```c
/*
 * File:   PDOS-main.c PlasmaDriveOperatingSystem
 * Author: jbs
 * Single Lithium Ion Cell/Supercap Train Controller with Flyback Transformer - Operating System
 * Created on 3 Oct 2020, latest update September 2021
 * Program for model-train speed controller regulating energy using PIC16F1829 & GaN FET
 *
 * In AUTO mode, the PB can start an auto-run (trigger) or stop it (trainArrived).
 * The auto-link input can halt with or without reversal according to R to ground;
 * run starts with the signal going to RED and the train accelerating away, then after a delay
 * the signal turns to YELLOW, and the processor looks for an input to be pulled low.
 * When PB pulled low, the train stops, the signal goes green;
 * after a delay the processor starts looking again for a trigger.
 * The signal dims with low battery.
 * Very low battery causes slow red flashes on the signal and no running
 * The signal goes dark after ~1 hour of inactivity (timeout), or if battery too low.
 * If the YELLOW-search condition continues for >25s, signal goes RED-YELLOW flashing to show "lost" train
 * and resumes search for trigger to try and recover train.
 * Thus automation is signalled fully on train 3-light signal.
 *
 * In MANUAL mode, the pot controls drive. The signal sits at RED until the pot is adjusted away from zero, at
 * which point the signal goes GREEN. YELLOW is reserved for fault indication.
 * The signal goes dark (and loco stops) after ~1 hour of inactivity, or if battery too low.
 *
 * The mode is set at boot by searching for a short to ground on the link, signifying MANUAL.
 * Serial data is sent to optional 2x16 character screen, powered up only when active.
 *
 * MORSE LED signals key statistics and errors using Morse code.
 * Has a power-dump function to protect battery/supercap from overvoltage (from solar panels?)
 * Uses 2-phase gate drive to SMPS GaN FET driving flyback converter at 30kHz on 8ms control cycle.
 *
 */

#include <xc.h>

// PIC16F1829 Configuration Bit Settings; CONFIG1
#pragma config MCLRE = OFF                  // MCLR Pin Function Select (MCLR/VPP pin function is digital input)
#pragma config CLKOUTEN = OFF
#pragma config WDTE = OFF
#pragma config PWRTE = ON
#pragma config CP = OFF
```

```
42 #pragma config BOREN = ON                    // Want to go into hibernation/reset if Li-Ion voltage low (BMS shutdown)
43 #pragma config FCMEN = ON
44 #pragma config CPD = OFF
45 #pragma config IESO = OFF
46 #pragma config FOSC = INTOSC                 // use internal clock (can be changed on the fly)
47 #pragma config BORV = HI
48 #pragma config LVP = OFF                     // NOTE: This is needed to release MCLRE!
49
50 #include <stdlib.h>
51 #include <string.h>
52 #include <stdio.h>
53
54 // COMPILE-TIME # defines for your layout configuration
55 #define LM285V  2499L                 // long int mV of actual LM285 voltage
56 #define VLOW    3400                  // warn supply low (3500 for LiNMC)
57 #define VHIGH   5500                  // V to start dumping power (4300 for LiNMC/LiNCA)
58 #define VRESUME 3600                  // out of low battery mode (to return to running)
59 #define STARTCLRINTERVAL 60           // tenths of seconds to clear sensor after start
60 #define STATIONINTERVAL 150           // tenths of seconds in station to halt & reverse
61 #define HALTINTERVAL 110              // tenths of seconds to halt in station
62 #define STATIONCLRINTERVAL 120        // tenths of seconds to clear sensor in reverse
63 #define HALTCLRINTERVAL 100           // tenths of seconds to clear halt
64 #define AUTORUNVOLTAGE 3700           // mV of supply to justify autoruns (6000 deactivates)
65 #define KIp 6                         // Ki is 2^(-KIp)
66
67 // NB: Morse spacings different for visual from sonic
68 #define LENGTHDOT 80                  // milliseconds for MORSE dispay
69 #define GAPDOTS 3                     // multiple of LENGTHDOT between symbols (1?)
70 #define DASHDOTS 4                    // multiple of LENGTHDOT in a DASH (3?)
71 #define LETTERDOTS 10                 // multiple of LENGTHDOT between letters (3?)
72 #define WORDDOTS 19                   // multiple of LENGTHDOT between words (7?)
73
74 #define VERSION 3                     // software release/version
75
76 // compiler #defines
77 #define LON 0
78 #define LOFF 1
79 #define MLON 0
80 #define MLOFF 1
81 #define DON 0
82 #define DOFF 1
```

```
#define TRUE    1
#define FALSE   0
#define NOPS    asm("NOP;");asm("NOP;");
#define PR2S 127
#define MAX(A,B) (((A)>(B))?(A):(B))
#define MIN(A,B) (((A)<(B))?(A):(B))


// Hardware defines
                                        // Pin 1 - Vdd (Lithium battery or 5.4V or 5.5 supercap, choice selected at compile)
#define PWR LATA5                       // Pin 2 - RA5 switched power to Vref, etc.
#define SELECTPOT ADCON0=0b00001101;NOPS;   // Pin 3 - AN3/RA4 pot input
#define PB RA3                          // Pin 4 - RA3, Push Button (active low)
#define PWM RC5                         // Pin 5 - RC5/P1A drive to MOSFET (not used manually)
#define nPWM LATC4                      // Pin 6 - RC4 ground clamp on MOSFET gate
#define SELECTVM   ADCON0=0b00011101;NOPS;  // Pin 7 - RC3/AN7, motor voltage if no FET drive
#define SELECTVREF ADCON0=0b00100001;NOPS;  // Pin 8 - RC6/AN8 Vref
#define DISPPWR RC7                     // Pin 9 - RC7 power to 2x16 display (active low)
#define TX LATC0                        // Pin10 - (RB7) TX to 2x16 display (not used manually)
#define DUMP RB6                        // Pin11 - RB6 Power DUMP
#define REV RB5                         // Pin12 - (RB5) REV (when reverse relay fitted)
#define GRNLED LATB4                    // Pin13 - RB4 GReeN signal LED
#define YELLED LATC2                    // Pin14 - RC2 YELlow signal LED
#define REDLED LATC1                    // Pin15 - RC1 RED signal LED
#define SELECTIFET ADCON0=0b00010001;NOPS;  // Pin16 - RC0/AN4 current sense on MOSFET
#define SELECTAUTO ADCON0=0b00001001;NOPS;  // Pin17 - RA2/AN2 MANUAL-AUTO/HALT-REVERSE
#define MORSE LATA1                     // Pin18 - RA1 (ICSP) Morse code LED
#define ISRLED LATA0                    // Pin19 - RA0 (ICSP) ISR LED
                                        // Pin 20 - Ground
#define PUSHED  0                       // state on PB when finger pushed down

// states available to the main state machine when in both AUTO and MANUAL modes
#define AWAITING 0
#define MOVING 1
#define MOVESCAN 2
#define LOWBATT 3
#define LOST 4
#define STOPPING 5
#define HOMED 6
#define MANUAL 7
#define PAUSING 8
```

```
124 #define REVERSING 9
125 #define RESUMING 10
126 // signal condition
127 #define RED 1
128 #define YEL 2
129 #define GRN 3
130 #define DARK 4
131 // LCD display
132 #define DSPRED 91
133 #define DSPYEL 92
134 #define DSPGRN 93
135 #define DSPAQUA 94
136 #define DSPWHITE 95
137 #define DSPCREAM 96
138
139 #define M_RUNS 1
140 #define M_VS 2
141 #define M_SHORT 3
142 #define M_LOST 4
143 #define M_OPEN 5
144
145 // state machine controls
146 char state=AWAITING;                                    // main state machine state
147 char signal=RED;                                        // the desired signal colour display
148
149 #define DISPLAYING ((!TXIE)&&(dspdly==0)&&(DISPPWR==DON))
150 #define MSGLEN 40                                  // serial outgoing message
151 char msg[MSGLEN+5];                                // allow for CR, LF and NULL
152 unsigned char mptr;                                // pointer into message
153
154 // subroutines
155 unsigned char NVReadByte(unsigned char);
156 void NVWriteByte(unsigned char, unsigned char);
157 void NVReadLong(unsigned char, unsigned long int *);
158 void NVWriteLong(unsigned char, unsigned long int *);
159 int median(int, int, int);
160 void writeDisplay(char *);
161 void setLCDcolour(unsigned char);
162
163 // global variables
164 short signed int error;                // proportional error
```

```
long signed int ierr;                   // integral error
short signed int Vs_mV;                 // Cell/SC supply voltage in mV
short signed int mthrust;               // PWM drive variable, average value
short signed int maxthrust,minthrust;   // PWM monitors
short signed int Vref=700;              // adc value of Vref pin (never written in main)
short signed int Vlink;                 // adc reading from LINK input (manual link, locoDetect, etc)
short signed int pot;                   // adc value of pot position
unsigned char dspdly;                   // counter for LCD display timeouts, 10ms multiples
unsigned short int dly1ms;              // counter for main state machine timeouts, 1ms multiples
unsigned char dly100ms;                 // counter for main state machine timeouts, 100ms multiples
unsigned char dly1s=0;                  // counter for main state machine timeouts, 1000ms multiples
unsigned char clearingInterval=50;      // period to spend clearing RED signal on start of run, 100ms multiples
unsigned int timeToRun=30;              // minutes delay counter for autostarts
short signed int speed=0;               // magnitude of speed from instantaneous back EMF
short signed int minspeed,maxspeed;     // min/max speed for analysis & display
unsigned char incOnMins=0;              // alive time yet to be counted
unsigned char incRunMins=0;             // rolling time yet to be counted
unsigned char tmomins;                  // timeout counter (signal & movement)
signed char displayState=0;             // Manual state machine for LCD display
char morseMessage=0;                    // message# to be sent in Morse code
signed short int Ifet,mIfet;            // driver current in mA, and its average
short signed int lastPotADC;            // previous value from ADC read, to detect change
unsigned char adjKp;                    // adjusted 8*(feedback gain), corrected for Vs
signed short int adjPWM=400;            // max thrust value, Vs-adjusted
char morseMsg[16];                      // text for Morse transmission
signed short int ticksPerStep;          // ABS braking variable
signed short int autorunPeriod=3;       // period in minutes to next autorun


bit AutoMode;               // AUTO or MANUAL mode?
bit autoOn;                 // thrust permitted?
bit lowBattery;             // flag saying flat soon, no runs
bit reboot;                 // zero in normal running, 1 to reboot
bit incRuns;                // flag to request run count increment
bit trigger;                // run request
bit halt;                   // train hit detector asking for halt
bit haltRev;                // train hit detector asking for reverse
bit incAutoRuns;            // request to increment record of #runs
bit sendingMorse;           // is a message in progress?
bit shunt;                  // in shunt vs cruise mode?
bit autodirection;          // sets/inverts direction of travel in auto
```

```
206 bit potTouched;             // flags adjustment made
207 bit saveABS;                // request ABS is written to NVRAM
208 bit saveKP;                 // request KP is written to NVRAM
209 bit pdump;                  // flag to say power was dumped (sticky)
210 bit openCircuit;            // OC flag
211 bit stickyOpen;             // sticky OC flag
212 bit shortCircuit;           // SC flag
213 bit stuck;                  // loco not moving despite thrust
214 bit stickyStuck;            // sticky "stuck loco" flag
215
216 //EEPROM initial constants
217 __EEPROM_DATA(0x01,0x00,0x00,0x00, 0x01,0x00,0x00,0x00);    // long #ONMINS, long #RUNMINS
218 __EEPROM_DATA(0x00,0x00,0x00,0x00, 0x00,0x00,0x00,0x00);    // long #autoruns, long #BOOTS
219 __EEPROM_DATA('c','J','B','S','2','0','2','1');             // copyright sig
220 __EEPROM_DATA('P','D','O','S','3','.','0','0');             // version
221 #define ONMINS  0                                           // where the age is stored
222 #define RUNMINS 4                                           // running time is here in EEPROM
223 #define AUTORUNS    8                                       // #automatic runs is here
224 #define BOOTS   12                                          // #reboots is here
225 #define ABS 128                                             // braking constant
226 #define KP  132                                             // P gain constant
227 // coding of Morse: up to 7 symbols/char supported, 1=dash, 0=dot, leading 1 signals start of symbols
228 __EEPROM_DATA(0b01111111,0,0,0, 0,0,0,0);                                  // 32,33,34,35,36,37,38,39 (32==' ' seven-dots of silence)
229 __EEPROM_DATA(0,0,0,0, 0b01110011,0b01100001,0b01010101,0);                // 40,41,42,43,44=',',45='-',46='.',47
230 __EEPROM_DATA(0b00111111,0b00101111,0b00100111,0b00100011, 0b00100001,0b00100000,0b00110000,0b00111000); // 48=='0',49,50,51,52,53,54,55=='7'
231 __EEPROM_DATA(0b00111100,0b00111110,0b01111000,0, 0,0,0,0b01001100);       // 56,57,58==':',59,60,61,62,63='?'
232 __EEPROM_DATA(0b01011010,0b00000101,0b00011000,0b00011010,0b00001100,0b00000010,0b00010010,0b00001110); // 64='@',65='A',66,67,68,69,70,71
233 __EEPROM_DATA(0b00010000,0b00000100,0b00010111,0b00001101, 0b00010100,0b00000111,0b00000110,0b00001111); // 'H','I','J','K','L','M','N','O'
234 __EEPROM_DATA(0b00010110,0b00011101,0b00001010,0b00001000, 0b00000011,0b00001001,0b00010001,0b00001011); // 'P','Q','R','S', 'T','U','V','W'
235 __EEPROM_DATA(0b00011001,0b00011011,0b00011100,0, 0,0,0,0);                // 'X','Y','Z',91,92,93,94,95
236 __EEPROM_DATA(0b00000000,0b00000101,0b00011000,0b00011010,0b00001100,0b00000010,0b00010010,0b00001110); // 96,97='a',...
237 __EEPROM_DATA(0b00010000,0b00000100,0b00010111,0b00001101, 0b00010100,0b00000111,0b00000110,0b00001111); // 'h','i','j','k','l','m','n','o'
238 __EEPROM_DATA(0b00010110,0b00011101,0b00001010,0b00001000, 0b00000011,0b00001001,0b00010001,0b00001011); // 'p','p','p','s', 't','u','v','w'
239 __EEPROM_DATA(0b00011001,0b00011011,0b00011100,0, 0,0,0,0);                // 'x','y','z' ...,127
240 __EEPROM_DATA(0,0,2,0, 32,0,0,0);                                 // ABS braking const 128-131, KP in 132
241 // almost half EEPROM unused (133--255) __EEPROM_DATA(0,0,0,0, 0,0,0,0);
242
243 void interrupt isr(void){                       // called every millisecond
244     signed short int tickCtr;                   // ABS counter variable
245     static unsigned char cnt100ms;              // counter to allow 100ms ticks
246     static unsigned int cntmins,cntThrustMins;  // counters to register minutes
```

```
247     short signed int adcval;                    // value from ADC read
248     static short signed int prev0, prev1, prev2; // past values for median filter
249     signed short int tempot;                    // temporary for processing knob dead zone
250     static signed short int thrust,ithrust;     // PWM push, and integral contribution
251     static char cycle;                          // count 8 1ms cycles -> 125Hz control rate
252     short signed int targetSpeed;               // user directive speed, set by pot
253     static signed short int inertialSpeed;      // target speed, after allowing for inertia
254     static unsigned char signalDuty;            // duty cycle for signal LEDs that fade ON and OFF
255     static unsigned char colour;                // signal colour
256     static unsigned char signalCount;           // counter for signal LED PWM
257     static short unsigned int pbcntr=0;         // debounce PB, time the press
258     static char s1ctr;                          // count seconds;
259     static char haltCtr, revCtr;                // link debounce counters
260     static unsigned char overCurrent=0;         // sc/overcurrent counter
261     static int stuckCntr=0;                     // stuck start counter
262     static unsigned char openCntr=0;            // O/C event counter
263     static unsigned short int safetyShutdown;   // >0 if current fault
264 
265     static short int ton, toff;                 // Morse symbol timing
266     static signed char morseMsgPtr=-1;          // pointer into string morseMsg
267     static char letter;                         // plain Morse char to be sent
268     static unsigned char morseletter;           // encoded Morse char to be sent
269     static signed char symbolctr;               // pointer to symbols in letter
270 
271 
272     // ---------- deal with the 2x16 display ---------------
273     if(TXIE && TXIF){                                   // TX enabled, TXREG empty
274         if(msg[mptr]==NULL||mptr>MSGLEN){               // end of msg
275             mptr=0;                                     // pointer to start
276             TXIE=0;                                     // disable EUSART interrupts
277             return;
278         }
279         TXREG=msg[mptr++];                              // load next char to Tx (clear TXIF)
280         // do not return from here; allow next instructions for TXIF to clear
281         if(tmomins==0){                                 // timed out
282             DISPPWR=DOFF;                               // turn off DISPLAY supply
283         }
284     }
285 
286     // -------------------- CHRON interrupts ---------------------------
287     if(T0IF){                                           // chron interrupt, 1.024ms
```

```
288         T0IF=0;                                 // clear flat at top, in case of long ISR
289         if(cycle!=0 && CCPR1L)ISRLED=LON;       // diagnostic/sync heartbeat LED if thrusting
290         // ------------ read supply voltage & keep time delays ------------
291         SELECTVREF;                             // redundant?
292         if(dly1ms){dly1ms-=1;}                  // decrement 1ms delay counter
293         // timekeeping in the cracks
294         if(++cnt100ms>98){                      // 100ms has elapsed
295             cnt100ms=0;                         // reset 100ms delay counter
296             if(dspdly){dspdly-=1;}              // decrement 0.1s delay counter
297             if(dly100ms){dly100ms-=1;}          // decrement 0.1s delay counter
298             if(++s1ctr>9){                      // 10*100ms -> 1s
299                 s1ctr=0;                        // 1 second passed
300                 if(dly1s){dly1s-=1;}            // decrement 1s delay counter
301             }
302         }
303         GO_nDONE=1;                             // measure Vref to find supply voltage
304         if(++cntmins>58594){                    // 1 minute elapsed
305             cntmins=0;                          // reset counter
306             if(tmomins){tmomins--;}             // decrement minutes of activity
307             if(timeToRun)timeToRun--;           // decrement minutes to automatic voyage
308             incOnMins+=1;                       // increment minutes passed
309         }
310         if(CCPR1L && ++cntThrustMins>58594){    // 1 minute of running elapsed
311                 cntThrustMins=0;                // reset counter
312                 incRunMins+=1;                  // increment minutes running
313             }
314         while(GO_nDONE){;}                      // done measure of Vref ~2.500V
315         adcval = (ADRESL+(ADRESH<<8));          // store value
316         SELECTPOT;                              // set up to read POT next
317         if(adcval>Vref){ Vref++; }              // slew filter up
318         if(adcval<Vref){ Vref--; }              // or down
319         Vs_mV = (LM285V*1024L/Vref)+10;         // supply in mV + drop on 10R resistor
320         GO_nDONE=1;                             // start to read POT
321
322         // ---------- low battery matters ----------
323         if(Vs_mV<VLOW){                         // warning time
324             trigger=0;                          // cancel any standing trigger
325             lowBattery=TRUE;                    // flag to other processes
326         }
327         if(Vs_mV>VRESUME){                      // OK to resume operation
328             lowBattery=FALSE;                   // flag to other processes
```

```
329         }
330         // ---------- high battery matters ----------
331         if(Vs_mV>VHIGH){                                // supply too high
332             DUMP = 1;                                   // waste power
333             pdump=TRUE;                                 // advertise this, sticky flag
334         }else{                                          // when not too high
335             DUMP = 0;                                   // stop dumping
336         }
337 
338         // --------------------- read pot (0->+/-500) ------------------------
339         while(GO_nDONE){;}                               // ADC finished
340         adcval = (ADRESL+(ADRESH<<8));                   // this is the pot position
341 //      adcval=1023-adcval;                              // pot mounted backwards on PCB - comment out otherwise
342         SELECTVM;                                        // select Vbemf for next time
343         if(abs(lastPotADC-adcval)>25){                   // pot moved  ~8 degrees
344             tmomins=15;                                  // reset timeout
345             potTouched=TRUE;                             // flag a change
346             DISPPWR=DON;                                 // ensure display on
347             lastPotADC = adcval;                         // store position for next time
348         }
349         tempot = 0;                                                 // in case value is zero...
350         if(shunt){                                                  // center-off control, "SHUNTing mode"
351             if(adcval< (512-45) ){ tempot = adcval - (512-45); }// backwards, below dead zone
352             if(adcval> (512+45) ){ tempot = adcval - (512+45); }// forwards, above dead zone
353         }else{                                                      // CRUISE mode
354             if(adcval>50){tempot = (adcval-50)>>1;}             // above dead zone
355         }
356         if(pot>tempot)pot--;                // slew pot to reading - kill Noise on wires
357         if(pot<tempot)pot++;                // roughly in range 0-500 or -500 to 500
358 
359         // ------------------- read motor back EMF (speed) -------------------
360         if(cycle==0){                            // 0 is the 'off' cycle
361             ISRLED=LON;                          // ISR LED when sampling Back EMF
362             GO_nDONE=1;                          // measure input from motor
363             // prev4=prev3;                      // 5th last <- 4th last
364             // prev3=prev2;                      // 4th last <- 3rd last
365             prev2=prev1;                         // 3rd last <- 2nd last
366             prev1=prev0;                         // 2nd last <- last
367             while(GO_nDONE){;}                   // wait (TAD=1us, so 22us)
368             adcval = ADRESL+(ADRESH<<8);         // store adc reading
369             if(adcval>0)adcval-=1;               // remove dc bias from pin 7
```

```
370             prev0 = adcval;                         // store for next time
371             // speed = median5(prev0,prev1,prev2,prev3,prev4);  // median of readings
372             speed = median(prev0,prev1,prev2);  // median of recent readings
373             if(speed<minspeed)minspeed=speed;   // capture min/max
374             if(speed>maxspeed)maxspeed=speed;   // for display purpose
375             ISRLED=LOFF;                        // kill LED when BEMF meas done
376         }
377         SELECTAUTO;                             // switch to "auto" input for later
378 
379         // --------------- requested Speed ------------
380         targetSpeed=pot;                                    // targetSpeed is pot speed
381         if(autodirection){targetSpeed=-pot;}                // in reverse, pot>=0 in auto
382         if(autoOn==FALSE){targetSpeed=0;}                   // overall kill function
383         if(reboot){targetSpeed=0;}                          // NO PWM before boot DO NOT REMOVE!
384         if(lowBattery){targetSpeed=0;}                      // no run if going flat
385         if(safetyShutdown){targetSpeed=0;inertialSpeed=0;}  // excess current full stop
386         // --------------- inertia ----------------      // cycle runs 0->7
387         if(++tickCtr>ticksPerStep){                         // braking rate, set by ticksPerStep
388             tickCtr=0;                                      // skipped enough ISRs
389             if(targetSpeed>inertialSpeed){inertialSpeed+=1;}// slew up
390             if(targetSpeed<inertialSpeed){inertialSpeed-=1;}// slew down
391         }
392 
393         // ------------ set direction relay ------------
394         if(inertialSpeed<0){REV=1;}
395         if(inertialSpeed>=0){REV=0;}
396         // ------------ set thrust, start by incrementing cycle count ------------
397         if(++cycle>7){                                      // stop for 1 in 8 cycles to measure BEMF
398             cycle=0;                                        // reset counter
399             CCPR1L=0;DC1B0=0;DC1B1=0;                       // PWM off this cycle
400         }
401         if(cycle==1){   // ---------- thrust to PWM (CONTROL LOOP) ------------
402             error = abs(inertialSpeed)-speed;               // wanted - measured
403             ierr+=error;                                    // sum integral error
404             if(shortCircuit){ierr=0;}                       // no windup on S/C
405             if(ierr<0)ierr=0;                               // no reverse integral thrust
406             if(abs(inertialSpeed)<2)ierr=0;                 // halt when demanded, no idle windup
407             ithrust = (ierr>>KIp);                          // integral sum converted to thrust units
408             if(ithrust>adjPWM){ierr=adjPWM<<KIp;}           // limit integral windup
409             thrust=0;if(pot)thrust=10;                      // start with good guess
410             thrust += (error*adjKp)>>3;                     // 9<adjKp<214, 1<Kp<26, 4-7 seems good
```

```
411             thrust += ithrust;                              // add integral component
412             if(thrust<0){ thrust=0; }                       // nothing less than zero
413             if(thrust>adjPWM){thrust=adjPWM;}               // nor greater max <4*(PR2+1)
414             if(stuck){stuck=FALSE;thrust=adjPWM;}           // 8ms of full power
415             if(thrust>maxthrust)maxthrust=thrust;           // monitor excursions...
416             if(thrust<minthrust)minthrust=thrust;           // ...for display
417             CCPR1L=(unsigned)(thrust>>2);                   // duty cycle
418             if(thrust&0x02){DC1B1=1;}else{DC1B1=0;}         // 9th bit
419             if(thrust&0x01){DC1B0=1;}else{DC1B0=0;}         // 10th bit
420             if(thrust>mthrust)mthrust+=1;                   // mthrust is slew filtered...
421             if(thrust<mthrust)mthrust-=1;                   // ...for display only
422             if(overCurrent)overCurrent--;                   // decrement I event once/cycle
423         }
424         // ----------- USE IF LOCO NEEDS MORE THAN 1ms to SETTLE  -----------
425         //      if(cycle==7){                                   // at end of each 8-cycle sequence
426         //          DC1B1=0;DC1B0=0;                            // reduce thrust pre Bemf cycle
427         //          CCPR1L=CCPR1L/2;                            // improve BEMF measurement?
428         //      }
429
430         // ---- AUTO/LINK input (SC for manual, else halt/reverse input) ----
431         GO_nDONE=1;                                         // measure AUTO input
432         while(GO_nDONE){;}                                  // wait
433         SELECTIFET;                                         // choose next analog ip
434         Vlink = (ADRESL+(ADRESH<<8));                       // store away AUTOSTOP level
435         // halt (between 10% and 50% supply) causes a station-stop;
436         // haltRev (< 10% supply V) causes a station-stop-with-reversal
437         if(AutoMode){                                       // look for automation signals
438             // if(Vlink>512){;}                             // input "idle"
439             if(Vlink<100){                                  // ground link for halt+reverse
440                 if(++revCtr>20){                            // 20ms gone by
441                     haltRev=1;                              // set flag
442                     clearingInterval=STATIONCLRINTERVAL;// x100ms, time to clear sensor
443                 }
444             }else{revCtr=0;}
445             if(Vlink>99 && Vlink<512){
446                 if(++haltCtr>20){                           // 20ms gone by
447                     halt=1;                                 // set flag
448                     clearingInterval=HALTCLRINTERVAL;   // x100ms, time to clear sensor
449                 }
450             }else{haltCtr=0;}
451         }
```

```
452
453         // ------------------ measure CURRENT (S/C check) ----------------------
454         GO_nDONE=1;                             // start ADC on Ifet
455         while(GO_nDONE){;}                      // wait till ADC done
456         adcval = (ADRESL+(ADRESH<<8));          // store away
457         SELECTVREF;                             // choose next analog ip (next ISR)
458         Ifet = ((2L*adcval) * Vs_mV)/1024L;     // current in Rs=0.5R in mA
459         if(Ifet>3200){                          // FET/coil overload (6A peak, sawtooth)
460             CCPR1L=0;DC1B0=0;DC1B1=0;           // cut PWM at once
461             mIfet=Ifet;                         // display worst
462             overCurrent+=2;                     // count high-current events
463             if(overCurrent>4){
464                 safetyShutdown=500;             // milliseconds off
465                 morseMessage=M_SHORT;           // "I" error
466             }
467         }
468         if(Ifet>mIfet)mIfet+=1;                 // mIfet is filtered...
469         if(Ifet<mIfet)mIfet-=1;                 // ...for display only
470         // code for comparator-activated shutdown here - future expansion?
471
472         // ------------------------ O/C detection -------------------------
473         if(cycle>=2 && cycle<=6){               // thrust applied, current settled
474             if(speed<3 && thrust>100){          // push but no move -> O/C
475                 openCircuit=TRUE;               // signal ->yellow
476         // ------------------------ STUCK detection -------------------------
477                 if(++stuckCntr>7143){           // more than 10*5*1000/7->10s O/C
478                     stuckCntr=0;
479                     stuck=TRUE;                 // summon pulse
480                     stickyStuck=TRUE;           // inform for display
481                 }
482         // ------------------------ O/C action -------------------------
483                 if(++openCntr>100){             // count O/C events before display
484                     openCntr=0;                 // clear for now
485                     morseMessage=M_OPEN;        // 5 is "O" for "Open"
486                     stickyOpen=TRUE;            // sticky flag for display, reset elsewhere
487                 }
488             }else{                              // not O/C
489                 openCircuit=FALSE;              // signal free
490                 if(stuckCntr>=2){stuckCntr-=2;} // negate gradually
491                 if(openCntr>=2){openCntr-=2;}   // negate gradually
492             }
```

```
493         }
494 
495         // ------------------------ shutdown counter ------------------------
496         if(safetyShutdown){                         // have an overload
497             safetyShutdown-=1;                      // head towards restart
498             shortCircuit=TRUE;                      // easy-to-check flag
499         }else{shortCircuit=FALSE;}                  // cleared after rest
500 
501         // ---------------------- deal with signal ----------------
502         if(signal!=colour){                             // moving to another colour
503             signalDuty--;                               // fade out, 255 counts is one second
504             if(signalDuty==0){colour=signal;}           // faded to dark, move to requested colour
505         }else{                                          // on colour
506             if((++signalDuty)==0){signalDuty=0xff;}     // overflowed, so limit to bright level
507             if(lowBattery && signalDuty>65){signalDuty=65;} // low supply, 2/8 max duty cycle
508         }
509         if(++signalCount>15) signalCount=0;             // duty cycle has 16 states, period 16ms, 64Hz
510         if(signalCount>(signalDuty>>4) || tmomins==0){  // determine brightness of displayed LED
511             REDLED=LOFF;GRNLED=LOFF;YELLED=LOFF;        // RC2, RC0, RA1 -> 0 all signal LEDs off
512         }else{
513             if(colour==RED) REDLED=LON;
514             if(colour==YEL) YELLED=LON;
515             if(colour==GRN) GRNLED=LON;
516         }   // do nothing if DARK
517 
518         // ------------ deal with PUSH BUTTON ------------
519         if(!AutoMode){                              // Manual mode
520             if(PB==PUSHED){                         // check PB
521                 DISPPWR=DON;                        // ensure display on
522                 tmomins=10;                         // stay awake
523                 if(++pbcntr>10000){reboot=1;}       // reboot if >10-sec push
524             }else{                                  // PB released
525                 if(pbcntr>100){                     // short push advances display
526                     displayState++;                 // advance what LCD shows
527                     morseMessage++;                 // and what Morse signals
528                 }
529                 if(pbcntr>1000){                    // >1s push retards display
530                     displayState-=2;                // back up what LCD shows
531                     if(displayState<0)displayState=0;   // right back to boot
532                 }
533                 pbcntr=0;
```

```
534             }
535         }
536         if(AutoMode){                           // in auto, PB input triggers/stops run
537             if(PB==PUSHED){                     // set trigger once per hit
538                 tmomins=60;                     // alive for 1 hr after event
539                 if(++pbcntr==20){               // hit is 20ms of closure
540                     trigger=TRUE;               // send trigger to state machine
541                     clearingInterval=STARTCLRINTERVAL;// x100ms, time to clear sensor
542                 }
543                 if(pbcntr==5000)                // 5 sec push for MORSE
544                     morseMessage++;             // increment Morse signal
545                 if(pbcntr>=15000)reboot=1;      // PB stuck down -> reboot
546             }else{
547                 pbcntr=0;                       // clear now PB dropped
548             }
549         // --------------- self-activation AUTORUNS ----------------
550             if(timeToRun==0 && Vs_mV>AUTORUNVOLTAGE){   // time & enough power...
551                 tmomins=60;                             // alive for 1 hr after event
552                 trigger=TRUE;                           // signal state machine
553                 timeToRun = autorunPeriod;              // time to next, minutes
554                 clearingInterval=STARTCLRINTERVAL;      // time to clear sensor
555             }
556         }
557 
558         ISRLED=LON; // heartbeat in absence of thrust, brightness hints Morse
559         // ------------------------ Morse -----------------------
560         if(sendingMorse && ton==0 && toff==0){      // time for new symbol
561             if(--symbolctr < 0){                    // end of symbols in this letter
562                 symbolctr=8;                        // get a new letter
563                 letter = morseMsg[morseMsgPtr++];
564                 toff=LETTERDOTS*LENGTHDOT;          // new letter, usually 3-dot space
565                 if(letter==' '){                    // inter-word spacing
566                     toff=WORDDOTS*LENGTHDOT;        // usually 7 dots
567                     letter = morseMsg[morseMsgPtr++]; // get next non-space
568                 }
569                 EEADR=letter;RD=1;morseletter=EEDATA;   // equivalent of NVReadByte
570                 if(letter=='\0' || morseletter==0){ // end of string || bad char
571                     sendingMorse=0;                 // end morse transmission
572                     morseMsgPtr=0;                  // reset string pointer
573                     symbolctr=0;                    // empty morseletter
574                     toff=1500;                      // some nothing to exit
```

```
575                 }
576             }
577             if(toff==0){            // no spacing to be done, so get symbol
578                 if(symbolctr>6){                                    // new letter
579                     while( ((morseletter>>(symbolctr--))&0x01) == 0){ // pointing at a '0'?
580                         if(symbolctr==0)break;                      // abort at 2nd-last bit
581                     }   // now pointing at 1st real symbol
582                 }
583                 if(((morseletter>>symbolctr)&0x01) == 1){   // is 1 so dah
584                     ton=DASHDOTS*LENGTHDOT;
585                     toff=GAPDOTS*LENGTHDOT;
586                 }else{                                      // else is dit
587                     ton=LENGTHDOT;
588                     toff=GAPDOTS*LENGTHDOT;
589                 }
590             }
591         }
592         if(ton){    // here we action MORSE millisecond by millisecond
593             ton-=1;
594             MORSE=MLON;
595         }else{      // toff only decrements when ton has got to 0
596             if(toff){MORSE=MLOFF;toff-=1;}
597         }
598 
599     } // end of T0IF
600 
601     ISRLED=LOFF;                                // kill blue LED when ISR done
602     return;
603 }
604 
605 
606 
607 void main(void) {
608     unsigned long ltmp, ltmp2;
609     unsigned char ctmp;
610     char stmp[16];
611     int maxpot=20;                      // to set ABS force
612     short signed int Vbemf;             // back EMF voltage in mV
613     short signed int minbemf,maxbemf;   // min & max back EMF voltages in mV
614     short signed int PmW;               // power switched through
615     short signed int signedSpeed;       // back EMF in ADC units, signed version
```

```
616     char lastMorseMessage;
617     signed char Vlinkpc;                // link level in percent
618     unsigned char Kpset;                // KP from NV memory
619     char schar;                         // SPICE-style units multiplier (m,k,etc)
620     unsigned long absdist;              // stopping distance
621     char sigmsg[8];                     // text version of RYG signal
622 
623     // INIT ----------------------------------------------------------------
624     nPWM=0;                 // clamp FET gate low on reboot
625     OSCCON=0b01111010;      // 16MHz for 4MIPS
626     INTCON=0b00100000;      // TMR0 interrupts but GIE not set yet
627     OPTION_REG=0b00000011;  // TMR0 clocking /16 for 1.024ms ISR chrons (976Hz)
628     TRISA= 0b00011100;      // RA0, RA1, & RA5 are outputs, 2,3,4 inputs
629     ANSELA=0b00010100;      // RA2 RA4 analog inputs
630     WPUA=0b00001100;        // pull up PB, AUTO/link input
631     TRISB= 0b00000000;      // RB4,6,7 outputs RB5 is EUART-RX (reverse?)
632     ANSELB=0b00000000;      // no analog inputs
633     WPUC=0x00;              // no pull up on port C
634     TRISC= 0b01001001;      // RC6/AN8(Vref), RC0/AN4(IFET) & RC3/AN7(Vm) analog
635     ANSELC=0b01001001;      // RC1,RC2,RC4,RC5,RC7 outputs, RC0/AN4,RC3/AN7,RC6/AN8 analog inputs
636     SELECTVM;               // turn ADC on
637     ADCON1=0b11010000;      // Fosc/16 for 1us TAD, right-justified
638     T2CON=0b00000100;       // TMR2 is ON, 1:1 prescale and postscale
639     PR2=PR2S;               // PR2+1 * 0.25us = Tperiod ->32kHz
640     CM1CON0=0;
641     CM2CON0=0;
642     TXSTA      = 0b00100000;    // async, 8-bit, enable, low rate
643     RCSTA      = 0b10000000;    // enable serial SPEN
644     BAUDCON    = 0b00000000;    // non-inverted TX data, 8-bit SPBRG (4th LSB BRG16=0)
645     SPBRGL     = 25;            // baud rate 9600 wants 25
646     SPBRGH     = 0;             // (207 gives 1202 16-bit mode @16MHz)
647     CCPR1L=0;               // zero duty cycle, no power until safe
648     CCP1CON=0b10001100;     // start PWM mode, half-bridge, active high, LSBs=0
649     // INIT ----------------------------------------------------------------
650 
651     reboot=1;               // not in main loop yet
652     PWR=1;                  // power up ref and pot
653     shunt=FALSE;            // start in cruise mode
654     signal=DARK;            // dark during boot
655     GIE=1;                  // enable ISR so ADC reads values
656 
```

```
657     DISPPWR=DON;                    // turn on DISPLAY supply (active low)
658
659     // boot indication allows ADC values to slew to equilibrium
660     for(ctmp=16;ctmp;ctmp--){       // one blink for each count
661         dly1ms=80;                  // 80ms ON
662         MORSE=LON;                  // MORSE (inboard) LED
663         while(dly1ms){;}
664         dly1ms=80;                  // 80ms OFF
665         MORSE=LOFF;
666         while(dly1ms){;}
667     }
668
669     // set up LCD display
670     displayState=0;         // set display pointer to initial state
671     msg[0]='|';             // setting...
672     msg[1]='/';             // no system messages (take too long)
673     msg[2]='|';             // setting...
674     msg[3]='1';             // SPLASH SCREEN ('1' = don't show)
675     msg[4]='\0';            // must terminate string
676     TXIE=TRUE;              // send to LCD
677
678     // choose mode and cycle signal to indicate AUTO/MANUAL
679     AutoMode=FALSE;                 // start with MANUAL mode...
680     if(Vlink>50){                   // link <50R to gnd (int pull-up ->30 LSB at short)
681         AutoMode=TRUE;              // so AUTO
682         shunt=FALSE;                // shunt not used in auto mode
683         ctmp=13;                    // show 13 signal cycles, confirm boot-to-auto
684         while(TXIE){;}              // ensure LCD ready
685         sprintf(msg,"%s Boot->Auto\rCruise mode v%d",(nRI)?"Cold":"Warm",VERSION);
686     }else{
687         ctmp=4;                     // only 4 signal cycles for manual
688         if(pot>65){shunt=TRUE;}     // sitting up above cruise-zero
689         while(TXIE){;}              // ensure LCD ready
690         sprintf(msg,"%c Boot->Manual\r%s mode v%d",(nRI)?'C':'W',shunt?"Shunt":"Cruise",VERSION);
691     }
692     writeDisplay(msg);          // show boot message
693
694     signal=RED;                     // start on RED
695     dly100ms=22;                    // sit 2.2 seconds
696     while(ctmp){                    // move through ctmp states...
697         if(signal==RED){
```

```
698             if(dly100ms==0){
699                 signal=YEL;         // from RED go to YELLOW
700                 dly100ms=12;
701                 ctmp--;
702             }
703         }
704         if(signal==YEL){
705             if(dly100ms==0){
706                 signal=GRN;         // from YELLOW go to GREEN
707                 dly100ms=12;
708                 ctmp--;
709             }
710         }
711         if(signal==GRN){
712             if(dly100ms==0){
713                 signal=RED;         // from GREEN go to RED
714                 dly100ms=12;
715                 ctmp--;
716             }
717         }
718     }
719
720     // increment the number of boots recorded in NVRAM
721     NVReadLong(BOOTS,&ltmp);            // read # boots
722     ltmp+=1;                            // increment
723     NVWriteLong(BOOTS,&ltmp);           // write
724     // load  distance
725     NVReadLong(ABS,&absdist);           // read stopping distance
726     Kpset = NVReadByte(KP);             // read in loop gain constant KP
727
728     if(AutoMode){state=MOVING; dly100ms=60;tmomins=30;}     // start as if triggered
729     else{state=MANUAL;tmomins=10;potTouched=FALSE;}         // only 1 state!
730     strcpy(morseMsg,"Boot");
731     sendingMorse=TRUE;                  // initiate sending message
732
733     // ************************** main loop ******************************
734     reboot=0;                           // setting this does warm reboot
735     while(!reboot){
736
737         // ----------------- routine non-urgent calculations ---------------
738         // dynamic adjustment of gain and max PWM thrust allows for low Vs
```

```
739         if(dly1ms==0){                          // periodic update due
740             dly1ms=1500;                        // only do every so often
741             // adjust max PWM, ABS braking, Kp and Ki
742             ltmp = 100L*Vs_mV/4500L;            // supply variation in %
743             adjKp = (Kpset*100)/ltmp;           // Kp down as supply up
744             adjPWM = 295 + 577500L/Vs_mV;       // max PWM level (400 to...
745             if(adjPWM>480)adjPWM=480;           // overload safety max
746             if(AutoMode){maxpot=MAX(25,pot);}   // 25 gives slow response
747             else{maxpot=MAX(99,pot);}           // in manual, respond faster
748             ticksPerStep = (short)(absdist/(maxpot*maxpot)+1);  // rate divider for ABS
749             // find interval between auto triggered runs (RARELY at VLOW and OFTEN at VHIGH)
750             autorunPeriod = (VHIGH-(unsigned)(Vs_mV));  // mV below max, 0->2000
751             autorunPeriod /= 2;                         // minutes before next run
752             if(autorunPeriod<30)autorunPeriod=30;       // 1/2 hour to < 1 day
753         }
754 
755         Vbemf = (((long)Vs_mV)*speed)/361L;     // ratio 0.353, mV
756         ltmp = (((unsigned long)mIfet)*Vs_mV);  // Power in uW
757         PmW = (short)(ltmp>>10);                // P approx mW
758         Vlinkpc = (unsigned char)(Vlink/11);    // auto link input as percentage
759         signedSpeed = REV?(-speed):speed;       // signed speed
760 
761 
762         // -------------- MORSE readout ----------------
763         if(lastMorseMessage!=morseMessage           // new message to send
764                 && sendingMorse==FALSE){            // & not still sending
765             switch(morseMessage){                   // choose msg
766                 default:
767                 case M_RUNS:
768                 NVReadLong(AUTORUNS,&ltmp);         // get autoruns
769                 sprintf(morseMsg,"Runs %ld",ltmp);
770                 break;
771                 case M_VS:                          // supply voltage
772                 sprintf(morseMsg,"Vs %d",Vs_mV);
773                 break;
774                 case M_SHORT:                           // machine requested
775                 strcpy(morseMsg,"I");               // S/C current (dash dash)
776                 break;
777                 case M_LOST:                        // machine requested
778                 sprintf(morseMsg,"LOST");           // auto has lost loco!
779                 break;
```

```
780                 case M_OPEN:                                // machine requested
781                     sprintf(morseMsg,"O");              // O/C (dash dash dash)
782                     break;
783                 }
784                 if(morseMessage>2)morseMessage=0;   // clear if beyond REQUESTS
785                 lastMorseMessage = morseMessage;    // remember so we detect changes
786                 sendingMorse=TRUE;                  // initiate sending message
787             }
788 
789 
790             // -------------- EEPROM update (messes with ISR) ----------------
791             if(state==AWAITING ||                       // free to fool with EERAM? (no PWM)
792                     (state==MANUAL && abs(pot)<2 && CCPR1L==0 && displayState)){ // PWM off, idle, past boot
793                 if(incOnMins){                          // on some # minutes
794                     NVReadLong(ONMINS,&ltmp);           // elapsed ON time
795                     ltmp+=incOnMins;                    // increment time spent up
796                     NVWriteLong(ONMINS,&ltmp);
797                     incOnMins=0;                        // reset request flag
798                 }
799                 if(incRunMins){                         // running some # minutes
800                     NVReadLong(RUNMINS,&ltmp);          // elapsed RUN time
801                     ltmp+=incRunMins;                   // increment time spent up
802                     NVWriteLong(RUNMINS,&ltmp);
803                     incRunMins=0;                       // reset request flag
804                 }
805                 if(saveABS){
806                     NVWriteLong(ABS,&absdist);          // update changed stopping distance
807                     saveABS=0;                          // clear request to update
808                 }
809                 if(saveKP){
810                     NVWriteByte(KP,Kpset);              // update changed K Proportionality
811                     saveKP=0;                           // clear request to update
812                 }
813             }
814 
815             // ---------- MAIN STATE MACHINE -----------------
816             // ------------ low battery -----------------------------------
817             if(state==LOWBATT){                             // low battery means reduce power
818                 autoOn=FALSE;                               // set speed to 0
819                 trigger=0;                                  // cancel any trigger
820                 DISPPWR=DOFF;                               // turn LCD display off
```

```
821             if(lowBattery==FALSE){                      // no longer low...
822                 if(AutoMode) state=AWAITING;            // return to default
823                 else state=MANUAL;
824             }
825             if(dly100ms==0){                            // every second or so
826                 dly100ms=22;
827                 if(signal==RED){                        // flash between RED and DARK
828                     signal=DARK;
829                 }else{
830                     signal=RED;
831                 }
832             }
833         }
834 
835         // ------------ default auto operating state --------------------------
836         if(state==AWAITING){                    // stationary, awaiting trigger
837             autoOn=FALSE;                       // no speed request, so no power to loco
838             signal=GRN;                         // show Green while waiting
839             DISPPWR=DOFF;                       // turn LCD display off
840             halt=haltRev=FALSE;                 // clear past AUTO/LINK signals
841             if(trigger){                        // about to run
842                 DISPPWR=DON;                    // turn on
843                 dspdly=5;                       // 500ms to boot display
844                 state=MOVING;                   // let us get MOVING
845                 dly100ms=clearingInterval;      // allow some seconds to clear loco sensor
846                 tmomins=60;                     // signal to stay awake
847             }
848             if(lowBattery) state=LOWBATT;       // exit to respond to lowish battery
849         }
850 
851         // ----------------- start train ---------------------------------------
852         if(state==MOVING){                  // state of trundling out of station
853             setLCDcolour(DSPCREAM);         // set backlight to white
854             signal=RED;                     // show red while leaving
855             autoOn=TRUE;                    // apply power
856             incAutoRuns=TRUE;               // flag to request increment of autoruns at end of run
857             if(dly100ms==0){                // check for timeout
858                 state=MOVESCAN;             // go to approach phase
859                 trigger=FALSE;              // clear report of train detection/trigger
860                 dly100ms=250;               // allow 25 seconds to get to a station or become LOST
861             }
```

```
862             if(DISPLAYING){                 // DISPLAY on, read && idle
863                 sprintf(msg,"Spd=%3d Vs=%4d Pot=%+4d Th=%3d",signedSpeed,Vs_mV,pot,mthrust);
864                 writeDisplay(msg);
865             }
866         }
867 
868         // ---------------------- train en route -----------------------------
869         if(state==MOVESCAN){
870             autoOn=TRUE;                        // power on
871             signal=YEL;                         // show yellow
872             // one of three events "finds" train: halt, haltRev, or trigger
873             // trigger ends the run
874             if(trigger){                        // train over PB (end) sensor...
875                 dly100ms=200;                   // yellow-signal lockout delay
876                 state=STOPPING;
877             }
878             if(halt){                           // train over station halt sensor...
879                 dly100ms=HALTINTERVAL;          // duration of PAUSE
880                 state=PAUSING;
881             }
882             if(haltRev){                        // train over sensor with reverse...
883                 dly100ms=STATIONINTERVAL;       // duration of PAUSE
884                 state=REVERSING;
885             }
886             if(dly100ms==0){
887                 state=LOST;
888             }
889             if(DISPLAYING){                         // DISPLAY on && idle
890                 NVReadLong(AUTORUNS,&ltmp);         // # auto runs
891                 sprintf(msg,"%ld runs   ",ltmp);    // 9 chars
892                 NVReadLong(RUNMINS,&ltmp);          // run minutes
893                 sprintf(msg+9,"%5ld m",ltmp);       // 7 chars
894                 sprintf(msg+16,"Spd=%d Scanning",signedSpeed);
895                 if(stickyOpen){msg[30]='V';stickyOpen=FALSE;}
896                 if(shortCircuit){msg[31]='I';}
897                 if(pdump){msg[29]='P';pdump=FALSE;}
898                 msg[32]='\0';
899                 writeDisplay(msg);
900             }
901         }
902 
```

```
903         // ------------ station halt -------------------------------------
904         if(state==PAUSING){
905             autoOn=0;                                   // power off loco
906             signal=YEL;                                 // show yellow
907             if(dly100ms==0){                            // halt time expired
908                 dly100ms=clearingInterval;              // time to start & leave
909                 state=RESUMING;
910             }
911             if(DISPLAYING){                             // DISPLAY on && idle
912                 sprintf(msg,"Vlink=%2d%% (%d)\rStation halt",Vlinkpc,dly100ms/10);
913                 writeDisplay(msg);
914             }
915         }
916 
917         // ------------ station resume in reverse ------------------------------
918         if(state==REVERSING){
919             autoOn=0;                                   // power off loco
920             signal=GRN;                                 // show green
921             if(dly100ms==0){
922                 if(autodirection){autodirection=FALSE;} // change direction
923                 else{autodirection=TRUE;}
924                 dly100ms=clearingInterval;              // time till clear of sensor
925                 state=RESUMING;
926             }
927             if(DISPLAYING){                             // DISPLAY on && idle
928                 sprintf(msg,"Vlink=%2d%% (%d)\rStop & reverse",Vlinkpc,dly100ms/10);
929                 writeDisplay(msg);
930             }
931         }
932 
933         // ------------ station resume from halt -------------------------------
934         if(state==RESUMING){
935             autoOn=1;                                   // power on loco
936             signal=GRN;                                 // show green
937             if(dly100ms==0){
938                 halt=0;                                 // clear auto event
939                 haltRev=0;                              // clear auto event
940                 dly100ms=250;                           // time till LOST
941                 state=MOVESCAN;
942             }
943             if(DISPLAYING){                             // DISPLAY on && idle
```

```
944                 sprintf(msg,"Speed=%3d (%c)\rLeaving station",speed,autodirection?'R':'F');
945                 writeDisplay(msg);
946             }
947         }
948 
949         // ------------ finished run, decelerating ----------------------------
950         if(state==STOPPING){
951             autoOn=0;                                   // power off loco
952             signal=YEL;                                 // show yellow
953             if(dly100ms==0){
954                 dly100ms=250;                           // extend non-retrigger interval
955                 state=HOMED;
956             }
957             if(DISPLAYING){                             // DISPLAY on && idle
958                 sprintf(msg,"Vb=%4dm LockoutTTAR=%4d TMO=%2d",Vs_mV,timeToRun,tmomins);
959                 writeDisplay(msg);
960             }
961         }
962 
963         // ------------ have stopped -------------------------------------
964         if(state==HOMED){
965             autoOn=0;                                   // power off loco
966             signal=GRN;
967             if(trigger){dly100ms=250;trigger=0;}        // re-trigger & clear
968             halt=0;
969             haltRev=0;
970             if(incAutoRuns==TRUE){                      // first time through this routine
971                 incAutoRuns=FALSE;                      // clear request
972                 NVReadLong(AUTORUNS, &ltmp);            // grab count
973                 ltmp++;                                 // increment
974                 NVWriteLong(AUTORUNS,&ltmp);            // store
975             }
976             if(DISPLAYING){                             // DISPLAY on && idle
977                 sprintf(msg,"Green lockout...Vb=%5d mV (%d)",Vs_mV,dly100ms/10);
978                 writeDisplay(msg);
979             }
980             if(dly100ms==0){                            // finished being halted
981                 state=AWAITING;                         // return to idle condition
982             }
983         }
984 
```

```
985         // ------------ loco lost -----------------------------------
986         if(state==LOST){                                // stay here until triggered for another try
987             autoOn=0;                                   // kill SMPS
988             if(dly100ms==0){                            // every second or so
989                 dly100ms=20;
990                 if(signal==RED){                        // flash between RED and DARK
991                     signal=YEL;
992                 }else{
993                     signal=RED;
994                 }
995             }
996             if(dly1s==0){
997                 dly1s=60;
998                 morseMessage=M_LOST;                    // "LOST" message
999             }
1000            if(trigger){                                // attempt to recover train
1001                DISPPWR=DON;                            // repower the LCD
1002                state=MOVESCAN;
1003                dly100ms=250;                           // give it 25 secs
1004                trigger=0;
1005            }
1006            if(DISPLAYING){     // DISPLAY on && idle
1007                sprintf(msg,"LOST...\rTrigger me!");    // clear & home
1008                writeDisplay(msg);
1009            }
1010        }
1011
1012        // ------------ MANUAL -------------------------------------
1013        if(state==MANUAL){
1014            if(lowBattery){
1015                setLCDcolour(DSPRED);               // set backlight to red
1016            }else{
1017                setLCDcolour(DSPAQUA);              // set backlight to aqua
1018            }
1019            autodirection=FALSE;                    // clear so no interference
1020            if(abs(pot)<2 || tmomins==0){           // knob down or timed out
1021                signal=RED;                         // stop
1022                autoOn=0;                           // power off
1023            }else{
1024                autoOn=1;                           // power on
1025                if(shortCircuit || openCircuit){
```

```
1026                    signal=YEL;                     // warning
1027                }else{
1028                    signal=GRN;                     // go
1029                }
1030            }
1031            // what would a user want to know?
1032            if(DISPLAYING){                         // DISPLAY on, viewed, && idle
1033                switch(displayState){
1034                    case 0:
1035                        sprintf(msg,"%c Boot->Manual\r%s KP=%d",(nRI)?'C':'W',shunt?"Shunt":"Cruise",Kpset);
1036                        break;
1037                    default:
1038                    case 1: // classic dash, speedo, fuel gauge, warning lights
1039                        strcpy(sigmsg," ");
1040                        if(signal==RED)strcpy(sigmsg,"Red");
1041                        if(signal==YEL)strcpy(sigmsg,"Yellow");
1042                        if(signal==GRN)strcpy(sigmsg,"Green");
1043                        sprintf(msg,"Speed=%4d %c%c%c%c Vs=%dmV %s",signedSpeed,pdump?'P':' ', stickyOpen?'V':' ', shortCircuit?'I':' ', stick
1044                        stickyOpen=FALSE;           // clear the sticky flag after use
1045                        stickyStuck=FALSE;          // clear the sticky flag after use
1046                        pdump=FALSE;    // this sticky flag is set if ISR dumps power
1047                        break;
1048                    case 2: // THRUST dashboard: speed, Vs (gas tank), pot (accelerator), PWM thrust
1049                        sprintf(msg,"Spd=%+4d Vs=%4d",signedSpeed,Vs_mV);
1050                        sprintf(msg+16,"Thr=%+4d Pwr=%3d",pot,mthrust);
1051                        break;
1052                    case 3: // control system readout, set, current, P & I contributions
1053                        sprintf(msg,"S=%+4d err%c%+5d",
1054                            signedSpeed,stickyStuck?'S':'=',error*adjKp/8);
1055                        sprintf(msg+16,"T=%+4d ie%c%6ld",
1056                            pot,stickyOpen?'O':'=',ierr>>KIp);
1057                        stickyStuck=0;
1058                        stickyOpen=0;
1059                        break;
1060                    case 4: // electrical info, current, supply voltage, power (VARS), power dumping/thrust
1061                        sprintf(stmp,"Th=%3d",mthrust);
1062                        sprintf(msg,"I=%4dm Vs=%4dm%s P=%5dmW",mIfet,Vs_mV,(pdump)?"Pdump!":stmp,PmW);
1063                        pdump=FALSE;    // this sticky flag is set if ISR dumps power
1064                        break;
1065                    case 5: // PWM thrust and mean Back EMF, tells you about your loco, push and speed
1066                        sprintf(msg,"drive=%d \rVbemf=%d mV",mthrust,Vbemf);
```

```
1067                        break;
1068                    case 6: // analysis of back-EMF noise
1069                        minbemf = (((long)Vs_mV)*minspeed)/361L;
1070                        maxbemf = (((long)Vs_mV)*maxspeed)/361L;
1071                        sprintf(msg,"BEMF=%4d minmax %c %4d / %4d",Vbemf,stickyStuck?'S':' ',minbemf,maxbemf);
1072                        maxspeed-=10;minspeed+=10;
1073                        stickyStuck=FALSE;
1074                        break;
1075                    case 7: // status of link, feedback gain, timeout, mode, overload
1076                        sprintf(msg,"Vlink=%2d%% TMO=%dm",Vlinkpc,tmomins);
1077                        sprintf(msg+16,"Thmax=%3d Ka=%3d",adjPWM,adjKp);
1078                        potTouched=FALSE;                       // clear for case 10 below
1079                        break;
1080                    case 8: // how long turned on, how long running
1081                        NVReadLong(RUNMINS,&ltmp);      // elapsed running time
1082                        NVReadLong(ONMINS,&ltmp2);      // On time
1083                        sprintf(msg,"On %ld mins\rRun %ld mins",ltmp2,ltmp);
1084                        break;
1085                    case 9: // how many automatic runs, how many reboots
1086                        NVReadLong(AUTORUNS,&ltmp);     // autoruns
1087                        NVReadLong(BOOTS,&ltmp2);       // boots
1088                        sprintf(msg,"%ld Autoruns\r%ld Boots",ltmp,ltmp2);
1089                        break;
1090                    case 10:        // adjust constant
1091                        if(DISPLAYING){
1092                            sprintf(msg,"Set ABS=%ld\rPB to exit%c",absdist,saveABS?'!':' ');
1093                            writeDisplay(msg);
1094                        }
1095                        if(potTouched){absdist=lastPotADC*1024L;saveABS=TRUE;potTouched=FALSE;}
1096                        break;
1097                    case 11:        // adjust constant
1098                        if(DISPLAYING){
1099                            sprintf(msg,"Set Kp=%d\rPB to exit%c",Kpset,saveKP?'!':' ');
1100                            writeDisplay(msg);
1101                        }
1102                        if(potTouched){Kpset=lastPotADC/5+9;saveKP=TRUE;potTouched=FALSE;}
1103                        break;
1104                }
1105                if(displayState>11)displayState=1;
1106                writeDisplay(msg);
1107            }
```

```
1108        }
1109
1110    // ------------------------------------------------------------------
1111
1112
1113    } // end of eternal while
1114
1115    exit(1);    // should only get here by soft reboot
1116}
1117
1118// RED is off=128, 157=max
1119// GREEN is off=158, 187=max
1120// BLUE is off=188, 217=max (blue seems more strong)
1121void setLCDcolour(unsigned char col){
1122    static unsigned char current=0;
1123    if(col==current)return;         // no change
1124    current=col;                    // switching to this
1125    while(TXIE){;}                  // wait till LCD free
1126    msg[0]=msg[2]=msg[4]='|';       // set...
1127    switch(col){
1128        case DSPAQUA:
1129            msg[1]=128+10;          // red almost off
1130            msg[3]=158+29;          // green max
1131            msg[5]=188+23;          // blue high
1132        break;
1133        default:
1134        case DSPWHITE:
1135            msg[1]=157;             // red max
1136            msg[3]=187;             // green max
1137            msg[5]=217;             // blue max
1138        break;
1139        case DSPCREAM:
1140            msg[1]=157;             // red max
1141            msg[3]=187;             // green max
1142            msg[5]=188+8;           // blue mid
1143//          msg[1]='+';     // why does this not work? on v1.1 ONLY?
1144//          msg[2]=0xff;    // R
1145//          msg[3]=0xff;    // G
1146//          msg[4]=0x00 ;   // B
1147//          msg[5]='\0';
1148        break;
```

```
1149            case DSPRED:
1150                msg[1]=157;             // red max
1151                msg[3]=158;             // green off
1152                msg[5]=188;             // blue off
1153            break;
1154            case DSPYEL:
1155                msg[1]=157;             // red max
1156                msg[3]=187;             // green max
1157                msg[5]=188;             // blue off
1158            break;
1159            case DSPGRN:
1160                msg[1]=128;             // red off
1161                msg[3]=187;             // green max
1162                msg[5]=188;             // blue off
1163            break;
1164        }
1165        msg[6]='\0';                    // terminate string
1166        TXIE=TRUE;                      // execute
1167        return;
1168    }
1169
1170    void writeDisplay(char *message){
1171        unsigned char p;
1172
1173        char tmp[44];
1174        strcpy(tmp,"|-");
1175        strcat(tmp,message);
1176        strcpy(message,tmp);
1177    //  for(p=0;message[p]!='\0';p++){;}         // find NULL position
1178    //  for(;p>3;p--){message[p+4]=message[p];} // move text 4 chars
1179    //  message[0]='|';                         // command coming
1180    //  message[1]='-';                         // move home
1181    //  message[2]='\\';                        // LF coming
1182    //  message[3]='n';                         // return
1183        TXIE=TRUE;                              // trigger EUSART send
1184        dspdly=3;                               // no update for 400ms
1185        return;
1186    }
1187
1188
1189    int median(int i1, int i2, int i3){
```

```
1190     if(i1>=i2 && i3<=i2) return(i2);
1191     if(i3>=i2 && i1<=i2) return(i2);
1192     if(i2>=i1 && i3<=i1) return(i1);
1193     if(i3>=i1 && i2<=i1) return(i1);
1194 //  if(i2>=i3 && i1<=i3) return(i3);
1195 //  if(i1>=i3 && i2<=i3)
1196     return(i3);
1197 }
1198 
1199 int median5(int A, int B, int C, int D, int E){
1200 
1201     int a,b,c,d,e;
1202 
1203     if (A > B){a=B;b=A;}else{a=A;b=B;}      // if A>B, swap A,B
1204     if (C > D){c=D;d=C;}else{c=C;d=D;}      // if C>D, swap C,D
1205     if (a > c){e=a;a=c;c=e;e=b;b=d;d=e;}    // swap a&c, b&d
1206     if (E > c){
1207         if (E > d){  // A C D E
1208             if (b > d) return(d);   // (A, C, D, E, B)
1209              else{
1210                 if (b < c)
1211                     return(c);  // (A, B, C, D, E)
1212                 else
1213                     return(b);  // (A, C, B, D, E)
1214              }
1215         }else{      // A C E D
1216             if (b > E) return(E);       // (A, C, E, B, D)
1217              else{
1218                 if (b < c)
1219                     return(c);  // (A, B, C, E, D)
1220                 else
1221                     return(b);  // (A, C, B, E, D)
1222              }
1223         }
1224     }else{
1225         if (E < a){  // E A C D
1226             if (b > c) return(c);   // (E, A, C, D, B)
1227              else    return(b); // (E, A, B, C, D)
1228         }else{  // A E C D
1229             if (b > c)
1230                 return(c);  // (A, E, C, B, D)
```

```
            else{
                if (b < E)
                    return(E);  // (A, B, E, C, D)
                else
                    return(b);  // (A, E, B, C, D)
            }
        }
    }
}


//****************************************************************************
//NonVolatile memory reads and writes
//****************************************************************************
unsigned char NVReadByte (unsigned char nvrbadr)
{
    unsigned char ctmp;
    GIE=0;
    EEADR = nvrbadr;
    RD = 1;                 /* Part of EECON1. */
    ctmp = EEDATA;
    GIE=1;
   return (ctmp);
}
void NVWriteByte (unsigned char nvwbadr, unsigned char nvwbdat)
{
    GIE = 0;                                            // disable ints globally
    do {
        EEADR = nvwbadr;                                // load EEPROM desired write address
        EEDATA = nvwbdat;                               // load EEPROM data byte to be written
        WREN = 1;                                       // enable writes
        WRERR = 0;                                      // clear error bit
        EECON2 = 0x55;                                  // start HW sequence
        EECON2 = 0xaa;                                  // middle HW step
        WR = 1;                                         // initiate HW write
        while (WR)                                      // until WRite completed...
            ;                                           // do nothing
        WREN = 0;                                       // disable further writes
        EEIF = 0;                                       // clear error int flag (not used)
    } while (WRERR);                                    // confirm no errors and data good
    GIE = 1;                                                // re-enable ints globally
```

```
1272        return;
1273    }
1274    void NVReadLong (unsigned char nvrladr, unsigned long int *lint)
1275    {
1276            unsigned char nvrli;
1277
1278            for(nvrli=0;nvrli<4;nvrli++) {
1279                    *((unsigned char*)lint+nvrli) = NVReadByte(nvrladr+nvrli);
1280            }
1281            return;
1282    }
1283    void NVWriteLong (unsigned char nvwladr, unsigned long int *plint)
1284    {
1285            unsigned char nvwli;
1286
1287            for(nvwli=0;nvwli<4;nvwli++) {
1288                    NVWriteByte(nvwladr+nvwli, *((unsigned char*)plint+nvwli) );
1289            }
1290            return;
1291    }
1292
```