CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

FEATURE ARTICLE

by Bruce D. Lightner

AVR® 2004 DESIGN CONTEST
Contest Winner

AVR-Based Fuel Consumption Gauge

With gas prices at record highs, wouldn't it be useful to be able to monitor your vehicle's fuel consumption rate in real time? Bruce's fuel consumption gauge is the answer.

I came up with this project because I wanted to be able to determine my gas-powered, four-wheel-drive 1999 Chevy Suburban's fuel consumption rate, especially when it's towing my large travel trailer. The best highway fuel consumption rate I ever saw was approximately 13 miles per gallon (mpg). During a summer trip across the country, my truck's highway mileage varied from a low of 6.5 to a high of 9 mpg while towing. I desperately wanted to see a real-time readout of the truck's fuel consumption rate in miles per gallon. I figured I could start adjusting my driving habits when the miles per gallon got too bad.

My truck's power train control module (the engine computer) supplies two key real-time engine parameters that enable me to calculate the instantaneous fuel consumption rate in miles per gallon: speed and the intake airflow rate. More on this later.

As for the dashboard display, I selected an off-the-shelf, after-market electronic tachometer from a local auto parts store. As you'll see, the inexpensive tachometer provides an easy-to-read analog display that features a one-wire digital input. I also wanted a visual indication whenever the engine's fuel system controller switched over to Open Loop mode. Modern engines enter this mode when they start, but they also reenter it for extra power and when trying to protect the catalytic converter—one of the only parts in your vehicle that's warranted for 100,000 miles! At high speed under heavy load, a modern engine computer protects the catalytic converter by enriching the air/fuel mixture, thereby cooling the exhaust gases and ruining your gasoline mileage. Avoid this operating mode if you care about gas mileage.

OBD-II

If your car was manufactured for sale in the U.S. after 1995, it includes a 16-pin on-board diagnostics (OBD-II) connector somewhere in the passenger compartment. The OBD-II connector enables a scan tool to read diagnostic data from the engine computer. By law, the OBD-II connector must supply a number of key parameters relating to the on-board monitoring of emissions from vehicles, including data such as engine speed, coolant temperature, and oxygen sensor readings. Table 1 is an abbreviated list of OBD-II parameters.

Although the OBD-II connector has provisions for a total of 16 signal lines, only two or three are needed to communicate with a given vehicle. The OBD-II connector standard provides for four distinct bit-serial electrical interfaces: SAE J1850 VPW, SAE J1850 PWM, ISO 9141-2, and ISO 15765 controller area network (CAN). Each manufacturer is free to pick any one of these buses to provide the legally mandated data defined by OBD-II regulations. In the case of ISO 9141-2, there is a choice of two different network protocols, the newest being Keyword 2000 (ISO 15031). The OBD-II connector also provides unswitched 12-V power from the battery and two ground connections.

Why so many bus choices? When OBD-II regulations were written in the early 1990s, there were three

PID	Size	Data format	Description
0x00	4	One bit each PID (1 = present)	PIDs supported (0x01–0x20):
0x01	4	See SAE J1979	MIL lamp status, monitor support/status, and no. of DTCs
0x03	2	0, 2, 3, 4 = open, 1 = closed	Fuel system status
0x04	1	100/255% per bit	Calculated load value
0x05	1	1°C per bit, –40°C offset	Engine coolant temperature
0x0A	1	3 kPa per bit	Fuel pressure
0x0B	1	1 kPa per bit	Intake manifold absolute pressure (MAP)
0x0C	2	1/4 rpm per bit	Engine speed (revolutions per minute)
0x0D	1	1 km/h per bit	Vehicle speed (kilometers per hour)
0x0E	1	1/2° per bit, –64° offset	Ignition timing spark advance
0x0F	1	1°C per bit, –40°C offset	Intake air temperature
0x10	2	0.01 gm/s per bit	Mass air flow (MAF) sensor rate
0x11	1	100/255% per bit	Absolute throttle position sensor
0x12	1	0 = upstream, 1 = down, 2 = off	Commanded secondary air status
0x13	2	See SAE J1979	Location of oxygen sensors
0x1C	1	1 = California, 2 = federal	OBD requirements level

Table 1—*An abbreviated list of OBD-II Mode 0x01 PIDs, this table shows the kinds of real-time data that you can read from you car's engine computer. Multi-byte data returns in big endian format. This is a tiny subset of the data that's available from your vehicle's OBD-II bus.*

widely used serial diagnostic bus standards: Ford's (now SAE J1850 PWM), General Motors's (now SAE J1850 VPW), and Chrysler's and others' (now ISO 1941-2). Each of the Big Three got to make its diagnostic bus of choice part of the OBD-II standard.

Keyword 2000 appeared in model year 2000. CAN OBD-II first appeared in model year 2003 vehicles. CAN-bus is scheduled to be the only OBD-II bus allowed for new vehicle designs by 2008. However, because the new vehicle design cycle takes approximately five years, and because vehicles last upwards of 15 years (30-plus years in my household!), CAN-bus will not take over any time soon.

SAE J1850 VPW

It's possible to build a generic diagnostic scan tool that can communicate with any vehicle produced today. But in the interests of cost and complexity, this project is confined to communicating with General Motors's SAE J1850 VPW diagnostic bus, which is what my 1999 Chevy Suburban uses.

The SAE J1850 Class B Data Communication Network Interface standard provides for variable pulse width (VPW), bit-serial communication at approximately 10 Kbps using a signal wire (OBD pin 2) referenced to the vehicle's ground (OBD pin 5). All communicating nodes in the vehicle share the same bus wire. The nodes signal one another using voltage transitions varying between 0 and approximately 8 V. The J1850 VPW bus also includes a fixed pull-down resistor to ground (500 to 1,500 Ω) such that when no node is actively driving current into the bus, the J1850 VPW bus returns to ground.

Any node connected to the J1850 VPW bus must be capable of accepting direct shorts to the chassis ground or the vehicle battery voltage. It must also tolerate the battery voltage reversed. (Sometimes people hook up their car batteries backwards!) I know from personal experience that the engine isn't guaranteed to run if the bus shorts out. Hardware hackers should note that this robust electrical interface standard means that it's unlikely you'll fry your vehicle's expensive engine computer while experimenting.

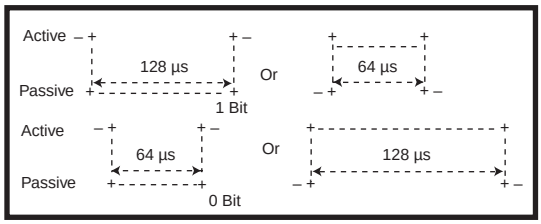

Figure 1—*These sample waveforms, straight from the SAE J1850 VPW standards document, show you how to send variable pulse width (VPW) data bits to the engine computer. The width and polarity of the data bit pulses determine whether you're sending a 0 bit or a 1 bit.*

SAE J1850 VPW messages are prioritized using a preassigned identifier. Nodes take turns signaling each other either by driving approximately 8 V onto the bus (active state) or not (passive state). This scheme provides for nondestructive collision detection. If a transmitting node sees a positive voltage (active state) on the bus when it expects no voltage (passive state), the node immediately knows that another (higher priority) node is using the bus. Lower-priority nodes back off the bus in such a way to not interfere with a higher-priority node's ongoing transmissions.

Data packets are sent over the J1850 VPW bus as a series of 8-bit bytes. A trailing cyclic redundancy check (CRC) byte is used for error detection. The beginning of a data packet is signaled by an active start-of-frame (SOF) pulse of 200 µs proceeded by a passive period of at least 300 µs. Successive bits are sent MSB first using alternating negative (passive) and positive (active) pulses of varying lengths (either 64 or 128 µs). After the SOF pulse, each voltage transition represents 1 bit.

The time between transitions, coupled with the pulse polarity, determines whether a 0 bit or a 1 bit was transmitted, as revealed in the waveforms shown in Figure 1. The end of the data packet (EOD) is signaled by a 200-µs passive period.

A sample J1850 VPW packet is shown in Figure 2. Note that the J1850 VPW data rate is variable, ranging from 976 to 1,953 bytes per second, depending on the exact bit patterns in a packet. The average data rate is approximately 1.3 Kbps. This relatively low data rate is why modern vehicles are transitioning to the CAN standard, which signals at either 250 or 500 Kbps (i.e., 30 or 60 KBps).

SAE J1979 PACKET FORMAT

The format of J1850 VPW OBD-II packets is specified by the SAE standard J1979 "E/E Diagnostic Test Modes." Every OBD-II data packet begins with a 3-byte header. That's 1 byte each for priority/type, target address, and source address. The header is followed by one or more data bytes and a trailing CRC byte (as defined by SAE J1850). OBD-II data is collected using a strict request-response protocol. For the purposes this project, real-time OBD-II data is requested using a header of 0x68, 0x6A, and 0xF0 followed by two data bytes indicating exactly which data item you want to read. I selected 0xF0 as the source address, but anything in the 0xF0 to 0xFF range indicates a diagnostic scan tool.

Diagnostic data items are selected by sending a request packet with a mode byte of 0x01 and a single parameter identifier (PID) byte. Table 1 shows a list of possible PID bytes. For example, vehicle speed can be requested using mode 0x01, PID 0x0D with the following data packet:

```
0x68, 0x6A, 0xF0, 0x01, 0x0D, CRC
```

where *CRC* is the SAE J1850-defined CRC byte. You may download the CRC source code from the *Circuit Cellar* FTP site.

The engine computer responds to this request packet within 100 ms

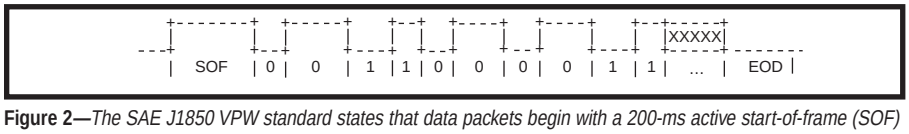

Figure 2—*The SAE J1850 VPW standard states that data packets begin with a 200-ms active start-of-frame (SOF) pulse, immediately followed by variable-width pulses of alternating polarity (one per bit, MSB first). The width and polarity of the bit pulses determine their value. The packet ends with a 200-ms end-of-data (EOD) passive period.*

with the following response packet:

```
0x48, 0x6B, ECU, 0x41, 0x0D,
  KPH, CRC
```

where *ECU* is the engine computer's ID byte (e.g., 0x10). *KPH* is a byte encoded with the current vehicle speed in kilometers per hour. Note that the response packet echoes the mode byte 0x01 with bit 6 set (i.e., 0x41).

Table 1 shows a selected list of mode 0x01 PIDs, including information on the size (bytes) and format of the response data values that are returned by the engine computer. Note that multi-byte response data values are returned in big endian order (i.e., MSB first).

For this project I was interested in PIDs supported (0x00), vehicle speed (PID 0x0D), fuel system status (0x03), and the airflow rate from mass airflow (MAF) sensor (0x10). Note that not all vehicles have an MAF sensor. You can read the PIDs supported bit mask to check if this sensor is present. Most large gasoline engines employ an MAF sensor to monitor engine intake airflow as part of the engine control firmware because this method provides for better air/fuel mixture control. You also need to read fuel system status because you'll want your miles per gallon gauge to indicate when the engine isn't running in Closed Loop mode.

DERIVING MILES PER GALLON

So, given the vehicle's speed (kilometers per hour) and the mass airflow rate (grams per second), how do you determine miles per gallon? Knowing a couple of other constants makes this work.

The first constant is the engine's air/fuel ratio. In modern, low-emissions vehicles, the air/fuel rate is maintained at a constant chemically ideal ratio of 14.7 g of air to 1 g of gasoline. You can convert grams of air per second into grams of gasoline per second by dividing by 14.7.

The second constant needed is the density of gasoline in grams per gallon. The density of gasoline varies somewhat according to the fuel grade and ambient temperature. But given the accuracy of the display, the following constant works well for brand-name unleaded gasoline: 6.17 pounds per gallon. Knowing that there are 454 g in a pound, you can divide the mass airflow rate by 14.7 and by 2,801 (i.e., 6.17 × 454) to determine the fuel flow rate in gallons per second. Multiply that number by 3,600 (the number of seconds in 1 h) to determine the gallons per hour.

Now you just need the vehicle speed in miles per hour so you can divide this by the previous number to yield the instantaneous miles per gallon. First, you need to convert the OBD-II vehicle speed reading (VSS) from kilometers per hour to miles per hour by multiplying by 0.621371. You also need to scale the MAF sensor reading by dividing it by 100 because the engine computer returns MAF as grams per second × 100, as a big endian 16-bit quantity.

Here is the final formula:

$$MPG = \frac{14.7 \times 6.17 \times 454 \times \left(VSS \times 0.621371\right)}{\frac{3{,}600 \times MAF}{100}} = \frac{710.7 \times VSS}{MAF}$$

You'll use an 8-bit microcontroller for this project, so let's get rid of the floating-point numbers. You'll calculate and display miles per gallon × 10. Therefore, you need to calculate:

$$MPGx10 = 7{,}107 \times \frac{VSS}{MAF}$$

The C code looks like the following:

```
MPGx10 = (unsigned short)((VSS
   * 7107L)/MAF);
```

`VSS` is an 8-bit reading. `MAF` and `MPGx10` are 16-bit quantities. No floating-point math required! This C code uses only the C runtime library's 32-bit integer multiply and divide routines. Because of the precision of the quantities involved, there is no worry about integer overflow.

ANALOG DISPLAY

An automobile's dashboard is a horrible environment for electronics in general and display electronics (e.g., liquid crystals) in particular. Whatever the display technology you use, it must function both in dark as well as light conditions. The temperature of a dashboard can go from below freezing to scorching temperatures that melt some plastics. Although modern digital displays offer flexibility, an old-fashion analog display (a needle swept over a meter face) would be hard to beat for this application.

Your local auto parts store probably has a fine selection of inexpensive analog gauges. One type of gauge, the electronic tachometer, has a one-wire digital interface that easily connects to a microcontroller. Electronic tachometers sense engine speed by connecting a single wire to the low-voltage side of the ignition coil where it connects to the distributor's points. The tachometer senses the 0- to 12-V pulses sent to the coil (a high-voltage transformer), which in turn causes your vehicle's spark plugs to spark.

Electronics in the tachometer measure the pulse repetition rate sensed at the coil after a bit of signal conditioning to keep noise spikes from frying the instrument's insides. The eight-cylinder, four-cycle gasoline engine in my truck sparks four times per engine revolution. To use an electronic tachometer as an analog display, I just needed to supply a simulated ignition coil pulse train. The tachometer that I chose goes from 0 to 8,000 rpm. Therefore, to get a full-scale deflection of the needle with the tachometer set to Eight Cylinder mode, I need to send 32,000 0- to 12-V pulses per minute, or 533.3 pulses per second, which is a 1.875-ms pulse period.

By varying the pulse period over the range of 0 to 1.875 ms, you can get any meter deflection you want using a single I/O pin. You'll use a pulse train duty cycle of 50%. This is the same as an ignition dwell angle of 180°. Modern electronic tachometers are mostly insensitive to the dwell angle.

NEW FACE ON THE METER

I used a $30 Sunpro Sun Super Tach II CP7903 electronic tachometer for this project. The tachometer, which has a white face that's approximately 3″, came with a four-wire interface (12-V, ground, coil, and panel lamp power). I had to change the meter's face to

show a miles per gallon scale in the range of 0 to 40. A little microsurgery on the bezel's plastic tabs exposed the metal faceplate. I removed the needle with a dinner fork by applying even pressure to the back.

I copied the stock meter face with a photo scanner and used my favorite drawing program to create a new meter face with a 0 to 40 mpg scale. I added a valid/error label for the bicolor LED to show the status of the engine computer communication link. The new meter face was printed on adhesive-backed label stock and applied over the original meter's faceplate. After reassembling the meter, I powered it up to determine the zero point. I then reattached the indicating needle with a drop of glue (see Photo 1).

I replaced the original meter's power-hungry, 12-V incandescent bulb with two bright white LEDs. I also added a Mode Select momentary push button switch. Finally, I added a tiny circuit board with a surface-mount bicolor LED behind the meter's faceplate.

My custom meter has seven signal wires: 12-V power, ground, ignition coil, white LED power, push button switch, red LED, and green LED. These were terminated with two four-pin connectors that mate with the microcontroller circuit card (see Figure 3).

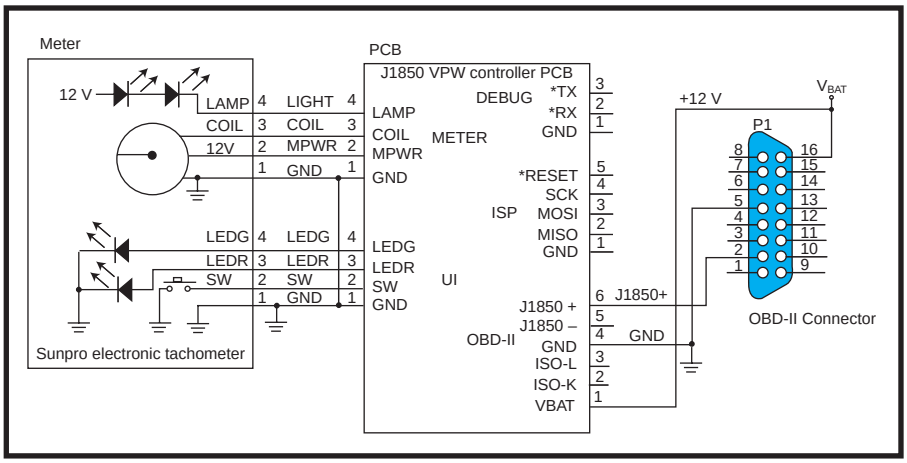

Figure 3—*Minor additions to the stock Sunpro tachometer include the bicolor indicator LED, a Mode Select push button, and two white LEDs to light the meter after dark.*

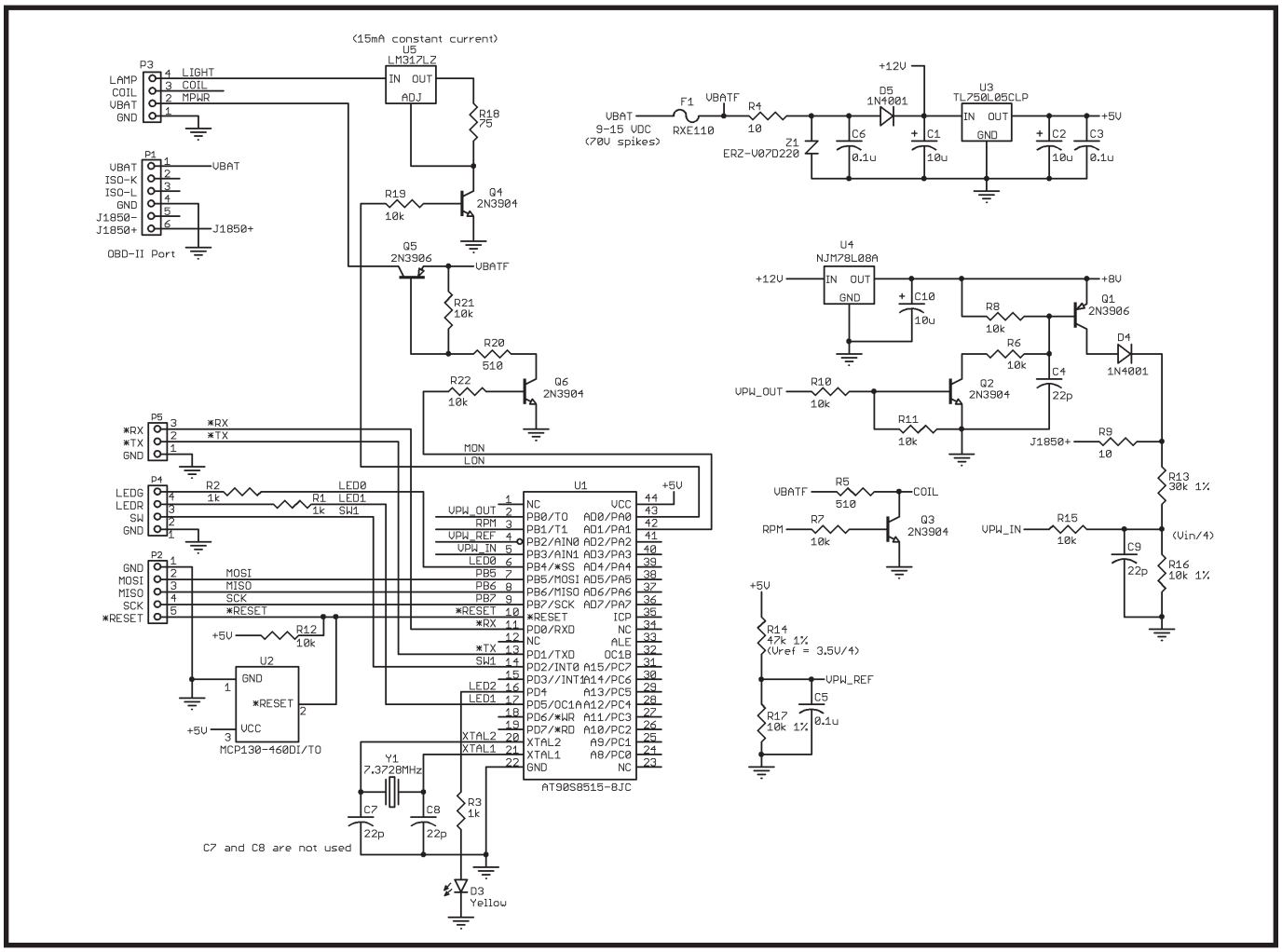

Figure 4—*An AT90S8515 microcontroller and a few transistors, diodes, resistors, and capacitors enable the firmware to read your vehicle's speed and intake airflow rate from the engine computer and then calculate and display miles per gallon using an off-the-shelf electronic tachometer.*

MCU INTERFACE

As you can see in Figure 4, I used an Atmel AVR AT90S8515 8-bit RISC microcontroller (U1) running at 7.3728 MHz. Today, the ATmega8515 would be a better choice. The ATmega8515 is 100% pin- and function-compatible with the older AT90S8515 part. Both parts come with 8 KB of in-circuit programmable flash memory-based instruction memory, 512 bytes of SRAM, and 512 bytes of on-chip EEPROM. The AVR RISC instruction set is well suited for programming in C and it's extremely efficient.

In the automotive electronics design business, we sometimes refer to the automobile's 12-V electrical bus as "the power supply from hell." You must pay careful attention to a number of unwritten rules when drawing power from the battery. Ignore these rules, and your sensitive digital electronics will be doomed!

The nominally 12-V lead-acid battery in your car can vary from a low of less than 9 V (when cold-cranking your engine with a weak battery) to more than 14 V when charging. Sometimes that voltage even jumps up to a more or less steady 24-plus V when the tow truck driver decides to jump-start your car with a dual 12-V battery system!

Also, be prepared for 70-plus-V noise spikes from various inductive loads attached to your vehicle's electrical system. Even more amazing is how bad things get if the 12-V battery is removed from the circuit. The chemistry of the lead-acid battery itself normally quiets the nominally 12-V power bus in an automobile. A loose battery wire or bad contact can take the battery, and its calming effects, in and out of the circuit. Bad news indeed!

Given everything I've mentioned, I tend to favor the belt and suspenders approach to drawing power from an automobile's battery. As you can see in Figure 4, battery power (V_{BAT}) is first fused with polysilicon fuse F1. It's then current limited with R4, overvoltage protected with the polysorb Z1, filtered with capacitors C6 and C1, and reverse voltage protected with diode D5 before being used by other circuitry.

Regulated 5-V power comes from U3. The AT90S8515's U1 is programmed in-circuit with programming connector P2. Connector P5 provides access to the microcontroller's UART transmit and receive pins for debugging. External RS-232 level-shifting logic is needed in this case.

The six-pin connector P1 connects to the OBD-II bus signals, including battery power and ground. The J1850 VPW signal is passed through current-limiting resistor R9 before it's divided by four using resistors R13 and R16 and filter capacitor C9. Protection resistor R15 routes the divided input voltage to one of the microcontroller's analog comparator pins (U1-5).

Resistors R14 and R17 and capacitor C5 provide a stable reference voltage for detecting the bit-serial input from the J1850 VPW bus using the second analog comparator pin (U1-5). The analog comparator's state changes each time the J1850 VPW bus input passes through 3.5 V.

The circuit that provides active J1850 VPW pulses to the OBD-II bus starts with a regulated 8 V from U4. This voltage is driven into the J1850 VPW bus through diode D4 and current-limiting resistor R9 using NPN transistor Q1. Transistor Q1 is normally biased off by pull-up resistor R8. The NPN transistor Q2, which is biased normally off by pull-down resistor R11, level shifts the microcontroller's 0- to 5-V digital output to turn on Q1 whenever pin U1-2 is driven high. The RC filter made up of R6 and C4 controls the slew rate of transistor Q1 via its base current. When the AT90S8515 resets, transistor Q1 is off, leaving the J1850 VPW bus in a safe passive mode.

Simulated ignition coil 0- to 12-V pulses are provided by Q3, R7, and R5. A high on the microcontroller RPM output pin U1-3 drives the COIL signal to ground, simulating closed points in the vehicle's distributor. This digital output generates a pulse train at the interrupt level under the control of the AT90S8515's 16-bit timer/counter using the CompareA and CompareB interrupts.

The microcontroller directly drives three LEDs using current-limiting 1-kΩ resistors R1, R2, and R3. One LED is mounted on the PCB, and the other two are part of a bicolor LED assembly inside the meter housing connected via P4. This connector also carries the signal from the momentary grounding push button switch in the meter. Push button sensing pin U1-14 on the microcontroller is applied as an input, using an on-chip pull-up resistor.

The COIL signal and the fused raw battery voltage (V_{BATF}) are supplied to the meter via connector P3. The LIGHT signal is a switched, constant-current path to ground wired to two bright white LEDs in series with the V_{BATF}. Regulator U5 limits current to a constant 15 mA. The NPN transistor Q4 switches the LEDs on and off using digital output pin U1-43 (LON).

The 12 VDC power to the meter can be switched on and off using the microcontroller's digital output MON

Photo 1—*These are before* **(a)** *and after* **(b)** *photos of my off-the-shelf electronic tachometer from the local auto parts store. I replaced the stock face with my own in order to display miles per gallon. I also added a red/green LED to indicate how well the meter is working.*

(U1-42), which controls the base of NPN transistor Q6. When MON is high, Q1 pulls the base of NPN transistor Q5 low, sending battery power to the meter's power pin P3-2.

FIRMWARE

I developed the firmware for this project in C using version 2.95.2 of the GNU C compiler GCC for AVR microcontrollers. Development was performed in Windows using the WinAVR tool set, which includes a complete set of the standard command-line UNIX utilities compiled for Windows, as well as AVR-GCC, which is the AVR version of GCC, and all of its attendant programs and libraries.

The firmware for this project is relatively simple. There are two basic functions that must be performed. One is to repeatedly read the vehicle speed and the MAF rate from the engine computer and convert the readings into miles per gallon. The other function involves generating a pulse train on the RPM output pin to drive the tachometer needle to the desired value. You may download the source code from the *Circuit Cellar* FTP site.

The RPM pulse train that drives the tachometer is generated using the AT90S8515's 16-bit counter/timer and a couple of interrupt service routines. The `meter_set()` firmware routine is responsible for this.

A background task sends commands to the engine computer via the J1850 VPW bus to repeatedly read vehicle speed and airflow rate, listen for the responses, and calculate and set the RPM pulse period to drive the meter using `meter_set()`. At the lowest level, you need a routine (`vpw_send()`) to send a command packet to the engine computer and a companion routine (`vpw_recv()`) to read any response packets from the computer.

The `vpw_send()` routine first waits for a gap of at least 300 µs on the J1850 VPW bus. It then it sends a positively moving (active) 200-µs start-of-frame (SOF) pulse that's followed by alternating transitions between the passive and active states (one transition per bit), delaying either 128 or 64 µs between each transition, according to the J1850 VPW standard (see Figure 1). After sending the trailing CRC byte, the bus is passive for 200 µs.

The `vpw_recv()` routine is slightly more complex because it must time out if no response from the engine computer is detected within 100 ms. The routine begins by waiting for a 200-µs SOF pulse. It then begins collecting bits, assembling them into bytes, and storing them in the RAM packet buffer. The routine exits after `vpw_recv()` detects that the bus has remained passive for more than 200 µs.

The fuel system's status is also read each read and display cycle, looking for open loop status. If this state is detected, the meter setting is tweaked each read and display cycle. This causes the miles per gallon needle to wiggle back and forth slightly around the true miles per gallon reading. The needle wiggle provides a clear visual indication of Open Loop mode.

The bicolor LED on the meter face is controlled by the high-level AVR

firmware. If the engine computer fails to respond to an OBD-II request, the red LED is lit. Error-free reads light the meter's green LED. The third activity LED on the PCB is toggled on each time a request is made to the engine computer; it turns off when the request is completed.

Besides displaying miles per gallon, the meter's firmware can be configured to display many other real-time engine computer parameters using the push button to select the data to display. Unfortunately, the meter needs a more complex faceplate to make this information readable by anyone besides a gear head like me!

CONSTRUCTION

The printed circuit card has 100% through-hole components, including a socket for the AT90S8515 in a 44-pin PLCC package. Axial components like resistors and diodes are inserted vertically to conserve space (see Photo 2).

I didn't use an enclosure for the PCB. The entire circuit board assembly was small enough to be inserted in 2.5″ (diameter) heatshrink tubing and sealed with a heat gun. The on-board LED pokes through a small hole. This "wart in the cable" package fits under my truck's dashboard. If converted to a completely surface-mount design, this circuit could be easily inserted in the tachometer's existing enclosure.

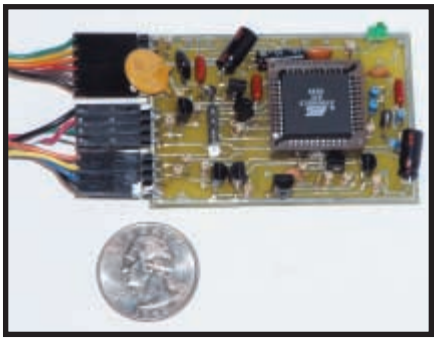

Photo 2—*The fully assembled circuit card is small enough to be slipped into a length of heatshrink tubing. The result is a compact "wart in the cable" package that disappears under the dashboard.*

I didn't hardwire the miles per gallon meter to my truck's J1850 VPW bus. Instead, the meter plugs into the OBD-II connector to access the three signals it uses. Because I switched to low-power white LEDs to light the meter, I can afford to leave the meter plugged in when the PCB is powered, even when the ignition key is off.

You may want to take the J1850 VPW signal and ground from the wires in the back of your OBD-II connector (pins 2 and 5, respectively). If you decide to use the tachometer's stock incandescent lamp, its power should come from the dashboard's dimmer switch. Because this meter may interfere with a factory OBD-II scan tool, you may also want to add a switch to disconnect the meter whenever a factory scan tool is used.

ENGINE COMPUTER

For development and testing, I used a stand-alone engine computer module naked on my lab bench. Used engine computers are cheap. They're easy to locate online from the nationwide junkyard network. I used Car-Part.com (www.car-part.com). A perfectly fine J1850VPW engine computer from a 1996 Chevy Lumina van costs less than $15. (I happen to have the com-

plete set of electrical schematics for my wife's old minivan.) Salvage yard professionals call these things "brains." The secret to running a brain outside its body is knowing that it's possible. Every brain that I've put on a lab bench has required only connections for ground, 12-V battery power, and an ignition switch. Of course, you need to locate the OBD-II bus connections (one or two wires) and connect these to your external equipment as well. In your case, J1850 VPW, that's a single wire. After it's powered, the brain should respond to all of the OBD-II requests listed in Table 1.

Determining which of the 100-plus signals going to and from a modern engine computer are for the ground, battery, ignition, and OBD-II bus lines typically requires a detailed diagram. Online auto repair manual providers like ALLDATA (www.alldata.com) offer such information. For approximately $20 per vehicle, you can rent an online repair manual, including electrical schematics, for 12 months. The other option is to guess which wires to use by looking at the engine computer's PCB and components. I've hooked up a couple of brains this way. (In one case, I was lucky that the designer at Ford had worried about the power supply from hell when I got confused and reversed 12 VDC and ground!)

SMOKE TEST

After debugging the firmware with a homemade AVR simulator, I loaded the AVR firmware into my circuit board using the ISP serial programming connector and then plugged it into the Lumina's brain in my lab. I added debug print statements to the firmware in order to log exactly what was being read from the engine computer. Remember that because the brain has no body, the engine speed sensor always reads zero. The same goes for the MAF sensor. The tests showed me that everything was working as expected, with the engine computer responding correctly to my J1850 VPW request packets. The simulation exercise paid off. So far so good!

Surprise! The meter worked the first time it was installed in my truck. Only a couple of firmware tweaks were required. I added retry logic to deal with occasional negative responses to J1850 VPW requests (indicating that my truck's engine computer was busy). In addition, my simple `vpw_resp()` firmware routine sometimes would mistake other J1850 VPW packets as valid responses to data requested by the `vpw_send()` routine. The red error LED told me that I was getting read errors, especially during periods of hard acceleration and sudden braking.

Over the last couple of years, I've put over 20,000 miles on the miles per gallon meter. I think it has helped contain my lead foot, especially when I tow a trailer. Today, with the new and improved price of gasoline here in California, I'm thinking about changing the meter's face to read dollars per mile! It's just a small matter of programming.

BUILD A GAUGE

This project and its accompanying design files and source code provides a clear path to unlocking the secrets of one of the five on-board diagnostic buses mandated by the U.S. OBD-II standard (i.e., SAE J1850 VPW). The standard, which was first defined by General Motors, now applies to a number of vehicle models, including many from Chevrolet, GMC, Buick, Pontiac, Saturn, Toyota, Chrysler, Isuzu, and Daewoo.

Now you can use an inexpensive AVR microcontroller to collect real-time vehicle speed and airflow data from an engine computer using the J1850 VPW bit serial bus and display that information as a fuel consumption rate (in miles per gallon). The analog display is an off-the-shelf electronic tachometer with a modified meter faceplate. Parts for the project should cost less than $50. The electronic tachometer is the most expensive component.

This microcontroller-based design gets its power safely from the power supply from hell—the 12-VDC automotive battery bus. Plus, it has a robust connection to the vehicle's SAE J1850 VPW bus, so it tolerates ground and power-supply short circuits as well as reversed battery voltage. This magic is performed without special automotive bus interface chips. Simple transistors, diodes, resistors, and capacitors are all you need.

Remember to use a junkyard brain from your favorite vehicle. This will enable you to inexpensively experiment with an engine computer before doing the same in the driveway or on the highway! As Robert Crumb's famous poster intones, "Keep on truckin'," to which the real truckers add, "Keep the rubber side down!"

Bruce D. Lightner works for Lightner Engineering in La Jolla, California. He discovered computers several decades ago and has been building hardware and software for them ever since. His name is on more than a dozen patents in the fields of computer architecture and telematics. Among his most recent ventures is Networkcar, which produces wireless diagnostics/tracking devices for vehicles big and small. You may reach him at lightner@lightner.net.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2005/183.

RESOURCES

Atmel Corp., "AT90S8515 8-bit AVR Microcontroller with 4K/8K Bytes In-System Programmable Flash," rev. 0841E-04/99, 1999.

SAE standard J1850, "Class B Data Communication Network Interface," Society of Automotive Engineers, April 2002.

SAE standard J1979, "E/E Diagnostic Test Modes," Society of Automotive Engineers, May 2001.

SOURCES

Sun Super Tach II (CP7903)
Actron, Inc.
www.sunpro.com

AT90S8515 Microcontroller
Atmel Corp.
www.atmel.com

WinAVR
SourceForge.net (OSTG, Inc.)
http://sourceforge.net/projects/winavr