```
/*
 * MODLOG - MODBUS DATA RECORDER
 *
 *      EE537 - Application of Embedded Processors
 *      MODBUS TCP Data Logger
 *      v1.0
 *
 *      All times are UTC
 *
 *      UNIVERSITY OF ALABAMA BIRMINGHAM
 *      James M. Olson
 */

#include <SPI.h>                          //SPI library for Ethernet Module
#include <SD.h>                           //SD library for SD card access
#include <Ethernet2.h>                    //Ethernet library for Ethernet Module
#include <Wire.h>                         //Wire Library for OLED & RTC
#include <Adafruit_GFX.h>                 //Adafruit grafix library for OLED
#include <Adafruit_SSD1306.h>             //Adafruit OLED library
#include <Adafruit_FeatherOLED.h>         //Adafruit Feather specific OLED library
#include "RTClib.h"                       //RTC library
#include "MgsModbusEth2.h"                //MODBUS protocol library
#include <EthernetUdp2.h>                 //UDP for NTP time sync RTC

//#define DEBUG                             //Uncomment this line to enable Serial port debug messages

#define OLED_RESET 9                      //Reset PIN for OLED (not used, but required). Default is 4
#define Bpin 5                            //Feather PIN for "B" button on OLED. Interrupt to change OLED screen
#define Cpin 6                            //Feather PIN for "C" button on OLED. Interrupt to enable/disable recording
#define SDCardIn 7                        //Feather PIN for physical SD card instertion detection (not chip select)
#define SDcs 4                            //Feather PIN for SD card SPI bus chip select
#define SDwriteLED 8                      //Feather PIN for ONBOARD LED to indicate SD card activity.
#define VBATPIN A7                        //Analog input pin for reading battery voltage (2:1 ratio)
#define VUSBPIN A1                        //Analog input pin for measuring USB voltage (2:1 ratio)
#define MODMEASURE 11                     //LED to indicate polling modbus slave device
```

```
#define CONNECTED 12                      //LED to indicate slave is connected.
#define USBMINVOLT 4.7                    //Minimum USB voltage to declare it is disconnected.

/***************************
 *   Function Prototypes
 *
 **************************/
void sendWebData(EthernetClient client,DateTime now);                       //Funciton for generating web server view
void updateDisplay(DateTime now);                                           //Function for updating OLED display
void getMODBUSdata();                                                       //Funciton for requesting MODBUS data from SLAVE
void changeDisplayScreen();                                                 //Interrupt function to change OLED display screen
void changeRecordState();                                                   //Interrupt function to enable/disable SD card recording
void readDeviceSettings();                                                  //Read device settings from SD Card
void convertIPfromStr(String address, byte IP[]);                           //convert string to IP address
void convertMACfromStr(String address, byte MAC[]);                         //convert string to hex MAC ID
void createModPoint(String point);                                          //Add modbus data point to linked list
void getDeviceVoltages();                                                   //Read USB (external DC) & Battery voltage
String PrintDateTime(DateTime t);                                           //Creates a formatted time/date string for display & web
void dateTime(uint16_t* date, uint16_t* time);                              //Callback function for correcting SD card file time stamps
void synchRTC(bool fastUpdate);                                             //Synchronize RTC with NTP time server.
unsigned long sendNTPpacket(char *address);                                 //NTP Packet transmission for retriving time sync information


/***************************
 *  GLOBAL VARIABLES
 **************************/
char ver[6] = "v1.0";                         //Version Number
RTC_DS3231 rtc;                               //Real time clock instance
MgsModbus Mb;                                 //MsgMODBUS instance (Master)
Adafruit_SSD1306 display(OLED_RESET);         //Create OLED display instance
//Sd2Card card;                                 //SD library function for card data
//SdVolume volume;                              //SD library function for card data
//SdFile root;                                  //SD library function for card data
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday",
                             "Thursday", "Friday", "Saturday"};             //Days of the week for Webserver display
```

```
byte SlaveIP[4];                              //Slave IP Address
byte DeviceIP[4];                             //Device IP Address
byte DeviceGW[4];                             //Device default gateway
byte DeviceSN[4];                             //Device Subnet Mask
byte mac[6];                                  //MAC ID for Ethernet Card
char DeviceID[16]="SLAVE";                    //Slave Device Type (Default = SLAVE)
IPAddress ip(192,168,1,10);                   //Webserver IP Address set to default value
IPAddress gateway(192,168,1,1);               //Webserver Gateway set to default value
IPAddress subnet(255, 255, 0, 0);             //Webserver Subnet Mask set to default value
EthernetServer server(80);                    //Initialize Ethernet Web Server on port 80
bool recordData = true;                       //Enable/disable data recording to SD card
bool changeDisplay = false;                   //Interrupt flag to tell program to rotate to next graphic screen
bool changeRecord = false;                    //Interrupt flag to tell program to enable/disable recording to SD
bool hideScreen = false;                      //Hide screen flag to shut off or turn on OLED display
bool recordMeasurement = false;               //Flag to tell program it is time to save measurement to SD
bool createNewLogFile = true;                 //Flag to tell program it is necessary to create a new logfile on SD card
unsigned int measureIntervalSeconds = 1;      //Interval in seconds (default) for device to send out MODBUS request to slave
unsigned int recordIntervalSeconds = 10;      //Interval in seconds (default) for device to record data to SD card
unsigned int screenTimeoutSeconds = 60;       //Interval in seconds (default) for OLED automatic shutdown
unsigned int timeSynchHours = 24;             //Interval to for automatic RTC time synchronization
DateTime lastMeasurement;                     //DateTime object (unix time) indicating when the last measurement was taken
DateTime lastRecord;                          //DateTime object (unix time) indicating the last time measurement was saved to SD
DateTime screenTimeout;                       //DateTime object (unix time) used for determining if it is time to shut off OLED
DateTime lastNTPsynch;                        //DateTIme object (unix time) indicating last NTP synch attempt
String logFile="NONE.TXT";                    //Default logfile name. Automatically changes when first record is taken
long SDwriteCount = 0;                        //Counter indicating the number of measurements in the current logfile
int MaxOLEDscreens = 1;                       //Number of OLED display screens - updated after reading setting file.
int displayScreen = 0;                        //Current display screen on OLED display
char *dotToken=NULL;                          //pointer for seperating setup file data
char *csvToken=NULL;                          //pointer for seperatign setup file data
char dotSep[2] = ".";                         //token for splitting string by '.'
char comSep[2] = ",";                         //token for splittign string by ','
bool SDmissing = false;                       //Flag for missing SD card during startup
bool RTCfail = false;                         //Flag for RTC failure during startup
float batteryVoltage=0.0f;                    //Battery/charger circuit voltage in VDC
float USBVoltage=0.0f;                        //USB external DC input voltage in VDC
bool useDHCP = false;                         //Flag set if DHCP is to be used
bool slaveConnectionSuccess = false;          //Flag set if last MODBUS slave data poll was sucessfull
bool autoRTCsynch = false;                    //Flag set by settings if automatic time synch is desired
const int NTP_PACKET_SIZE= 48;                //NTP time stamp is in the first 48 bytes of the message
byte packetBuffer[NTP_PACKET_SIZE];           //buffer to hold incoming and outgoing packets
char timeServer[24] = "time.nist.gov";        //User defined (from settings) time server.
unsigned int localPort = 8888;                //Local UDP port for NTP synch
EthernetUDP Udp;                              //Ethernet UDP instance


//Modbus data structure built from setting file & populated by MODBUS data requests.
struct ModPoint{                              //Structure to hold data points and data
  char pName[36] = "";                        //Display name for the data point
  float value = 0.0f;                         //Current value of the data point
  char units[6] = "";                         //Units for modpont
  bool flipLSB = true;                        //Flip LSB/MSB for floating point values
  int functionCode = 3;                       //MODBUS function code
  int address = 0;                            //Data register address (DECIMAL)
  int registerCount = 2;                      //Number of registers to read
  struct ModPoint *next = NULL;               //pointer to next data point (Linked-list)
};

//Linked list pointers
struct ModPoint* head = NULL;                 //Pointer to head of Datapoint list
struct ModPoint* last = NULL;                 //Pointer to tail of Datapoint list;
struct ModPoint* dataDisplay = NULL;          //Pointer to first data display screen.

/************************************
 *  Initialize device & system Variables
 *    This function only runs once during inital powerup
 *************************************/
void setup() {
  //Initialize Arduino Pins
  pinMode(Bpin, INPUT_PULLUP);                    //Assign B Button on OLED as input
  pinMode(Cpin, INPUT_PULLUP);                    //Assing A Button on OLED as input
  pinMode(SDCardIn, INPUT_PULLUP);                //Assing SD card detect on Feather as input
  pinMode(SDwriteLED, OUTPUT);                    //Utilize the onboard LED to show SD card activity
  pinMode(MODMEASURE, OUTPUT);                    //LED to indicate polling modbus slave.
  pinMode(CONNECTED, OUTPUT);                     //LED to indicate slave is connected.

  //Enable serial port if DEBUG is defined
  #ifdef DEBUG
    Serial.begin(9600);                           //Enable serial port if in DEBUG mode
    while (!Serial){;}                            //Needed for native USB only
  #endif

  //Attempt to connect to RTC
  if (! rtc.begin()) {                            //Check if RTC is present
    #ifdef DEBUG
      Serial.println("Couldn't find RTC");        //DEBUG - Note RTC is 'missing'
    #endif
    RTCfail = true;                               //Set failed RTC flag
  }

  //Initialize SD Card
  SD.begin(SDcs);

  // set date time callback function for SD card files
  SdFile::dateTimeCallback(dateTime);

  //Read device settings from SD Card & build operating template
  //If IP address is set to 0.0.0.0 in device config continue with DHCP
  readDeviceSettings();

  //Initialize and Clear OLED display
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);      //Initialize with the I2C addr 0x3C (for the 128x32)
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  //Display Welcome Screen
  display.print("MODBUS LOGGER ");
  display.println(ver);
  display.println("by JAMES M. OLSON");
  display.println("UNIV OF AL at BHM");
  display.display();
  delay(2000);

  //Connect to network using DHCP or Static IP
  if(useDHCP){
    //Start Ethernet Service using DHCP if config file IP address was 0.0.0.0
    display.clearDisplay();
    display.setCursor(0,0);
    display.println("DHCP:Aquiring IP...");
    display.display();
    delay(1500);
    if(Ethernet.begin(mac) == 0)         //DCHP Failed. start using default IP address
    {
      display.clearDisplay();
      display.setCursor(0,0);
      display.println("DHCP Failed..");
      display.println("Using Default IP");
      display.display();
      delay(1500);
      DeviceIP[0] = 192;
      DeviceIP[1] = 168;
      DeviceIP[2] = 1;
      DeviceIP[3] = 10;
      ip[0] = DeviceIP[0];
      ip[1] = DeviceIP[1];
      ip[2] = DeviceIP[2];
      ip[3] = DeviceIP[3];
      Ethernet.begin(mac,ip,gateway,subnet);
    }else
    {
      splitDHCPip(Ethernet.localIP());
      display.println("IP from DHCP !");
      display.print("    ");
      display.print(DeviceIP[0]);
      display.print(".");
      display.print(DeviceIP[1]);
      display.print(".");
      display.print(DeviceIP[2]);
      display.print(".");
      display.println(DeviceIP[3]);
      display.display();
      delay(2000);
    }
  }else
    Ethernet.begin(mac, ip, gateway, subnet);     //Start Ethernet Service with fixed IP set in config file

  //Start Web server
  server.begin();

  //If RTC has lost power or if B&C buttons are held down during startup
  // Launch the NTP time synch subroutine.
  if (rtc.lostPower()||(!digitalRead(Bpin)&&!digitalRead(Cpin))) {
    synchRTC(false);      //Do slow update RTC synch with user display
  }

  //Initialize MODBUS Holding Registers to zero
  Mb.MbData[0] = 0;
  Mb.MbData[1] = 0;

  //Assign interrupts for panel buttons
  attachInterrupt(digitalPinToInterrupt(Bpin), changeDisplayScreen,FALLING);  //Assign interrupt to B Button for display change
  attachInterrupt(digitalPinToInterrupt(Cpin), changeRecordState, FALLING);   //Assing interrupt to C Button for record enable/disable

  //initialize all Datetime objects to power-up time
  lastMeasurement = rtc.now();                                //Time of last measurement
  lastRecord = rtc.now();                                     //Time of last record to SD card
  screenTimeout = rtc.now();                                  //Screen timeout timestamp
  lastNTPsynch = rtc.now();                                   //Assume synch on power-up

}

/**********************
 * Function splits IP address into individual bytes
 */
void splitDHCPip(IPAddress address)
{
    DeviceIP[0] = address[0];
    DeviceIP[1] = address[1];
    DeviceIP[2] = address[2];
    DeviceIP[3] = address[3];

}


/***********************
 *    INTERRUPT FUNCTION
 *      Sets flag to change display screen on OLED
 **********************/
void changeDisplayScreen()
{
  changeDisplay = true;
}

/***********************
 *    INTERRUPT FUNCTION
 *      Sets flag to toggle record to SD state
 */
void changeRecordState()
{
  changeRecord = true;
}


/****************************
 *  MAIN PROGRAM LOOP
 */
void loop() {
  EthernetClient client = server.available();             //Check Ethernet port to see if there is a webpage request
  DateTime now = rtc.now();                              //Record the current time for delta measurements

  //Go to error screens if boot failure
  if(displayScreen == -1){
    updateDisplay(now);                                  //error during startup. launch display.
    delay(1000);
  }

  //STEP 1 -  CHECK TO SEE IF CHANGE (SCROLL) DISPLAY INTERRUPT FLAG IS SET
  if(changeDisplay)
  {
    changeDisplay = false;                            //Reset change display flag set by interrupt

    if(hideScreen)                                    //if screen is hidden just wake it up, and dont scroll
      hideScreen = false;
    else                                              //screen is not hidden move forward one screen
    {
      hideScreen = false;
      //If at the last screen on the list roll back to the home screen
      if((displayScreen > MaxOLEDscreens) || (dataDisplay==last->next)){
        displayScreen = 0;
        dataDisplay = head;
      }else                                           //Not at last screen so scroll forward one screen
      {
         displayScreen+=1;
         //move display forward 4 data points
         //each data screen include 4 data points
         if(displayScreen>=4)
         {
           char i = 0;
           while(i<=3 && (dataDisplay != NULL))
           {
                dataDisplay = dataDisplay->next;
                i++;
           }
         }
      }//END SCROLL SCREEN ELSE
    }//END HIDE SCREEN ELSE

    screenTimeout = rtc.now();                        //RESET the display timout because a button was pushed
  }

  //STEP 2 - CHECK TO SEE IF RECORD FEATURE HAS BEEN ENABLED/DISABLED
  if(changeRecord)                                    //If change record state is set (via interrupt) change state
  {
    if(hideScreen){                                   //just wake up screen if it is currently hidden & dont toggle record
      hideScreen = false;                             //reset hide screen flag
      changeRecord = false;                           //reset change record flag
    }else
    {
      changeRecord = false;                             //Reset change record selection flag set by interrupt
      hideScreen = false;                               //'Wake up' OLED if it was shut off
      if(!recordData)                                   //If record state is currently off, set flag to create new logfile
        createNewLogFile = true;
      recordData = !recordData;                         //Toggle record state
    }
    screenTimeout = rtc.now();                          //Reset the display timeout because a button was pushed
  }


  //STEP 3 - Check if measurement interval has been exceeded. If it has, poll the MODBUS slave
  // for data and update data structure.
  if(now.unixtime() >= (lastMeasurement.unixtime() + measureIntervalSeconds))
  {
      lastMeasurement = rtc.now();                      //record time of measurement
      digitalWrite(MODMEASURE, true);                   //turn on the measurement LED
      getMODBUSdata();                                  //launch measurement subroutine
      digitalWrite(MODMEASURE, false);                  //turn off measurement LED
  }

  //STEP 4 - Determine if its is time to record measurement to SD card. If interval has been met, the record
  //         data option is set, and the connection to the slave was sucessful, record the datapoint.
  if((now.unixtime() >= (lastRecord.unixtime() + recordIntervalSeconds))&& recordData && slaveConnectionSuccess)
  {
    lastRecord = rtc.now();                             //record time for current record to SD
    recordMeasurement = true;                           //set flag to record data
  }


  //STEP 5 - Determine if it is time to 'shut of' display. If not time to hide, update screen.
  if((now.unixtime() >= (screenTimeout.unixtime() + screenTimeoutSeconds)) && !hideScreen)
  {
    hideScreen = true;
    display.clearDisplay();
    display.display();
  }

  //If screen is not hidden due to timeout, update the screen
  if(!hideScreen)
    updateDisplay(now);


  //STEP 6 - Check to see if SD card is installed. if it is not, set flag to create a new log file.
  if(!createNewLogFile && !digitalRead(SDCardIn))
    createNewLogFile = true;
  //STEP 7 - If SD card is installed, time to record measurement is set, and record option is selected,
  //         write current measurement data to SD card.
  if(recordData && digitalRead(SDCardIn) && recordMeasurement)
    recordDatatoSD();

  //STEP 8 - Check if there is a web Client polling for data. If not, loop here back to begining of function
  if(client)
    sendWebData(client,now);

  //STEP 9 - Check to see if time synch is required
  if((now.unixtime() >= (lastNTPsynch.unixtime() + timeSynchHours*3600)) && autoRTCsynch)
  {
    synchRTC(true);                     //perform fast RTC synchronization
    lastNTPsynch = rtc.now();
  }

  delay(5);    //wait a short delay before starting next loop
}


/***********************************
 *   Record latest measurement to the SD Card
 */
void recordDatatoSD()
{
      recordMeasurement = false;                                  //Reset record measurement to SD flag set by interval

      //Create CSV data string to be written to SD card
      String dataString = "";                                     //CSV measurement data string
      String titleLine = "";                                      //CSV title line for file (only on top row)
      dataString += String(lastMeasurement.unixtime());         //first column of data is UTC UNIX time
      dataString += ",";                                          //add comma after first row
      titleLine += String(DeviceID);                              //first column of title string is the device ID
      titleLine += ",";                                           //add comma after first row
      //LOOP THROUGH LINKED LIST AND PRINT DATA
      struct ModPoint *link = head;                             //Create pointer to head of data list
      while(link != NULL)                                       //Loop through list and add data points to record
      {
        if(link->address != 0)                                  //If the modbus register address is zero. skip the blank line
        {
          dataString += String(link->value);                    //Add the recorded value to the datastring
          titleLine += String(link->pName);                     //Add the name of the value to the title string
          if((link->functionCode == 3 || link->functionCode == 4) && (strncmp(link->units,"NA",2)!= 0)){
            titleLine += "[";
            titleLine += String(link->units);                   //add units in [] after name
            titleLine += "]";
          }
          if(link->next != last){                               //Add commas if it there is more data to add
            titleLine += String(',');
            dataString += String(',');
          }
        }
        link = link->next;                                      //move to the next datapoint in the linked list
        if(link==NULL)break;                                    //if at the end of the list breake the while loop
      }//END OF DATA STEP WHILE LOOP

      //Create a new random log file name if createNewLogFile flag is set.
      if(createNewLogFile)
      {
        char attempt = 0;
        while(attempt<10){                                      //Create a random log filename
          logFile=String("LF");                                 //Filenames start with "LF"
          randomSeed(analogRead(0));                            //Seed random number generator with 'noise' on analog input 0
          char tmp[6];                                          //temp char for file digits with leading zero.
          sprintf(tmp,"%05d",random(0,99999));                  //Filename has 5 digit random number. include leading zero's
          //logFile+=String(random(0,99999));                   //Filename has 5 digit random number
          logFile+=String(tmp);
          logFile+=String((char)random(65,90));                 //end of filename is a random letter A-Z
          logFile+=".csv";                                      //file extension is .CSV
          if(!SD.exists(logFile))                               //if the random file name doesnt exsit break out of file gen loop
```

```
            break;
          else
            attempt++;                                          //file name exists. try again 9 more times.
        }//END FILE CREATION LOOP
        createNewLogFile = false;                               //Reset createNewLogFile flag
        SDwriteCount = 0;                                       //Reset the saved SD record count
        File dataFile = SD.open(logFile,FILE_WRITE);            //open new log file to write header line (title's)
        dataFile.println(titleLine);                            //write header line for new file
        dataFile.flush();
        dataFile.close();
      }

      //Open/Create file and write data line
      File dataFile = SD.open(logFile, FILE_WRITE);
      if (dataFile) {
          digitalWrite(SDwriteLED,1);                           //Illuminate the WRITTING to SD LED
          dataFile.println(dataString);                         //write line of recorded data to SD card
          dataFile.flush();
          dataFile.close();                                     //close file for writing
          digitalWrite(SDwriteLED,0);                           //Shut off the WRITITING to SD LED
          SDwriteCount++;                                       //increase the number of records in the record count
      }
}


/***************************
 * Create MODBUS request and send to MODBUS Slave to get remote data.
 *
 */
void getMODBUSdata()
{
  struct ModPoint *link = head;                 //Create link to the head of the data list
  slaveConnectionSuccess = false;               //reset sucessfull slave connection flag

  float tempVal = 0.0f;                         //temporary value to hold modbus float data.
  word b1[2];                                   //temp word for holding Modbus Data

  //Step through data points and request data from slave
  while(link!=last)
  {
      if(link->address == 0)
      {
        link = link->next;                      //skip blank lines with address = 0
        continue;
      }

        tempVal = 0.0f;                         //reset value of the temp variable
        //b1[0]=0;                                //reset word to zero
        //b1[1]=0;                                //reset word to zero
        Mb.MbData[0] = 0;                       //Clear local holding register
        Mb.MbData[1] = 0;                       //Clear local holding register

        digitalWrite(MODMEASURE,0);             //Blink measure light during measurement

        //Call function from MsgModbus Class to get data
        MB_FC fc;                                  //Modbus Function code enumerated list from MsgModbus
        fc = (MB_FC)(int)link->functionCode;       //Assing function code type to selected data item
        slaveConnectionSuccess = Mb.Req(fc,(int)link->address,(int)link->registerCount,0,SlaveIP);  //Request Data from Slave

        if(!slaveConnectionSuccess)             //If connection fails on first node, dont try for subsequent nodes.
          break;

        delay(100);                             //Wait 100ms to allow time for slave to respond
        Mb.MbmRun();                            //Read & Process data from Ethernet port
        digitalWrite(MODMEASURE,1);             //Blink measure light during measurement
        delay(10);

        //If data is only over a single 16-bit register. Assign the value
        if(link->registerCount == 1){
          link->value = (float)Mb.MbData[0];
        }else if(link->registerCount ==2 && (link->flipLSB == true))        //Flip LSB/MSB if selected
        {
          b1[0] = Mb.MbData[1];
          b1[1] = Mb.MbData[0];
          memcpy(&tempVal,&b1,sizeof(tempVal));
          link->value = tempVal;
        }else if(link->registerCount ==2 && (link->flipLSB == false))
        {
          b1[0] = Mb.MbData[0];
          b1[1] = Mb.MbData[1];
          memcpy(&tempVal,&b1,sizeof(tempVal));
          link->value = tempVal;
        }

        link = link->next;      //move to next data point;
   }//END STEP THROUGH DATA WHILE LOOP

    //slaveConnectionSucesss id determined by measurement. Toggle LED to show if connection was sucessfull
    digitalWrite(CONNECTED,slaveConnectionSuccess);     //SET LED status to show connected to slave;
}// END GET MODBUS DATA FUNCTION


/****************************
 *   Update OLED display
 */
void updateDisplay(DateTime now)
{
  display.clearDisplay();
  display.setCursor(0,0);


  //Display the desired screen based on user input.
  switch(displayScreen){
    case -1:                                            //BOOT ERROR SCREEN
```

```
        display.println("MODLOGGER-BOOT FAIL");
        display.print("RTC: ");
        if(RTCfail)
          display.println("FAIL");
        else
          display.println("OK");
        display.println(PrintDateTime(now));
        if(SDmissing){
          display.println("Cfg File Missing");
        }

    case 0:                                              //Default Display showing IP addresses & current device time
        display.print("MODLOGGER: ");
        display.println(DeviceID);
        display.print("IP : ");
        display.print(DeviceIP[0]);
        display.print(".");
        display.print(DeviceIP[1]);
        display.print(".");
        display.print(DeviceIP[2]);
        display.print(".");
        display.println(DeviceIP[3]);
        display.print("SIP: ");
        display.print(SlaveIP[0]);
        display.print(".");
        display.print(SlaveIP[1]);
        display.print(".");
        display.print(SlaveIP[2]);
        display.print(".");
        display.println(SlaveIP[3]);
        display.println(PrintDateTime(now));
        break;
    case 1:                                                 //Show SD card Data & logging status
        if(!digitalRead(SDCardIn))
          display.println("SD CARD: REMOVED");
        else
          display.println("SD CARD: INSTALLED");

        if(recordData)
          display.print("LOGGING: ENABLED");
        else
          display.print("LOGGING: DISABLED");

        display.println();

        display.print("FILE: ");
        display.println(logFile);
        display.print("RECORDS: ");
        display.println(SDwriteCount);
        break;
     case 2:                                                 //SHOW POWER SUPPLY DATA
        display.println("Supply/Battery Info");
        getDeviceVoltages();

        display.print("Source: ");
        if(USBVoltage>4.7)
          display.println("External DC");
        else
          display.println("Battery");

        display.print("BAT/CHG: ");
        display.print(batteryVoltage);
        display.println(" VDC");
        display.print("USB IN : ");
        if(USBVoltage>4.7)
          display.print(USBVoltage);
        else
          display.print("0.00");
        display.println(" VDC");
        break;
     default:                                            //DATA DISPLAY
        char i = 0;
        struct ModPoint* link = dataDisplay;
        while((i<=3) && (link!=last)){                            //Read/display 4 lines of data to display
          if(link == NULL)break;                                  //END OF LIST - BREAK

          if(link->address == 0 && link->functionCode == 0)       //Text ONLY tite line
          {
            display.println(String(link->pName));
          }
          else if(link->address == 0 && link->functionCode != 0)  //Blank line
          {
            display.println("");
          }
          else                                                    //Display data & Value
          {
            display.print(String(link->pName));
            display.print(" = ");
            if(link->functionCode == 1 || link->functionCode ==2)   //Status input (true/false input)
            {
                if(link->value >0)
                  display.println("ON");
                else
                  display.println("OFF");

            }else if(link->functionCode == 3 || link->functionCode ==4)   //holding register
            {
                display.print(link->value);
                if(strncmp(link->units,"NA",2)!=0){
                  display.print(" ");
                  display.println(link->units);
                }else
                  display.println();
            }
          }
            i++;
            link=link->next;                                      //move to next link
        }
   }//END SWITCH
  display.display();                                              //Display the built screen to OLED
}//END DISPLAY TO OLED FUNCTION


/****************************************
 *  Create web view and send to client
 *
 *****************************************/
void sendWebData(EthernetClient client,DateTime now)
{
  if (client) {
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;                            //flag set when blank line is recieved
    String vars;                                                  //String holding web request data (for parsing)
    String filename;                                              //Web file download request
    bool downloadFile = false;                                    //Flag set that download request was sent

    //While client is connected read data request & respond
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        vars.concat(c);                 //add port data to string.

        //if request included GET and fileanme then flag for download & strip off requested filename
        if(vars.startsWith("GET") && (vars.endsWith(".CSV")||vars.endsWith(".cfg")) && !downloadFile)
        {
          downloadFile = true;
          filename = vars;
          filename = filename.substring(filename.indexOf("/"),filename.length());
        }
        #ifdef DEBUG
          Serial.write(c);               //DEBUG - SEND WEB data traffic to Serial port
        #endif

        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply
        if (c == '\n' && currentLineIsBlank) {
          if(downloadFile){                   //download request recieved. Send file from SD card
            downloadFile = false;
            //START DOWNLOAD HTTP HEADER
            client.println(F("HTTP/1.1 200 OK"));
            client.println(F("Content-Type: application/octet-stream"));
            client.print(F("Content-disposition: attachment; filename='"));
            String tmp = filename;            //remove the forward slash in filename for download
            tmp.remove(0);
            client.print(tmp);
            client.println("'");
            client.println(F("Connection: keep-alive"));
            client.println();
            //END DOWNLAOD HTTP HEADED

            if(SD.exists(filename))         //verify the file exists. If it does, send to web client
            {
              File dl = SD.open(filename);
              while(dl.available()&&client.connected())
              {
                  client.write(dl.read());
              }
              dl.close();
            }
          }else                             //Standard HTTP request. Send WEB data
          {
```

```
          // send a standard http response header
          client.println(F("HTTP/1.1 200 OK"));
          client.println("Content-Type: text/html");
          client.println("Connection: close");          // the connection will be closed after completion of the response
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");

          //SECTION 1 - Device ID & Settings
          client.print("<h1>Modlogger - ");
          client.print(DeviceID);
          client.println("</h1>");
          client.print("<p>Device Time (UTC): ");
          client.print(daysOfTheWeek[now.dayOfTheWeek()]);
          client.print(" ");
          client.print(PrintDateTime(now));
          client.println("</p>");

          //SECTION 2 - Device Measurement Status
          client.print("<p>");
          client.print("Last Measurement (UTC): ");
          client.print(daysOfTheWeek[lastMeasurement.dayOfTheWeek()]);
          client.print(" ");
          client.print(PrintDateTime(lastMeasurement));
          client.println("</p>");

          //SECTION 3 - Interval settings
          client.print("<p>");
          client.print("Measurement Interval: ");
          client.print(measureIntervalSeconds);
          client.print(" second(s)");
          client.println("</p>");
          client.print("<p>Record Interval:      ");
          client.print(recordIntervalSeconds);
          client.print(" second(s)");
```

```
        client.println("</p>");
        client.print("<p>Display Timeout:      ");
        client.print(screenTimeoutSeconds);
        client.print(" second(s)");
        client.println("</p>");

        //SECTION 4 - SD CARD data - Only show if SD Card is installed.
        if(digitalRead(SDCardIn))
        {
            client.println("<p>SD Card: INSTALLED</p>");
            client.print("<p>Current SD Log File Name: ");
            client.print(logFile);
            client.println("</p>");
            client.print("<p>Number of Data Points Stored to currend Log File: ");
            client.print(SDwriteCount);
            client.println("</p>");
            client.print("<p>Configuration File: ");
            client.println("<a href='/config/modlog.cfg' target='_blank'>MODLOG.CFG</a></p>");
        }else
            client.println("<p>SD Card: REMOVED</p>");                              //Show this if SD card is removed

        //SECTION 5 - Show Device & MODBUS Slave Info
        client.print("<p>MODBUS Slave IP: ");
        client.print(SlaveIP[0], DEC);
        client.print(".");
        client.print(SlaveIP[1], DEC);
        client.print(".");
        client.print(SlaveIP[2], DEC);
        client.print(".");
        client.print(SlaveIP[3], DEC);
        client.println("</p>");
        client.print("<p>MODBUS Slave Status: ");
        if(slaveConnectionSuccess)
          client.print("CONNECTED");
        else
```

```
            client.print("NOT CONNECTED");
          client.println("</p>");

          //SECTION 6 - Device Supply Voltages & Power supply status
          getDeviceVoltages();
          client.println("<h2>Power Supply Data</h2>");
          client.print("<p>Current Power Source: ");
          if(USBVoltage <4.8)
            client.println(" Battery</p>");
          else
            client.println(" External DC</p>");
          client.print("<p>Battery/Charger Voltage: ");
          client.print(batteryVoltage);
          client.println("VDC</p>");
          client.print("<p>External DC Voltage: ");
          client.print(USBVoltage);
          client.print("VDC</p>");

          //SECTION 7 - MODBUS Last Measurement Data in unordered list
          struct ModPoint* link = head;
          client.println("<h2>Data Points from MODBUS Slave</h2>");
          client.println("<ul>");
          while(link!= last)
          {
              if(link->address == 0)  //heading
              {
                client.print("<h3>");
                client.print(String(link->pName));
                client.println("</h3>");
              }else
              {
                client.print("\t<pre>");
                client.print("<li>");
                client.print(String(link->pName));
                client.print(" = ");
              if(link->functionCode == 1 || link->functionCode ==2)   //Status input (true/false input)
              {
                  if(link->value >0)
                    client.print("ON");
                  else
                    client.print("OFF");

              }else if(link->functionCode == 3 || link->functionCode ==4)   //holding register
              {
                  client.print(link->value);
                  if(strncmp(link->units,"NA",2)!=0){
                    client.print(" ");
                    client.print(link->units);
                  }
              }
              client.print("</li>");
              client.println("</pre>");
            }
            link=link->next;
        }
        client.println("</ul>");

        //SECTION 8 - SHOW LIST OF FILES ON SD CARD
        client.println("<h2>SD Card Files  - Click Filename to Download</h2>");
        client.println("<table cellpadding=4 cellspacing=4 border=1>");
        if(digitalRead(SDCardIn))
        {
          File root;
          root = SD.open("/");
          File entry;
          root.rewindDirectory();
          client.println("\t<tr>");
          client.println("\t\t<th>Bytes</th>");
          client.println("\t\t<th>Filename</th>");
          client.println("\t</tr>");
```

```
            while(entry = root.openNextFile()){
                if(strstr(entry.name(),".CFG")!= NULL)        //skip over config file
                  continue;
                if(entry.isDirectory())
                  continue;                                   //skip over directories
                client.println("\t<tr>");
                client.print("\t\t<td>");
                client.print(entry.size());
                client.println("</td>");
                client.print("\t\t<td>");
                client.print("<a href='/");
                client.print(entry.name());
                client.print("' target='_blank'>");
                client.print(entry.name());
                client.print("</a>");
                client.println("</td>");
                client.println("\t</tr>");
                entry.close();
            } //END FILE LIST WHILE LOOP
            root.close();
          } //END FIE LIST IF SD CARD INSTALLED "IF"
          client.println("</table>");
          client.println("</html>");
        }//END STANDARD HTTP REQUEST ELSE

        break;

        }else if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
```

```
      }//END CLIENT AVAILABVLE "IF"
    }//END CLIENT CONNECTED WHILE LOOP

    delay(10);            //Give client time to recieve data
    client.stop();        //disconnect the client.
  }//END IF-CLIENT LOOP
}//END SEND WEB DATA FUNCTION


/*******************
 * Convert a string IP address to byte array with decimal IP address
 *  Used during setup to parse IP address string from config file
 */

void convertIPfromStr(String address,byte IP[])
{

  int i=0;
  char stringBuffer[64];
  address.toCharArray(stringBuffer,64);
  dotToken = NULL;
  dotToken = strtok(stringBuffer,dotSep);
  while((dotToken != NULL) && (i<=3)){
    IP[i] = (byte)atoi(dotToken);
    dotToken = strtok(NULL,dotSep);
    i++;
  }
}

/*******************
 * Convert a string MAC address to byte array with hex MAC address
 *   Used during startup to parse MAC address from config file
 */

void convertMACfromStr(String address,byte MAC[])
```

```
{

  int i=0;
  char stringBuffer[64];
  address.toCharArray(stringBuffer,25);
  dotToken = NULL;
  dotToken = strtok(stringBuffer,dotSep);
  while((dotToken != NULL) && (i<=5)){
    MAC[i] = strtol(dotToken,NULL,0);
    dotToken = strtok(NULL,dotSep);
    i++;
  }
}



/*******************
 * Add MODBUS POINT to mod list
 *  Used during startup to create data linked list from config file
 */
void createModPoint(String point)
{

    char strBuffer[64];
    point.toCharArray(strBuffer,64);

    char *data[6];
    int i=0;
    csvToken = NULL;
    csvToken = strtok(strBuffer,comSep);
    while((csvToken != NULL)&&(i<6)){
      data[i] = csvToken;
      csvToken = strtok(NULL,comSep);
      i++;
    }
```

```
   /* FORMAT FOR IMPORTED CSV DATA
    *
    *  data[0] = Name of modpoint
    *  data[1] = Flip LSB/MSB
    *  data[2] = MODBUS function code (1,2,3,4 supported for requests) 0 = display feature
    *  data[3] = MODBUS slave data register
    *  data[4] = Number of registers to read (span of value)
    *  data[5] = Units of datapoint
    */

   //create a link in the list
   struct ModPoint *link = (struct ModPoint*) malloc(sizeof(struct ModPoint));

   //Populate node with data
   strncpy(link->pName,data[0],16);
   if(atoi(data[1])>0)
    link->flipLSB = true;
   else
    link->flipLSB = false;
   link->functionCode = atoi(data[2]);
   link->address = atoi(data[3]);
   link->registerCount = atoi(data[4]);
   link->value = 0.0f;
   strncpy(link->units,data[5],6);
   link->next = NULL;

   if(head==NULL) {
      head = link;                //Empty list so make first link the head
      last = link;                //Empty list so make first link the last link
      dataDisplay = link;         //Set display to link to first data item
   } else {
      last->next = link;          //make the last link to the new node
   }
   last = link;                   //move last pointer to new end node
```

```
}




/*******************
 * Get battery/charger circuit and USB voltages from analog pins.
 *   If USB voltage is below 4.7V, assume USB port is disconnected,
 *   and set USB voltage to 0.00 VDC.
 */

void getDeviceVoltages()
{
    batteryVoltage = 0.0f;
    USBVoltage = 0.0f;
    batteryVoltage = analogRead(VBATPIN);
    batteryVoltage *= 2;                    // we divided by 2, so multiply back
    batteryVoltage *= 3.3;                  // Multiply by 3.3V, our reference voltage
    batteryVoltage /= 1024;                 // convert to voltage
    USBVoltage = analogRead(VUSBPIN);
    USBVoltage *= 2;                        // voltage is divided by 2, so mutiply back
    USBVoltage *= 3.3;                      // multiply by 3.3V reference votlage
    USBVoltage /=1024;                      // convert to voltage using bit rate
    if(USBVoltage<USBMINVOLT)               // if USB voltage < USB minimum voltage, set to 0
      USBVoltage = 0.00f;
}

/***********
 * Format time/date string to include leading zero's
 */
String PrintDateTime(DateTime t)
{
    char datestr[24];
    sprintf(datestr, "%02d/%02d/%04d %02d:%02d:%02d", t.month(), t.day(), t.year(), t.hour(), t.minute(), t.second());
    return String(datestr);
```

```
}


/*********************
 *  Callback function for setting correct time/date stamps on SD card files
 */
void dateTime(uint16_t* date, uint16_t* time) {
     DateTime nowdt = rtc.now();

     // return date using FAT_DATE macro to format fields
     *date = FAT_DATE(nowdt.year(), nowdt.month(), nowdt.day());

     // return time using FAT_TIME macro to format fields
     *time = FAT_TIME(nowdt.hour(), nowdt.minute(), nowdt.second());
}


/**********************
 *  Synchronize with NTP time server
 *    Used during startup if Manually initiated by holding B&C buttons down,
 *    or if RTC battery failed. Must be connected to internet source.
 *    Fast update is used for automatic re-sych. bypasses display & delays
 */
void synchRTC(bool fastUpdate)
{
    DateTime before;                            //Datetime before synch
    DateTime after;                             //Datetime after synch

    if(!fastUpdate)
    {
      //UPDATE DISPLAY TO SHOW SYNCRHONIZING
      display.clearDisplay();
      display.setCursor(0,0);
      display.println("RTC SYNCH TO NTP");
      display.print("TS:");
      display.println(String(timeServer));
      display.println("SYNCHRONIZING...");
      display.display();
      delay(3000);
      before = rtc.now();                      //Record the date/time before synch
    }

    Udp.begin(localPort);                             //open UDP on local port
    sendNTPpacket(timeServer);                        //Send NTP time request packet to timeserver
    delay(1000);                                      //give some time for packet response to arrive

    if ( Udp.parsePacket() ) {                        //Read NTP response packet
      // We've received a packet, read the data from it
      Udp.read(packetBuffer, NTP_PACKET_SIZE); // read the packet into the buffer

      //the timestamp starts at byte 40 of the received packet and is four bytes,
      // or two words, long. First, esxtract the two words:

      unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
      unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
      // combine the four bytes (two words) into a long integer
      // this is NTP time (seconds since Jan 1 1900):
      unsigned long secsSince1900 = highWord << 16 | lowWord;

      // Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
      const unsigned long seventyYears = 2208988800UL;
      // subtract seventy years:
      unsigned long epoch = secsSince1900 - seventyYears;

      rtc.adjust(DateTime(epoch));                       //Adjust RTC to new time

      if(!fastUpdate)                                   //Dont display results for fast update
      {
        after = rtc.now();                        //Record time after synch
    //DISPLAY RESULTS OF SYNCHRONIZATION
    display.clearDisplay();
    display.setCursor(0,0);
    display.println("NTP SYNCH COMPLETE");
    display.println("UTC (GMT) OLD/NEW");
    display.println(PrintDateTime(before));
    display.println(PrintDateTime(after));
    display.display();
    delay(5000);
   }
  }else                      //FAILED TO CONNECT TO SERVER. SET DEFAUT TIME
  {
   if(!fastUpdate){          //dont force time synch if fast update fails
    //DISPLAY FAILED TO SYNCH NOTIFICATION
    display.clearDisplay();
    display.setCursor(0,0);
    display.println("NTP SYNCH FAIL");
    display.println("SETTING GENERIC TIME");
    rtc.adjust(DateTime(2016, 1, 21, 3, 0, 0));
    after = rtc.now();
    display.println(PrintDateTime(after));
    display.display();
    delay(5000);
   }
  }
  Udp.stop();                //CLOSE UDP Port
}

/***************************
 * Create NTP Packet for time synchronization request
 */
unsigned long sendNTPpacket(char *address)
{
 // set all bytes in the buffer to 0
 memset(packetBuffer, 0, NTP_PACKET_SIZE);
```

```
  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  packetBuffer[1] = 0;     // Stratum, or type of clock
  packetBuffer[2] = 6;     // Polling Interval
  packetBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;

  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  Udp.beginPacket(address, 123); //NTP requests are to port 123
  Udp.write(packetBuffer, NTP_PACKET_SIZE);
  Udp.endPacket();
}

/************************
 *  READ DEVICE SETTINGS FROM SD CARD
 ************************/
//void readDeviceSettings(IPAddress *device,IPAddress *gw, IPAddress *sn)
void readDeviceSettings()
{
    File myFile;                            //Setup File instance
    int displayCounter = 1;
    String fileline;                        //Temp string to store reading data from file line at a time.


    //Check if modlog.cfg file exisits. If it doesnt exist exit function and
    //return to error screen '-1'. Config file must be present to load device.
    if (SD.exists("/config/modlog.cfg")) {
      SDmissing = false;
      if(!RTCfail)
        displayScreen = 0;          //If RTC is functioning & config file is present. proceed to first operation screen.
    }else
    {
      displayScreen=-1;
      SDmissing = true;
      return;
    }


    myFile = SD.open("/config/modlog.cfg");

    while(myFile.available())
    {
      fileline = myFile.readStringUntil('\n');

      //If line starts with ":" skip the line
      if(fileline.startsWith(":"))
        continue;

      //Load settings from DEVICE section
      if(fileline.startsWith("[DEVICE]"))
      {
        //Serial.println("START DEVICE SETTINGS");
        while(myFile.available())
        {

          fileline=myFile.readStringUntil('\n');
          if(fileline.startsWith("[END_DEVICE]"))break;

          //Parse Device settings
          fileline.trim();            //Remove whitespace
          if(fileline.startsWith("IP="))                     //Get device IP Address
          {
              fileline.remove(0,fileline.indexOf("=")+1);
```

```
            if(fileline.equals("0.0.0.0"))                //if zero IP address entered. defer to dhcp
              useDHCP=true;
            convertIPfromStr(fileline,DeviceIP);
            ip[0] = DeviceIP[0];
            ip[1] = DeviceIP[1];
            ip[2] = DeviceIP[2];
            ip[3] = DeviceIP[3];
        }else if(fileline.startsWith("SIP="))             //Get slave IP Address
        {
            fileline.remove(0,fileline.indexOf("=")+1);
            convertIPfromStr(fileline,SlaveIP);

        }else if(fileline.startsWith("MAC="))            //Get ethernet card MAC ID
        {
            fileline.remove(0,fileline.indexOf("=")+1);
            convertMACfromStr(fileline,mac);
        }else if(fileline.startsWith("GW="))             //Get default Gateway
        {
            fileline.remove(0,fileline.indexOf("=")+1);
            convertIPfromStr(fileline,DeviceGW);
            gateway[0] = DeviceGW[0];
            gateway[1] = DeviceGW[1];
            gateway[2] = DeviceGW[2];
            gateway[3] = DeviceGW[3];
        }else if(fileline.startsWith("SN="))            //Get subnet mask
        {
            fileline.remove(0,fileline.indexOf("=")+1);
            convertIPfromStr(fileline,DeviceSN);
            subnet[0] = DeviceSN[0];
            subnet[1] = DeviceSN[1];
            subnet[2] = DeviceSN[2];
            subnet[3] = DeviceSN[3];
        }else if(fileline.startsWith("ID="))            //Get device Name
        {
            fileline.remove(0,fileline.indexOf("=")+1);
```

```
              fileline.toCharArray(DeviceID,16);
          }else if(fileline.startsWith("MEASURE="))       //Get measurement interval in seconds
          {
              fileline.remove(0,fileline.indexOf("=")+1);
              measureIntervalSeconds = fileline.toInt();
          }else if(fileline.startsWith("RECORD="))       //get record to SD interval in seconds
          {
              fileline.remove(0,fileline.indexOf("=")+1);
              recordIntervalSeconds = fileline.toInt();
          }else if(fileline.startsWith("SCREEN="))        //get screen timeout interval in seconds
          {
              fileline.remove(0,fileline.indexOf("=")+1);
              screenTimeoutSeconds = fileline.toInt();
          }else if(fileline.startsWith("TIMESERVER="))    //get time server for RTC synch
          {
              fileline.remove(0,fileline.indexOf("=")+1);
              fileline.toCharArray(timeServer,24);
          }else if(fileline.startsWith("AUTOTS="))        //enable bit for automatic time synchronization
          {
              fileline.remove(0,fileline.indexOf("=")+1);
              int q = fileline.toInt();
              if(q>0)
                autoRTCsynch = true;
              else
                autoRTCsynch = false;
          }else if(fileline.startsWith("TSINTERVAL="))   //autotime synch interval in hours
          {
              fileline.remove(0,fileline.indexOf("=")+1);
              timeSynchHours = fileline.toInt();
          }
        }//END WHILE FILE AVAILABLE FOR DEVICE SETTINGS

      }else if(fileline.startsWith("[REGISTERS]"))        //START MODBUS REGISTER LIST
      {
        while(myFile.available())
        {
          fileline=myFile.readStringUntil('\n');
          fileline.trim();                                 //remove whitespace
          fileline.remove(fileline.length(),1);            //remove newline character.
          if(fileline.startsWith("[END_REGISTERS]"))break;
          createModPoint(fileline);
          displayCounter++;
          if(displayCounter%4 == 0)                        //Every 4 lines read increase display count
            MaxOLEDscreens++;

        }//END FILE AVAILABLE REGISTERS WHILE LOOP
      }//END REGISTERS IF
    }//END WHILE LOOP FOR FILE AVAIABLE
    myFile.close();                                       //All settings imported. Close file & return to setup
}
```