Sieć ZigBee w praktyce (3)

Przykładowe oprogramowanie modułów Telegesis ETRX357

W poprzednich artykułach zostały opisane rejestry konfiguracyjne modułu komunikacyjnego ETRX357 oraz znaczenie poszczególnych bitów konfigurujących. W tej części opisana zostanie prosta aplikacja przedstawiająca na ekranie komputera stan portów I/O modułu komunikacyjnego podłączonego do portu szeregowego.

Program napisano za pomocą kompilatora Borland C++ Builder. Wygląd ekranu głównego po uruchomieniu pokazano na **rysunku 1**. Pracę z programem zaczynamy od wpisania numeru portu szeregowego (COM), za pomocą którego aplikacja połączy się z modułem komunikacyjnym, a następnie naciskamy przycisk „Otwórz”. Jeśli otwarcie portu nie powiedzie się, to zobaczymy okienko jak na **rysunku 2**. Wówczas należy sprawdzić połączenia i to, czy port COM nie jest używany przez jakąś inną aplikację.

Poprawne otwarcie portu szeregowego nie jest sygnalizowane dodatkowymi komunikatami i można od razu przejść do sprawdzenia stanu portów I/O modułu ETRX357. W tym celu należy wcisnąć przycisk „Zapytaj”. Za sprawdzenie komunikacji z modułem ETRX jest odpowiedzialny fragment programu pokazany na **listingu 1**.

W następnym kroku należy wcisnąć przycisk „Odbierz”. Odsyłana odpowiedź przez moduł komunikacyjny ma format zamieszczony w **tabeli 1**.

Należy zaznaczyć, iż jest możliwe skonfigurowanie modułu za pomocą odpowiednich rejestrów w taki sposób, aby w odpowiedziach nie były umieszczane znaczniki początku i końca komunikatu (bit 7 rejestru S12) oraz potwierdzenie poprawnego wykonania komendy (bit 1 rejestru S0E). Biorąc pod uwagę budowę odpowiedzi (jak została przedstawiona w tab. 1), w celu wizualizacji stanu portów wejściowych należy odpowiednio zinterpretować bajty 4–11. Odpowiedzialny za to fragment programu zamieszczono na **listingu 2**.

W ten sposób realizowana jest wizualizacja stanu portów wejściowych na podstawie zawartości rejestru S1A. Aby zmienić stan portów, należy zmodyfikować zawartość rejestru S18. W tym celu należy zaznaczyć/odznaczyć kwadrat odpowiadający danej linii I/O modułu komunikacyjnego, a następnie wcisnąć przycisk „Ustaw”. Jest wówczas uruchamiany fragment programu pokazany na **listingu 3**.

Listing 1. Sprawdzenie stanu portów I/O modułu ETRX357

```
{strcpy(Buffer_O_COM, „atsla?\x0d”);  // skopiowanie pytania do bufora
wyjściowego
if (hCommDev != INVALID_HANDLE_VALUE) // sprawdzenie, czy port jest otwarty
  {
  for (i = 0; i <= cbInQueue - 1; i++)
    Buffer_I_COM[i] = NULL;  // wyczyszczenie bufora wejściowego
  do
    {
    FlushFileBuffers(hCommDev);  // wysłanie zawartości bufora wyjściowego
    }while (Write_Comm(hCommDev, Buffer_O_COM, strlen(Buffer_O_COM)) == 0);
  }
}
```

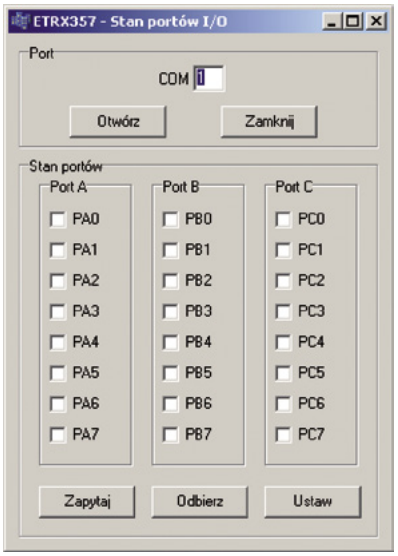
Tabela 1. Odpowiedź na zapytanie o stan portów wejściowych

Postać ASCII	Postać HEX	Opis
	02 0D 0A	Znacznik początku komunikatu i przejście do nowego wiersza
00FEBF73	30 30 46 45 42 46 37 33 0D 0A	Stan portów w postaci szesnastkowej
	03 02 0D 0A	Znacznik końca i początku nowego komunikatu oraz przejście do nowego wiersza
OK	4F 4B 0D 0A	Potwierdzenie poprawnego wykonania komendy i przejście do nowego wiersza
	03	Znacznik końca komunikatu

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 16732, pass: 630v2nfb
• poprzednie części kursu

Podsumowanie

Za pomocą tych trzech funkcji uzyskano możliwość odczytu i zmiany stanu portów I/O w module ETRX357. Na ich podstawie



Rysunek 1. Okno aplikacji

Rysunek 2. Komunikat o błędzie otwarcia portu szeregowego

Listing 2. Fragment programu odpowiedzialny za interpretację odebranych danych

```
  {
    Read_Comm(hCommDev, &Buffer_I_COM[0], &Number_Bytes_Read, sizeof(Buffer_I_
COM));
    if (Number_Bytes_Read > 0) {  // sprawdzenie, czy odebrano dane
      for (r=0; r < Number_Bytes_Read; r++) {
      Collected[r]=Buffer_I_COM[r]; // skopiowanie danych z bufora wejściowego
do zmiennej
    }
    Parse[0] = ,0';  // dopisanie na początku zmiennej Parse znaków „0x"
wymaganych przez funkcję TryStrToInt()
    Parse[1] = ,x';
    for (i=2; i<10; i++) {
      Parse[i+1]=NULL;  // dopisanie na końcu zmiennej Parse znaku kończącego
łańcuch znaków, wymagany przez funkcję TryStrToInt()
      Parse[i] = Collected[i+1];  // skopiowanie bajtów 4-11 do zmiennej Parse
    }
    TryStrToInt(Parse,ports);  // konwersja liczby szesnastkowej zapisanej
w kodzie ASCII (Parse) do zmiennej liczbowej typu całkowitego (ports)
    Form1->PA0->Checked = ports & 1;  // zaznaczenie odpowiedniego kwadratu
w obszarze „Stan portów" jako wynik iloczynu logicznego liczby „ports" i maski
bitowej podanej w postaci liczby dziesiętnej
    Form1->PA1->Checked = ports &       2;
    Form1->PA2->Checked = ports &       4;
    Form1->PA3->Checked = ports &       8;
    Form1->PA4->Checked = ports &      16;
    Form1->PA5->Checked = ports &      32;
    Form1->PA6->Checked = ports &      64;
    Form1->PA7->Checked = ports &     128;
    Form1->PB0->Checked = ports &     256;
    Form1->PB1->Checked = ports &     512;
    Form1->PB2->Checked = ports &    1024;
    Form1->PB3->Checked = ports &    2048;
    Form1->PB4->Checked = ports &    4096;
    Form1->PB5->Checked = ports &    8192;
    Form1->PB6->Checked = ports &   16384;
    Form1->PB7->Checked = ports &   32768;
    Form1->PC0->Checked = ports &   65536;
    Form1->PC1->Checked = ports &  131072;
    Form1->PC2->Checked = ports &  262144;
    Form1->PC3->Checked = ports &  524288;
    Form1->PC4->Checked = ports & 1048576;
    Form1->PC5->Checked = ports & 2097152;
    Form1->PC6->Checked = ports & 4194304;
    Form1->PC7->Checked = ports & 8388608;
    for (i = 0; i <= cbInQueue - 1; i++) {
      Buffer_I_COM[i] = NULL;  // wyczyszczenie bufora wejściowego
    }
  }
}
```

można zbudować własną aplikację wykorzystującą te moduły w konkretnym zastosowaniu. Oprogramowanie napisano w języku C i jest łatwo przenieść je na dowolny model mikrokontrolera, oczywiście modyfikując wymagane elementy składni. Program źródłowy dla kompilatora Borland Builder jest zamieszony na płycie CD dołączonej do miesięcznika i serwerze FTP.

Arkadiusz Hutnik
a.hutnik@op.pl

Listing 3. Modyfikacja zawartości rejestru S18

```
{
    ports =  // wpisanie do zmiennej liczbowej (ports) sumy iloczynów liczb
dziesiętnych i wyniku testów logicznych sprawdzenia, czy kwadrat danej linii
I/O jest
    (Form1->PA0->Checked & 1) *       1 +
    (Form1->PA1->Checked & 1) *       2 +
    (Form1->PA2->Checked & 1) *       4 +
    (Form1->PA3->Checked & 1) *       8 +
    (Form1->PA4->Checked & 1) *      16 +
    (Form1->PA5->Checked & 1) *      32 +
    (Form1->PA6->Checked & 1) *      64 +
    (Form1->PA7->Checked & 1) *     128 +
    (Form1->PB0->Checked & 1) *     256 +
    (Form1->PB1->Checked & 1) *     512 +
    (Form1->PB2->Checked & 1) *    1024 +
    (Form1->PB3->Checked & 1) *    2048 +
    (Form1->PB4->Checked & 1) *    4096 +
    (Form1->PB5->Checked & 1) *    8192 +
    (Form1->PB6->Checked & 1) *   16384 +
    (Form1->PB7->Checked & 1) *   32768 +
    (Form1->PC0->Checked & 1) *   65536 +
    (Form1->PC1->Checked & 1) *  131072 +
    (Form1->PC2->Checked & 1) *  262144 +
    (Form1->PC3->Checked & 1) *  524288 +
    (Form1->PC4->Checked & 1) * 1048576 +
    (Form1->PC5->Checked & 1) * 2097152 +
    (Form1->PC6->Checked & 1) * 4194304 +
    (Form1->PC7->Checked & 1) * 8388608 ;
    strcpy(Buffer_O_COM, "ats18="");  // skopiowanie do bufora wyjściowego
komendy modyfikującej rejestr S18
    strcat(Buffer_O_COM, IntToHex(ports,8).c_str());  // konwersja zmiennej
liczbowej typu całkowitego do postaci liczby szesnastkowej zapisanej w kodzie
ASCII, a następnie dopisanie tak uzyskanego łańcucha znaków na końcu bufora
wyjściowego
    strcat(Buffer_O_COM, „\x0d");  // dopisanie na końcu bufora wyjściowego
bajtu oznaczającego koniec komendy, wymaganego przez moduł ETRX357
    if (hCommDev != INVALID_HANDLE_VALUE) { // sprawdzenie, czy port jest
otwarty
    for (i = 0; i <= cbInQueue - 1; i++)
    Buffer_I_COM[i] = NULL;  // wyczyszczenie bufora wejściowego
    do {
      FlushFileBuffers(hCommDev);  // wysłanie zawartości bufora wyjściowego
    }while (Write_Comm(hCommDev, Buffer_O_COM, strlen(Buffer_O_COM)) == 0);
  }
}
```