

Contents

A GPS LOGGING WITH A RABBIT?	2
INTRODUCTION	2
WHY USE A RABBIT?	3
EMBEDDED C FROM ZWORLD.....	3
THE LOGGING PROGRAM.....	4
READING THE GPS DATA	4
PRINTING OUT THE RECORDED DATA.....	6
FUNCTION 2, READING THE PRESS BUTTON AND OPERATING THE LEDs.....	6
PART 2	8
WHERE WE ARE - THE DISPLAY PROGRAM.....	8
CAPTURE THE GPS DATA.....	8
PLOTTING.....	8
COLLECTING THE 'LIVE DATA'	10
SUMMING UP.....	12
LISTING 1 - RABBIT C PROGRAM LISTING	13
WHERE TO GET THINGS	15

A GPS Logging with a Rabbit?

Summary: The Global Positioning System is used more and more and can be useful for many projects. Combine this with the new Rabbit Semiconductor microprocessor kit including a C compiler gives a powerful programmable GPS data logger with 128K bytes of battery backed RAM.

Introduction

I am working on a project, which uses a Global Positioning System (GPS) receiver to measure position. Part of the problem, or should I say challenge, is to save parts of the data so that finally it is used to plot the receiver position over time. The GPS modules produce RS232 data, which can be read on the screen through a terminal emulator like Hyperterminal [1] on Windows 95/98.

What I need is a device to store the RS232 text lines. One option is to carry around a portable PC along with the rest of the system. That makes quite a large and heavy unit, which I do not want, so I looked for a different way to make the logger.

There are many single chip microprocessors, such as the PIC range, which can easily do the processing, but they have only a few hundred bytes of RAM. This is not enough for my design, but looking at the electronic magazines I saw the adverts for a new microprocessor called the Rabbit. I guess what caught my eye was the

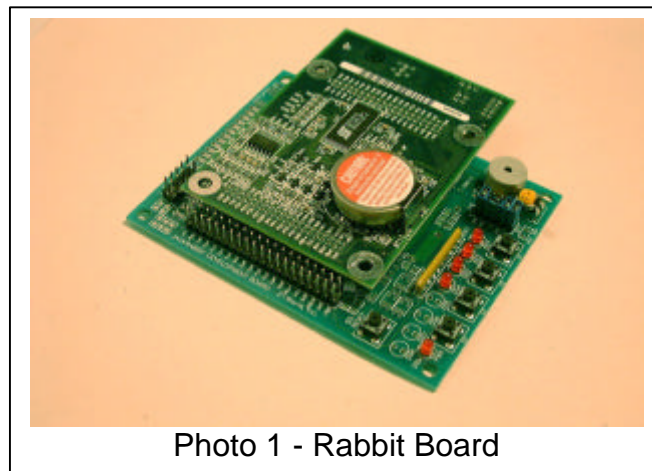


Photo 1 - Rabbit Board

complete system which consists of a development board with 128K bytes of RAM and 128K bytes of Flash memory, and a C compiler complete with development environment. [See Photo 1 - Rabbit Board.] The price of \$99.00 was also affordable, being on a par with the new Flash PIC development kit, but with the added advantage of being C programmable. The price is somewhat higher today, though. Off I went to the web site and ordered the kit [3]. Within a few days the package arrived with all I needed except the power supply. The included one was a 110V US version. I just replaced it with a 220V UK power supply. The kit takes about 150mA with the LEDs turned on and it has its own regulator to provide 5V for the circuit, so a small power module is OK. I see the Rabbit kit can be bought in the UK, from Impulse Corporation Limited or the distributor 2001 [4].

Why use a Rabbit?

The Rabbit is a variation on the Z80 processor, but with many differences which are designed to assist programming with the C language. The most useful part of the system from my point of view is the 128K bytes of RAM. That is plenty of memory to store the data from the GPS logger. The RAM is battery backed, so it could be used to store data when the program isn't running. I haven't figured out how to do that yet, though!

If you get round to building your own hardware, the Rabbit microprocessor provides most of the 'glue' logic to connect four memory chips with no extra logic required. There are four serial ports available though one is used for programming and debugging, but it can be reused. The development kit has two of the serial ports connected to an RS232 interface chip and a third to an RS485 chip. Also the large Flash memory allows easy changes to the program. Overall it is certainly worth looking at for a variety of uses, and its always good to try a new 'toy'.

The Windows based development software provides a C compiler with built in assembler, an editor and a visual debugger. You get a lot of value out of this kit. There are several examples to try straight away. So within minutes you can have the serial cable plugged into the PC and the other end into the Rabbit, load an example, press the run button and within 30 seconds (at least on my computer) the LEDs can be flashing.

Embedded C from ZWorld

Having read the documents my first change was to make sure the program was running in RAM using the menu Options/Compiler. If you use the Flash all the time it will quickly wear out. The C system has its own bios, which is compiled and loaded before your own program. The source is included so that you can see how it works and maybe adapt it to your own hardware. All the code is downloaded through a facility of the microprocessor, which provides a hardware loader using either the serial port or the parallel slave port. The development system uses the serial port, so loading is quick and simple, and your own hardware could do the same. That is another convenience of this clever design. I'm getting to like it even more. When your program is complete, you change the compiler to load into Flash and then remove the serial debugging cable. The loaded program runs when you apply the power. The ease of development is all worked out. Now all I need to do is devise my code.

The ZWorld version of C is a little different from the ones you will see from others since it is designed to compile quickly and doesn't use header files in the same way. If you are used to C or Basic you should be able to make things work quite easily. Have a good look at the examples and you will see how things work.

There are several extensions to C which are there to help program in real time without having to use difficult program techniques. For instance you do not need a real-time operating system to carry out more than one function at a time. The various 'co' functions are used to group program lines into blocks, which in effect are parallel executing programs. It's probably easier to understand by seeing how my program uses them.

The Logging Program

My program does three things in parallel:

- 1 Read the GPS serial data, analyse and store the data
- 2 Print out the stored data
- 3 Read the press button and operate the LEDs.

Each uses a 'costate' to group the instructions related to how it works. I've added line numbers to Listing 1 so that I can point out which part is which.

Much of the program is based on one of the samples provided to get you going with Rabbit system, but lets look at it in some detail.

Reading the GPS data

Function 1, reading the GPS serial data uses lines 30 to 57. Line 39 has the wfd, wait for done, keyword which tells the costate to wait for the rest of the line to finish. In this case that means, wait for the line of serial characters to arrive. The cof_serBgets function refers to serial port B and is a cofunction which allows the program to continue if the whole line has not arrived. With the debugger, you can single step to this instruction and see that the program jumps to after the end of the costate block if it is not complete. So you can see that waiting, maybe for a long time, for the line of text to arrive doesn't stop the rest of the program working. This is the great advantage of the idea of cofunctions. When the line of text does arrive the program runs on to the next lines of code. The rest of this costate looks for a text line starting with \$GPGGA.

Its time to explain a little about the data which comes out of the GPS module. The module itself is a Garmin GPS25-LVS, which you can buy from Maplin, though, there are several other manufactures who provide similar devices. You could also use any GPS receiver, which has an RS232 output and sends standard NMEA (National Marine Electronics Association) sentences.

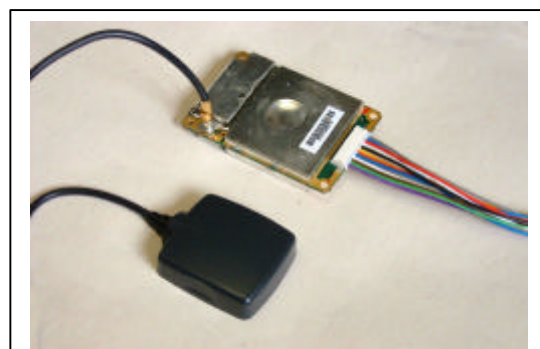


Photo 2 - Garmin Module and Antenna

[Photo 2 - Garmin Module and Antenna.] As well as the module you will also need an antenna. I chose the Sigem active antennae which needed the

coaxial connector changing, but was much cheaper than any other in the catalogue. It also contains a magnet for temporary mounting say, on the roof of a car. The module also has an unusual ribbon cable connector, but it comes with a length of cable to make your own connections. I made connections for the 5V-power supply, at about 150mA, and three wires for the RS232 serial interface. Most RS232 GPS data is sent out at 4800 baud, which is fast enough to allow a burst of sentences every second. You can usually change the baud rate, though it is best not to go much slower.

GPS Sentences

The NMEA standardises the sentences used by GPS systems and other related electronic devices. A sentence is just a line of text containing identification and data. The first part of the sentence is something like \$GPGGA and ends with a carriage return character. The \$GP identifies the data as GPS information. The GGA string says this is the Global Positioning System Fix Data. The rest of the line is made up of fields separated by commas. I am only interested in the GGA sentence, because it contains the position and time information. The fields are made up of:

\$GPGGA,daytime,latitude,north/south,longitude,east/west,GPS quality, number of satellites,horizontal precision,"M",geoid height,"M",differential, differential reference station,*checksum

daytime - UTC (GMT) time as hhmmss (hours minutes seconds) NOT local time

latitude - ddmm.mmmm (degrees minutes and decimal minutes)

north/south - N or S (hemisphere of the latitude measurement)

longitude - dddmm.mmmm (degrees minutes and decimal minutes)

east/west - E or W (east or west of 0 degrees for the longitude measurement)

GPS quality - 0,1 or 2 (0 no fix, 1 just GPS, 2 differential measurement)

number of satellites - 00 to 12 (tracked satellites)

horizontal precision - 0.5 to 99.9 (measure of precision)

"M" - just the capital letter M

geoid height - -999.9 to 9999.9 (antenna height)

"M" - just the capital letter M

differential - seconds since last valid differential data

differential reference station - 0000 to 1023 (differential station ID)

***checksum** - a * followed by a two character checksum

```
$GPGGA,181930,5314.6826,N,00208.1973,W,1,03,2.8,146.9,M,49.4,M,,*57
$GPGGA,181945,5314.6812,N,00208.2037,W,1,03,2.8,146.9,M,49.4,M,,*58
$GPGGA,182000,5314.6812,N,00208.2037,W,1,00,2.8,146.9,M,49.4,M,,*50
$GPGGA,182015,5314.6812,N,00208.2037,W,0,00,,M,,M,,*42
$GPGGA,182030,5314.6812,N,00208.2037,W,0,00,,M,,M,,*45
```

Figure 1 - GPS sentences

Figure 1 shows some of the data that I recorded. The first three lines show all the data I need. The bottom two lines show the data when there are not enough satellites in view. The good data uses only three satellites so it may not be very accurate. In particular the error in height may well be a long way out. Height errors are often twice the error in Latitude and Longitude.

The data is good enough to test the system, though. Going back to the program, lines 42 to 47 check that the GPS data contains the GGA string and selects readings at 15-second intervals. The GPS data is sent out every second, but to reduce the amount of data I need to store, only every 15th value is put into memory. I could use the same technique to halve the volume of data again if I selected 30-second steps. The if statement may look a bit verbose, but embedded code has to run quickly otherwise you can run out of time to do everything. The byte by byte comparison is quite fast. An alternative might be to use string comparisons, but that is almost certainly slower to execute. Remember embedded systems may be programmed in C but care must be taken not to waste time or memory.

Line 48 copies the text line into an array stored in RAM. The integer `reci` counts through the array. When it reaches the end of the array, lines 50 and 51 point it back to the start. Finally the flag `vswitch1` is used to indicate the state of an LED. Each time a valid GPS line is received the LED changes from on to off or off to on depending on its last state. This is really a confidence indicator to tell me the program is working. At line 41 `vswitch2` is used in a similar way, but flashes more often, several times a second, in response to any completely received GPS text line.

Printing out the recorded data

Well that gives me a way to filter out what I want and save it into RAM. Function 2, the next piece of code, lines 60 to 76, is to print the saved data. It uses serial port C to write to. For testing this was plugged into my PC running Hyperterminal. Once the code runs, it prints each logged GPS line. You could capture the text from Hyperterminal and put it into a spreadsheet to analyse it, or even plot a graph of position. The code is placed in a costate so that it runs without stopping the rest of the program. Again I use another integer, `vswitch3`, to flash another LED to show this part of the program is active.

Function 2, reading the press button and operating the LEDs.

The last costate in lines 79 to 97 is used to detect if the button is pressed, and has logic to debounce the inevitable multiple makes and breaks when a mechanical contact changes. The whole costate is taken from the example but is very instructive in showing how to insert real time delays without locking up the rest of the program. Stepping through it with the debugger is most revealing. The logic to operate the LEDs is at the top of the program in lines 31 to 36. Each `vswitch` is compared with the state of the pin attached to a LED, and the program sets or resets the corresponding values.

Well that's the logging part of my project. I am very pleased with how easy it is to use the Rabbit with its C compiler. The 128K bytes of RAM and the 128K of FLASH, gives me plenty of room to add other facilities, but that will have to be at another time, though. [See Photo 3 - All the parts.]

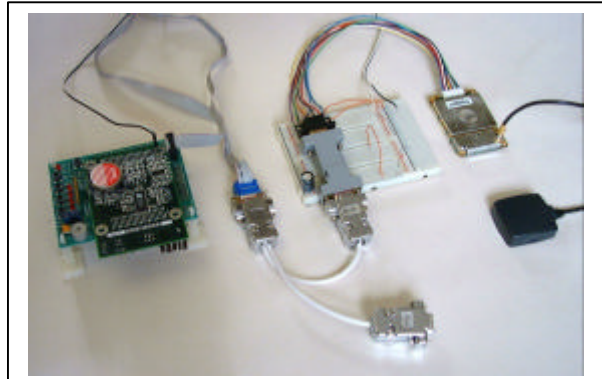


Photo 3 - All the parts

Part 2

Where we are - the Display program

The first part of this article showed how to collect the position data, but now its time to find a way to plot the data on a map.

Capture the GPS data

First I can use Hyperterminal to capture the data from the Rabbit board. The capture buffer can be set to save to a file from the Transfer/Capture Text menus, or you can mark text and use Copy and Paste to transfer the screen contents. Using a file means you don't need to bother with real time collection of the serial data, which makes the display program a lot simpler. It also makes testing easier, since the data never changes between test runs. See Figure 2.

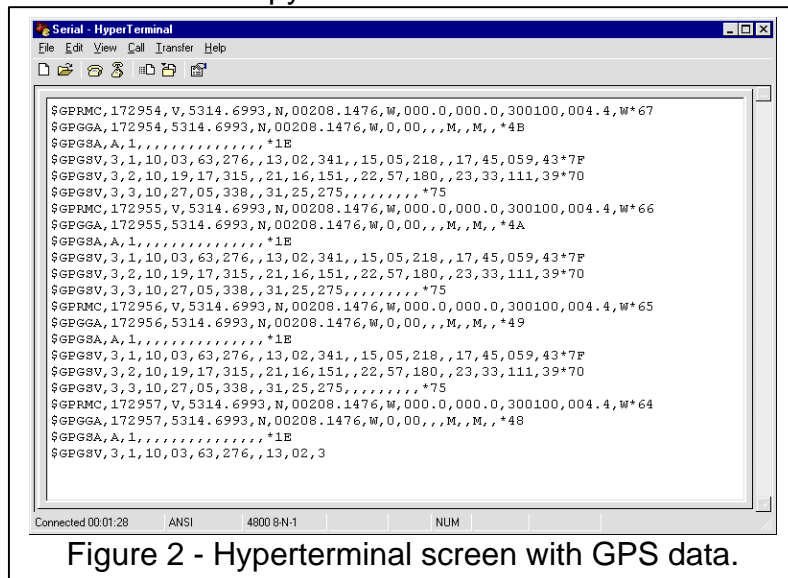


Figure 2 - Hyperterminal screen with GPS data.

Next I need a program which plots Latitude and Longitude data.

There are several ways to do plotting from the code of a program. In Windows programs it is possible to use almost any rectangular window, picture box or text box as a canvas to draw points on. The big problem is in managing the data, which makes up the plotted points. The code of the program must handle repainting the screen when another window covers the map and then uncovers it again. There is quite a bit of code in handling all the niceties, so to get things going quickly I chose Delphi and its TeeChart graphing component to do most of the work. Almost all versions of Delphi include TeeChart, but you will have to look around the components for it. In my Delphi 5 it is on the Additional tab and labelled as Chart with an icon which looks like some kind of pie chart. Its a very good component but not well documented.

TeeChart is designed to plot data so all I need to give it is the X-Y coordinates of the points collected from the GPS. It deals with all the Windows bits. The maximum and minimum extents of the graph axes control the magnification of the plot. I set them to show the largest plot I can get. [Figure

3 - raw position plot.] Its worth watching out for the East West axis which on a graph increases from left to right, but the GPS data is in degrees West, which increases from right to left. Again the graph plotter component allows selection of an inverted X-axis to correct this. The North South vertical axis is OK since it increases from bottom to top, which is the same as the degrees North from the GPS data. If you are in the Southern Hemisphere the plot axes may need to take account of latitude in degrees South, which needs an inverted Y-axis. There are several choices in the GPS receiver which can achieve the same inversion of axes, for example, degrees East. You must learn how to program them though.

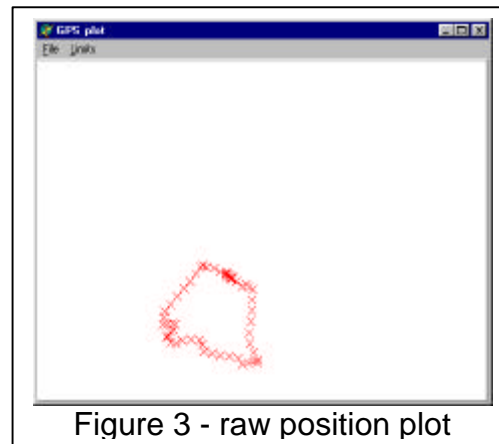


Figure 3 - raw position plot

The plotted data looks good, and it looks like the track of my route, but it would be much better if it were superimposed on top of a map. Then I could see just where the data leads. First I tried to plot the graph on paper. Getting the scaling to match a map correctly is almost impossible and unless I use clear film, the map cannot be seen very well. I guess the best result for me would be to use the PC program to plot the position onto a map on the screen. TeeChart allows a background picture, which acts, as a background to the plot, so what I need, is a map image to put in it.

It's possible to scan a map and use the image as a background for the plot. I tried this, but maps are quite large and getting a good scan with no creases and also well aligned is difficult. I could do with another solution. So looking through my old cover disks from the computer magazines I found a couple of programs, which show maps of Britain. These give quite good looking images when they are zoomed to maximum, so I decided to try and use one. Although some allow the GPS data to be fed directly into the program that is not much use to me, since I have collected all the data before hand. So my thoughts are back with the idea of a background image with my own program. The simplest thing I can do is screen capture the map as a BMP file and load that into the graph plot program. This is OK but only if I can calibrate the map against the graph plot extents. Several of the programs will not show the co-ordinates of the mouse on the map, but fortunately the AA Milemaster 99 version 2 does, and it is on the July 2000 cover disk of PC Plus magazine.

First zoom the map to cover the area of interest, then screen capture the image. I use Paint Shop Pro's capture but there are many other ways to do this including using the Print Screen button on the keyboard and then pasting the screen into the Paint program which is part of Windows. The whole screen image contains more than the map picture, so use the selection tool to draw a rectangle around the map image and use Copy to... on the Edit menu to save it to a file.

Now we have the image of the map, but we must get the co-ordinates of the corners of the image to do the calibration. I use the mouse to slowly approach the corners of the map in the Milemaster program. At the bottom of the screen a set of Latitude and Longitude co-ordinates show the mouse position. It is interesting that the projection of the map is not rectangular. For instance the X co-ordinate of the bottom left corner is not the same as the X co-ordinate of the top left corner. I guess the exact map projection depends on the mapping program, and we don't really know what it is. By collecting the co-ordinates of all four corners we are in a position to allow some corrections for these differences.

The other piece of work the program does is decode the file of GPS NMEA sentences. Only part of the \$GPGGA line is used. The program scans along the line looking for the Latitude and Longitude parts and plots them as a cross on the graph. If you use the program with your own data, make sure the data is output to the file as degrees minutes and decimal minutes. GPS receivers can be set up to give different data formats such as decimal degrees, which would still be read but produce unusual plotted points. You should also note that for the plot to match the map, the map image needs to just fill the graph window. Achieve this by pulling the edges of the program window to show just the complete map image.

Collecting the 'Live Data'

Now I have a system to collect position data from a GPS receiver. It uses the Rabbit module to save a position just every 30 seconds or so, and a PC program which shows the positions overlaid on a map. But I needed a route to check the whole system was working as expected.

Running the GPS module on the table inside my house limits the number of satellites in view and although it sometimes saw three or four satellites through the window, this seemed to only happen at 9:30pm for about 30 minutes! Not very satisfactory at all, so I decided to car mount the system so it could be taken for a drive.

Power becomes a problem. The mains power supply in the house isn't available in the car, only the 12V of the car battery. I should be able to access this 12v from the cigarette lighter, and having a collection of cigarette lighter plugs made me think it would be no problem. But alas none fitted my car. It was news to me that there are so many variations.

Not wanting to wait for a new plug to arrive, I examined my alternatives. Both the GPS and the Rabbit have voltage regulators, which work to give 5V. If I can provide, say at least 7V, the regulators can provide the stable supplies. So what power do I have available? Rechargeable batteries fitted into a battery holder seem a good choice. The holder carries eight AA cells, which gives about 9.6V, so it should work. A quick test needed just a few wires and proved effective. The next question is will the battery last long enough? Well the batteries are 600mA hour cells, and the GPS together with the Logger use

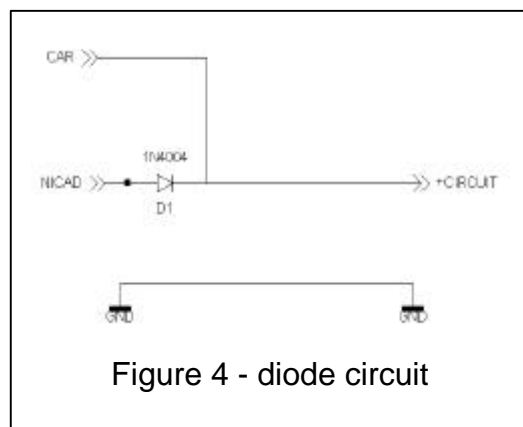
about 300mA between them, so I should get about 120 minutes from a fully charged battery pack.

That sounds good so I put the batteries in the charger to get them ready. In the mean time I checked the car to see where the antennae could be fitted and still allow the cable to enter the vehicle without leaving a door open. For the first time I found a reasonable use for the sunroof. You may guess that I don't like sunroofs. They usually give me a burnt head and make a lot of noise when moving. But back to the antennae, which sticks magnetically to the car roof just outside the sunroof. The cable drops down onto the passenger seat where the GPS module and logger can sit in a plastic crate. [See Photo 4 - Antenna on the car.] I haven't packaged the parts properly because in the end I want a hand portable or back portable system to use when walking, but I need to work out a suitable power supply for the system first.



All the elements are ready so I plugged in the battery power and went for a drive round a set of roads, taking about 20 minutes to complete. The LEDs flash regularly showing the logger is receiving and storing data. Off I went. Everything worked well until about 100m from the end of the run when the LEDs stopped flashing and one stayed on. What's gone wrong? When I measured the battery pack the voltage had dropped below the reset voltage of the Rabbit board. Not wanting to give up, I changed the batteries and tried again and once more the batteries failed just as I finished the journey! This wasn't going to work, and I needed a new idea.

Car cigarette plugs are quite expensive when you look at the prices in the catalogues, but I recalled seeing many cheap mobile phone car adapters in the local shops. Off I went to see if I could get an adapter for less than a new plug. My luck was in and I bought a suitable plug with phone adapter for £5. I pulled the plug apart and rewired it to provide 12V to the circuits. That got me going in the car. To get the circuits back into the house I needed to keep the power supplied. A diode, [See Figure 4 - diode circuit] and the battery pack attached through a plug in connector, allows me to run the equipment either from the car battery or the rechargeable battery pack. The battery pack powers the crate so I can carry it indoors to unload the data into Hyperterminal.



So with the system set up in the car I drove round the test route on roads just south of Macclesfield. The power held out this time and at the end I used the battery to carry the logger to the computer. The plot on the computer screen is shown in Figure 5 (route plot on map). The GPS track doesn't lie on the road. It is close though. This data was collected before selective availability was turned off. There is also a question of how accurate the map is when it is fully zoomed. I also guess my understanding of GPS datum's is not good, but at least the track shows on the map and is within 100m or so of the road.

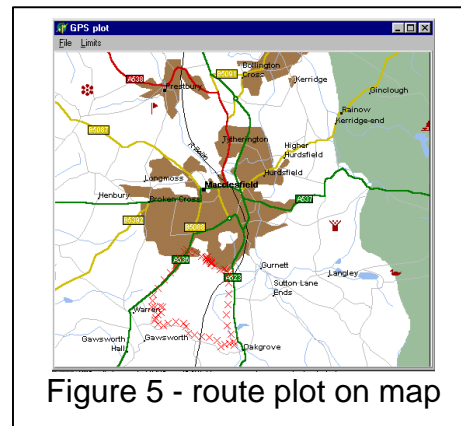


Figure 5 - route plot on map

Summing Up

Well there is the complete project from logging the data from the GPS to plotting it on the screen of the PC. When I first started the project a GPS module was the cheapest way to get going but now there are complete GPS units like the eTrek which include the antennae and its own battery power. You can sit the eTrek in the car window and it receives satellite signals very well. Also the plotting software needs improving to better match the GPS Latitude and Long data to the map co-ordinates. Give it a go for yourself, though.

Listing 1 - Rabbit C Program Listing

```
1 // GPS logger - © Copyright Vision Software 2000
2 //*****
3 #define BINBUFSIZE 255 // serial buffer size must be a power of 2 - 1
4 #define BOUTBUFSIZE 255 // output buffer
5 #define CINBUFSIZE 255 // serial buffer size must be a power of 2 - 1
6 #define COUTBUFSIZE 255 // output buffer
7 #define maxs 254
8 #define timeout 3000UL // will time out 3000 milliseconds after receiving
9 // a character unless cof_serBread completes
10
11 main()
12 {
13 int getOk, done;
14 int vswitch1,vswitch2,vswitch3,reci,outcount,wrapcount;
15 char s[maxs + 1]; // plus 1 for null terminator
16 char recs[300][70];
17 wrapcount=299; // maximum number of records
18 done = 0;
19 vswitch1=0; // initialize virtual switch as off
20 vswitch2=0;
21 vswitch3=0;
22 reci=0; // recorded lines counter
23 outcount=0; // output lines counter
24
25 WrPortl(SPCR, &SPCRShadow, 0x84); // setup parallel port A as output
26 WrPortl(PADR, &PADRShadow, 0xff); // turn off all LED's
27
28 serBopen(4800);
29 serCopen(4800);
30 while (!done) {
31 if( (PADRShadow & 1) == vswitch1)
32 BitWrPortl(PADR, &PADRShadow, !vswitch1, 0); // light the LED
depending on the switch
33 if( ((PADRShadow & 2) >> 1) == vswitch2)
34 BitWrPortl(PADR, &PADRShadow, !vswitch2 , 1); // light the LED
depending on the switch
35 if( ((PADRShadow & 4) >> 2) == vswitch3)
36 BitWrPortl(PADR, &PADRShadow, !vswitch3 , 2); // light the LED
depending on the switch
37 loophead();
38 costate { // read a line of the GPS data
39 wfd getOk = cof_serBgets(s, maxs, timeout); // yields until return or null
terminated string
40 if (getOk) {
41 vswitch2 = !vswitch2;
42 if ( (s[3]=='G') && (s[4]=='G') && (s[5]=='A') ) { //only $GPGGA lines
```

```

43  if ( ((s[11]=='0') && (s[12]=='0')) ||
44        ((s[11]=='1') && (s[12]=='5')) ||
45        ((s[11]=='3') && (s[12]=='0')) ||
46        ((s[11]=='4') && (s[12]=='5'))
47      ){ //00, 15, 30, or 45 seconds only
48    strcpy(recs[reci],s);
49    reci++;
50    if (reci>wrapcount) //wrap the index at the end of the array
51      reci=0;
52    vswitch1 = !vswitch1;
53    s[3]=0;          //prevent a false repeat
54  }
55 }
56 }
57 }
58
59 // print the saved lines
60 costate {
61   if ((PADDRShadow & 4) == 4){
62     serCputs("GPS Logger v1.0\n\r");
63     while(reci!=0) {
64 //   printf("%s\n",recs[outcount]); // output to stdio debugger window
65     wfd cof_serCputs(recs[outcount]); // then yields until the string is
written
66     wfd cof_serCputs(" \n\r");
67     yield;
68     outcount++;
69     if (outcount>=reci){ //end
70       reci=0;
71       outcount=0;
72     }
73   }
74   vswitch3 = !vswitch3; // reset the LED
75 }
76 } // end of costate
77
78 // also check button 1 and toggle vswitch on or off
79 costate {
80   if (BitRdPortI(PBDR, 2))
81     abort;          // if button not down skip out of costatement
82
83   waitFor(DelayMs(50)); // wait 50 ms
84
85   if(BitRdPortI(PBDR,2))
86     abort;          // if button not still down skip out
87
88   vswitch3 = !vswitch3; // toggle virtual switch since button was down 50
ms
89

```

```
90 // now wait for the button to be up for at least 200 ms before considering
another toggle
91 while (1) {
92   waitfor(BitRdPortI(PBDR, 2)); // wait for button to go up
93   waitfor(DelayMs(200)); // wait additional 200 milliseconds
94   if (BitRdPortI(PBDR,2))
95     break; // if button still up break out of while loop
96 }
97 } // end of costate
98
99 }
100
101 while (serBwrFree() != BOUTBUFSIZE) ; // allow transmission to
complete before closing
102 serBclose();
103 }
```

Where to get things

1. A version of Hyperterminal can be downloaded from <http://www.hilgraeve.com/> or you can buy an enhanced version from the same address. It may also be on your Windows CD-rom.
2. Software for this article is available from <http://www.visionsoftware.freemove.co.uk/>
3. Rabbit 2000 kit <http://www.rabbitsemiconductor.com/>
4. UK supplier of the Rabbit development kit.

2001 - Web: <http://www.2001elec.co.uk>

Impulse Corporation Limited, Unit 2 Littleton Business Park, Littleton Drive,
Huntingdon, Staffordshire WS12 4TR.

Web: <http://www.impulse-corp.co.uk/>

Email: sales@impulse-corp.co.uk

5. Garmin GPS information from www.garmin.com.