Übung UART: GPS - Empfang und Dekodierung von NMEA-Daten

Obwohl die serielle Schnittstelle um 1960 entwickelt und spezifiziert wurde, ist sie nach wie vor - d.h. heutzutage- bei vielen Modulen anzutreffen. Die meisten GPS-Empfänger stellen ihre Daten per UART zur Verfügung; so auch die hier verwendete GPS-Maus GM-210 von Holux:

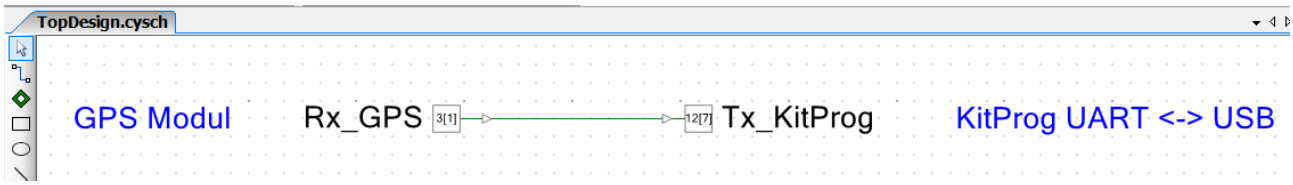
Sobald dieses GPS-Modul mit +5V versorgt wird, beginnt es GPS Signale zu empfangen, zu dekodieren und die Informationen in sog. NMEA-Datensätzen (engl. NMEA-Sentences) per UART auszugeben. Empfängt das Modul keine (oder nur sehr schwache) Signale, dann werden Leerdaten generiert – die jedoch NMEA-konform sind. D.h. man kann an seinem Arbeitsplatz einen NMEA-Dekoder programmieren, der dann z.B. für Longitude und Latitude (0)0000.0000 ausgibt. Aus bereits sehr schwachen Empfangssignalen werden nach wenigen Minuten bereits Uhrzeitinformationen gewonnen; und wenn mehr als 2 Satelliten „sichtbar“ sind, wird die FIX-Kennung gesetzt und Koordinaten-Informationen (in der verfügbaren Genauigkeit) ausgegeben.

Ein solches GPS-Modul eignet sich daher gut, um mit der seriellen Schnittstelle zu arbeiten… und nebenbei das Wissen über Pointer, Arrays & Zeichenketten aufzufrischen ;-)

1. Übung: Kennenlernen NMEA-Datensätze

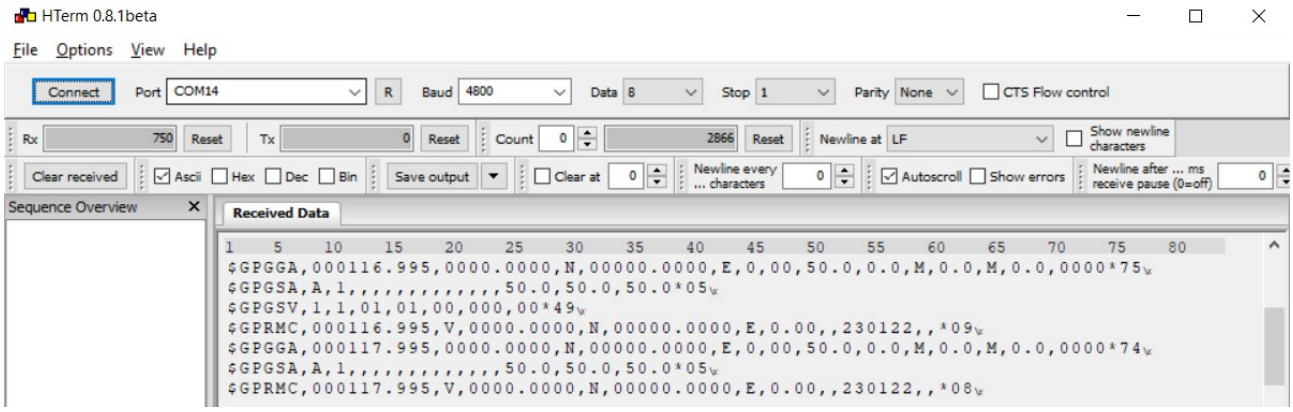
Verbinde die GPS-Maus mit dem PSoC5LP Edu-Board. Dafür eignet sich die „untere“ Arduino-Buchsenreihe sehr gut – hier liegt +5V und GND nebeneinander… als Dateneingang kann z.B. P3.1 (A0) verwendet werden. Bei allen Modulen gilt: rot oder orange = +5V, schwarz oder grau = GND, andere Farbe = TX-Daten (d.h. Daten vom GPS-Modul; TTL). Leuchtet die grüne LED (auf der Modulvorderseite), dann wird das Modul korrekt mit Spannung versorgt und ist empfangsbereit. Blinkt die grüne LED, dann werden ausreichend viele GPS-Satellitensignale empfangen um Positionsdaten auszugeben (GPS-Fix). Das GPS-Modul sendet die Daten mit 4800 Baud, 8 Datenbits, keiner Parität und 1 Stopp-Bit.

Bei der ersten Übung wird dieses empfangene UART-Signal unverändert auf den TX-Eingang des KitProg weitergeleitet; die KitProg-Einheit nimmt dann die UART Daten entgegen und sende diese per USB an den PC/Laptop:



Pin-Mapping: *Rx_GPS: P3[1]* *Tx_KitProg: P12[7]*

Auf dem Computer/Laptop wir dann mit einem Terminalprogramm der COM-Port (mit den oben erwähnten Einstellungen) geöffnet und man sieht sofort NMEA-Rohdaten (ohne Fix mit Leermeldungen):



Auszug https://www.rfwireless-world.com/Terminology/GPS-sentences-or-NMEA-sentences.html :

GPS Sentences | NMEA Sentences | GPGGA GPGLL GPVTG GPRMC

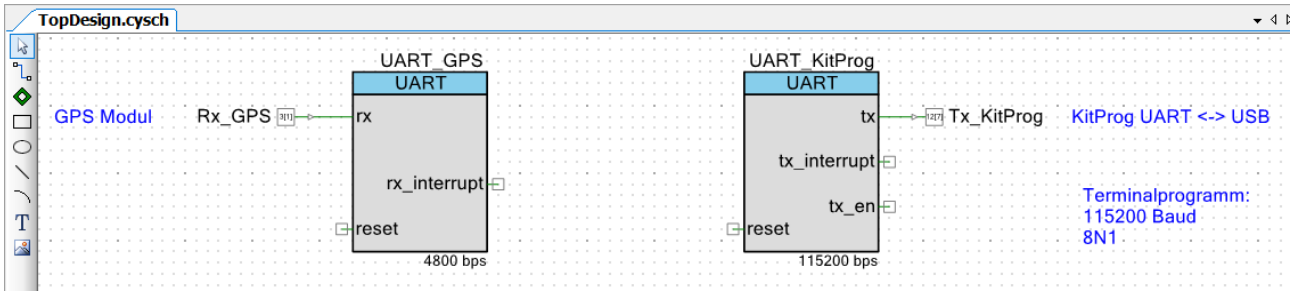
This page describes GPS Sentences or NMEA Sentences with example patterns. These GPS Sentences (i.e. NMEA Sentences) covers GPGGA, GPGLL, GPVTG, GPRMC etc.

***Introduction**:*
• A GPS receiver module requires only DC power supply for its operation. It will start outputting data as soon as it has identified GPS satellites within its range.
• GPS module uses plain ASCII protocol known as NMEA developed by National Marine Electronics Association. Hence they are also known as NMEA sentences.
• Each block of data is referred as "sentence". Each of these sentences are parsed independently.
*• The default transmission rate of these **gps sentences** is 4800 bps. Certain GPS modules use serial rate of 9600 bps also. It uses 8 bits for ASCII character, no parity and 1 stop bit.*
• Sentence begins with two letters to represent GPS device. For example, "GP" represent GPS device and so on.
• Remainder of sentence consists of letters/numerals in plain ASCII. A sentence can not have more than 80 characters.
• …

Recherchiere im Internet um einen Überblick über die NMEA-Datensätze und die darin enthaltenen Daten zu erhalten. Eine kurze händische Dekodierung der empfangenen (und per Terminal ausgegebenen) NMEA-Daten dürfte damit keine Schwierigkeit darstellen…

2. Übung: UART @ PSoC

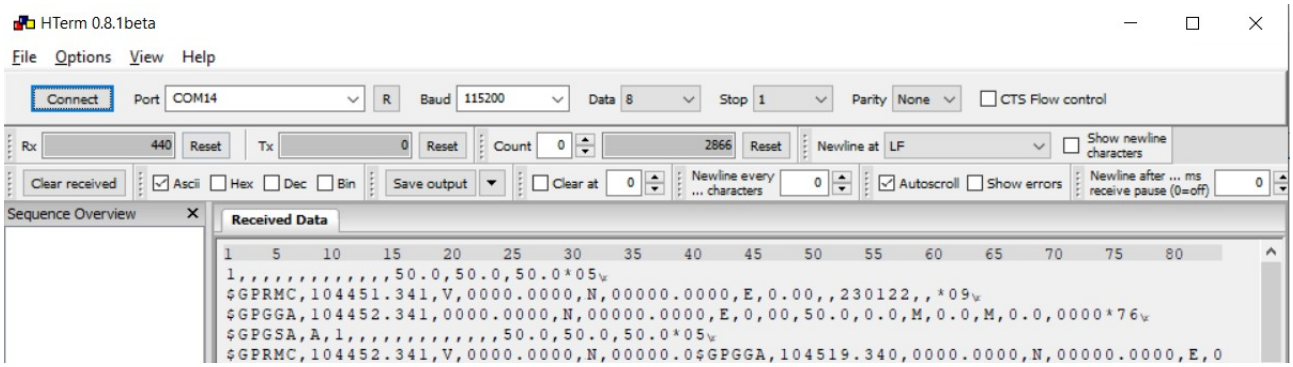
Die GPS-Maus sendet die Daten mit 4800 Baud (= default transmission rate; siehe Auszug oben). Die Kommunikation zwischen KitProg und Computer sollte jedoch mit 115.200 Baud erfolgen. Für diese Aufgabe werden daher zwei UART-Komponenten innerhalb des PSoC5LP benötigt. Eine UART-Komponente empfängt die Daten mit 4800 Baud, die zweite UART-Komponente sendet die Daten dann mit 115.200 Baud:



Vorschlag main.c ...

```c
int main(void)
{

    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g. MyInst_Start()) */
    UART_KitProg_Start();
    UART_GPS_Start();


    for(;;)
    {
        /* get GPS data if received, otherwhise 0 is returned */
        uint8 gps_data = UART_GPS_GetChar();

        if(gps_data != 0u)
        {
            /* send GPS data to KitProg / PC */
            UART_KitProg_PutChar(gps_data);
        }
    }
}
```

… ist sehr „einfach“ gehalten; wir hätten hier effizienter & eleganter mit Interrupts arbeiten können (RX-Interrupt UART_GPS). Am Terminalprogramm ist nun die Baud-Rateneinstellung zu ändern; ansonsten sieht die Ausgabe -im Prinzip- unverändert aus:



→ Überlege, warum zu erst die KitProg-UART gestartet wurde (und erst dann die GPS-UART).

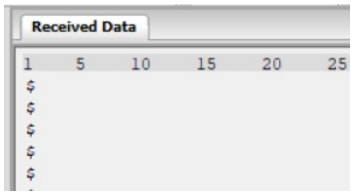
3. Übung: rxBuf und Pointer, Arrays und Zeichenketten...

Sobald das GPS-Modul mit Spannung versorgt wird, sendet es Daten. Unabhängig davon, ob der Empfänger bereit ist, die Daten zu empfangen. Dies sieht man schön beim oben abgebildeten Screenshot – mit der Verbindung zur seriellen Schnittstelle sieht man noch „Reste“ des vorherigen NMEA-Datensatzes. Eine Dekodierung kann erst mit der Beginn der Startkennung (= $-Zeichen) erfolgen.

In einem ersten Schritt wird im empfangenden NMEA-Datenstrom auf das $-Zeichen sowie das Zeilenende-Zeichen (LineFeed) „getriggert“ und nur diese zwei Zeichen per UART ausgegeben. Sehr elegant lässt sich dies mit einer switch-case-Struktur realisieren; Vorschlag:

```
25      for(;;)
26      {
27          /* get GPS data if received, otherwhise 0 is returned */
28          uint8 gps_data = UART_GPS_GetChar();
29
30          switch (gps_data)
31          {
32              case 0x00:
33                  break;
34
35              case '$':  //Begin with '$'
36              {
37                  UART_KitProg_PutChar(gps_data);
38                  break;
39              }
40
41              case 0x0A: //End with '<LF>'
42              {
43                  UART_KitProg_PutChar(gps_data);
44                  break;
45              }
46
47
48              default:
49                  // do nothing...
50                  break;
51
52          }
53      }
```

Terminal-Ausgabe:



Im nächsten Schritt sollen nun die vollständigen NMEA-Datensätze in einen Empfangsbuffer geschrieben werden. Die NMEA-Daten liegen im ASCII Code vor und der Buffer sollte als Zeichenkette (String) interpretierbar sein.

```
#include "project.h"

char rxBuf[81];
char *rxBufPtr = rxBuf;

int main(void)
{
```

→ Überlege, was es mit den beiden *char* … Codezeilen auf sich hat!

Mit Hilfe eines Flags wird festgestellt, ob ein kompletter NMEA-Datensatz im Empfangspuffer abgelegt wurde.

```
17  int main(void)
18  {
19      /* if set, we have a complete NMEA sentence in our receive buffer */
20      uint8 end_flag = 0;
21
22      CyGlobalIntEnable; /* Enable global interrupts. */
```

Innerhalb der Switch-Case-Struktur wird nun der Puffer gefüllt…

```
33          switch (gps_data)
34          {
35              case 0x00:
36                  break;
37
38              case '$':  //Begin with '$'
39              {
40                  rxBufPtr = rxBuf;
41                  *rxBufPtr = gps_data;
42                  rxBufPtr++;
43                  break;
44              }
45
46              case 0x0A: //End with '<LF>'
47              {
48                  *rxBufPtr = gps_data;
49                  *(rxBufPtr + 1) = '\0';
50                  end_flag = 1;
51                  break;
52              }
53
54
55              default:
56                  *rxBufPtr = gps_data;
57                  rxBufPtr++;
58          }
```

→ Vollziehe den Codevorschlag nach. Warum wird ein \0 in dem Empfangspuffer geschrieben, bevor das end_flag gesetzt wird?

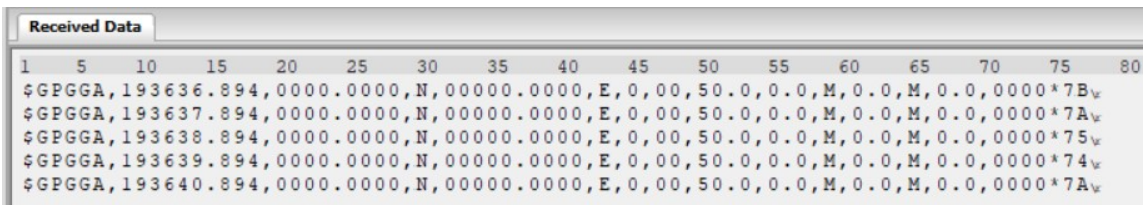
Erst wenn das Flag gesetzt wird, wird der Empfangspuffer über die UART ausgegeben:

```
63          if (end_flag == 1)
64          {
65              end_flag = 0;
66              rxBufPtr = rxBuf;
67
68              UART_KitProg_PutString(rxBuf);
69
70          }
```

4. Übung: Einfacher Dekoder… YOUR task!

Ein NMEA-Dekoder würde alle Datensätze auswerten um alle Informationen zur Verfügung zu stellen. Wir wollen nur Teile des GPGGA – Datensatz dekodieren, da dieser (unter anderem) die Uhrzeit, Positionsdaten sowie den Position-Fix-Indikator beinhaltet.

1. Aufgabe: Erstelle eine Funktion `SimpleDecode()`. Diese Funktion soll anstelle der rxBuf-UART-Ausgabe aufgerufen werden. In dieser Funktion soll überprüft werden, ob sich ein GPGGA – Datensatz im Empfangspuffer befindet – und wenn dies der Fall ist, so sollte dieser Datensatz ausgegeben werden (Anmerkung: der Einfachheit halber soll KEINE Checksum-Verifikation erfolgen).



2. Aufgabe: Die `SimpleDecode()` Funktion soll nun die (im UTC-Format übertragene) Uhrzeit im Format hhmmss extrahieren und in einem `time`-Array ablegen. Ausgabe der Uhrzeit-Information per UART z.B. mit

```
99          UART_KitProg_PutString("\t\t GPS Time: ");
100         UART_KitProg_PutString(time);
101         UART_KitProg_PutString("\n");
```

```
Received Data
1    5    10    15    20    25    30    35    40    45    50    55    60    65    70    75    80
$GPGGA,194347.868,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7C
    GPS Time: 19:43:47
$GPGGA,194348.868,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*73
    GPS Time: 19:43:48
$GPGGA,194349.868,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*72
    GPS Time: 19:43:49
$GPGGA,194350.868,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7A
    GPS Time: 19:43:50
```

3. Aufgabe: Dekodiere nun die Latitude- und Longitude-Information und gib diese ebenfalls per UART aus:

```
Received Data
1    5    10    15    20    25    30    35    40    45    50    55    60    65    70    75    80
$GPGGA,194947.846,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7A
    GPS Time: 19:49:47
    Latitude: 0000.0000
    Longitude: 00000.0000

$GPGGA,194948.846,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*75
    GPS Time: 19:49:48
    Latitude: 0000.0000
    Longitude: 00000.0000
```

4. Aufgabe: Nicht jeder GPS-Empfänger füllt fehlende Daten mit Nullinformationen auf. Das Trennzeichen ist das Komma – d.h. fehlende Daten könnten auch in der Form „,,, übermittelt werden (siehe z.B. GPGSA-Datensatz). Schreibe Deinen Dekoder nun dahingehend um, dass die Positionsdaten anhand der Kommapositionen gesucht, extrahiert und ausgegeben werden.

5. Aufgabe: Relevant ist noch die Information, ob ein GPS-Fix vorliegt oder nicht. Extrahiere diese Information anhand der Kommaposition. Ist der Fix-Indikator gleich 1 oder 2, so ist „GPS Fix!“ auszugeben.

```
Received Data
1    5    10    15    20    25    30    35    40    45    50    55    60    65    70    75    80    85
$GPGGA,195600.829,4732.2246,N,01205.1809,E,1,03,10.8,541.1,M,47.2,M,0.0,0000*46
    GPS Time: 19:56:00
    GPS Fix!
    Latitude: 4732.2246
    Longitude: 01205.1809

$GPGGA,195601.829,4732.2246,N,01205.1810,E,1,03,10.8,541.2,M,47.2,M,0.0,0000*4C
    GPS Time: 19:56:01
    GPS Fix!
    Latitude: 4732.2246
    Longitude: 01205.1810
```

6. Aufgabe: Noch Unterrichtszeit? Dann extrahiere weitere Daten – z.B. Anzahl Satelliten, oder rechne die Positionsdaten um, oder ...