Application Note 36

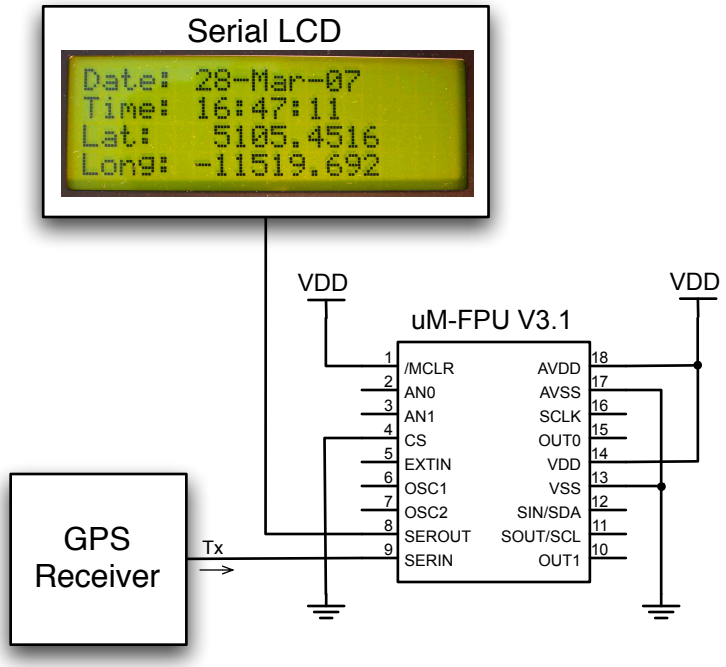
Reading GPS Data

Micromega Corporation

Introduction

GPS data is used in wide range of embedded systems applications. Reading and processing GPS data can consume a significant resources on a microcontroller. This application note describes how the uM-FPU V3.1 chip can be used to read data from a GPS receiver with very little overhead on the microcontroller. In fact, the example shown below doesn't even require a microcontroller.

Figure 1: Simple example that reads GPS data and displays the current location on a serial LCD.



The uM-FPU V3.1 chip has built-in support for NMEA sentence parsing and a full set of string instruction to facilitate reading the data fields. By offloading these task from the microcontroller, valuable resources can be saved for use by the main application, with the added benefit that the GPS data is already loaded on the uM-FPU V3.1 chip, ready for further navigational calculations using the powerful floating point instruction set. The uM-FPU V3.1 chip allows even small microcontrollers with limits resources to work with GPS data.

Brief Overview of the NMEA Data Interface

The NMEA-0183 data interface is commonly used for marine instruments and GPS receivers. In many GPS applications, this interface provides serial data continuously at a regular interval of one or two seconds. The data is readable ASCII text, consisting of a series of NMEA sentences as shown in the following example:

```
$GPGGA,140123.000,4415.5312,N,07622.4761,W,1,09,1.1,81.7,M,-34.5,M,,0000*57
$GPGSA,A,3,30,24,06,10,12,04,05,07,02,,,,2.1,1.1,1.8*3D
$GPGSV,3,1,12,10,74,219,46,02,69,060,49,04,31,087,36,06,30,309,46*75
$GPGSV,3,2,12,30,25,266,47,24,24,282,47,12,22,232,45,05,20,244,38*7A
$GPGSV,3,3,12,07,18,313,42,13,20,042,,29,12,171,,15,04,276,*79
$GPRMC,140123.000,A,4415.5312,N,07622.4761,W,0.79,95.83,260407,,*2E
```

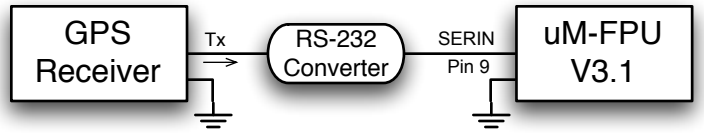
Each sentence begins with a dollar sign ($) and ends with a carriage return and linefeed. An optional checksum is usually included, consisting of an asterisk (*) and two checksum digits at the end of the sentence. Each field is separated by a comma. The first field indicates the sentence type, and the remaining fields contain the data. If the data for a field is unavailable, the field is empty. There are several different types of sentences.

Most applications generally only use a portion of the information provided. The NMEA data is scanned until a particular sentence types are recognized, then the sentences are parsed to extract the data fields of interest.

Connecting the GPS receiver to the uM-FPU V3.1 chip

The uM-FPU V3.1 chip has a built-in debug monitor that uses the `SERIN` and `SEROUT` pins to provide a serial interface for debugging and programming user-defined functions. When the debug monitor is not being used, the `SERIN` and `SEROUT` pins can be used for general purpose serial I/O. The `SERIN` instruction is used to read data from the `SERIN` pin, and the `SEROUT` instruction is used to send data to the `SEROUT` pin. Most GPS receivers that provide NMEA-0183 output, only require a single data line to be connected. The serial output from the GPS receiver is connected to the `SERIN` pin on the uM-FPU V3.1 chip as follows:

Figure 2: Connecting a GPS to the uM-FPU V3.1 chip.



Note: If the GPS serial output uses RS-232 signals, a converter will be required to convert the signal to the logic levels used by the uM-FPU V3.1 chip (e.g. MAX232 or equivalent).

Overview of NMEA sentence parsing on the uM-FPU V3.1 chip

The `SERIN` instruction has a special input mode for NMEA sentence parsing. Using this input mode, the uM-FPU V3.1 will scan for valid NMEA sentences, strip the $, checksum, carriage return and linefeed characters, confirm the checksum, then set the status byte and store the remaining string in a 200 byte NMEA sentence buffer. The buffer can store multiple sentences, allowing one sentence to be processed while other sentences are being received. The `SERIN` instruction reads NMEA sentence from the buffer, sets the status byte, and moves the sentence to the string buffer. The sentences can then be parsed, and the data fields extracted using various string instructions.

Example: Reading Latitude and Longitude from the GPRMC sentence

The following example reads the latitude and longitude fields from the GPRMC sentence and converts them to the values in degrees.

The default baud rate for most GPS receivers is 4800 baud. The following instructions enable the `SERIN` and `SEROUT` pins for serial I/O at 4800 baud, and selects the NMEA input mode.

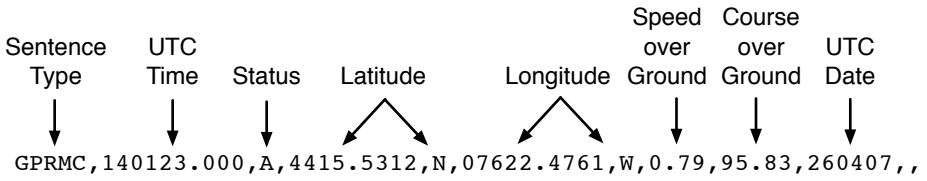
```
SEROUT,0,5          ; set baud rate to 4800
```

```
    SERIN,4             ; enable NMEA serial input
```

The `SERIN,6` instruction waits for the next available NMEA sentence, sets the status byte, transfers the sentence to the string buffer, and selects the first field of the string buffer. If you need to do other things while you wait for a sentence to be available, the `SERIN,5` instruction can be used. It checks the NMEA buffer and sets the status byte to zero (Z) if the buffer is empty, or non-zero (NZ) is at least one sentence is available. In this example, we'll let the `SERIN,6` instruction wait until the next sentence is available. Since we're only interested in the GPRMC sentence, we simply loop until the sentence type matches.

```
_loop:
    SERIN, 6                ; wait for NMEA sentence
    STRCMP, "GPRMC"         ; check for GPRMC sentence
    BRA, NZ, _loop          ; no, then get next sentence
    …                       ; yes, then process GPRMC sentence
```

The following diagram shows an example of the contents of the string buffer after receiving a GPRMC sentence, and a description of the various fields.



We're interested in the latitude and longitude fields. Latitude is stored in field 4 as DDMM.MMMM, where DD is degrees and MM.MMMM is minutes. Field 5 is the North/South indicator (N for North, S for South). Longitude is stored in field 6 as DDDMM.MMMM, where DDD is degrees and MM.MMMM is minutes. Field 7 is the East/West indicator (E for East, W for West). Navigation calculations require the latitude and longitude values expressed as decimal degrees, so we'll convert the values as follows:

Latitude = DD + (MM.MMMM / 60)
Longitude = DDD + (MM.MMMM / 60)

Instructions are used to select the DD or DDD portion of the field, convert it to floating point and store it as degrees. The MM.MMMM portion of the field is then selected, converted to floating point, divided by 60, and added to the degrees. If the North/South indicator is S (South), or the East/West indicator is W (West) the value is negated.
e.g.

`4415.5312,N` is converted to 44.258853 degrees latitude.
`07622.4761,W` is converted to -76.374603 degrees longitude.

Note: if you check the conversion with a calculator, you'll probably see a slight difference in the last digit of longitude. That's because the precision of a 32-bit floating point number is just over seven significant digits, so the eight digit may vary slightly from the calculator value.

The following example converts the latitude field to decimal degrees. To complete the conversion, the North/South indicator would be checked, and the value negated if the latitude was South.

```
    SELECTA, Degrees        ; get Degrees as reg[A]
    STRFIELD, 4             ; select the DDMM.MMMM field
    STRTOL                  ; get integer value (DDMM)
    LSET0
    LDIVI, 100              ; get degrees (DD)
```

```
   FLOAT                  ; convert to floating point

   LEFT                   ; get temporary reg[A]
   STRFIND, "."           ; find the decimal point
   STRDEC                 ; move selection point left by 2 characters
   STRDEC
   READVAR, 15            ; get selection point
   STRSEL, $80, 7         ; select MM.MMMM
   STRTOF                 ; convert to floating point
   FSET0
   FDIVI, 60              ; divide by 60
   RIGHT
   FADD0                  ; add to degrees
```

User-defined function

A similar conversion to the one shown above would be done for longitude. Several NMEA sentences have longitude and latitude values. This conversion can be generalized to handle both latitude and longitude and is an excellent candidate for a user-defined function. The following user-defined function is passed the field number in register 1, and returns the degree value in register 0.

```
;-------------------------------------------------------------------------------
#function NMEA_Degrees     ; convert latitude or Longitude to degrees
;
; input:     NMEA sentence stored in string buffer
;  reg[1]    latitude or longitude field number (long)
; output: reg[0]           degrees (float)

#asm
   LEFT                   ; get temporary reg[A]
   STRFIELD, $81          ; select DDMM.MMMM or DDDMM.MMMM field
   STRTOL                 ; get integer value (DDMM or DDDMM)
   LSET0
   LDIVI, 100             ; get degrees (DD or DDD)
   FLOAT                  ; convert to floating point

   LEFT                   ; get temporary reg[A]
   STRFIND, "."           ; find the decimal point
   STRDEC                 ; move selection point left by 2 characters
   STRDEC
   READVAR, 15            ; get selection point
   STRSEL, $80, 7         ; select MM.MMMM
   STRTOF                 ; convert to floating point
   FSET0
   FDIVI, 60              ; divide by 60
   RIGHT
   FADD0                  ; add to degrees

   LINC, 1                ; select E/W or N/S field
   STRFIELD, $81
   STRCMP, "S"            ; check for South or West
   BRA, EQ, _exit1
   STRCMP, "W"
   BRA, NE, _exit2
_exit1:
   FNEG                   ; yes, then negate value
_exit2:
   RIGHT                  ; return value in reg[0]
```

```
#endasm
```

Calling the User-defined Function

The following example waits for the next GPRMC sentence, then calls the `NMEA_Degrees` routine described above to convert latitude and longitude to degrees. These values can then be used for further calculations.

```
_loop:
   SERIN, 6               ; wait for NMEA sentence
   STRCMP, "GPRMC"        ; check for GPRMC sentence
   BRA, NZ, _loop         ; no, then get next sentence

   SELECTA, Latitude      ; get latitude
   COPYI 4, 1             ; (field 4, 5)
   FCALL, NMEA_Degrees
   FSET0

   SELECTA, Longitude     ; get longitude
   COPYI 6, 1             ; (field 6, 7)
   FCALL, NMEA_Degrees
   FSET0
```

Miscellaneous Items

Wait Instruction after SERIN,6

The `SERIN,6` instruction will only wait for an NMEA sentence if the instruction buffer is empty. This provides a way for the microcontroller to abort the instruction (e.g. by sending a `NOP` instruction). To ensure that the instruction buffer is empty when the `SERIN,6` instruction is executed, the microcontroller should wait for the Ready status after any `SERIN,6` instruction, or any FCALL instruction that calls a user-defined function that uses `SERIN,6`.

Infrequent readings

If GPS data is not being read continuously, the SERIN,0 instruction should be used to disable serial input when not required, and the SERIN,4 instruction should be used to enable NMEA input before taking a reading. This eliminates the serial input interrupt overhead when not required, and avoids overrun errors.

Error Checking

The SERIN,6 instruction sets the status byte for each sentence. If a sentence is valid, the status byte is set to greater-than (GT). If not valid, the status byte is set to less-than (LT), and bit 4 is set if an overrun error occurred, and bit 5 is set if a CRC error occurred. To recover from an overrun error, sentences can be flushed from the buffer, or the serial input can be disabled and re-enabled to start with an empty buffer.

Overrun Errors

The NMEA sentence buffer is 200 bytes in length. The average NMEA sentence is about 60 to 70 bytes, so the buffer can hold two to three sentences of average length, or more if there are smaller sentences. The sentences should be read at a rate that is fast enough to avoid overrun. For example, if the GPS is sending data at 4800 baud, and the average sentence length is 60 bytes, then sentences arrive about every 125 milliseconds and must be processed at a faster rate to avoid overrun. Sending data to a serial port on the microcontroller at baud rate that's equal to or less than the baud rate for the incoming GPS data can be very time consuming, and is a common source of overrun errors. If GPS data is only required periodically, the best solution is to disable the input between readings.

Debugging Tips

When the SERIN pin is used for GPS data input, the debug monitor is no longer available. This can make it difficult to debug code that processes NMEA sentences, but since the NMEA sentence is processed from the string buffer, an effective method for debugging is to simulate the GPS data by loading the string buffer with a known NMEA string and then executing the code to parse the data. Sample data can easily be captured from a GPS using a terminal emulator, and sample sentences can be copied to your test application for processing.

In your test application, temporarily comment out the SEROUT and SERIN instructions, enable the debug monitor, and load the string buffer with a `STRSET` instruction.
e.g.

Temporarily replace:

```
SERIN, 6
```

with:

```
STRSET, "GPRMC,140123.000,A,4415.5312,N,07622.4761,W,0.79,95.83,260407,,"
STRFIELD, 1
```

The debugger can then be used to develop and test the parsing code. Once the parsing code is tested, remove the `STRSET` and `STRFIELD` instructions, enable the `SEROUT` and `SERIN` instructions, and you're ready to go with live data.

Further Information

See the Micromega website (http://www.micromegacorp.com) for additional information regarding the uM-FPU V3.1 floating point coprocessor, including:

uM-FPU V3.1 Datasheet
uM-FPU V3.1 Instruction Set