```
/* GPS Control Algorithm. Version 6.4 (06/02/09)

This program receives the signals from GPS, analyzes them, and issues
engine-state-decisions based on the vehicle's speed only. Next version
might attempt to account for drag losses while going down the hill and
decelerate ahead of time. It also serves as a navigation device - it
tells the driver (through LCD) instanteneous speed, average speed (over
last 6 minutes), time the system has been on for, and average time per
lap (assuming 1.6 mile lap and using 6 minute average speed).

For this software to run properly, the following must be true:

    -Etek GPS must be connected to USART1
    -LCD must be connected to USART0
    -Starter relay must be connected to PORTC pin1
    -Coil relay must be connected to PORTC pin2
    -MegaSquirt power relay must be connected to PORTC pin3
    -Starter LED (green) must be connected to PORTC pin5
    -Kill LED (red) must be connected to PORTC pin6
    -Fuel Pump controller line must be connected to PORTB pin6
    -The frequency of the chip should be set to be 8MHz (for baud rates)

If different GPS/LCD are used - change baud rates accordingly.
If optimum speeds change, go to lines 159 and 177 and change them.

Alex Morozov. 06/02/09
*/


#define F_CPU 8000000UL     //Define the clock speed: 8 MHz
#include <util/delay.h>     //delay library
#include <avr/io.h>         //basic avr input output stuff
#include <avr/interrupt.h>  //avr interrupt library
#include <math.h>


volatile int GPSscnt;       //Counter variable (holds index of the
                            //  current character in GPS string)
volatile int tcnt;          //Similar counter but for time string
volatile int scnt;          //Same counter, but for speed string
volatile int volt;          //Variable used to turn on the fuel pump
volatile double speed;      //Actual numerical value for speed (in MPH)
volatile double avgspeed;   //Cumulative average speed (in MPH)
volatile long avgscnt;      //Count how many speed data poimts received
                            // up to this moment
volatile double avglaptime; //Average lap time (in MINUTES)
volatile long time;         //How many 5Hz cycles the system has been on
volatile int door1;         //Variable used for program-flow control
volatile int door2;         //Variable used for program-flow control
volatile int door3;         //Variable used for program-flow control
volatile int door4;         //Variable used for program-flow control
volatile int door5;         //Variable used for program-flow control
volatile long event1time;   //How many 5Hz cycles the door1 has been open
volatile long event2time;   //How many 5Hz cycles the door2 has been open
volatile long event3time;   //How many 5Hz cycles the door3 has been open
volatile long event4time;   //How many 5Hz cycles the door4 has been open
volatile long event5time;   //How many 5Hz cycles the door5 has been open
volatile char GPSstr[80];   //Actual GPS sentence string
volatile char timestr[10];  //Time string
volatile char speedstr[5];  //Speed string


int main()                  //Main program
{

//;;;;;;;;;;;;;;;;;;;;;;;;;Set up GPS
```

```
 68
 69      UBRR1L=12;              //Set baud rate, 38400 bps (12, 0x0c)
 70      UBRR1H=0;               // so high bit is zero
 71      UCSR1B=0x90;            //Enable the receiver and the receive
 72                              // complete interrupt (RxCIEn)
 73
 74
 75  //;;;;;;;;;;;;;;;;;;;;;;;;;Set up LCD
 76
 77      UBRR0L=51;              //Set baud rate, 9600 bps (51, 0x33)
 78      UBRR0H=0;               // so high bit is zero
 79      UCSR0B=8;               //Enable the TX-r
 80
 81
 82  //;;;;;;;;;;;;;;;;;;;;;;;;;Set up car controller
 83
 84      DDRC =0b00110111;       //Set up pins 1, 2, and 3 in PORTC as outputs
 85                              // Starter, Coil, and MegaSquirt signals will
 86                              // go there. StarterLED is connected to the
 87                              // pin 5 and KillLED is connected to the pin
 88                              // 6.
 89
 90
 91  //;;;;;;;;;;;;;;;;;;;;;;;;;Set up PWM for Fuel Pump
 92      DDRB =0b10000000;       //Make pin 8 of port b an output (for this
 93                              //  processor 1 in the register means output
 94                              //  pin, 0 means input)
 95      OCR2=150;               //Start by outputting 5V for 13/255 of the
 96                              //  period (5%) - means just tell the FP
 97                              //  computer to turn on later should be
 98                              //  switched to 8% to actually run the fuel
 99                              //  pump at the right pressure
100      TCCR2=0b00011101;       //Put PWM into CTC mode (see pg. 150 of
101                              //  mega128 manual)
102      TIMSK=0b10000000;       //Enable Output compare interrupt
103
104
105  //;;;;;;;;;;;;;;;;;;;;;;;;;Set up Analog to Digital Converter (ADC)
106                              //Used only for testing the fuel pump.
107      ADMUX=0b01100000;
108      ADCSRA=0b11001111;
109
110
111
112      sei();                  //Enable Interrupts
113
114
115  //;;;;;;;;;;;;;;;;;;;;;;;;;Infinite Loop
116
117    for ( ; 1==1 ; ){         //The infinite loop
118
119
120  /*  for (int j=0;j<scnt;j++){//Loop for outputting the speed string to
121                              //  the LCD
122          UDR0=speedstr[j];   //Output one character of the 'speed' to the
123                              //  screen
124          _delay_ms(2);       //Wait 2 ms (that's the minimum wait time,
125      }                       //  per char)
126
127
128      UDR0=' ';               //Output a space to the LCD to separate the
129                              //  strings
130      _delay_ms(2);           //Wait 2 ms
131
132      for (int j=0;j<10;j++){ //Same LCD-output loop, but for current time
133          UDR0=timestr[j];
134          _delay_ms(2);       //Wait 2 ms
```

```
135       }
136
137       UDR0=' ';               //Separate the data with a space
138       _delay_ms(2);
139   */
140
141       DisplayDoub((double)speed);
142       DisplayDoub((double)avgspeed);
143       UDR0='M'; delay ms(2);
144       UDR0='P'; delay ms(2);
145       UDR0='H'; delay ms(2);
146       UDR0=' '; delay ms(2);
147       //DisplayLong(avgscnt);
148       //DisplayDoub((double)avglaptime);
149       DisplayTime(time);
150       DisplayAvgLapTime(avglaptime);
151
152
153       UDR0=0xFE;              //Reset LCD by sending it a 'command'
154                               //  character (254, see LCD manual)
155       _delay_ms(200);         //Wait a while, because if we reset it right
156                               //  away, the last characters will be dim
157
158
159       if (speed>12*2.2369 && door1==0){//If going too fast
160                               //(above 12 m/s->mi/hr) kill the engine
161           volt=0;                     //Turn down the fuel pump
162           PORTC=PORTC & 0b11111010;   //Turn off the MS (and just in case,
163                                       //  kill the starter)
164           PORTC=PORTC | 0b00100010;   //Kill the engine by engaging coil
165
166           door1=1;                    //Close the door behind yourself
167           event1time=time;
168       }
169
170           if (door1==1 && time>event1time+10){//wait 2 seconds (2*5=10)
171               door1=0;
172               PORTC=PORTC & 0b11011101; //Now can turn the coil relay off
173           }
174
175
176
177       if (speed<4*2.2369 && door2==0 && door3==0 && door4==0 && door5==0){
178           //If going too slow (below 4 m/s->mi/hr)
179
180           PORTC=PORTC | 0b00010100;   //Turn on the MS, and the StarterLED
181           door2=1;                    //Wait till FP computer makes its
182           event2time=time;            //  acknowledgement sound (1s)
183       }
184
185           if (door2==1 && time>event2time+5){//wait 1 second
186               door2=0;
187               volt=5;                     //Turn the FP up to the needed
188                                           //  pressure (60 psi)
189               door3=1;            //Wait till FP catches up to speed
190               event3time=time;
191           }
192
193           if (door3==1 && time>event3time+5){//wait 1 second
194               door3=0;
195               PORTC=PORTC & 0b11111101;   //Kill the coil, just in case
196               PORTC=PORTC | 0b00000001;   //Turn on the Starter
197               door4=1;
198               event4time=time;
199           }
200
201
```

```
202          if (door4==1 && time>event4time+10){//Wait till engine starts
             (2s)
203              door4=0;
204              PORTC=PORTC & 0b11101110;//Now can turn the starter (and LED)
205                                       // off
206              door5=1;
207              event5time=time;
208          }
209
210          if (door5==1 && time>event5time+25){//Give driver 5 seconds
211              door5=0;
212
213          }
214
215
216      else{                       //If between speeds, do nothing
217      }
218
219
220      UDR0=1;                     //Send the actual 'rst' character to the LCD
221      _delay_ms(4);
222    }                             //End of infinite loop
223  return 0;                       //End of main method
224  }
225
226
227
228  //;;;;;;;;;;;;;;;;;;;;;;;;;GPS interrupt handler
229
230
231  void StoreGPSchar(){            //Used in the interrupt routine below to
232                                  //  store characters in memory
233  GPSstr[GPSscnt]=UDR1;           //Save character at a specified position
234  GPSscnt++;                      //Advance character-count (its position)
235  }
236
237
238  void DisplayDoub(double displayvar){
239      if (displayvar>=10){
240          UDR0=(int)displayvar/10+0x30;
241           delay ms(2);
242          UDR0=(int)fmod(displayvar,10)+0x30;
243           delay ms(2);
244          UDR0='.';
245           delay ms(2);
246          UDR0=(int)fmod(10*displayvar,10)+0x30;
247           delay ms(2);
248          UDR0=(int)fmod(100*displayvar,10)+0x30;
249          }
250
251      else{
252          UDR0=(int)displayvar+0x30;
253          _delay_ms(2);
254          UDR0='.';
255           delay ms(2);
256          UDR0=(int)fmod(10*displayvar,10)+0x30;
257           delay ms(2);
258          UDR0=(int)fmod(100*displayvar,10)+0x30;
259          }
260
261          _delay_ms(2);
262
263          UDR0=' ';               //Separate the data with a space
264          _delay_ms(2);
265  }
266
267  void DisplayLong(long displayvar){
```

```
268
269         UDR0=(int)fmod(displayvar,10000)/1000+0x30;
270         _delay_ms(2);
271         UDR0=(int)fmod(displayvar,1000)/100+0x30;
272          delay ms(2);
273         UDR0=(int)fmod(displayvar,100)/10+0x30;
274          delay ms(2);
275         UDR0=(int)fmod(displayvar,10)+0x30;
276         _delay_ms(2);
277
278         UDR0=' ';               //Separate the data with a space
279         _delay_ms(2);
280 }
281
282 void DisplayTime(long displayvar){
283
284     if (displayvar>2999){
285         UDR0=(int)displayvar/3000+48; delay ms(2);
286         UDR0=(int)fmod(displayvar,3000)/300+48;_delay_ms(2);
287         UDR0=':'; delay ms(2);
288         UDR0=(int)fmod(displayvar,300)/50+48;_delay_ms(2);
289         UDR0=(int)fmod(displayvar,50)/5+48;_delay_ms(2);
290     }
291
292     else {
293         UDR0=(int)displayvar/300+48;_delay_ms(2);
294         UDR0=':'; delay ms(2);
295         UDR0=(int)fmod(displayvar,300)/50+48;_delay_ms(2);
296         UDR0=(int)fmod(displayvar,50)/5+48;_delay_ms(2);
297     }
298
299     UDR0=' ';
300     _delay_ms(2);
301 }
302
303 void DisplayAvgLapTime(double displayvar){
304
305         UDR0=(int)displayvar+0x30;
306         _delay_ms(2);
307         UDR0=':';
308          delay ms(2);
309         UDR0=(int)fmod(displayvar*60,60)/10+0x30;
310          delay ms(2);
311         UDR0=(int)fmod(displayvar*60,10)+0x30;
312         _delay_ms(2);
313
314         UDR0=' '; delay ms(2);
315         UDR0='m'; delay ms(2);
316         UDR0=':'; delay ms(2);
317         UDR0='s'; delay ms(2);
318         UDR0='e'; delay ms(2);
319         UDR0='c';_delay_ms(2);
320 }
321
322
323 SIGNAL(USART1_RX_vect)
324 {
325     if (UDR1=='$'){             //See if received character (char) is a
326                                 // dollar sign (stands for the beginning
327                                 // of the GPS sentence (NMEA 0183)).
328         GPSscnt=0;              //If it was, start recoding the sentence
329                                 //  from scratch
330         if (GPSstr[4]=='R'){    //We just want to look for GPRMC sentence
331                                 //  (see NMEA 0183 manual or tutorial -
332                                 //  GPRMC gives velocity, position,
333                                 //  heading - all the parameters we need.
334
```

```
335                             ///Speed Extraction
336
337             int i=46;           //Speed characters start on the 46th
338                                 //  character (fixed) in GPRMC sentence
339                                 //  and go till the next comma (not fixed
340             int j=0;            //Initialize the character counter for
341                                 //  speed counter
342
343             while(GPSstr[i]!=','){  //Extract the speed string from the
344                 speedstr[j]=GPSstr[i]; //  data
345                 i++;
346                 j++;
347             }
348
349
350             scnt=strlen(speedstr); //Take note of the piece's length
351
352
353 /*          speed=0;
354
355             int k=0;            //Convert speed string to a number (int)
356             while(k!=scnt-3){
357
358
359                 speed=speed+(speedstr[k]-0x30)*pow(10,((scnt-3)-k-1));
360                 k++;
361                 }
362
363             int l=scnt-2;           //Convert speed string to a number
                (int)
364             while(l!=scnt){
365                 speed=speed+(speedstr[l]-0x30)*pow(10,-l+2);
366                 l++;
367                 }
368 */
369             time++;
370
371             if (avgscnt<1800){
372             avgscnt++;
373             }
374
375             avgspeed=((avgscnt-1)*avgspeed+speed)/avgscnt;
376             avglaptime=1.6*60/avgspeed;
377
378
379 /*                          ///Time (GMT) extraction
380             i=8;                //The time characters take on positions
381                                 //  from 8 till comma
382             j=0;
383             while(GPSstr[i]!=','){
384                 timestr[j]=GPSstr[i];
385                 i++;
386                 j++;
387             }
388             tcnt=strlen(timestr); //Take note of the piece's length
389 */
390         }
391     }
392     StoreGPSchar();             //Finally store the character (see above)
393 }
394
395
396
397 //;;;;;;;;;;;;;;;;;;;;;;;;;;Fuel Pump PWM interrupt handler
398
399 SIGNAL(SIG_OUTPUT_COMPARE2){ //The fuel pump controller has to initially
400                             //  send a low-duty-cycle-signal to turn the
```

```
401                                 //  pump computer on and only then engage the
402                                 //  pump itself. That's what is done using
403                                 //  the "volt" variable here.
404       if (OCR2==(7+volt)){      //PWM basics: if the high part of the signal
405       OCR2=150;                 //  is already over, turn to the (longer) low
406       }                         //  part of the signal
407
408       else if (OCR2==150){      //  if the low part is over, switch back to
409       OCR2=7+volt;              //  high
410       }
411
412       else {                    //  if not sure where to start, start low
413       OCR2=150;
414       }
415  }
416
417
418  SIGNAL(SIG ADC){
419  speed=(double)ADCH/9;
420  ADCSRA=0b11011111;
421  }
422
```