

Zdalny system pomiarowy z interfejsem Ethernet,

część 3

AVT-5111

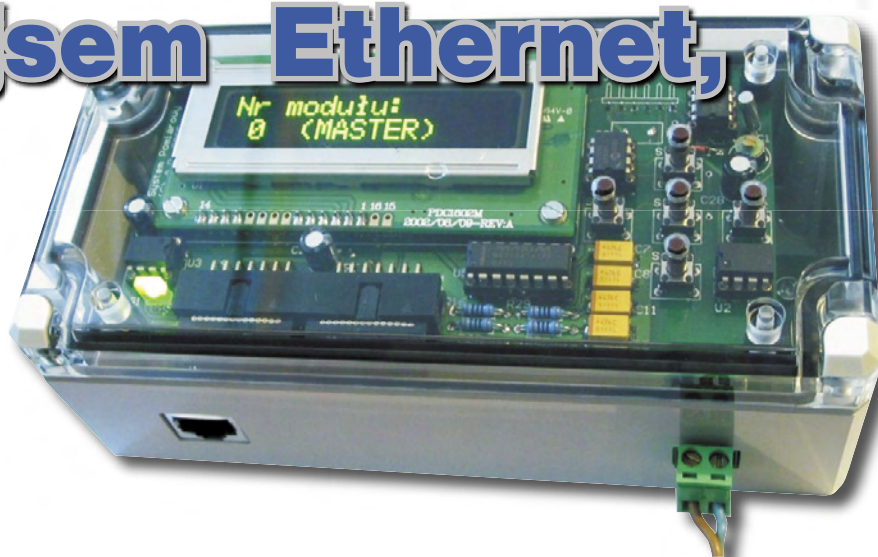
Układy monitoringu i zdalnego sterowania zyskały nową jakość wraz z upowszechnieniem szerokopasmowego dostępu do Internetu. Podobnie stało się z rozproszonymi systemami pomiarowymi pozwalającymi sterować nawet złożonymi procesami produkcyjnymi z poziomu jednej aplikacji.

W artykule prezentujemy przykładowy projekt systemu pomiarowego, w którym sterowanie odbywa się poprzez Internet.

Rekomendacje: projekt mimo sporych możliwości i dość rygorystycznych założeń technicznych stanowi doskonale pole doświadczalne dla projektantów właśnie ujarzmiających Ethernet.

PODSTAWOWE PARAMETRY

- Płytki o wymiarach: 136x128 (151 z wypustkami) mm
- Podstawowe cechy systemu pomiarowego:
 - liczba modułów połączonych wspólną magistralą RS485: od 1 do 32,
 - liczba wejść/wyjść jednego modułu: 4 wejścia z pętlą prądową 0/4...20 mA, 2 wejścia stykowe oraz 2 wyjścia przekąźnikowe,
 - możliwość zapisywania kolejnych pomiarów w wewnętrznej pamięci EEPROM (w każdym module),
 - stempel czasowy dodawany do każdego pomiaru,
 - możliwość odczytu mierzonych wartości na wbudowanym wyświetlaczu,
 - zabezpieczenie zapisanych parametrów i nastaw zegara czasu rzeczywistego przed zanikami napięcia zasilania,
 - zdalny podgląd stanu czujników oraz ustawianie wyjść przekąźnikowych,
 - obsługa systemu przez aplikację uruchomioną na komputerze PC



Oprogramowanie modułów

Część elektryczna urządzenia nie należy do bardzo skomplikowanych. Za to dużo większy nacisk musi być położony na oprogramowanie zapisane w pamięci mikrokontrolera. Dodatkowym utrudnieniem jest tutaj fakt, że program modułu Master jest w dużej mierze inny niż program modułów Slave.

Oprogramowanie modułów pełni następujące funkcje:

- zapewnia poprawną komunikację reszty układu z modułem ethernetowym Tibbo, przy pomocy sprzętowego łącza szeregowego mikrokontrolera,
- analizuje polecenia przesyłane z aplikacji komputera zdalnego i realizuje wymagane zadania,
- wysyła do zdalnego komputera odpowiedzi na przesyłane zapytania,
- zapewnia poprawną łączność między modułami Master i Slave, przy użyciu protokołu Modbus RTU,
- obsługuje pamięć EEPROM oraz zegar czasu rzeczywistego, pracujące w standardzie magistrali I²C,
- pełni funkcję interfejsu pomiędzy przetwornikiem analogowo-cyfrowym a mikrokontrolerem,
- zapewnia interakcję z użytkownikiem poprzez obsługę wyświetlacza alfanumerycznego PLED i przycisków.

Sprzętowe łącze szeregowe. Komunikacja z modułem EM100

Moduł Master – w odróżnieniu od modułu Slave – musi obsługiwać dwa niezależne łącza szerego-

we. Pierwsze z nich (komunikacja z modułem ethernetowym) zrealizowano wbudowanym w mikrokontroler ATmega32 sprzętowym układem USART. Słowo „sprzętowy” oznacza tutaj samodzielną transmisję poszczególnych bitów ramki, kontrolę błędów i generację bitów parzystości. Do zadań programisty należy jedynie wybór prędkości transmisji, tryb pracy (liczba bitów danych, stopu, z kontrolą parzystości lub bez) oraz wpisywanie/odbieranie kolejnych bajtów do/z specjalnego rejestru mikrokontrolera. Ponieważ mikrokontroler musi obsługiwać prawie jednocześnie kilka niezależnych operacji, konieczne było zastosowanie buforowania bajtów wysyłanych i odbieranych. System przerwań (w tym przypadku konkretnie od układu USART) pełni tutaj ważną rolę. Dla przykładu, przerwanie wygenerowane po odebraniu kolejnego bajtu danych powoduje przepisanie jego wartości do bufora, czyli specjalnej tablicy umieszczonej w pamięci RAM mikrokontrolera i zwiększenie licznika bufora o jeden. Następnie obsługa wraca do programu głównego. Autor, aby nie wywahać otwartych drzwi, wykorzystał istniejące i udostępnione fragmenty kodu realizujące dla przykładu buforowaną transmisję szeregową. Fragmenty te w zasadzie powinny być nazwane dodatkowymi bibliotekami dołączonymi do programu głównego.

Tak na marginesie, komercyjne kompilatory mają szereg takich bibliotek dołączanych przez producenta. Dla przykładu można tu wy-

mienić obsługę wyświetlacza LCD czy magistrali I²C. W przypadku darmowego kompilatora, jakim jest GCC, sami użytkownicy piszą takie biblioteki, aby ułatwić sobie nawzajem pisanie oprogramowania. W wielu przypadkach konieczna wszakże była pewna modyfikacja zapożyczonych fragmentów, tak aby program spełniał założone funkcje.

Jak już wcześniej wspomniano, sprzętowy układ interfejsu szeregowego zapewnia komunikację między modulem ethernetowym a mikrokontrolerem. Zasadniczo transmisja tym kanałem odbywa się w dwóch trybach. Są to:

- normalna, dwukierunkowa transmisja danych między modulem Tibbo i będącym „za nim” zdalnym komputerem a mikrokontrolerem,
- praca w trybie programowania modułu Tibbo przez mikrokontroler.

W pierwszym trybie bajty wysyłane przez mikrokontroler są w niezmięnionej postaci odbierane na zdalnym komputerze. Dane są przesyłane tekstowo (wynika to m.in. z użycia takiego, a nie innego komponentu do obsługi protokołu TCP w aplikacji zdalnej), a każda porcja danych (komunikat) zakończony jest parą znaków CR LF, czyli znaków powrotu karetki i nowej linii (znaki ASCII o kodach #13 i #10 dziesiętnie). Powyższa zasada dotyczy jednakowo danych wysyłanych i odbieranych.

Znaki końca komunikatu nie mogą – co jest zrozumiałe – występować w treści komunikatu, z tego też powodu mogą zostać wykorzystane do sygnalizacji jego końca. Po ich wykryciu aplikacja, czy też mikrokontroler mogą przejść do analizy komunikatu i związanych z nim dalszych działań.

W drugim trybie, służącym do programowania modułu Tibbo, dane są również przesyłane tekstowo, inna jest tylko enkapsulacja komunikatów. Zaczyna się on od znaku STX (kod ASCII #2), a kończy znakiem CR (kod ASCII #13). Zostało to już wcześniej opisane w części poświęconej modułowi EM100. Tak więc, aby poprawnie odbierać komunikaty zarówno z trybie pierwszym, jak i drugim, konieczna jest zmiana znaku jego końca w zależności od aktualnego trybu pracy.

Tab. 6. Zestawienie poleceń z aplikacji i możliwych odpowiedzi

Komunikat	Opis
<p>połączenie: CRnn.eee</p> <p>odpowiedź dla eee w zakresie 0...7: RCRnn.eee.aaaa.rr-mm-dd.gg:tt:ss</p> <p>odpowiedź dla eee=255: RCRnn.eee.aa.bb.cc.dd.ee.ff.gg.hh.rr-mm-dd. gg:tt:ss</p> <p>timeout i błąd: RCRnn.eee.TIMEOUT RCRnn.eee.ERROR</p>	<p>odczyt wskazanego we/wy: nn – numer urządzenia Modbus (0...31), eee – nr wejścia/wyjścia: 0 – we A1, 1 – we A2, 2 – we A3, 3 – we A4, 4 – we B1, 5 – we B2, 6 – wy C1, 7 – wy C2, 255 – wszystkie, aaaa – wynik pomiaru (0...4095), aaaa do hhhh – wyniki pomiarów dla kolejnych kanałów we/wy (0...4095), rr – rok (0...15), mm – miesiąc (1...12); dd – dzień (0...31), gg – godzina (0...23), tt – minuta (0...59), ss – sekunda (0...59).</p>
<p>połączenie: RNDnn.ssss.</p> <p>odpowiedź jeśli są nowe dane: RRNDnn.uuuu-pppp:m0.m1.m2.m3.m4.m5. m6.m7, m8.m9.m10.m11.m12.m13.m14.m15, (itd.)</p> <p>odpowiedź jeśli nie ma nowych danych: RRNDnn.NONEWDATA</p> <p>timeout i błąd: RRNDnn.TIMEOUT RRNDnn.ERROR</p>	<p>odczyt nowych pomiarów automatycznych z pamięci EEPROM modułu, nn – numer urządzenia Modbus (0...31), ssss – numer ostatniego odebranego rekordu, uuuu – numer pierwszego przesyłanego rekordu (ssss + 1), pppp – numer ostatniego przesyłanego rekordu, 8-bajtowe paczki danych rozdzielone są znakami przecinka, paczek maksymalnie może być 31</p>
<p>połączenie: Mnn.ssss.xxx.</p> <p>odpowiedź: RMnn.ssss:dd,dd,dd,dd,...</p> <p>timeout i błąd: RMnn.TIMEOUT RMnn.ERROR</p>	<p>odczyt 8-bajtowych segmentów z pamięci EEPROM, nn – numer urządzenia Modbus (0...31), ssss – numer 8-bajtowego segmentu początkowego – 0 dla pierwszego segmentu w pamięci, max. 16383 (dla pamięci 128 kB), xxx – liczba przesyłanych segmentów, dd – kolejne bajty odczytane z pamięci</p>
<p>połączenie: TAnn.eee.xxxx.</p> <p>odpowiedź: RTAnn.eee.OK</p> <p>timeout i błąd: RTAnn.eee.TIMEOUT RTAnn.eee.ERROR</p>	<p>ustawienie automatycznego pomiaru: nn – numer urządzenia Modbus (0...31), eee – nr wejścia/wyjścia (0...7) patrz opis wyżej, xxxx – czas automatycznego pomiaru co xxxx sekund</p>
<p>połączenie: TPnn.eee.gd.yyyyy.</p> <p>odpowiedź: RTPnn.eee.OK</p> <p>timeout i błąd: RTPnn.eee.TIMEOUT RTPnn.eee.ERROR</p>	<p>ustawienie dolnego i górnego progu pomiarowego dla wyznaczonej jednostki, nn – numer urządzenia Modbus (0...31), eee – nr wejścia/wyjścia (0...7) patrz opis wyżej, gd – 0 dla progu dolnego, 1 dla progu górnego, yyyyy – liczba zmiennoprzecinkowa z opcjonalnym znakiem (maksymalnie 8 znaków łącznie ze znakiem)</p>
<p>połączenie: TJnn.eee.pppppppp.</p> <p>odpowiedź: RTJnn.eee.OK</p> <p>timeout i błąd: RTJnn.eee.TIMEOUT RTJnn.eee.ERROR</p>	<p>ustawienie jednostki kanału pomiarowego A1...A4, nn – numer urządzenia Modbus (0...31), eee – nr wejścia/wyjścia kanałów A (0...3) patrz opis wyżej, pppppppp – jednostka (8 znaków),</p>
<p>połączenie: QSnn.eee.onoff.</p> <p>odpowiedź: RQSnn.ee.OK</p> <p>timeout i błąd: RQSnn.eee.TIMEOUT RQSnn.eee.ERROR</p>	<p>włączenie bądź wyłączenie wyjścia przekaźnikowego, nn – numer urządzenia Modbus (0...31), eee – 6 dla wyjścia C1, 7 dla wyjścia C2 onoff – 0 – wyłączenie, 1 – włączenie</p>
<p>połączenie: TCrr-mm-dd.gg:tt:ss.</p> <p>odpowiedź: RTC.OK</p>	<p>ustawienie zegara i daty, rr – rok (0...15), mm – miesiąc (1...12); dd – dzień (0...31), gg – godzina (0...23), tt – minuta (0...59), ss – sekunda (0...59).</p>
<p>połączenie: TGCO.</p> <p>odpowiedź: RTGCr-mm-dd.gg:tt:ss.</p>	<p>odczytanie czasu i daty modułu Master, rr – rok (0...15), mm – miesiąc (1...12); dd – dzień (0...31), gg – godzina (0...23), tt – minuta (0...59), ss – sekunda (0...59).</p>
<p>połączenie: SM</p> <p>odpowiedź: RSMxxxx</p>	<p>skanowanie aktywnych modułów Slave, xxxx – 32-bitowa liczba, każdy kolejny bit reprezentuje jeden moduł Slave (bit 1 – moduł Slave nr 1, itd.)</p>

W czasie normalnej pracy urządzenia znakiem końca komunikatu jest LF (#10); tryb programowania i znacznik CR (#13) używany jest jedynie podczas zmiany numeru IP oraz portu TCP modułu MASTER.

Programowe łącze szeregowe. Obsługa interfejsu RS485

Mikrokontroler ATmega32 niestety nie posiada drugiego interfejsu łącza szeregowego. Tym samym, konieczna jest jego programowa emulacja. Linie we/wy mikrokontrolera wyznaczone do obsługi tego interfejsu to PD2 (linia nadawcza TXD) i PD3 (linia odbiorcza RXD). Wyprowadzenie PD3 może także pełnić rolę źródła zewnętrznego przerwania INT1 i ta funkcja została tutaj celowo wykorzystana.

Wyprowadzenie PD3 zostało skonfigurowane jako wejście generujące przerwanie INT1 przy pojawieniu się na nim opadającego zbocza. Jest to wykorzystane do wykrycia bitu startu ramki. Następnie przerwanie INT1 zostaje wyłączone, a włączone zostaje przerwanie licznikowe OC0 (Output Compare 0) odliczające okresy czasu równe długości bitu przy założonej prędkości transmisji (pierwszy odmierzany okres jest równy 1,5-krotności tego czasu). W tych wyznaczonych odpowiednio odstępach czasu linia PD3 (RXD) jest próbkowana, a kolejny bit jest dopisywany do odbieranego bajtu. Po odebraniu bitu stopu i skompletowaniu całego bajtu, zostaje on wpisany do bufora odbiorczego, przerwanie licznikowe OC0 zostaje wyłączone, a INT1 ponownie włączone w celu wykrycia kolejnego bitu startu.

Emulacja programowa łącza szeregowego jest na pewno bardziej skomplikowana i wymaga czasochłonnego nakładu programowego, co ogranicza maksymalną prędkość transmisji. W prototypie użyto prędkości 78,6 kbit/s.

Obsługa protokołu MODBUS RTU

Działające programowe łącze szeregowe to jedynie pierwszy krok do uzyskania poprawnej komunikacji między modułami. W trybie MODBUS RTU nie jest możliwe wyznaczenie konkretnego znaku sygnalizującego koniec transmisji komunikatu (pamiętamy, że komunikat kończy się sumą kontrolną CRC). Standard definiuje za to czas przerwy w nadawaniu kolejnych bajtów, powyżej której zakładany jest koniec transmisji. Czas ten ustalono na 3,5-krotność długości znaku, czyli dla trybu RTU niecałe 2 bajty.

W celu poprawnego określenia końca komunikatu w pewien sposób musiała zostać zmodyfikowana procedura obsługi programowego łącza szeregowego. Odbywa się to w sposób opisany niżej.

W momencie odebrania bitu stopu ramki zerowana jest pewna wyznaczona zmienna i odblokowywane jest dodatkowe przerwanie licznikowe (Timer 0 Overflow). W procedurze obsługi tego przerwania wspomniana zmienna jest zwiększana o jeden i sprawdzana jest jej wartość. Gdy zostanie osiągnięta pewna wyznaczona wartość maksymalna, przerwanie jest blokowane i zostaje ustawiona inna zmienna sygnalizująca już programowi głównemu odebranie kolejnego komunikatu. W czasie transmisji kolejnych bajtów danych zmienna licznikowa jest cyklicznie zerowana, tym samym nie dopuszcza się do osiągnięcia przez nią wartości granicznej. Wartość ta została ustalona tak, aby przerwa w transmisji równa ok. 2 bajtom sygnalizowała odebranie całej paczki danych.

Polecenia przekazywane z aplikacji

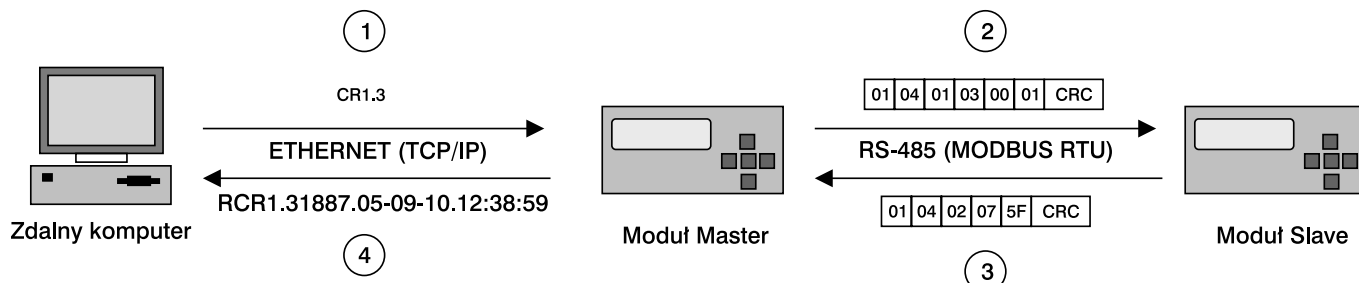
Aby móc komunikować się zdalnie z systemem pomiarowym przez sieć Internet, należy napisać osobny program pracujący na komputerze

z zainstalowanym systemem operacyjnym Microsoft Windows. Program ten będzie wysyłał i odbierał komunikaty z systemu pomiarowego przesyłane siecią komputerową protokołem TCP/IP.

Interakcja aplikacji na komputerze zdalnym z systemem pomiarowym polega w uproszczeniu na wysyłaniu przez aplikację poleceń, ich realizacji przez moduł Master, a następnie przetwarzaniu odpowiedzi wysłanych przez system. Konieczne stało się opracowanie do tego celu specjalnego „języka” poleceń. Autor użył w tym celu formatu komunikatów nieco podobnych do tych, jakie występują w komunikacji w trybie programowania z modułem EM100. Polecenia są wysyłane w trybie tekstowym, mają postać jednolub kilkukiliterowego skrótu polecenia oraz opcjonalnych parametrów. Zestawienie wszystkich możliwych poleceń i odpowiedzi przedstawiono w **tab. 6**.

W przypadku, gdy polecenie jest kierowane do któregoś z modułów Slave, moduł Master z kolei przesyła polecenie magistralą RS485 do urządzenia podrzędnego, czeka na odpowiedź, a następnie w zależności od niej wysyła odpowiedź do komputera zdalnego. Przykład polecenia odczytu jednego z kanałów modułu Slave i obieg komunikatów pokazano na **rys. 11**. W przypadku, gdy moduł Slave, do którego zostało skierowane zapytanie nie odpowiada, moduł Master informuje o tym fakcie aplikację na zdalnym komputerze, wysyłając komunikat TIMEOUT (przekroczenie limitu czasu).

Protokół Modbus definiuje różne obszary pamięci dla różnych kodów funkcji. Przykładowo dla funkcji Read Input Register, czyli odczytu rejestrów wejściowych, przyjęto adresowanie kolejnych rejestrów od prefiksu 3XXXX heksadecymalnie, gdzie XXXX jest 16-bitowym adresem rejestru. Także same rejestry są traktowane jako 16-bitowe. Dla funkcji Read Holding Registers, czyli odczytu rejestrów sta-



Rys. 11. Przykładowy obieg komunikatów po wydaniu polecenia odczytu pojedynczego kanału modułu Slave

Hi 4bit	dana Lo 8-bit	nr Modbus	br I/O	rok	miesiąc	dzień	godzina	minuta	sekunda
bajt 1	bajt 2	bajt 3	bajt 4	bajt 5	bajt 6	bajt 7	bajt 8	bajt 9	bajt 10

Rys. 12. Format pojedynczego rekordu pomiarowego

nu, przyjmuje się adresowanie 4XXXX heksadecymalnie. Sposób mapowania obszaru wejścia/wyjścia oraz pamięci pomiarów w poszczególnych modułach był więc kwestią dość dowolną i mógł być rozwiązany na wiele sposobów, byle tylko jednoznacznie określał poszczególne obszary.

Pamięć EEPROM

W komórce pamięci EEPROM o adresie 0000H zapisywany jest numer modułu Slave (tożsamy z adresem używanym w protokole Modbus). Dwa bajty z literami „OK” są zapisywane przy pierwszym użyciu pamięci i są używane do rozpoznania czystej pamięci. Dalej zapisane są parametry kanałów pomiarowych, czyli dolne i górne progi zakresów pomiarowych (odpowiadające prądom pętli 4 mA i 20 mA), następnie jednostki pomiarowe. Od adresu 0068H zapisane są 2-bajtowe liczby odpowiadające odstępom między kolejnymi pomiarami w sekundach (maksymalnie 65535 s). Od adresu 0078H rozpoczyna się obszar przeznaczony na zapisywanie wyników pomiarów. Rozciąga się on do samego końca pamięci. Rekord pomiarowy (rys. 12) składa się z 8 bajtów – składa się na nią 12-bitowy wynik pomiaru (2 pierwsze bajty), następnie numer modułu i numer wejścia/wyjścia (trzeci bajt). Kolejne bajty służą do zapisu daty i czasu pomiaru.

Jak można zauważyć, zarówno dane pomiarowe, jak i część parametrów zapisane są w pamięci w grupach 8-bajtowych. Nie jest to działa-

nie przypadkowe. Konkretnie wynika to ze sposobu zapisu w zastosowanej pamięci EEPROM. Ma on m.in. możliwość zapisu stronicowego, gdzie strona to 256 bajtów. Występuje tu jednak pewne niebezpieczeństwo: gdy zapisując kolejne bajty przekroczymy granicę strony, kolejne bajty będą nadpisywane od początku TEJ SAMEJ strony. Aby uniknąć takiego przypadku najbardziej sensowne jest takie zorganizowanie danych w pamięci, aby poszczególne „paczki” danych nigdy nie przechodziły przez granice stron. Organizacja 8-bajtowa, począwszy od adresu podzielnego bez reszty przez 8 spełnia ten warunek.

Adresowanie w komunikacji protokołem Modbus

Polecenia przesyłane do modułów Slave można podzielić następująco:

- odczyt kanałów we/wy – wykorzystuje się tu funkcję Read Input Registers,
- odczyt konfiguracji bądź pamięci pomiarów – wykorzystuje się w tym przypadku funkcję Read Holding Registers,
- zapis nowych konfiguracji kanałów – funkcja Preset Multiple Registers,
- ustawianie wyjść dwustanowych C1 i C2 – funkcja Force Single Coil

Pewnym problemem jest tutaj 16-bitowa postać zarówno adresu rejestru, jak i założenie istnienia 16-bitowych rejestrów. Pamięć EEPROM modułów po pierwsze ma organizację


bajtową, a po drugie rozmiar 128 kB wymaga 17-bitowego adresu. Autor zdecydował się więc na pewne naruszenie standardu Modbus i przyjął nieco inne założenia dotyczące adresowania pamięci. Jednym z rozwiązań (dla danych pomiarowych) jest adresowanie nie konkretnych komórek pamięci EEPROM, a całych rekordów 8-bajtowych. Dla pamięci 128 kB mamy $(131072-120)/8=16369$ rekordów, zatem bez problemu mieścimy się w 16-bitowym polu adresowym. Dodatkowo zastosowano przesunięcie początku obszaru pamięci o adres 8000H. Adres na przykład 10. rekordu będzie więc równy 800AH. Można oczywiście przed właściwym adresem dopisać prefiks ujęty w standardzie (dla funkcji Read Holding Registers jest to 4X), jest to jednak tylko kwestia umowna.

Od adresu 0000H umieszczono kolejne parametry konfiguracyjne, do których może się odnosić moduł Master. Tutaj z kolei zastosowano adresowanie bajtowe, co jest pewną niekonsekwencją, ale ma to niewielkie znaczenie w samodzielnie pracującym systemie. Dla kanałów wejścia/wyjścia zarezerwowano adresy od 0100H do 0107H (kolejno A1...A4, B1, B2, C1, C2).

Ustawianie wyjść dwustanowych to funkcja Force Single Coil; adresy dla kanałów C1 i C2 pozostawiono takie same jak wyżej, czyli 0106H dla kanału C1 oraz 0107H dla kanału C2.

Andrzej Piernikarczyk
andrzej.piernikarczyk@gmail.com

R	E	K	L
≡ montaż powierzchniowy SMD	≡ montaż przewlekany THT uzupełniający		
≡ programowanie, testy ICT i funkcjonalne	≡ projektowanie obwodów drukowanych		
≡ kompleksowe przygotowanie dokumentacji	≡ zakup i kompletacja podzespołów		



nte

ul. Gaudiego 7
44-100 Gliwice

tel: +48 32 33 82 200
fax: +48 32 33 82 210

e-mail: produkcja@ente.com.pl
http://www.ente.com.pl

TUV
IPPC EKRA
JUKI
ERSA

A	M	A
---	---	---



www.micromotors.pl
silniki@discotech.pl



Silniczki prądu stałego od 20 mA/12 V z wbudowaną przekładnią
Silniki krokowe cztero- i sześcioprzewodowe
Silniczki 12 lub 230 V 50 Hz od 2,5 do 60 obr./min. już od 15 zł + VAT