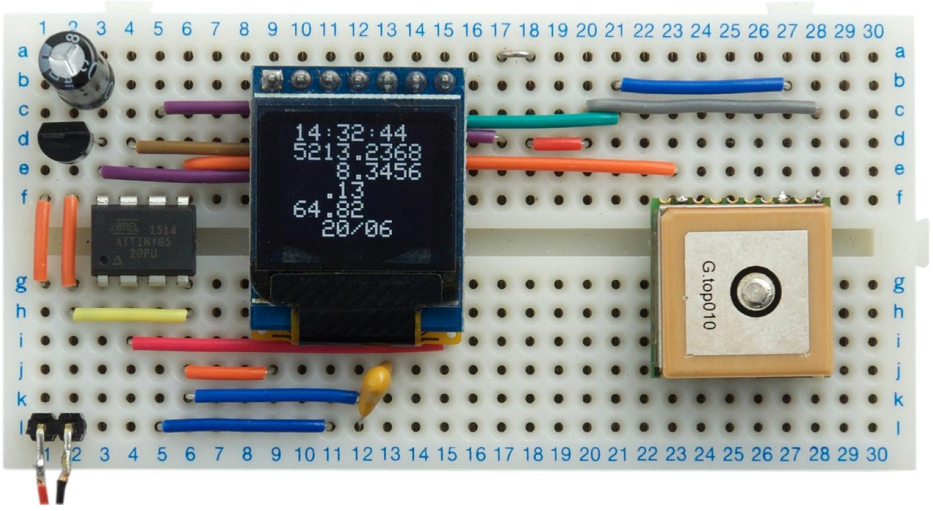
Flexible GPS Parser

20th June 2017

This article describes a compact GPS parser to decode the NMEA sentences from a range of low-cost GPS modules. It runs in a small amount of memory, and uses integer arithmetic, and so is small enough to fit on an ATtiny processor such as the ATtiny85. I've tested it with four popular low-cost GPS modules, each available for under $20: the GTPA010/PA6C, Ublox NEO-6M, GP-20U7, and VK2828U7G5LF. I also describe the test program I used, which shows the main GPS parameters on a 64x48 OLED display:



GPS Parser Display shows the GPS parameters from a GPS module.

Introduction

My original GPS parser Minimal GPS Parser [Updated] used a novel method to decode NMEA messages from a GPS decoder, using a state machine driven by a pattern which was used to match the NMEA sentences.

The original version was designed for use with a particular GPS module, but I have now improved the program so that one version of the **ParseGPS()** routine is flexible enough to work with a wide range of GPS modules. I have tested it with four popular low-cost GPS modules in the $10 to $20 price range, and I give details of these below. The test program runs on an ATtiny85, using four I/O pins to drive the display, and one to process the serial input from the GPS module.

Decoding the GPS string

The information in the NMEA string from the GPS module is decoded by the routine **ParseGPS()**. This decodes each character as it is received, rather then buffering the characters and then using string operations on them like other GPS libraries. It uses a simple state machine to ensure that each character is processed quickly, so the work can be done in the time between each received character:

```
// Example: $GPRMC,092741.00,A,5213.13757,N,00008.23605,E,0.272,,180617,,,A*7F
char fmt[]="$GPRMC,dddtdd.ds,A,eeae.eeee,l,eeeae.eeee,o,jdk,c,dddy";

int state;
unsigned int temp;
long ltmp;

// GPS variables
volatile unsigned int Time, Csecs, Knots, Course, Date;
volatile long Lat, Long;
volatile boolean Fix;

void ParseGPS (char c) {
```

Search

Recent posts

▼ 2020
Simple sprite routines for the Wio Terminal
Saving screenshots from a TF display
Simple Sprite Routines for th PyGamer/PyBadge
Reading the PyBadge Display
Minimal ATmega4809 on a Breadboard
Big Time
Four Sample Player
Mega Tiny Time Watch
► 2019
► 2018
► 2017
► 2016
► 2015
► 2014

Topics

► Games
► Sound & Music
► Watches & Clocks
► GPS
► Power Supplies
► Computers
► Graphics
► Thermometers
► Tools
► Tutorials

By processor

AVR ATtiny

► ATtiny10
► ATtiny2313
► ATtiny84
► ATtiny841
► ATtiny85
► ATtiny861
► ATtiny88

AVR ATmega

► ATmega328
► ATmega1284

AVR 0-Series and 1-Series

► ATtiny3216
► ATtiny402
► ATtiny414
► ATmega4809

Latitudes in the southern hemisphere and longitudes in the western hemisphere are represented by a negative number.

This is designed to allow the arithmetic to be done using long integers, and is ideal for parsing the values returned by the GPS module. If you prefer to work in millionths of a degree like the other GPS libraries simply multiply by 5/3.

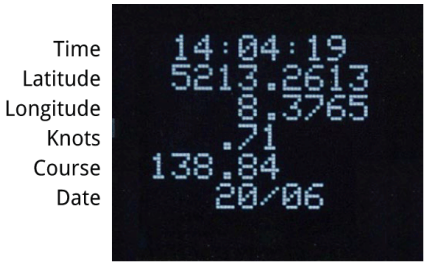
The routine decodes the time, latitude, longitude, speed, track, and date. It ignores the checksum.

The routine currently only decodes the RMC string, but it could easily be extended to decode other NMEA strings, such as the GGA string if you want to read the altitude information.

With an 8MHz clock the **ParseGPS()** routine takes at most about 20µsec to process one character. For comparison, at 9600 baud the characters arrive at intervals of approximately 1000µsec.

Text display

To test each of the GPS modules I used a 64x48 OLED display to display the GPS parameters from the module:



The display has an SPI interface and is available from Aliexpress [1], or a similar one is available from Sparkfun [2].

The display program is a version of my ATtiny85 Graphics Display project. To save memory I rewrote the display interface to write characters directly to the display memory, without using a memory buffer. This results in a simpler interface when you don't need to plot graphics on the display like the original project.

It uses the following routines:

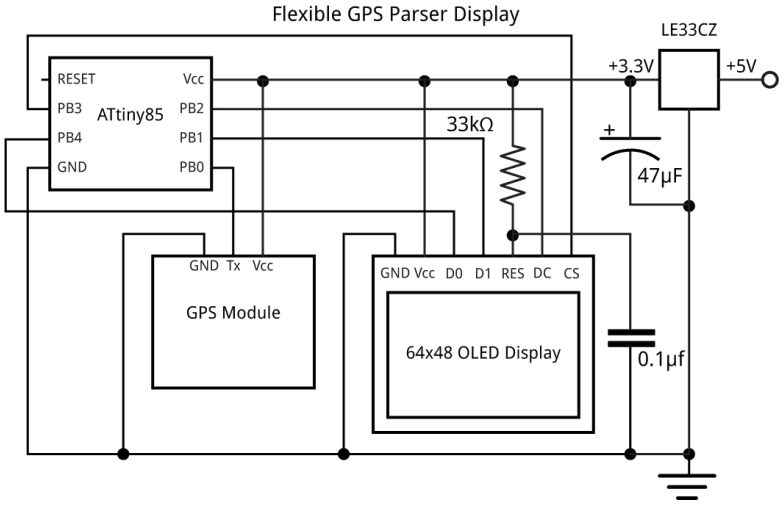
PlotInteger() - plots an unsigned integer, up to 65535, to the display, displayed with two decimal places.

PlotAngular() - plots a latitude or longitude to the display in the format DDDMM.MMMM where DD are the degrees and MM.MMMM are the minutes.

To convert from this format to degrees and fractions of a degree divide the number of minutes by 60 and add it to the number of degrees. For example, if the latitude in DDDMM.MMMM format is 5213.2375, the latitude in degrees is 52 + 13.2375/60 = 52.220625 degrees. You can use this format to type the latitude and longitude into Google maps, and display the location.

The circuit

Here's the circuit of the whole project:



Circuit for the GPS Parser Display which shows the GPS parameters from a GPS module.

```
  if (c == '$') { state = 0; temp = 0; ltmp = 0; }
  char mode = fmt[state++];
  // If received character matches format string, return
  if (mode == c) return;
  char d = c - '0';
  // Ignore extra digits of precision
  if (mode == ',') state--;
  // d=decimal digit; j=decimal digits before decimal point
  else if (mode == 'd') temp = temp*10 + d;
  else if (mode == 'j') { if (c != '.') { temp = temp*10 + d; state--; } }
  // e=long decimal digit
  else if (mode == 'e') ltmp = (ltmp<<3) + (ltmp<<1) + d; // ltmp = ltmp*10 + d;
  // a=angular measure
  else if (mode == 'a') ltmp = (ltmp<<2) + (ltmp<<1) + d; // ltmp = ltmp*6 + d;
  // t=Time - hhmm
  else if (mode == 't') { Time = temp*10 + d; temp = 0; }
  // s=Centisecs
  else if (mode == 's') { Csecs = temp*10 + d; temp = 0; }
  // l=Latitude - in minutes*1000
  else if (mode == 'l') { if (c == 'N') Lat = ltmp; else Lat = -ltmp; ltmp = 0; }
  // o=Longitude - in minutes*1000
  else if (mode == 'o') { if (c == 'E') Long = ltmp; else Long = -ltmp; ltmp = 0; }
   // k=Speed - in knots*100
  else if (mode == 'k') { Knots = temp*10 + d; temp = 0; }
  // c=Course (Track) - in degrees*100. Allow for empty field.
  else if (mode == 'c') {
    if (c == ',') { Course = temp; temp = 0; state++; }
    else if (c == '.') state--;
    else { temp = temp*10 + d; state--; }
  }
  // y=Date - ddmm
  else if (mode == 'y') { Date = temp*10 + d ; Fix = 1; state = 0; }
  else state = 0;
}
```

Routine to parse the RMC NMEA sentence from the GPS module.

Here's how it works:

As each character is received from the GPS sentences it is checked against the format string **fmt[]**, which is a RMC (Recommended Minimum) NMEA sentence with the data replaced by lower-case letters. Each character is tested as follows:

- If the received character is a '$' the state pointer is reset to the start of the **fmt[]** string.
- If the received character matches the next character in the **fmt[]** string the match succeeds.
- If the character in the **fmt[]** string is a lower-case letter it specifies how to process the received character, and the match succeeds.
- Otherwise, the match fails and the state pointer is reset to the start of the string.

The letters 'd', 'e', and 'a' read a received digit and accumulate the value in one of the temporary variables **temp** or **ltmp**. For example, a 'd' in the **fmt[]** string reads a decimal digit into the variable **temp**.

The letter 'j' accumulates digits until a decimal point is encountered. This caters for fields with a variable number of digits before the decimal point.

The letter 'c' accumulates digits until a comma is encountered, and ignores a decimal point. This caters for an empty field, output by some GPS modules.

The last letter in each field copies the temporary variable into an appropriate GPS variable; for example, the letter 'l' copies **ltmp** into the latitude variable, **Lat**.

The routine stores the angular measures, latitude or longitude, in signed units of 1e-4 arc minutes. Thus one degree is represented as 600,000 units:

```
const long DEGREE = 600000;
```

Because some of the modules are 3.3V I've included a LE33CZ low-dropout regulator to drop the supply voltage from 5V, or from a 3.7V LiPo battery, but this isn't necessary if you're using a module that is happy with 5V. The 33kΩ resistor and 0.1µF capacitor ensure that the display is reset correctly when power is first applied.

USI UART

The ATtiny85 doesn't include a hardware UART, so the serial input from the GPS module is decoded by a software UART, using the ATtiny85's USI interface, as described in the earlier article Simple ATtiny USI UART 2.

The processor uses the 8MHz internal oscillator, to avoid the need for an external crystal. My ATtiny85 worked fine with the default setting, but because the serial interface is quite fussy about the data rate you may need to calibrate the oscillator to an accurate frequency using the OSCCAL register.

To do this, load the following program:

```
void setup () {
  OSCCAL = 0xB2;
  TCCR0A = 1<<COM0B0 | 2<<WGM00;// Toggle PB0
  TCCR0B = 0<<WGM02 | 2<<CS00;  // Divide by 8
  OCR0A = 249;                  // Divide by 250
  pinMode(1, OUTPUT);           // Adjust for 2000 Hz on PB1
}
```

Measure the frequency on PB1 with a frequency meter, or a multimeter with a frequency range. I use the Victor VC921, available from Adafruit [3] or Banggood [4]. Then recompile the program with different values of OSCCAL until you get the frequency as close as possible to 2000Hz, using large jumps in value first, then smaller jumps as you get close to the correct frequency.

Compiling the program

I compiled the program using Spence Konde's ATTiny Core [5]. Choose the **ATtiny25/45/85** option under the **ATtinyCore** heading on the **Board** menu. Then choose **Timer 1 Clock: CPU**, **B.O.D. Disabled**, **ATtiny85**, **8 MHz (internal)** from the subsequent menus.

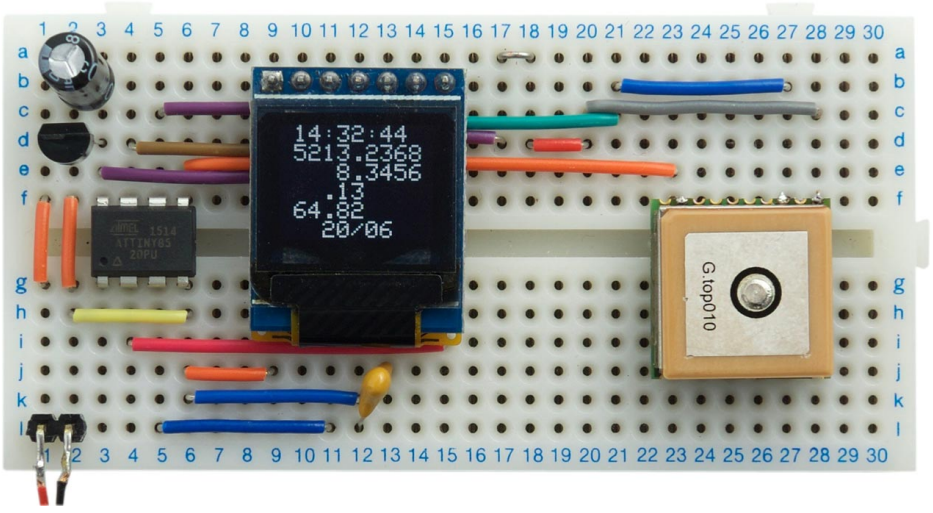
I programmed the ATtiny85 using Sparkfun's Tiny AVR Programmer Board; see ATtiny-Based Beginner's Kit. Choose **Burn Bootloader** to set the fuses appropriately, and then upload the program.

Here's the whole Flexible GPS Parser Display program: Flexible GPS Parser Display Program.

GPS Modules

Here's a survey of the four low-cost GPS modules I've tested with the program; they all include built-in aerials, and are rated at 2.5m positional accuracy.

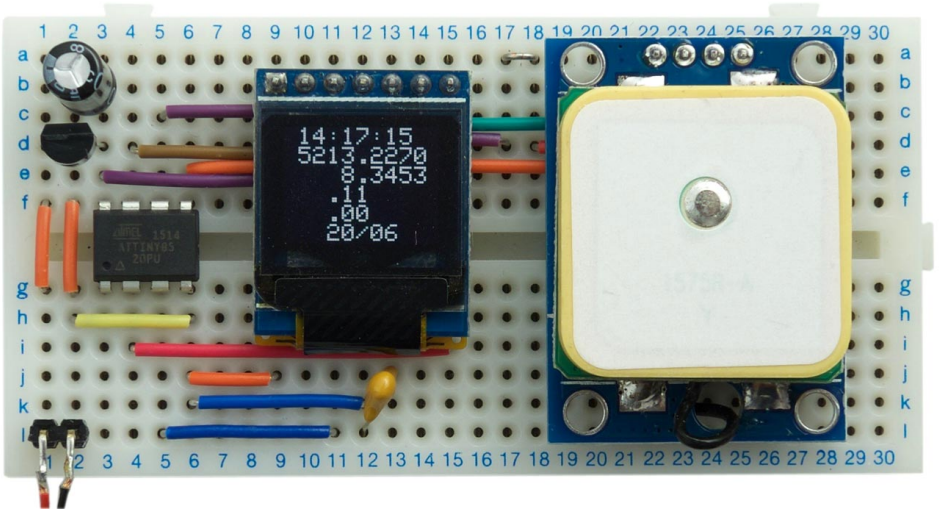
GTPA010/PA6C



The GTPA010 or PA6C is a small surface-mount module available from Aliexpress [6] or from PV Electronics in the UK [7]. Although it's a surface-mount module, it's quite easy to fit three wires to the necessary connections; see Odometer/Speedometer Pendant [Updated]. Alternatively, Adafruit provide a breadboard-friendly Ultimate GPS Breakout with a later version of the same module [8].

This is a 66-channel receiver which operates from 3.0V to 4.3V, so you will need to use a regulator if you have a 5V supply. I measured the current consumption at 3.3V as about 22mA.

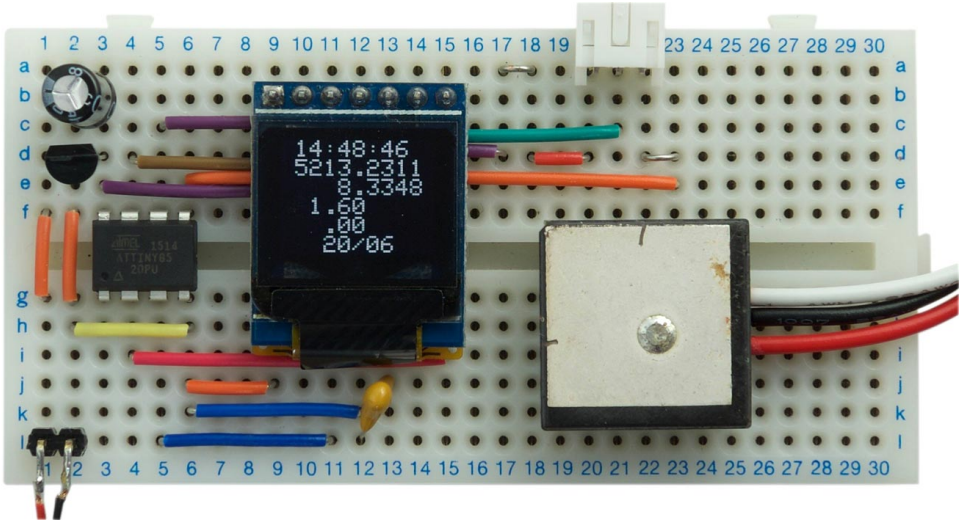
Ublox NEO-6M



The Ublox NEO-6M is a low-cost receiver, available from HobbyKing [9] or HobbyTronics in the UK [10].

This supports 50 channels, and the module is compatible with 3.3V to 5V. At 3.3V I measured a current consumption of about 45mA.

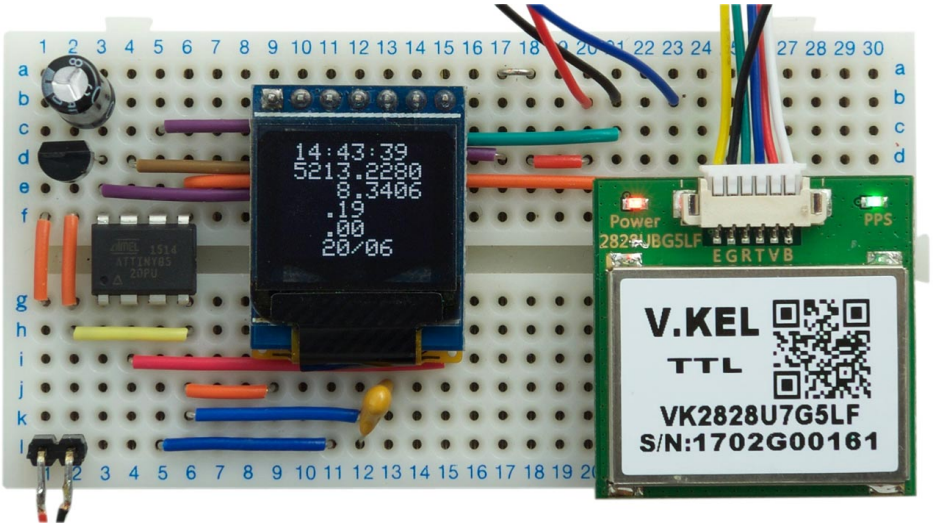
GP-20U7



This is a small GPS module available for under $20 from SparkFun [11] or HobbyTronics in the UK [12].

It supports 56 channels, but note that it is designed to work from 3.3V, so you should use a regulator to drop the voltage down from a 5V supply. At 3.3V I measured a current consumption of about 40mA.

VK2828U7G5LF



This is a very low cost GPS receiver from Banggood [13] or AliExpress [14].

It supports 56 channels, and operates from 3.3V to 5.5V. At 3.3V I measured a current consumption of about 45mA.

Updates

15th August 2017: Removed a redundant reference to **Buffer[]** from the program (ignored by the compiler).

20th September 2017: The **PlotAngular()** routine in the original version of the program displayed the wrong values for the latitude or longitude when the **number** parameter was negative. This is fixed in the latest version of the program, and in addition, the values are now displayed with a N, S, E, or W suffix as appropriate. I'm grateful to Miles Treacher for reporting this problem.

23rd September 2018: Corrected the declaration of the character map from uint32_t to uint8_t.

---

1. ^ White 0.66 inch OLED Display Module 64x48 from e_goto Processors on Aliexpress.
2. ^ SparkFun Micro OLED Breakout on Sparkfun.
3. ^ Pocket Autoranging Digital Multimeter on Adafruit.
4. ^ VICTOR VC921 LCD Digital Voltmeter Ohmmeter on Banggood.
5. ^ ATTinyCore on GitHub.
6. ^ MTK3339 ultra-small GPS module with Dual Antenna on AliExpress.
7. ^ Micro GPS Module on PV Electronics.
8. ^ Ultimate GPS Breakout on Adafruit.
9. ^ NEO-6M GPS Module on HobbyKing.
10. ^ Ublox NEO-6M 56 Channel GPS Receiver on HobbyTronics.
11. ^ GPS Receiver - GP-20U7 (56 Channel) on SparkFun.
12. ^ 56 Channel GPS Receiver - GP-20U7 on HobbyTronics.
13. ^ Geekcreit 1-5Hz VK2828U7G5LF TTL Ublox GPS Module With Antenna on Banggood.
14. ^ VK2828U7G5LF GPS Module with Antenna on AliExpress.

Next: Four-Channel Thermometer

Previous: Tiny Face Watch

johnsondavies Mod ➚ LOVE • 4 months ago

You mean an I2C one? Probably, but you'd have to change the program.

^ | v • Reply • Share ›

yau wai • 10 months ago

Hi, can I use Quectel's L80-R instead of the G.top010? I can't seem to find it anymore nowadays. It also has 66 channels. I don't really want to use the others as they are quite big and draw higher current. The L80-R is in a similar package size as well.
Thanks in advance!

^ | v • Reply • Share ›

johnsondavies Mod ➚ yau wai • 10 months ago

They're both based on the MT3339 chipset, so I think they should be very similar, although it looks like the board connections are different. Let me know how you get on.

^ | v • Reply • Share ›

Chetan Bhargava • 3 years ago

Has anyone tried to install ATTinyCore boot-loader on an aftermarket Digispark PCB?

^ | v • Reply • Share ›

Chetan Bhargava • 3 years ago • edited

Wouldn't it be more economical to use readily assembled digispark PCBs from ebay? Ready assembled Digispark PCB are available @ $1.40 (free shipping) and on the other hand a PDIP ATTiny85 is available for $1.23 at Mouser (Plus shipping). Reason I chose PDIP is because SO versions can't be directly used on breadboards (while Digispark PCB can be).

Your program complies on both Digispark and ATTinycore in Arduino env. ATTinycore give more flash to the application as compared to Digispark. I really want to give it a try but don't have any PDIP ATTINY85. :-)

In my opinion, it would be a lot effective to build such useful project (not on breadboard) using a small circuit board and some wiring.

The concept presented is right-on.

^ | v • Reply • Share ›

John • 3 years ago

I have been reading your article with interest !! I am busy with a project with an Arduino Mega R3. I have various sensors on it (temp, pressure) together with ..... a GPS !!! I have the NEO6M. I am using the TinyGPS++ library and it works, but as soon as I add calculations and updates to the graphics display, at some stage it starts interfering with the GPS data. It works on the principle of : interrupt routine clocks data into buffer and then only does parsing start. I am going to enter your routine here. I use the RMC and the GGA streams. Wonder if one can make an adaption for the GGA? Neat work !!!

^ | v • Reply • Share ›

John ➚ John • 3 years ago

I have made an attempt at adding the GGA stream to your code ...

```
void serialEvent1()

{

if (Serial1.available())

{

char inChar = (char)Serial1.read();

ParseGPS(inChar);

ParseGPS1(inChar);

}

}
```

see more

^ | v • Reply • Share ›

John ➚ John • 3 years ago

Ooooops. I forgot to put this in.....

```
char
fmt1[]="$GPGGA,dddtdd.ds,eeae.eeeee,l,eeeae.eeeee,o,m,dk,?.??,ddddjc,?,????,?,

char fmt[]="$GPRMC,dddtdd.ds,A,eeae.eeee,l,eeeae.eeee,o,jdk,c,dddy";
```

Reply • Share ›

johnsondavies Mod ➚ John • 3 years ago

Could you email me your GPS code and I'll have a look at it - it gets a bit messed up by Disqus and it's hard to read.

Reply • Share ›

YuryMonZon ➚ johnsondavies • 2 years ago

Great parser!
Any updates on GGA implementation? Thanks a lot!

Reply • Share ›

johnsondavies Mod ➚ YuryMonZon • 2 years ago

Sorry, forgot about this. I'll have another think about it. David

Reply • Share ›

John ➚ johnsondavies • 3 years ago

Thank you. I have sent you a mail !!! Regards.

Reply • Share ›

Miles Treacher • 3 years ago • edited

This module looks very interesting, I have one on order, will let you know what it's like when I have built it.

http://shop.qrp-labs.com/qlg1
"The kit features a large PCB, specifically to provide a large ground-plane that provides 4.5dBic gain relative to a 30 x 30mm ground-plane"

Reply • Share ›

bgolab • 3 years ago

Hi,
As usual perfect content, elegant design.

GTPA010/PA6C - this is nice from the current consumption, and is considered by some guys as better than NEO-6M.

I wonder of you have any other comments when comparing these two.
Do you find any of them better than the other? Why?

Thanks,
Bogdan

Reply • Share ›

johnsondavies Mod ➚ bgolab • 3 years ago

Thanks for your comments! The GTPA010 is definitely my favourite because if its small size and low current consumption.

Reply • Share ›