// SEMESTER PROJECT — END-OF-YEAR FINAL

S.A.D — Speed Awareness Device

A GPS-based, microcontroller-driven device that displays the current road's speed limit.

Caleb Cohoon · 2006 · Original college coursework, reproduced from the archived manuscript

Introduction & Project Goal

For the semester project, I designed a system which provides the current speed limit of the road being traveled by the use of a GPS module. I got this idea from traveling one day in a car and wondering what the speed limit was on the road. To my dismay, there was no speed limit posted. I thought it would be nice to always know the speed limit by some sort of digital device that could sit on the dashboard and display the legal limit at all times. This was when the idea was born.

During the initial planning phase of the idea, the preliminary design was going to implement custom wireless “hot spot” modules placed around certain roads. Each device would emit data (speed limit) specific to that road. Obviously, this design would be very costly and unintuitive. Updates would be very difficult because each device placed around a certain proximity would require manual service and the devices could easily be stolen. These factors guided me to eventually use a GPS device for the use of tracking individual streets instead of relying on stationary “hot spot” modules. This allows for easy updating and is much more cost effective because only one device has to be constructed. If more street information is needed, just a simple update to the device via a serial connector is needed through custom built software running on a PC. This entire approach to building the GPS device allowed it to be finished inexpensively and remain simple to control.

There are two modules to the system: the base unit which is placed on the car floor and the external LCD screen that is attached to the dashboard. The device's use is to keep the driver aware of the current speed limit at all times and display the street name information. The base module of the system is responsible for calculating the vehicle’s current position and finding out which road is being traveled to display the speed limit. The base unit also contains status lights, a serial DB9 connector for map information updates and input buttons.

The hardware of the device contains four different Atmel 8051 microcontroller models, a TTL voltage converter and an I²C 32KByte EEPROM. The microcontrollers are 8-bit RISC based processors. The four models I used were the AT89C2051, AT89C55, AT89C52, and AT89C51. They are all very similar in their specifications but differ in the on-board code capacity. The reason I chose to use four different models of microcontrollers was that I was trying to select the appropriate chip that would closely fit the size of each software module.

Memory Format

One of the ATMEL microcontrollers is constantly downloading positioning data from the GPS module and parsing the incoming NMEA 0183 format to locate the longitude and latitude of the user's current position. This information is then sent to the second controller which is responsible for comparing the current longitude and latitude to map information stored on the EEPROM. The street map information is stored in 36-byte chunks. Each chunk contains the latitude and longitude information, speed limit and street description. The specific memory format is shown below. If the vehicle's current position is found to be in the area of a street that has been mapped, it then sends the street name and speed limit to the external LCD screen. The third controller allows for street map updates to the internal EEPROM.

Latitude Point 1	4 bytes
Latitude Point 2	4 bytes
Longitude Point 1	4 bytes
Longitude Point 2	4 bytes
Speed Limit	4 bytes
Street Description	16 bytes

The way the system knows what street you are on is by the use of boundary boxes. Two points from a street — top-left corner and bottom-right corner — create a virtual box, and these two points, along with the street name and speed limit, are stored into memory. When the system finds your position, it checks all of the boundary boxes in memory, and if your location is in one of the virtual boxes, then the display will show that street's information.

The external LCD module is responsible for displaying the road information. It connects to the base unit via an RJ-11 cable. A microcontroller is also running on this unit to download the incoming data from the main unit. A custom format is used for data transmission between the devices and the data runs at a rate of 4800 baud. The data is transmitted serially through one of the four data lines in the phone cable and the other lines are used for powering up the device so no external power supply is needed in order for it to work. Originally I was planning on using just a parallel cable because the LCD has 8 data bit lines that are used for sending bytes to the LCD processor, but later decided that this would be very bulky and hard to maintain. That's why there is a microcontroller running on the external LCD device — the main base sends the LCD code serially through the small phone cable and the microcontroller converts the data to parallel for the LCD screen. A little harder to implement but it avoided the weight of a heavy cable.

On top of the device are three input switches. Each of these input buttons controls a certain function. The first input allows the device to show NMEA positioning information on the LCD screen instead of the normal display of speed limit and street name. This is useful if you wish to record latitude and longitude coordinates from the surroundings. The second input button switches the device to EEPROM mode. This mode is used when the EEPROM will be getting updated map information via the custom software running on the PC. Before any updating, this mode must be selected in order to avoid corrupted data transfer. The reason is the EEPROM is simultaneously connected to the microcontroller that checks map information and the microcontroller that updates information to the EEPROM. If both devices try to use the EEPROM, errors could occur. The final input is simply used to toggle the backlight on the external LCD screen in case the device is used at night or in dim lighting conditions.

Some of the anticipated problems that I encountered were electrical noise interference, writing compact software and learning how to communicate with the EEPROM. The reason I anticipated electrical noise was due to the fact I would be building this project on breadboards and the GPS module is very sensitive to gaining a satellite fix. It needs to have a very clean source of power. I had to design it in such a way that the wiring and other components would not interfere much with the module. Another problem was writing compact software for the microcontrollers. Since the controllers are very limited in on-board code space and their memories are very small, the code is small and thoughtfully planned. Splitting up all the code on subsequent chips aided in this process

and relieved some of the pressure of having to program in an extremely optimized fashion.

The major challenge in this project was the use of the I²C 32KByte EEPROM chip. I had never worked with the I²C protocol and trying to figure out any protocol can be tedious. I began by looking up articles pertaining to the format, but most of the code examples were in pure assembly. This made it difficult because I was programming in C. Even when I figured out the protocol, I had to make sure I implemented the correct timing because communication formats have some type of cycle times. Luckily I found some source code that described how to implement, in software, communication with general I²C devices. The code took a while to implement into my project and some modification was necessary for it to work appropriately for this project's specification.

Project Update & Conclusion

As of this project update, the unit is finished and working pretty well. It was tricky getting the whole device into suitable casing because of the breadboards. There were other unanticipated problems that arose but I figured out a way around them. Communication with the I²C EEPROM turned out not to be as difficult as planned and now I have good source code for future projects. The GPS module connects relatively well even though it is encased with a rat's nest of wires. Maybe in the future I can create a modified version that uses an external GPS device to gain faster and stronger satellite fixes. The external LCD screen works very well and the custom format for data transmission seems to do a good job. I got very close to running out of memory on one of the microcontrollers because parsing the incoming NMEA data from the module requires a lot of variables and memory function calls. When I was first designing this project I did not realize that I would have to write software for the desktop computer to be able to update the EEPROM with new map information via the serial port on the device. It turned out not to be that hard and it is very easy to use. There were also some added improvements to the device. I was able to make it portable so one could walk around with it instead of relying on a car for a power supply, and the communication software for updating the device was totally revamped. The new software interface is much easier to use and operate.

I'm very happy with the results of this project. I had this idea long ago and it seemed like it would be a very hard challenge. As it turned out, there were some challenging points in the project but I eventually solved the problems. I learned a great deal through the process of building this device, such as learning how to communicate with an EEPROM

via the I²C protocol, learning how to interface to a GPS module and understand the NMEA data, parsing serial data and learning how to program with limited space available. There are several areas of improvement that could be incorporated into this project since I now have the benefit of having a big part of the learning curve behind me. These update ideas will be saved for a later time, but I'm very happy with the outcome.

Works Cited

- "General: Byte-Wise Access to Float Values." *Keil*. 5 July 2006. <http://www.keil.com/support/docs/2768.htm>
- Schultz, Thomas. *C and the 8051*. 3rd ed. Otsego: PageFree, 2004.
- http://futurlec.com/LED/LCD16x2.shtml
- www.mhd.miun.se/~stok/ghetto-gps/
- http://www.stubaby.f2s.com/microcon/mc_proj1.htm
- http://www.usglobalsat.com/downloads/engineboards/EM406.pdf