Uniwersalne moduły I²C

Interfejs sprzętowy I²C i przykład jego wykorzystania

kit AVT-816

Dokładne opisy interfejsu I²C zastosowanego w procesorze '652 i pochodnych są dostępne w materiałach producenta. Był on też przedstawiony w artykule opisującym projekt płytki 80C552 (EP4 i 5/96). Postanowiłem jednak jeszcze raz przypomnieć szczegóły obsługi interfejsu, skupiając się na praktycznych aspektach jego stosowania. Dzięki temu kończący się cykl artykułów stanowi kompendium wiedzy o I²C w '51.

Tajemnice interfejsu

W procesor '652 wbudowano sprzętowy, zorientowany bajtowo interfejs komunikacji szeregowej. Program sterujący jego pracą musi więc obsługiwać - podobnie jak w przypadku RS232 - wysyłanie i odbieranie całych znaków (bajtów), bez konieczności dbania o szczegóły transmisji i niuanse protokołu I²C.

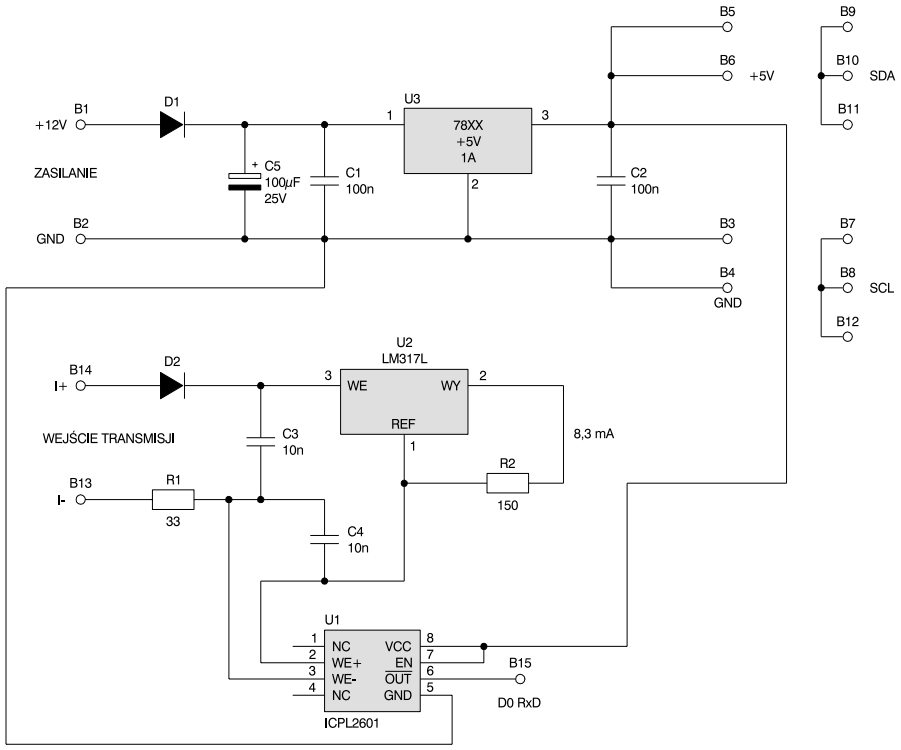
Podobnie odbywa się konfigurowanie interfejsu - poprzez wpis danych do odpowiednich SFR-ów.

Podobieństwo występuje także w sposobie programowej realizacji transmisji: możemy wykorzystywać „polling“, czyli okresowe sprawdzanie przez program czy coś się zdarzyło w interfejsie, ale jest to marnowanie czasu i możliwości procesora. Znacznie lepiej jest skorzystać z bogatego systemu przerwań, bo polling zdaje egzamin tylko w prostych programikach, zaś tam gdzie mamy do czynienia z rozbudowanym, wielofunkcyjnym programem wykorzystującym zależności czasowe może być źródłem niespodziewanych kłopotów. Poza tym stosując przerwania otrzymujemy do ręki dodatkowe atuty:

- Raz napisany uniwersalny moduł obsługi sesji komunikacyjnej może być bez zmian lub z niewielkimi zmianami stosowany w późniejszych aplikacjach.
- Użycie słowa statusu interfejsu do wektoryzacji obsługi przerwania radykalnie upraszcza proces ustalania, co się w interfejsie wydarzyło. Jest to o tyle istotne, że pomimo wszelkich podobieństw do UART-u, sesja komunikacyjna I²C jest jednak dużo bardziej skomplikowana (UART to odbiór znaku - flaga RI lub nadanie znaku - flaga TI, natomiast tutaj szczegółowych przypadków jest co najmniej kilkanaście).

Wszystko to brzmi pewnie mało zrozumiale, ale zaraz wszystko się wyjaśni. Zacznijmy od konfiguracji. W pracy interfejsu możemy wyróżnić cztery podstawowe tryby pracy:

- nadawanie jako Master (Master transmitter);
- odbieranie jako Master (Master receiver);
- odbieranie jako Slave (Slave receiver);
- nadawanie jako Slave (Slave transmitter).



Rys. 1. Schemat elektryczny bloku zasilania i odbioru transmisji.

Master jest urządzeniem nadrzędnym, inicjującym i kończącym sesję transmisyjną oraz generującym sygnał zegarowy SCL.

Slave - to urządzenie podrzędne, które zgłasza się na wywołanie *Mastera*, korzysta z jego sygnału SCL i zgodnie z poleceniami *Mastera* przyjmuje oraz wysyła dane.

Interfejs jest konfigurowany za pomocą rejestrów S1CON i S1ADR. Rejestr S1ADR ładujemy praktycznie jednorazowo: umieszczamy tam adres (7-bitowy) procesora jako wywoływanego urządzenia Slave (o ile rzecz jasna przewidujemy takie jego wykorzystanie). Ósmy, najmłodszy bit określa, czy procesor ma wykrywać - oprócz swojego adresu *Slave* - również tzw. wywołanie ogólne (0).

Rejestr S1CON (adresowany bitowo) jest używany zarówno przy pierwszej konfiguracji jak i w trakcie transmisji. Jego poszczególne bity oznaczają:

7	6	5	4
CR2	ENS1	STA	STO
3	2	1	0
SI	AA	CR1	CR0

ENS1 - uruchomienie (1) lub zablokowanie (0) interfejsu.

CR2, CR1, CR0 - ustawienie szybkości transmisji w trybie Master - pozwala dopasować szybkość do warunków pracy magistrali oraz częstotliwości kwarcu. Najbardziej typowe zestawienia podano w **tab. 1**.

Flaga AA (ang. Assert Acknowledge) ma rozmaite znaczenie w zależności od momentu jej wykorzystania. Na etapie konfiguracji włącza (1) lub wyłącza (0) tryb Slave. Oznacza to, że jeśli wyzerujemy flagę, to interfejs nie będzie reagować na wywołanie swojego adresu (lub wywołanie ogólne). Przy odbieraniu danych (niezależnie od trybu) wyzerowanie flagi spowoduje brak potwierdzenia odbioru następnego bajtu - jest to na ogół wykorzystywane do poinformowania nadajnika, że nie potrzeba więcej danych i ma zakończyć nadawanie. Przy nadawaniu jako Slave - wyzerowanie flagi informuje układ interfejsu, że następny wysyłany bajt danych ma być traktowany jako ostatni (powoduje to po wysłaniu tego bajtu jedynie odmienny wpis do rejestru statusu - praktyczna przydatność tego przypadku jest niewielka: końcowy bajt wykrywamy raczej na podstawie braku jego potwierdzenia).

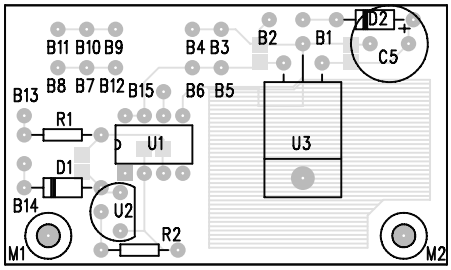
W opisach możemy napotkać określenia: tryb *Slave* zaadresowany i tryb *Slave* nie zaadresowany. Nie zaadresowany - to oczekiwanie (z ustawioną flagą AA) na wywołanie własnego adresu przez jakiegoś Mastera. Zaadresowany - to realizacja transmisji po wykryciu i potwierdzeniu własnego adresu - trwa ona dopóki nie zakończy jej *Master*, zaś flagą AA możemy posługiwać się jak powyżej.

STA - ustawienie tej flagi generuje znacznik startu na magistrali i powoduje przejście układu w tryb Master transmitter. Po pełnej realizacji transmisji Master powinien albo zakończyć ją ustawiając flagę STO (wysłanie znacznika stopu) albo od razu wznowić poprzez tzw. ponowny start (ponowne ustawienie STA bez wcześniejszego STO). Flagi STA i STO są zerowane automatycznie po wykonaniu operacji.

SI - flaga przerwania, zostaje ustawiona w razie wystąpienia zdarzenia na magistrali I²C. Ponieważ zdarzenia mogą być różne (np. odebrano własny adres, wysłano sygnał startu, wysłano bajt danych itd.) jednocześnie do słowa statusu S1STA wpisany zostaje kod określający co zaszło. Zarazem - o ile jest odblokowane - zostaje zgłoszone przerwanie od portu szeregowego 1. Flaga SI - podobnie jak dla UART - nie gaśnie samoczynnie i musi być wyzerowana programowo.

Z opisu powyższego widać, że wyszczególnione na wstępie tryby pracy nie mają charakteru sztywnych ustawień - zależą zarówno od stanu flag jak i od przebiegu programu oraz od interakcji z innymi użytkownikami magistrali I²C (nie ma np. czegoś takiego jak ustawienie konfiguracyjne trybu Master receiver - interfejs wchodzi w ten tryb z trybu Master transmitter po wysłaniu adresu urządzenia Slave z żądaniem odczytu i otrzymaniu potwierdzenia ACK).

Rejestr S1STA zwalnia nas z zapamiętywania w programie kolejnych etapów sesji transmisyjnej, jednakże w obsłudze każdego przerwania musimy sprawdzić co w nim jest. W wersji tradycyjnej byłaby to rozbudowana instrukcja



Rys. 2. Rozmieszczenie elementów na płytce drukowanej.

porównująca (typu *case*) mocno komplikująca i wydłużająca program. Dlatego przewidziano zalecane przez producenta wykorzystanie zawartości S1STA do wspomnianej już wektoryzacji obsługi przerwania.

Polega to na uformowaniu z wartości S1STA (młodszy bajt) i dowolnej (w ramach pojemności stosowanego EPROM-u), ale wstępnie (na etapie kompilacji) jednoznacznie ustalonej 8-bitowej wartości (starszy bajt) adresu skoku do odpowiadającej danemu przypadkowi procedury obsługi.

Kto kiedyś programował Z80SIO pamięta może, że podobny jest jeden z trybów obsługi przerwania - tyle że młodsza część wektora przerwania jest przekazywana przez urządzenie zewnętrzne po magistrali danych, a tu pobieramy ją z SFR.

Wartości S1STA są ustawiane modulo 8 (czyli co 8 bajtów) - na ogół nie wystarcza to dla pełnej procedury i praktycznie lokujemy tu instrukcję skoku do dalszej części obsługi (analogicznie jak w obszarze przerwań sprzętowych). Wszystkie przypadki zajmują całą stronę (256 bajtów) - co przy 32kB zastosowanych w naszym systemie nie ma żadnego znaczenia.

W zestawie instrukcji '51 nie ma skoku pod adres podany jako argument (czyli nie znany na etapie kompilacji). Dlatego wykorzystuje się tu instrukcję powrotu: RET, która jest niczym innym jak skokiem pod adres pobrany z wierzchołka stosu, a żądany adres (czyli nasz adres odpowiedniej procedury) lokujemy na stosie instrukcjami PUSH:

```
PUSH S1STA
MOV A,#STARSZY_BAJT_ADRESU
PUSH ACC
```

Naszym zadaniem jest więc ulokowanie w programie odpowiednich fragmentów kodu pod odpowiednimi (absolutnymi) adresami. Bardzo wygodnie jest przygotować sobie dobrze opisany uniwersalny szablon, który później wypełnimy w zależności od potrzeb aplikacji. Nie jest to trudne - jedynie nieco pracochłonne - natomiast szczegółowe rozwiązania zależą od posiadanego assemblera (np. postać dyrektyw, używanie relokowalnych modułów i linkera itd.).

Przykład

Aby nie pozostawać w sferze teorii, opisane w poprzednich częściach artykułu moduły posłużyły do zbudowania przykładowej aplikacji: 8-kanałowego uniwersalnego wyświetlacza wartości temperatur. Na przykład wybrałem wyświetlanie temperatury, ponieważ wyświetlacz służy też do prezentacji modułu pomiaru temperatur z obiektowego minisystemu akwizycji danych opartego na I²C. Aby utrzymać ciągłość opisu interfejsu I²C przedstawię urządzenie odwrotnie niż zazwyczaj - zaczynając od oprogramowania.

Mamy tu do czynienia z pracą wyłącznie typu *Master* (transmitter i receiver). Większość małych pojedynczych urządzeń właśnie tak pracuje - dlatego przygotowałem uniwersalny moduł programowy do obsługi nadawania i odbioru. Dla realizacji sesji komunikacyjnej I²C posługuje się on parametrami zadanymi w programie głównym i zwraca flagi określające rezultat sesji.

Może być więc wielokrotnie, bez żadnych przeróbek, wykorzystywany w szeregu różnych programów (był użyty również w konwerterze RS232/I²C). Potrafi wysłać zadaną liczbę bajtów z bufora nadajnika, wygenerować ponowny start po określonym numerze bajtu oraz przyjąć zadaną liczbę bajtów lokując je kolejno w buforze odbiornika i sygnalizując ostatni bajt brakiem potwierdzenia ACK. Przeglądając opisy katalogowe stwierdzimy, że jest to typowy schemat realizacji komunikacji z większością układów (*start>>wywołanie do zapisu>>wpis adresów, komend i danych>>ponowny start>>wywołanie do odczytu>>odczyt danych>>stop*).

Moduł jest obsługiwany tylko po zgłoszeniu przerwania. W głównej pętli programu są zadawane parametry oraz jest uruchomiana sesja (wynikiem sesji może być flaga błędu, flaga poprawnego zakończenia i flaga odbioru danych). W podstawowej wersji bufory nadawczy i odbiorczy są ulokowane w pamięci wewnętrznej - na ogół wystarcza to do obsłużenia większości spotykanych układów (transmituje się nie więcej niż po kilkanaście bajtów). Natomiast dla dłuższych transmisji przewidziano wersję z lokacją buforów w zewnętrznym RAM.

Obie wersje programu znajdują się w zestawie kodów źródłowych, które można znaleźć na stronie *www.ep.com.pl/ftp*. Napisałem je w makroassemblerze MCC-51 jako moduły relokowalne, ale nie powinno być problemów z odtworzeniem ich z użyciem innego narzędzia. Nie korzystają z oddzielnego banku rejestrów - jedynie R0 jest oddzielnie chroniony. Zauważmy, że wiele zdarzeń interfejsu wcale nie powinno zajść, ale oprogramowanie tych zdarzeń jest pozostawione w postaci minimalnej - jako powrót interfejsu do stanu podstawowego. Poprawia to odporność na ewentualne błędy, a także umożliwia bezbolesne zastosowanie urządzenia w systemie z kilkoma Masterami.

Sposób wykorzystania interfejsu w programie jest następujący:

a) Konfiguracja, która obejmuje:
- odblokowanie odpowiedniego przerwania (ES1, EA);
- ustawienie adresu Slave (nie korzystamy);
- ustawienie S1CON: szybkość przyjąłem nie maksymalną, a rzędu 70kHz - dla działania urządzenia nie ma to znaczenia a poprawia odporność na zakłócenia;
- uruchomienie zegara systemowego (odliczanie timeoutu).

b) Obsługa programowa transmisji. Istotną rolę odgrywa flaga informująca o rozpoczęciu sesji (*i2c_busy*). Pozwala ona na odliczanie timeoutu (w przerwaniu timera t0) oraz eliminuje próby zakłócenia przez program już

rozpoczętej sesji. Wprowadzenie timeoutu jest ważne ze względu na możliwość zawieszenia transmisji bez zgłoszenia błędu, np. przy oczekiwaniu na nie nadchodzące dane. Kluczowa dla poprawnego działania programu jest też zmienna *device*, której zawartość określa, które z podłączonych urządzeń właśnie obsługujemy.

Obsługę możemy podzielić na następujące bloki funkcjonalne:
- Wykrywanie błędu: obejmuje sprawdzenie flagi timeoutu oraz flagi błędu transmisji. Pojedynczy błąd inkrementuje licznik błędów - dopiero kilkakrotne powtórzenie błędu wywołuje dalszą reakcję (to eliminuje efekty przypadkowych zakłóceń). Reakcja ta może być ogólna, a może też zależeć od zawartości „device" (np. możemy sygnalizować przyłączenie lub odłączenie jakiegoś urządzenia).
- Czynności po prawidłowo zakończonej sesji: w razie wykrycia flagi poprawnego zakończenia sesji jest podejmowane działanie zależne od zawartości „device": o ile aktualnie obsłużone urządzenie przewidywało odbiór danych - wykorzystujemy je (po dodatkowym sprawdzeniu flagi kompletacji odbioru), następnie ustawiamy kolejną wartość „device". Tu możemy sobie dowolnie manipulować kolejnością oraz częstotliwością obsługi poszczególnych urządzeń, wprowadzać opóźnienia czasowe itp.
- Rozpoczynanie kolejnej sesji:

O ile magistrala jest wolna - czyli flaga *i2c_busy* jest zgaszona - w zależności od zawartości *device* ustawiamy potrzebne wartości parametrów sesji (ile bajtów do nadania i odbioru, kiedy ponowny start) oraz ładujemy odpowiednie dane do bufora nadajnika, a następnie wysyłamy na magistralę znacznik startu (i ustawiamy *i2c_busy*). Powyżej opisane mechanizmy sprawiają, że dalszym przebiegiem sesji nie musimy się już w programie głównym kłopotać! Zauważmy, że przypomina to nieco asynchroniczne wywoływanie funkcji.

Powyższy schemat obsługi jest uniwersalny dla trybu Master i może być - uwzględniając różnice szczegółowe - stosowany w każdej innej aplikacji naszego mini systemu. Nie ma też rzecz jasna znaczenia, czy będzie opracowany w asemblerze, czy w języku wyższego poziomu, aczkolwiek trzeba zauważyć, że ze względu na stosowanie wielu warunków typu case używanie np. C znakomicie ułatwia programowanie.

W zestawie kodów źródłowych znajdziemy program główny w C (użyłem kompilatora MCC, dużo skromniejszego niż IAR czy KEIL, stąd pewna toporność niektórych zapisów: nie ma np. deklaracji stałych globalnych, zmiennych typu *uchar* czy *uint* i trzeba to załatwiać obejściami; stąd też konieczność - dla zachowania szybkości - assemblerowej obsługi przerwań).

Program ten zapewnia:
- obsługę watchdoga;
- obsługę miniklawiatury, która tu służy do przełączania kanałów;
- obsługę mini sygnalizatora LED, określającego numer włączonego kanału;
- obsługę wyświetlacza LCD $3^1/_2$ cyfry.

Dane do wyświetlenia przesyłane są w trybie podstawowym łączem RS (w komplecie oprogramowania znajduje się testowy nadajnik pracujący pod Windows), natomiast po dołączeniu modułu MD590 systemu akwizycji danych - wyświetlacz przechodzi do prezentacji temperatur mierzonych przez moduł.

Tab. 1.

CR2	CR1	CR0	fosc=6MHz	fosc=12MHz	fosc=16MHz	Jednostka
0	0	0	23	47	63	kHz
0	0	1	27	54	71	kHz
0	1	0	31	63	83	kHz
0	1	1	37	75	100	kHz
1	0	0	6,25	12,5	17	kHz
1	0	1	50	100	-	kHz
1	1	0	100	-	-	kHz
1	1	1	taktowanie timerem T1 w trybie 2 (praktycznie mamy do wyboru tyle wariantów, że nie musimy korzystać z tej opcji).			

Opis urządzenia

Urządzenie zostało zmontowane w obudowie typu Z34. Wyświetlacz, klawiatura oraz LED-y są wmontowane w cieńszą część obudowy, w grubszej mieści się płytka procesora oraz zasilanie. Kontakt ze światem zapewnia gniazdo DB9F (Vcc, GND, SCL, SDA, linia prądowa TTY: I+, I-).

Obudowę należy obrobić i otworować z użyciem szablonu. Jednocześnie z szablonem należy wykonać nakładkę czołową, co zapewnia dobrą zgodność wymiarów. Nakładkę obrabiamy poprzez zafoliowanie i wykonanie potrzebnych otworów (okienka dla LED możemy wyciąć przed foliowaniem) - prostokątne wycinamy nożykiem, a okrągłe wycinakami o odpowiedniej średnicy.

Wstępnie dopasowujemy moduły:
- Płaszczyzna wyświetlacza powinna ułożyć się równo z powierzchnią obudowy.
- Diody LED powinny wejść w otwory bez dociskania. Jeśli wykonamy w folii otwory na wylot, to diody mogą wystawać ponad obudowę. W prototypie otwory były przykryte folią, w związku z czym po przyklejeniu modułu z układem PCF8574 do obudowy, diody dochodzą prawie do powierzchni folii.
- Przyciski powinny działać bez zacięć i nie ocierać się o obudowę (długość trzpieni jest dobrana tak aby uzyskać ich odpowiednie wystawanie przy oparciu korpusów przycisków o obudowę). Nie należy zapomnieć, aby przed zmontowaniem nagwintować otwory do skręcenia obudowy.

Następnie wklejamy nakładkę czołową przy pomocy odpowiednio wyciętej taśmy dwustronnej, ustawiamy moduły w odpowiednich miejscach i mocujemy kroplami kleju Poxipol (moduł LED przyklejamy do obudowy grzbietem układu PCF8574). Kleju nie dajemy zbyt dużo, aby pozostawić sobie możliwość

później szego demontażu oraz nie dopuścić do sklejenia trzpieni przycisków.

Przewody modułów po związaniu kleju (kilkanaście minut) zostały doprowadzone do przygotowanej z kawałeczka płytki uniwersalnej łączówki. Łączówka posiada wlutowane poziomo 4 goldpiny (linii INT klawiatury nie wykorzystuję) do nasunięcia płaskiej listwy kontaktowej - można oczywiście wykonać zamiennie stałe połączenie lutowane. Gotową płytkę czołową możemy teraz sprawdzić testerem, aby ułatwić sobie uruchamianie oprogramowania.

Dolną część obudowy przygotowujemy wycinając otwory pod gniazdo i przycisk zerowania oraz przewiercając i gwintując otwory montażowe. Złącze DB9 przykręcamy, natomiast przycisk ręcznego zerowania także przyklejamy Poxipolem (tu trzpień jest krótki - aby dostęp był tylko przy pomocy cienkiego narzędzia). Z tyłu naklejamy etykietę z opisem przycisku oraz gniazda (bardzo ułatwia późniejsze podłączanie wyświetlacza, a przy tym maskuje otwory montażowe).

Do zasilania i odbioru transmisji służy oddzielna płytka. Mieści ona typowy stabilizator U3 z filtrującymi pojemnościami C1, C2, C5 i diodą D1 zabezpieczającą przed odwróceniem polaryzacji. Linia transmisyjna (typu TTY - dwustanowa prądowa) jest dołączona przez diodę zabezpieczającą D2 i filtr R1, C3 oraz ogranicznik prądu U2, R2 do wejścia transoptora U1.

Zastosowałem wygodny - bo z wyjściem TTL - szybki transoptor ICPL2601 (odpowiednik HCPL).

Nominalny prąd wejściowy wynosi 5mA - ogranicznik ustawiłem na nieco więcej (1,25V/150Ω=8,3mA). Płytka jest wmontowana drukiem do góry. Od strony druku oprócz kondensatorów SMD jest wlutowany elektrolit C5 (nie mieści się pomiędzy płytką i obudową). Płaszczyzna miedzi pod 7805 służy jako niewielki radiator. Wystarcza on z powodzeniem dla napięcia zasilania do ok. 12V (uwaga przy symulatorach EPROM zasilanych z układu - wówczas prąd wzrasta dwukrotnie i stabilizator może się przegrzewać, a więc zasilajmy podczas uruchamiania napięciem nie wyższym niż 8..9V).

Obie sprawdzone płytki przykręcamy do obudowy i wykonujemy przewodami odpowiednie połączenia (płytka zasilacza służy zarazem jako łączówka). Przycisk resetu jest w prototypie (wykonanym z płytką jednowarstwową bez oddzielnego wyprowadzenia resetu) włączony bezpośrednio pomiędzy wejście zerowania procesora i +5 V. Odgięta jedna nóżka przycisku pozwala na łatwe dołączenie chwytaka resetu z symulatora EPROM.

Po włożeniu EPROM-u z przykładowym programem i podłączeniu zasilania 9..12VDC (pobór ok. 40 mA) powinien zaświecić się wskaźnik kanału 1 zaś po kilkusekundowym opóźnieniu wyświetlacz LCD powinien pokazać „0,0“. Przełączanie kanałów przyciskami nie zmienia tej wartości (wszystkie kanały są zerowane przy resecie). Teraz do linii transmisyjnej dołączamy nadajnik pętli prądowej - w rozwiązaniu przykładowym jest to prostu port szeregowy komputera sterowany przez niewielki program pracujący w środowisku Windows.

Mały pobór prądu sprawia, że możemy obciążyć bezpośrednio linię TxD portu - praktycznie każdy typowy port sobie z tym poradzi. Po uruchomieniu programu wyświetlacz będzie nadążać za wprowadzanymi przez nas wartościami temperatur. Rzecz jasna praktyczna przydatność tego testowego rozwiązania jest niewielka - sensu nabiera to z chwilą przesyłania przez nadajnik obiektowy wartości rzeczywiście mierzonych temperatur (czy też innych parametrów).

Jerzy Szczesiul, AVT

Oprogramowanie omówione w artykule jest dostępne na stronie www.ep.com.pl/ftp.

WYKAZ ELEMENTÓW

Rezystory
R1: 33Ω
R2: 150Ω

Kondensatory
C1, C2: 100nF ceramiczne SMD (0805)
C3, C4: 10nF ceramiczne SMD (0805)
C5: 100µF/25V

Półprzewodniki
D1, D2: 1N4148
U1: transoptor ICPL2601
U2: stabilizator LM317L
U3: stabilizator LM7805