by Robin Brophy (USA)

FEATURE ARTICLE

eDASH

A Virtual Dashboard Using OBD-II

eDASH is a virtual car dashboard that utilizes the OBD-II port and enables you to scan many different engine functions. The dashboard can be easily constructed for OBD-II-compatible vehicles using readily available modules.

This article describes the construction of an electronic car dashboard utilizing the OBD-II port. The project makes use of readily available modules that allow for easy assembly by the DIYer. It enables you to monitor various engine parameters and view or reset any diagnostic trouble codes. I call my project eDASH (see Photo 1).

The project started nearly 20 years ago when the speedometer on my car stopped working. A self-proclaimed gearhead with a degree in electronic engineering, I get a little giddy when I can combine those two fields. So, instead of trying to fix the broken speedometer, I figured I would build a digital one. I used a Motorola MC68705 microcontroller, a three-digit seven-segment LED display, a Hall effect sensor, and two magnets strapped onto the drive shaft to create the digital speedometer. After that, I had visions of a complete digital dashboard. Remember the 1980s television show *Knight Rider* and the car, KITT? That is the kind of car I wanted. However, interfacing to all the different signals I wanted would involve installing custom transducers, a lot of analog circuitry, and a complete rewiring of the car. Needless to say, that vision didn't go very far.

Fast forward a few years, when I discovered that General Motors had an assembly line diagnostic link (ALDL) connector on most of their vehicles starting in the mid-1980s. There is certain data available via this connector.
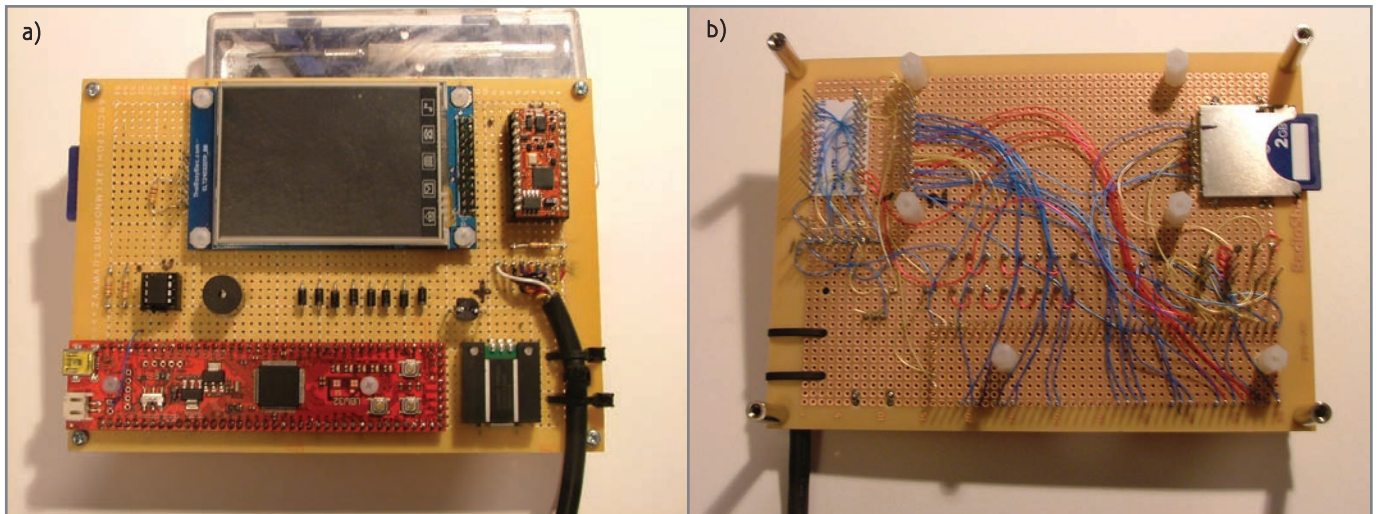
Photo 1—The top (**a**) and bottom (**b**) of the eDASH virtual dashboard

I found a software program called WinALDL and a schematic for a cable. I happened to have an '86 Camaro sitting in the garage, so I made a cable, hooked it up, and was able to scan some data. But, it was rather inconvenient having a laptop sliding around in the car. I thought about building an interface with a dedicated display, but after more research, I realized that just about every different car model had a slightly different interface. This didn't allow for a generic interface, so this idea didn't go very far, either.

Fast forward a few more years, when I discovered Elm Electronics's ELM327 OBD-to-RS-232 interpreter. OBD-II is a government-mandated standard that U.S. cars were required to comply with starting in 1996. The OBD-II standard basically states that car manufacturers must make certain operating parameters available for monitoring via a standard connector and protocol. These parameters were mainly related to emissions equipment, but basic functions such as speed, RPM, and temperature were also included. Even though there was a standard on the top layer, the lower layer was open, so each car manufacturer came up with its own interface scheme. The ELM327 interpreter took care of translating all the different schemes into a common language and essentially made every U.S. car manufactured after 1996 look like a serial port. There were commercially available interface units that utilized the ELM327 interpreter, but they all used software that ran on laptops. I learned my lesson earlier and swore I would not use a laptop. I wanted a standalone scanner with a display and input device. No computer!

I soon obtained an ELM327 interpreter and built a prototype OBD-II scanner using a Parallax BASIC Stamp microcontroller, a 7" composite video monitor from a portable DVD player, a serial-to-NTSC video module, and a TV infrared remote. This proved that the concept of monitoring data on the OBD-II port was feasible, but I soon ran out of memory on the BASIC Stamp. The next version involved a Microchip Technology PIC18F2620 microcontroller, a Maxim Integrated Products MAX7456 single-channel monochrome on-screen display (OSD) generator, and, again, a composite monitor with a TV remote. The problems with this version were the TV remote was always getting lost and it was not easy to find a place to mount the video monitor. The next version used a bigger PIC, a 128 × 64 monochrome graphic LCD, and a small five-button pendant wired with a phone cord. On this unit, the display was limited and the pendant kept
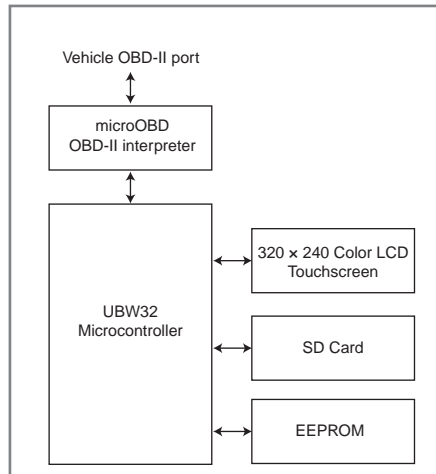

Figure 1—A block diagram of the project

getting tangled up—you know how tangled phone cords can be, almost as bad as lost TV remotes.

The processors used in the later versions were surface-mount devices, which made it somewhat difficult to prototype. The OBD-II vehicle interface itself requires approximately 40 discrete components in order to cover all the different manufacturers' schemes. I ended up having to etch circuit boards for three different versions of the project in order to test them; there was no easy way to breadboard them.

The ultimate goal was to have something about the size of a GPS navigator with a color touchscreen. It had to be easy to build, which meant no surface-mount components, if possible. With the proliferation of MP3 players and cell phones, a supply of color LCD touch-screens is now available to amateurs and professionals alike. The last piece of the puzzle was a ScanTool.net microOBD 200, which is a complete ELM327-compatible OBD-II interpreter and interface in a standard 24-pin DIP footprint. The microOBD drastically reduces the number of discrete components required and therefore simplifies construction.

DISPLAY & DEVELOPMENT BOARD

The display module is a Gravitech 320 × 240 color LCD with a touchscreen. The module also contains a touch-screen controller as well as some buffer chips and a back-light driver. The physical connection to the display is with a 12" × 2.1" header. The software interface to the touch controller is via a SPI port. An 8-bit bidirectional parallel port along with several control lines connects to the display itself.

I chose a SparkFun Electronics UBW32 USB 32-bit whacker development board which contains a PIC32MX795 processor, an oscillator, 3.3- and 5-V power regulators, five LEDS, three pushbuttons, and a USB bootloader. The UBW32 has all of the processor pins brought to the edge of the board for easy connection with 0.1" headers. The PIC32MX795 is a 100-pin surface-mount device, so having the chip already mounted on a circuit board is a lifesaver. The UBW32 is basically a breakout board for the PIC. I considered some other single-board solutions, but most carry too much baggage trying to be too versatile. The UBW32 is a nice, simple board that is easy to use. A side benefit of using a PIC chip is that the MPLAB IDE and C compiler are completely free!

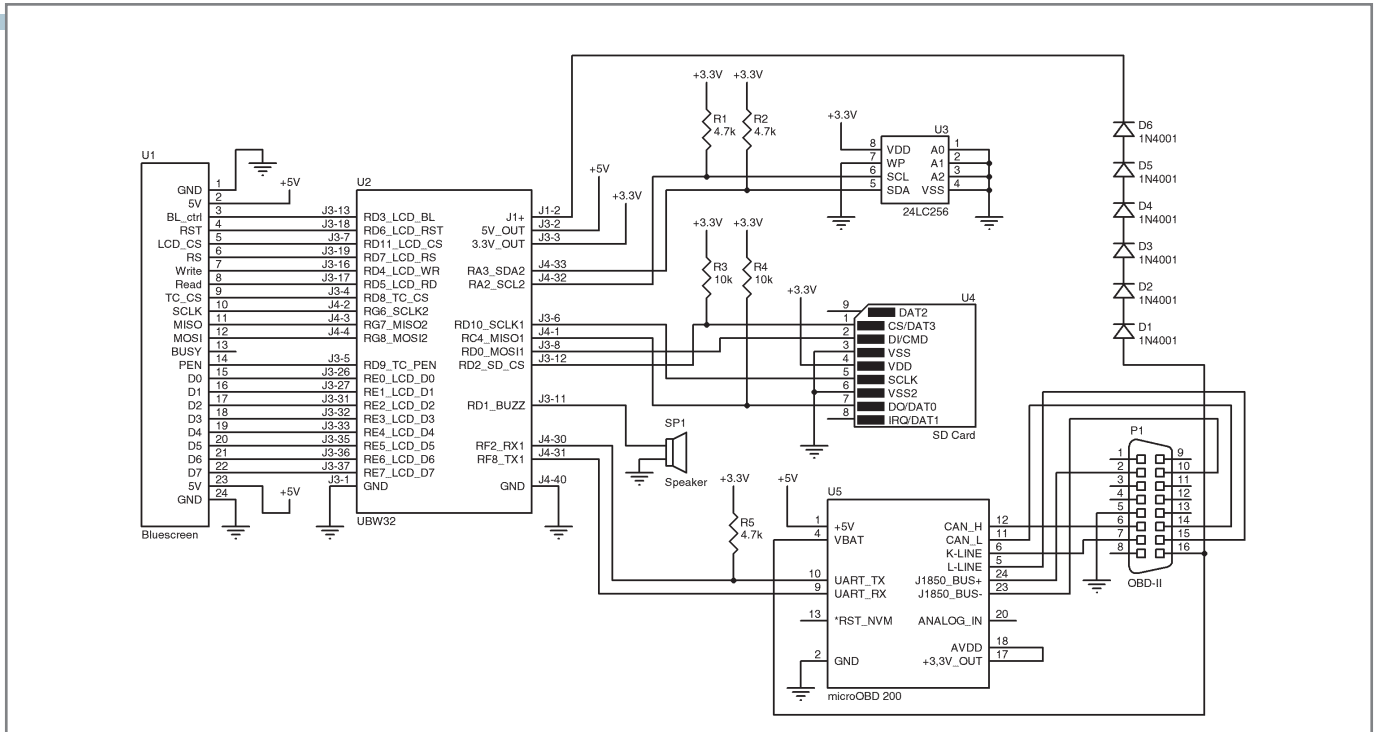The microOBD module contains the OBD-II interpreter

"Using a modular approach, it is easy to construct a monitor for an OBD-II-compatible vehicle that enables you to scan many different engine functions."

Figure 2—This schematic is for prototype testing. It has minimal filtering and protection. A later version will include a fuse and an LC filter.

as well as all the discrete components necessary for interfacing to the vehicle OBD-II diagnostic connector. Even though OBD-II is a standard, each manufacturer is free to use any electrical interface they desire, as long as the protocol follows the standard. This means you need different wiring to accommodate the various interfaces,

such as VPW, KPW, and CAN. Along with different electrical wiring there is also different software involved. The microOBD takes care of all these headaches and makes every car, no matter what the manufacturer is, appear identical to the application program. The UBW32 connects to the microOBD with a TTL serial port. The application program talks to the vehicle by sending PID requests to the engine control unit (ECU). The ECU then responds with the desired data. For example, if you want to know the vehicle speed, send the string "01 0D" to the microOBD. The microOBD translates the request, sends it to the vehicle ECU, waits for the ECU response, translates it, and sends a response such as, "41 0D 32" to the application program, where "32" is the scaled vehicle speed.

A SparkFun Electronics 256-Kb I²C EEPROM is used for nonvolatile storage of various configuration settings. The interface to the EEPROM is via an I²C port. An SD card socket is used for data logging storage. A handful of resistors, diodes, and a SparkFun buzzer round out the required components.

The 3.3- and 5-V power regulators on the UBW32 are used to power the entire project and the 5-V regulator is powered from the 12-V vehicle power through the OBD-II connector P1. Feeding a 5-V linear regulator with 12 V is a recipe for heat generation if there is much current involved. There are several diodes used to drop the 12-V vehicle power to around 8 V for feeding the 5-V regulator on the UBW32. The display has an LED backlight that can draw almost 100 mA at full brightness. At that kind of current, the 5-V regulator on the UBW32 got a little toasty when running at 12 V on the input, which is why the dropping diodes were added. A PWM output is used to drive the backlight so that the intensity can be varied, which helps lower the current requirements. A block diagram of the project is shown in Figure 1.

Warning: an automobile power system can be a very nasty environment. Alternator noise and high-voltage spikes are not uncommon.

The schematic is for prototype testing and has minimal filtering and protection (see Figure 2). A fuse and some type of LC filter will be added in a later version.

PROTOTYPE ASSEMBLY

The prototype was constructed on a 4.5″ × 6.25″ piece of 0.1″ perfboard with copper pads on one side. Standard wire wrap techniques were used to connect the modules. Photo 1a shows a picture of the top of the board and Photo 1b shows the bottom of the board. Header pins were soldered onto the display module and the UBW32, then the modules were soldered onto the perfboard. A standard 24-pin DIP wire-wrap socket was used to hold the microOBD module. Note that the SD socket was mounted on the bottom of the board so it could be soldered to the copper pads as a way to physically mount it. The pins on the SD socket are not exactly 0.1″ on center, but
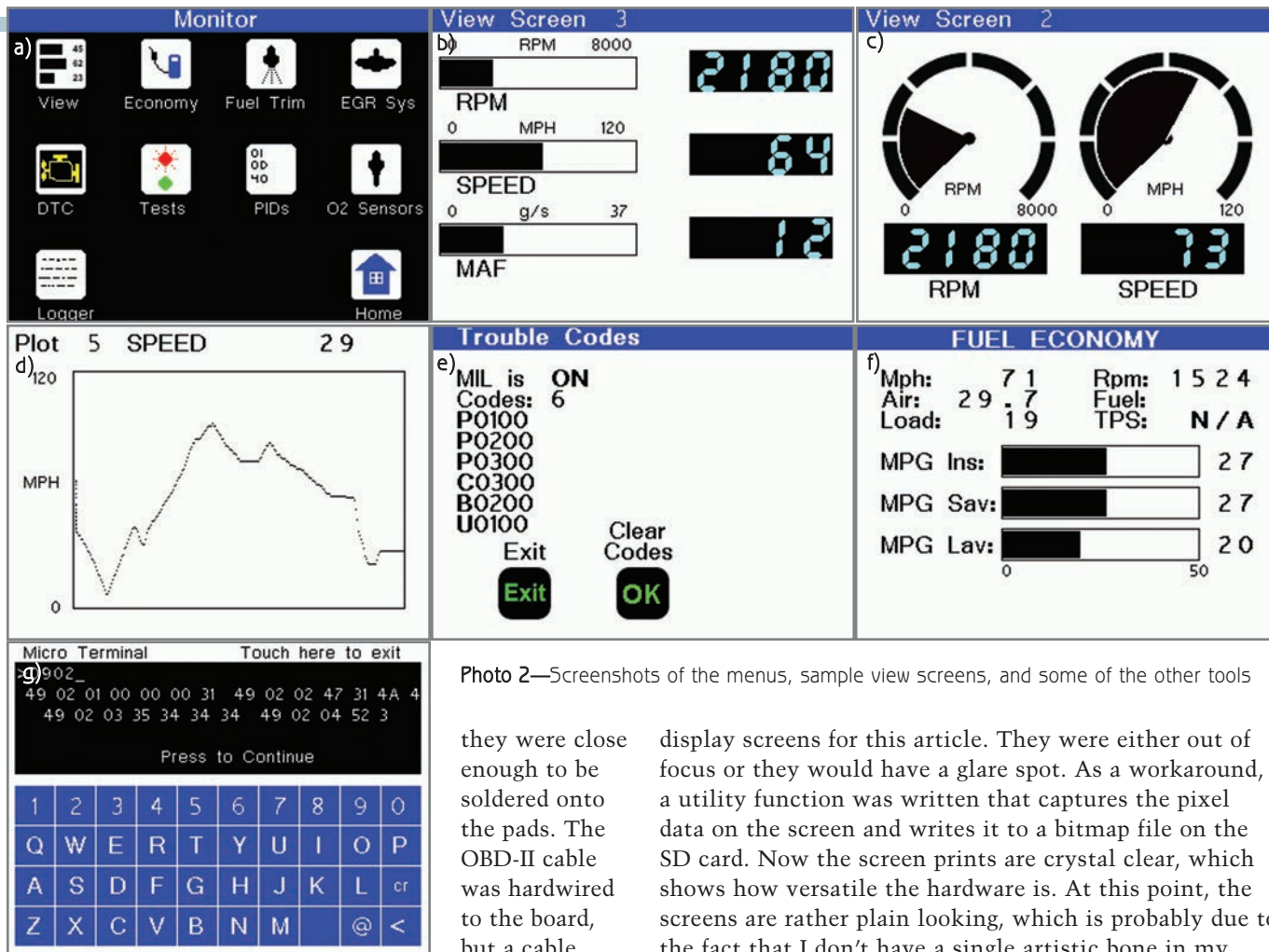
Photo 2—Screenshots of the menus, sample view screens, and some of the other tools

they were close enough to be soldered onto the pads. The OBD-II cable was hardwired to the board, but a cable with a DB9 connector is also available and could be used with a corresponding DB9 connector on the board, if desired.

SOFTWARE & OPERATION

The software is written completely in C and is compiled with Microchip's MPLAB compiler. It is a collection of pieces from different sources as well as my own work over several years of evolution. It's a constantly moving target. I am a self-taught programmer, so if you choose to use my code, please use it only as a guide, it is far from optimized.

The software can be broken down into a few main parts that correspond to the hardware modules: text and graphics functions, EEPROM read and write functions, OBD commands and responses, touchscreen scanning and coordinate calculation, SPI functions for the SD card interface, and a FAT file system.

The unit is set up with 16 user-configurable monitor screens and several preprogrammed screens, as well as some diagnostic tools and utilities. On each configurable monitor screen up to six variables can be shown, depending upon what type of screen is selected. You can choose the type of display parameters shown. There are five different types of monitor screens to choose from.

It was difficult to get good quality pictures of the various

display screens for this article. They were either out of focus or they would have a glare spot. As a workaround, a utility function was written that captures the pixel data on the screen and writes it to a bitmap file on the SD card. Now the screen prints are crystal clear, which shows how versatile the hardware is. At this point, the screens are rather plain looking, which is probably due to the fact that I don't have a single artistic bone in my body. With a little effort and artistic talent they could be dressed up a bit. The border for the circular gages was actually drawn with Inkscape and converted to an .h file for displaying on the LCD, so there are possibilities for improvement.

One of the preprogrammed screens is used for monitoring fuel economy. It shows speed, RPM, load, mass airflow, and calculated fuel usage in miles per gallon (MPG). With the ever-rising price of gas, this screen comes in handy when trying to squeeze out a little more MPG.

Another screen gives you access to the malfunction indicator lamp (MIL) status. If the Check Engine light in your car comes on, you can use the scanner to retrieve the trouble codes and turn off the MIL, if desired.

There is a HyperTerminal-type utility that enables direct interaction with the microOBD module by issuing an AT/ST command and displaying the response. The scanner is also useful as a diagnostic tool. Watching the long- and short-term fuel trim values can help diagnose injector issues. Photo 2 shows screenshots of the menus, sample view screens, and some of the other tools.

WHAT'S NEXT?

Items on the to do list include adding a realtime clock/calendar backed up with a super capacitor, a mass storage device driver so log files can be easily transferred to a PC, a

power manager to put the entire unit into low-power mode, and power supply protection and filtering.

In conclusion, using a modular approach, it is easy to construct a monitor for an OBD-II-compatible vehicle that enables you to scan many different engine functions. ⬚

Robin Brophy (myedash@yahoo.com) has a BSEE and an MSEE from North Dakota State University. He currently works in instrumentation and process control.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2011/254.

RESOURCES

Electronic Lives Manufacturing, FatFs Generic FAT File System Module, http://elm-chan.org/fsw/ff/00index _e.html.

Inkscape, Open Source Scalable Vector Graphics Editor, http://inkscape.org.

ScanTool.net, "User Guide ECUsim 2000 Multiprotocol Software Configurable OBD-II ECU Simulator," 2000, www.scantool.net/scantool/downloads/101/ecusim_2000-ug.pdf.

SOURCES

WinALDL ALDL reader
Jonas Bylund | http://winaldl.joby.se

ELM327 OBD-to-RS-232 Interpreter
Elm Electronics, Inc. | www.elmelectronics.com

Motorola MC68705 microcontroller
Freescale Semiconductor, Inc. | www.freescale.com

TFT Color LCD
Gravitech | www.gravitech.us

MAX7456 On-screen display (OSD) generator
Maxim Integrated Products | www.maxim-ic.com

PIC18F2620 Microcontroller
Microchip Technology, Inc. | www.microchip.com

BASIC Stamp microcontroller
Parallax, Inc. | www.paralax.com

microOBD 200 microcontroller and OBD cable
ScanTool.net | www.scantool.net

UBW32 USB 32-bit Whacker, PIC32MX795 development board, I²C EEPROM, and Buzzer
SparkFun Electronics | www.sparkfun.com