



## **A Fuel-Consumption Gauge for Your GM Car** ***Real-time Data from Your Engine Computer***

With the *new and improved* price of gasoline here in the U.S. many of us that own large SUVs want to know our fuel consumption efficiency *while* we are driving. Some newer vehicles display this “real time” data on a dashboard display, but those of us with older cars were out of luck...until now. If you drive a 1996 or newer “gas guzzler” made by General Motors you probably can make your own easy-to-read miles per gallon (MPG) display.

This project and its accompanying design files unlock the secrets of one of the five onboard diagnostic busses mandated by the U.S. OBD-II standard, namely SAE J1850 VPW, first used by General Motors, now found in a number of vehicle models, including those from Chevrolet, GMC, Buick, Pontiac, Saturn, Toyota, Chrysler, Isuzu and Daewoo. The project demonstrates how to use an inexpensive Atmel ‘8515 AVR microcontroller to collect real-time vehicle speed and air-flow 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 (MPG) using an “analog display” made from an off-the-shelf, electronic tachometer with a modified meter face-plate. Bill of materials costs are under \$50, most of that going to the electronic tachometer.

The article shows how a microcontroller-based design can be connected safely to the “power supply from hell”, the 12VDC automotive battery bus, and provide a simple, robust connection to a vehicle’s SAE J1850 VPW bus, while still tolerating ground and power-supply short circuits, as well as reversed battery voltage. This “magic” is performed without the need for special automotive bus interface chips. Simple transistors, diodes, resistors, and capacitors are all that are needed.

Through the use of embedded C-code the mysteries of the J1850 VPW protocol are revealed and a gateway to the real-time data stored in your engine computer is opened. We also see how emulating the instruction-by-instruction behavior of the AVR microcontroller can be used to speed firmware development. Finally, we show how to acquire an inexpensive junkyard “brain” for your favorite vehicle, so you can experiment with an engine computer in the lab, *before* doing the same in your driveway...or on the highway!

# Listings

## Listing 1 - Vpw\_send:

```
unsigned char vpw_send(      // returned status: 0 = error, 1 = OK
    unsigned char *buf,     // buffer to be sent via J1850 VPW
    unsigned char n        // number of bytes to sent from buf
){
    unsigned char bits, ch, timer0, period;

    wait_vpw_idle();        // wait for idle J1850 bus
    timer0_start(T0_CK8);   // this clears timer0 count
    vpw_active();           // send SOF pulse
    timer0 = VPW_SOF_CNT8;
    while (timer0_get() != timer0) ;
    while(n--) {           // sent next byte in buffer
        ch = *buf++;
        bits = 8;
        while (bits--) {  // send each bit in the byte
            if (bits & 1) {
                vpw_passive();
                period = (ch & 0x80) ? VPW_LONG_CNT8 : VPW_SHORT_CNT8;
                timer0 += period / 2; // set to delay 1/2 of period
                while (timer0_get() != timer0) ; // wait for switch
                timer0 += period - (period / 2); // wait rest of period
                while (timer0_get() != timer0) {
                    if (is_vpw_active()) return 0; // collision!!
                }
            } else {
                vpw_active();
                timer0 += (ch & 0x80) ? VPW_SHORT_CNT8 : VPW_LONG_CNT8;
                while (timer0_get() != timer0) ;
            }
            ch <<= 1;
        }
    }
    vpw_passive();         // output EOD "symbol"
    timer0 += VPW_EOD_CNT8;
    while (timer0_get() != timer0) ; // wait for EOD complete
    return 1;
}
```

## Listing 2 - Vpw\_recv:

```
unsigned char vpw_recv(     // returned no. bytes received
    unsigned char *buf,     // buffer to receive into buf
    unsigned char max       // max. bytes to receive
){
    unsigned char n, bits, ch, delay, state;
    unsigned int num100usec;

    num100usec = 0;
again:
    timer0_start(T0_CK8);
    while (!is_vpw_active()) {
        if (timer0_get() == US2T0CNT8(100)) {
            ++num100usec;
            timer0_start(T0_CK8);
        }
    }
}
```

```

        if (num100usec > 1000) { // exit if >100 msec
            return 0;
        }
    }
}
// expect SOF
timer0_start(T0_CK8);
while (is_vpw_active()) {
    if (timer0_get() == VPW_SOF_MAX_CNT8 ) goto again;
}
delay = timer0_get();
timer0_start(T0_CK8);
state = is_vpw_active();
if (delay < VPW_SOF_MIN_CNT8) goto again;
for (n = 0; n < max; ++n) {
    bits = 8;
    ch = 0;
    while (bits--) {
        ch <<= 1;
        while (is_vpw_active() == state) {
            if (timer0_get() == VPW_SOF_MAX_CNT8 ) return n;
        }
        delay = timer0_get();
        timer0_start(T0_CK8);
        state = is_vpw_active();
        if (delay <= VPW_BIT_MIN_CNT8) goto finis;
        if (delay > VPW_BIT_MAX_CNT8) goto finis;
        ch |= (delay > VPW_BIT_MID_CNT8) ? 1 : 0;
    }
    *buf++ = ch ^ 0x55;
}
return n;
}

```

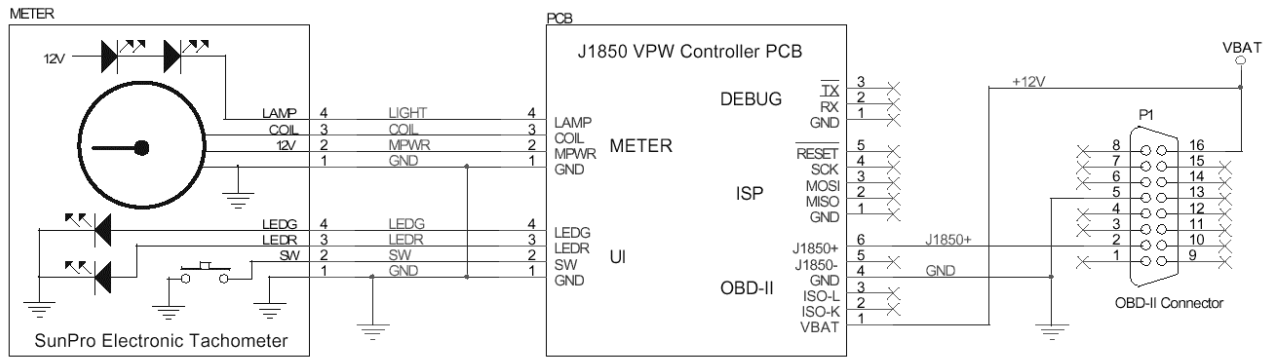
### Listing 3 - Crc8buf:

```

unsigned char crc8buf(
    unsigned char *buf, // buffer with bytes to be checksummed
    unsigned char len // number of bytes
){
    unsigned char val, i, chksum;

    chksum = 0xff; // start with all one's
    while (len--) {
        i = 8;
        val = *buf++;
        while (i--) {
            if (((val ^ chksum) & 0x80) != 0) {
                chksum ^= 0x0e;
                chksum = (chksum << 1) | 1;
            } else {
                chksum = chksum << 1;
            }
            val = val << 1;
        }
    }
    return ~chksum;
}

```



System Block Diagram

