jak i odpowiedź. Odczytanie znaku końca przez urządzenie odbierające dane oznacza koniec komunikatu. Następuję zakończenie odczytu danych i urządzenie powinno zacząć przetwarzać komunikat.

3.3 Kody błędów

W sytuacji, gdy format komunikatu jest niepoprawny, nieprawidłowy jest kod polecenia, przekroczone są adresy rejestrów lub ich wartości w przypadku zapisu, albo wystąpiła usterka sprzętowa, polecenie przesłane w żądaniu z urządzenia nadrzędnego nie może zostać poprawnie wykonane przez urządzenie podrzędne. Wówczas urządzenie podrzędne zwraca odpowiedni kod błędu w odpowiedzi do urządzenia nadrzędnego. Niezależnie od kodu polecenia oraz danych w żądaniu, format komunikatu odpowiedzi zawierającej kod błędu, jest taki sam jak pozostałych komunikatów. Do urządzenia nadrzędnego zostanie przesłana odpowiedź zawierająca znak początku, numer stacji, kod polecenia, w polu danych kod błędu informujący urządzenie nadrzędne co się stało, wyliczoną sumę kontrolną oraz znak końca. Wyjątkiem jest polecenie o kodzie 0x4E, czyli testowa pętla zwrotna. Z założenia odpowiedź jest tutaj identyczna jak żądanie, dlatego nie ma kodu ewentualnego błędu.

Kod błędu	Opis
0x00	Brak błędu
0x02	Niepoprawna wartość
0x03	Zapis zabroniony
0x04	Niepoprawna składnia polecenia, kod polecenia lub polecenie nie może być wykonane
0x05	Błędna suma kontrolna programu
0x06	Nie zgodne identyfikatory sterownika PLC oraz programu
0x07	Błąd syntaktyczny
0x09	Niewspierane instrukcje w kodzie programu
0x0A	Niepoprawny adres odwołania

Tabela 3.1: Kody błędów odpowiedzi w protokole FACON.

3.4 Polecenia

W tym rozdziale zostaną przedstawione poszczególne polecenia i ich odpowiedzi występujące w protokole FACON. Tutaj opisane zostaną jedynie poprawne odpowiedzi, natomiast kody błędów opisane są w rozdziale 3.3.

3.4.1 Zestawienie dostępnych rejestrów

Głównym celem komunikacji w protokole FACON jest odczyt i zapis wartości oraz statusów sterownika PLC. W tabeli 3.2 zebrane zostały dostępne rejestry do odczytu i zapisu.

Symbol	Nazwa	Adresy rejestrów		
		1-bitowych	16-bitowych	32-bitowych
X	Styk wejściowy	X0000 - X9999	WX000 - WX9984	DWX0000 - DWX9968
Y	Przełącznik wyjściowy	Y0000 - Y9999	WY000 - WY9984	DWY0000 - DWY9968
M	Przełącznik wewnętrzny	M0000 - M9999	WM000 - WM9984	DWM0000 - DWM9968
S	Przełącznik krokowy	S0000 - S9999	WS000 - WS9984	DWS0000 - DWS9968
T	Zegar	T0000 - T9999	WT000 - WT9984	DWT0000 - DWT9968
C	Licznik	C0000 - C9999	WC000 - WC9984	DWC0000 - DWC9968
T	Zegar	-	RT0000 - RT9999	DRT0000 - DRT9998
C	Licznik	-	RC0000 - RC9999	DRC0000 - DRC9998
R	Rejestr danych	-	R00000 - R65535	DR00000 - DR65534
D	Rejestr danych	-	D00000 - D65535	DD00000 - DD65534

Tabela 3.2: Zestawienie dostępnych rejestrów w sterowniku PLC.

Dyskretny rejestr 16-bitowy oraz 32-bitowy (o symbolu X, Y, M lub S) utworzony jest z, odpowiednio, 16 lub 32 kolejnych rejestrów 1-bitowych. Adres takiego rejestru powinien być wielokrotnością liczby 8.

Adresy rejestrów 1-bitowych przekazywane są jako 5 znaków (1 znak na symbol plus 4 znaki na adres), adresy rejestrów 16-bitowych przekazywane są jako 6 znaków (1 lub 2 znaki na symbol i odpowiednio 5 lub 4 znaki na adres), natomiast adresy rejestrów 32-bitowych przekazywane są jako 7 znaków (2 lub 3 znaki na symbol i odpowiednio 5 lub 4 znaków na adres). Wartość numeryczna adresu jest zawsze dopełniana znakiem 0 z lewej strony do żądanej ilości znaków.

Podane w tabeli 3.2 zakresy adresów są największe z możliwych dla wszystkich modeli sterowników Fatek PLC. Dla konkretnego modelu sterownika zakresy te mogą być mniejsze. Przekroczenie dopuszczalnego zakresu sterownik będzie sygnalizował w odpowiedzi kodem błędu 0x0A.

3.4.2 Zestawienie dostępnych poleceń

W protokole FACON, *znak* oznacza wartość 8 bitową, czyli 1 bajt, liczbę całkowitą z zakresu od 0 do 255 lub od 0x00 do 0xFF heksadecymalnie. Wartość taką określa się znakiem ponieważ w komunikacie przekazywana jest jako odpowiadający tej wartości znak ASCII. Na przykład wartość 65, albo 0x41 szesnastkowo, przesyłana jest jako znak A.

Wartość całkowita z zakresu od 0 do 65535, czyli od 0x0000 do 0xFFFF heksadecymalnie, określana jest jako *słowo* i przesyłana jest jako 4 znaki

ASCII. Te 4 znaki kodują daną wartość w systemie szesnastkowym. To znaczy, na przykład, liczba 431 przekazywana jest jako 01AF bo 431 szesnastkowo to właśnie 0x1AF.

Kod	Opis polecenia	Rozmiar danych
0x40	Odczyt uproszczonego statusu sterownika PLC	—
0x41	Uruchomienie i zatrzymanie sterownika PLC	—
0x42	Zapis statusu rejestru 1-bitowego	1 znak
0x43	Odczyt statusu kolejnych rejestrów 1-bitowych	1–256 znaków
0x44	Odczyt wartości kolejnych rejestrów 1-bitowych	1–256 znaków
0x45	Zapis wartości kolejnych rejestrów 1-bitowych	1–256 znaków
0x46	Odczyt wartości kolejnych rejestrów 16 lub 32-bitowych	1–64 słów
0x47	Zapis wartości kolejnych rejestrów 16 lub 32-bitowych	1–64 słów
0x48	Odczyt wartości dowolnych rejestrów	1–64 słów
0x49	Zapis wartości dowolnychh rejestrów	1–32 słów
0x4E	Testowa pętla zwrotna	0–256 znaków
0x4F	Odczyt programu ze sterownika PLC	64 słowa
0x50	Zapis programu do sterownika PLC	64 słowa
0x53	Odczyt szczegółowego statusu sterownika PLC	—

Tabela 3.3: Zestawienie dostępnych poleceń w protokole FACON.

Podczas przesyłania wartości rejestru 1-bitowego, w polu danych komunikatu przekazywany jest jeden znak: 0 oznacza wyłączony, 1 oznacza załączony. Wartość rejestru 16-bitowego to słowo i przekazywana jest jako 4 znaki reprezentujące tę wartość heksadecymalnie.

Wartość rejestru 32-bitowego to dwa kolejno po sobie następujące słowa. Do przesłania wartości takiego rejestru używa się 8 znaków reprezentujących jego wartość. Rejestry 32-bitowe, podczas przesyłania ich wartości, powinny być traktowane jako 2 rejestry 16-bitowe. Na przykład, w poleceniach 0x46 i 0x47 można przekazać maksymalnie 64 słowa, czyli wartości 64 rejestrów 16-bitowych albo wartości 32 rejestrów 32-bitowych. Polecenia te pozwalają przysyłać albo tylko rejestry 16-bitowe, albo tylko rejestry 32-bitowe.

Przy pomocy polecenia 0x48 możemy odczytać maksymalnie 64 słowa, czyli 256 znaków, natomiast przy pomocy polecenia 0x49 możemy zapisać maksymalnie 32 słowa, czyli 128 znaków. Przy pomocy tych poleceń można przekazywać wartości rejestrów dowolnego typu. Należy pamiętać jednak o tym, że jedna wartość 32-bitowa zajmuje tyle miejsca co dwie wartości 16-bitowe, albo 4 wartości 1-bitowe. Odpowiednio zajmują one 8, 4 lub 1 znak.

Aby przesłać wartości kolejnych rejestrów wystarczy wskazać adres pierwszego z nich oraz ich ilość. Nie podaje się adresów poszczególnych rejestrów. W jednym poleceniu wszystkie rejestry muszą być tego samego typu.

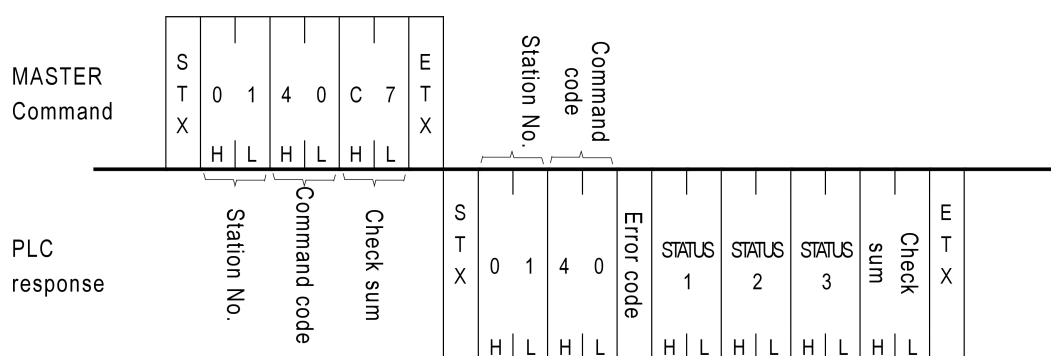
Jeśli w jednym poleceniu przesyłane są wartości rejestrów różnych typów, to należy podać adresy każdego z rejestrów. Wówczas kolejność w jakiej dane

są zapisywane do lub odczytywane z rejestrów może być losowa.

Przesyłając program z lub do sterownika PLC w ramach jednego polecenia możemy przysłać maksymalnie 64 słowa, czyli 256 znaków, tego programu. Gdy program jest większy należy dane polecenie wykonać odpowiednią ilość razy.

3.4.3 Polecenie 0x40 – odczyt uproszczonego statusu sterownika

Polecenie o kodzie 0x40 używane jest w celu sprawdzenia aktualnego stanu sterownika PLC. Schemat żądania i odpowiedzi przedstawiono na rys. 3.3.



Rysunek 3.3: Schemat komunikatu polecenia 0x40.

W żądaniu tego polecenie nie przekazujemy żadnych danych. W odpowiedzi, jako dane, sterownik zwraca 6 znaków – po dwa znaki na jedną wartość statusu. Tak więc w odpowiedzi dostajemy 3 składowe: STATUS₁, STATUS₂, STATUS₃. Znaczenie poszczególnych bitów STATUS₁ przedstawione są w tabeli 3.4.

Bit	Znaczenie	Wartość 0	Wartość 1
B0	stan sterownika	zatrzymany	działa
B1	status baterii	normalny poziom	niski poziom
B2	suma kontrolna programu	OK	błąd
B3	ROM PACK	nieużywany	używany
B4	status WDT	OK	błąd
B5	status ID	nieustawiony	ustawiony
B6	status awaryjnego zatrzymania	OK	zatrzymanie awaryjne
B7	zarezerwowany	–	–

Tabela 3.4: Znaczenie bitów statusu sterownika PLC.

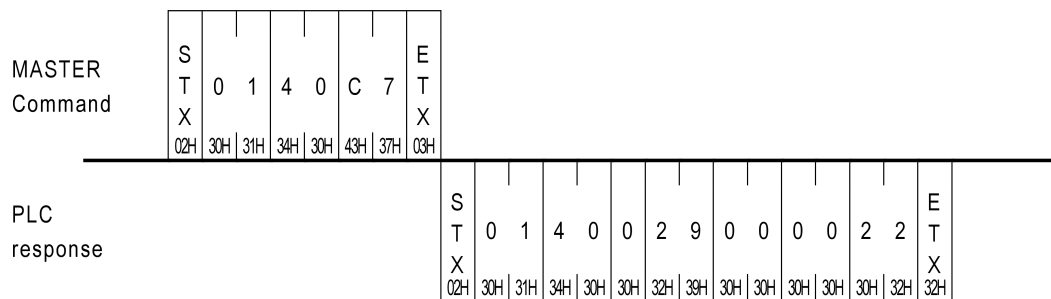
Druga składowa STATUS₂, gdy program w sterowniku PLC jest programem krokowym, może przyjmować jedną z wartości zamieszczonych w tabeli 3.5

Wartość	Znaczenie
0x00	wartość zarezerwowana
0x53	program krokowy FBE 8k
0x54	program krokowy FBE 13k
0x55	program krokowy FBN 8k
0x56	program krokowy FBN 13k
0xFF	program krokowy FB 8k

Tabela 3.5: Znaczenie drugiej składowej STATUS₂ uproszczonego statusu sterownika PLC.

Trzecia składowa STATUS₃ jest zarezerwowana do przyszłych zastosowań.

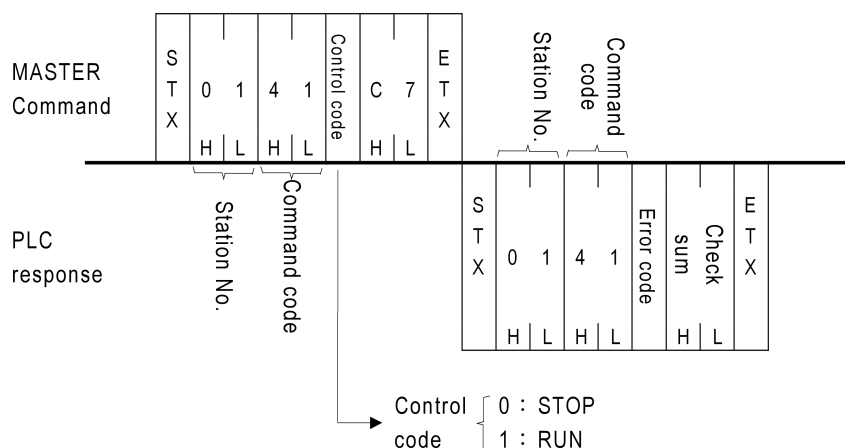
Przykład Jeśli sterownik wyposażony jest w ROM PACK, jego ID oraz ID w ROM PACK są ustawione oraz sterownik pracuje normalnie to w odpowiedzi zwróci STATUS₁ o wartości 0x29. Bity B0, B3 oraz B5 ustawione są na 1 pozostałe na 0. Daje to wartość 0x29 (por. rys. 3.4).



Rysunek 3.4: Przykład komunikatu polecenia 0x40.

3.4.4 Polecenie 0x41 – uruchomienie i zatrzymanie sterownika

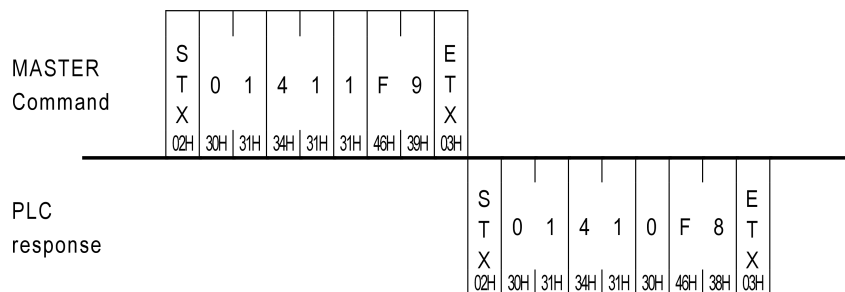
Polecenie o kodzie 0x41 używane jest aby uruchomić lub zatrzymać sterownik. Schemat żądania i odpowiedzi przedstawiono na rys. 3.5.



Rysunek 3.5: Schemat komunikatu polecenia 0x41.

W żądaniu tego polecenie przekazywany jest 1 znak o możliwych dwóch wartościach: 0 – zatrzymaj sterownik, 1 – uruchom sterownik. W odpowiedzi sterownik nie zwraca żadnych danych.

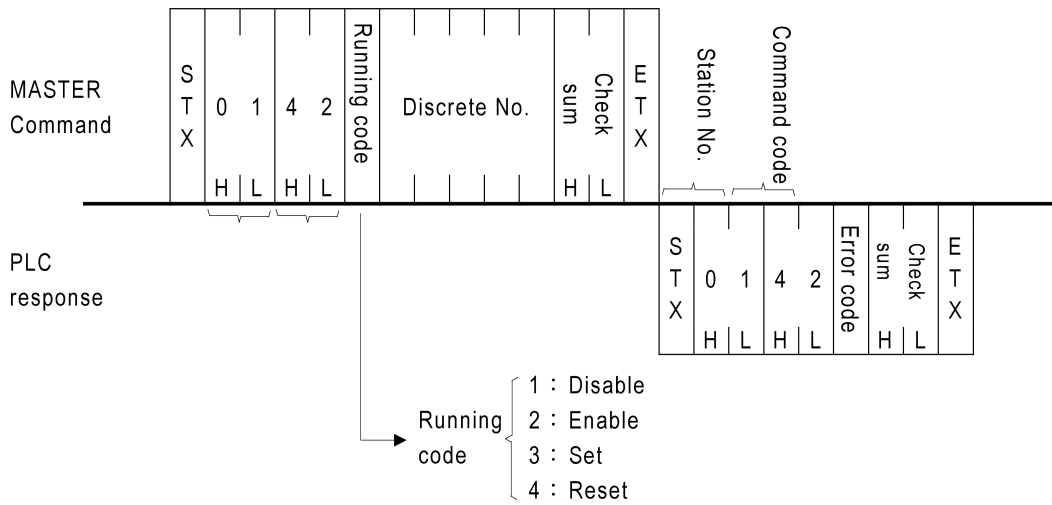
Przykład Aby uruchomić sterownik wysyłamy w polu danych znak 1. W odpowiedzi na to żądanie nie ma żadnych danych.



Rysunek 3.6: Przykład komunikatu polecenia 0x41.

3.4.5 Polecenie 0x42 – zapis statusu rejestru 1-bitowego

Polecenie o kodzie 0x42 używane jest do zapisania statusu pojedynczego rejestru 1-bitowego. Schemat żądania i odpowiedzi przedstawiono na rys. 3.7.



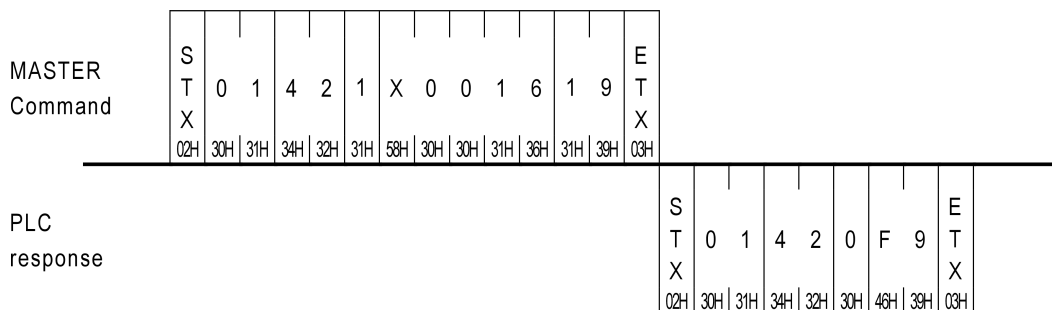
Rysunek 3.7: Schemat komunikatu polecenia 0x42.

W żądaniu tego polecenia przekazywany jest docelowy status rejestru 1-bitowego jako 1 znak o możliwych czterech następujących wartościach:

- 1: wyłącza rejestr z użycia w programie PLC,
- 2: załącza rejestr do użycia w programie PLC,
- 3: ustawia wartość rejestru na 1,
- 4: ustawia wartość rejestru na 0.

Po nim wysyłany jest symbol rejestru i jego adres jako 4 znaki, razem 5 znaków. W odpowiedzi na to polecenie sterownik nie zwraca żadnych danych.

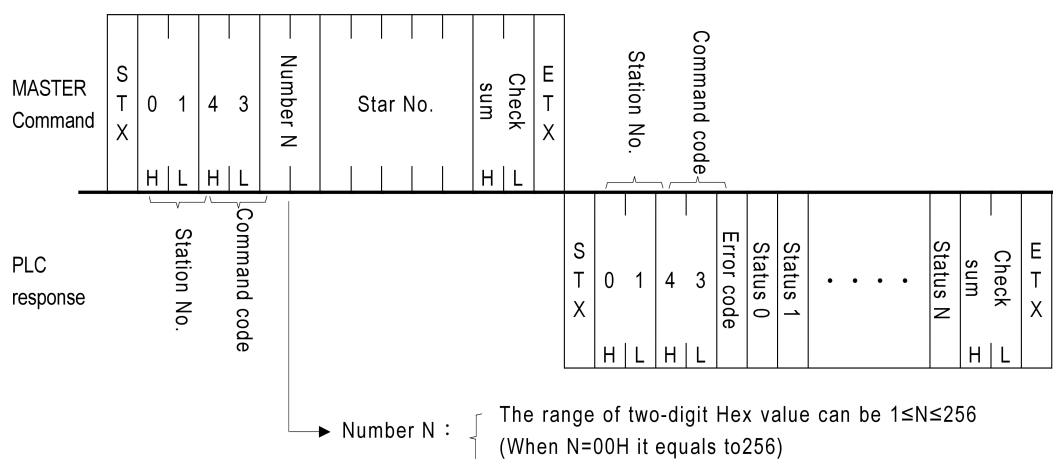
Przykład Aby wyłączyć rejestr 1-bitowy X16 wysyłany jest znak 1, czyli heksadecymalnie 0x31. Kolejny znak w żądaniu to symbol X rejestru, po którym podany jest adres rejestru jako 4 znaki ASCII, czyli napis 0016.



Rysunek 3.8: Przykład komunikatu polecenia 0x42.

3.4.6 Polecenie 0x43 – odczyt statusu kolejnych rejestrów 1-bitowych

Polecenie o kodzie 0x43 używane jest do odczytania statusu kolejnych rejestrów 1-bitowych. Schemat żądania i odpowiedzi przedstawiono na rys. 3.9.



Rysunek 3.9: Schemat komunikatu polecenia 0x43.

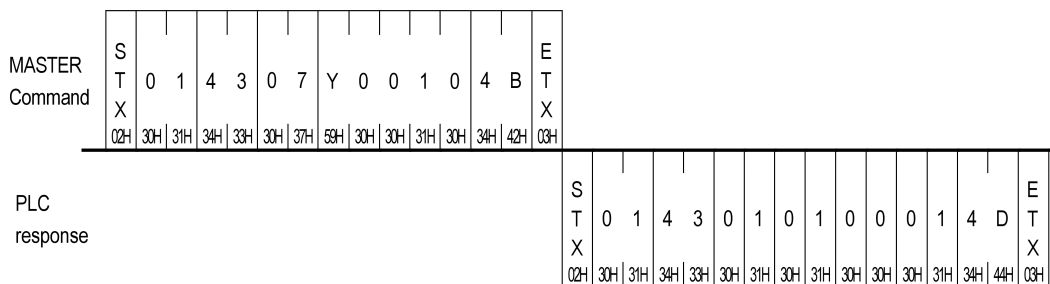
Status rejestru 1-bitowego możemy modyfikować za pomocą polecenia o kodzie 0x42, które zostało opisane w poprzedniej sekcji. Status rejestru w tym poleceniu rozumiany jest nieco inaczej i może przyjmować tylko dwie wartości:

- 0: rejestr jest załączony do użycia w programie PLC,
- 1: rejestr jest wyłączony z użycia w programie PLC.

W poleceniu tym, jako dane przesyłamy najpierw ilość rejestrów, których status nas interesuje. Liczbę tę podajemy jako 2 znaki reprezentujące jej heksadecymalną wartość. Przesłanie napisu 12 oznacza 18 rejestrów. Wartość 00 odpowiada 256 rejestrów i tyle maksymalnie statusów rejestrów możemy odczytać w jednym poleceniu. Kolejnych 5 znaków to: 1 znak na symbol i 4 znaki na adres rejestru początkowego.

W odpowiedzi polecenie zwraca ciąg znaków 0, 1 będących statusami kolejnych rejestrów począwszy od rejestru wskazanego w żądaniu. Ilość znaków w odpowiedzi powinna być taka jak ilość wskazana w żądaniu.

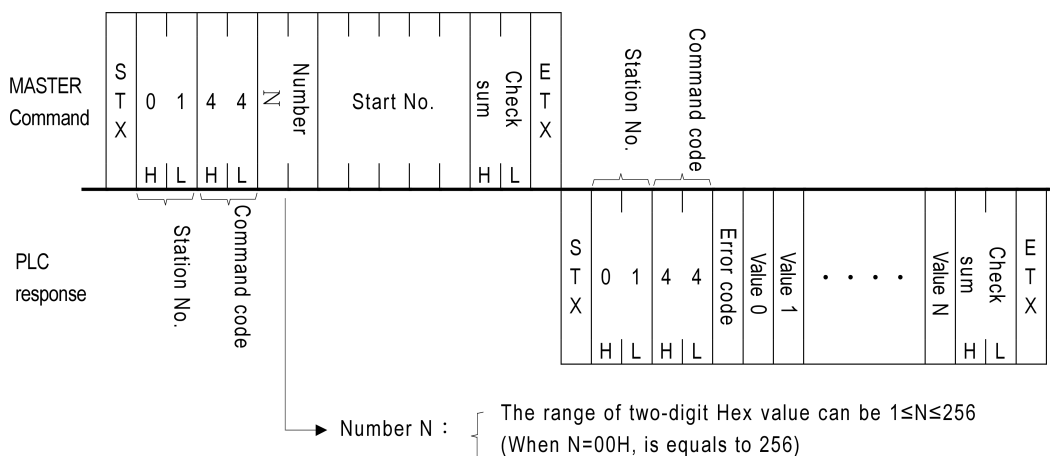
Przykład Jeśli chcemy odczytać statusy rejestrów począwszy od Y10 do Y16, to w polu danych komunikatu polecenia 0x43 umieszczamy 07, bo interesuje nas 7 kolejnych rejestrów. Następnie umieszczamy tak symbol i adres pierwszego rejestru, czyli Y0010 (por. rys. 3.10). W odpowiedzi dostajemy ciąg 1010001, co oznacza, że rejestry Y10, Y12 i Y16 są wyłączone, a pozostałe załączone.



Rysunek 3.10: Przykład komunikatu polecenia 0x43.

3.4.7 Polecenie 0x44 – odczyt wartości kolejnych rejestrów 1-bitowych

Polecenie o kodzie 0x44 stosujemy aby odczytać wartości kolejnych rejestrów 1-bitowych. Schemat żądania i odpowiedzi przedstawiono na rys. 3.11.

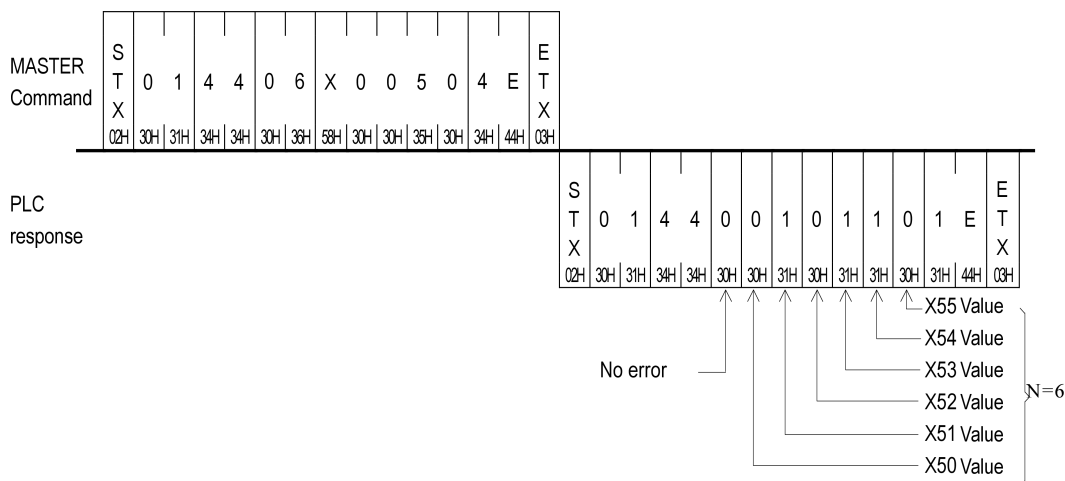


Rysunek 3.11: Schemat komunikatu polecenia 0x44.

Tego polecenia używa się bardzo podobnie jak polecenia 0x43 do odczytu statusów. W polu danych umieszczamy najpierw ilość rejestrów, których wartości nas interesują. Liczbę tę podajemy jako 2 znaki reprezentujące jej heksadecymalną wartość. Przesłanie napisu 21 oznacza 33 rejestry. Wartość 00 odpowiada 256 rejestrom i tyle maksymalnie rejestrów możemy odczytać w jednym poleceniu. Kolejnych 5 znaków to: 1 znak na symbol i 4 znaki na adres rejestru początkowego.

W odpowiedzi dostajemy ciąg znaków 0, 1 będących wartościami kolejnych rejestrów począwszy od rejestru wskazanego w żądaniu. Ilość znaków w odpowiedzi powinna być taka jak ilość wskazana w żądaniu.

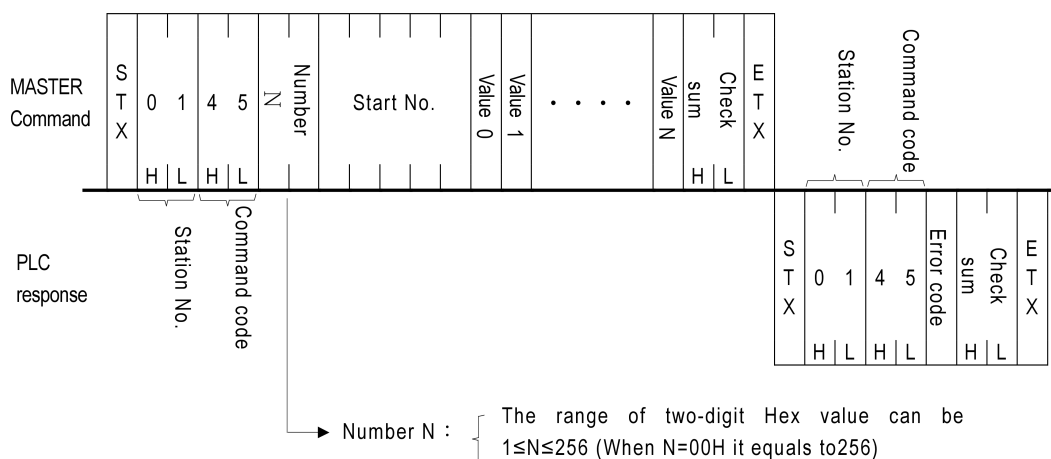
Przykład Powiedzmy, że interesują nas wartości 6 kolejnych rejestrów od X50 do X55 włącznie. W polu danych żądania umieszczamy napis 06, a następnie napis X0050, co zostało pokazane na rysunku 3.12. Sterownik zwraca wartości interesujących nas rejestrów jako ciąg znaków 010110. Zatem rejestry X50, X52 i X55 mają wartość 0, natomiast rejestry X51, X53 i X54 mają wartość 1.



Rysunek 3.12: Przykład komunikatu polecenia 0x44.

3.4.8 Polecenie 0x45 – zapis wartości kolejnych rejestrów 1-bitowych

Polecenie o kodzie 0x45 stosujemy aby zapisać wartości kolejnych rejestrów 1-bitowych. Schemat żądania i odpowiedzi przedstawiono na rys. 3.13.

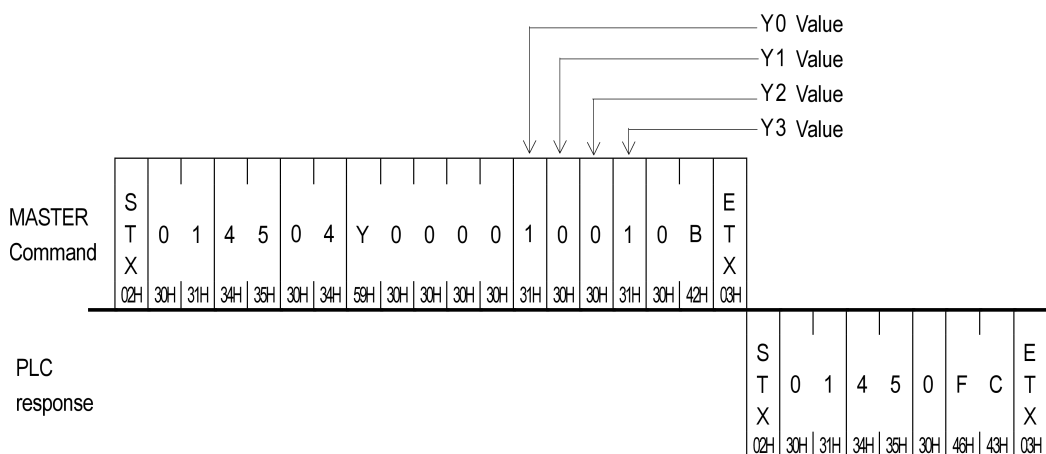


Rysunek 3.13: Schemat komunikatu polecenia 0x45.

Aby zapisać wartości kolejnych rejestrów 1-bitowych w polu danych komunikatu umieszczamy najpierw ilość tych rejestrów. Ilość podajemy jako 2 znaki reprezentujące jej heksadecymalną wartość, podobnie jak w poleceniach 0x43 oraz 0x44. Przesłanie napisu 1A oznacza 26 rejestrów. Wartość 00 odpowiada 256 rejestrów i tyle maksymalnie rejestrów możemy zapisać w jednym poleceniu. Kolejnych 5 znaków to: 1 znak na symbol i 4 znaki na adres rejestru początkowego.

To polecenie nie zwraca żadnych danych.

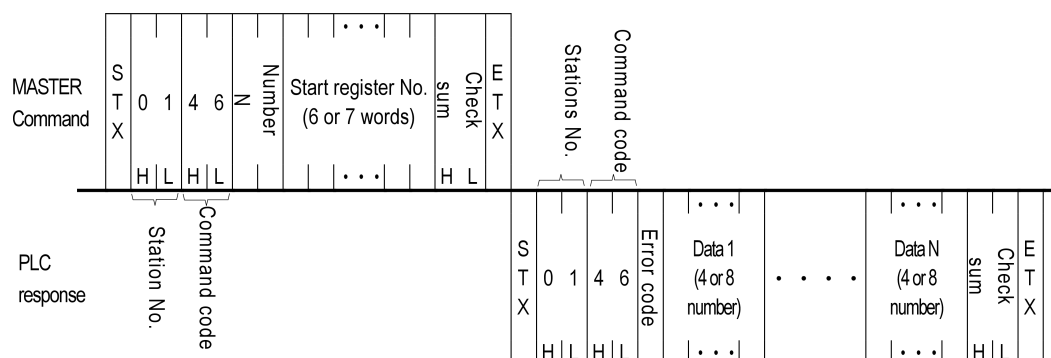
Przykład Załóżmy, że zapisujemy wartość 1 do rejestrów Y0, Y3 oraz wartość 0 do rejestrów Y1, Y2. Są to 4 kolejne rejestry, więc stosujemy polecenie 0x45. W danych umieszczamy napis 04 wskazujący ilość rejestrów do zapisu, dalej symbol i adres rejestru początkowego, czyli Y0000, a następnie musimy podać ciąg 4 znaków odpowiadających poszczególnym rejestrów, czyli 1001, tak jak na rysunku 3.14.



Rysunek 3.14: Przykład komunikatu polecenia 0x45.

3.4.9 Polecenie 0x46 – odczyt wartości kolejnych rejestrów 16 lub 32-bitowych

Polecenie o kodzie 0x46 używane jest do odczytywania wartości kolejnych rejestrów 16-bitowych lub 32-bitowych. Schemat żądania i odpowiedzi przedstawiono na rys. 3.15.

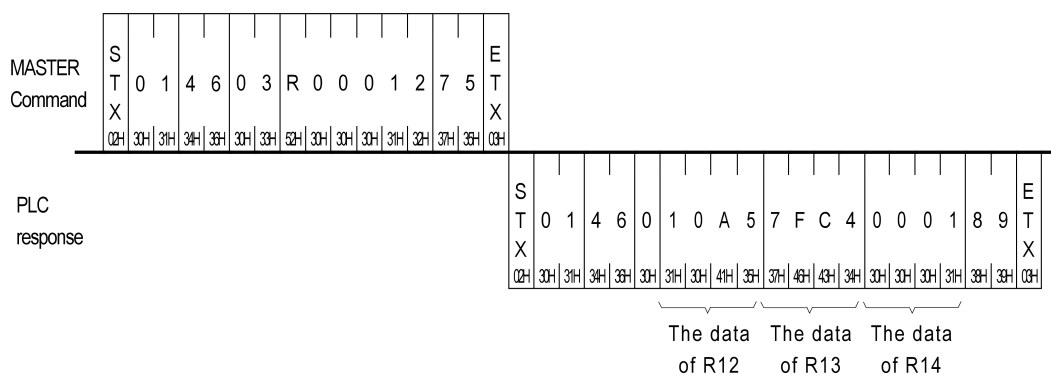


Rysunek 3.15: Schemat komunikatu polecenia 0x46.

W jednym poleceniu możemy odczytać maksymalnie 64 rejestry 16-bitowe albo 32 rejestry 32-bitowe. Ponieważ odczytujemy wartości kolejno po sobie następujących rejestrów, w polu danych polecenia najpierw umieszczamy ilość rejestrów do odczytania. Podobnie jak w poleceniach 0x43 – 0x45 przesyłamy 2 znaki reprezentujące tę ilość szesnastkowo, to znaczy napis 1B oznacza 27 rejestrów. Dalej, w polu danych umieszczamy pełny, alfanumeryczny adres rejestru, którego format zależy od typu i symbolu. Zgodnie z tabelą 3.2 adres rejestru 16-bitowego to zawsze 6 znaków, a rejestru 32-bitowego to 7 znaków. Tak więc jako dane przesyłamy 8 lub 9 znaków w zależności od tego, czy odczytujemy rejestry 16-bitowe, czy 32-bitowe.

Zwracana wartość rejestru 16-bitowego składa się z 4 znaków reprezentujących wartość tego rejestru w kodzie szesnastkowym. Natomiast wartość rejestru 32-bitowego składa się z 8 znaków. Tak więc, w odpowiedzi dostajemy ciąg znaków po 4 znaki na każdy rejestr 16-bitowy, albo po 8 znaków na każdy rejestrów 32-bitowy.

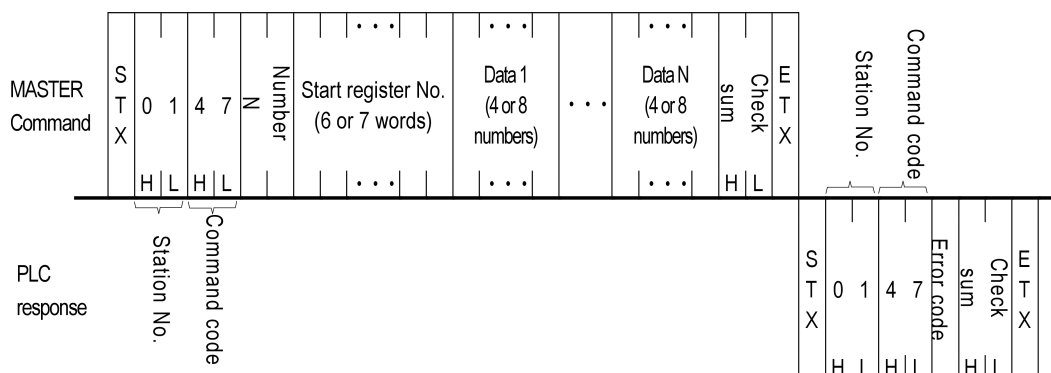
Przykład Odczytujemy 3 kolejne rejestry 16-bitowe począwszy od R12, czyli R12, R13 i R14. W polu danych umieszczamy ciąg 03, a po nim R00012. W odpowiedzi dostajemy razem 12 znaków, po 4 znaki kolejno na każdy z rejestrów. W przykładzie na rysunku 3.16 są to 10A5 dla rejestru R12, 7FC4 dla rejestru R13 oraz 0001 dla rejestru R14.



Rysunek 3.16: Przykład komunikatu polecenia 0x46.

3.4.10 Polecenie 0x47 – zapis wartości kolejnych rejestrów 16 lub 32-bitowych

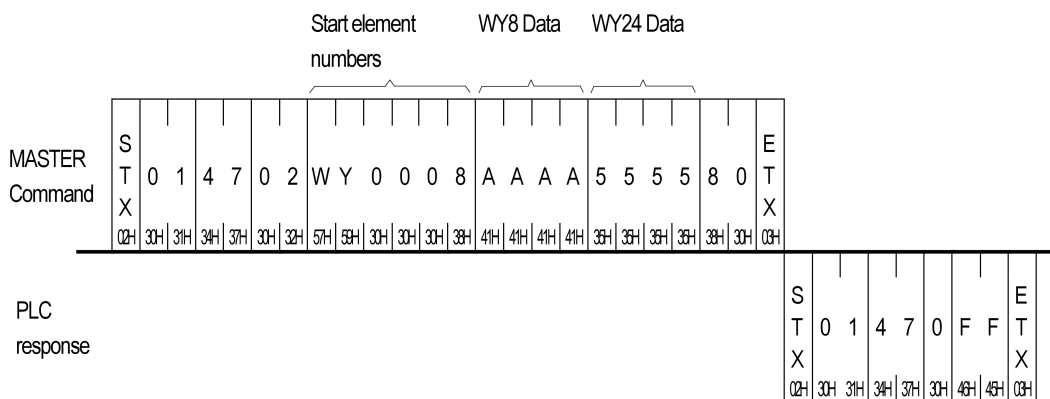
Polecenie o kodzie 0x47 używane jest do zapisu wartości kolejnych rejestrów 16-bitowych lub 32-bitowych. Schemat żądania i odpowiedzi przedstawiono na rys. 3.17.



Rysunek 3.17: Schemat komunikatu polecenia 0x47.

Podobnie jak w poleceniu 0x46 możemy tutaj zapisać maksymalnie 64 rejestry 16-bitowe albo 32 rejestry 32-bitowe. Na początku pola danych polecenia wskazujemy ilość rejestrów do zapisu. Umieszczamy tam 2 znaki reprezentujące tę ilość szesnastkowo. Dalej umieszczamy pełny, alfanumeryczny adres rejestru sformatowany zgodnie z tabelą 3.2. Pełny adres rejestru 16-bitowego składa się z 6 znaków, natomiast 32-bitowego z 7 znaków. Po adresie podajemy wartości kolejnych rejestrów. Wartości rejestrów 16-bitowych przesyłamy jako 4 znaki, natomiast 32-bitowych jako 8 znaków każdy.

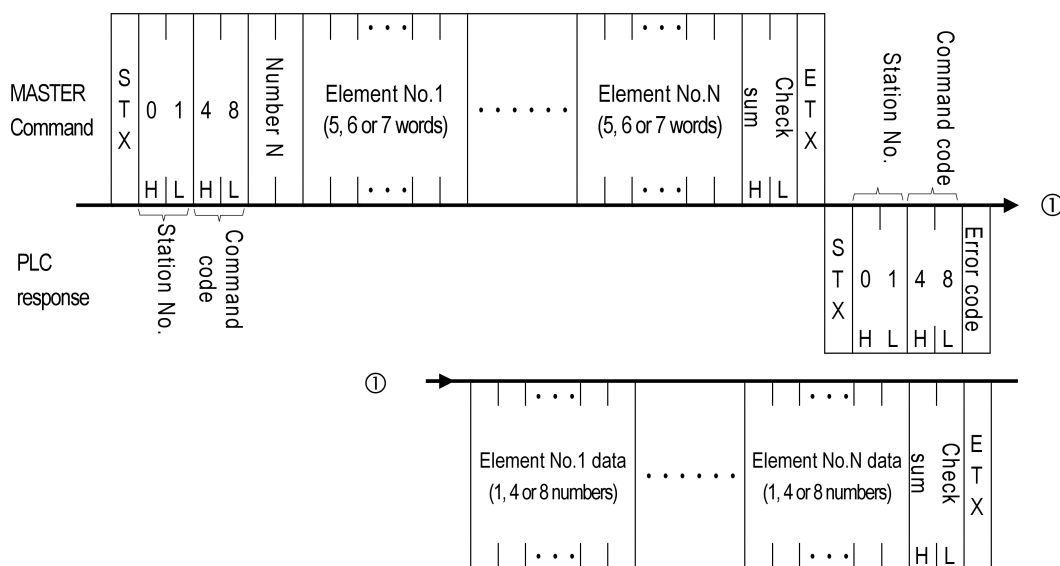
Przykład W rejestrze WY8 zapisujemy wartość 0xAAAA, a w rejestrze WY24 wartość 0x5555. Pozornie rejestry te nie wyglądają na kolejne, ale w rzeczywistości, z uwagi na tabelę 3.2, każdy z nich odpowiada 16 rejestrów 1-bitowym. Tak więc chcemy zapisać 32 kolejne bity zaadresowane jako WY8. W danych umieszczamy ilość rejestrów 02, adres pierwszego rejestru WY00008, a po nim napis reprezentujący 2 wartości AAAA5555.



Rysunek 3.18: Przykład komunikatu polecenia 0x47.

3.4.11 Polecenie 0x48 – odczyt wartości dowolnych rejestrów

Polecenie o kodzie 0x48 używamy do doczytania wartości dowolnych, niekoniecznie kolejnych rejestrów. Schemat żądania i odpowiedzi przedstawiono na rys. 3.19.

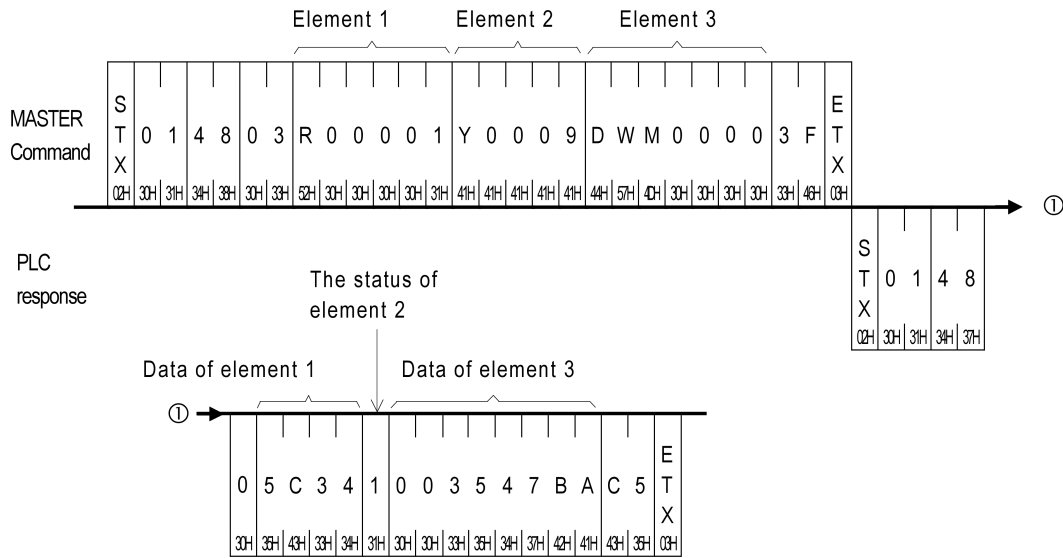


Rysunek 3.19: Schemat komunikatu polecenia 0x48.

Przy pomocy tego polecenia możemy odczytywać wartości do 64 różnych rejestrów. Pierwsze dwa znaki pola danych to ilość rejestrów do odczytania. Standardowo jest to napis reprezentujący tę ilość w postaci szesnastkowej. Dalej w polu danych umieszczamy pełne, alfanumeryczne adresy rejestrów do odczytania sformatowane zgodnie z tabelą 3.2. Pojedynczy adres zajmuje 5 znaków dla rejestru 1-bitowego, 6 znaków dla rejestru 16-bitowego i 7 znaków dla rejestru 32-bitowego.

W odpowiedzi otrzymujemy wartości rejestrów w tej kolejności jak przesłane zostały adresy w żądaniu. Na rejestr 1-bitowy przypada 1 znak odpowiedzi, na rejestr 16-bitowy 4 znaki i na rejestr 32-bitowy 8 znaków. Są to wartości w kodzie szesnastkowym.

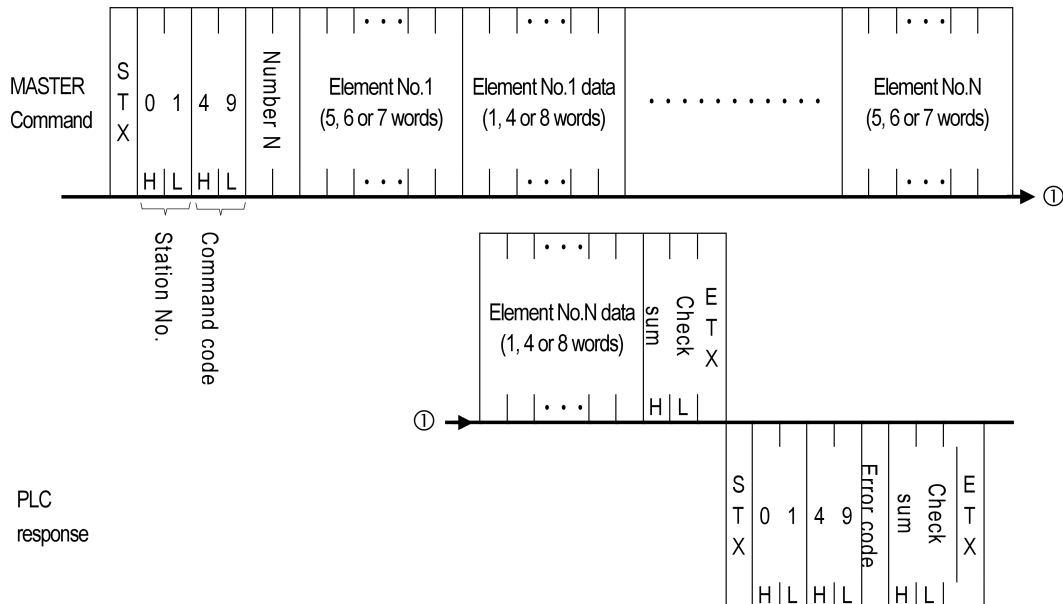
Przykład Jeśli chcemy odczytać wartości rejestrów R1, Y9 i DWM0, to w polu danych komunikatu wysyłamy ciąg znaków utworzony przez trzy następujące napisy: R00001, Y0009, DWM0000. W przykładzie na rysunku 3.20 w odpowiedzi otrzymujemy odpowiednio: 5C34, 1, 003547BA. Zwróćmy uwagę, że rejestr DWM0 jest utworzony przez 32 kolejne rejestry 1-bitowe od M0 do M31 włącznie. Tak więc wartość DWM0 to 32-bitowa liczba całkowita której poszczególne bity to rejestry od M0 do M31, przy czym najmniej znaczący bit znajduje się w M0.



Rysunek 3.20: Przykład komunikatu polecenia 0x48.

3.4.12 Polecenie 0x49 – zapis wartości dowolnych rejestrów

Polecenie o kodzie 0x49 używamy do zapisu wartości dowolnych, niekoniecznie kolejnych rejestrów. Schemat żądania i odpowiedzi przedstawiono na rys. 3.21.

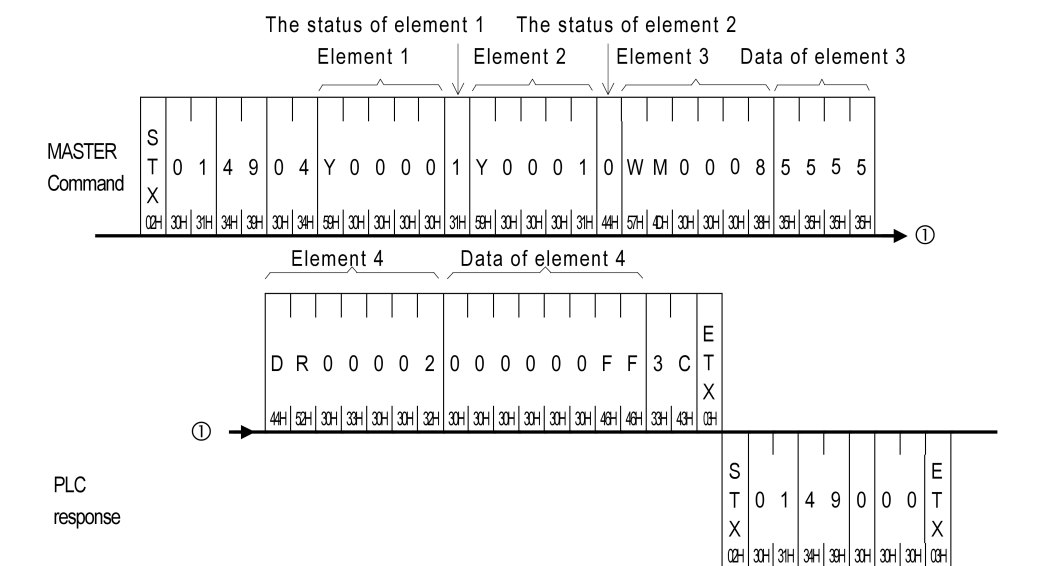


Rysunek 3.21: Schemat komunikatu polecenia 0x49.

Za pomocą tego polecenia możemy zapisywać wartości do 32 różnych rejestrów. Pierwsze dwa znaki pola danych to ilość rejestrów do zapisania. Standardowo jest to napis reprezentujący tę ilość w postaci szesnastkowej. Dalej w polu danych umieszczamy pełne, alfanumeryczne adresy rejestrów, których wartości chcemy zapisać, sformatowane zgodnie z tabelą 3.2. Pojedynczy adres zajmuje 5 znaków dla rejestru 1-bitowego, 6 znaków dla rejestru 16-bitowego i 7 znaków dla rejestru 32-bitowego. Bezpośrednio za każdym adresem znajduje się wartość danego rejestru w postaci 1 znaku dla rejestru 1-bitowego, 4 znaków dla rejestru 16-bitowego i 8 znaków dla rejestru 32-bitowego.

W odpowiedzi nie otrzymujemy żadnych danych.

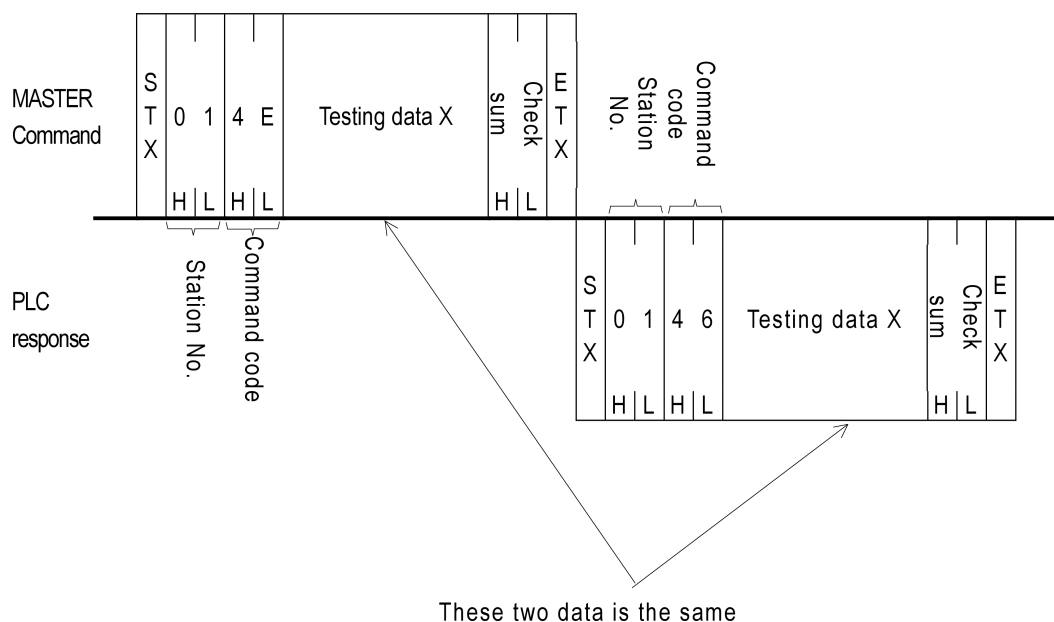
Przykład Powiedzmy, że do rejestru Y0 zapisujemy 1, do Y1 zapisujemy 0, do WM8 zapisujemy 0x5555, a do DR2 zapisujemy 0xFF. Wówczas w polu danych komunikatu umieszczamy ciągi znaków: Y0000, 1, Y0001, 0, WM0008, 5555, DR00002, 000000FF tak jak na rysunku 3.22.



Rysunek 3.22: Przykład komunikatu polecenia 0x49.

3.4.13 Polecenie 0x4E – testowa pętla zwrotna

Polecenie o kodzie 0x4E służy do testowania, czy komunikaty docierają do sterownika w niezmienionej postaci, tak jak zostały wysłane. Schemat żądania i odpowiedzi przedstawiono na rys. 3.23.

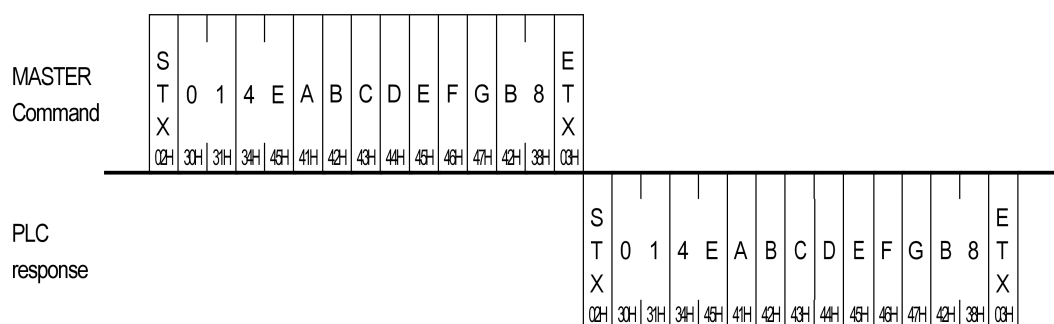


Rysunek 3.23: Schemat komunikatu polecenia 0x4E.

Ciąg znaków alfanumerycznych przekazany w żądaniu powinien zostać zwrócony w odpowiedzi. Zauważmy, że wyjątkowo odpowiedź w tym wypadku nie zawiera kodu błędu.

Polecenie to służy wyłącznie do testowania komunikacji ze sterownikiem, nie wpływa na działanie sterownika i na program PLC, który na nim działa.

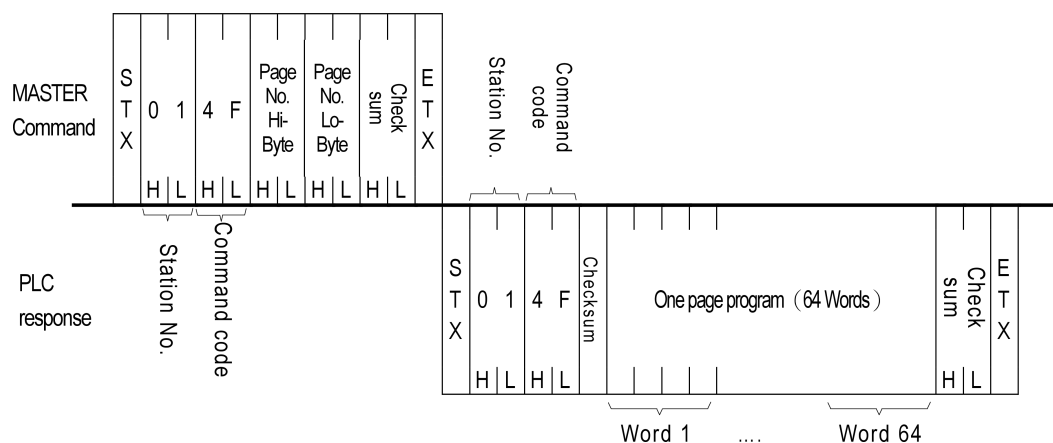
Przykład Wysyłamy ciąg 7 znaków ABCDEFG, aby sprawdzić, czy sterownik udzieli odpowiedzi i czy nie ma przekłamań podczas komunikacji.



Rysunek 3.24: Przykład komunikatu polecenia 0x4E.

3.4.14 Polecenie 0x4F – odczyt programu ze sterownika

Polecenie o kodzie 0x4F używane jest do odczytania programu ze sterownika. Schemat żądania i odpowiedzi przedstawiono na rys. 3.25.



Rysunek 3.25: Schemat komunikatu polecenia 0x4F.

Przed odczytaniem programu PLC ze sterownika urządzenie nadrzędne powinno odczytać szczegółowy status sterownika za pomocą polecenia o kodzie 0x53. Pola STATUS₅ i STATUS₆ odpowiedzi na to polecenie zawierają rozmiar programu znajdującego się w sterowniku. Rozmiar programu N wylicza się w następujący sposób:

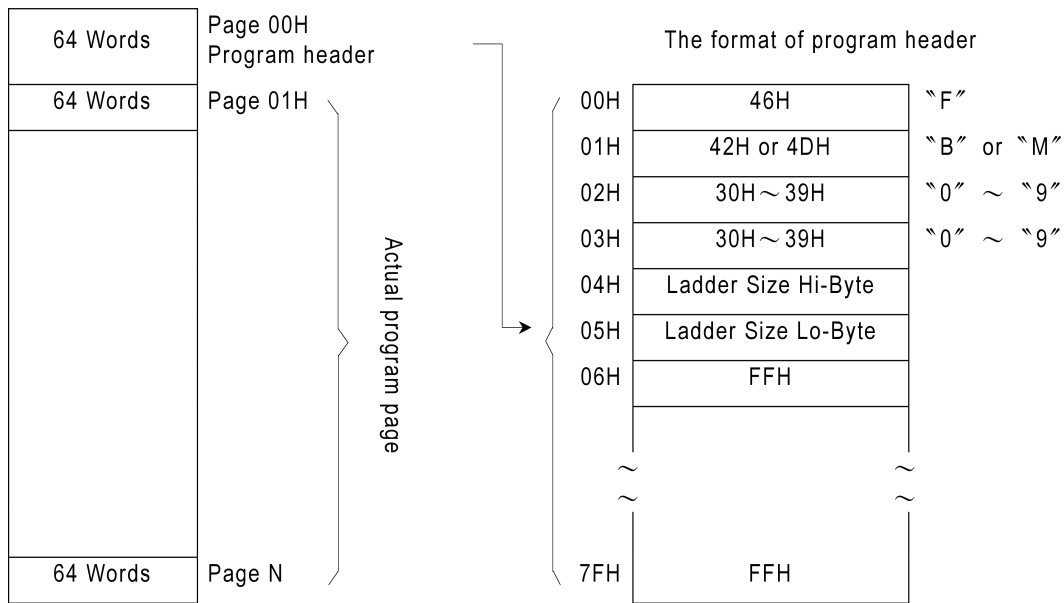
$$N = \frac{\text{STATUS}_5 * 256 + \text{STATUS}_6}{64}.$$

W jednym poleceniu 0x4F możemy odczytać maksymalnie jedną stronę programu, czyli 64 słowa. Dlatego musimy wcześniej znać rozmiar programu, aby wiedzieć ile razy wykonać to polecenie do odczytania całego programu.

Pierwsza odczytywana strona zawiera nagłówek programu PLC. Dlatego też, należy przeczytać $N + 1$ stron, aby uzyskać cały program z nagłówkiem.

Z uwagi na to, że w nagłówku programu (por. rys. 3.26), w bajtach 4 i 5, także znajduje się rozmiar programu, zamiast odczytywać szczegółowy status sterownika można pobrać nagłówek programu, obliczyć ilość stron do pobrania i kontynuować odczyt programu.

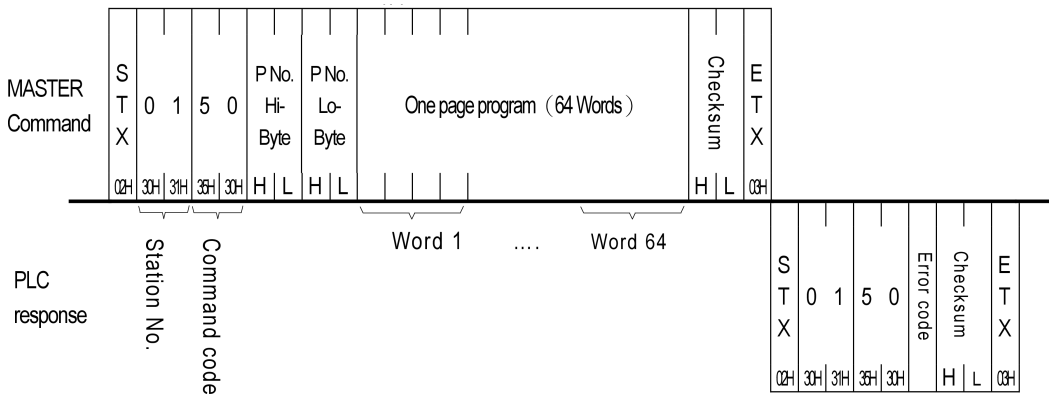
Aby przeczytać konkretną stronę programu w polu danych danych przekazujemy dwa znaki. Pierwszy to heksadecymalna reprezentacja bardziej znaczącego bajtu numeru strony, a drugi znak to heksadecymalna reprezentacja mniej znaczącego bajtu numeru strony.



Rysunek 3.26: Podział programu PLC na strony i schemat jego nagłówka.

3.4.15 Polecenie 0x50 – zapis programu do sterownika

Polecenie o kodzie 0x50 używane jest w celu zapisania programu do sterownika. Schemat żądania i odpowiedzi przedstawiono na rys. 3.27.



Rysunek 3.27: Schemat komunikatu polecenia 0x50.

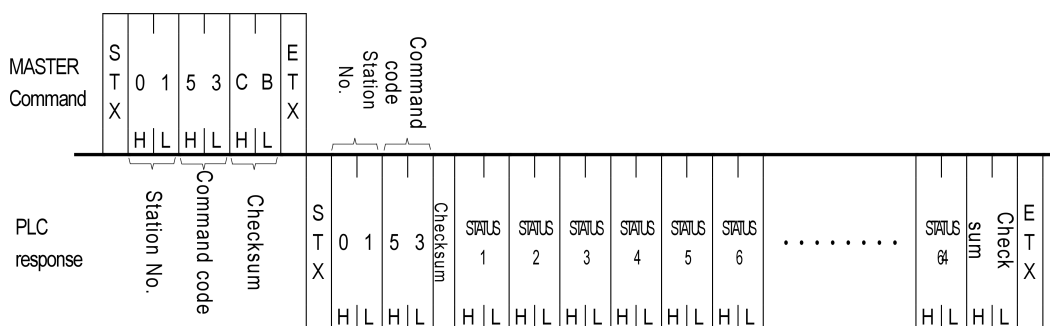
Jedno polecenie 0x50 może przesłać 1 stronę zawierającą do 64 słów. Aby zapisać program PLC do sterownika najpierw należy przygotować nagłówek tego programu zgodnie ze schematem na rysunku 3.26 i przesłać go jako pierwszą stronę. Następnie dzielimy kod programu na strony, maksymalnie po 64 słowa każda. Polecenie 0x50 wykonujemy tyle razy ile jest właściwych stron

programu PLC do przesłania, za każdym razem przesyłając kolejną stronę programu.

Aby zapisać konkretną stronę programu w polu danych danych przekazujemy dwa znaki. Pierwszy to heksadecymalna reprezentacja bardziej znaczącego bajtu numeru strony, a drugi znak to heksadecymalna reprezentacja mniej znaczącego bajtu numeru strony. Dalej w polu danych umieszczamy kod programu PLC.

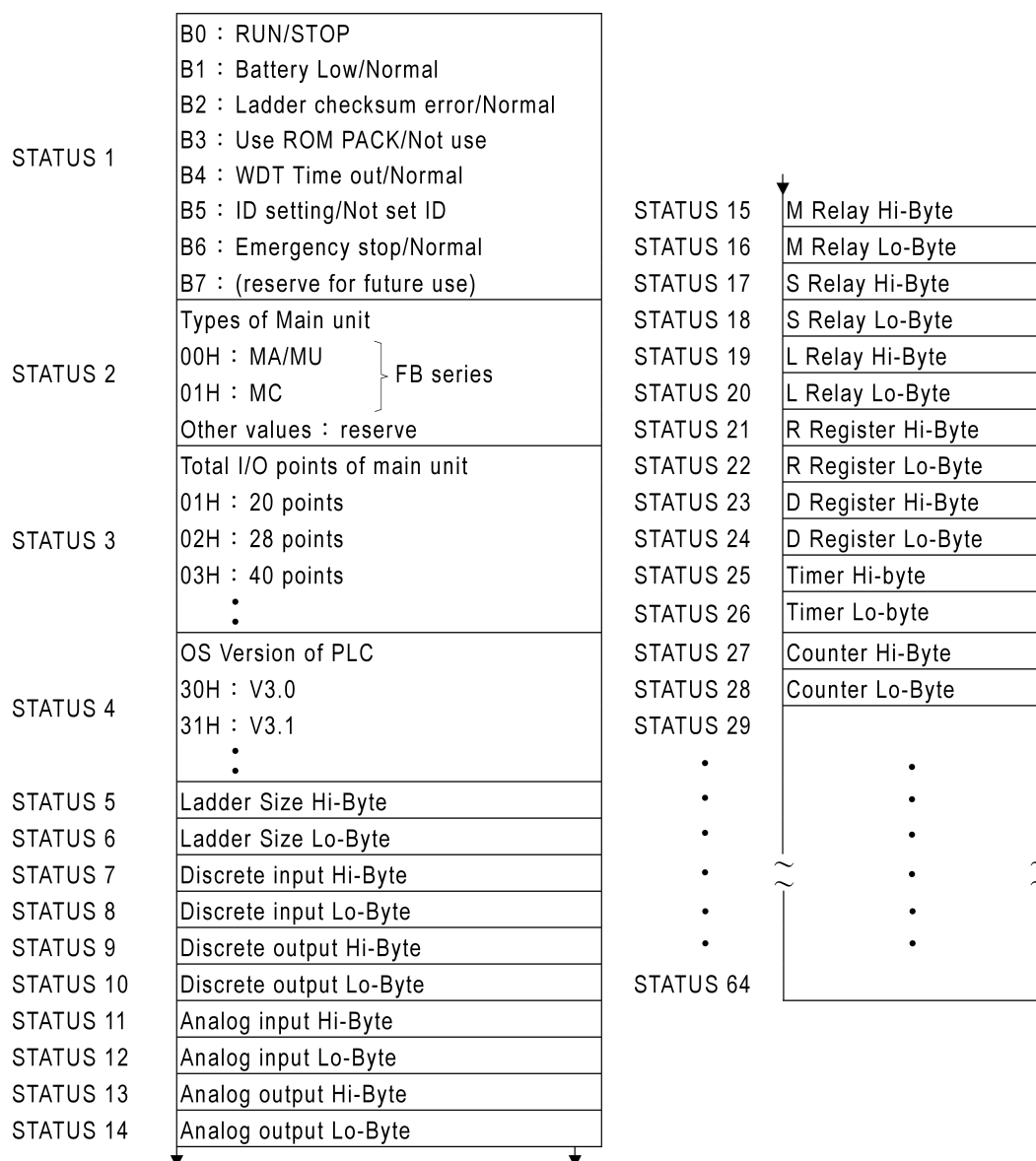
3.4.16 Polecenie 0x53 – odczyt szczegółowego statusu sterownika

Polecenia o kodzie 0x53 używa się do odczytania szczegółowego statusu sterownika. Schemat żądania i odpowiedzi przedstawiono na rys. 3.28.



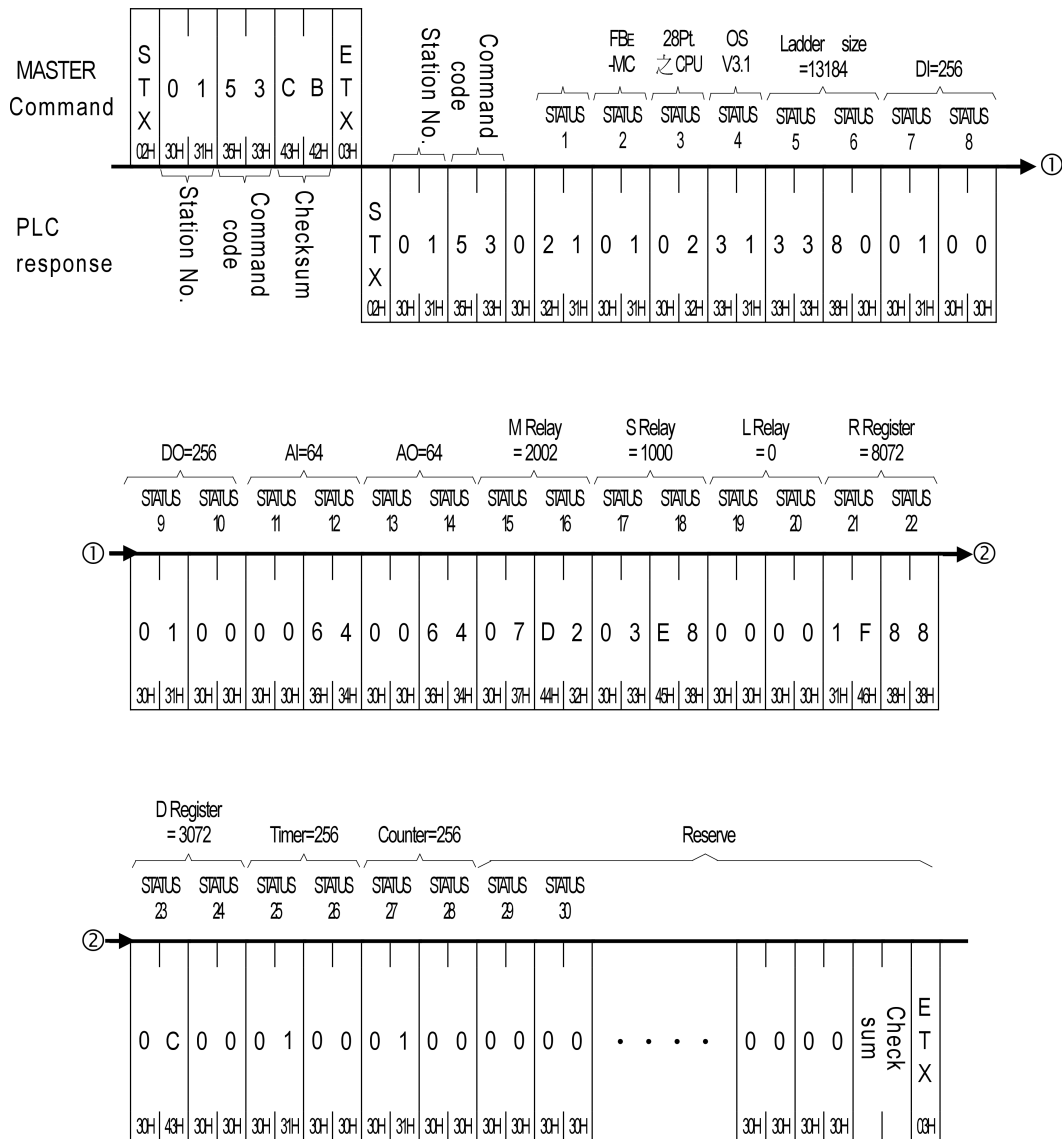
Rysunek 3.28: Schemat komunikatu polecenia 0x53.

W żądaniu tego polecenia nie przesyłamy żadnych danych. W odpowiedzi natomiast dostajemy 128 znaków, po 2 znaki na 1 bajt. Tak więc mamy 64 bajty nazywane jako STATUS₁, STATUS₂ itd. Znaczenie poszczególnych bitów i bajtów statusu zebrano na rysunku 3.29.



Rysunek 3.29: Schemat szczegółowego statusu sterownika.

Przykład Jeśli posiadany model sterownika PLC to FBE-28MC, wersja systemu operacyjnego to 3.10, rozmiar programu wynosi 13k słów, sterownik nie jest wyposażony w ROM PACK, ID nie są ustawione, parametry sterownika są ustawione na wartości domyślne i sterownik jest uruchomiony w normalnym trybie to komunikat odczytu szczegółowego statusu sterownika będzie wyglądał jak na rysunku 3.30.



Rysunek 3.30: Przykład szczegółowego statusu sterownika.

Rozdział 4

Biblioteki programistyczne

W ramach pracy powstały dwie biblioteki: jedna napisana w języku C, a druga w PHP. Stanowią one interfejs pomiędzy sterownikami PLC marki FATEK a oprogramowaniem, które wymaga komunikacji z takimi sterownikami. W obu bibliotekach zaimplementowane zostały wszystkie polecenia z protokołu komunikacyjnego FACON opisanego w rozdziale 3 z wyjątkiem poleceń 0x4F i 0x50 do odczytu i zapisu kodu programu w sterowniku PLC.

4.1 API w C

4.1.1 Stałe

Funkcje biblioteki mogą zwracać jedną z poniższy wartości:

```
#define FATEK_OK 0
#define FATEK_ERROR_READ (-1)
#define FATEK_ERROR_WRITE (-2)
#define FATEK_ERROR_RESPONSE (-3)
#define FATEK_ERROR_CHECKSUM (-4)
#define FATEK_ERROR_OVERFLOW (-5)
#define FATEK_ERROR_MEMALLOC (-6)
#define FATEK_ERROR_REGTYPE (-7)
```

Ich znaczenie jest następujące:

FATEK_OK operacja została wykonana pomyślnie.

FATEK_ERROR_READ błąd podczas odczytu danych ze sterownika PLC,

FATEK_ERROR_WRITE błąd podczas przekazywania danych do sterownika PLC,

FATEK_ERROR_RESPONSE nieprawidłowy format komunikatu lub niepoprawny rozmiar pola danych,

FATEK_ERROR_CHECKSUM błąd sumy kontrolnej podczas odczytu danych ze sterownika PLC,

FATEK_ERROR_OVERFLOW za duży rozmiar pola danych,

FATEK_ERROR_MEMALLOC nie można zaalokować odpowiedniej ilości pamięci,
FATEK_ERROR_REGTYPE niepoprawny typ rejestru.

W sterownikach PLC marki FATEK mamy trzy typy rejestrów: 1-bitowe, 16-bitowe oraz 32-bitowe. Odpowiadają im następujące stałe określające ilość bitów na dany rejestr:

```
#define FATEK_REG_TYPE_BOOL      0x01
#define FATEK_REG_TYPE_16BIT     0x10
#define FATEK_REG_TYPE_32BIT     0x20
```

Adres rejestru prefiksowany jest symbolem tego rejestru. Możliwe symbole zostały zebrane w tabeli 3.2. Przypisujemy im stałe będące wartościami kodów ASCII odpowiednich znaków:

```
#define FATEK_REG_SYMBOL_X      0x58
#define FATEK_REG_SYMBOL_Y      0x59
#define FATEK_REG_SYMBOL_M      0x4D
#define FATEK_REG_SYMBOL_S      0x53
#define FATEK_REG_SYMBOL_T      0x54
#define FATEK_REG_SYMBOL_C      0x43
#define FATEK_REG_SYMBOL_R      0x52
#define FATEK_REG_SYMBOL_D      0x44
```

Rejestry poza wartością mają określony stan. Mogą być wyłączone z użycia lub załączone do użycia w programie PLC. Poza tym dla rejestrów 1-bitowych można ich wartość ustawić na 1 lub 0. Wartości odpowiednich stałych zgodne są z protokołem FACON (rysunek 3.7):

```
#define FATEK_REG_STATE_DISABLE 0x01
#define FATEK_REG_STATE_ENABLE  0x02
#define FATEK_REG_STATE_SET      0x03
#define FATEK_REG_STATE_RESET   0x04
```

W zwracanym przez polecenie o kodzie 0x40 uproszczonym statusie sterownika PLC jest rozmiar programu PLC. Może on mieć specjalne znaczenie, gdy program w PLC jest programem krokowym, nie drabinkowym. Możliwe wówczas są następujące wartości:

```
#define FATEK_CAPACITY_FB      0xFF
#define FATEK_CAPACITY_FBE8     0x53
#define FATEK_CAPACITY_FBE13    0x54
#define FATEK_CAPACITY_FBN8     0x55
#define FATEK_CAPACITY_FBN13    0x56
```

4.1.2 Struktury

Ponieważ żądanie i odpowiedź w komunikacji ze sterownikiem PLC mają praktycznie tę samą budowę (rysunek 3.2) w bibliotece przewidziano jedną, wspólną strukturę opisującą taki komunikat:

```
typedef struct fatek_msg_s {
    int    station;
    int    command;
    int    error;
    char   *data;
    size_t size;
} fatek_msg_t;
```

Pole **station** to numer stacji identyfikujący sterownik PLC, **command** to kod komunikatu (tabela 3.3), **error** to ewentualny kod błędu w odpowiedzi, **data** to ciąg znaków przekazywanych danych, a **size** to długość tego ciągu.

Komunikacja ze sterownikiem PLC polega głównie na odczycie i zapisie wartości rejestrów. Dlatego też bardzo ważna jest struktura opisująca taki rejestr:

```
typedef struct fatek_reg_s {
    uint32_t type;
    uint32_t symbol;
    uint32_t address;
    uint32_t state;
    uint32_t value;
} fatek_reg_t;
```

Dla każdego rejestru określony jest jego typ w polu **type**, symbol w polu **symbol**, adres numeryczny w polu **address**. Te dane zwykle nie zmieniają się w trakcie wykonania programu. To co zmienia się podczas komunikacji ze sterownikiem to stan rejestru **state** oraz wartość rejestru **value**.

Uproszczony status sterownika PLC opisuje następująca struktura:

```
typedef struct fatek_status_s {
    uint8_t run : 1;
    uint8_t battery : 1;
    uint8_t ladder : 1;
    uint8_t rom_pack : 1;
    uint8_t wdt_status : 1;
    uint8_t id_status : 1;
    uint8_t emergency : 1;
    uint8_t : 1;
    uint8_t capacity;
} fatek_status_t;
```

Pierwszych 7 pól to wartości bitowe opisane w tabeli 3.4. Pole **capacity** to rozmiar programu drabinkowego w sterowniku PLC. Jeśli program jest programem krokowym wówczas pole to ma specjalne wartości takie jak w tabeli 3.5.

4.1.3 Zmienne globalne

W bibliotece **fatek** przewidziano dwie zmienne globalne do sterowania tybem diagnostycznym.

```
int io_debug = 0;
int fatek_debug = 0;
```


Przypisanie `io_debug` wartości niezerowej przy kompilacji spowoduje wypisywanie na `stderr` informacji diagnostycznych na poziomie komunikacji za pomocą gniazd (ang. sockets). Wypisywane będą surowe dane wysyłane do i ze sterownika PLC, co pomaga podczas uruchamiania programów korzystających z biblioteki.

Zmienna `fatek_debug` odpowiada za wypisywanie na `stderr` danych diagnostycznych na poziomie funkcji bibliotecznych. Wypisywane są parametry oraz wyniki działania poszczególnych funkcji z biblioteki.

4.1.4 Funkcje

Najważniejszą częścią biblioteki są oczywiście funkcje interfejsu. Funkcje w bibliotece `fatek` można podzielić na trzy rodzaje: są dwie funkcje pomocnicze, jedna ogólna realizująca przekazanie komunikatu oraz te najważniejsze, odpowiadające poszczególnym poleceniom z protokołu komunikacyjnego FACON.

Nawiązanie połączenia

```
int fatek_connect(const char *host, uint16_t port);
```

Funkcja `fatek_connect()` służy do nawiązania połączenia TCP/IP ze sterownikiem PLC, którego nazwa lub adres IP podane są w `host`, natomiast jego port TCP w `port`. Standardowy port komunikacyjny TCP to 500.

Po nawiązaniu połączenia przechowywany jest wewnętrznie punkt końcowy (ang. endpoint) do utworzonego gniazda komunikacyjnego (ang. socket). Może być tylko jeden punkt końcowy i jedno gniazdo. W konsekwencji, przy pomocy biblioteki `fatek` można nawiązać połączenie tylko z jednym sterownikiem PLC. Aby połączyć się z innym, należy rozłączyć się z pierwszym. Rzadko jednak w praktyce mamy więcej niż jeden sterownik PLC. Zwykle jest jeden sterownik rozbudowany o dodatkowe moduły rozszerzające. W przyszłości można nieco zmodyfikować bibliotekę umożliwiając łączność z wieloma sterownikami PLC w tym samym czasie.

Jako pierwszy argument `station` w funkcjach realizujących poszczególne polecenia protokołu FACON przekazywany jest numer stacji identyfikujący sterownik PLC. Ponieważ komunikacja ograniczona jest do jednego tylko sterownika, a adres IP i port jednoznacznie go wyznaczają, może się to wydać nadmiarowe. W łatwy jednak sposób można, modyfikując tylko warstwę I/O wydzieloną w `io.c` w bibliotece, łączyć się z wieloma sterownikami PLC w sieci RS485. Wówczas konieczna jest identyfikacja poprzez numer stacji, a uchwyt połączeniowy (ang. handle) jest tylko jeden tak jak jedno jest gniazdo.

Funkcja ta, jak wszystkie kolejne, zwraca `FATEK_OK` lub kod błędu.

Zakończenie połączenia

```
int fatek_disconnect(void);
```

Aby zakończyć połączenie ze sterownikiem wołamy `fatek_disconnect()`.

Przesłanie komunikatu

```
int fatek_message(fatek_msg_t *req, fatek_msg_t *rsp);
```

Funkcja `fatek_message()` służy do wysłania żądania określonego strukturalną wskazywaną przez `req` i odebrania odpowiedzi, która zostanie umieszczona w strukturze wskazywanej przez `rsp`. Wołający tę funkcję musi zadbać o odpowiednie zaalokowanie pamięci dla struktur żądania i odpowiedzi, a potem o jej zwolnienie. Zwolnić należy również bufora danych odpowiedzi `rsp->data`.

W praktyce rzadko będzie potrzeba korzystania z tej funkcji bo wszystkie polecenia protokołu zostały zaimplementowane jako osobne funkcje.

Odczyt uproszczonego statusu sterownika

```
int fatek_get_status(int station, fatek_status_t *status);
```

Przy pomocy funkcji `fatek_get_status()` odczytujemy uproszczony status sterownika PLC. Wynik umieszczany jest w strukturze wskazywanej przez `status`. Wołający musi zadbać o zaalokowanie i zwolnienie pamięci wskazywanej przez `status`. Opis struktury `fatek_status_t` znajduje się w podrozdziale 4.1.2.

Uruchomienie i zatrzymanie sterownika

```
int fatek_runstop(int station, int ccode);
```

Funkcja `fatek_runstop()` służy do zatrzymania i wznowienia pracy sterownika PLC. Jeśli `ccode` ma wartość 0 sterownik zostanie zatrzymany, gdy 1 to zostanie on uruchomiony. Kilkukrotne zawołanie tej funkcji, z tą samą wartością `ccode`, nie ma żadnego wpływu na działanie sterownika PLC.

Zapis statusu rejestru 1-bitowego

```
int fatek_set_state(int station, fatek_reg_t reg);
```

Tej funkcji używamy w celu zmiany statusu rejestru 1-bitowego. W `reg` znajduje się opis rejestru, to znaczy, jego typ, symbol, adres numeryczny i status jaki ma być ustawiony. Dopuszczalne są tutaj tylko rejestry 1-bitowe.

Odczyt statusu kolejnych rejestrów 1-bitowych

```
int fatek_get_state(int station, fatek_reg_t regs[], int number);
```

Funkcja `fatek_get_state()` służy do odczytu statusu kolejnych, w sensie adresów, rejestrów 1-bitowych. Liczba `number` może przyjmować wartości od 0 do 255 włącznie. Jeśli `number` wynosi 0, to odczytanych zostanie 256 statusów rejestrów. W tablicy `regs` powinno być co najmniej `number` rejestrów. Istotną rolę odgrywa pierwszy z nich. Odczytany zostanie status kolejnych `number` rejestrów począwszy od pierwszego rejestru z tablicy `regs`. Wszystkim `number` rejestrom w tej tablicy zostanie przypisany typ i symbol takie jak pierwszego rejestru oraz adres i status kolejno odczytanego rejestru ze sterownika PLC.

Odczyt i zapis wartości kolejnych rejestrów

```
int fatek_get_bvalue(int station , fatek_reg_t regs [], int number);  
int fatek_set_bvalue(int station , fatek_reg_t regs [], int number);  
int fatek_get_ivalue(int station , fatek_reg_t regs [], int number);  
int fatek_set_ivalue(int station , fatek_reg_t regs [], int number);
```

Odczyt i zapis wartości jednego rejestru w jednym komunikacie byłby nieefektywny. Narzut wymagany przez protokół jest stosunkowo duży do danych dla jednego rejestru. Dlatego w sterownikach FATEK przewidziano możliwość odczytu i zapisu wielu rejestrów w jednym komunikacie. Wymienione wyżej funkcje pozwalają odczytać i zapisać wartości kolejnych, w sensie adresów, rejestrów. Typ oraz symbol odczytywanych bądź zapisywanych rejestrów są zawsze takie same, wyznaczone poprzez pierwszy rejestr z tablicy `regs`.

Funkcje `fatek_get_bvalue()` i `fatek_set_bvalue()` służą odpowiednio do odczytu i zapisu wartości kolejnych rejestrów 1-bitowych, natomiast funkcje `fatek_get_ivalue()` i `fatek_set_ivalue()` do odczytu i zapisu wartości kolejnych rejestrów 16 lub 32-bitowych.

W `fatek_get_bvalue()` i `fatek_set_bvalue()` liczba `number` może przyjmować wartości od 0 do 255 włącznie. Jeśli `number` wynosi 0, to odczytanych bądź zapisanych zostanie 256 wartości rejestrów. W `fatek_get_ivalue()` i `fatek_set_ivalue()` liczba `number` może przyjmować wartości z zakresu od 1 do 64.

Tablica `regs` powinna zawierać przynajmniej `number` rejestrów. Istotną rolę odgrywa pierwszy z nich. Typy, symbole oraz adresy pozostałych są ignorowane. Przy odczycie, ze sterownika PLC zostaną pobrane wartości kolejnych `number` rejestrów począwszy od pierwszego rejestru z tablicy `regs`. Wszystkim `number` rejestrom w tej tablicy zostanie przypisany typ i symbol pierwszego rejestru, oraz adres i wartość kolejno odczytanego rejestru. Przy zapisie natomiast w sterowniku PLC zapisane zostaną wartości kolejnych `number` rejestrów począwszy od pierwszego rejestru z tablicy `regs`. Ignorowane są typ, symbol oraz adres wszystkich rejestrów w tablicy `regs` poza pierwszym.

Odczyt wartości dowolnych rejestrów

```
int fatek_get_value(int station , fatek_reg_t regs [], int number);
```

W praktyce `fatek_get_value()` to najczęściej używana funkcja ponieważ pozwala odczytać wartości wielu, różnych, pod względem typu i adresu, rejestrów naraz w jednym komunikacie. Typ, symbol oraz adres rejestru, którego wartość ma zostać odczytana znajdują się w poszczególnych pozycjach tablicy `regs`. Po zawołaniu tej funkcji, na każdej z `number` pozycji tej tablicy zostanie umieszczona wartość rejestru odczytana z se sterownika PLC.

Liczba `number` może przyjmować wartości od 1 do 64 włącznie.

Zapis wartości dowolnych rejestrów

```
int fatek_set_value(int station, fatek_reg_t regs [], int number);
```

`fatek_set_value()` to druga, z najczęściej używanych w oprogramowaniu do komunikacji ze sterownikiem PLC, funkcji. Umożliwia zapisanie wartości wielu rejestrów o różnych typach i adresach w jednym komunikacie, oszczędzając na transmisji danych. W tablicy `regs` znajduje się przynajmniej `number` rejestrów. Do sterownika PLC zapisane zostanie `number` wartości rejestrów zgodnie z podanymi typami, symbolami i adresami w tablicy `regs`.

Liczba `number` może przyjmować wartości od 1 do 32 włącznie.

Testowa pętla zwrotna

```
int fatek_loopback(int station, char *data);
```

Funkcja `fatek_loopback()` używana jest do testowania komunikacji ze sterownikiem PLC. Do sterownika wysyłany jest ciąg znaków podany w `data`. Jeśli przekazano wartość `NULL` to wysłany zostanie ciąg znaków:

```
TEST abcdefghijklmnopqrstuvwxyz 0123456789
```

Funkcja zwraca `FATEK_OK` jeśli ciągi znaków w żądaniu i odpowiedzi są takie same, natomiast `FATEK_ERROR_RESPONSE` w przeciwnym razie.

Odczyt szczegółowego statusu sterownika

```
int fatek_get_details(int station);
```

Do odczytania szczegółowych parametrów sterownika PLC służy funkcja `fatek_get_details()`. Przy obecnej implementacji na `stdout` wypisywane są surowe dane odczytane za sterownika w formie tablicy, także z podanymi wartościami w postaci binarnej.

4.2 API w PHP

PHP jest składniowo językiem bardzo przypominającym C, co ułatwia przetłumaczenie kodu z C na PHP. Interfejsy w C i w PHP są bardzo podobne. Różnice wynikają oczywiście ze składni tych języków oraz z tego, że interfejs w PHP jest obiektowy. Implementacja w PHP znacznie się upraszcza bo nie ma konieczności pilnowania gospodarki pamięcią, alokowania i zwalniania pamięci.

4.2.1 Stałe

W PHP mamy analogiczne stałe jak w C. W wyniku uproszczenia komunikacji za pomocą gniazd w PHP nie mamy tak szczegółowych informacji o błędach. Pozostałe stałe mają te same wartości.

```
define( 'FATEK_OK' , 0 );
define( 'FATEK_ERROR_RESPONSE' , -1 );

define( 'FATEK_REG_TYPE_BOOL' , 0x01 );
define( 'FATEK_REG_TYPE_16BIT' , 0x10 );
define( 'FATEK_REG_TYPE_32BIT' , 0x20 );

define( 'FATEK_REG_SYMBOLX' , 0x58 );
define( 'FATEK_REG_SYMBOLY' , 0x59 );
define( 'FATEK_REG_SYMBOLM' , 0x4D );
define( 'FATEK_REG_SYMBOLS' , 0x53 );
define( 'FATEK_REG_SYMBOLT' , 0x54 );
define( 'FATEK_REG_SYMBOLC' , 0x43 );
define( 'FATEK_REG_SYMBOLR' , 0x52 );
define( 'FATEK_REG_SYMBOLD' , 0x44 );

define( 'FATEK_REG_STATE_DISABLE' , 0x01 );
define( 'FATEK_REG_STATE_ENABLE' , 0x02 );
define( 'FATEK_REG_STATE_SET' , 0x03 );
define( 'FATEK_REG_STATE_RESET' , 0x04 );

define( 'FATEK_CAPACITY_FB' , 0xFF );
define( 'FATEK_CAPACITY_FBE8' , 0x53 );
define( 'FATEK_CAPACITY_FBE13' , 0x54 );
define( 'FATEK_CAPACITY_FBN8' , 0x55 );
define( 'FATEK_CAPACITY_FBN13' , 0x56 );
```

4.2.2 Klasy

Cały interfejs programistyczny biblioteki `fatek` w PHP składa się z trzech klas reprezentujących: komunikat, rejestr oraz sterownik PLC. Prywatne właściwości i metody zostały ukryte.

Klasa komunikat

```
class FatekMessage {
    public $station;
    public $command;
    public $error;
    public $data;
    public $size;
}
```

Klasa `FatekMessage` reprezentuje żądanie i odpowiedź w komunikacji ze sterownikiem PLC. Odpowiada ona strukturze `fatek_msg_s` w C.

Klasa rejestr

```
class FatekRegister {
    public $type;
    public $symbol;
    public $address;
    public $state;
    public $value;

    public function __construct($type, $symbol, $address, $value = 0);
    public function details();
    public function format_symbol_address();
    public function format_symbol_address_value();
}
```

Klasa `FatekRegister` to odpowiednik struktury `fatek_reg_s`. Posiada ona kilka użytecznych metod. Konstruktor wymaga określenia typu `$type`, symbolu `$symbol` oraz numerycznego adresu `$address`. Opcjonalnie można podać wartość rejestru `$value`.

`details()`

Metoda wypisuje sformatowane własności rejestru.

`format_symbol_address()`

Metoda zwraca sformatowany, pełny adres rejestru zależny od jego typu, zgodny z tabelą 3.2.

`format_symbol_address_value()`

Metoda zwraca konkatenację sformatowanego, pełnego adresu rejestru (jak `format_symbol_address()`) oraz jego wartości, gotowe do umieszczenia w polu danych komunikatu.

Klasa sterownik

```
class FatekPLC {
    protected $socket;
```

```
protected $host;
protected $port;

public function __construct($host = PLC_HOST, $port = PLC_PORT,
    $debug = PLC_DEBUG);
public function connect();
public function disconnect();
protected function read();
protected function write($buf);
protected function lrc($buf);
protected function message(FatekMessage $req,
    FatekMessage $rsp);
public function get_status($station);
public function runstop($station, $cocode);
public function set_state($station, FatekRegister $reg);
public function get_state($station, array $regs);
public function get_bvalue($station, array $regs);
public function set_bvalue($station, array $regs);
public function get_ivalue($station, array $regs);
public function set_ivalue($station, array $regs);
public function get_value($station, array $regs);
public function set_value($station, array $regs);
public function loopback($station, $data = NULL);
public function get_details($station);
}
```

Konstruktor klasy `FatekPLC` wymaga podania nazwy hosta bądź jego adresu IP, portu TCP oraz flagi diagnostyki. Można te wartości umieścić w predefiniowanych w pliku konfiguracyjnym stałych: `PLC_HOST`, `PLC_PORT` oraz `PLC_DEBUG`. Domyślny port TCP ma numer 500.

Metody tej klasy odpowiadają funkcjom API analogicznej biblioteki `fatek` napisanej w C.

`connect()`

Metoda tworzy gniazdo (ang. socket) połączenia ze sterownikiem PLC. W przeciwieństwie do implementacji w C nie ma tutaj ograniczenia do ilości sterowników PLC, z którymi jednocześnie odbywa się komunikacja, ponieważ gniazdo jest prywatną własnością każdego obiektu typu `FatekPLC`.

`disconnect()`

Metoda kończy połączenie ze sterownikiem PLC.

`read()`

Metoda zwraca odebrane, surowe dane ze sterownika PLC.

`write($buf)`

Metoda wysyła surowe dane do sterownika PLC umieszczone w `$buf`.

`lrc($buf)`

Metoda zwraca sumę kontrolną dla łańcucha umieszczonego w `$buf`.

`message(FatekMessage $req, FatekMessage $rsp)`

Metoda wysyła żądanie `$req` do sterownika PLC. Odebrana odpowiedź przekazywana jest w `$rsp`. Błędy podczas komunikacji powodują wyrzucenie wyjątku.

`get_status($station)`

Metoda odczytuje uproszczony status sterownika PLC i wypisuje go w formie tabelarycznej na standardowe wyjście.

`runstop($station, $ccode)`

Sterownik PLC zostaje wyłączony, gdy `$ccode` jest równe 0, natomiast załączony gdy `$ccode` jest równe 1.

`set_state($station, FatekRegister $reg)`

Metoda zmienia status rejestru 1-bitowego na określony w `$reg`.

`get_state($station, array &$regs)`

Metoda odczytuje status kolejnych rejestrów 1-bitowych. Tablica `$regs` zawiera obiekty typu `FatekRegister`. Długość tej tablicy powinna wynosić od 1 do 256, poindeksowana powinna być kolejnymi liczbami naturalnymi od 0. Typ, symbol oraz adres początkowy rejestrów do odczytu są takie jak w rejestrze `$regs[0]`. Odczytanych zostanie tyle rejestrów jaka jest długość tablicy `$reg`. Wszystkim rejestrom z tablicy `$regs` zostaną przypisane typ i symbol pierwszego rejestru, kolejny adres i odczytany status ze sterownika PLC.

`get_bvalue($station, array $regs)`

`set_bvalue($station, array $regs)`

`get_ivalue($station, array $regs)`

`set_ivalue($station, array $regs)`

Metody `get_bvalue()` i `set_bvalue()` służą do odczytu i zapisu wartości rejestrów 1-bitowych, natomiast `get_ivalue()` i `set_ivalue()` do odczytu i zapisu wartości rejestrów 16 lub 32-bitowych. Tablica `$regs` zawiera obiekty typu `FatekRegister`. Długość tej tablicy powinna wynosić od 1 do 256 w `get_bvalue()` i `set_bvalue()`, albo od 1 do 64 w `get_ivalue()` i `set_ivalue()`. Poindeksowana powinna być kolejnymi liczbami naturalnymi od 0. O tym, które rejestry są odczytywane lub zapisywane decyduje zawsze rejestr `$regs[0]`. Metody działają tak jak analogiczne funkcje z API w C.

`get_value($station, array $regs)`

`set_value($station, array $regs)`

Metody do odczytu i zapisu wartości wielu, różnych rejestrów w jednym komunikacie. Zasady działania tak jak dla powyższych metod oraz analogicznych funkcji z API w C.

`loopback($station, $data = NULL)`

Metoda realizuje testową pętlę zwrotną na zasadach ja analogiczna funkcja z API w C.

`get_details($station)`

Metoda odczytuje szczegółowy status sterownika PLC i wypisuje go w formie tabularycznej na standardowe wyjście.

Spis tabel

3.1	Kody błędów odpowiedzi w protokole FACON.	20
3.2	Zestawienie dostępnych rejestrów w sterowniku PLC.	21
3.3	Zestawienie dostępnych poleceń w protokole FACON.	22
3.4	Znaczenie bitów statusu sterownika PLC.	23
3.5	Znaczenie drugiej składowej STATUS ₂ uproszczonego statusu sterownika PLC.	24

Spis rysunków

1	Zestaw urządzeń używanych w ramach pracy licencjackiej. . .	2
1.1	System przekaźników używany przed stworzeniem PLC [4]. . .	4
1.2	Pierwszy PLC [5].	5
1.3	Model 184 [5].	5
1.4	Współczesny sterownik PLC marki FATEK [6].	6
1.5	Tryb działania PLC [7].	7
1.6	Diagram blokowy PLC [7].	8
1.7	Programowanie drabinkowe.	9
2.1	Program sterownika do obsługi alarmu antywłamaniowego [8].	11
2.2	Program sterownika do obsługi parkingu [9].	13
2.3	Przykładowy wygląd wyświetlacza [9].	14
2.4	Program sterownika do obsługi zbiornika z grzałką [10].	15
3.1	Urządzenia nadrzędne i podrzędne.	17
3.2	Format komunikatu w protokole FACON.	18
3.3	Schemat komunikatu polecenia 0x40.	23
3.4	Przykład komunikatu polecenia 0x40.	24
3.5	Schemat komunikatu polecenia 0x41.	25
3.6	Przykład komunikatu polecenia 0x41.	25
3.7	Schemat komunikatu polecenia 0x42.	26
3.8	Przykład komunikatu polecenia 0x42.	26
3.9	Schemat komunikatu polecenia 0x43.	27
3.10	Przykład komunikatu polecenia 0x43.	28
3.11	Schemat komunikatu polecenia 0x44.	28
3.12	Przykład komunikatu polecenia 0x44.	29
3.13	Schemat komunikatu polecenia 0x45.	29
3.14	Przykład komunikatu polecenia 0x45.	30
3.15	Schemat komunikatu polecenia 0x46.	31
3.16	Przykład komunikatu polecenia 0x46.	32
3.17	Schemat komunikatu polecenia 0x47.	32
3.18	Przykład komunikatu polecenia 0x47.	33
3.19	Schemat komunikatu polecenia 0x48.	34
3.20	Przykład komunikatu polecenia 0x48.	35

3.21 Schemat komunikatu polecenia 0x49.	35
3.22 Przykład komunikatu polecenia 0x49.	36
3.23 Schemat komunikatu polecenia 0x4E.	37
3.24 Przykład komunikatu polecenia 0x4E.	37
3.25 Schemat komunikatu polecenia 0x4F.	38
3.26 Podział programu PLC na strony i schemat jego nagłówka. . .	39
3.27 Schemat komunikatu polecenia 0x50.	39
3.28 Schemat komunikatu polecenia 0x53.	40
3.29 Schemat szczegółowego statusu sterownika.	41
3.30 Przykład szczegółowego statusu sterownika.	42

Bibliografia

- [1] Konrad Trzeciak, *Interfejs obsługi systemu inteligentny dom opartego na programowalnym sterowniku logicznym*, Uniwersytet w Białymstoku, Instytut Informatyki, praca licencjacka, 2019.
- [2] Dag H. Hanssen *Programmable Logic Controllers: A Practical Approach to IEC 61131-3 using CoDeSys*, 2015.
- [3] *FACON-PLC Communication Protocol*, Fatek Automation Corp., 2001, Ver. 1.1.
- [4] <https://library.automationdirect.com/history-of-the-plc/>
dostęp 11.02.2019
- [5] <https://www.controleng.com/articles/inside-the-competition-for-the-first-plc/>
dostęp 11.02.2019
- [6] http://www.almanachprodukcji.pl/produkty/opis/0/3191/sterownik_plc_fatek_fbs_32maj2_ac
dostęp 10.06.2019
- [7] <http://www.tesla-institute.com/index.php/automation-articles/304-plc-principles-of-operation>
dostęp 11.02.2019
- [8] <https://www.sanfoundry.com/plc-program-burglar-alarm-security-system/>
dostęp 11.02.2019
- [9] <https://www.sanfoundry.com/plc-program-car-parking-system/>
dostęp 11.02.2019
- [10] <https://www.sanfoundry.com/plc-program-heating-liquid-tank-heater/>
dostęp 11.02.2019