```
/*--------------------------------------------------------------------------------
 *  REED Controller - Renewable Energy Excess Diversion
 *
 *      Author:     Paul Alting van Geusau
 *      Created:    26 Jun 2015
 *      Modified:   02 Aug 2015
 *
 *  Configured for:
 *                  Arduino Mega
 *                  Modbus RTU master
 *                  Two outputs, one as lead PID PWM control, other as on/off lag.
 *
 *  This control program communicates with a solar charge controller
 *  to access current state information of the battery bank and
 *  charging process.
 *
 *  Based on the state, this control program is designed to divert excess
 *  solar energy available from a PV array to other loads, typically heating
 *  such as water storage heating.
 *
 *  This control program allows for two loads to be individually controlled in
 *  a lead - lag strategy, whereby the lead load will be the initial load connected
 *  and will always have the energy supplied to it modulated via PWM. The lag load
 *  will only ever be operated if the lead load is fully on and the PV array system
 *  is able to supply extra energy. When this happens, the lag lod will be switched
 *  fully on and the lead load will have its energy cut back to maintain balance.
 *
 *  Allowance for swapping over the lead and lag loads can be made via input to the
 *  control program via input selection.
 *
 *  Control Process:
 *      [From Chris]
 *      Any controlled changes to the diversion load (by adjusting the PWM value) should make sure
 *      that the current battery voltage remains as close as possible to the target battery voltage.
 *      As long as the current battery voltage is within about 0.4 volts of the target battery
 *      voltage then it will not affect the charge controller mode, ie BULK, ABSORB, FLOAT, EQ.
 *      Ideally, the current battery voltage should be kept at 0.2 volts under the target battery voltage.
 *
 *      Attempting to maintain the 0.2 volts under target will ensure that even when the
 *      sun sets or is obscured by clouds temporarily, the system will compensate and
 *      reduce the power going to the diversion load.
 *      The battery bank should never be discharged due to the diversion load.
 *      [/Chris]
 *
 *      The diversion control output signal is derived from the difference between actual
 *      battery voltage and the setpoint for battery voltage.
 *
 *      While the battery voltage is not less than 0.3 volts below setpoint voltage, the
 *      charge controller mode will not be effected.
 *
 *      For optimum operation, the differential between battery voltage and setpoint
 *      voltage should not fall to within 0.2 volts below the setpoint target setting.
 *
 *      The lead output is only ever modulated by PID PWM.
 *      The lag output is only ever off or on.
 *
 *          if lead and pid_cv > 95% (980 ADC counts) and lag not on then
 *              lag on
 *
 *          if lead and pid_cv < 5% (50 ADC count) and lag on then
 *              lag off
 *
 *  PWM Options:
 *      Conceptually, there are two options for diverting available PV energy:
 *          1. Direct from the PV array as high voltage DC,
 *          2. Taken from the battery side via inverter as high voltage AC.
 *
 *      The two options differ in terms output SSR control, where option 1 uses
 *      a DC SSR and will be PWM at a fast PWM duty cycle. Option 2 uses an AC SSR
 *      and as such can only be PWM at the AC line frequency or lower due to zero
 *      crossing on and off switching of the AC SSR.
 *
 *      During the diversion process, the battery voltage is continuously monitored,
 *      either via direct analog input or via communications with the charge controller.
 *
 *      The amount of energy diverted is controlled and maintained through monitoring
 *      the PV array volts as well as battery volts through using a PID loop contol
 *      function. The output of the PID contol is then passed into a PWM function to
 *      provide the correct switching of the SSR.
 *
 *  PID:
 *      I developed this PID based on other code I have looked over.
 *      This PID routine is fast as it is toatlly based on integer math, and makes
 *      no use of any floating point math or variables.
 *
 *      The input and output terms are integer, as are the loop paramneters.
 *      The output, being scaled from 0 to 1023 fits magically with a timer based PWM.
 *
 *      For the PID to function, a desired setpoint must also be passed into the
 *      PID loop control function. The error difference between the setpoint and
 *      actual monitored data is what determines the basic PID process.
 *
 *  Libraries:
 *      Mention needs to be made on a few of the libraries.
 *      Please ensure yoyu get the libraries form the links specified.
 *      I have researched these libraries for what I consider the best.
 *      There will be others by the same namne, either old or other, but not as good.
 *
 *  •   The Timer1 library has been fully updated and optimised by Paul Stoffregen
 *      from the original Timer1 code.
 *      Download from Paul's GitHub site - https://github.com/PaulStoffregen/TimerOne
 *
 *  •   DDModbusMaster is an updated and improved verion by Frank Dietrich of the
 *      SimpleModbusMaster library originally written by Juan Bester. It is not on
 *      GitHib as yet and is included in the files list with this code.
 *
 *  •   LiquidCrystal is the new LCD library, not the standard Arduino IDE.
 *      Typical LCD http://www.robotshop.com/ca/en/dfrobot-i2c-twi-lcd-module.html
 *
 *  Notes:
 *      The communications protocol used to request data from a charge controller
 *      is Modbus, and options will be for either Modbus RTU or Modbus TCP.
 *      Modbus RTU uses the async serial port on the Arduino and the port may be
 *      selected depending on the model Arduino and its number of ports available.
 *      Modbus TCP uses Ethernet and will use either an onboard or attached
 *      Ethernet controller.
 *
 *      REED controller is best operated on an Arduino Mega, but can operate in a
 *      limited fashion on smaller Arduino boards like UNO. In this case, a choice
 *      needs to be made concerning the use of Modbus TCP and number of PWM loads.
 *      Using TCP means Ethernet, whicg uses pins 10, 11, 12 and 13 on a UNO.
 *      The PWM outputs are on pins 9 and 10. Therefore pin 10 is in contention
 *      between Ethernet and PWM. Modbus TCP can be used if only one PWM load is
 *      required. If Modbus RTU is selected, then both PWM outputs may be used for loads.
 *      On the Arduino Mega, there is no such issue.
 *
 *      This control program which operates on an Arduino Mega will be Modbus master,
 *      where this control program initiates all comunication requests to devices.
 */

/*--------------------------------------------------------------------------------------------------------------------
 *  Includes
 */
#include    "main.h"                            // header for main entry point:
//#include  "wire.h"                            // header for IC2 / TWI interface:
//#include  "EEPROM.h"                          // header https://github.com/arduino/Arduino/blob/master/hardware/arduino/avr/libraries/EEPROM
#include    "BasicTerm.h"                       // header for VT100/VT220 terminal: - https://github.com/nottwo/BasicTerm
#include    "TimerOne.h"                        // header for updated Timer1 - https://github.com/PaulStoffregen/TimerOne
//#include  "LiquidCrystal_I2C.h"               // header https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads
#include    "includes/pidControl.h"             // header for PID loop controller routine:
#include    "includes/modbus_maps.h"            // header for Modbus routines and controller templates:
//#include  "includes/sensor.h"                 // header for reading analogue sensors:
#include    "includes/defines.h"                // header for definitions:

//--------------------------------------------------------------------------------------------------------------------
//  const   type            variable            =   value;                      comment
            boolean         firstScan           =   true;                   // first scan of code, if needed:
            boolean         rtcOneOS            =   false;                  // Oneshot 1 second trigger:
            boolean         rtcFiveOS           =   false;                  // Oneshot 5 second trigger:
            boolean         rtcTenOS            =   false;                  // Oneshot 10 second trigger:
            boolean         rtcMinOS            =   false;                  // Oneshot 60 second trigger:
            boolean         rtc5MinOS           =   false;                  // Oneshot 300 second trigger:
            boolean         rtc15MinOS          =   false;                  // Oneshot 900 second trigger:
            boolean         rtcHourOS           =   false;                  // Oneshot 3600 second trigger:
            boolean         rtcDayOS            =   false;                  // Oneshot end of day trigger:

            uint8_t         process100Counter   =   9;                      // counter decremented by 100mS interrupt for 1000mS flag:
            uint8_t         spIndex             =   0;                      // setpoint index:

            uint32_t        lastValues          =   0;                      // last time we repaint the screen:
    const   uint32_t        periodValues        =   500;                    // rate period to repaint LCD with live data:
            uint32_t        rtcTime             =   0;                      // internal software rtc time in seconds:

    struct  load_t {
            boolean         lead_state_a;                                   // load a in lead being PWM, load b in lag:
            boolean         lag_state;                                      // load b lag status, either on or off:
            uint8_t         lead_output;                                    // selected lead PWM pin:
            uint8_t         lag_output;                                     // selected lag PWM pin:
            uint16_t        pid_cv;                                         // PID output as percentage:
        };
            load_t          load;                                           // instance of load:

//                                      sp kp  ki kd pid_min  pid_max     // PID setpoint and pid terms:
            PID             diversion_pid(0, 10, 5, 0, PID_MIN, PID_MAX);   // instance of PID with initial parameters:

            BasicTerm       term(&Serial);                                  // instance of BasicTerm VT100/VT220 terminal handler:

/*--------------------------------------------------------------------------------
 *  void setup()
 *  Configure initial states and IO ports.
 */
void setup() {
    pinMode(LED, OUTPUT);                                                   // set onboard LED pin as ooutput:
    digitalWrite(LED, LOW);                                                 // set LED off:

//  uint8_t i;
//  for(i = A0; i < A6; i++)                                                // quick method to set range of I/O as inputs:
//      pinMode(i, INPUT);
//
//  for(i = 2; i < 13; i++)                                                 // quick method to set range of I/O as outputs:
//      pinMode(i, INPUT);
//  pinMode(pin_CSether, OUTPUT);                                           // set ethernet CS as output, if used:

    load.lead_output = PIN_OUTPUT_A;                                        // set lead output to A:
    load.lag_output = PIN_OUTPUT_B;                                         // set lead output to B:
    load.lead_state_a = true;                                               // set output A to lead state:

    mb_status.poll_count = 0;
    mb_status.poll_count_10K = 0;
    mb_status.timeouts = 0;
    mb_status.address = 0;
    mb_status.code = 0;

    reed_param.pid_sp_offset = 3;                                           // set the setpoint offset trim to be -0.3 volts:
    reed_param.pid_sp_low_clamp = 0;                                        // diversion lower clamp, typically -0.4V below charge controller setpoint:

    cc_param.vBattery_pv = 0;
    cc_param.vArray_pv = 0;
    cc_param.battery_amps = 0;
    cc_param.kwh = 0;
    cc_param.watts = 0;
    cc_param.mode = 0;
    cc_param.vBattery_sp = 0;

    mbPacketInit();                                                         // initialise Modbus packets based on charge controller definition:

    Timer1.initialize(100000);                                              // initialize Timer3, and set a 100 milli second period:
    Timer1.pwm(PIN_OUTPUT_A, 0);                                            // pre-initialise as pwm output on pin and set for 0% duty cycle:
    Timer1.pwm(PIN_OUTPUT_B, 0);                                            // pre-initialise as pwm output on pin and set for 0% duty cycle:
    Timer1.attachInterrupt(process100mS);                                   // attach interrupt to routine, for PID and other timing control:

#if defined(__AVR_ATmega2560__)                                             // AVR 2560 CPU as found on Mega:
    pinMode(pin_SPI_CS, OUTPUT);                                            // set hardware SPI CS pin as output if using SPI / Ethernet:
#define HAVE_VT100                                                          // if serial VT100 / VT220 terminal:
    Serial.begin(115200);                                                   // if Mega, then init default port for VT100 GUI:
    term.init();                                                            // initialise terminal screen:
    term.cls();                                                             // clear the terminal screen:
    term.show_cursor(false);                                                // hide the cursor:
    guiPrintStatic();                                                       // go and display the static screen data:
#endif /* defined(__AVR_ATmega2560__) */
}

/*--------------------------------------------------------------------------------
 *  void loop()
 *  Mission Control Centre:
 *  Where all controll events are handled from.
 *  Check the Modbus state machine continuously.
 */
void loop() {
#if defined(HAVE_VT100)
    if (term.available())
    {
        term.position(48, 1);
        term.print(F(vtfCyan"Command: "clearEOL));

        switch(term.get_key()) {
        case 'e':                                                           // e toggle LED
            digitalWrite(LED, !digitalRead(LED));
            term.print(F("Controller LED toggled"));
            break;
        case 'c':                                                           // c clear screen:
            term.init();
            term.cls();
            term.show_cursor(false);
            guiPrintStatic();
            break;
        case 'p':                                                           // p to select setpoint:
            term.print(F("Setpoint PID P term"));
            spIndex = 1;
            break;
        case 'i':                                                           // i to select setpoint:
            term.print(F("Setpoint PID I term"));
            spIndex = 2;
            break;
        case 'd':                                                           // d to select setpoint:
            term.print(F("Setpoint PID D term"));
            spIndex = 3;
            break;
       case 's':                                                            // s to select setpoint:
            term.print(F("Setpoint Offset Adjust"));
            spIndex = 4;
            break;
       case '-':                                                            // - to decrement setpoints:
            switch(spIndex) {
            case 1:
                diversion_pid.kpPID--;                                      // decrement pid p term:
                break;
            case 2:
                diversion_pid.kiPID--;                                      // decrement pid i term:
                break;
            case 3:
                diversion_pid.kdPID--;                                      // decrement pid d term:
                break;
            case 4:
                reed_param.pid_sp_offset--;                                 // decrement setpoint offset:
                break;
            }
            break;
       case '+':                                                            // + to increase setpoints:
            switch(spIndex) {
            case 1:
                diversion_pid.kpPID++;                                      // increment pid p term:
                break;
            case 2:
                diversion_pid.kiPID++;                                      // increment pid i term:
                break;
            case 3:
                diversion_pid.kdPID++;                                      // increment pid d term:
                break;
            case 4:
                reed_param.pid_sp_offset++;                                 // increment setpoint offset:
                break;
            }
            break;
       case 'l':                                                            // l to swap load lead / lag:
            if (load.lead_state_a)
            {
                load.lead_state_a = false;                                  // set output A a as lag:
                load.lag_output = PIN_OUTPUT_A;
                load.lead_output = PIN_OUTPUT_B;
                Timer1.disablePwm(PIN_OUTPUT_A);                            // disable PWM mode for output A:
                term.print(F("Lead is Output A, Lag is output B"));
            }
            else
            {
                load.lead_state_a = true;                                   // set output A a as lead:
                load.lag_output = PIN_OUTPUT_B;
                load.lead_output = PIN_OUTPUT_A;
                Timer1.disablePwm(PIN_OUTPUT_B);                            // disable PWM mode for output B:
                term.print(F("Lead is Output B, Lag is output A"));
            }
```

```
            break;
        }
    }

    if (rtcTenOS)                                                   // if ten second timer set:
    {
        if (!spIndex)                                               // if setpoint indexer is not set:
        {
            term.position(48, 1);                                   // postion cursor at command line:
            term.print(F(vtfCyan"Command: "clearEOL));              // clear current setpoint text:
        }
        else
            spIndex = 0;                                            // reset spIndex after 10 second timeout:
        rtcTenOS = false;
    }
#endif /* defined(HAVE_VT100) */

    if (cc_param.vBattery_pv > reed_param.pid_sp_low_clamp) {       // if operationg within the proportional band:
//  if (true) {
        Timer1.setPwmDuty(load.lead_output, diversion_pid.cvPID);   // set the pwm with the output of the pid output to control the SSR:
        if ((diversion_pid.cvPID > (PID_MAX - 2)) && !load.lag_state)   // if PID output near max and lag load is off:
            load.lag_state = true;                                  // set trigger to set lag load on:

        if ((diversion_pid.cvPID < (PID_MIN + 2)) && load.lag_state)    // if PID output near min and lag load is on:
            load.lag_state = false;                                 // reset trigger to set lag load off:

        digitalWrite(load.lag_output, load.lag_state);              // operate lag output from trigger:
    }
    else
        Timer1.setPwmDuty(load.lead_output, 0);                     // if not in the proportional band, then set the pwm to zero:

//  regs[cc_pwm] = (uint16_t)diversion_pid.cvPID;                   // used by simulator testing:
    charge_controller.modbus_update();                              // read the Classic's modbus registers:

    if (firstScan)
        firstScan = !firstScan;                                     // end of firstScan:
}

/*--------------------------------------------------------------------------------
 *  void process100mS()
 *  Routine is called every 100mSec by an interrupt.
 *  This is the place to do all things that need predictable time deterministic processing,
 *  such as time based statsitical crunching, and importantly PID loop controllers.
 *  Set flags on specific periodic time bases.
 */
void process100mS() {
    if (!process100Counter) {                                       // if counter is zero:
        process100Counter = 9;                                      // reset counter to preset, 10 x 100mS is 1000mS:
        ++rtcTime;                                                  // increment software rtc value:
        if (!(rtcTime % 1)) {                                       // check one second:
            rtcOneOS = true;
            if (!(rtcTime % 5)) {                                   // check five second:
                rtcFiveOS = true;
                if (!(rtcTime % 5)) {                               // check five second:
                    rtcTenOS = true;
                    if (!(rtcTime % 60)) {                          // check 60 second:
                        rtcMinOS = true;
                        if (!(rtcTime % 300)) {                     // check 300 second:
                            rtc5MinOS = true;
                            if (!(rtcTime % 3600)) {                // check 3600 second:
                                rtcHourOS = true;
                                if (!(rtcTime % 86400L)) {          // check 86400 second:
                                    rtcDayOS = true;
                                }
                            }
                        }
                    }
                }
            }
        }
    } else
        --process100Counter;                                        // decrement counter:

    diversion_pid.spPID = reed_param.volt_setpoint;                 // pass the _sp value into the PID block:
    diversion_pid.pvPID = cc_param.vBattery_pv;                     // pass the _pv value into the PID block:
    diversion_pid.loop();                                           // call PID controller:
    load.pid_cv = (uint16_t)(100L * diversion_pid.cvPID / 1023L);   // calculate % value of PID output for display:
}

/*--------------------------------------------------------------------------------
 * void guiPrintStatic()
 * This routine is only for static text data for display and only needs to be called at
 * startup or when clear display is needed. This reduces overall CPU time n each loop cycle.
 */
void guiPrintStatic() {
    uint8_t i = 0;

    term.position(0, 0);
    term.print(F(vtfLightYellow "REED Controller\r\nThe Renewable Energy Excess Diversion Controller version 0.2.0"));
    term.position(0, 120);
    term.print(F("Aug 2015" vtfCyan));

    term.position(3, 0);
    for (i = 1; i < 133; i++)                                       // horizontal line at top:
        term.print(char(241));

//      term.print(char(240));
//      term.print(char(242));
//      term.print(char(243));
//      term.print(char(244));
//      term.print(char(245));
//      term.print(char(246));
//      term.print(char(247));

//  term.position(44, 0);
//  for (i = 1; i < 133; i++)
//      term.print(char(241));

    term.position(47, 0);
    for (i = 1; i < 133; i++)                                       // horizontal line at bottom:
        term.print(char(241));

    term.position(46, 1);
    term.print(F(vtfDarkGrey"Messages:"));                          // message label:

    term.position(48, 1);
    term.print(F(vtfCyan"Command:"));                               // command label:

    term.position(4, 0);
    term.println(F(posx4 vtfCyan "Charge Controller Status" vtfDarkGrey));
    term.println(F(posx8"Mode"));                                   // charge controller mode:
    term.println(F(posx8"Array Volts"));                            // charge controller array volts:
    term.println(F(posx8"Battery Volts"));                          // charge controller battery volts:
    term.println(F(posx8"Charge Amps"));                            // charge controller charge current:
    term.println(F(posx8"Charge Watts"));                           // charge controller charge watts:
    term.println(F(posx8"Charge KWHr"));                            // charge controller accumulated chareg KWHrs:
    term.println(F(posx8"Battery Setpoint"));                       // charge controller battery setpoint target:

    term.position(4, 0);
    term.println(F(posx48 vtfCyan "Diversion Controller Status" vtfDarkGrey));
    term.println(F(posx52 "Mode"));                                 // Off or On:
    term.println(F(posx52 "Setpoint Offset"));                      // reed controller battery setpoint offset volts:
    term.println(F(posx52 "Diversion Setpoint"));                   // reed controller battery setpoint target volts:

    term.position(9, 0);
    term.println(F(posx52 "Lead Load"));                            // Output A or Output B:
    term.println(F(posx52 "Output A Load"));                        // % value:
    term.println(F(posx52 "Output B Load"));                        // % value:

    term.position(13, 0);
    term.println(F(posx52 "Load Estimate "));                       // Watts:
    term.println(F(posx52 "Run Time Today"));                       // Hours:
    term.println(F(posx52 "Run Time Total"));                       // Hours:

    term.position(17, 0);
    term.println(F(posx52 "PID P term"));                           // P term:
    term.println(F(posx52 "PID I term"));                           // I term:
    term.println(F(posx52 "PID D term"));                           // D term:
    term.println(F(posx52 "PID Output"));

    term.position(35, 0);
    term.println(F(posx4 vtfCyan "Modbus status" vtfDarkGrey));     // modbus status section:
    term.println(F(posx8"Poll Count 0 - 9999"));                    // modbus poll count 0 to 9999:
    term.println(F(posx8"Poll Count  x 10000"));                    // modbus poll count x 10k:
    term.println(F(posx8"Error Timeouts"));                         // modbus timeouts:
    term.println(F(posx8"Error Exceptions"));                       // modbus exception errors:
    term.println(F(posx8"Error Unhandled"));                        // modbus unhandled errors:

//  term.position(20, 1);
//  term.print(F("No. Din   Ain"));
//  term.position(19, 0);
//  term.print(F("Analog Inputs"));
//
//  myTerm.row = 20;
//  for(i = 0; i <= 5; i++)
//  {
//      term.position(myTerm.row, i * 6);
//      term.print(i);
//  }
}

/*--------------------------------------------------------------------------------
 * void guiPrintMessages()
 * Displays formatted real-time status information.
 */
void guiPrintMesages(char * msg) {
//  term.position(46, 1);
//  term.print(F(vtfDarkGrey"Messages:"));

    term.position(46, 11);
    term.print(F(vtfDarkGrey));
    term.println(msg);
}

/*--------------------------------------------------------------------------------
 * void guiPrintLive()
 * Displays formatted real-time status information.
 */
void guiPrintLive() {
    term.position(5, 28);
    term.print(F(vtfGreen));

    switch (cc_param.mode)
    {
    case mode_resting:                                              // resting:
        term.print(F("Resting   "));
        break;
    case mode_absorb:                                               // absorb:
        term.print(F("Absorb    "));
        break;
    case mode_bulkmppt:                                             // bulk:
        term.print(F("Bulk-Mppt "));
        break;
    case mode_float:                                                // float:
        term.print(F("Float     "));
        break;
    case mode_floatmppt:                                            // float MPPT:
        term.print(F("Float-Mppt"));
        break;
    case mode_equalise:                                             // equalise:
        term.print(F("Equalize  "));
        break;
    case mode_hypervoc:                                             // hyper VOC:
        term.print(F("Hyper-Voc "));
        break;
    case mode_equlisemppt:                                          // equalise MPPT:
        term.print(F("Eq-Mppt   "));
        break;
    default:                                                        // undefined error if get here:
        break;
    }

    term.position(6, 28);
    term.print(cc_param.vArray_pv / 10);
    term.print('.');
    term.print(cc_param.vArray_pv % 10);

    term.position(7, 28);
    term.print(cc_param.vBattery_pv / 10);
    term.print('.');
    term.print(cc_param.vBattery_pv % 10);

    term.position(8, 28);
    term.print(cc_param.battery_amps / 10);
    term.print('.');
    term.print(cc_param.battery_amps % 10);

    term.position(9, 28);
    term.print(cc_param.watts / 10);
    term.print('.');
    term.print(cc_param.watts % 10);

    term.position(10, 28);
    term.print(cc_param.kwh / 10);
    term.print('.');
    term.print(cc_param.kwh % 10);

    term.position(11, 28);
    term.print(cc_param.vBattery_sp / 10);
    term.print('.');
    term.print(cc_param.vBattery_sp % 10);

    term.position(6, 72);
//  term.print('-');
    term.print(reed_param.pid_sp_offset / 10);
    term.print('.');
    term.print(abs(reed_param.pid_sp_offset % 10));

    term.position(7, 72);
    term.print(reed_param.volt_setpoint / 10);
    term.print('.');
    term.print(reed_param.volt_setpoint % 10);

    term.position(9, 72);
    term.print(load.lead_state_a ? "Output A" : "Output B");

    term.position(10, 72);
    if (load.lead_state_a)
    {
        term.print(load.pid_cv);
        term.print(F("%  "));
    }
    else
        term.print(load.lag_state ? "On " : "Off");

    term.position(11, 72);
    if (!load.lead_state_a)
    {
        term.print(diversion_pid.cvPID);    //load.pid_cv
        term.print(F("%  "));
    }
    else
        term.print(load.lag_state ? "On " : "Off");

    guiPrintHBar();

//  term.position(14, 72);
//  if(digitalRead(LED))
//      term.print(F(vtfLightRed "ON " vtfGreen));
//  else
//      term.print(F(vtfYellow "OFF" vtfGreen));

    term.position(17, 72);
    term.print(diversion_pid.kpPID);
    term.print(F("  "));

    term.position(18, 72);
    term.print(diversion_pid.kiPID);
    term.print(F("  "));

    term.position(19, 72);
    term.print(diversion_pid.kdPID);
    term.print(F("  "));

    term.position(20, 72);
    term.print(diversion_pid.cvPID);
    term.print(F("  "));
//  term.print(load.pid_cv);
//  term.print(F("%  "));

    term.position(36, 28);
    term.print(mb_status.poll_count);
    term.print(F("   "));

    term.position(37, 28);
    term.print(mb_status.poll_count_10K);

//  term.set_color(BT_MAGENTA, BT_BLACK);
//  myTerm.row = 20;
//  term.print(F(vtfCyan));
//  term.print(F(vtfGreen));
```

```
643 //
644 //  for(i = 0; i <= 5; i++)
645 //  {
646 //      term.position(myTerm.row + 1, i * 6);
647 //      term.print(analogRead(i));
648 //  }
649 //  term.print(F(vtfGreen));
650 //  myTerm.row = 21;
651 //
652 //  for(i = 0; i < 16; i++)
653 //  {
654 //      term.position(myTerm.row + i, 2);
655 //      term.print(i);
656 //      term.position(myTerm.row + i, 6);
657 //      term.print(digitalRead(i), HEX);
658 //      term.position(myTerm.row + i, 11);
659 //      term.print(analogRead(i));
660 //  }
661 }
662
663 /*-------------------------------------------------------------------------------
664  *  guiPrintHBar()
665  *  Update GUI display horizontal bargraph for PID output.
666  */
667 void guiPrintHBar() {
668     uint8_t lag_pos = 11;
669     uint8_t lead_pos = 10;
670     int16_t bar = load.pid_cv / 5;
671
672     if (load.lead_state_a)
673     {
674         lead_pos = 10;
675         lag_pos = 11;
676     }
677     else
678     {
679         lead_pos = 11;
680         lag_pos = 10;
681     }
682
683 //  term.position(load.lead_state_a ? 10 : 11, 84);             // more clever method:
684     term.position(lead_pos, 84);
685     term.print(F("|"));
686     for (int j = 0; j < bar + 1; j++)
687         term.print(F("#" clearEOL));
688
689     term.position(lag_pos, 84);
690     term.print(F("|"));
691     if (load.lag_state)
692     {
693         for (int j = 0; j < 20 + 1; j++)
694             term.print(F("#"));
695     }
696     else
697         term.print(F(clearEOL));
698
699 }
700
701 /*-------------------------------------------------------------------------------
702  *  guiPrintModbusTimeout()
703  *  Update GUI display with Modbus status if called.
704  */
705 void guiPrintModbusTimeout() {
709
710 /*-------------------------------------------------------------------------------
711  *  guiPrintModbusException()
712  *  Update GUI display with Modbus status if called.
713  */
714 void guiPrintModbusException() {
720
721 /*-------------------------------------------------------------------------------
722  *  guiPrintModbusUnhandled()
723  *  Update GUI display with Modbus status if called.
724  */
725 void guiPrintModbusUnhandled() {
735
```