Politechnika Warszawska
Zaoczne Studia Magisterskie na Odległość
Wydział Mechatroniki Politechniki Warszawskiej
Kierunek: Automatyka i Robotyka
Specjalność Informatyka Przemysłowa

SYSTEMY CZASU RZECZYWISTEGO

Przedmiot kierunkowy wydziałowy

Michał Bartyś

Materiały dydaktyczne

Warszawa 2008

O autorze

Imię i nazwisko	Michał Bartyś
Kontakt	bartys@mchtr.pw.edu.pl
Wykształcenie	1973 – mgr inż. 1983 – dr inż.
Stanowisko	adiunkt w Instytucie Automatyki i Robotyki Politechniki Warszawskiej
Publikacje naukowe	współautor 4 książek i 4 podręczników, autor i współautor 98 artykułów i referatów naukowych
Obszary działalności naukowej	Logika rozmyta Systemy wbudowane Technika mikroprocesorowa Sieci przemysłowe automatyki Diagnostyka procesów przemysłowych Inteligentne elementy pomiarowe i wykonawcze automatyki
Doświadczenie inżynierskie	5 patentów, konstruktor 58 unikalnych urządzeń mechatronicznych, 12 wdrożeń przemysłowych
Znajomość języków	angielski, niemiecki, rosyjski
Hobby	narciarstwo zjazdowe, pływanie

Zakres tematyczny podręcznika

1. Wstęp

1.1. Krótki esej o wymiarze semantycznym

Nad wieloma problemami technicznymi przechodzimy często do porządku dziennego nie zastanawiając się głębiej nad ich znaczeniem semantycznym, zakładając nie bez racji, iż ważniejsze jest postawienie i rozwiązanie problemu, niż jego nazwanie i próba jego precyzyjnego zdefiniowania. Z całą pewnością do pojęć niedookreślonych zaliczyć można **systemy czasu rzeczywistego.** Spróbujmy zatem na samym początku, nie wnikając w szczegóły i meritum tego wszystkiego, co pod tym określeniem się kryje, dokonać prostego rozbioru semantycznego tego pojęcia.

Pojęcie pierwotne **system** wywodzi się z języka greckiego (*σιστημα*) i określa rzecz złożoną. Traktując rzecz swobodnie możemy sformułować pewną prowokacyjną hipotezę, która twierdzi, że: jesteśmy otoczeni wyłącznie przez systemy, ponieważ rzeczy prostych w zasadzie nie ma, a jeżeli są, to nie jesteśmy do końca przekonani lub nie mamy wystarczającej wiedzy, czy tak jest w rzeczywistości. Pojęcie system stosowane jest powszechnie zarówno w naukach technicznych i humanistycznych i zawiera pewne barwy i odcienie znaczeniowe zależne od tego czy stosowane jest do określenia systemów technicznych, informatycznych cybernetycznych, filozoficznych, psychologicznych, społecznych, czy innych. Znaczeniowo, pojęcie system akcentuje cechę złożoności rzeczy w sensie strukturalnym i relacji pomiędzy elementami struktury systemu. Doprecyzowanie pojęcia **system** wymaga zatem co najmniej określenia jego **struktury** (elementów składowych) jak również **relacji** między tymi elementami. Elementami systemu mogą być zarówno składniki materialne (rzeczywiste) jak i niematerialne (nierzeczywiste). Dla przykładu, istotnymi elementami systemów technicznych (np. systemów automatycznego sterowania procesów wytwórczych) są obiekty materialne. Całkowicie wirtualne elementy konstytuują z kolei systemy psychologiczne czy filozoficzne. Co do zasady, struktura systemu może mieć charakter niehomogeniczny, to znaczy zawierać zarówno elementy materialne jak i niematerialne. Co więcej, systemy takie należy uznać obecnie za reprezentatywne w zastosowaniach technicznych. Typowymi przykładami takich systemów są np. systemy komputerowe, systemy sterowania i monitorowania procesów, systemy komunikacyjne, inteligentne systemy pomiarowe i wykonawcze, itd., w których elementy sprzętowe (materialne) współdziałają (pozostają w relacjach) z elementami programowymi (niematerialnymi). Systemy mieszane, a więc o strukturze niejednorodnej, nazywane bywają niekiedy **systemami hybrydowymi**.

Pojęcie **system czasu rzeczywistego** w świetle przedstawionej wyżej definicji i interpretacji pojęcia systemu jest nieprecyzyjne i niejasne. Nie określa bowiem w sposób przejrzysty struktury takiego systemu i relacji pomiędzy jego elementami. Należy się jedynie domyślać, że pojęcie to odnosi się do relacji pomiędzy bliżej nieokreślonymi elementami systemu zachodzącymi na bieżąco (w czasie rzeczywistym). Dotyczy, więc takich systemów, w których zachodzą co najmniej uwarunkowane czasowo relacje pomiędzy ich elementami. Z tego powodu za właściwsze określenie tych systemów należałoby uznać **systemy z uwarunkowaniami czasowymi.**

Pojęcie i nazwa systemu czasu rzeczywistego wywodzi się z języka angielskiego (**R**eal **T**ime **S**ystem - RTS*)*. Mimo niejednoznaczności, termin ten ugruntował się i jest stosowany i akceptowany powszechnie w technice. Z tego powodu, będzie on stosowany w dalszym ciągu tego podręcznika. Istnieje wiele definicji systemu czasu rzeczywistego.

Definicja 1. System czasu rzeczywistego jest systemem „*..., którego wynik przetwarzania zależy nie tylko od jego logicznej poprawności, ale także od chwili, w której taki wynik się pojawi*" [1]. Ta definicja odwołuje się zatem do podstawowego paradygmatu informatyki jako dziedziny zajmującej się przetwarzaniem informacji.

Definicja 2. System czasu rzeczywistego to: *„... system komputerowy, w którym obliczenia, przeprowadzane równolegle z przebiegiem zewnętrznego procesu, mają na celu nadzorowanie, sterowanie lub terminowe reagowanie na zachodzące w procesie zdarzenia*" [2,3]. Ta definicja jest nieco bardziej szczegółowa i odwołuje się do zastosowań informatyki, w tym zwłaszcza tych, które leżą w domenie informatyki przemysłowej.

Wspólną cechą obu wyżej wymienionych definicji jest położenie akcentu na uwarunkowania czasowe występujące w tych systemach. Definicja 2 jest nieco szersza bowiem odwołuje się i zwraca uwagę na **równoległość** przebiegu procesu przetwarzania informacji i procesu zewnętrznego z podkreśleniem wzajemnych interakcji, uwarunkowanych czasowo lub zdarzeniowo. Ze względu na rozwój systemów czasu rzeczywistego, w tym zwłaszcza **systemów zdecentralizowanych o strukturze rozproszonej**, definicję 2 należy uznać za nieaktualną. W związku z tym na potrzeby tego podręcznika wprowadzimy następująca prostą ale jednocześnie wystarczająco ogólną definicję systemu czasu rzeczywistego:

Definicja 3. System czasu rzeczywistego to system, w którym pomiędzy jego elementami i procesami w nim występującymi, zachodzą relacje uwarunkowane czasowo lub zdarzeniowo.

Definicja ta abstrahuje od wymiaru praktycznej realizacji systemu. System czasu rzeczywistego nie może bowiem być postrzegany jedynie jako określona realizacja konkretnego systemu technicznego. W tym sensie obliczenia mogą być prowadzone w systemie w sposób rozproszony i niekoniecznie przez system komputerowy jak w definicji 2.

Na przykład, obliczenia mogą być prowadzone przez systemy komputerowe, systemy sterowania zdecentralizowanego DDC, sterowniki programowalne PLC, programowalne sterowniki automatyki PAC, urządzenia inteligentne, systemy wbudowane, itp.

Istotne w tym przypadku jest podkreślenie nie tyle realizacji technicznej systemu czasu rzeczywistego, ale warunków realizacji jego zadań.

1.2. Co zawiera ten podręcznik?

Podręcznik, który oddajemy do waszych rąk za pośrednictwem środków komunikacji elektronicznej ma na celu przede wszystkim służyć dla was pomocą w zdobyciu podstawowej wiedzy z zakresu teorii i praktyki systemów czasu rzeczywistego. W podręczniku tym skupimy raczej uwagę na mechanizmach działania systemów niż na ich szczegółowych realizacjach. Podręcznik należy więc traktować jako pewnego rodzaju kompendium wiedzy z zakresu systemów czasu rzeczywistego. Podręcznik jest adresowany do każdego inżyniera automatyka i informatyka. Wybór i zakres tematów zawartych w podręczniku został ograniczony ze względów objętościowych, a także ze względu na zachowanie prostoty i jasności wykładu.

Powstaje pytanie, czy po lekturze podręcznika czytelnik będzie w stanie samodzielnie zaprojektować i zrealizować system czasu rzeczywistego przeznaczony np. do sterowania: ruchem ulicznym, procesem rozlewania napojów gazowanych, procesem montażu karoserii w fabryce samochodów, procesem produkcji cukrzycy w cukrowni, procesem wytwarzania pary w kotle energetycznym, procesem sterowania kolumną rektyfikacyjną itp.?

Na wyżej postawione pytanie należy odpowiedzieć rzetelnie, że z całą pewnością nie. Wiedza zawarta w podręczniku jest wiedzą zbyt skromną. Wyżej wymienione przykładowe zadania wymagają szczegółowej wiedzy także z innych dziedzin nauki i techniki, a więc miedzy innymi z zakresu technologii procesów, budowy instalacji technologicznych, urządzeń pomiarowych i wykonawczych, dynamiki procesów, automatyki oraz informatyki. Konieczne jest również odpowiednie doświadczenie i dostateczna wiedza szczegółowa. Tego typu odpowiedzialne zadania rozwiązywane są zawsze zespołowo.

Co zatem ten podręcznik ma do zaoferowania? Podręcznik ma za zadanie przedstawić niezbędną wiedzę w takim zakresie, że pozwoli to czytelnikowi na wskazanie i uzasadnienie celowości zastosowania określonych systemów czasu rzeczywistego do realizacji określonych zadań projektowych. Po starannej lekturze podręcznika, czytelnik powinien mieć świadomość istniejących ograniczeń i znajomość pól aplikacyjnych dla przemysłowych systemów czasu rzeczywistego. Pozwala to na włączenie się do realizacji projektów przemysłowych już na etapie specyfikacji. Wiedza zawarta w podręczniku pozwoli dokonać ogólnej analizy procesu pod kątem wymagań, które należy postawić systemowi czasu rzeczywistego, który ten proces będzie obsługiwał.

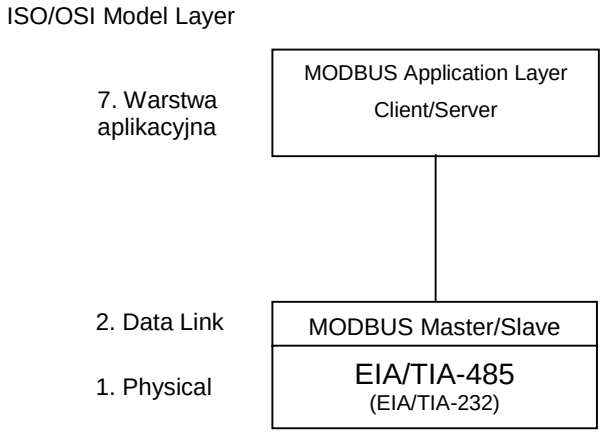
W podręczniku przedstawiono zarówno charakterystykę systemów operacyjnych czasu rzeczywistego (QNX) jak również przemysłowych sieciowych systemów czasu rzeczywistego (HART, Profibus PA, Profibus DP, Modbus RTU Foundation Fieldbus, AS-i, LonWorks, CAN).

Czytelników pragnących pogłębić wiadomości z zakresu systemów czasu rzeczywistego odsyłamy do literatury. Literatura na temat systemów operacyjnych czasu rzeczywistego jest ogólnie dostępna np. [1, 2]. Literatura na temat systemów przemysłowych sieciowych czasu rzeczywistego jest bardziej rozproszona. Materiały źródłowe dotyczące tych systemów są w znacznej mierze niedostępne w domenie publicznej jakkolwiek z nielicznymi wyjątkami np. [4].

5. Lekcja 4: Modbus

5.1. Rys historyczny

Tak naprawdę Modbus był pierwszym, powszechnie zaakceptowanym przemysłowym systemem komunikacji sieciowej. Jego historia sięga późnych lat siedemdziesiątych poprzedniego wieku. W roku 1979 producent sterowników PLC firma Modicon opublikowała protokół komunikacyjny przeznaczony dla sieci wieloelementowych, opartych na architekturze typu master/slave. Komunikacja pomiędzy urządzeniami odbywała się za pomocą wiadomości. Warstwa fizyczna interfejsu Modbus była dowolna. Pierwotna postać protokołu wykorzystywała komunikację szeregową RS-232, lecz kolejne implementacje używały RS-485 ze względu na większe dopuszczalne odległości, wyższe prędkości transmisji oraz większą liczbę urządzeń w systemie. W krótkim czasie, pomimo tego, że protokół nie był promowany przez żadne konsorcjum lub organizację standaryzacyjną, setki wytwórców zaczęło stosować Modbus w swoich urządzeniach. Stał się on de facto standardem dla sieci przemysłowych.

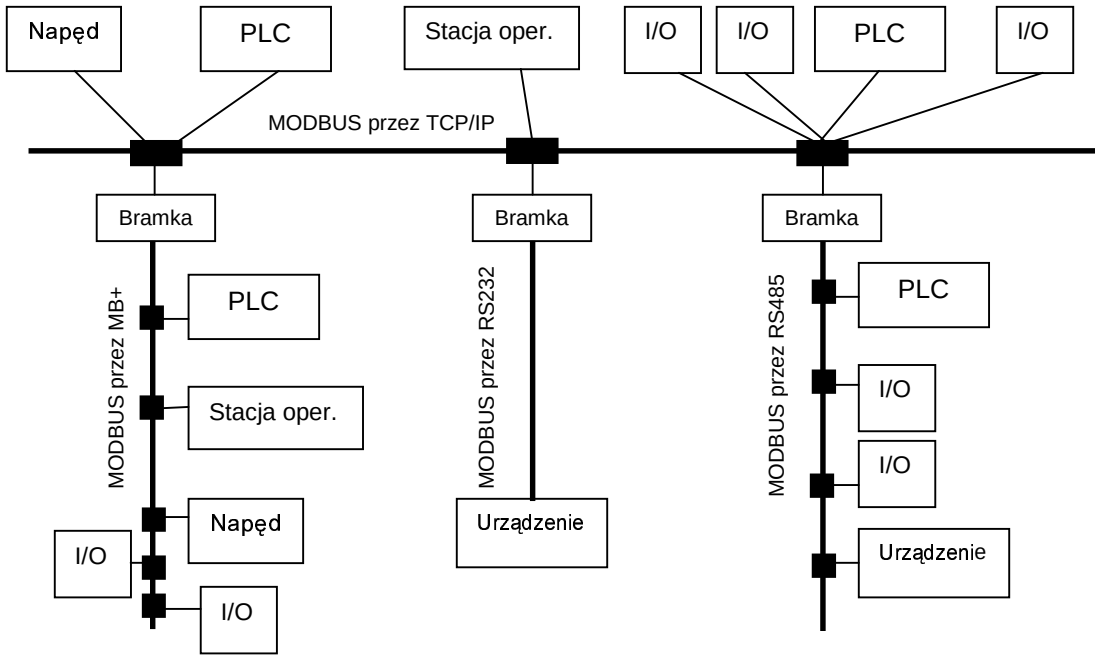


Modbus ściśle definiuje strukturę wiadomości, która umożliwia komunikację między urządzeniami. Określa więc sposób adresowania jednostek, interpretację i przetwarzanie wiadomości, ewentualnie proces konstruowania odpowiedzi. Protokół opisuje też wykrywanie i sygnalizację błędów, tych wynikających z transmisji fizycznej, jak i tych, których skutkiem jest błędna składnia wiadomości.

Modbus jest protokołem prostym, łatwym w implementacji a jednocześnie bardzo elastycznym. Nie tylko urządzenia takiego typu, jak sterowniki PLC, mogą komunikować się używając protokołu Modbus. Także wiele inteligentnych przetworników jest w stanie wysyłać dane do jednostek nadrzędnych.

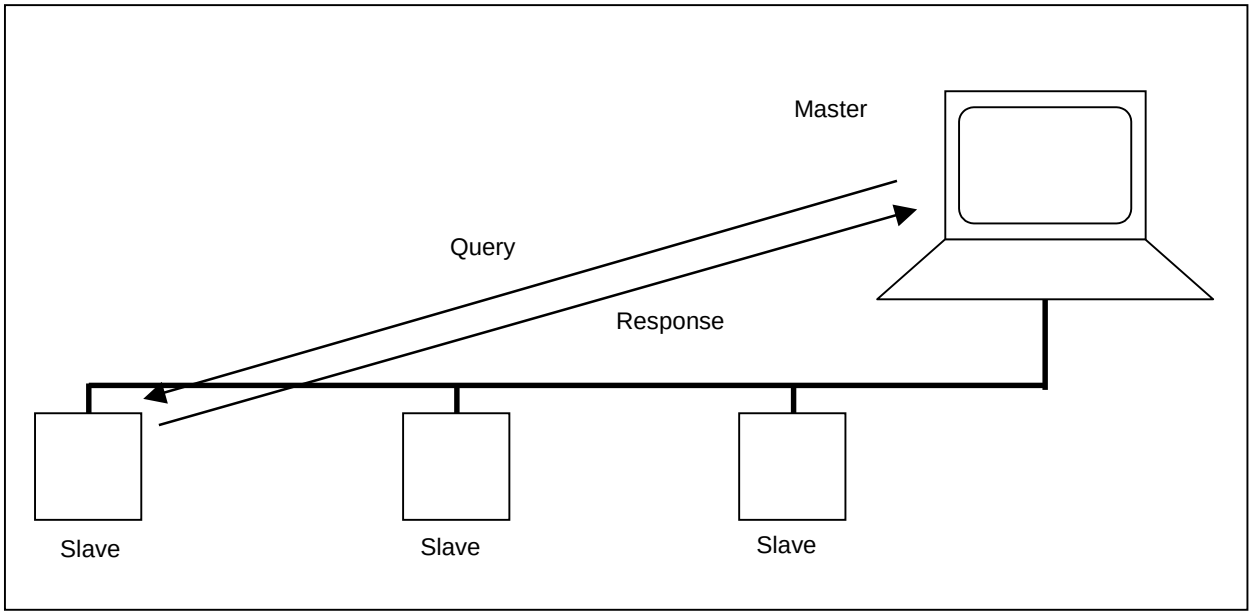
Pomimo tego, że Modbus jest używany głównie w tradycyjnej asynchronicznej

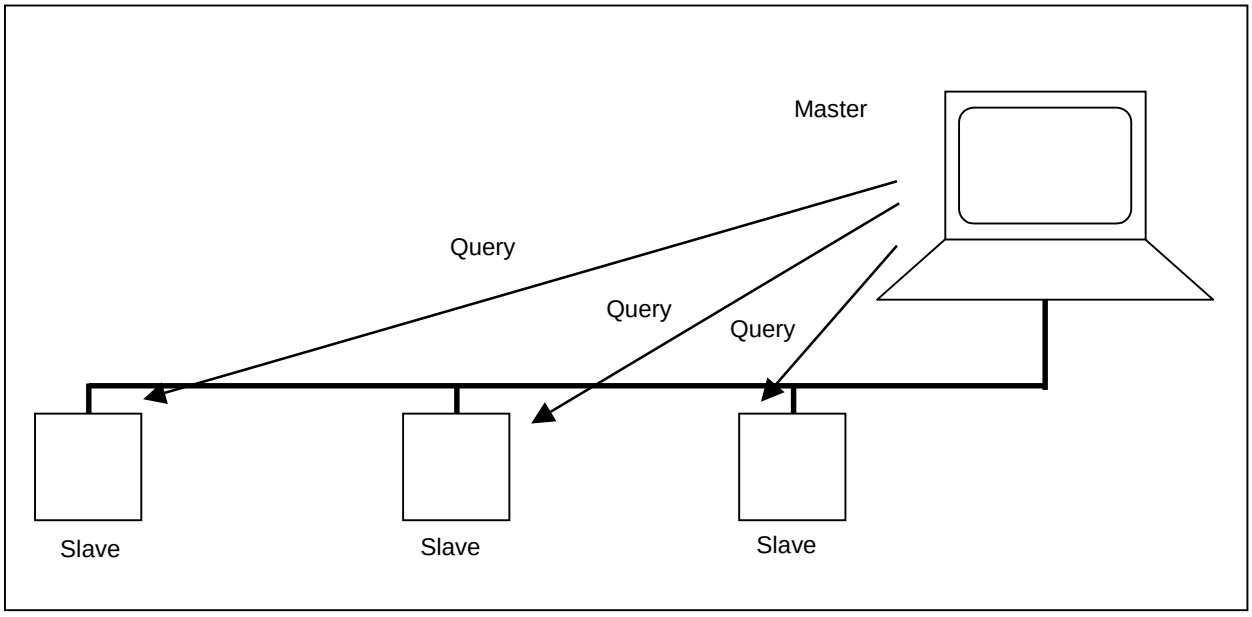
transmisji szeregowej, istnieją także rozszerzenia do standardu pozwalające na komunikację bezprzewodową oraz TCP/IP przez Ethernet. Istnieje również Modbus Plus – sieć o wysokich prędkościach transmisji, oparta na przekazywaniu znacznika (token). Szeroka gama możliwych warstw fizycznych i analogiczna konstrukcja wiadomości w każdym przypadku (część zwana PDU – Protocol Data Unit – jest niezmienna), pozwalają na tworzenie sieci hierarchicznych.



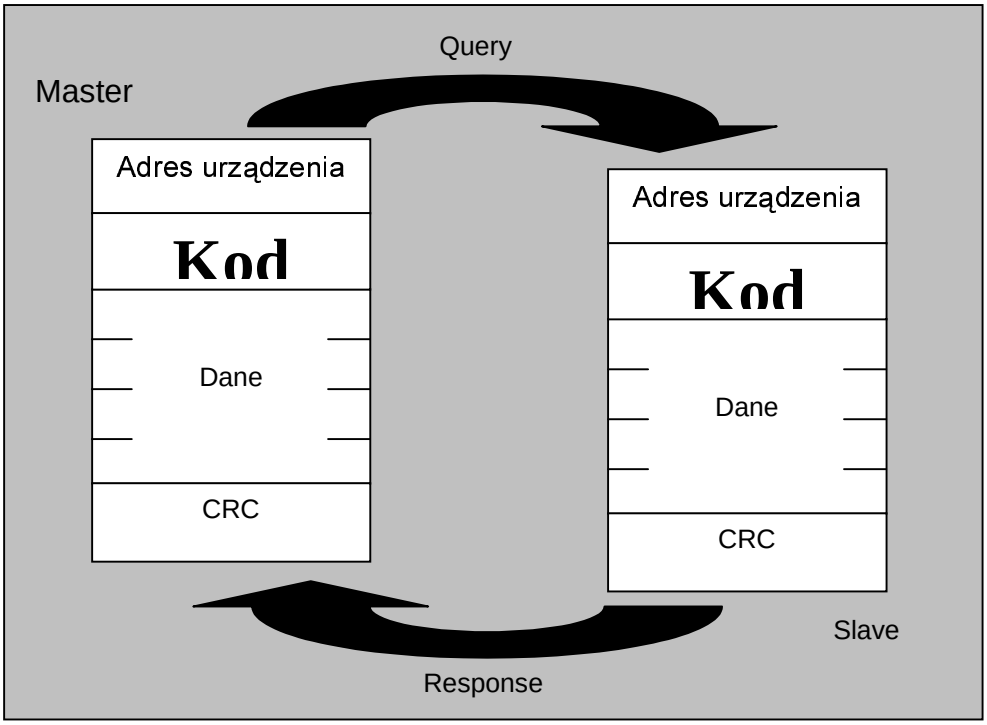
5.2. Wymiana wiadomości pomiędzy jednostkami

Urządzenia wykorzystujące interfejs RS-232C komunikują się używając techniki master-slave, w której tylko jedna jednostka (master) może zainicjować wymianę wiadomości, wysyłając zapytanie (query). Inne urządzenia (slave) odpowiadają przekazując wymagane dane do jednostki nadrzędnej lub podejmując określone w zapytaniu działania. Master może adresować pojedyncze jednostki podrzędne lub wysłać jedną wiadomość (broadcast) do wszystkich urządzeń. Jednostki podrzędne odpowiadają wiadomością (response) na zapytania, które zostały zaadresowane indywidualnie do nich. Odpowiedzi na zapytania typu broadcast nie są generowane.





Protokół Modbus określa format zapytania. Wiadomość ta składa się więc z adresu jednostki podrzędnej (lub broadcast), kodu funkcji, danych do wysłania, i pola kontroli błędów (CRC). Odpowiedź jednostki podrzędnej jest także konstruowana według protokołu Modbus. Zawiera pole potwierdzające wykonanie wymaganych czynności, zwracane dane, i pole kontroli błędów. Jeśli w strukturze wiadomości pojawi się błąd, lub jeśli urządzenie nie jest w stanie wykonać zażądanych działań, slave wygeneruje odpowiedź sygnalizującą usterkę (exception response).

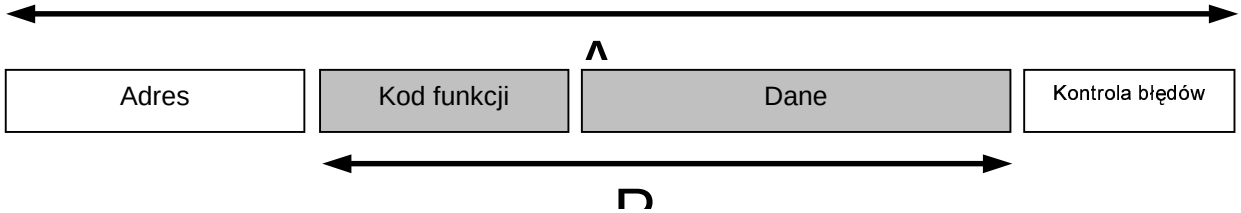


Query: Kod funkcji w zapytaniu wskazuje zaadresowanej jednostce podrzędnej jakie czynności wykonać. Dane (podzielone na bajty) zawierają wszystkie dodatkowe informacje, których slave będzie potrzebował do zrealizowania polecenia. Pole kontroli błędów pozwala na sprawdzenie poprawności składni wiadomości.

Response: Jeśli jednostka podrzędna konstruuje normalną odpowiedź, to kod funkcji jest taki sam jak w zapytaniu. Dane zawierają informacje takie jak zawartości rejestrów urządzenia slave. W przypadku pojawienia się błędu, kod funkcji jest modyfikowany, aby zakomunikować, że odpowiedź jest odpowiedzią wyjątkową (exception response) a dane zawierają kod wskazujący na rodzaj błędu. Pole kontroli błędów pozwala jednostce nadrzędnej na weryfikację zawartości wiadomości.

Urządzenia używające innych sieci (Ethernet, MB+) komunikują się wykorzystując technikę peer-to-peer (kolega do kolegi), w której każda jednostka może zainicjować transakcję z inną, dowolną, równorzędną jednostką. Na poziomie wiadomości protokół nadal spełnia reguły master-slave. Jeśli urządzenie wysyła zapytanie, robi to jak jednostka nadrzędna, i oczekuje odpowiedzi od jednostki podrzędnej. Podobnie, jeśli urządzenie odbierze zapytanie, to wygeneruje odpowiedź i wyśle ją do jednostki nadrzędnej.

W ogólnym przypadku protokół Modbus definiuje podstawową jednostkę danych PDU (Protocol Data Unit). Dodatkowe pola w wiadomości zależne są od typu sieci i dopełniają PDU do ADU (Application Data Unit).



Rozmiar PDU jest ograniczony przez maksymalną liczbę bajtów, wynikającą z pierwszej implementacji protokołu przy użyciu transmisji szeregowej RS485.
Dla RS485 maksymalny rozmiar ADU: 256 bajtów.
MODBUS PDU = 256 - Adres (1 bajt) - CRC (2 bajty) = 253 bajty.

5.3 Dwa rodzaje transmisji szeregowej

Urządzenia wykorzystujące do komunikacji standardowe sieci szeregowe RS-232C i RS-485, mogą używać dwóch trybów transmisji: ASCII i RTU. Tryb definiuje zawartość poszczególnych pól przesyłanych wiadomości. Określa sposób upakowania informacji w każdym bajcie ramki. Mówi też, jak informacje te będą dekodowane. Tryb razem z parametrami portu komunikacyjnego (baud rate, parity, itd.) musi być jednakowy dla wszystkich elementów sieci.

Tryb ASCII

Każdy bajt wiadomości (8 bitów) jest przesyłany jako dwa znaki ASCII (American Standard Code for Information Interchange). Główną zaletą tego trybu jest to, że dozwolone są długie (aż do 1[s]), nie powodujące błędu, przerwy transmisji między znakami.

Format bajtu (ramki transmisji szeregowej) w trybie ASCII:

System kodowania: Heksadecymalny, znaki ASCII: 0–9, A–F
Jeden znak heksadecymalny w każdej ramce transmisji szeregowej
(np. bajt 9A zostanie przesłany w dwóch ramkach jako dwa znaki 9 i A)

Bity w Bajcie: 1 bit startu
7 bitów danych, najmniej znaczący bit wysyłany jako pierwszy
1 bit jeśli even/odd parity; 0 bitów jeśli no parity
1 bit stopu jeśli włączona jest kontrola parzystości; jeśli nie: 2 bity stopu
Kontrola błędów: Longitudinal Redundancy Check (LRC)

Tryb RTU

W trybie RTU (Remote Terminal Unit) każdy bajt wiadomości (8-bitów) jest przesyłany w jednej ramce transmisji szeregowej. Za główną zaletę tego trybu należy uznać gęstsze upakowanie danych oraz szybszą transmisję niż w przypadku trybu ASCII (przy tym samym baud rate). W przypadku RTU cała wiadomość musi zostać wysłana jako jeden ciągły strumień.

Format bajtu (ramki transmisji szeregowej) w trybie RTU:

System kodowania: binarny 8–bitowy
Bity w Bajcie: 1 bit startu
8 bitów danych, najmniej znaczący bit wysyłany jako pierwszy
1 bit jeśli even/odd parity; 0 bitów jeśli no parity
1 bit stopu jeśli włączona jest kontrola parzystości; jeśli nie: 2 bity stopu
Kontrola błędów: Cyclical Redundancy Check (CRC)

5.4. Struktura ramki

W przypadku obu trybów transmisji (ASCII i RTU), urządzenie nadające umieszcza wiadomość Modbus w ramce, która ma określony początek i koniec. To pozwala urządzeniom odbierającym na: rozpoczęcie odczytu na początku wiadomości, zinterpretowanie adresu jednostki (lub jednostek w przypadku trybu broadcast) i wykrycie końca ramki.

Ramka w trybie ASCII

W trybie ASCII ramka rozpoczyna się od znaku dwukropka ‘:’ (ASCII 3A hex), a kończy parą znaków ‘carriage return – line feed’ (CRLF) (ASCII 0D i 0A hex). Dla innych pól, dozwolonymi do transmisji znakami są heksadecymalne 0–9, A–F. Wszystkie urządzenia stale monitorują sieć, oczekując na wystąpienie znaku ‘:’. Kiedy zostanie on odebrany, każda jednostka dekoduje następne pole (pole adresowe), aby dowiedzieć się, czy wiadomość została zaadresowana do niej.

START	ADRES	KOD FUNKCJI	DANE	LRC	KONIEC
1 znak ‘:’	2 znaki	2 znaki	N znaków	2 znaki	2 znaki CRLF

Ramka w trybie RTU

W trybie RTU ramka rozpoczyna się ciszą (przerwą w transmisji), trwającą co najmniej 3,5 raza dłużej niż czas przesłania pojedynczego znaku. Następnym wysyłanym polem jest adres urządzenia. Na jego pojawienie się oczekują wszystkie jednostki i stale monitorują sieć. Gdy już pole z adresem zostanie odebrane, każde urządzenie ustala, czy jest adresatem wiadomości. Koniec wiadomości jest oznaczany taką samą przerwą (3,5 x czas przesłania pojedynczego znaku). Nowa wiadomość może się rozpocząć dopiero po tej ciszy. Cała wiadomość musi być przesłana jako jeden ciągły strumień. Zostaje ona uznana za niekompletną, jeśli w trakcie odbioru wystąpi przerwa dłuższa niż 1,5 x czas przesłania pojedynczego znaku.

START	ADRES	KOD FUNKCJI	DANE	CRC	KONIEC
T1-T2-T3-T4	8 bitów	8 bitów	n x 8 bitów	16 bitów	T1-T2-T3-T4

5.4.1 Adres

Pole adresowe ramki składa się z dwóch znaków (ASCII) lub 8 bitów (RTU). Poprawne adresy zawierają się w przedziale 0 – 247 (decymalnie). Adresy indywidualne mogą przyjmować wartości 1 – 247. Adres 0 jest zarezerwowany dla trybu broadcast.
Jednostka nadrzędna adresuje wiadomość, umieszczając adres jednostki podrzędnej w polu adresowym. Gdy slave konstruuje odpowiedź, umieszcza swój własny adres w polu adresowym wiadomości response.

5.4.2 Kod funkcji

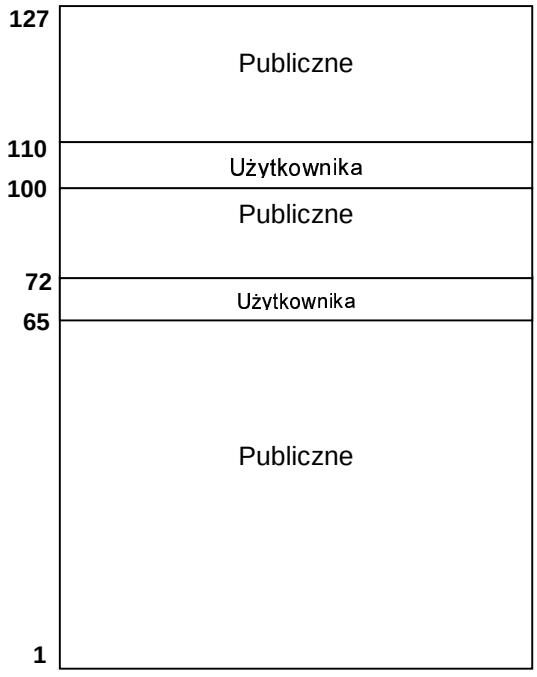
Kod funkcji składa się z dwóch znaków (ASCII) lub 8 bitów (RTU). Poprawne kody funkcji zawierają się w przedziale 1 – 255 (decymalnie). Należy zaznaczyć, że zestaw funkcji może być inny dla różnych urządzeń.

Istnieją trzy rodzaje kodów funkcji MODBUS:
Kody publiczne:
• zdefiniowane,
• niepowtarzalne,
• zatwierdzone przez społeczność MODBUS-IDA.org,
• udokumentowane,
• zgodne dla wszystkich urządzeń.
Kody użytkownika:
• zawierają się w dwóch przedziałach: 65-72, 100-110 decymalnie,
• użytkownik może wybrać i zaimplementować własną funkcję o kodzie, którego nie ma w specyfikacjach MODBUS,
• nie ma gwarancji, że kod funkcji został użyty w niepowtarzalny sposób,
• użytkownik może starać się o włączenie funkcji do grupy funkcji publicznych,
Kody zastrzeżone:
• kody używane przez konkretne firmy i niedostępne publicznie,

Kod funkcji, umieszczony w zapytaniu, mówi jednostce podrzędnej, jakie czynności należy wykonać. Jeśli wszystkie operacje zakończą się pomyślnie, wygenerowana odpowiedź zawierać będzie niezmieniony kod funkcji. Jeśli natomiast z jakichś powodów wystąpi błąd, jednostka podrzędna skonstruuje odpowiedź wyjątkową, używając kodu funkcji z ustawionym w stan 1 najbardziej znaczącym bitem.

Na przykład, wiadomość z jednostki nadrzędnej, nakazująca odczyt zawartości wielu rejestrów, będzie miała kod:
0000 0011 (Heksadecymalnie 03)
Jeśli nie wystąpią błędy, w odpowiedzi zostanie zwrócony taki sam kod. W przeciwnym razie kod w odpowiedzi wyjątkowej przyjmie postać:
1000 0011 (Heksadecymalnie 83)

Można więc zauważyć, że kody funkcji z przedziału 128-255 są zarezerwowane dla odpowiedzi wyjątkowych.
Niektóre funkcje wymagają także kodów dodatkowych (sub-function codes).



5.4.3 Pole danych

Pole danych składa się z par znaków heksadecymalnych z zakresu 00 – FF. Pary te

mogą być utworzone z dwóch znaków ASCII lub jednego znaku RTU, zależnie od trybu transmisji.

Pole danych wiadomości, wysyłanej przez jednostkę nadrzędną, zawiera dodatkowe informacje niezbędne jednostce podrzędnej do wykonania operacji wskazanych przez kod funkcji. Dane mogą zatem obejmować adresy rejestrów i pól jednobitowych, liczność elementów oraz liczbę bajtów zawartych w polu.

Na przykład, jeśli master zażąda, aby slave odczytał grupę rejestrów (kod funkcji 03), pole danych określa rejestr początkowy i liczbę elementów, które mają być odczytane. Jeśli jednostka nadrzędna dokonuje zapisu grupy rejestrów w urządzeniu slave (kod funkcji 10 heksadecymalnie), pole danych zawiera adres pierwszego rejestru, liczbę rejestrów, liczbę bajtów, które znajdują się jeszcze w tym polu oraz dane, które mają być wpisane do rejestrów.

Jeśli transakcja przebiega pomyślnie, pole danych odpowiedzi jednostki podrzędnej zawiera „zamówione” informacje. Jeśli wystąpi błąd, pole danych zawiera kod wyjątku.

W niektórych typach wiadomości pole danych może być puste (zerowej długości).

Model danych MODBUS

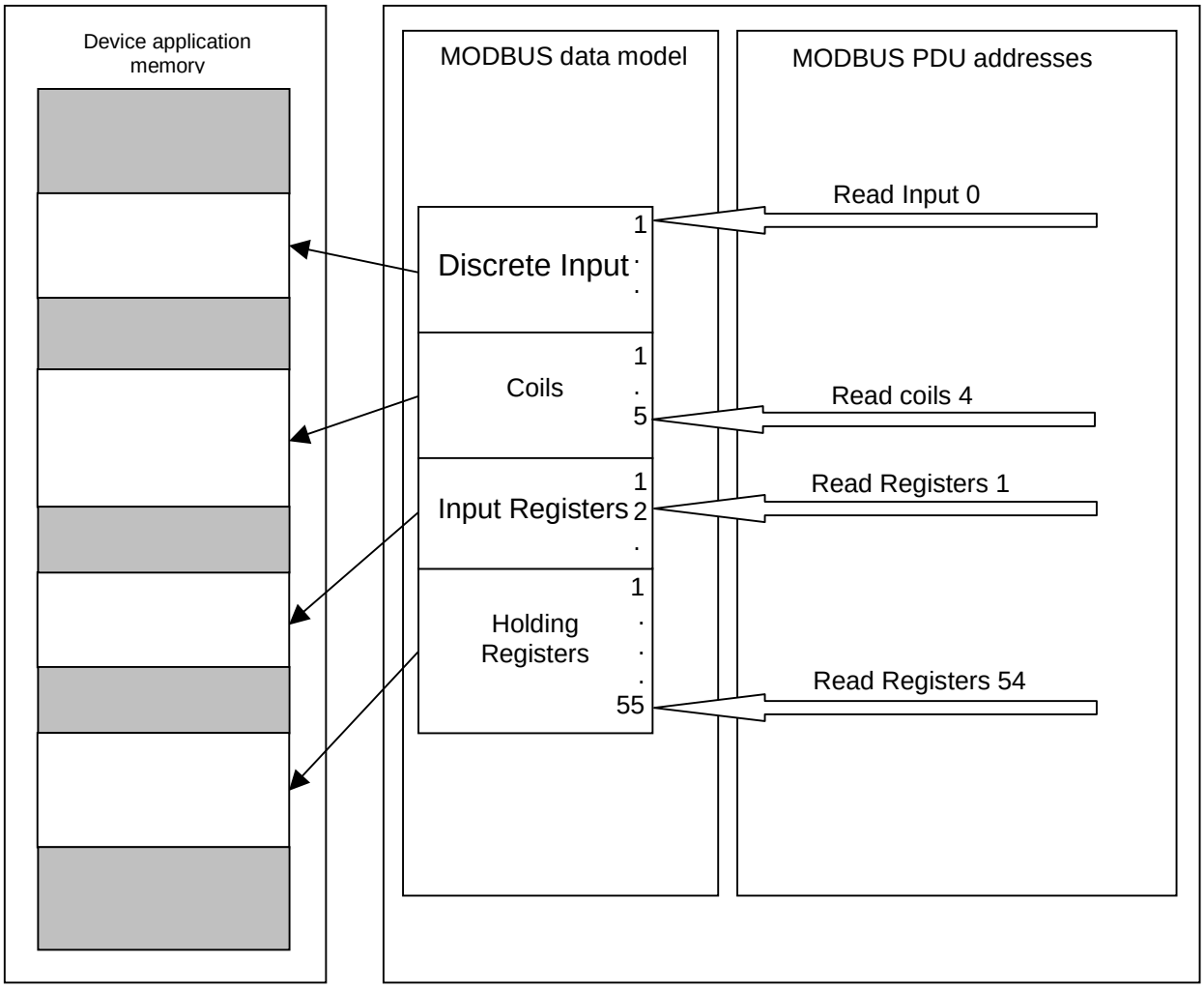
Model danych Modbus oparty jest na czterech podstawowych jednostkach:

Discretes Input	Single bit	Read-Only	Pole jednobitowe przeznaczone jedynie do odczytu. Modyfikacja nie może być dokonana za pomocą funkcji Modbus.
Coils	Single bit	Read-Write	Pole jednobitowe przeznaczone do zapisu i odczytu. Możliwa modyfikacja za pomocą funkcji Modbus.
Input Registers	16-bit-word	Read-Only	Rejestr szesnastobitowy przeznaczony jedynie do odczytu. Modyfikacja nie może być dokonana za pomocą funkcji Modbus.
Holding Registers	16-bit-word	Read-Write	Rejestr szesnastobitowy przeznaczony do zapisu i odczytu. Możliwa modyfikacja za pomocą funkcji Modbus.

Dla każdego spośród wymienionych czterech typów danych, protokół pozwala na wybór spośród 65536 (2^{16}) elementów.

Adresowanie w wiadomościach Modbus

Adresowanie w wiadomościach Modbus obywa się względem zera. Pierwszy element danego typu ma zatem adres zerowy.
Pole bitowe istniejące w urządzeniu pod adresem ‘1’, w wiadomości posiadać będzie adres 0000. Analogicznie pole bitowe 127 (decymalnie) jest adresowane jako 007E hex (126 decymalnie).
Rejestr 40001 jest adresowany w wiadomości jako 0000. Kod funkcji precyzuje, że elementem, na którym przeprowadzamy operacje, jest rejestr. Dlatego część ‘4XXXX’ nie jest potrzebna. Rejestr 40108 jest adresowany jako 006B hex (107 decymalnie).



Oczywisty jest fakt, że wszystkie rejestry i pola bitowe muszą znajdować się w pamięci fizycznej urządzenia. Adresy fizyczne nie powinny być jednak mylone z adresami związanymi z modelem danych (MODBUS data model).

Kodowanie danych

Modbus wykorzystuje system „big-Endian” do zapisu adresów i danych. Oznacza to, że jeśli przesyłana jest jednostka danych dłuższa niż jeden bajt, najbardziej znaczący bajt wysyłany jest jako pierwszy. Na przykład, szesnastobitowe słowo 0x1234 zostanie przesłane jako bajty kolejno: 0x12 i 0x34.

Pozostałe systemy kodowania (dla podwójnego słowa 0x10203040 – 32 bity; funkcje

użytkownika mogą pozwalać na operacje na takich rejestrach, zawierających np. liczby zmiennoprzecinkowe w standardzie IEEE 32-bit floating-point):

System kodowania	Kolejność bajtów
big-Endian	0x10 0x20 0x30 0x40
big-Endian with byte-swapped	0x20 0x10 0x40 0x30
little-Endian	0x40 0x30 0x20 0x10
little-Endian with byte-swapped	0x30 0x40 0x10 0x20

5.4.4 Pole kontroli błędów

Modbus wykorzystuje dwa niezależne rodzaje kontroli poprawności transmisji. Kontrola parzystości stosowana jest do sprawdzania pojedynczego znaku. Metoda kontroli całej ramki wiadomości (LRC lub CRC) zależna jest od trybu transmisji szeregowej.

Tryb ASCII

W trybie ASCII kontrola błędów oparta jest na metodzie LRC (Longitudinal Redundancy Check). Pole LRC stanowi liczba ośmiobitowa, przesyłana jako dwa znaki ASCII, w dwóch oddzielnych ramkach transmisji szeregowej. LRC weryfikuje poprawność wiadomości, a więc poza znakami początkowym (:) i końcowym (CRLF) ramki.
Wartość LRC jest obliczana i dołączana do wiadomości przez urządzenie nadające. Jednostka odbierająca także generuje LRC w trakcie odczytu ramki i porównuje z polem LRC odebranej wiadomości. Błąd zostaje wykryty, gdy obie wartości nie są identyczne.
Wartość LRC jest obliczana przez dodanie kolejnych bajtów wiadomości. Wszystkie przeniesienia są ignorowane. Na koniec zawartość jest dopełniana do dwójki. Procedura postępowania wygląda następująco:
1. Dodanie wszystkich bajtów wiadomości do ośmiobitowego pola 0x00.

2. Dopełnienie do jedynki.
3. Dodanie 0x01, aby dopełnić do dwóch.

Funkcja generująca LRC może wyglądać następująco:

```
unsigned char LRC(unsigned char *pWiadomosc, unsigned short LiczbaBajtow)
{
unsigned char LRC = 0x00 ;

while (LiczbaBajtow—)
LRC += *pWiadomosc++ ;

return (~LRC + 0x01) ;
}
```

Podczas transmisji szeregowej pole LRC jest przesyłane w postaci dwóch znaków ASCII. Znak zawierający cztery starsze bity jest wysyłany jako pierwszy.

Tryb RTU

W trybie RTU kontrola błędów oparta jest na metodzie CRC (Cyclical Redundancy Check). Pole CRC służy do weryfikacji całej wiadomości, niezależnie od kontroli parzystości każdego znaku.
Pole CRC zawiera liczbę binarną 16-bitową, obliczaną i dołączaną do wiadomości przez urządzenie nadające. Jednostka odbierająca także generuje CRC w trakcie odczytu ramki i porównuje z polem CRC odebranej wiadomości. Błąd zostaje wykryty, gdy obie wartości nie są identyczne.
Generacja CRC rozpoczyna się od załadowania do 16–bitowego rejestru liczby 0xFFFF. Do następnych operacji wykorzystywane są kolejne 8-bitowe bajty wiadomości (bity startu, stopu i parzystości nie są brane pod uwagę). Każdy bajt jest poddawany operacji wykluczającej sumy bitowej Xor z zawartością rejestru. Wynik jest przesuwany o jeden bit w kierunku najmniej znaczącego bitu (LSB), a w miejsce najbardziej znaczącego bitu (MSB) wpisywane jest zero. Następnie zależnie od zawartości LSB, rejestr jest poddawany operacji Xor z ustaloną liczbą (A001 hex) – jeśli LSB jest równe 1, lub pozostawiany bez zmian – jeśli LSB jest równe 0. Proces

ten jest powtarzany aż do 8 przesunięć. Następnie wszystkie działania są powtarzane dla pozostałych bajtów wiadomości. Końcowa wartość to CRC. Procedura postępowania wygląda następująco:

1. Załadowanie do 16–bitowego rejestru 0xFFFF.
2. Xor młodszego bajtu rejestru z bajtem wiadomości.
3. Przesunięcie rejestru o jeden bit w prawo i wstawienie 0 w miejsce MSB.
4. Jeśli LSB jest równe 0 – powtórzenie punktu 3. Jeśli LSB jest równe 1 – Xor rejestru z wartością A001 hex (1010 0000 0000 0001).
5. Powtórzenie punktów 3 i 4 aż do ośmiu przesunięć.
6. Powtórzenie punktów od 2 do 5 dla kolejnego bajtu wiadomości. Kontynuowanie aż do momentu, gdy wszystkie bajty wiadomości zostaną wykorzystane.
7. Wartość końcowa do CRC.
8. Przy dołączaniu CRC do wiadomości należy zamienić miejscami bajty (16-bit CRC wysyłane w taki sposób, że młodszy bajt jest wysyłany jako pierwszy).

Funkcja generująca CRC może wyglądać następująco:

```
unsigned short CRC(unsigned char *pMessage, unsigned int NumberOfBytes)
{
register unsigned short reg16 = 0xFFFF;
unsigned char reg8;
unsigned char i;

while (NumberOfBytes--)
{
                reg16 ^= *pMessage++;
i = 8;

while(i--)
{
if (reg16 & 0x0001)
{
reg16 >>= 1;
reg16 ^= 0xA001;
}
else
reg16 >>= 1;
}
};
reg8 = reg16 >> 8;
return (reg16 << 8 | reg8);
}
```

Powyższy algorytm okazuje się zbyt wolny w rzeczywistych zastosowaniach. Dlatego też wykorzystujemy funkcję opartą na dwóch tablicach, w których znajdują się wszystkie możliwe starsze i młodsze bajty CRC.

```
unsigned short CRC(unsigned char *pMessage, unsigned int NumberOfBytes)
{
static unsigned char aCRCHi[] =
{
0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1,
0x81,
0x40, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x00, 0xc1, 0x81, 0x40, 0x01,
0xc0,
0x80, 0x41, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x00, 0xc1, 0x81, 0x40,
0x01,
0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x01, 0xc0, 0x80,
0x41,
0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x00, 0xc1,
0x81,
0x40, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x01,
0xc0,
0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41,
0x01,
0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81,
0x40,
0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1,
0x81,
0x40, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x01,
0xc0,
0x80, 0x41, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x00, 0xc1, 0x81, 0x40,
0x01,
0xc0, 0x80, 0x41, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80,
0x41,
0x00, 0xc1, 0x81, 0x40, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1,
0x81,
0x40, 0x01, 0xc0, 0x80, 0x41, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x01,
0xc0,
0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41,
0x01,
0xc0, 0x80, 0x41, 0x00, 0xc1, 0x81, 0x40, 0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80,
0x41,
0x00, 0xc1, 0x81, 0x40, 0x01, 0xc0, 0x80, 0x41, 0x01, 0xc0, 0x80, 0x41, 0x00, 0xc1,
0x81,
0x40
};
```

```
static unsigned char aCRCLo[] =
{
0x00, 0xc0, 0xc1, 0x01, 0xc3, 0x03, 0x02, 0xc2, 0xc6, 0x06, 0x07, 0xc7, 0x05, 0xc5,
0xc4,
0x04, 0xcc, 0x0c, 0x0d, 0xcd, 0x0f, 0xcf, 0xce, 0x0e, 0x0a, 0xca, 0xcb, 0x0b, 0xc9,
0x09,
0x08, 0xc8, 0xd8, 0x18, 0x19, 0xd9, 0x1b, 0xdb, 0xda, 0x1a, 0x1e, 0xde, 0xdf, 0x1f,
0xdd,
0x1d, 0x1c, 0xdc, 0x14, 0xd4, 0xd5, 0x15, 0xd7, 0x17, 0x16, 0xd6, 0xd2, 0x12, 0x13,
0xd3,
0x11, 0xd1, 0xd0, 0x10, 0xf0, 0x30, 0x31, 0xf1, 0x33, 0xf3, 0xf2, 0x32, 0x36, 0xf6,
0xf7,
0x37, 0xf5, 0x35, 0x34, 0xf4, 0x3c, 0xfc, 0xfd, 0x3d, 0xff, 0x3f, 0x3e, 0xfe, 0xfa,
0x3a,
0x3b, 0xfb, 0x39, 0xf9, 0xf8, 0x38, 0x28, 0xe8, 0xe9, 0x29, 0xeb, 0x2b, 0x2a, 0xea,
0xee,
0x2e, 0x2f, 0xef, 0x2d, 0xed, 0xec, 0x2c, 0xe4, 0x24, 0x25, 0xe5, 0x27, 0xe7, 0xe6,
0x26,
0x22, 0xe2, 0xe3, 0x23, 0xe1, 0x21, 0x20, 0xe0, 0xa0, 0x60, 0x61, 0xa1, 0x63,
0xa3, 0xa2,
0x62, 0x66, 0xa6, 0xa7, 0x67, 0xa5, 0x65, 0x64, 0xa4, 0x6c, 0xac, 0xad, 0x6d, 0xaf,
0x6f,
0x6e, 0xae, 0xaa, 0x6a, 0x6b, 0xab, 0x69, 0xa9, 0xa8, 0x68, 0x78, 0xb8, 0xb9,
0x79, 0xbb,
0x7b, 0x7a, 0xba, 0xbe, 0x7e, 0x7f, 0xbf, 0x7d, 0xbd, 0xbc, 0x7c, 0xb4, 0x74, 0x75,
0xb5,
0x77, 0xb7, 0xb6, 0x76, 0x72, 0xb2, 0xb3, 0x73, 0xb1, 0x71, 0x70, 0xb0, 0x50,
0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54,
0x9c, 0x5c,
0x5d, 0x9d, 0x5f, 0x9f, 0x9e, 0x5e, 0x5a, 0x9a, 0x9b, 0x5b, 0x99, 0x59, 0x58, 0x98,
0x88,
0x48, 0x49, 0x89, 0x4b, 0x8b, 0x8a, 0x4a, 0x4e, 0x8e, 0x8f, 0x4f, 0x8d, 0x4d, 0x4c,
0x8c,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41,
0x81, 0x80,
0x40
};

unsigned char HiByte = 0xFF;
unsigned char LoByte = 0xFF;
unsigned char Index;
while (NumberOfBytes--)
{
Index  = HiByte ^ *pMessage++;
HiByte = LoByte ^ aCRCHi[Index];
LoByte = aCRCLo[Index];
};
return (HiByte << 8 | LoByte);
}
```

5.5. Odpowiedzi wyjątkowe

Po wysłaniu zapytania, jednostka nadrzędna oczekuje normalnej odpowiedzi. Oczywiście nie dotyczy to wiadomości typu broadcast. Istnieją cztery typy sytuacji, które mogą wystąpić po wysłaniu wiadomości przez urządzenie master.

- Jeśli slave odbierze wiadomość bez błędów transmisji i będzie w stanie przeprowadzić zażądane działania, wygeneruje normalną odpowiedź.
- Jeśli slave nie odbierze wiadomości z powodu błędów w komunikacji, żadna odpowiedź nie będzie zwrócona.
- Jeśli slave odbierze wiadomość, ale wykryje w niej błędy (parzystość, LRC lub CRC), żadna odpowiedź nie będzie zwrócona.
- Jeśli slave odbierze wiadomość bez błędów w transmisji, ale nie będzie w stanie zrealizować funkcji (np. odczytu nieistniejącego rejestru), zwróci odpowiedź wyjątkową, informującą rodzaju błędu.

Odpowiedź wyjątkowa posiada dwa pola, odróżniające ją od odpowiedzi normalnej.

Kod funkcji: W przypadku normalnej odpowiedzi, jednostka podrzędna wysyła taki sam kod funkcji, jaki odebrała w zapytaniu. Wszystkie kody funkcji mają zerowy najbardziej znaczący bit (MSB) (ich wartości są mniejsze od 80 hex). W odpowiedzi wyjątkowej, jednostka podrzędna wstawia 1 w miejsce MSB kodu funkcji (wartość większa o 80 hex od normalnego kodu). Pozwala to urządzeniu master na łatwe rozpoznanie odpowiedzi wyjątkowej i interpretację kodu wyjątku (w polu danych).

Pole danych: W przypadku normalnej odpowiedzi, w polu danych znajdują się wszystkie zażądane informacje. W odpowiedzi wyjątkowej, jednostka podrzędna wykorzystuje to pole do przesłania kodu wyjątku, określającego rodzaj błędu

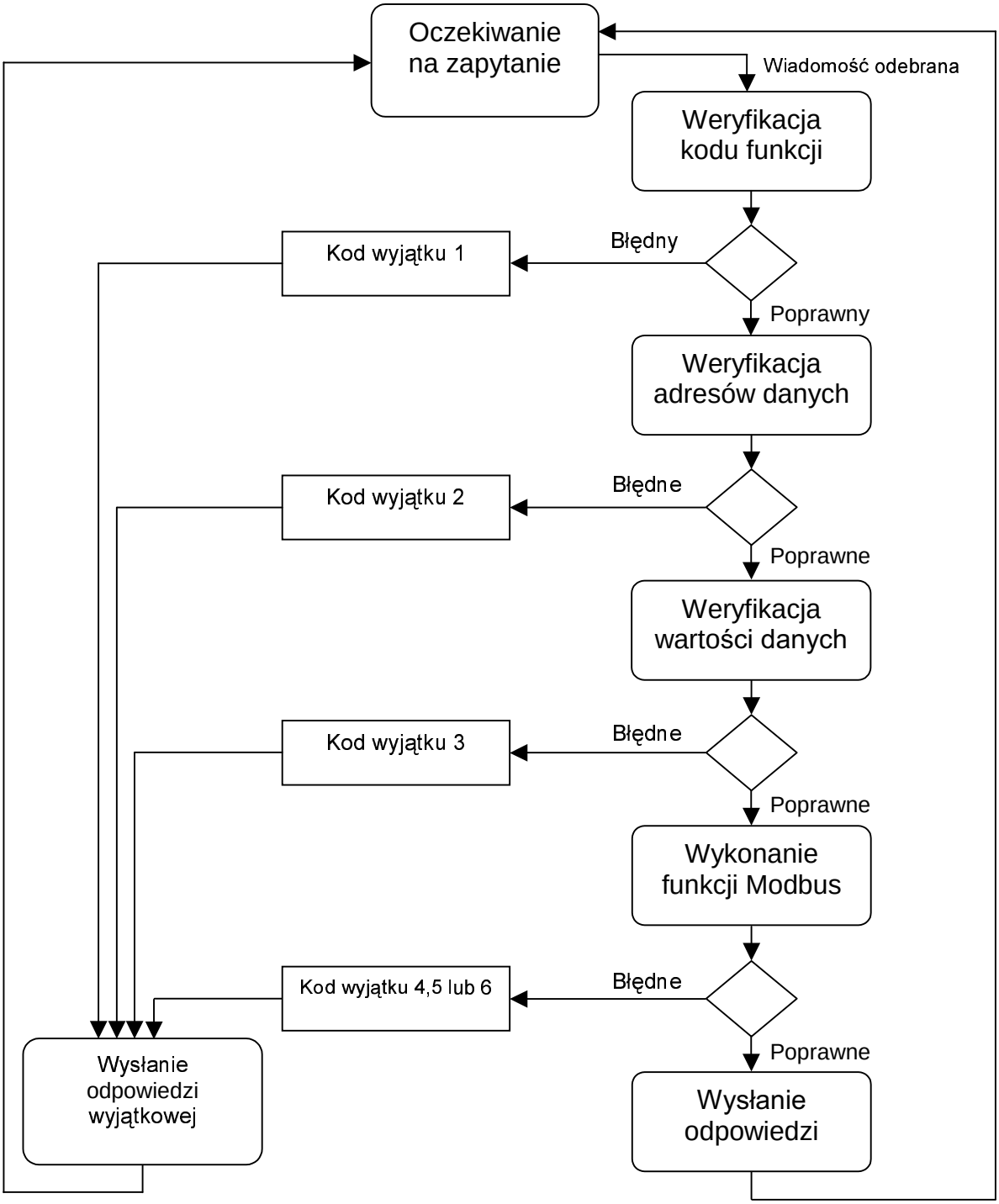
ADRES	KOD FUNKCJI	DANE	CRC lub LRC
	Normalny kod + 80 hex	Kod wyjątku	

Kody wyjątków (dla sieci standardowych):

Kod	Nazwa	Opis
01	Niedozwolona funkcja	Kod funkcji odebrany w zapytaniu nie jest obsługiwany przez jednostkę podrzędną.
02	Niedozwolony adres danych	Próba odczytu lub zapisu nieistniejącego rejestru lub pola bitowego.
03	Niedozwolona wartość danych	Próba wpisania do rejestru lub pola bitowego niewłaściwej wartości.
04	Niepowodzenie	Wystąpienie nieokreślonego błędu podczas próby wykonania operacji.
05	Potwierdzenie	Potwierdzenie przyjęcia i rozpoczęcia realizacji funkcji. Proces wymagać jednak będzie długiego czasu.
06	Jednostka podrzędna zajęta	Jednostka podrzędna jest zajęta wykonywaniem czasochłonnej funkcji. Zapytanie powinno zostać wysłane ponownie później.
08	Błąd parzystości pamięci	Błąd parzystości pamięci przy próbie odczytu.

5.6. Realizacja transakcji po stronie jednostki podrzędnej

Poniższy schemat przedstawia sposób realizacji transakcji po stronie jednostki podrzędnej Modbus. Jest to postępowanie w przypadku braku błędów transmisji. W przypadku ich wystąpienia nie zostanie wygenerowana żadna odpowiedź na zapytanie – ani normalna, ani wyjątkowa.


Oczekiwanie na zapytanie
Wiadomość odebrana
Weryfikacja kodu funkcji
Błędny
Kod wyjątku 1
Poprawny
Weryfikacja adresów danych
Błędne
Kod wyjątku 2
Poprawne
Weryfikacja wartości danych
Błędne
Kod wyjątku 3
Poprawne
Wykonanie funkcji Modbus
Błędne
Kod wyjątku 4,5 lub 6
Poprawne
Wysłanie odpowiedzi
Wysłanie odpowiedzi wyjątkowej

5.7. Komunikacja

Realizując komunikację pomiędzy rzeczywistymi urządzeniami należy zwrócić uwagę na następujące zagadnienia:

- skończony czas transmisji ramki przez jednostkę nadrzędną i odbiór przez jednostkę podrzędną,
- kluczowe dla przebiegu transmisji kolejnych wiadomości okresy ciszy,
- czas przeterminowania transakcji – jednostka nadrzędna nie może bezwzględnie oczekiwać na odpowiedź urządzenia slave, gdyż w przypadku błędu, może nie zostać wygenerowana żadna odpowiedź.

Czas przesłania pojedynczego bajtu (ramki transmisji szeregowej) można obliczyć następująco:

$$t_{byte} = \frac{1000 \cdot BPC}{BR} [ms]$$

, gdzie:

BPC – Bits Per Character – ilość bitów w pojedynczej ramce,

BR – Baud Rate – prędkość transmisji w $\left[\frac{baud}{s}\right]$.

Czas transmisji ramki Modbus (wraz z przerwą w nadawaniu):

$$t_{frame} = (n+4) \cdot t_{byte} = \frac{1000 \cdot (n+4) \cdot BPC}{BR} [ms]$$

, gdzie:

n – liczba bajtów w wiadomości Modbus.

Czas przeterminowania transakcji (timeout) zależy od parametrów konkretnego urządzenia slave. Jednostka podrzędna potrzebuje bowiem czasu na zinterpretowanie wiadomości, zrealizowanie funkcji oraz ewentualne wygenerowanie i wysłanie odpowiedzi. Oczywiście tempo realizacji zależy także od rodzaju funkcji i ilości danych, które mają być przetworzone.

5.8. Przykłady realizacji funkcji

Zestaw obsługiwanych funkcji zależy od typu urządzenia. Standardowa grupa rozkazów to:

- 01 Read Coil Status
- 02 Read Input Status
- 03 Read Holding Registers
- 04 Read Input Registers
- 05 Force Single Coil
- 06 Preset Single Register
- 07 Read Exception Status
- 11 (0B Hex) Fetch Comm Event Ctr
- 12 (0C Hex) Fetch Comm Event Log
- 15 (0F Hex) Force Multiple Coils
- 16 (10 Hex) Preset Multiple Regs
- 17 (11 Hex) Report Slave ID
- 20 (14 Hex) Read General Reference
- 21 (15 Hex) Write General Reference
- 22 (16 Hex) Mask Write 4X Register
- 23 (17 Hex) Read/Write 4X Registers
- 24 (18 Hex) Read FIFO Queue

5.8.1 Odczyt pola bitowego - 01 Read Coil Status

Funkcja służy do odczytu statusu pól bitowych (coils – discrete outputs) jednostki podrzędnej. Tryb broadcast nie jest możliwy.

Query

Wiadomość określa adres początkowy oraz liczbę elementów do odczytu. Pola są adresowane począwszy od zera (0-15: pola 1-16).

Przykład żądania odczytu pól 20-56 z urządzenia 17.

Nazwa pola	**Przykład (hex)**
Adres urządzenia	11
Funkcja	01
Adres startowy Hi	00
Adres startowy Lo	13
Liczba elementów Hi	00
Liczba elementów Lo	25
Kontrola błędów (LRC lub CRC)	-

Response

Stany pól w odpowiedzi są upakowane tak, że jeden bit danych odpowiada jednemu polu (1 = ON; 0 = OFF). Najmniej znaczący bit pierwszego bajtu zawiera status pola o adresie podanym w zapytaniu.

Jeśli liczba pól nie jest wielokrotnością ósemki, pozostałe bity w ostatnim bajcie zostaną wypełnione zerami.

Nazwa pola	**Przykład (hex)**
Adres urządzenia	11
Funkcja	01
Liczba bajtów	05
Dane (Coils 27-20)	CD
Dane (Coils 35-28)	6B
Dane (Coils 43-36)	B2
Dane (Coils 51-44)	0E
Dane (Coils 56-52)	1B
Kontrola błędów (LRC lub CRC)	-

Status pól 27-20 znajduje się w pierwszym bajcie danych CD hex lub 11001101 binarnie (ON-ON-OFF-OFF-ON-ON-OFF-ON). Pole 27 to najbardziej znaczący bit.

Zapisywanie stanu kolejnych pól o prawej do lewej sprawia, że są one przesyłane w normalnym porządku.

5.8.2 Odczyt rejestru – 03 Read Holding Registers

Funkcja służy do odczytu zawartości rejestrów (holding registers) jednostki podrzędnej. Tryb broadcast nie jest możliwy.

Query

Wiadomość określa adres początkowy oraz liczbę rejestrów do odczytu. Rejestry są adresowane począwszy od zera (0-15: rejestry 1-16).

Przykład żądania odczytu rejestrów 40108-40110 z urządzenia 17.

Nazwa pola	**Przykład (hex)**
Adres urządzenia	11
Funkcja	03
Adres startowy Hi	00
Adres startowy Lo	6B
Liczba elementów Hi	00
Liczba elementów Lo	03
Kontrola błędów (LRC lub CRC)	-

Response

Zawartość każdego rejestru jest zapisana w odpowiedzi w postaci dwóch bajtów. Pierwszy bajt zawiera bity wyższego rzędu, drugi niższego.

Nazwa pola	Przykład (hex)
Adres urządzenia	11
Funkcja	03
Liczba bajtów	06
Dane Hi (Rejestr 40108)	02
Dane Lo (Rejestr 40108)	2B
Dane Hi (Rejestr 40109)	00
Dane Lo (Rejestr 40109)	00
Dane Hi (Rejestr 40110)	00
Dane Lo (Rejestr 40110)	64
Kontrola błędów (LRC lub CRC)	-

Zawartość rejestrów 40108-40110 to kolejno: 022B hex (555 dec), 0000 hex (0 dec), 0064 hex (100 dec).

5.8.3. Zapis rejestrów - 16 (10 Hex) Preset Multiple Regs

Funkcja dokonuje zapisu grupy rejestrów (holding registers). W trybie broadcast następuje nadpisanie rejestru o tym samym adresie we wszystkich urządzeniach.

Query

Wiadomość określa adres początkowy oraz liczbę rejestrów do zapisu. Dane do zapisu są upakowane po 2 bajty na rejestr.

Przykład wpisania do rejestrów 40002 i 40003 kolejno wartości 000A hex i 0102 hex.

Nazwa pola	Przykład (hex)
Adres urządzenia	11
Funkcja	10
Adres startowy Hi	00
Adres startowy Lo	01
Liczba elementów Hi	00
Liczba elementów Lo	02
Liczba bajtów	04
Dane Hi	00
Dane Lo	0A
Dane Hi	01
Dane Lo	02
Kontrola błędów (LRC lub CRC)	-

Response

Normalna odpowiedź zawiera adres urządzenia, kod funkcji, adres początkowy oraz liczbę rejestrów.

Nazwa pola	Przykład (hex)
Adres urządzenia	11
Funkcja	10
Adres startowy Hi	00
Adres startowy Lo	01
Liczba elementów Hi	00
Liczba elementów Lo	02
Kontrola błędów (LRC lub CRC)	-

5.9. Podsumowanie

Implementacja protokołu Modbus z wykorzystaniem transmisji szeregowej, pozwala na tworzenie sieci scentralizowanych o dostępie zdeterminowanym. Pomimo pewnych niewątpliwych zalet, rozwiązanie to posiada też szereg ograniczeń. Ograniczenia te wynikają zarówno z konstrukcji samego protokołu, jak i z właściwości warstwy fizycznej (RS232C, RS485).

Zalety to przede wszystkim:

- prostota i łatwość implementacji – Modbus jest standardem otwartym; konstrukcja master/slave; wykorzystanie prostych, bardzo popularnych warstw fizycznych.
- elastyczność – właściwie brak jest ograniczeń, co do typu urządzeń, które mogą wykorzystywać protokół; możliwość tworzenia własnych funkcji, które będą uzupełniać standardowy zestaw.
- dobre zabezpieczenie przed błędami i przekłamaniami – kontrola parzystości i LRC lub CRC; algorytm powiadamiania o sytuacjach wyjątkowych.
- zdeterminowany czas operacji – dostęp do sieci zdeterminowany; dzięki konstrukcji master/slave brak jest kolizji ramek.
- łatwa modyfikacja sieci – sieć oparta na strukturze magistrali pozwala na łatwe dodawanie i usuwanie kolejnych jednostek.
- brak skomplikowanego okablowania.

Do wad należą:

- ograniczona liczba urządzeń w sieci – nie pozwala to na tworzenie rozbudowanych sieci; wydaje się to niewielkim problemem, biorąc pod uwagę fakt, że duża scentralizowana sieć byłaby mało efektywna.
- ograniczona prędkość transmisji – komunikacja szeregowa ustępuje pod tym względem rozwiązaniom takim, jak Ethernet.
- ograniczony rozmiar wiadomości – dla porównania Ethernet pozwala na przesyłanie wiadomości o długości nawet 1500 bajtów; ograniczenie to nie pozwala

na konstruowanie własnych, rozbudowanych funkcji.

- brak możliwości zaimplementowania jakiegokolwiek algorytmu sterowania lub regulacji pomiędzy jednostkami slave bez udziału mastera.

14. Literatura

[1]. Jędrzej Ułasiewicz (2007). Systemy czasu rzeczywistego QNX6 Neutrino, Wydawnictwo BTC, Warszawa 2007, ISBN 978-83-60233-27-6, s.301.

[2]. Krzysztof Sacha (2006). Systemy czasu rzeczywistego, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2006, ISBN 83-7207-124-1, s. 135.

[3]. Standard Computer Dictionary, IEEE Std. 610,1990.

[4]. Modbus Protocol Reference Guide. PI-MBUS-300. Rev. J, 1996, Modicon, http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf, http://modbus.org/docs/PI_MBUS_300.pdf, s.121., http://www.wingpath.co.uk/modbus/modbus_protocol.php