# System nawigacji satelitarnej GPS, część 12

## Komunikacja z odbiornikiem GPS

*Jest to ostatnia część cyklu poświęcona opisom sposobów komunikacji z odbiornikami GPS. Skupiamy się w niej na pokazaniu sposobu weryfikacji identyfikatora wiadomości RMC oraz wydzieleniu z odebranych danych informacji nawigacyjnych.*


### Sprawdzanie identyfikatora

W funkcji *ReceiveGPSPacket()* przedstawionej na **list. 1** (EP12/2006) pominięto sprawdzenie, czy odbierana jest wiadomość RMC, co doprowadziłoby do błędów w pracy urządzenia, gdyby zastosowany odbiornik GPS wysyłał przez port szeregowy inne wiadomości NMEA. Może się to zdarzyć, jeśli opracowywane przez nas urządzenie będzie współpracowało z odbiornikami GPS, których nie możemy wstępnie skonfigurować tak, aby wysyłały wyłącznie wiadomości RMC. Aby uniknąć tego problemu, początek funkcji należy rozbudować o sprawdzenie identyfikatora odbieranej wiadomości. Na **list. 2** przedstawiono fragment funkcji *GetGPRMC(),* stanowiącej nieznacznie zmodyfikowaną część kodu źródłowego oprogramowania mikrokontrolera z rodziny 8051 (AT89S8252), zastosowanego w opisywanym już na łamach Elektroniki Praktycznej (EP4...5/2005) GPS–owym rejestratorze trasy. Dane odbierane z GPS są tu przechowywane w tablicy *RawGPS []*, o rozmiarze 63 bajtów. Jest ona o 7 bajtów mniejsza od tablicy *GPS. Packet[]* z list. 1, ponieważ zapisywane są w niej znaki bez znaku końca tekstu '\0', identyfikatora wiadomości GPRMC (5 znaków) i następującego po nim przecinka. Funkcja *GetGPRMC()* zawiera elementy podobne do przedstawionej na listingu list. 1 funkcji *ReceiveGPSPacket()*, jednak dzięki sprawdzaniu identyfikatora wiadomości, do rejestratora mogą docierać dowolne wiadomości NMEA z odbiornika GPS, a mimo to będzie on pracował poprawnie i odbierał wyłącznie wiadomości RMC. Zwiększa to uniwersalność urządzenia i umożliwia jego współpracę np. z odbiornikami wysyłającymi stałą, niepoddającą się modyfikacji, listę wiadomości NMEA. W przedstawionym na list. 2 fragmencie programu, do odbioru znaków z portu szeregowego wykorzystano funkcję *_getkey()* z biblioteki standardowego wejścia/wyjścia języka C *stdio.h*.

List. 2. Funkcja realizująca odbiór wiadomości RMC z odbiornika GPS

```
unsigned char RawGPS [63];              // tablica na wiadomości $GPRMC z GPS

void GetGPRMC( void )
  {
  bit  MessageReceived = 0;
  unsigned char temp;
  unsigned char i = 0;
  unsigned char ChSum, ChSumRcv;        // suma kontrolna obliczona i odebrana z GPS
  while (!MessageReceived)
     {
     ChSum = 0;
     while ( (temp=_getkey()) != '$');  // oczekiwanie na początek wiadomości $GPRMC
     temp=_getkey();
     if ( temp != 'G') continue;        // sprawdzenie czy kolejny znak to 'G'
     ChSum = ChSum ^ temp;
     temp=_getkey();
     if ( temp != 'P') continue;        // sprawdzenie czy kolejny znak to 'P'
     ChSum = ChSum ^ temp;
     temp=_getkey();
     if ( temp != 'R') continue;        // sprawdzenie czy kolejny znak to 'R'
     ChSum = ChSum ^ temp;
     temp=_getkey();
     if ( temp != 'M') continue;        // sprawdzenie czy kolejny znak to 'M'
     ChSum = ChSum ^ temp;
     temp=_getkey();
     if ( temp != 'C') continue;        // sprawdzenie czy kolejny znak to 'C'
     ChSum = ChSum ^ temp;
     temp=_getkey();
     ChSum = ChSum ^ temp;
     while ( (temp=_getkey()) != '*')   // zapis wiadomości $GPRMC
        {                               // w tablicy RawGPS[]
        RawGPS[i++] = temp;
        ChSum = ChSum ^ temp;
        }
     temp = _getkey();                  // odbiór 2 bajtów sumy kontrolnej
     if ( temp > '9' )  temp - = 55;    // kończącej wiadomość $GPRMC
     else          temp - = 48;
     ChSumRcv = 16 * temp;
     temp = _getkey();
     if ( temp > '9' )  temp - = 55;
     else          temp - = 48;
     ChSumRcv += temp;
     if (ChSum!=ChSumRcv) continue;
     MessageReceived = 1;               // ustawienie flagi kończącej odbiór wiadomo-
ści
     }
  }
```

List. 3. Program do odbioru wiadomości RMC i zobrazowania danych na wyświetlaczu LCD

```
#include <AT89x051.H>

unsigned char RawGPS [63];          // tablica na wiadomości $GPRMC z GPS

extern void Init (void);
extern void GetGPRMC(void);
extern void InitLCD (void);
extern void ClearLCD (void);
extern void WriteLCD (unsigned char);
extern void SetCursor (unsigned char, unsigned char);
extern void Line2LCD (unsigned char *);

void main (void)
  {
  bit Valid;
  unsigned char i;

  Init();                           // inicjalizacja urządzeń peryferyjnych mikrokontrolera
  InitLCD();                        // inicjalizacja wyświetlacza LCD

  while(1)
     {
     ClearLCD();                    // czyszczenie zawartości wyświetlacza
     GetGPRMC();                    // odbiór wiadomości RMC
     i=0;
     while ( RawGPS[i++] != ',' );  // oczekiwanie na przecinek przed polem statusu
     Valid = (RawGPS[i++]=='A');    // sprawdzenie statusu danych: 'A' - dane poprawne
                                    // (Valid=1), 'V' - dane niepoprawne (Valid=0)
     while ( RawGPS[i++] != ',' );  // oczekiwanie na przecinek przed polem szerokości geogr.
     SetCursor(1,1);
     Line2LCD("Sz:");
     SetCursor(1,5);
     while ( RawGPS[i] != ',' )     // wyświetlanie na LCD kolejnych znaków szerokości
        WriteLCD( RawGPS[i++] );    // geogr., aż do napotkania przecinka kończącego to pole
     WriteLCD( RawGPS[++i] );       // wyświetlanie wskaźnika półkuli N/S
     WriteLCD( Valid? ' ' : '*' );  // wyświetlenie '*' na końcu, jeśli dane są niepoprawne

     while ( RawGPS[i++] != ',' );  // oczekiwanie na przecinek przed polem długości geogr.
     SetCursor(2,1);
     Line2LCD("Dl:");
     SetCursor(2,4);
     while ( RawGPS[i] != ',' )     // wyświetlanie na LCD kolejnych znaków długości
        WriteLCD( RawGPS[i++] );    // geogr., aż do napotkania przecinka kończącego to pole
     WriteLCD( RawGPS[++i] );       // wyświetlanie wskaźnika półkuli E/W
     WriteLCD( Valid? ' ' : '*' );  // wyświetlenie '*' na końcu, jeśli dane są niepoprawne
     }
  }
```

Przykłady z list. 1 i list. 2 przedstawiają sposób odbioru wiadomości RMC, która jest jedną z najbardziej przydatnych w praktyce. Zasada odbioru innych wiadomości NMEA jest jednak analogiczna do przedstawionej i wymaga tylko kosmetycznych zmian kodu. Zmiany te obejmują dobranie rozmiaru tablicy do liczby przechowywanych w niej znaków, ponieważ poszczególne wiadomości NMEA mają różne długości oraz zmianę identyfikatora wiadomości (list. 2), który będzie poszukiwany w danych przychodzących z odbiornika GPS.

## Wydzielanie danych nawigacyjnych

Przedstawione dotychczas fragmenty programów wyjaśniały sposób odbierania wiadomości i zapisywania jej w tablicy. Obecnie zajmiemy się zasadą wydzielania i formatowania poszczególnych pól odebranej wiadomości. Sposób postępowania z zawartością wiadomości zależy głównie od przeznaczenia odbieranych danych GPS. Inaczej będzie się odbywało formatowanie danych w urządzeniach zobrazowania informacji nawigacyjnej, w których dane z odbiornika służą wyłącznie do podania użytkownikowi jego położenia i parametrów ruchu, a inaczej w rejestratorach trasy, urządzeniach śledzenia pojazdów, czy też w zintegrowanych systemach nawigacyjnych, gdzie zachodzi konieczność przechowywania, przesyłania lub przetwarzania dużej ilości informacji.

W zależności od potrzeb, z wiadomości NMEA mogą być wydzielane wszystkie lub tylko wybrane pola, istotne z punktu widzenia aplikacji użytkownika. Dane nawigacyjne w wiadomościach NMEA mają format tekstowy, w którym każda cyfra jest reprezentowana przez jeden znak ASCII, a tym samym zajmuje 1 bajt przesyłanej wiadomości. Zaletą tego formatu jest jego czytelność. Obserwując przychodzące wiadomości NMEA na komputerze PC za pomocą programu komunikacyjnego takiego jak *Hyperterminal* bez trudu odnajdziemy w nich interesujące nas informacje. Format tekstowy charakteryzuje się jednak słabym „upakowaniem" danych. Wydzielając poszczególne pola wiadomości należy zdecydować czy mają one pozostać w formacie tekstowym, czy należy je przekształcić do postaci bardziej skompresowanej, tzn. do formatu binarnego. Pozostawienie danych w postaci tekstowej bardzo upraszcza program mikrokontrolera. Z drugiej strony, zapisywanie danych w postaci tekstowej w rejestratorach GPS spowodowałoby gorsze wykorzystanie dostępnej pamięci, zaś w urządzeniach śledzenia pojazdów spowodowałoby konieczność przesyłania większych ilości danych i mogłoby wpływać na zwiększone koszty eksploatacji systemu. Ponadto, w wielu aplikacjach jest niezbędne nie tylko wydzielenie danych nawigacyjnych z wiadomości, ale również ich bieżące przetwarzanie. W takich przypadkach, konieczne staje się przekształcenie odebranych danych z formatu tekstowego do postaci liczbowej.

Na początek zajmiemy się prostym przykładem urządzenia, w którym zmiana formatu tekstowego danych nie jest konieczna. Załóżmy, że konstruowane przez nas urządzenie z mikrokontrolerem z rodziny 8051 będzie służyło do wyświetlania informacji o położeniu z odbiornika GPS na wyświetlaczu alfanumerycznym LCD, np. 2x16.

Na listingu **list. 3** przedstawiono fragment prostego programu napisanego dla mikrokontrolerów z rodziny 8051 (w tym przypadku AT89C4051 firmy Atmel), przeznaczonego do odbierania wiadomości RMC z odbiornika GPS, wydzielania z niej pól zawierających szerokość i długość geograficzną położenia użytkownika i wyświetlania tych danych na wyświetlaczu LCD.

W przedstawionym fragmencie kodu wykorzystano opisaną wcześniej i zamieszczoną na list. 2 funkcję *GetGPRMC()*, funkcję inicjalizującą procesor *Init()* oraz funkcje do obsługi wyświetlacza LCD, których przykłady można znaleźć w licznych źródłach, m.in. w Internecie. Założono, że wymienione funkcje znajdują się w osobnych plikach źródłowych, stąd słowo kluczowe *extern* w ich deklaracjach.

W programie przedstawionym na list. 3, po inicjalizacji mikrokontrolera i wyświetlacza LCD, w pętli nieskończonej *while(1)* jest wykonywane odbieranie wiadomości RMC, wydzielanie z niej informacji o położeniu użytkownika i wyświetlanie położenia na wyświetlaczu alfanumerycznym LCD. Po odebraniu wiadomości za pomocą funkcji *GetGPRMC()*, interesujące nas dane nawigacyjne są dostępne do dalszego wykorzystania w tablicy *RawGPS[]*. Przykładowo, jeśli z odbiornika GPS otrzymamy wiadomość RMC o treści:

```
$GPRMC,092842.094,A,5215.207
8,N,02054.3681,E,0.13,1.29,1
80706,,*0A
```

w tablicy *RawGPS[]* znajdą się liczby stanowiące kody ASCII następujących znaków:

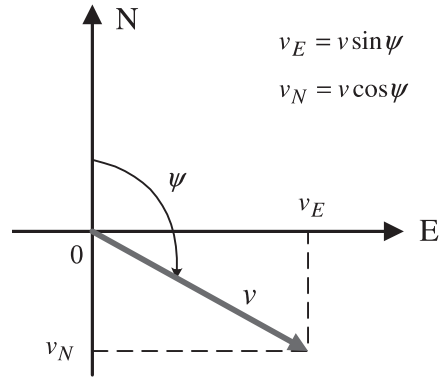
```
092842.094,A,5215.2078,N,020
54.3681,E,0.13,1.29,180706,,
```

W dalszej części programu, z powyższego ciągu znaków jest wydzielane pole statusu, w celu sprawdzenia czy otrzymana wiadomość zawiera poprawne dane. Wówczas w polu statusu znajduje się znak 'A'. Jeśli dane nie są poprawne, wystąpi w tym miejscu znak 'V'. Poszukiwanie pola statusu sprowadza się do odnalezienia pierwszego przecinka w tablicy *RawGPS[]*. Informację o bieżącym statusie odebranych danych przechowuje zmienna bitowa *Valid*. Następnie w programie jest odnajdywany kolejny przecinek, który poprzedza pole szerokości geograficznej. Szerokość geograficzna jest znak po znaku odczytywana i wyświetlana na wyświetlaczu LCD. Przepisywanie znaków trwa do momentu odnalezienia przecinka kończącego pole szerokości geograficznej. Alternatywnym rozwiązaniem, równie skutecznym w przypadku wielu odbiorników GPS, byłoby wydzielanie i wyświetlanie stałej liczby znaków z pola szerokości geograficznej. Sposób zaproponowany na list. 3 jest jednak bardziej uniwersalny, ponieważ sprawdza się w przypadku dowolnej, spotykanej w praktyce, rozdzielczości położenia (liczby miejsc po kropce dziesiętnej). Po wyświetleniu szerokości geograficznej, jest odczytywany i wyświetlany kolejny znak po przecinku, który wskazuje czy ustalone położenie jest na półkuli północnej (N) czy południowej (S). Zasada wydzielania z wiadomości RMC i wyświetlania pola długości geograficznej wraz ze wskaźnikiem półkuli wschodniej (E) lub zachodniej (W) jest analogiczna. Dodatkowo w programie wykorzystano zawartą w wiadomości RMC informację o statusie danych nawigacyjnych. Jeśli dane są niepoprawne, to są one mimo wszystko wyświetlane, ale oznaczane na zakończenie gwiazdką, co daje użytkownikowi informację, że odbiornik nie może obecnie ustalić położenia. Wyświetlanie tych danych może być jednak przydatne, ponieważ do czasu ustalenia nowego położenia odbiorniki GPS zwykle wysyłają ostatnią znaną pozycję.

## Przetwarzanie danych z odbiornika GPS

Na zakończenie części poświęconej wykorzystaniu wiadomości NMEA zajmiemy się przypadkiem, kiedy dane pochodzące z odbiornika GPS muszą być przetwarzane w naszej aplikacji. Potrzeba wykonania obliczeń z wykorzystaniem wydzielonych danych nawigacyjnych wymusza konieczność ich przekształcenia z postaci tekstowej do postaci liczbowej. Jako przykład rozważymy działanie urządzenia, którego zadaniem jest wyznaczanie składowych prędkości ruchu pojazdu w kierunku wschodnim $v_E$ i północnym $v_N$. Dane te nie są bezpośrednio dostępne w żadnej standardowej wiadomości NMEA, a więc oprogramowanie mikrokontrolera będzie musiało poradzić sobie z ich obliczeniem. Zależność wielkości przesyłanych w wiadomości RMC, tj. prędkości $v$ i kursu $\psi$ oraz wielkości, które mają zostać obliczone, tj. składowej wschodniej prędkości $v_E$ i składowej północnej prędkości $v_N$ wyjaśniono na **rys. 36**.

Przykładową funkcję realizującą obliczanie składowych prędkości




**Rys. 36. Relacje geometryczne kursu, prędkości i jej składowych**

na podstawie danych otrzymanych z odbiornika GPS przedstawiono na **list. 4**.

Przedstawiony fragment kodu źródłowego został napisany dla mikrokontrolera AVR ATMega-128 i pochodzi ze wspomnianego wcześniej programu służącego do wspólnego przetwarzania danych z odbiornika GPS i systemu nawigacji inercjalnej. W programie tym są naprzemiennie wywoływane 2 funkcje, tj. pokazana na list. 1 funkcja *ReceiveGPSPacket()* i funkcja *ProcessGPSPacket()* z list. 4. Obie funkcje operują na strukturze danych o nazwie GPS, służącej do przechowywania ciągu znaków z odebranej wiadomości RMC i wydzielonych z niej danych nawigacyjnych. Struktura przedstawiona na List. 4 stanowi rozszerzoną wersją struktury z list. 1. Dodane do niej pola *Vel*, *VelN*, *VelE* i *Head* służą do przechowywania obliczonych prędkości i kursu. Rola funkcji *ReceiveGPSPacket()* sprowadza się do przepisania fragmentu odebranej wiadomości RMC do tablicy *GPS.Packet[]*. Z przykładowej wiadomości:

```
$GPRMC,092842.094,A,5215.207
8,N,02054.3681,E,0.13,1.29,1
80706,,*0A
```

w tablicy *GPS.Packet[]* znajdą się liczby stanowiące kody ASCII następujących znaków:

```
GPRMC,092842.094,A,5215.207
8,N,02054.3681,E,0.13,1.29,
180706,,
```

i dodatkowo znak końca tekstu '\0'.

Wywoływana następnie funkcja *ProcessGPSPacket()* poszukuje w zapisanym tekście kolejnych przecinków, aż do znalezienia przecinka poprzedzającego pole prędkości. Następnie do momentu napotka-

List. 4 Funkcja obliczająca składowe prędkości podróżnej

```
#define MAX_RMC_SIZE       69
#define MPH_2_METERSPERSEC    0.51444444
#define RADIANS_PER_DEGREE    1.74532952e-2

struct GPS_TYPE
  {
  unsigned char Packet[MAX_RMC_SIZE+1];
  unsigned char ChSumCorrect;
  float Vel;
  float VelN;
  float VelE;
  float Head;
  };

void ProcessGPSPacket( void )
  {
  unsigned char Vel[6], Head[7];     // tymczasowe tablice na zawartość pól prędkości i kursu
  unsigned char Count1 = 0;
  unsigned char Count2;
  unsigned char Temp;

  while ( GPS.Packet[Count1++] != ',' );          // poszukiwanie przecinka przed polem czasu
  while ( GPS.Packet[Count1++] != ',' );          // poszukiwanie przecinka przed polem statusu danych
  while ( GPS.Packet[Count1++] != ',' );          // poszukiwanie przecinka przed polem szerokości geogr.
  while ( GPS.Packet[Count1++] != ',' );          // poszukiwanie przecinka przed polem wskaźnika N/S
  while ( GPS.Packet[Count1++] != ',' );          // poszukiwanie przecinka przed polem długości geogr.
  while ( GPS.Packet[Count1++] != ',' );          // poszukiwanie przecinka przed polem wskaźnika E/W

  while ( GPS.Packet[Count1++] != ',' );          // poszukiwanie przecinka przed polem prędkości
  Count2=0;
  while ( (Temp=GPS.Packet[Count1++]) != ',' )    // wydzielenie zawartości pola prędkości i zapisanie
     Vel[Count2++] = Temp;         // w postaci liczby typu float
  if (Count2<5) Vel[Count2] = '\0';
  GPS.Vel = MPH_2_METERSPERSEC * atof(Vel);

  Count2=0;
  while ( (Temp=GPS.Packet[Count1++]) != ',' )    // wydzielenie zawartości pola kursu i zapisanie
     Head[Count2++] = Temp;        // w postaci liczby typu float
  if (Count2<6) Head[Count2] = '\0';
  GPS.Head = atof(Head);

  GPS.VelN = GPS.Vel*cos(RADIANS_PER_DEGREE*GPS.Head);     // składowa północna prędkości
  GPS.VelE = GPS.Vel*sin(RADIANS_PER_DEGREE*GPS.Head);     // składowa wschodnia prędkości
  }
```

nia przecinka kończącego to pole, wszystkie jego znaki są przepisywane do tablicy *Vel[]*. W typowych odbiornikach GPS pole prędkości liczy nie więcej niż 6 znaków. Jego długość może być przy tym zmienna. Jeśli w implementacji protokołu NMEA producent odbiornika GPS nie zastosował zer prowadzących, to przy niewielkich prędkościach ilość znaków w polu prędkości może być mniejsza niż 6 i nie wszystkie elementy tablicy *Vel[]* zostają wypełnione. W takiej sytuacji, w celu oznaczenia końca przepisanego tekstu, w tablicy *Vel[]* jest po nim dopisywany znak '\0'. Po wydzieleniu ciągu znaków ASCII reprezentujących prędkość jest on przekształcany w liczbę typu *float* za pomocą funkcji *atof()* należącej do biblioteki standardowej języka C *stdlib.h*. Obliczona prędkość jest następnie mnożona przez stałą *MPH_2_METERSPERSEC* w celu zamiany jednostek z węzłów na m/s. W dalszej części funkcji *ProcessGPSPacket()*, na podobnej zasadzie jak w przypadku prędkości jest wydzielany i przekształcany w liczbę kąt kursu. Kiedy obie te wielkości są już znane, można obliczyć interesujące nas składowe prędkości ruchu w kierunku wschodnim i północnym. Są one obliczane zgodnie z zależnościami z rys. 36, przy wykorzystaniu funkcji trygonometrycznych *cos()* i *sin()* wchodzących w skład biblioteki matematycznej *math.h*. Funkcje trygonometryczne wymagają argumentu wyrażonego w radianach i z tego względu wyrażony w stopniach kurs *GPS.Head* jest przed wywołaniem tych funkcji mnożony przez stałą *RADIANS_PER_DEGREE*.

Przedstawione na listingach przykładowe fragmenty programów i funkcji zostały nieco okrojone w stosunku do wersji oryginalnych i wybrane w taki sposób, aby pokazać ogólne zasady realizacji typowych zadań programistycznych związanych z wykorzystaniem danych GPS w formacie NMEA. Nie uwzględniają one wszystkich możliwych przypadków szczególnych i nie są zoptymalizowane ani pod kątem szybkości działania, ani wymaganej pamięci. Z tego względu powinny być traktowane raczej jako wskazówka przy tworzeniu własnego oprogramowania niż jako gotowe rozwiązania.

**Piotr Kaniewski**
**pkaniewski@wat.edu.pl**