ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2000
Seria: INFORMATYKA z. 38 Nr kol. 1451

Dariusz CABAN
Instytut Informatyki Teoretycznej i Stosowanej, PAN

UWAGI DO DOKUMENTACJI PROTOKOŁU MODBUS

Streszczenie. W artykule wskazano na sprzeczności i braki w dokumentacji protokołu Modbus. Szczególną uwagę poświęcono wyznaczaniu sumy kontrolnej CRC.

REMARKS ON MODBUS PROTOCOL DOCUMENTATION

Summary. In this article contradictions and lacks in Modbus protocol documentation were indicated. Special attention was paid to CRC calculation.

1. Wprowadzenie

Od opracowania protokołu Modbus upłynęło ponad ćwierć wieku [2], nadal jednak jest on często stosowany. Autor publikacji [3] podaje, że protokół Modbus, a w szczególności jego tryb RTU (ang. *Remote Terminal Unit*), jest typowym protokołem wykorzystywanym dla organizacji komunikacji regulatorów i programowalnych sterowników logicznych z komputerem nadrzędnym (tzw. komunikacji pionowej). W procedury komunikacyjne realizujące ten protokół wyposażone są niemal wszystkie pakiety SCADA (ang. *Supervisory Control and Data Acquisition*). W podsumowaniu wspomnianej publikacji, zawierającym kryteria wyboru regulatora, znajduje się wskazówka, że podstawowym typem protokołu komunikacji pomiędzy regulatorem a komputerem nadrzędnym powinien być Modbus RTU.

Protokół Modbus swą dużą popularność - wkrótce po opracowaniu został przyjęty przez większość znanych producentów sterowników przemysłowych - zawdzięcza prostocie zastosowanych w nim rozwiązań oraz dostępności dokumentacji. W łączu komunikacyjnym wykorzystującym ten protokół, sprzęgającym wiele urządzeń pomiarowo-kontrolnych, informacja jest przesyłana w postaci znaków (transmisja asynchroniczna). Dostęp do łącza odbywa się

według prostej reguły polecenie - odpowiedź (ang. *query - response*), gwarantującej brak konfliktów przy poprawnie skonfigurowanym łączu (w dowolnej chwili tylko jedno z urządzeń transmituje informację). Budowa łącza od strony sprzętowej jest bardzo prosta - wystarcza układ interfejsu umożliwiającego zestawianie łączy wielopunktowych, np. szeroko obecnie stosowanego standardu RS485 oraz sterownik asynchronicznej transmisji szeregowej [4]. Istnieje liczna grupa sterowników scalonych, jak również mikrokontrolerów, które posiadają wbudowane sterowniki transmisji asynchronicznej [7]. W sposób programowy realizowane są takie funkcje protokołu Modbus, jak: dostęp do łącza, formowanie ramek, zabezpieczanie informacji przed skutkami błędów transmisji, potwierdzanie poprawnego odbioru informacji, zabezpieczanie przed zawieszeniem pracy łącza oraz usługi świadczone programom użytkowym, wykonywanym w urządzeniach sprzęgniętych łączem. Najbardziej pracochłonna w implementacji jest ostatnia z wymienionych funkcji.

Podstawowym źródłem informacji dla projektanta oprogramowania protokołu Modbus jest dokumentacja *Modicon Modbus Protocol. Reference Guide. PI-MBUS-300 Rev. D.* [1]. Zawiera ona jednak pewne drobne sprzeczności oraz braki. Niniejsza publikacja to wynik własnych doświadczeń autora nabytych w trakcie praktycznej realizacji oprogramowania komunikacyjnego, wykonanego w ramach współpracy z jednym z przedsiębiorstw produkujących aparaturę pomiarową.

2. Zależności czasowe w protokole Modbus RTU

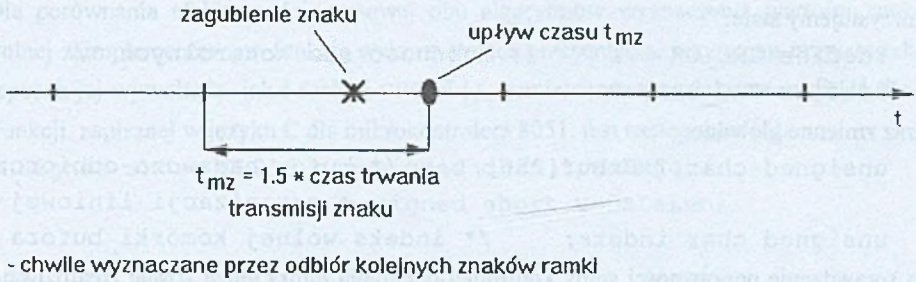
W dokumentacji [1] podane są dwa typy zależności czasowych występujących w trybie RTU protokołu Modbus:

- odstęp między kolejnymi ramkami t_{mr},
- odstęp między kolejnymi znakami t_{mz}.

Odstęp między kolejnymi ramkami powiniem wynosić co najmniej t_{mr} = *3.5 * czas trwania transmisji znaku*, w trakcie którego w łączu nie powinna być prowadzona transmisja (tzw. cisza w łączu). Odstęp pomiędzy kolejnymi znakami nie powienien przekraczać wartości t_{mz} = *1.5 * czas trwania transmisji znaku*. Jeżeli ten odstęp zostanie przekroczony, odbiorca uzna ramkę za niekompletną i następny znak zostanie zinterpretowany jako zawartość pola adresowego nowej ramki. Jeżeli nowa ramka pojawi się przed upływem czasu t_{mr}, to zostanie ona potraktowana jako dalszy ciąg poprzedniej ramki. Takie skutki naruszenia zależności czasowych podano w dokumentacji [1].

Jeżeli wszystkie znaki ramki odebrano poprawnie, po ostatnim znaku zostanie najpierw wykryte przekroczenie czasu t_{mz}. W tym przypadku nie ma jednak podstaw do uznania ramki

za niekompletną. Nowy znak nie pojawi się w łączu przed upływem wymaganego odstępu między kolejnymi ramkami, a sprawdzenie sumy kontrolnej wykaże poprawność całej ramki.



Rys. 1. Naruszenie zależności czasowych spowodowane zagubieniem znaku ramki
Fig. 1. Disturbing of time dependencies caused by loss of a character in a frame

Rozpatrzmy sytuację, kiedy jeden ze znaków ramki został zagubiony. Zakładając, że nadawca wysyła znaki w sposób ciągły, odstęp czasu między kolejnymi znakami wyniesie w tym przypadku t_{kz} = *2 * czas trwania transmisji znaku* (rysunek 1.). Ponieważ $t_{mz} < t_{kz} < t_{mr}$, drugi z odebranych znaków zostanie potraktowany jednocześnie jako zawartość pola adresowego nowej ramki i kontynuacja poprzedniej.

Wydaje się prawdopodobne, że w praktyce stosuje się tylko odmierzanie odstępu czasu t_{mr}. Niekompletność ramki spowodowana zagubieniem znaku (-ów) zostanie wykryta na etapie sprawdzania poprawności sumy kontrolnej.

3. Wyznaczanie sumy kontrolnej w protokole Modbus RTU

W ramce umieszczana jest wiadomość złożona z adresu odbiorcy, kodu funkcji i ewentualnie danych. Zawartość wiadomości jest zabezpieczana 16-bitową sumą kontrolną, wyznaczaną za pomocą wielomianu generacyjnego CRC-16 [5] i umieszczaną na końcu ramki, przy czym jako pierwszy występuje młodszy bajt sumy. W dokumentacji protokołu [1] podany jest algorytm wyznaczania sumy kontrolnej za pomocą metody wykorzystującej przesunięcia i operację sumy modulo 2. Ponieważ obliczanie wartości CRC przy zastosowaniu tego algorytmu trwa stosunkowo długo, zwłaszcza przy dużych rozmiarach wiadomości, w dokumentacji protokołu [1] podano również gotową funkcję w języku C, będącą zapisem algorytmu wykorzystującego tablice zawierające tzw. wielomiany modyfikujące [6]. Funkcja `CRC16()`, której parametrami wywołania są: adres początkowy obszaru pamięci z zapisaną wiadomością oraz rozmiar wiadomości, zwraca wartość 16-bitową za pomocą instrukcji:

```
return(uchCRCHi << 8 | uchCRCLo);
```

gdzie uchCRCHi i uchCRCLo oznaczają odpowiednio starszy i młodszy bajt sumy, przechowywane w zmiennych lokalnych funkcji. Jeżeli w oprogramowaniu komunikacyjnym wykorzystujemy stałe:

```
#define CRC_OK   1      /* zgodność sum kontrolnych */
#define CRC_BAD  0
```

oraz zmienne globalne:

```
unsigned char TxRxBuf[256];      /* bufor nadawczo-odbiorczy
                                    o organizacji liniowej */
unsigned char indeks;      /* indeks wolnej komórki bufora */
```

to sprawdzenie poprawności sumy kontrolnej odebranej ramki może zostać zrealizowane za pomocą następującej funkcji (w chwili jej wywołania zmienna indeks zawiera liczbę bajtów ramki):

```
unsigned char checkCRC(void)
{
  if (((TxRxBuf[indeks - 1] << 8) | TxRxBuf[indeks - 2]) ==
        CRC16(TxRxBuf,indeks - 2))
    return(CRC_OK);
  else
    return(CRC_BAD);
}
```

W zaimplementowanym przez autora niniejszej publikacji oprogramowaniu komunikacyjnym dla jednostki podrzędnej sprawdzanie zgodności sum: odebranej i wyliczonej odbywało się w podany wyżej sposób. Do testowania komunikacji wykorzystano odpowiednie oprogramowanie jednostki nadrzędnej (wersja demonstracyjna tego oprogramowania jest dostępna w sieci Internet na stronie http://www.win-tech.com/), działające na mikrokomputerze IBM PC w środowisku systemu operacyjnego WINDOWS 95. Pierwsze próby przeprowadzenia transakcji odczytu zawartości grupy rejestrów okazały się nieudane, pomimo poprawnego zaadresowania jednostki podrzędnej komunikacja pomiędzy nią a jednostką nadrzędną nie została nawiązana. Oprogramowanie jednostki podrzędnej było uruchamiane za pomocą emulatora układowego [9], który umożliwia m. in. podgląd zawartości zmiennych oraz pracę krokową programu. Okazało się, że ramki gromadzone w buforze transmisyjnym jednostki podrzędnej miały prawidłową postać, ale były one odrzucane z powodu niezgodności sum. Po zastosowaniu algorytmu wyznaczania sumy kontrolnej za pomocą przesunięć komunikacja odbywała się w sposób poprawny. Porównanie zawartości bajtów sumy kontrolnej dla obu algorytmów wykazało, że funkcja CRC16() wyznacza wartość sumy prawidłowo, ale w wyniku zwracanym przez funkcję zamieniona jest kolejność bajtów. W dokumentacji protokołu [1] brak jednak o tym jakiejkolwiek wzmianki. Sposobem na uzyskanie właściwej kolejności

bajtów jest zmiana ostatniej instrukcji funkcji `CRC16()` na:

```
return(uchCRCLo << 8 | uchCRCHi);
```

i opatrzenie jej stosownym komentarzem.

Dla porównania efektywności czasowej obu algorytmów wyznaczania wartości sumy kontrolnej zaimplementowano funkcję wykorzystującą przesunięcia, przy czym przyjęto taki sam sposób jej wywołania jak i funkcji `CRC16()`, zamieszczonej w dokumentacji [1]. Postać funkcji, zapisanej w języku C dla mikrokontrolera 8051, jest następująca [8]:

```
unsigned short CRC16_p(unsigned char *puchMsg,
                       unsigned short usDataLen)
{
unsigned short usCRC = 0xFFFF;
bit            shbit;       /* wartość wysuwanego bitu */
unsigned char  uchshift;    /* licznik przesunięć */

  while(usDataLen--)
  {
      usCRC ^= *puchMsg++;
      uchshift = 8;
      while(uchshift--)
      {
          shbit = (usCRC & 1) ? 1 : 0;
          usCRC >>= 1;
          if (shbit)
              usCRC ^= 0xA001;
      }
  }
  return(usCRC);
}
```

Funkcję `CRC16()` można bez modyfikacji składni stosować w programach pisanych dla mikrokontrolera 8051. Program źródłowy zawierający obie funkcje przetłumaczono za pomocą kompilatora C-51 ver. 4.0 firmy Archimedes Software. Przy użyciu emulatora układowego zmierzono czasy wykonania ciała pętli `while(usDataLen--)` przy następujących warunkach:

- częstotliwość taktowania mikrokontrolera - 12 MHz,
- wszystkie zmienne były umieszczone w zewnętrznej pamięci danych,
- zabezpieczana wiadomość zawierała odpowiedź jednostki podrzędnej na polecenie odczytu zawartości grupy 6 rejestrów, przy czym w rejestrach były zapisane na przemian wartości `5555H` oraz `AAAAH`; rozmiar wiadomości wynosił 15 bajtów [1].

W wyniku pomiarów uzyskano m.in. średni czas wykonania jednego obiegu pętli. Wynosił on:

- dla funkcji `CRC16()` - 105 μs,
- dla funkcji `CRC16_p()` - 483 μs.

Widać zatem istotną przewagę algorytmu tablicowego.

4. Podsumowanie

Pewne rozwiązania techniczne, pomimo znacznego upływu czasu od chwili ich wprowadzenia, nadal są często wykorzystywane. Tak rzecz się ma z protokołem Modbus. Swą dużą popularność zawdzięcza on m.in. dostępności dokumentacji [1]. Dokumentacja ta stanowi podstawowe źródło wiadomości dla projektanta sprzętu i oprogramowania łącza komunikacyjnego wykorzystującego protokół Modbus. Oprócz opisu zadań protokołu, formatu jednostek informacyjnych i interfejsu z programami użytkowymi zawarto w niej także gotowe do użycia funkcje w języku C, służące do wyznaczania wartości sum kontrolnych. W trybie RTU protokołu Modbus stosowane jest zabezpieczenie przesyłanych wiadomości 16-bitową sumą kontrolną CRC, która w ramce jest zapisywana w postaci dwóch znaków. Ważna przy tym jest kolejność znaków: najpierw znak zawierający młodszy bajt sumy, a następnie starszy. Zamieszczona w dokumentacji funkcja obliczania CRC wyznacza jej wartość prawidłowo, ale we zwracanym przez funkcję wyniku zamieniona jest kolejność bajtów. Niewłaściwe umieszczenie bajtów sumy kontrolnej w ramce sprawi, że nawet po poprawnym przesłaniu wszystkich znaków ramka zostanie odrzucona przez odbiorców. Brak jednak w dokumentacji wzmianki o zamienionej kolejności bajtów w wyniku zwracanym przez funkcję obliczania CRC. Projektant, który implementuje oprogramowanie protokołu po raz pierwszy, po stwierdzeniu faktu odrzucania ramek wskutek niezgodności sum kontrolnych będzie szukał błędu raczej w napisanym przez siebie kodzie, a nie w zawartości dokumentacji. Spowoduje to stratę czasu pracy projektanta, podczas gdy uzyskanie właściwej kolejności bajtów w wyliczonym CRC wymaga niewielkiej poprawki w kodzie funkcji.

LITERATURA

1. AEG Schneider Automation: Modicon Modbus Protocol. Reference Guide. PI-MBUS-300 Rev. D.
2. Melore P.: Your personal PLC tutor. Dokument dostępny w sieci Internet na stronie

http://www.plcs.net/

3. Trybus L.: Poziom techniczny aparatowych regulatorów i sterowników końca lat 90. Pomiary, Automatyka, Kontrola, 3/1999.
4. Mielczarek W.: Szeregowe interfejsy cyfrowe. Helion, Gliwice 1993.
5. Mielczarek W.: Tłumienie zakłóceń i ochrona informacji w systemach pomiarowych. Skrypt Uczelniany Politechniki Śląskiej, nr 1921, Gliwice 1995.
6. Grzywak A. (red.): Laboratorium sieci komputerowych. Skrypt Uczelniany Politechniki Śląskiej, nr 1959, Gliwice 1995.
7. Pełka R.: Mikrokontrolery. Architektura, programowanie, zastosowania. WKŁ, Warszawa 1999.
8. Majewski J., Kardach K.: Mikrokontrolery jednoukładowe 8051. Programowanie w języku C w przykładach. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 1995.
9. Holland R.: Testowanie i diagnostyka systemów mikrokomputerowych. WNT, Warszawa 1993.

Recenzent: Dr inż. Wojciech Mielczarek

Wpłynęło do Redakcji 26 listopada 1999 r.

Abstract

Though Modbus protocol has been known since 1973, it's widely used so far in communication links among supervisory computer and controllers. Popularity of the protocol was achievied owing to simplicity of hardware and availability of protocol documentation.

The documentation entitled *Modicon Modbus Protocol. Reference Guide. PI-MBUS-300 Rev. D.* is a fundamental source of information for designers. However, this source contains some little contradictions and lacks, described in chapter 2 and 3, respectively. In this article two methods of CRC calculation were presented. Measured time efficiency of both methods was also given in chapter 3.