ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: INFORMATYKA z. 34 Nr kol. 1381

Ryszard PAWŁOWSKI
Politechnika Śląska, Instytut Informatyki

STACJA MASTER HIERARCHICZNEJ SIECI POLOWEJ STEROWANEJ ROZKAZAMI JĘZYKA SCPI*)

Streszczenie. W publikacji przedstawiono algorytmy działania stacji master hierarchicznej sieci polowej sterowanej rozkazami języka SCPI. Moduł master pełni w sieci rolę urządzenia łączącego dwie podsieci na kolejnych poziomach hierarchii i pośredniczącego w przesyłaniu komunikatów pomiędzy nimi. W urządzeniu zaimplementowany został prosty, znakowy protokół ASCII oraz protokół ModBUS w trybie ASCII. Sterowanie przesyłem komunikatów odbywa się z wykorzystaniem rozkazów języka SCPI.

MASTER STATION OF THE HIERARCHICAL FIELDBUS CONTROLLED BY SCPI COMMANDS

Summary. In the paper the algorithms of the master station of the hierarchical fieldbus controlled by SCPI commands are presented. The master module acts as a linking device in communication between consecutive levels of hierarchy. In this module the simple, character oriented ASCII protocol and ModBUS protocol in ASCII mode are implemented. Data flow is controlled by the SCPI language commands.

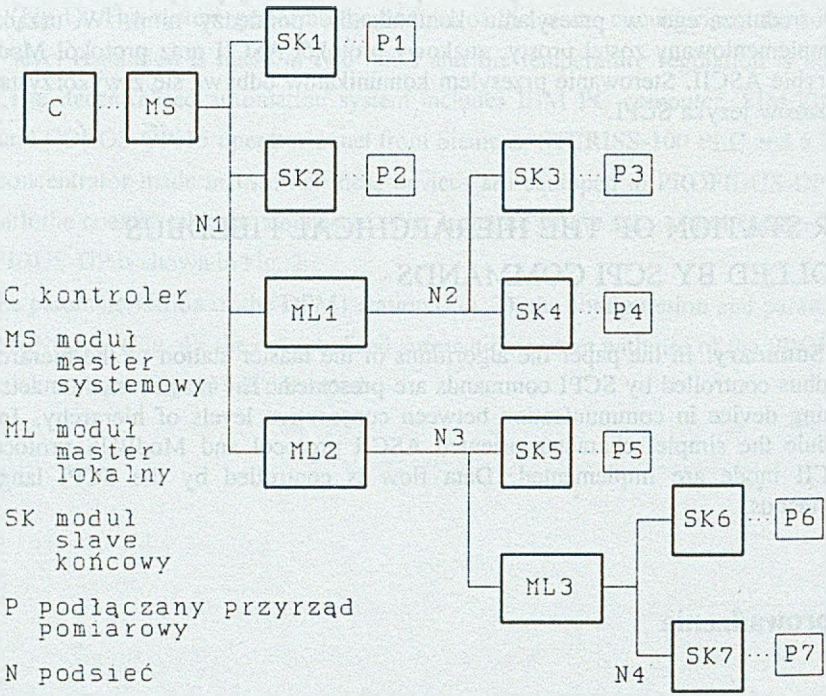
1. Wprowadzenie

Niniejsza publikacja przedstawia budowę i działanie stacji master hierarchicznej sieci polowej sterowanej rozkazami języka SCPI (Standard Commands for Programmable

*) Praca finansowana przez KBN w ramach projektu badawczego 302/T11/97/12

Instruments) [4], której koncepcję zaprezentowano w publikacji [1]. Na rys. 1 przedstawiono konfigurację hierarchicznej sieci polowej i określono podstawowe moduły komunikacyjne używane w sieci. Są nimi:

- kontroler C (stacja nadrzędna sterująca pracą całej sieci);
- master systemowy MS (stacja łącząca kontroler z pierwszym poziomem hierarchicznej sieci polowej);
- slave końcowy SK (stacja umożliwiająca podłączenie urządzenia pomiarowego P do sieci);
- urządzenie pomiarowe P (urządzenie pomiarowo-kontrolne podłączane do sieci);
- master lokalny ML (stacja zarządzająca pracą urządzeń na danym poziomie hierarchicznej sieci polowej oraz pośrednicząca w wymianie komunikatów pomiędzy sieciami na kolejnych poziomach hierarchii: siecią nadrzędną na wyższym poziomie i siecią podrzędną na niższym poziomie).



Rys. 1. Konfiguracja hierarchicznej sieci polowej

Fig. 1. The hierarchical fieldbus configuration

Master lokalny stanowi zmianę w stosunku do rozwiązania przedstawionego w [1]. W pierwotnej koncepcji połączenie pomiędzy kolejnymi poziomami hierarchii wymagało dwóch modułów komunikacyjnych: modułu slave pośredniczący i modułu master podrzędny. Po przeanalizowaniu problemu zaproponowano połączenie funkcji obu tych modułów w pojedynczym module mastera lokalnego. Równoczesną obsługę transmisji na obu sieciach umożliwia mikrokontroler Dallas 80C320 wyposażony w dwa porty szeregowe. Układ ten jest rozbudowaną wersją mikrokontrolera 80C32. W kolejnych punktach przedstawiono najważniejsze założenia do projektu oprogramowania modułu master lokalny oraz algorytmy najważniejszych procesów realizowanych przez moduł.

2. Protokoły komunikacyjne

W module zaimplementowane zostały dwa protokoły komunikacyjne: ModBUS [3] oraz protokół znakowy ASCII. Do sterowania przepływem komunikatów wykorzystano język SCPI.

Protokół ModBUS jest protokołem typu master-slave, przeznaczonym dla sieci opartych na asynchronicznym łączu znakowym. Transmisja oparta jest na tzw. transakcji, na którą składa się przesłanie rozkazu z modułu master do określonego modułu slave, a następnie odesłanie odpowiedzi z modułu slave do modułu master. W ramce odpowiedzi moduł slave powiela w polu <adres> swój adres, natomiast w polu <rozkaz> powiela kod otrzymanego rozkazu w przypadku poprawnego jego wykonania lub wpisuje kod rozkazu z ustawionym bitem <7> sygnalizując błąd wykonania rozkazu (tzw. odpowiedź szczególna). Każdy moduł slave ma przyporządkowany adres z zakresu 1-247. Istnieje również rozgłoszeniowy tryb przesyłania rozkazów (tzw. broadcast), w którym moduły slave nie odsyłają odpowiedzi. Protokół ModBUS może pracować w jednym z dwóch trybów transmisji: ASCII lub RTU. W prezentowanym module zaimplementowany został protokół ModBUS ASCII. Ramka w tym trybie składa się z następujących pól:

<znacznik początku ramki>	znak ':';
<adres>	adres modułu slave, dla którego przeznaczona jest ramka;
<rozkaz>	kod rozkazu ModBUSa;
<dane>	opcjonalne pole parametrów rozkazu;
<LRC>	suma kontrolna LRC;
<znacznik końca ramki>	znaki CR i LF.

Poszczególne bajty ramki (od pola <adres> do <LRC>) przesyłane są jako dwa znaki ASCII cyfr szestnastkowych (znaki '0'-'9' i 'A'-'F') reprezentujące odpowiednio starszą i młodszą czwórkę bitów w danym bajcie.

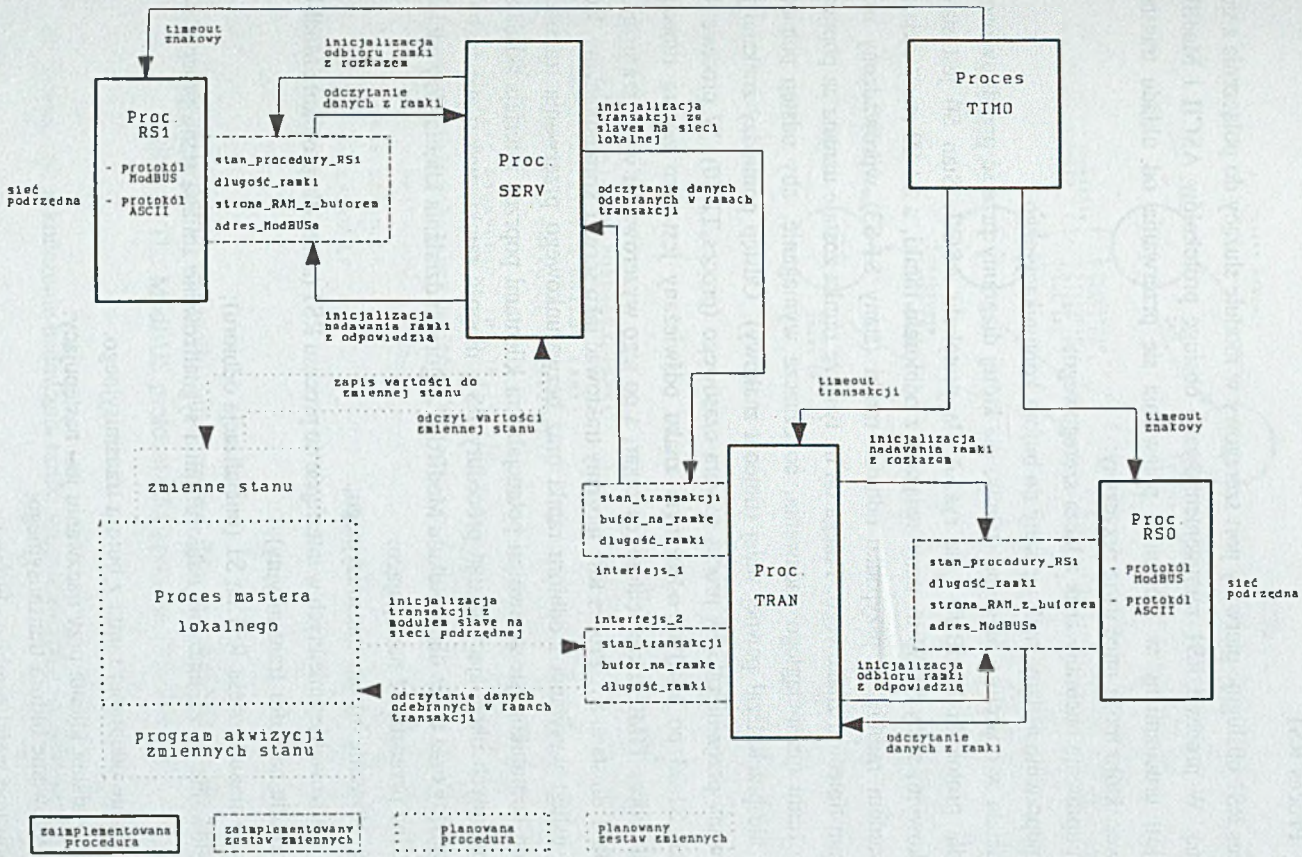
Protokół ASCII jest prostym, znakowym protokołem przesyłania danych dla asynchronicznego łącza znakowego typu punkt-punkt. Ramka w tym protokole składa się ze znaków ASCII o kodach od 20h (SPACJA) do 7fh. Znacznikiem końca ramki jest para znaków CR i LF. Protokół ASCII stosowany jest w dwóch miejscach w proponowanej sieci (rys. 1): na jej "wejściu" przy połączeniu pomiędzy kontrolerem a masterem systemowym oraz na jej "wyjściu" przy połączeniu modułów typu slave końcowy z przyrządami pomiarowymi (np. z przyrządami, które mają zaimplementowany język SCPI). W opisywanej hierarchicznej sieci polowej w ramkach protokołu ASCII transmitowane są rozkazy języka SCPI.

Język SCPI jest standardowym językiem programowania przyrządów pomiarowych wywodzącym się z normy IEEE-488.2 . Rozkazy języka SCPI posiadają hierarchiczną strukturę odzwierciedlającą hierarchiczną budowę dekoderów rozkazów przyrządów SCPI. Moduły komunikacyjne hierarchicznej sieci polowej można traktować jako rozproszony dekoder rozkazów języka SCPI i wykorzystywać rozkazy SCPI do sterowania przepływem komunikatów w sieci [1].

3. Oprogramowanie stacji master

Przy opracowaniu oprogramowania modułu master lokalny przyjęto następujące założenia:
- Oprogramowanie powinno mieć budowę modułową, aby w przyszłości możliwe były zmiany polegające np. na dodaniu kolejnego protokołu komunikacyjnego.
- Ze względu na konieczność równoległej realizacji wielu zadań (odbiór ramek z sieci nadrzędnej, obsługa timeoutów, obsługa transmisji na sieci podrzędnej itd.) oprogramowanie zostało zrealizowane jako zespół współpracujących ze sobą procesów.
- Komunikacja pomiędzy poszczególnymi procesami odbywa się poprzez wspólne zmienne.
- Poszczególne procesy mogą być traktowane jako serwery lub klienci.
- Serwery udostępniają określony zestaw usług, np. proces obsługujący łącze szeregowe realizuje: odbiór ramki, nadawanie ramki oraz bezwarunkowe przerwanie transmisji.
- Klienci korzystają z usług oferowanych przez serwery.
- Niektóre z procesów mogą być zarówno serwerami, jak i klientami. Dzieje się tak wtedy, gdy jakiś proces oferuje zestaw usług (jest serwerem), a do ich realizacji wykorzystuje usługi oferowane przez inny serwer (dany proces występuje wtedy w roli klienta).
- Większość procesów pracuje w oparciu o mechanizm maszyny stanów.

Na rys. 2 przedstawiono podstawowe procesy wchodzące w skład oprogramowania modułu master lokalny oraz powiązania między tymi procesami.



Rys. 2. Podstawowe elementy składowe oprogramowania modułu master lokalny

Fig. 2. The basic software elements of the local master module

3.1. Proces RS1

Proces RS1 obsługuje pierwszy port szeregowy w module służący do połączenia z siecią nadrzędną. W procesie RS1 zaimplememtowano obsługę protokołów ASCII i ModBUS. Proces jest uruchamiany w momencie pojawienia się przerwania od układu transmisji szeregowej, które może mieć dwie przyczyny:

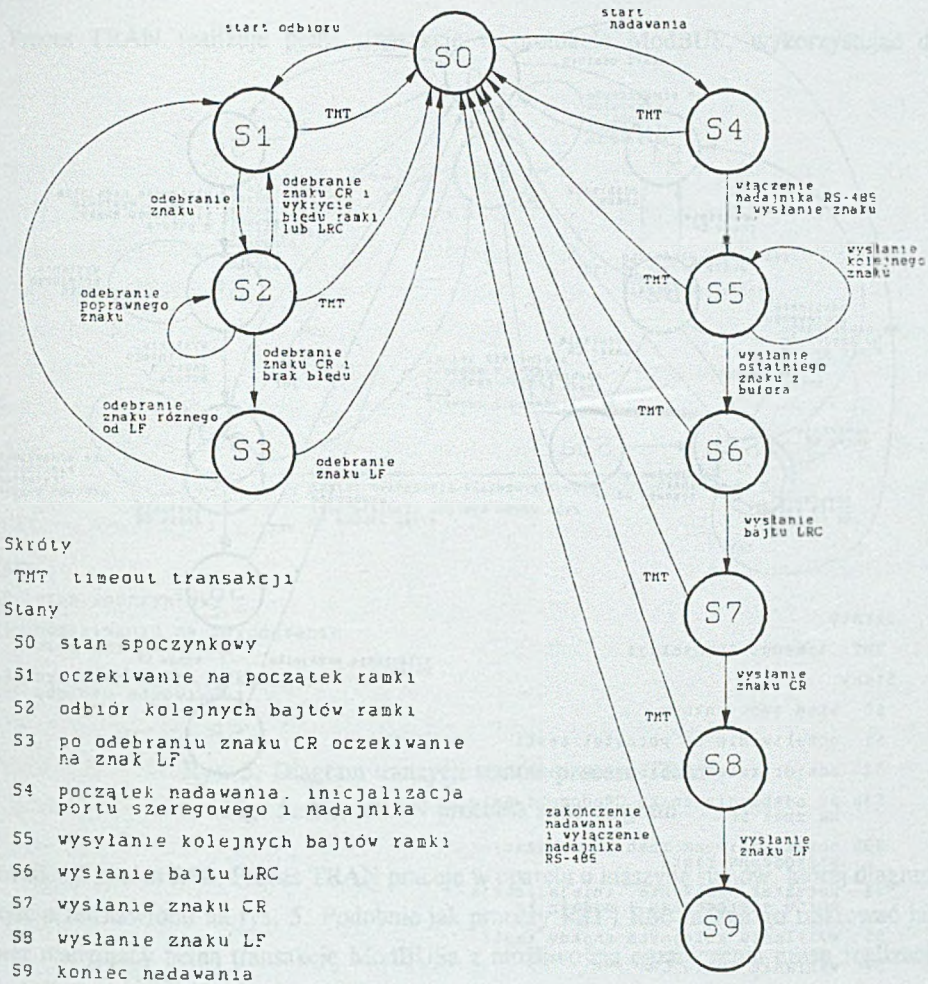
- został poprawnie odebrany znak z łącza szeregowego;
- został poprawnie nadany znak wpisany do bufora komunikacyjnego.

Proces działa w oparciu o maszynę stanów, dla której diagramy przejść przedstawiono na rys. 3 dla protokołu ModBUS i na rys. 4 dla protokołu ASCII. Stan S0 jest stanem spoczynkowym, stany S1-S3 to stany związane z odbiorem ramki, a S4-S9 stany związane z nadawaniem ramki. W przypadku odbioru ramki (stany S1-S3) wprowadzony został mechanizm timeoutu znakowego. Polega on na tym, że ramka zostaje uznana za poprawną przy założeniu quasi-ciągłego nadawania, co oznacza wymaganie, aby odstęp pomiędzy znakami nie przekraczał pewnej stałej (timeout znakowy). Odstęp pomiędzy znakami jest kontrolowany przez niezależny proces servera czasowego (proces TIM0). W procesie RS1 w stanach S1-S3 po każdym odebranym znaku odświeżany jest stan zegara timeoutu, natomiast proces TIM0 dekrementuje ten zegar, a po jego wyzerowaniu wymusza przejście procesu RS1 do stanu S1. Proces RS1 możemy traktować jako serwer komunikacyjny, który realizuje usługi wysyłania i odbioru ramki oraz bezwarunkowego przerwania transmisji. Serwer ten komunikuje się ze swoimi potencjalnymi klientami poprzez interfejs składający się ze zmiennych określających stan procedury RS1, długość ramki, wskazanie na bufor komunikacyjny oraz adres dla protokołu ModBUS. Algorytm działania klienta przy odbiorze ramki można przedstawić następująco:

- przygotowanie bufora transmisyjnego;
- zainicjalizowanie zmiennych w interfejsie do procesu RS1 (adres dla protokołu ModBUS i wskazanie na bufor transmisyjny);
- zmiana stanu procesu RS1 na S1 (inicjalizacja odbioru);
- oczekiwanie na zakończenie odbioru ramki sygnalizowane zmianą stanu procesu RS1 na S0;
- odczytanie odebranej ramki z bufora transmisyjnego.

Algorytm działania klienta przy nadawaniu jest następujący:

- przygotowanie bufora transmisyjnego;
- przepisanie treści ramki do bufora;
- zainicjalizowanie zmiennych w interfejsie do procesu RS1 (długość ramki i wskazanie na bufor transmisyjny);
- zmiana stanu procesu RS1 na S4 (inicjalizacja nadawania);

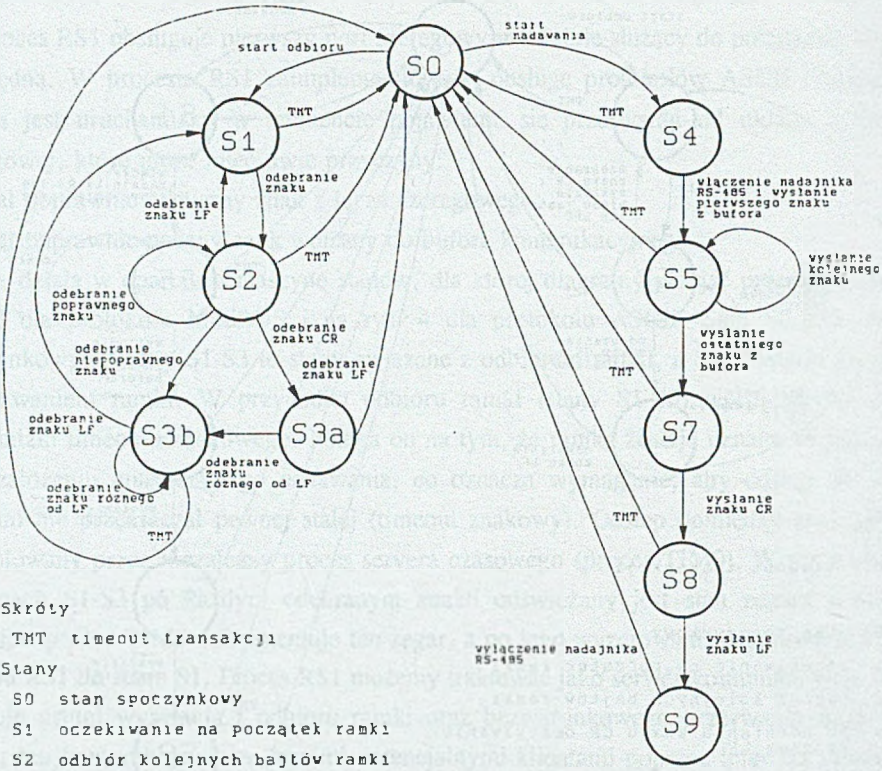


Rys. 3. Diagram tranzycji stanów procesu RS1 dla protokołu ModBUS

Fig. 3. The ModBUS protocol RS1 proccess state diagram

- oczekiwanie na zakończenie nadawania ramki sygnalizowane zmianą stanu procesu RS1 na S0.

W obu przypadkach można bezwarunkowo przerwać operację realizowaną przez proces RS1 i wymusić przejście do stanu S0.



Skróty

TMT timeout transakcji

Stany

S0 stan spoczynkowy

S1 oczekiwanie na początek ramki

S2 odbiór kolejnych bajtów ramki

S3a po odebraniu znaku CR oczekiwanie na znak LF

S3b oczekiwanie na znak LF kończący uszkodzoną ramkę

S4 początek nadawania, inicjalizacja portu szeregowego i nadajnika

S5 wysyłanie kolejnych znaków ramki

S7 wysłanie znaku CR

S8 wysłanie znaku LF

S9 koniec nadawania

Rys. 4. Diagram tranzycji stanów procesu RS1 dla protokołu ASCII

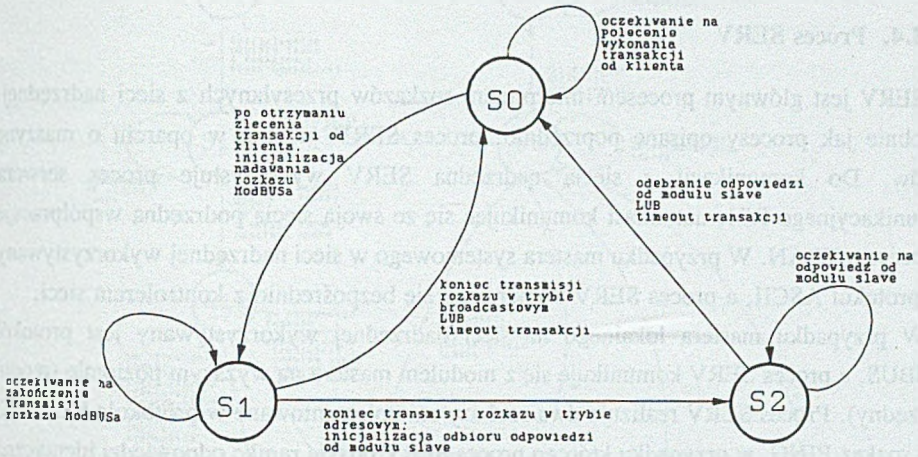
Fig. 4. The ASCII protocol RS1 proccess state diagram

3.2. Proces RS0

Proces RS0 obsługuje zerowy port szeregowy w mikrokontrolerze Dallas działając identycznie jak proces RS1. Pomimo że proces RS0 obsługuje sieć podrzędną, na której wykorzystywany jest protokół ModBUS, proces RS0 realizuje także protokół ASCII.

3.3. Proces TRAN

Proces TRAN realizuje pełną transakcję w protokole ModBUS, wykorzystując do



Stany:

S0 stan spoczynkowy

S1 oczekiwanie na zakończenie nadawania rozkazu

S2 oczekiwanie na zakończenie odbioru odpowiedzi

Rys. 5. Diagram tranzycji stanów procesu TRAN

Fig. 5. The TRAN proccess state diagram

komunikacji proces RS0. Proces TRAN pracuje w oparciu o maszynę stanów, której diagram przejść przedstawiono na rys. 5. Podobnie jak procesy RS1 i RS0 można go traktować jak serwer realizujący pełną transakcję ModBUSa z możliwością ograniczenia czasu realizacji transakcji (timeout transakcji). Sam proces wywoływany jest w przerwaniu zegarowym co 1ms. Obecnie jedynym klientem procesu TRAN może być opisany poniżej proces SERV, ale oprogramowanie mastera lokalnego zostanie w przyszłości rozbudowane o dodatkowy proces realizujący lokalne sterowanie w obrębie własnej sieci. Z tego powodu proces TRAN został wyposażony w dwa interfejsy do komunikacji ze swoimi klientami. Każdy z tych interfejsów zawiera zmienne określające polecenie wykonania transakcji i jej wynik (status), wskazanie na bufor komunikacyjny i długość ramki w buforze. Mechanizm timeoutu transakcji jest realizowany przez proces serwera czasowego TIM0 niezależnie od procesu TRAN. Zegar timeoutu pozwala na ograniczenie czasu transakcji od 10ms do 2.5s. W przypadku zadziałania zegara timeoutu proces TRAN zostaje natychmiast przerwany przez zmianę jego

stanu na S0. Dodatkowo wymuszane jest bezwarunkowe przerwanie transmisji w procesie RS0. Aktualnemu klientowi przekazywany jest kod błędu timeoutu transakcji.

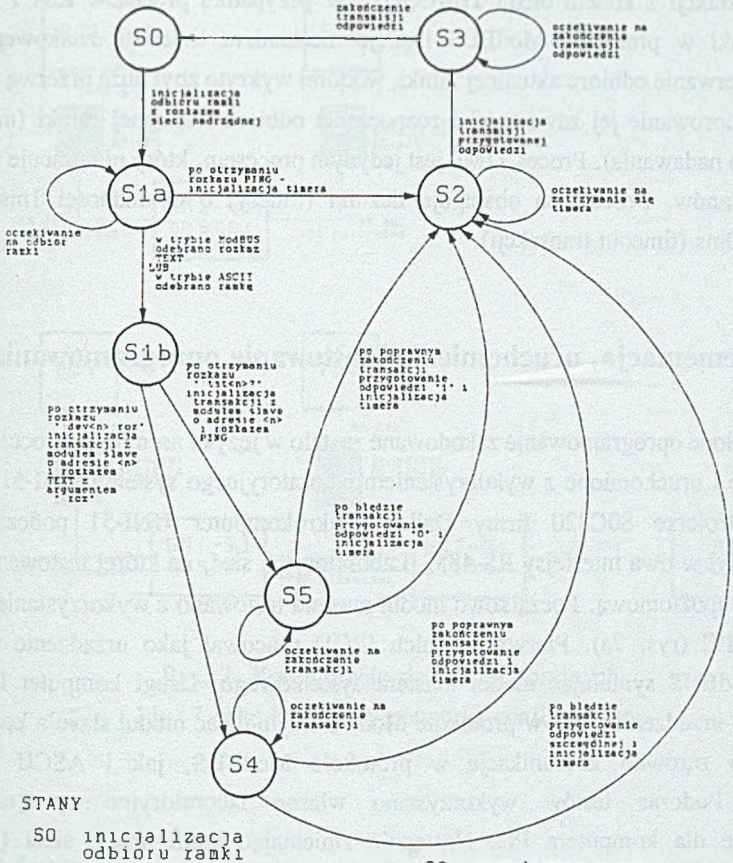
3.4. Proces SERV

SERV jest głównym procesem interpretera rozkazów przesyłanych z sieci nadrzędnej. Podobnie jak procesy opisane poprzednio, proces SERV pracuje w oparciu o maszynę stanów. Do komunikacji z siecią nadrzędną SERV wykorzystuje proces serwera komunikacyjnego RS1, natomiast komunikując się ze swoją siecią podrzędną współpracuje z procesem TRAN. W przypadku mastera systemowego w sieci nadrzędnej wykorzystywany jest protokół ASCII, a proces SERV komunikuje się bezpośrednio z kontrolerem sieci.

W przypadku mastera lokalnego na sieci nadrzędnej wykorzystywany jest protokół ModBUS, a proces SERV komunikuje się z modułem mastera na wyższym poziomie (moduł nadrzędny). Proces SERV realizuje dwa rozkazy zaimplementowane w protokole ModBUS:

- rozkaz PING, w przypadku którego proces SERV odsyła ramkę odpowiedzi identyczną jak ramka z rozkazem;
- rozkaz TEXT, w przypadku którego proces SERV wyodrębnia z pola danych rozkaz SCPI i przekazuje go do dekodera SCPI.

Dekoder języka SCPI rozpoznaje dwa rozkazy SCPI ":dev<n>:polecenie" i ":tst<n>?". W przypadku rozkazu "dev<n>:polecenie" wyodrębniany jest adres <n> modułu slave oraz rozkaz ":polecenie" dla tego modułu, a następnie proces SERV zleca wykonanie transakcji procesowi TRAN. Wynik realizacji transakcji jest odsyłany modułowi nadrzędnemu za pośrednictwem procesu RS1. Wynikiem może być ramka ModBUSa zawierająca poprawną odpowiedź SCPI lub ramka ModBUSa z odpowiedzią szczególną. Na początku transakcji proces SERV programuje czas timeoutu transakcji zlecając kontrolę tego czasu procesowi TIM0. W przypadku przekroczenia timeoutu transakcji proces TRAN jest natychmiast przerywany, a proces SERV odsyła odpowiedź szczególną z kodem błędu TIMEOUT. W przypadku rozkazu ":tst<n>?" wyodrębniany jest adres <n> modułu slave, a następnie proces SERV zleca wykonanie transakcji procesowi TRAN. W transakcji tej do modułu slave wysyłany jest rozkaz PING ModBUSa. Jeżeli pod danym adresem znajduje się moduł slave, to odeśle on odpowiedź w protokole ModBUS. Proces SERV po poprawnym zakończeniu transakcji ModBUSa odeśle do modułu nadrzędnego wynik "1". Jeżeli pod danym adresem nie ma żadnego modułu, to transakcja zakończy się błędem timeoutu, a proces SERV odeśle jako odpowiedź SCPI "0". Rozkaz ":tst<n>?" jest w istocie rozkazem testującym obecność modułu slave o adresie <n>. Diagram przejść dla maszyny stanów procesu SERV przedstawiono na rys. 6.



Rys. 6. Diagram tranzycji stanów procesu SERV

Fig. 6. The SERV proccess state diagram

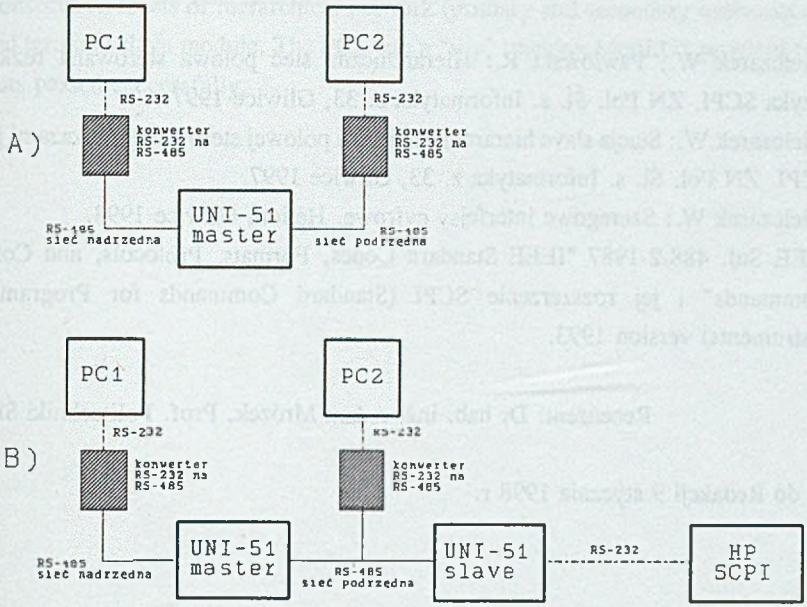
3.5. Proces TIM0

Proces obsługujący szereg timerów i generujący komunikaty do innych procesów informujące je o zaistnieniu zdarzenia typu "minął maksymalny czas przeznaczony na daną operację". Po otrzymaniu tego komunikatu proces powinien przerwać aktualnie wykonywaną

operację. Tak dzieje się np. w przypadku procesu TRAN, gdzie następuje przerwanie realizacji transakcji z kodem błędu TIMEOUT. W przypadku procesów RS1 i RS0 przy odbiorze ramki w protokole ModBUS istnieje mechanizm timeoutu znakowego, który powoduje przerwanie odbioru aktualnej ramki, w której wykryto zbyt dużą przerwę pomiędzy znakami, zignorowanie jej zawartości i rozpoczęcie odbioru następnej ramki (mechanizm quasi-ciągłego nadawania). Proces TIM0 jest jedynym procesem, który nie pracuje w oparciu o maszynę stanów. Proces ten obsługuje liczniki (timery) o dokładności 1ms (timeout znakowy) i 10ms (timeout transakcji).

4. Implementacja, uruchomienie i testowanie oprogramowania

Przedstawione oprogramowanie zakodowane zostało w języku asemblera procesora 8051, skompilowane i uruchomione z wykorzystaniem laboratoryjnego systemu UNI-51 opartego na mikrokontrolerze 80C320 firmy Dallas. Mikrokomputer UNI-51 podczas testów wyposażony był w dwa interfejsy RS-485. Laboratoryjna sieć, na której testowano moduł, była siecią dwupoziomową. Początkowo moduł mastera testowano z wykorzystaniem dwóch komputerów PC (rys. 7a). Pierwszy z nich (PC1) pracował jako urządzenie master w protokole ModBUS symulując moduł mastera systemowego. Drugi komputer PC (PC2) pracował jako urządzenie slave w protokole ModBUS symulając moduł slave'a końcowego. Przetestowano zarówno komunikację w protokole ModBUS, jak i ASCII (na sieci nadrzędnej). Podczas testów wykorzystano własne laboratoryjne oprogramowanie komunikacyjne dla komputera PC. Następnie zmieniając konfigurację sieci (rys. 7b), połączono moduł mastera lokalnego z modułem slave'a końcowego [2], który współpracował z urządzeniem pomiarowym (multimetr HP34401A z interpreterem języka SCPI, na rysunku oznaczony jako HP SCPI). W tej konfiguracji komputer PC1 pracował jako master systemowy i wysyłał rozkazy SCPI transportowane w ramkach protokołu ModBUS do urządzenia końcowego (multimetr) za pośrednictwem dwóch modułów: mastera lokalnego oddzielającego dwa poziomy hierarchicznej sieci polowej (sieć nadrzędna i sieć podrzędna na rys. 7b) oraz modułu slave'a końcowego. Komputer PC2 pełnił rolę "szpiega" pozwalającego śledzić zawartość ramek protokołu ModBUS. Podczas testów sprawdzono zarówno komunikację w protokole ModBUS (rozkazy PING i TEXT), jak i mechanizmy sterowania przepływem komunikatów w sieci wykorzystujące język SCPI (rozkazy :dev<n> i :tst?).



Rys. 7. Konfiguracje sieci laboratoryjnych

Fig. 7. The laboratory networks configurations

5. Podsumowanie

Przedstawione oprogramowanie mastera lokalnego realizuje komunikację pomiędzy sąsiednimi poziomami w hierarchicznej sieci polowej sterowanej rozkazami języka SCPI pełniąc w istocie rolę routera. Oprogramowanie wykonano jako system modułowy składający się z kilku współpracujących ze sobą procesów (rys. 2). Wśród procesów możemy wyróżnić serwery, które udostępniają określony zestaw usług (np. proces TRAN realizujący transakcję ModBUSa) oraz klientów, korzystających z usług serwerów (np. proces SERV, który jest klientem procesu TRAN). Część procesów funkcjonuje w systemie zarówno jako serwery, jak i klienci (przykładem jest proces TRAN). Z uwagi na ograniczenia mikrokontrolerów serii 8051 zasoby poszczególnych procesów (pamięć, stos) nie są chronione, a komunikacja pomiędzy nimi odbywa się za pośrednictwem zmiennych globalnych.

LITERATURA

[1] Mielczarek W., Pawłowski R.: Hierarchiczna sieć polowa sterowana rozkazami języka SCPI. ZN Pol. Śl. s. Informatyka z. 33, Gliwice 1997.

[2] Mielczarek W.: Stacja slave hierarchicznej sieci polowej sterowanej rozkazami języka SCPI. ZN Pol. Śl. s. Informatyka z. 33, Gliwice 1997.

[3] Mielczarek W.: Szeregowe interfejsy cyfrowe. Helion, Gliwice 1993.

[4] IEEE Std. 488.2-1987 "IEEE Standard Codes, Formats, Protocols, and Common Commands" i jej rozszerzenie SCPI (Standard Commands for Programmable Instruments) version 1993.

Recenzent: Dr hab. inż. Adam Mrózek, Prof. Politechniki Śląskiej

Wpłynęło do Redakcji 9 stycznia 1998 r.

Abstract

The local master software presented in this publication is a kind of router for the hierarchical fieldbus controlled by the SCPI commands. The software was built as a set of proccesses communicating together (fig. 2). There are two types of proccesses: server proccess with its characteristic services (for example TRAN proccess performing ModBUS transaction) and client proccess (for example SERV proccess - a client of TRAN proccess). Some of proccesses works as server proccess and client proccess (for example TRAN proccess). The resources of proccesses (memory, stack) are not protected and communication between proccesses uses global variables. The software was coded in 8051 assambler language, compiled and executed in UNI-51 lab system with Dallas 80C320 microcontrollers. For tests the UNI-51 was equiped with two RS-485 interfaces. The lab test network was two level. First master module with two PC computers (fig. 7a) was tested. The PC1 worked as ModBUS protocol master device, simulating primary master, network module (sytem master). The PC2 worked as ModBUS protocol slave device, simulating terminal slave network module. The ModBUS and ASCII protocol communication in primary network was tested. MTST software for PC computer was used for communication testing. Next, the network configuration was changed (fig. 7b). The local master module was connected to terminal slave module [2], that worked with terminal measurement device (HP34401A multimetr with SCPI interpreter). In this case the PC1 working as the system master module sended SCPI commands inside ModBUS protocol frames to terminal measurement device.

This transmission used two communication modules: the local master module that separated two consecutive levels of hierarchical network (primary and secondary networks on the fig. 7b) and terminal slave module. The PC2 was a "spy" tracking ModBUS protocol frames. All the tests passed successfully.