STUDIA INFORMATICA 2001
Volume 22 Number 3 (45)

Dariusz CABAN
Politechnika Śląska, Instytut Informatyki

PROGRAMOWA REALIZACJA KOMUNIKACJI PRZEZ SYNCHRONICZNE INTERFEJSY SZEREGOWE

Streszczenie. W artykule przedstawiono zasadę działania synchronicznych interfejsów szeregowych: Microwire, SPI oraz I^2C, przeznaczonych do sprzęgania układów scalonych w systemach mikrokomputerowych. Wskazano możliwości realizacji komunikacji przez te interfejsy w sposób całkowicie programowy.

SOFTWARE REALISATION OF COMMUNICATION THROUGH SYNCHRONOUS SERIAL INTERFACES

Summary. This article presents principle of operation of synchronous serial interfaces: Microwire, SPI and I^2C, that enable connections among integrated circuits in microcomputer systems. Possibilities of entirely software realisation of communication through these interfaces are pointed out.

1. Wprowadzenie

Transmisja szeregowa, powszechnie wykorzystywana w sieciach komputerowych, znajduje również zastosowanie do realizacji komunikacji między jednostką centralną systemu mikrokomputerowego a układami zewnętrznymi. Konstruktorzy sprzętu mają do wyboru wiele układów o dostępie szeregowym: pamięci nieulotne EEPROM, sterowniki wyświetlaczy LED i LCD, zegary czasu rzeczywistego, przetworniki A/C i C/A, moduły rozszerzeń I/O, a nawet sterowniki sieciowe. W przeważającej liczbie przypadków są one wyposażone w interfejs synchroniczny.

Cechą charakterystyczną transmisji synchronicznej jest przesyłanie wraz z danymi sygnałów synchronizujących, umożliwiających odbiornikom poprawne rozeznawanie poszczegól-

nych bitów danych. Przy transmisji na niewielkie odległości stosuje się synchronizację za pomocą impulsów zegarowych dostarczanych po osobnej linii. Synchroniczne interfejsy szeregowe przeznaczone do realizacji komunikacji z układami o dostępie szeregowym pozwalają sprzęgać z jednostką centralną systemu wiele takich układów. Jednostka centralna pełni rolę układu nadrzędnego, układy zewnętrzne są układami podrzędnymi. Transmisję informacji przez interfejs może inicjować tylko układ nadrzędny i tylko on jest źródłem impulsów zegarowych.

Wśród istniejących rozwiązań synchronicznych interfejsów szeregowych dominującą pozycję zajmują trzy interfejsy [4]:

- Microwire, opracowany przez firmę National Semiconductor,
- SPI (ang. *Serial Peripheral Interface*), wprowadzony przez firmę Motorola,
- I^2C (ang. *Inter-Integrated Circuit*), powstały w firmie Philips.

Występujące pomiędzy nimi różnice to m. in. liczba linii potrzebnych do przesyłania informacji oraz sposób realizacji wyboru układu podrzędnego.

Sterowniki interfejsów synchronicznych stanowią element struktury niektórych mikrosterowników [5], w przypadku interfejsów: Microwire i I^2C dostępne są też sterowniki w postaci osobnych układów scalonych. Możliwa jest także wyłącznie programowa realizacja komunikacji z układami zewnętrznymi wyposażonymi w interfejs synchroniczny, co pozwala przy projektowaniu systemu uniezależnić ich dobór od zasobów posiadanych przez jednostkę centralną; realizacja taka została przedstawiona w niniejszym artykule.

2. Synchroniczne interfejsy szeregowe

2.1. Interfejs Microwire

Microwire to najstarszy z omawianych interfejsów synchronicznych [4]. Potocznie jest on nazywany interfejsem trójprzewodowym, co wynika z faktu, że do przesyłania informacji są w nim używane trzy linie:

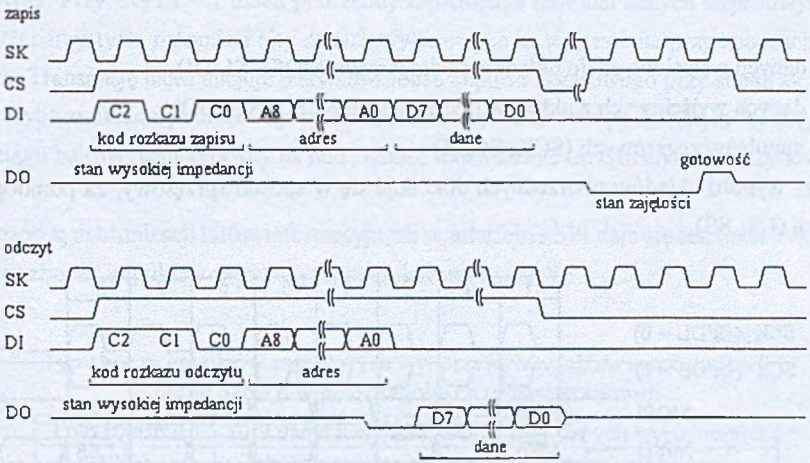
- danych wejściowych do układów podrzędnych (stosowane oznaczenia: SI, DI),
- danych wyjściowych z układów podrzędnych (SO, DO),
- impulsów zegarowych (SK, CLK).

Komunikacja z układami podrzędnymi wymaga jednak użycia większej liczby linii, gdyż do każdego takiego układu musi być jeszcze doprowadzona osobna linia wyboru (CS).

Do synchronizacji bitowej w interfejsie Microwire są wykorzystywane zbocza impulsów zegarowych, których częstotliwość może dochodzić do 2.1 MHz. Bity informacji wejściowej są wpisywane do wybranego układu podrzędnego przy narastających zboczach. Natomiast bity

danych wyjściowych mogą być odczytywane przez układ nadrzędny po zboczach narastających lub opadających – zależy to od konkretnego typu układu podrzędnego.

Układ nadrzędny inicjuje komunikację z wybranym układem podrzędnym poprzez wymuszenie stanu aktywnego na jego linii wyboru i wygenerowanie narastającego zbocza impulsu zegarowego. Na początku nadawany jest kod i ewentualne parametry rozkazu do wykonania przez układ podrzędny. Jeżeli wykonanie rozkazu wiąże się z odczytem lub zapisem danych, są one transmitowane w dalszej kolejności. Spotyka się też układy, z którymi komunikacja jest realizowana w trybie dwukierunkowym jednoczesnym. Zakończenie komunikacji sygnalizuje narastające zbocze impulsu zegarowego przy stanie nieaktywnym na linii wyboru układu podrzędnego.



Rys. 1. Przebiegi czasowe transmisji przez interfejs Microwire przy zapisie i odczycie zawartości komórki pamięci EEPROM o organizacji 512 x 8 bitów (Ci – i-ty bit kodu rozkazu, Ai – i-ty bit adresu, Di – i-ty bit danych)

Fig. 1. Timings of transmission through Microwire interface during writing to and reading from AT93C66 EEPROM with internal organization 512 x 8 bits (Ci – i-th operation code bit, Ai – i-th address bit, Di – i-th data bit)

Przesyłane kody i parametry rozkazów oraz dane mogą mieć różną długość, co pokazano na rysunku 1 przedstawiającym przebiegi czasowe występujące przy komunikacji z pamięcią EEPROM typu AT93C66, przy jej organizacji wewnętrznej ustalonej na 512 x 8 bitów [2]. Rozkaz zapisu pamięci EEPROM charakteryzuje się długim czasem wykonania, dochodzącym nawet do 10 ms. Wysyłane w tym czasie do pamięci rozkazy są ignorowane. Układ nadrzędny może po wysłaniu rozkazu zapisu, a przed kolejnym odwołaniem do pamięci, odmierzyć odcinek czasu o długości nieco przekraczającej podaną wartość. Istnieje także możliwość sprawdzenia bieżącego statusu wykonania tego rozkazu, co również pokazano na rysunku 1. Po

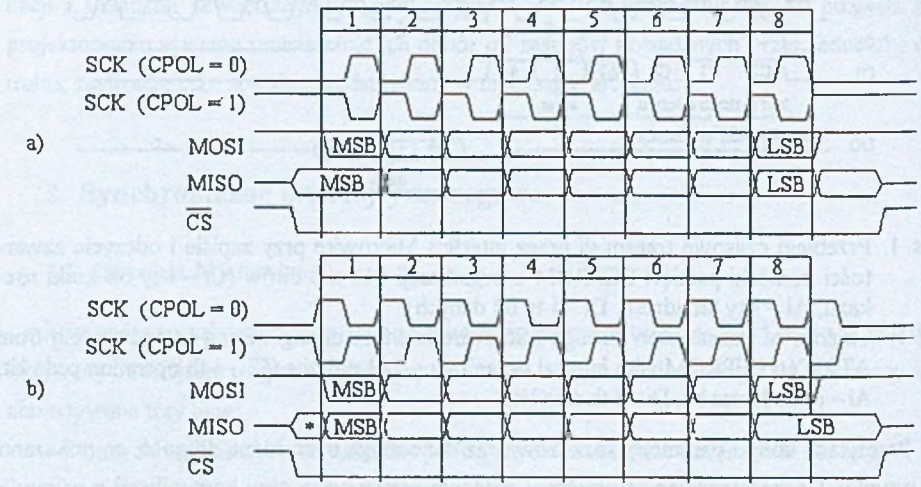
wysłaniu ostatniego bitu zapisywanej danej układ nadrzędny powinien ustawić na linii wyboru pamięci CS stan nieaktywny na czas co najmniej 250 ns. Informację statusową uzyskuje się, po upływie tego czasu, poprzez ponowne wymuszenie na linii wyboru stanu aktywnego. Stan niski na linii DO oznacza, że pamięć jest zajęta realizacją zapisu. Układ nadrzędny może wówczas wprowadzić linię CS w stan nieaktywny i powtórzyć test po pewnym czasie, może też, przy zachowanym stanie aktywnym na linii CS, biernie oczekiwać na zakończenie zapisu, co zostanie zasygnalizowane stanem wysokim na linii DO.

2.2. Interfejs SPI

Tak samo jak w Microwire, w interfejsie SPI do przesyłania informacji są używane trzy linie:

- danych wejściowych do układów podrzędnych (MOSI, SI, DI),
- danych wyjściowych z układów podrzędnych (MISO, SO, DO),
- impulsów zegarowych (SCK, SCLK).

Również wyboru układów podrzędnych dokonuje się w sposób sprzętowy, za pomocą osobnych linii (CS, SS).



Rys. 2. Przebiegi czasowe transmisji bajtu przez interfejs SPI przy a) CPHA = 0, b) CPHA = 1 (* - stan niezdefiniowany, z reguły jest to LSB poprzednio przesłanego bajtu)

Fig. 2. Timings of byte transmission through SPI interface when a) CPHA = 0, b) CPHA = 1 (* - undefined state, usally it's LSB of byte previously sent)

Do synchronizacji bitowej w interfejsie SPI wykorzystuje się zbocza impulsów zegarowych, których maksymalna częstotliwość wynosi 2.1 MHz. Sposób realizacji tej synchroniza-

cji zależy od trybu transmisji opisanego parą liczb dwójkowych (CPOL,CPHA). Parametr CPOL (ang. *Clock POLarity*) definiuje stan linii impulsów zegarowych przed rozpoczęciem i po zakończeniu transmisji. Parametr CPHA (ang. *Clock PHAse*) określa, którymi ze zboczy impulsów zegarowych są synchronizowane bity informacyjne oraz zasadę wytwarzania sygnału wyboru układu podrzędnego. Na rysunku 2 przedstawiono przebiegi czasowe transmisji bajtu dla poszczególnych trybów. Przy CPHA = 0 układ podrzędny próbkuje linię MOSI przy zboczach nieparzystych (zbocza są numerowane od 1), natomiast bity danych wyjściowych są przez ten układ wyprowadzane na linię MISO przy zboczach parzystych. Układ nadrzędny sygnalizuje rozpoczęcie transmisji poprzez ustawienie stanu aktywnego na linii wyboru układu podrzędnego, a po zakończeniu transmisji bajtu linia ta powinna zostać wprowadzona w stan nieaktywny. Przy CPHA = 1 układ podrzędny zapamiętuje stan linii danych wejściowych przy zboczach parzystych, natomiast bity danych wyjściowych są wystawiane przy zboczach nieparzystych. Transmisję bajtu inicjuje pierwsze zbocze impulsu zegarowego przy stanie aktywnym na linii wyboru układu podrzędnego; jeśli komunikacja z układem podrzędnym wymaga przesłania ciągu bajtów, stan aktywny na linii wyboru może zostać utrzymany pomiędzy kolejnymi bajtami.

Sposób synchronizacji bitów informacyjnych w interfejsie SPI daje się też opisać za pomocą typów zboczy impulsów zegarowych, co pokazano w tabeli 1.

Tabela 1

Typ zboczy impulsów zegarowych wykorzystywanych do synchronizacji bitów danych w poszczególnych trybach transmisji

Tryb transmisji	Bity danych wejściowych	Bity danych wyjściowych
(0,0)	zbocza narastające	zbocza opadające
(1,0)	zbocza opadające	zbocza narastające
(0,1)	zbocza opadające	zbocza narastające
(1,1)	zbocza narastające	zbocza opadające

Zdarzają się odstępstwa od zasady, że przy CPHA = 0 po zakończeniu transmisji bajtu należy wprowadzić linię wyboru układu podrzędnego w stan nieaktywny. Większość z produkowanych układów podrzędnym jest przystosowana do pracy w trybach: (0,0) i (1,1), a komunikacja z nimi wymaga przesłania przeważnie więcej niż jednego bajtu. Z zamieszczonych w kartach katalogowych tych układów przebiegów czasowych komunikacji wynika, że niezależnie od wybranego trybu transmisji linia wyboru musi pozostać w stanie aktywnym przez cały czas trwania komunikacji.

Protokół komunikacji z układami podrzędnymi wyposażonymi w interfejs SPI jest taki sam, jak w przypadku układów z interfejsem Microwire. Transmitowane kody i parametry rozkazów oraz dane mogą mieć różną długość, przy czym stanowią one zawsze całkowitą wielokrotność bajtu.

2.3. Interfejs I^2C

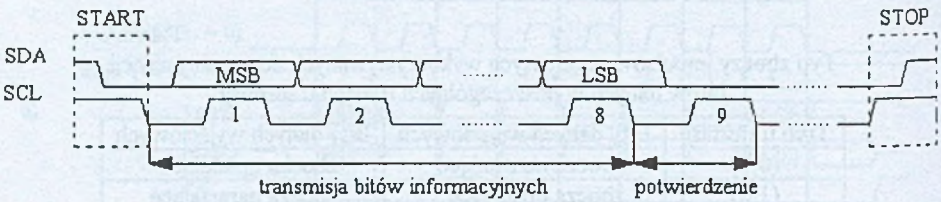
W interfejsie I^2C do przesyłania informacji wykorzystuje się tylko dwie linie:

- dwukierunkową linię danych (SDA),
- impulsów zegarowych (SCL).

Wybór układów podrzędnych jest realizowany poprzez adresowanie.

Obie linie tworzące magistralę interfejsu powinny być dołączone do napięcia zasilania przez oporniki podciągające. Układy z interfejsem I^2C mają wyjścia typu "otwarty kolektor" lub "otwarty dren", stąd na liniach interfejsu jest realizowana funkcja "iloczynu na drucie". Dzięki takiej konstrukcji magistrali osiągnięto dwa cele:

- możliwość przyłączenia do magistrali wielu układów nadrzędnych. Istnieje przez to ryzyko wystąpienia konfliktów przy próbie zajęcia magistrali przez kilka układów nadrzędnych, ale dla ich rozwiązywania konstruktorzy interfejsu I^2C przewidzieli odpowiednią procedurę arbitrażową;
- źródłem impulsów zegarowych, niezależnie od kierunku transmisji, jest układ nadrzędny, a ich częstotliwość określa prędkość transmisji informacji. Układy podrzędne mogą jednak w razie potrzeby spowalniać transmisję poprzez wydłużenie czasu trwania stanu niskiego impulsów zegarowych.



Rys. 3. Ilustracja warunków START i STOP oraz przebieg transmisji bajtu przez interfejs I^2C
Fig. 3. START and STOP conditions and timing of byte transmission through I^2C interface

W interfejsie I^2C, odmiennie niż w dotychczas omówionych interfejsach, bity informacyjne są synchronizowane nie zboczami, lecz poziomem impulsów zegarowych. Stan na linii SDA może się zmieniać, z wyjątkiem dwóch przypadków, tylko podczas stanu niskiego na linii SCL. Zmiany przy stanie wysokim na tej linii mają specjalne znaczenie (rysunek Rys. 3):

- zmiana 1 → 0 oznacza rozpoczęcie transmisji i jest nazywana warunkiem START,
- zmiana 0 → 1 oznacza zakończenie transmisji i nosi nazwę warunku STOP.

Maksymalna częstotliwość impulsów zegarowych wynosi, w zależności od typu układu podrzędnego, 100 lub 400 kHz.

Informacja po linii SDA jest przesyłana w postaci bajtów, a pierwszym nadawanym bitem jest zawsze bit najbardziej znaczący (rysunek Rys. 3). Do przesłania bajtu potrzeba jednak

dziewięciu impulsów zegarowych. Czas wyznaczony przez osiem impulsów zajmuje transmisja bitów informacyjnych. W czasie trwania dziewiątego impulsu odbiorca powinien potwierdzić odbiór bajtu; nadawca ustawia wówczas na linii SDA stan wysoki, a potwierdzenie jest realizowane poprzez wymuszenie na tej linii stanu niskiego.

Komunikacja przez interfejs I^2C rozpoczyna się od warunku START i kończy warunkiem STOP, pomiędzy którymi można przesłać dowolną liczbę bajtów. Dopuszczalne jest powtarzanie warunku START i jest to wykorzystywane do zmiany kierunku transmisji. Po każdym warunku START jako pierwszy przesyłany jest adres układu podrzędnego wraz z bitem określającym kierunek przepływu informacji. Brak potwierdzenia odbioru bajtu z adresem może oznaczać, że:

- podano błędny adres,
- wskazany układ podrzędny nie istnieje,
- wskazany układ podrzędny istnieje, ale jest zajęty realizacją wewnętrznych operacji (np. w przypadku pamięci EEPROM trwa jej zapis).

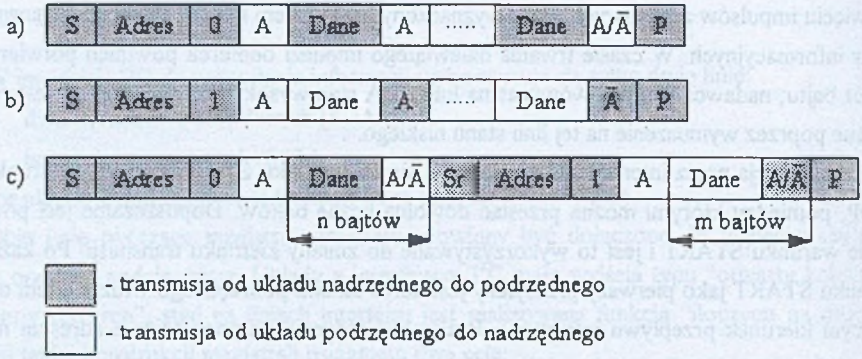
Przyczyny braku potwierdzenia przy transmisji następnych bajtów są różne, zależy to od typu odbiorcy. Jeżeli odbiorcą jest układ podrzędny, oznacza to niemożność przyjęcia przez niego danych. Układ nadrzędny powinien wówczas przerwać nadawanie i wygenerować warunek STOP. Jeżeli odbiorcą jest układ nadrzędny, przez brak potwierdzenia sygnalizuje on zakończenie odczytu informacji z układu podrzędnego. Układ podrzędny zwalnia wtedy linię SDA, co umożliwi układowi nadrzędnemu zakończenie komunikacji lub powtórzenie warunku START.

W interfejsie I^2C podstawowym trybem adresowania jest tryb 7-bitowy. Produkowane są również układy adresowane w trybie 10-bitowym. Adresy dla poszczególnych rodzajów układów podrzędnych są przydzielane przez komitet interfejsu I^2C. W niektórych przypadkach adres składa się z części stałej oraz programowalnej przez użytkownika. Rozmiar części programowalnej określa, ile maksymalnie układów danego rodzaju może zostać przyłączonych do wspólnej magistrali.

Przewidzianych jest kilka wariantów współpracy układu nadrzędnego z układami podrzędnymi. Warianty te różnią się strukturą przesyłanej informacji i w terminologii stosowanej przez firmę Philips noszą nazwę formatów transmisji. Wyróżnia się trzy formaty:

- zapis do układu podrzędnego,
- odczyt z układu podrzędnego,
- format kombinowany, pozwalający na zmianę kierunku przepływu informacji lub zaadresowanie innego układu podrzędnego.

Przebieg transmisji dla poszczególnych formatów przy 7-bitowym trybie adresowania przedstawiono na rysunku 4. Analogiczne przebiegi przy trybie 10-bitowym można znaleźć w [6].



Rys. 4. Formaty transmisji przez interfejs I^2C: a) zapis do układu podrzędnego, b) odczyt z układu podrzędnego, c) format kombinowany (S – warunek START, Sr – powtórzony warunek START, P – warunek STOP, A – potwierdzenie odbioru, $\overline{A}$ - brak potwierdzenia odbioru)

Fig. 4. Formats of transmission through I^2C interface: a) writing to slave device, b) reading from slave device, c) combined format (S – START condition, Sr – repeated START condition, P – STOP condition, A – acknowledgment, $\overline{A}$ - negative acknowledgment)

3. Programowa realizacja komunikacji z układami wyposażonymi w interfejs synchroniczny

W omówionych w poprzednim rozdziale interfejsach synchronicznych źródłem impulsów zegarowych, wykorzystywanych do synchronizacji bitów danych, jest zawsze układ nadrzędny. Impulsom tym nie stawia się rygorystycznych wymagań czasowych, w szczególności ich częstotliwość i wypełnienie mogą się zmieniać w trakcie trwania transmisji. Dzięki temu komunikację z układami podrzędnymi z interfejsami: Microwire, SPI oraz I^2C daje się zrealizować także bez użycia specjalizowanych sterowników transmisji, w sposób wyłącznie programowy.

Komunikacja z układem podrzędnym odbywa się zgodnie z określonym protokołem, opisanym w karcie katalogowej układu. Na przykład odczyt lub zapis komórki pamięci EEPROM typu AT93C66 wymaga przesłania trzech ciągów bitów, reprezentujących kolejno: kod rozkazu, adres komórki oraz daną (rysunek 1). Każdy z ciągów wejściowych dla układu podrzędnego, niezależnie od swej długości, struktury i znaczenia, jest przesyłany według tych samych zasad, podobnie jak każdy z ciągów wyjściowych. Przy wyłącznie programowej realizacji komunikacji przez interfejs Microwire podstawowymi modułami funkcjonalnymi oprogramowania będą moduły umożliwiające:

- wysłanie do układu podrzędnego ciągu o długości l_{in} bitów,
- odbiór z układu podrzędnego ciągu o długości l_{out} bitów
- jeśli komunikacja wymaga transmisji w trybie dwukierunkowym jednoczesnym – wysłanie ciągu l_{in}-bitowego połączone z odbiorem ciągu l_{out}-bitowego.

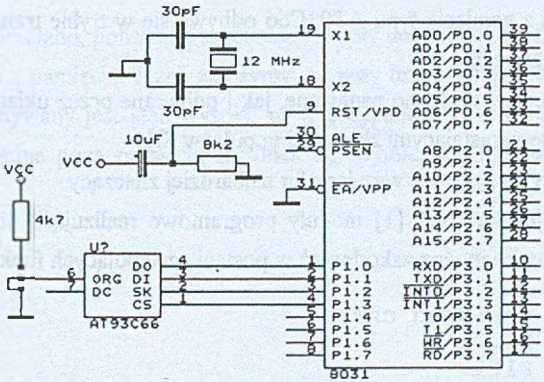
Niezbędne są jeszcze moduły realizujące operacje rozpoczęcia i zakończenia komunikacji z układem podrzędnym. Za pomocą tych modułów oraz wymienionych modułów transmisyjnych można następnie zaimplementować moduły komunikacji z dowolnym typem układu podrzędnego.

Przy programowej realizacji komunikacji przez interfejs SPI zestaw podstawowych modułów funkcjonalnych jest taki sam, jak dla interfejsu Microwire, przy czym długość przesyłanych ciągów jest zawsze całkowitą wielokrotnością bajtu. W przypadku komunikacji przez interfejs I^2C zestaw taki będzie obejmował moduły umożliwiające:

- generację warunku START,
- generację warunku STOP,
- wysłanie bajtu do układu podrzędnego wraz z generacją dodatkowego impulsu zegarowego na potwierdzenie odbioru,
- odbiór bajtu z układu podrzędnego,
- potwierdzenie odbioru bajtu przez układ nadrzędny,
- niepotwierdzenie odbioru bajtu przez układ nadrzędny.

Należy zaznaczyć, że zastosowanie tego typu realizacji komunikacji przez interfejs I^2C lepiej ograniczyć do sytuacji, gdy w systemie jest tylko jeden układ nadrzędny; w praktyce sytuacja taka występuje jednak najczęściej.

W dalszej części artykułu zostanie przedstawione oprogramowanie umożliwiające komunikację mikrosterownika 8031 z pamięcią EEPROM typu AT93C66, wyposażoną w interfejs



Rys. 5. Sposób połączenia pamięci EEPROM typu AT93C66 z mikrosterownikiem 8031
Fig. 5. Connections between AT93C66 EEPROM and 8031 microcontroller

Microwire. Przykładowy sposób połączenia mikrosterownika z pamięcią pokazano na rysunku 5. Pojemność pamięci AT93C66 wynosi 4 kilobity, a jej organizacja wewnętrzna jest ustalana poprzez wymuszenie odpowiedniego stanu logicznego na wejściu ORG. Przy stanie wysokim na tym wejściu pamięć ma organizację 256 x 16 bitów, przy stanie niskim – 512 x 8 bitów. Do operacji na pamięci służy siedem rozkazów (tabela 2). Jak wynika z tabeli, w przypadku niektórych rozkazów ciąg nazywany adresem jest w rzeczywistości rozszerzeniem kodu rozkazu. Produkowanych jest kilka wersji tego układu, różniących się zakresem dopuszczalnych napięć zasilających [2].

Tabela 2

Format rozkazów pamięci EEPROM typu AT93C66 (x – wartość bitu nieistotna)

Rozkaz	Kod	Adres		Dane		Działanie
		512 x 8	256 x 16	512 x 8	256 x 16	
READ	110	$A_8 - A_0$	$A_7 - A_0$	$D_7 - D_0$	$D_{15} - D_0$	odczyt zawartości komórki o podanym adresie
EWEN	100	11xxxxxxx	11xxxxxx	–	–	odblokowanie zapisu i kasowania pamięci
ERASE	111	$A_8 - A_0$	$A_7 - A_0$	–	–	kasowanie zawartości komórki o podanym adresie
WRITE	101	$A_8 - A_0$	$A_7 - A_0$	$D_7 - D_0$	$D_{15} - D_0$	zapis komórki o podanym adresie
ERAL	100	10xxxxxxx	10xxxxxx	–	–	kasowanie zawartości całej pamięci
WRAL	100	01xxxxxxx	01xxxxxx	$D_7 - D_0$	$D_{15} - D_0$	zapełnienie pamięci wzorcem
EWDS	100	00xxxxxxx	00xxxxxx	–	–	zablokowanie zapisu i kasowania pamięci

Na rysunku 1 przedstawiono przebiegi czasowe transmisji dla rozkazów: WRITE oraz READ. Jak łatwo zauważyć:

- komunikacja z pamięcią typu AT93C66 odbywa się w trybie transmisji dwukierunkowej niejednoczesnej,
- bity informacyjne, zarówno nadawane, jak i odbierane przez układ nadrzędny, są synchronizowane narastającymi zboczami impulsów SK,
- jako pierwszy jest zawsze wysyłany bit najbardziej znaczący.

Przy zastosowaniu języka C-51 [1] moduły programowe realizujące przesyłanie informacji przez interfejs Microwire można zakodować w postaci następujących funkcji:

```
#define uchar unsigned char
#define DO    P1.0
#define DI    P1.1
#define SK    P1.2
```

```
/* wysłanie ciągu o maksymalnej długości 8 bitów; maska
   powinna mieć ustawiony bit na pozycji pierwszego wysyłanego
   bitu ciągu */

void send(uchar sequence, uchar mask)
{
  while(mask)
  {
    if (sequence & mask)    /* wartość wysyłanego bitu */
      DI = 1;
    else
      DI = 0;
    mask >>= 1;             /* pozycja kolejnego bitu */
    SK = 1;
    SK = 0;
  }
}

/* odbiór ciągu o maksymalnej długości 8 bitów; maska powinna
   mieć ustawiony bit na pozycji pierwszego odbieranego bitu
   ciągu */

uchar receive(uchar mask)
{
uchar sequence = 0;

  while(mask)
  {
    SK = 1;
    _opc(0);                /* rozkaz NOP */
    if (DO)                 /* odczyt bitu */
      sequence |= mask;
    SK = 0;
    mask >>= 1;             /* pozycja kolejnego bitu */
  }
  return(sequence);
}
```

Jak wcześniej wspomniano, potrzebne są jeszcze moduły umożliwiające zainicjowanie i zakończenie komunikacji z pamięcią. Jeżeli założymy, że przy braku komunikacji na linii impulsów zegarowych utrzymywany jest stan wysoki, to w przypadku, gdy w systemie nie występują inne układy podrzędne poza pamięcią (rysunek 5), wymienione operacje będą realizowane przez funkcje:

```
#define CS P1.3

void start(void)
{
  SK = 0;
  CS = 1;
}
```

```
void stop(void)
{
  CS = 0;
  SK = 1;
}
```

Zerowanie mikrosterownika 8031 powoduje ustawienie na jego liniach wejścia/wyjścia stanu wysokiego, stąd w części inicjalizacyjnej oprogramowania systemu powinna zostać wykonana instrukcja `CS = 0`.

Przy użyciu powyższych funkcji można w prosty sposób zaimplementować funkcje realizujące rozkazy podane w tabeli 2. Ograniczymy się tu do rozkazów odczytu i zapisu pamięci, przy jej organizacji 512 x 8 bitów:

```
#define uint  unsigned int
#define READ  0x06
#define WRITE 0x05

uchar read(uint address)
{
uchar value;

  start();
  send(READ, 0x04);
  send((uchar)(address >> 8), 0x01);
  send((uchar)address, 0x80);
  value = receive(0x80);
  stop();
  return(value);
}

void write(uint address, uchar value)
{
  start();
  send(WRITE, 0x04);
  send((uchar)(address >> 8), 0x01);
  send((uchar)address, 0x80);
  send(value, 0x80);
  stop();

  busy();           /* oczekiwanie na zakończenie zapisu */
}
```

Wykonanie rozkazu WRITE jest czasochłonne - zapis komórki pamięci typu AT93C66 może trwać do 10 ms. W podanym przykładzie przyjęto, że mikrosterownik biernie oczekuje na zakończenie zapisu:

```
void busy(void)
{
  CS = 1;
  _opc(0);        /* rozkaz NOP */
```

```
  while(!DO);
  CS = 0;
}
```

Funkcja ta musi być również używana przy realizacji rozkazów: ERASE, ERAL i WRAL.

Kod wynikowy podanych funkcji oraz funkcji realizujących pozostałe rozkazy z tabeli 2, po skompilowaniu przy ustawionym modelu pamięci COMPACT, zajmuje 361 bajtów pamięci programu, a zapotrzebowanie na wewnętrzną pamięć danych wynosi zaledwie kilka bajtów [1]. Oprogramowanie to zostało przetestowane w rzeczywistym systemie mikrokomputerowym, w którym zastosowano mikrosterownik 8031 w jego podstawowej wersji, taktowany impulsami o częstotliwości 12 MHz. Testy wykazały poprawne działanie zaimplementowanego oprogramowania.

W posiadaniu autora niniejszego artykułu znajduje się również kod źródłowy oprogramowania umożliwiającego komunikację między mikrosterownikiem 8031 a układami podrzędnymi z interfejsami: SPI oraz I^2C.

4. Podsumowanie

Zastosowanie transmisji szeregowej do realizacji komunikacji między jednostką centralną systemu mikrokomputerowego a układami zewnętrznymi przynosi wiele korzyści. Pozwala to zredukować liczbę połączeń w systemie, co z kolei upraszcza konstrukcję obwodów drukowanych i zmniejsza ich rozmiary, podnosi niezawodność systemu. W pewnych przypadkach użycie układów zewnętrznych o dostępie szeregowym wręcz umożliwia współpracę jednostki centralnej z otoczeniem. Większość z produkowanych układów zewnętrznych o dostępie szeregowym jest wyposażona w interfejs synchroniczny.

Komunikacja z układami zewnętrznymi przez interfejs synchroniczny może być realizowana za pomocą specjalizowanego sterownika transmisji lub całkowicie programowo. W niniejszym artykule skupiono się na drugim z wymienionych sposobów. Jako przykład przedstawiono oprogramowanie umożliwiające komunikację mikrosterownika 8031 z pamięcią EEPROM z interfejsem Microwire. Oprogramowanie to nie jest skomplikowane, a jego wymagania pamięciowe, nawet po zakodowaniu w języku wysokiego poziomu, też nie są wygórowane.

Na zakończenie warto wspomnieć o nowym zastosowaniu jednego z zaprezentowanych w artykule interfejsów, a mianowicie interfejsu SPI. Jest to szeregowe programowanie mikrosterowników z wewnętrzną pamięcią programu typu FLASH. Firma Atmel oferuje mikrosterowniki: AT89S8252 (rodzina MCS-51) oraz serii AT90S (rodzina AVR), wyposażone m.in. w sprzętowy sterownik interfejsu SPI, za którego pośrednictwem można również dokonać zmiany zawartości pamięci programu [3]. Programowanie szeregowe tych mikrosterowników jest

możliwe także po ich umieszczeniu w systemie docelowym. Do realizacji programowania potrzebny jest tylko port drukarki mikrokomputera IBM PC oraz odpowiedni program ładujący. Użytkownik może samodzielnie zaimplementować program ładujący, co nie jest zadaniem szczególnie trudnym, ale czasochłonnym; alternatywą jest wykorzystanie jednego z gotowych programów, udostępnianych bezpłatnie w sieci Internet.

LITERATURA

1. Archimedes Software: C-51 Compiler. July 1991.
2. Atmel Corp.: AT93C46/56/57/66. 3-wire Serial EEPROMs. http://www.atmel.com/.
3. Caban D.: 8-bitowe mikrosterowniki z pamięcią FLASH programowalne przez synchroniczny interfejs szeregowy SPI. Techniki Komputerowe 2/2000, Instytut Maszyn Matematycznych, Warszawa 2000.
4. Eck A.: Serial Interface for Embedded Design. Circuit Cellar Ink Online, January 2000, http://www.circuitcellar.com/online/.
5. Pełka R.: Mikrokontrolery. Architektura, programowanie, zastosowania. WKŁ, Warszawa 1999.
6. Philips Semiconductors: The I^2C-bus and how to use it (including specifications). April 1995, http://www.semiconductors.philips.com/.

Recenzent: Dr inż. Włodzimierz Boroń

Wpłynęło do Redakcji 15 marca 2001 r.

Abstract

Using of serial transmission to accomplish communication among central processor and peripheral devices in a microcomputer system is very useful. It reduces number of connections in the system, thus printed circuit boards can be simpler and smaller, it also increases system's reliability. In some cases application of serial peripheral devices simply enables interaction between processor and outside world. Most of such devices are equipped with synchronous interface.

There're three main types of synchronous serial interfaces: Microwire, SPI and I^2C. They differ in, among other things, number of lines required to transfer data. Principle of operation

of mentioned interfaces is described in this article. Timings of transmission through each of them are presented on figures 1÷4.

Synchronous serial transmission can be supported by dedicated controller or realised entirely by software. In this article special attention was paid to software realisation. Software enabling 8031 microcontroller to communicate with AT93C66 EEPROM is given as an example. Possible connections between them are shown on figure 5. AT93C66 memory is equipped with Microwire interface and is controlled by seven instructions. These instructions are listed in table 2. The communication software isn't complicated, also its requirements concerning code and data memory aren't too excessive.