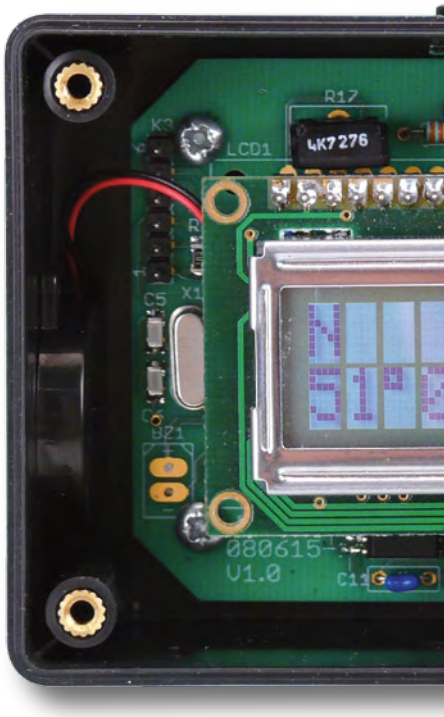
Speed Camera W

Drive wisely!

Gilles Le Maillot (France)



The little module described here lets you detect geographical points of interest (POIs) using the frames output from a GPS receiver module. These POIs might be restaurants, petrol stations, or —why not? — the positions of fixed speed cameras!

Having found it hard to find fully-developed, ambitious projects every year, the circuit published online by Christophe Le Lann [1] seemed to me a good starting point. So we adapted this Electronics Design project for the course taught at our College (*ENSIETA* [2]). We've used a PIC microcontroller and added several new options like a bigger memory, the possibility to update the memory via USB, speed display, etc. In addition, we produced the program under Flowcode using E-blocks [3].

Flowcode is a high-performance graphical development environment for microcontrollers (PIC and AVR) that makes it possible to swiftly create quite complex electronics systems, and above all, to simulate them. The program description is in the form of a standardized (ISO5807) flowchart using macros that make it easier to control complex peripherals, like 7-segment displays, motor controllers, LCD displays, Bluetooth, TCP/IP, etc. Elektor has already published numerous articles about this product. For myself, I was quite surprised by how powerful, user-friendly, and easy to learn this software is. Of course, it's not a magic tool, it does have its limitations — for example, the PIC interrupt library is not comprehensive enough, and it only recognizes whole number values to a maximum of 16 bits — but these are fairly easy to work around.

It's up to you...

The project described in this article may be used as a warning device for fixed speed cameras, which is perfectly legal in France at the time of publication of this article. However, this does not mean to say that the use of this project is legal in other countries, nor that it is going to remain legal for use in France.

In addition, the Flowcode simulation mode allowed us to test the code for this project (except for the serial connection interrupt part) before implementing it.

Thanks to Flowcode, we've been able to produce a quite substantial project in a limited time. The use of a tool like Flowcode (in an educational context) was a first for us — most students appreciated it, and some of them actually managed to see the project right through to the end!

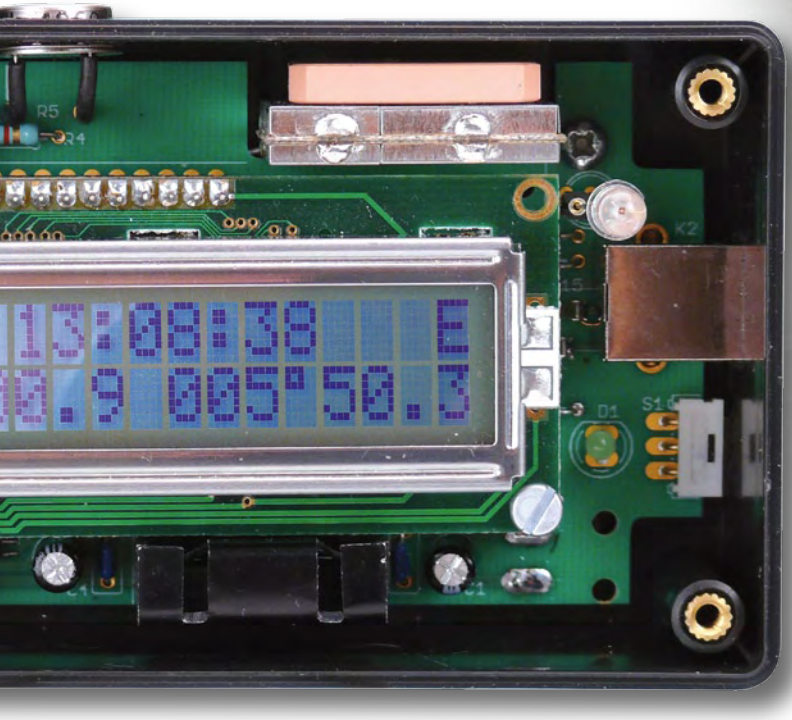
Block diagram

As the block diagram (**Figure 1**) shows, the system is fairly simple: a GPS receiver provides the system's geographical position once a second. This position is then compared to the POI locations stored in a database. If there is a POI within around 500 m of the current position, a visual and audible warning is triggered.

The heart of the system is a 16F876A-I/SO microcontroller from Microchip, which receives the vehicle's positions from the GPS, looks them up in the database, and drives the man/machine interface (MMI).

This MMI consists of an LCD display, a sounder and a bi-colour LED. If there is no POI in the vicinity, the LCD just displays the position and the time or speed. The bi-colour LED flashes green every time a GPS frame is received. In the event of a POI nearby, the sounder

arning Device



sounds, the bi-colour LED lights up red and steady, and the LCD displays a warning message. The MMI has one little unexpected extra: automatic backlighting that adjusts itself to the ambient light level.

An I²C memory is used to store the geographical position of the POIs. A USB interface is available for loading the POIs into the memory from a computer.

The GPS receiver, which sends its data via a serial link, shares the microcontroller's serial link with the USB interface. A multiplexer allows the serial data source to be selected using a simple switch.

E-blocks

The first platform was achieved using these E-blocks: an **EB006** for the development platform (this is directly usable under Flowcode for programming the PIC and supports many types of PIC) and an **EB005** for the LCD. For the rest of the project's components, we've created our own E-block, connected to the PIC PORT C. In this DIY E-block (**Figure 2**) we find the I²C memory, the FT232BL USB/RS-232 interface, the bi-colour LED, the sounder, and a MAX232 to allow us to dispense with the USB port in the first instance and be able to simulate the GPS frames on a PC. **Figure 3** shows the prototype in all its splendour.

The program

The program, developed under Flowcode, comprises two distinct sections. The first and most important section handles the dialogue with the GPS module, compares the data from the GPS with the locations stored in the

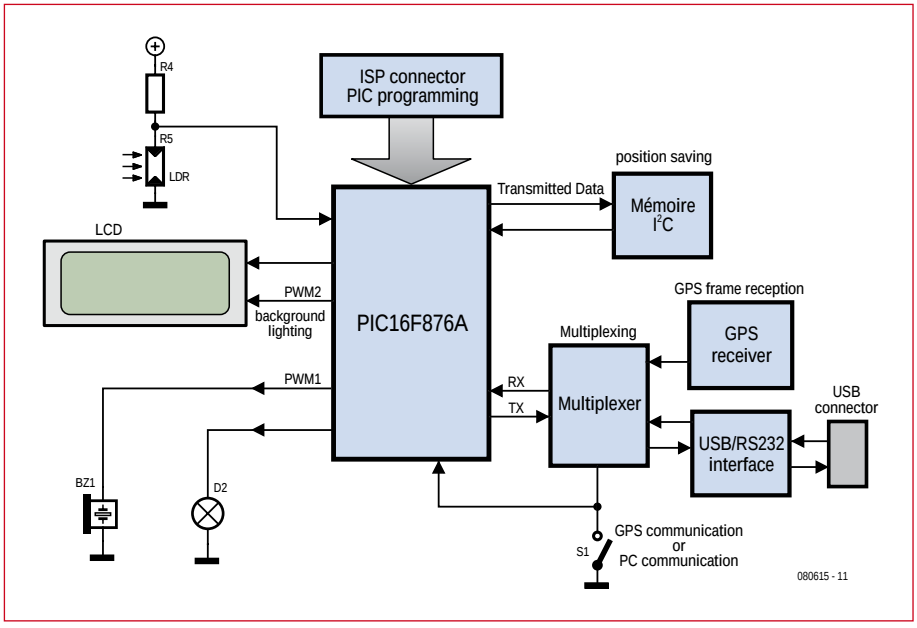


Figure 1. Block diagram of POI warning device.

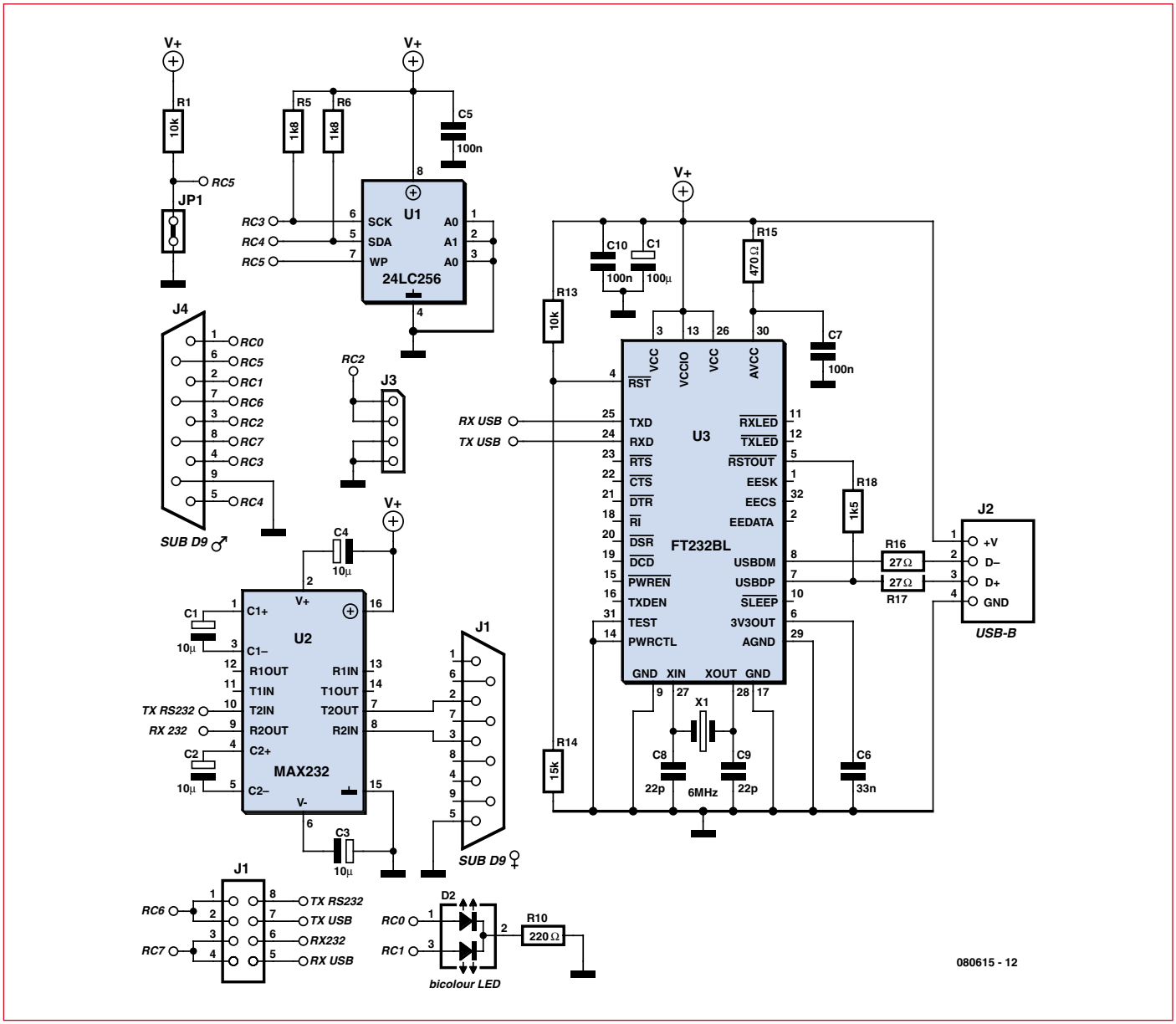


Figure 2. Circuit diagram of the home-made E-block. The serial data input is selected manually by the position of the jumpers on J1.

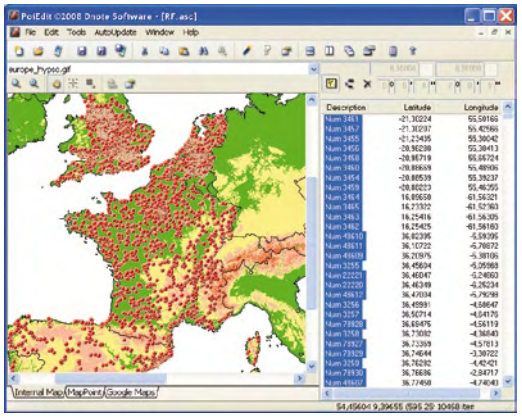
POI files

There are lots of different types of POI files, but the ones we're using contain nothing more than a list of geographical positions in ASCII, hence their .ASC extension. On one line of this list we find three comma-separated fields: longitude, latitude, and a name, often a number:

2.68111, 44.43686, "Num 40235"

The longitude and latitude are in decimal degrees.

The simplest way to obtain a POI file that can be used by our project is to pay a visit to the PoiEdit website [5]. PoiEdit is a shareware application that lets you display and edit the contents of a POI file. This website also has lots of links to other sites where you can get hold of POI files (for free). To display a POI file, all you have to do is load it into PoiEdit and pick 'Select All' in the 'Edit' menu. Don't forget to load, and if necessary calibrate, a map. Some maps are also available on the PoiEdit website.



To sort a POI file by longitude (if you're using the 080615-11_1 program), all you have to do is click on the Longitude bar and save the file in .ASC format.

The POI file thus created or downloaded can be directly read by the transfert.exe update program, as described elsewhere in this article.

I²C memory (**Figure 4**), looks after displaying the data, and drives the sounder and bi-colour LED. The second section is used for updating the I²C memory with the help of a computer. A switch determines which section of the program is run.

Primary loop

Out of the NMEA0183 frames provided by the GPS, we're going to use the RMC frame [4]. This frame contains all the information we need: latitude, longitude, time, date, and speed. After decoding an RMC frame, we then need to read the I²C memory. If we find a location corresponding to our current position – minus a certain margin, of course, otherwise it's too late! – that means we are near a POI. In this event, we leave our read loop and set off the alarms, i.e. the sounder sounds, the bi-colour LED light up red, and a message is displayed on the LCD warning of a POI close by.

Next time a GPS frame is received, we start again and decode, read the I²C memory, compare, etc.

Updating the database

The second section of the program is used for updating the database via a serial link. The transfer is initiated by the PC which sends the character 13h (19 in decimal) to the PIC, and the transfer starts once the PC receives the same character back. The PC then sends the file to the PIC, which acknowledges each character received by sending the character 13h. When 128 characters have been received, the PIC writes them into the I²C memory. To do this, we've used the I²C routine available in Flowcode, which makes it very easy to use the I²C bus. The transfer ends with a special character FFh, which is the signal for the PIC to display on the LCD the number of points stored in memory. This number is also stored in the PIC EEPROM, as we need it to be able to get out of our comparison loop correctly in the other section of the program.

For updating to be as fast as possible, it is done at 115,200 baud. But the component routine is already configured to 4,800 baud for dialogue with the GPS. We have got round this problem by inserting a little bit of assembler code into our program.

Another complication concerned the interrupt used to detect the reception of a character. The Flowcode library does not include this interrupt, so we had to create a user source for it.

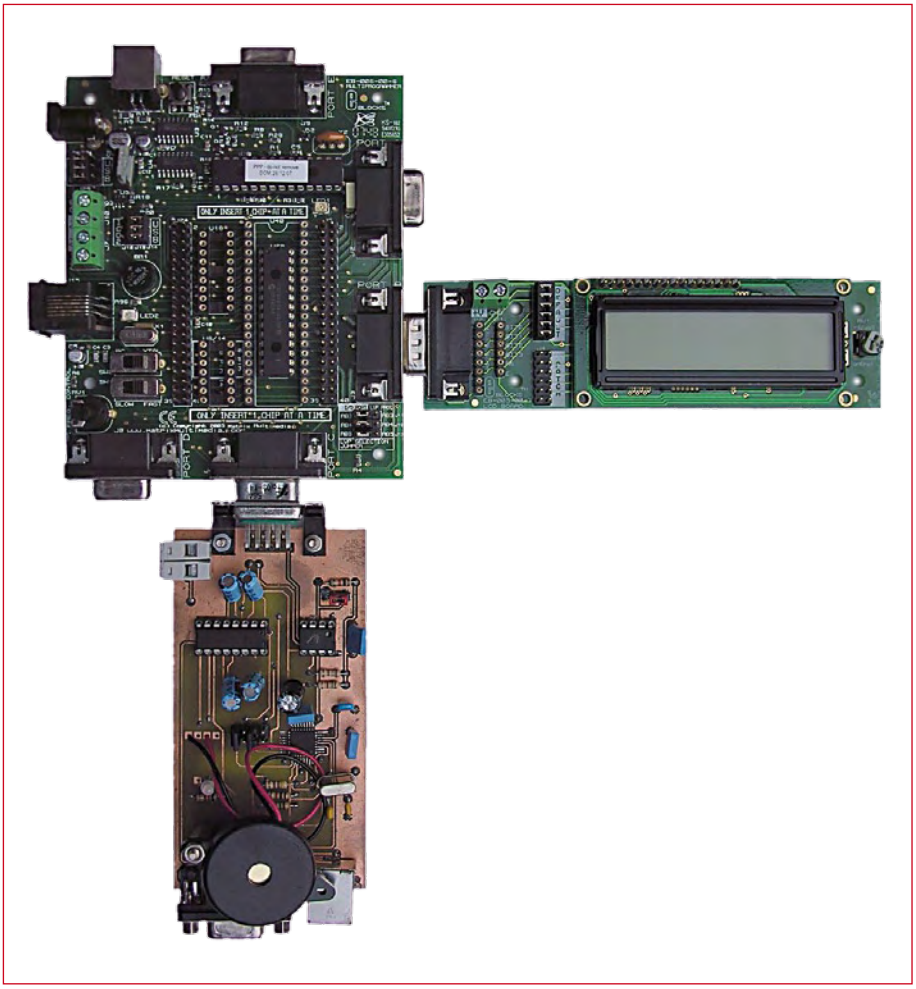


Figure 3. The speed camera warning device built using E-blocks. Our own E-block is the one with the sounder.

Automatic backlight

One option that deserves to be mentioned here is the automatic adjustment of the display backlight depending on the ambient light level. This was easily achieved using the PIC's ADC, which measures the voltage at the terminals of the light dependent resistor (LDR), and a PWM (pulse width modulation) output to control the backlighting via a transistor. The ADC and PWM are component routines included within Flowcode.

Simulation

Virtually the whole of the program can be simulated in the Flowcode environment, except for the reception of the characters during transfer of the file containing the POIs, where we have used some assembler code. Each component of the project can be simulated: the LCD, the PWM output, reading the I²C memory, GPS frame reception, and even the ADC for use with the LDR.

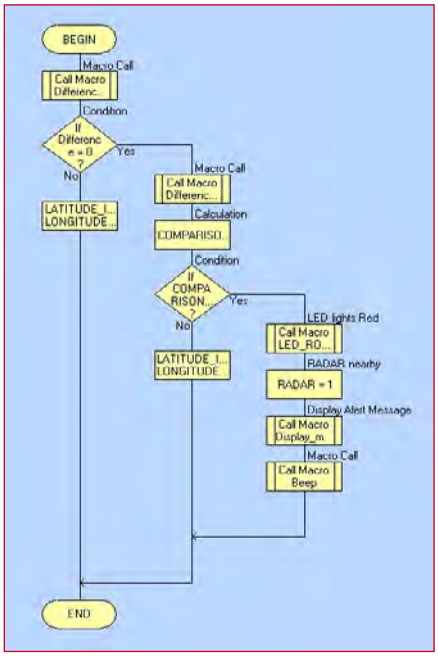


Figure 4. The full program is much too long to be shown complete. So we'll just give the most interesting part: the detection algorithm.

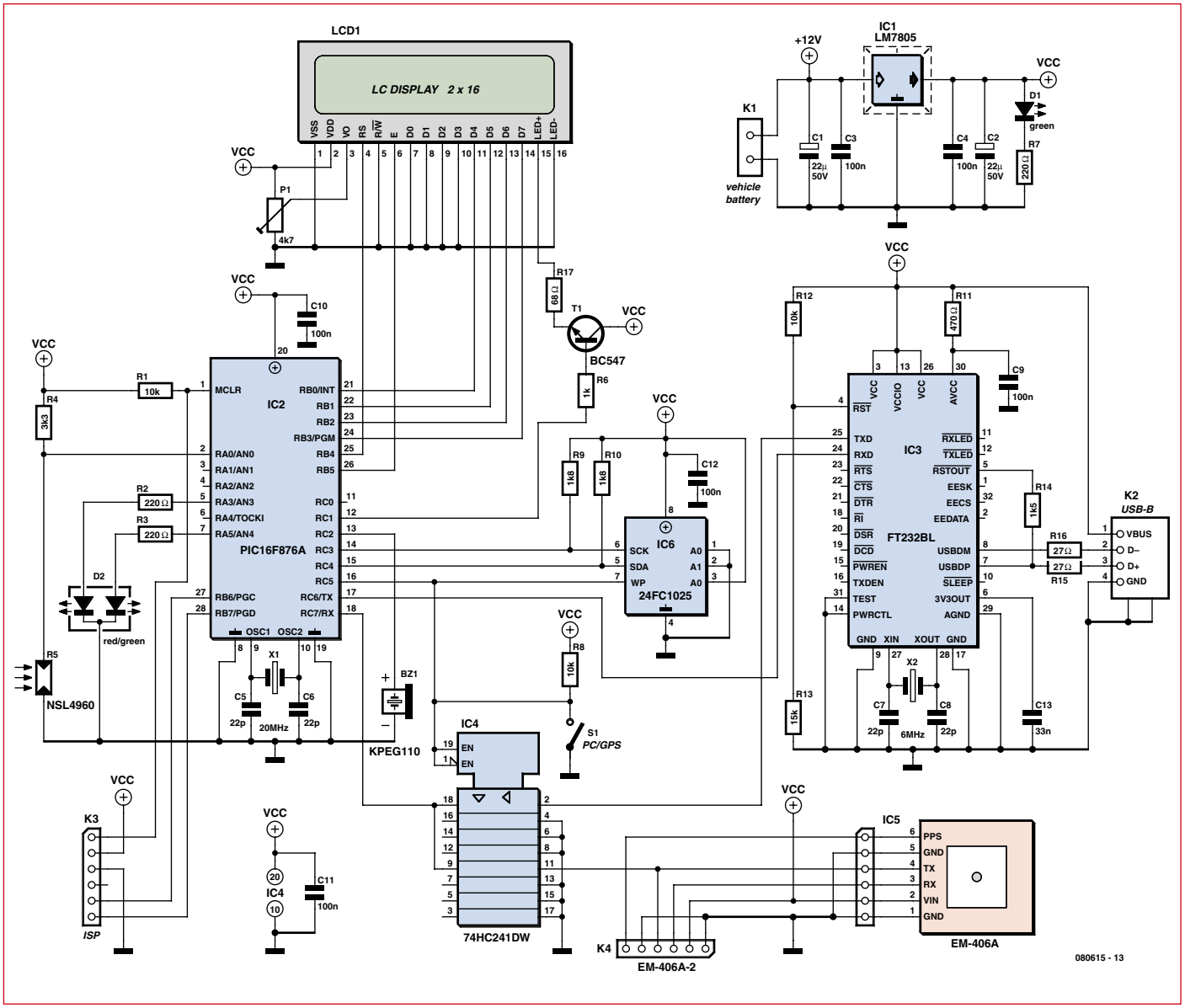


Figure 5. The full circuit diagram of the speed camera warning device.

To simulate decoding a GPS frame, we need to input a GPS frame to the RS-232 component module. We can then see the reading of the memory in the I²C routine, byte by byte.

The values of the variables can be displayed (or changed), and simulation can be performed in step-through mode.

Circuit

Once our E-block prototype was operational, we redesigned the circuit without the actual E-blocks (**Figure 5**) – the EB006 E-block has been replaced by a 16F876A PIC (IC2) running at 20 MHz and the EB005 E-block by a standard alphanumeric LCD with backlight (LCD1) – the contrast can be adjusted with potentiometer R17. We have eliminated the components that are no longer needed, like the MAX232, replaced the manual multiplexer by a 74HC241, and added photoresistor R5.

The PIC connects to the I²C EEPROM via its special I²C bus inputs SCL and SDA. The GPS receiver and the USB/RS-232 interface (IC3) are connected to the PIC USART by way of the multiplexer IC4. In normal mode, the multiplexer connects the PIC RX input to the GPS TX output to receive the GPS frames. In I²C memory update mode, the RX input is connected by the multiplexer to the TX output of the USB/RS-232 converter. The PIC TX output is directly connected to the RX input of the convertor IC3. Switch S1 lets you choose between normal and update mode, and at the same time controls the EEPROM write protection at the same time.

The display is connected to PORT B of the PIC in 4-bit mode. Input AN0 of the ADC is connected to a potential divider made up of R4 and the photoresistor R5, which enables us to vary the display backlighting depending on the ambient light level. The backlight is adjusted by means of the signal on one of the PIC's two PWM outputs. The other output is used to drive the sounder. The bi-colour LED D2 uses another two outputs of PORT A, RA3 and RA5.

EEPROM chip IC6 contains the position of the POIs, each listed by latitude and longitude to 6 bits. For our

project, we've chosen the 24FC1025 from Microchip, a 1,024 kbit memory that allows us to store the position of 21,845 POIs.
The most expensive part in the whole project is the EM406-A GPS receiver module with built-in antenna from GlobalSat, already familiar to regular Elektor readers [4]. It interfaces directly to a microcontroller via its 'almost' TTL-level serial port.
The USB/RS-232 interface is taken care of by an FT232BL IC from FTDI (IC3). This forms the interface between the PIC and the PC and requires a driver to be installed on the PC in order to be used as a virtual COM port.
And lastly, the project is powered via a 7805 regulator.

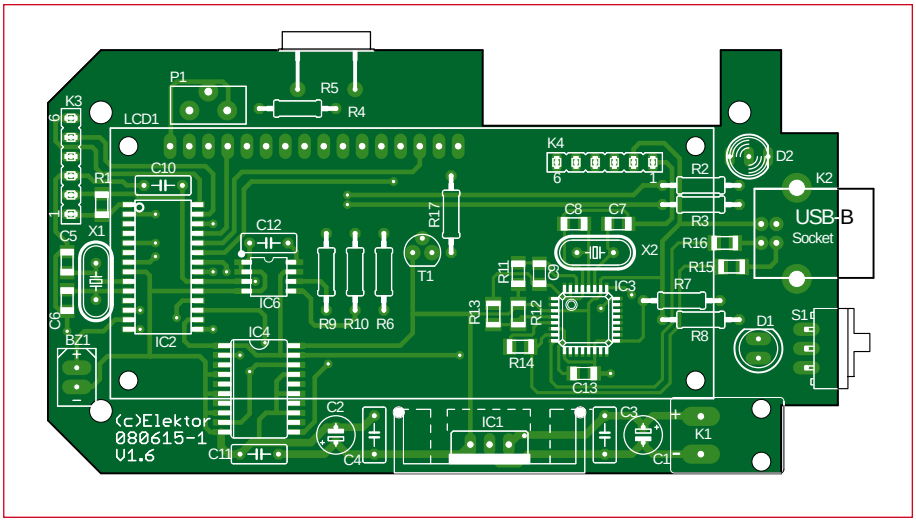


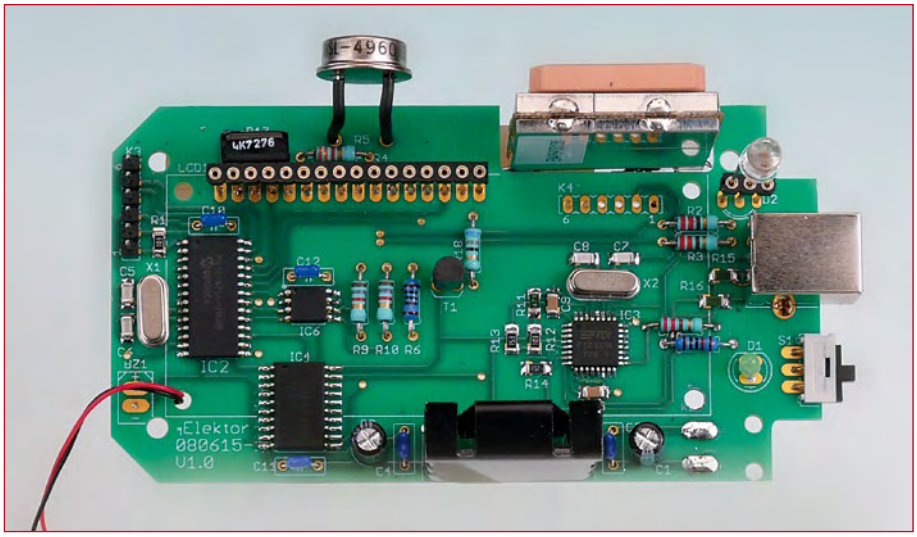
Figure 6. Board component layout.

Construction

It'll take you just a few hours to build this project. Refer to **Figure 6** for the board component layout. Note the use of a 'wire-wrap' socket to bring the display up to the height of the housing, and the same for the bi-colour LED.
The first step is to solder the SMD components. The FT232BL IC is the trickiest, but with a very fine tip and a bit of patience, it can be done (you can use solder flux to help). The other SMD components ought not to present any real problem. Next, solder the discrete resistors, non-polarised capacitors, and then the electrolytic ones (observing the correct polarity carefully). After soldering all these components, check that the supply voltage is reaching the ICs on the appointed pins.
Two .HEX files are available for programming the PIC (see components list). The executable called 080615-11_2 can be used with downloaded POI files directly. The 080615-11_1 file requires a POI file sorted by increasing longitude, which speeds up the POI detection algorithm.
With the circuit powered and the PIC programmed, the green LED D1 lights and the display shows a start-up message (depending on the position of S1). If the display appears blank, adjust the contrast using R17.

First steps

The first time the circuit is powered up, the EEPROM has to be programmed with a POI database. Close S1 and connect the circuit to your computer's USB port. **Never connect the USB cable and the cigarette lighter plug at the same time!** Now's the



COMPONENTS LIST

Resistors
(0.25W 5%)
R1,R8,R12 = 10kΩ
R2,R3,R7 = 220Ω
R4 = 3kΩ3
R5 = NSL4960 (LDR)
R6 = 1kΩ
R9, R10 = 1kΩ8
R11 = 470Ω
R13 = 15kΩ
R14 = 1kΩ5
R15, R16 = 27Ω
R17 = 4kΩ7preset, vertical mounting
R18 = 68Ω

Capacitors
C5,C6,C7,C8 = 22pF
C1,C2 = 22µF 50V
C3,C4,C8-C12 = 100nF
C13 = 33nF

Semiconductors
D1 = LED, 3mm, green
D2 = LED, 3mm, bi-colour
T1 = BC547
IC1 = 7805
IC2 = PIC16F876A-I/SO, programmed, Elektor SHOP # **080615-41**
IC3 = FT232BL
IC4 = 74HC241DW
IC5 = EM406A GPS receiver (Sparkfun, Lextronic)
IC6 = 24FC1025

Miscellaneous
X1 = 20MHz quartz crystal
X2 = 6MHz quartz crystal
LCD1 = LCD, general purpose, 2 lines,16 characters, with backlight
BZ1 = KPEG110 buzzer (Farnell)
K1 = cigarette lighter plug
K2 = USB-B socket
K3,K4 = 6-way SIL socket strip
S1 = switch, 1 pole, 2 positions
Enclosure, Hammond type 1591XXCBK (Farnell)
PCB, Elektor SHOP # **080615-1**
Project software: free download from www.elektor.com/080615

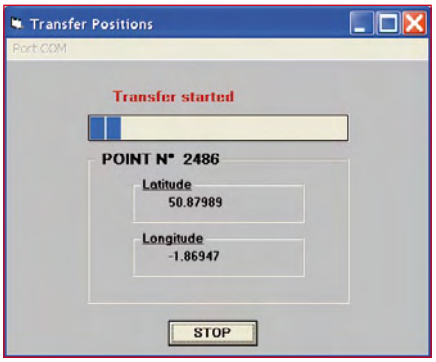
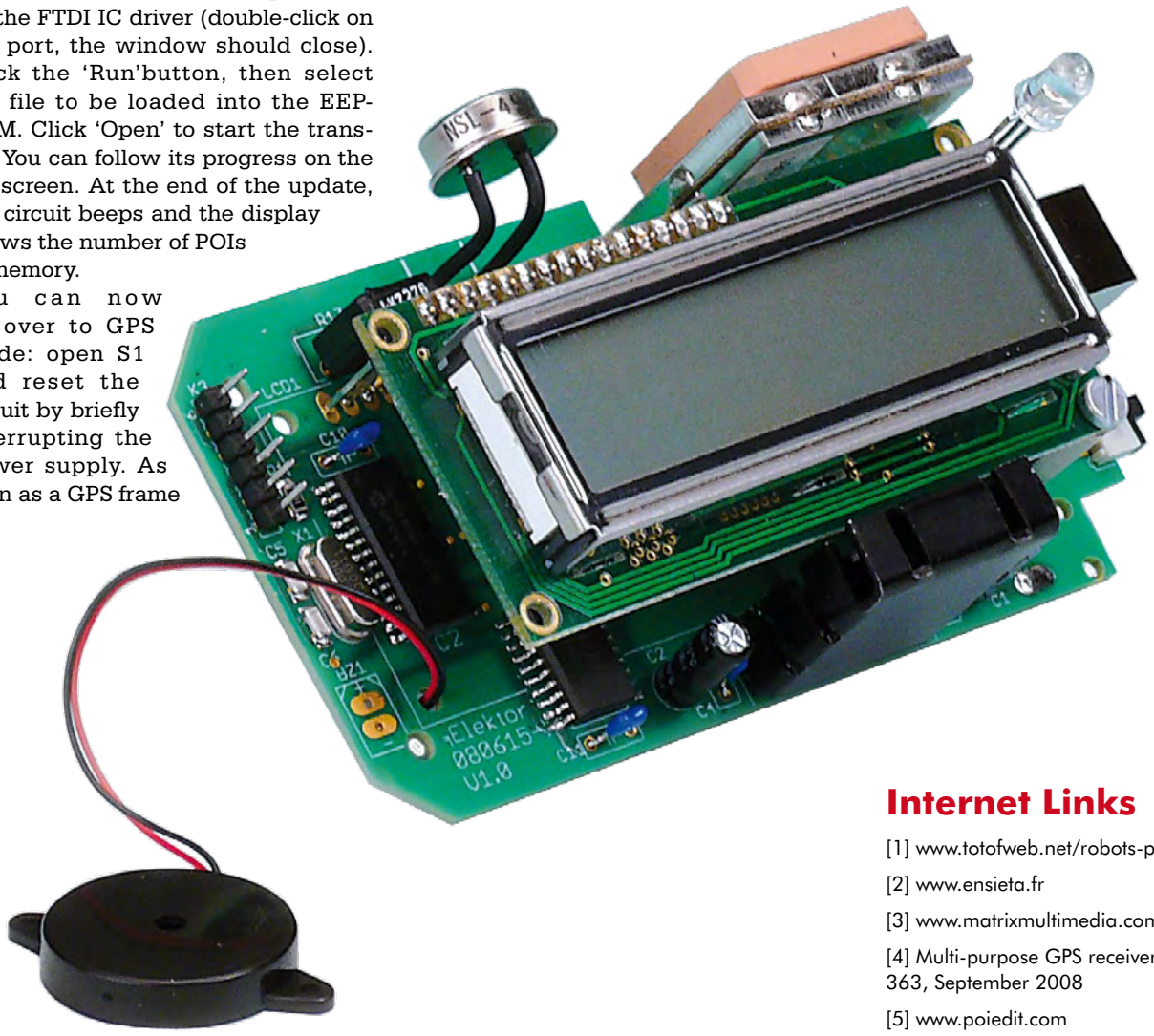


Figure 7. The database is updated using the Transfer.exe program.

moment to install the FTDI drivers, if needed. Then, in the Windows 'Device manager', set the speed of the virtual COM port to 115,200 bits/s and, under 'advanced' settings, change the latency to 1 ms, then click OK.
Now run the Transfer.exe program (**Figure 7**), available on the web page for this project. When the program starts, you need to select the serial port used by the FTDI IC driver (double-click on the port, the window should close). Click the 'Run'button, then select the file to be loaded into the EEPROM. Click 'Open' to start the transfer. You can follow its progress on the PC screen. At the end of the update, the circuit beeps and the display shows the number of POIs in memory.
You can now go over to GPS mode: open S1 and reset the circuit by briefly interrupting the power supply. As soon as a GPS frame is received, the bi-colour LED will flash green and the display will show the position, alternating with the time and speed. It may take a while to receive the first frame from the GPS; the EM406 module has a red LED that flashes each time the GPS receives a frame.
And there you have your POI warning device finished — safe journey, and above all, remember to obey the speed limits!

(080615-I)



About the author

Gilles Le Maillot is an electronics design engineer in the ***DTN*** department at the ***ENSIETA***. He is passionate about electronics and computers.

The ***ENSIETA*** (Higher National Engineering College) is a multi-disciplinary engineering college based in Brest, France. It trains mechanical, electrical, and IT engineers for all sectors of industry (automotive, naval, aeronautics, and so on). The ***ENSIETA*** runs numerous research and development programs within its four laboratories (***DTN, E3I2, MSN, SHI***).

Acknowledgements

Dominique Kerjean: design engineer at ENSIETA.

Pierre Cambon: research lecturer at ENSIETA.

André Mininno: design engineer with Multipower.

Internet Links

[1] www.totofweb.net/robots-projet-53.html

[2] www.ensieta.fr

[3] www.matrixmultimedia.com

[4] Multi-purpose GPS receiver, p.34, Elektor 363, September 2008

[5] www.poiedit.com