TACHOMETER AND ENGINE HOURS COUNTER

ATMEL/CIRCUIT CELLAR Contest Entry A3768

ABSTRACT

Engines in boats, airplanes, generators, and other applications need a way to show total running time so that oil changes, overhauls, and other preventative maintenance can be done on schedule. This project answers that need. For the best human interface an analog radial meter is used to display RPM, and engine hours are shown on a digital display.

Tachometers are available that use mechanical wheels for counting engine time, but they are expensive and not always reliable. This project presents a tach with an analog meter to display RPM and a 6-digit 7-segment LED display showing engine hours with a resolution of 1/100 hour. The hour meter counts only during the time that the engine is actually running. The total hours are stored in the EEPROM of an ATMEL microcomputer, using a circular buffer scheme to reduce the number of erase cycles required. The RPM is displayed on a meter that has a full-scale current of 10 milliamps.

The system measures the period of pulses from the engine alternator tachometer output line. A calibration constant can be set as necessary to take care of meters with different currents, different engine to alternator pully ratios, and the number of pulses per revolution put out by the alternator.

Each time the hours counter increments (every 1/100 hour) the new time is stored in the EEPROM for recovery on power-up. It can take up to 12 milliseconds to write the three bytes required, and if power were to be interrupted during a write the EEPROM could become corrupted. To avoid this unlikely but possible senario, the micro senses power and does not write the EEPROM if power is failing. A capacitor holds enough charge to finish a write if power fails after a write cycle has started. To maximize the time the micro will run on capacitor power, the meter and led display are turned off when a power failure is detected.

The meter is driven with the timer1 PWM of the AT90S4433 (readers duplicating this project might use the AVR MEGA8 which is suggested as a replacement for the older but still very capable 4433.) Timer0 is used to measure the period between alternator pulses. Measuring frequency would be more straightforward, but in order to get sufficient resolution a time base of a second or more would be required, leading to slow response times when RPM is changed.

The 4433 analog comparator is used along with the internal band-gap voltage reference to sense system voltage. If the input voltage falls below 9.5 volts writes to the EEPROM are disabled to avoid possible EEPROM corruption.

The 20 milliamp drive capability of the 4433 I/O ports is put to good use driving the LED segments, which take about 13 milliamps each - no buffer is required. The UART functionality of the 4433 is used in the implementation of a LIN bus interface, so the tach values could be monitored remotely if desired. It would also be possible to switch the

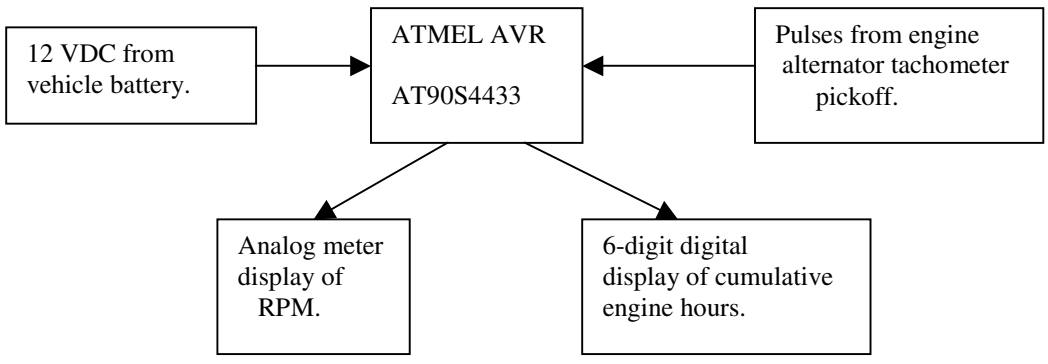
display to show oil pressure, water temperature, or other parameters, with the values coming to the tach via LIN.

The breadboard was built using the analog meter and housing from an old, defunct VDO tachometer. Any analog meter with a full-scale current of 1 to 50 mA could be used.

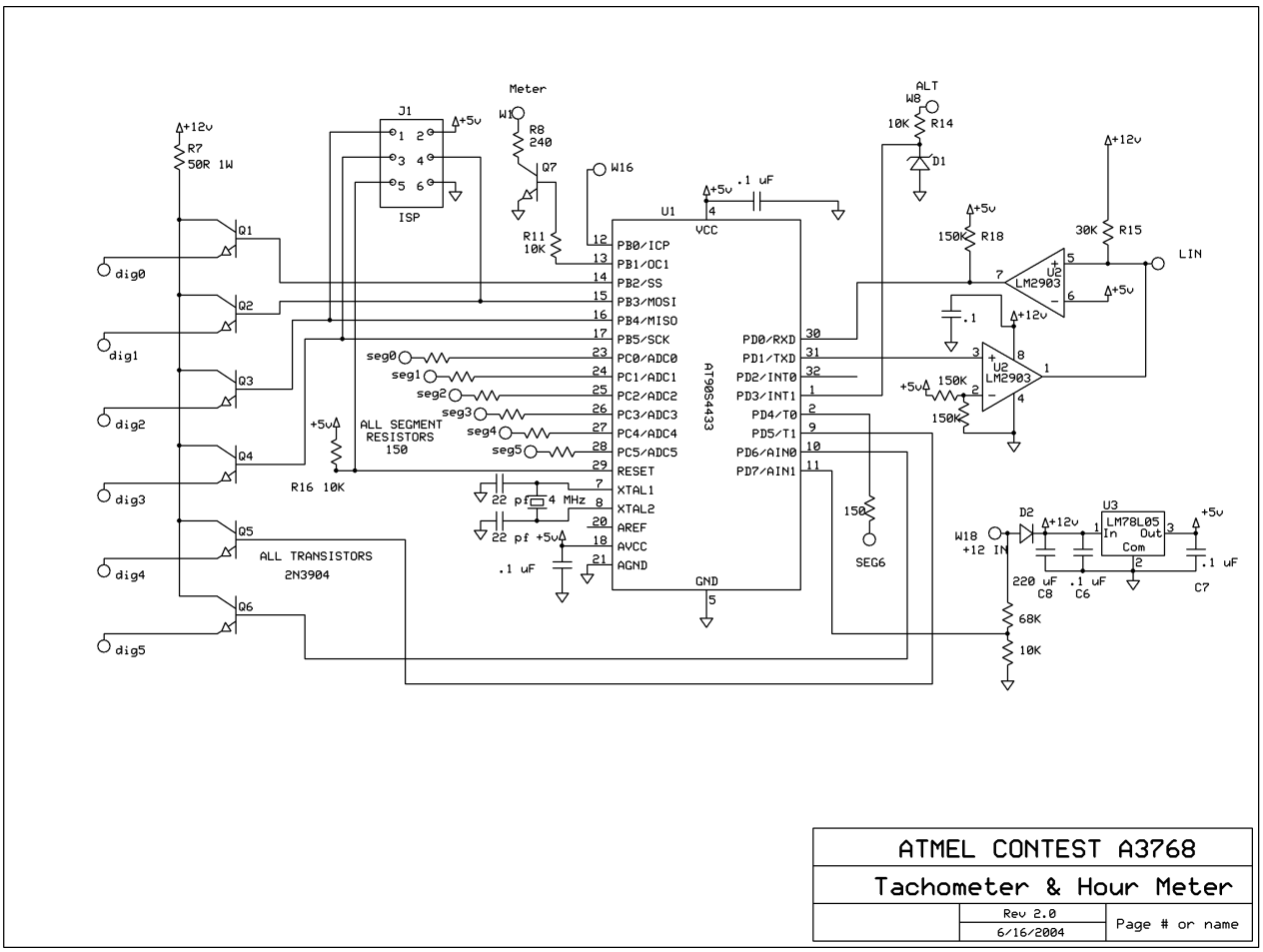
BREADBOARD METER

I was lucky to find 7-segment LED displays of exactly the right size to fit the holes where the VDO rotary mechanical indicator wheels were.

BLOCK DIAGRAM



SCHEMATIC



ATMEL CONTEST A3768		
Tachometer & Hour Meter		
	Rev 2.0 6/16/2004	Page # or name

CODE SAMPLE

The free demo IAR compiler was used – it is limited to 4K code, which is more than enough for this project.

```
// ATMEL/Circuit Cellar project A3768 - Tachometer and engine hour counter.

// tachISR.c external INT1 interrupt service routine (ISR).


// This interrupt service routine captures the value in the timer0 counter for
//     calculation of the time required for 20 alternator pulses to occur.

// A global variable ttg (time-to-go) is used to start and stop the counting.
// ttg is set by the main program to signal this routine to start counting.
// To avoid timing issues counting starts with the INT1 interrupt, some time
//     after ttg is set.

// As the resolution of timer0 is only 8 bits, the number of overflows of timer0
//     are accumulated. Timer0 counts at 62.5 KHz, and overflows occur every 256
//     counts, or 4.096 milliSeconds.

#include <ioavr.h>
#include <inavr.h>

extern unsigned char ttg,overflows,tachBusy; // time-to-go, the number of tach pulses
remaining
extern volatile unsigned int clockTicks;     // the number of 62.5 KHz ticks in 20 tach
pulses

#pragma vector =  INT1_vect
__interrupt void tachISR(void) {
  static volatile unsigned char startTime,endTime;

  endTime = TCNT0;                           // to minimize latency, get the timer
                                             // counter value first.
  ttg -= 1;
  if (ttg == 20) {                           // start counting when ttg reaches 20
    startTime = endTime;
    overflows = 0;
    if (startTime < 10) {                    // Fix possible race condition.
                                             // If overflow interrupt flag is set after
                                             //   this isr starts,

                                                    //the overflow count will be in error
by 1 count.
      TIFR = 2;                              // Clear overflow interrupt flag if it is set.
    }
  }
  else if (ttg == 0) {                       // Done counting when ttg reaches zero.
    tachBusy = 0;
    GIMSK = 0;                               // Disable tach interrupts. They are enabled
by
                                             //  the main program.
                                             // Calculate the number of clock ticks.
    clockTicks = overflows*256 + endTime - startTime;
  }
}
```