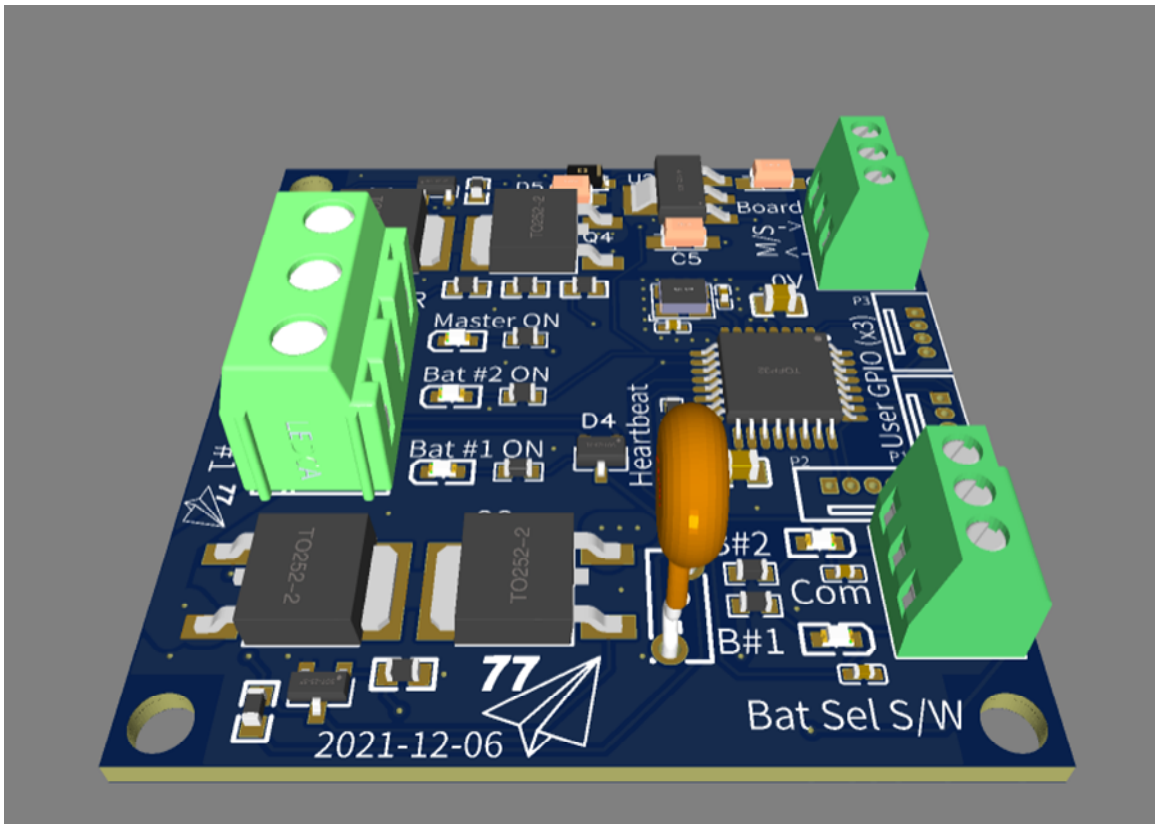


Glider Battery Management System

OPERATION AND MAINTENANCE MANUAL

David Nash | DDSC Airworthiness | Rev 0 (May 2022)

Contents

Introduction

This glider battery management system (GBMS) has the following functional features:

- very low power consumption (<20mA)
- traditional master switch function using a low current input primary circuit connected to the glider master switch (eg similar to a relay)
- traditional battery changeover/selection function using low current input primary circuits connected to the glider battery selection switch (eg similar to a relay)
- providing a reliable make-before-break battery changeover function eliminating the need for large capacitors (which are really bad for the glider electrics and don't always work) or hard (impossible?) to source changeover switches (eg make-before-break)
- automatic battery changeover/selection/management (if desired) is highly recommended for battery overall longevity.

Future options can be added with spare mcu GPIOs available (eg an OLED display showing the status of the batteries, power consumption, battery in use, etc).

Operation

MASTER SWITCH

The glider battery management system (GBMS) is connected to a traditional master switch.

The GBMS takes power from either battery (one with higher voltage) to perform its function.

When the power to the GBMS is disconnected (via the master switch), its built-in power MOSFETs (transistors) automatically isolate the power flow from the batteries (ie similar to a relay). This is fail safe.

When the master switch is turned on, the GBMS first function is to measure the installed glider battery voltages. If a particular glider battery hasn't been installed (eg battery 1 (only) and battery 2 is missing), the pilot may notice a short period of no power/even a brief blackout on startup. This is normal.

BATTERY CHANGOVER

The GBMS is connected to a traditional battery selector switch. This may be a toggle or rotary type. The GBMS looks for the control signal provided by the battery selector switch to provide the desired function (via computer logic)

For example, when battery 1 is selected, battery 1 MOSFETs are turned on and all other battery MOFETs are turned off. A similar sequence occurs when other batteries are selected.

When a battery is changed over via the battery selector switch (eg battery 1 to 2), the new battery selection (ie battery 2) MOSFETs are turned on, then after a very short delay (75mS), the old battery selection (ie battery 1) MOSFETs are turned off.

When the battery selector is in auto mode, the battery voltage (under load) is monitored, and the batteries are changed over to evenly discharge them

Note:

- if a battery is not fitted to the glider
    - the pilot may manually select this battery (ie the electrics will go out 😊),
    - however auto mode will not select the missing battery
- a faulty battery often has ok voltage with no-load which is why only the under-load voltage is used in the coded logic.
- the glider placarding will indicate where the 'auto' function is available.

Typical GBMS system installation. NOTE the AUTO position

Maintenance

INSTALLATION

All wiring shall conform to Basic Sailplane Engineering.

The wiring to/from the smaller signal terminals may be of a small size (eg awg24) as the currents carried by these are in the milli-amps. The GBMS mcu circuitry has a 300mA auto reset fuse and reverse polarity protection.

All batteries shall be fuse protected (as close to the battery as possible) in accordance with best practice as the 'power' circuits of the GBMS provide no protection.

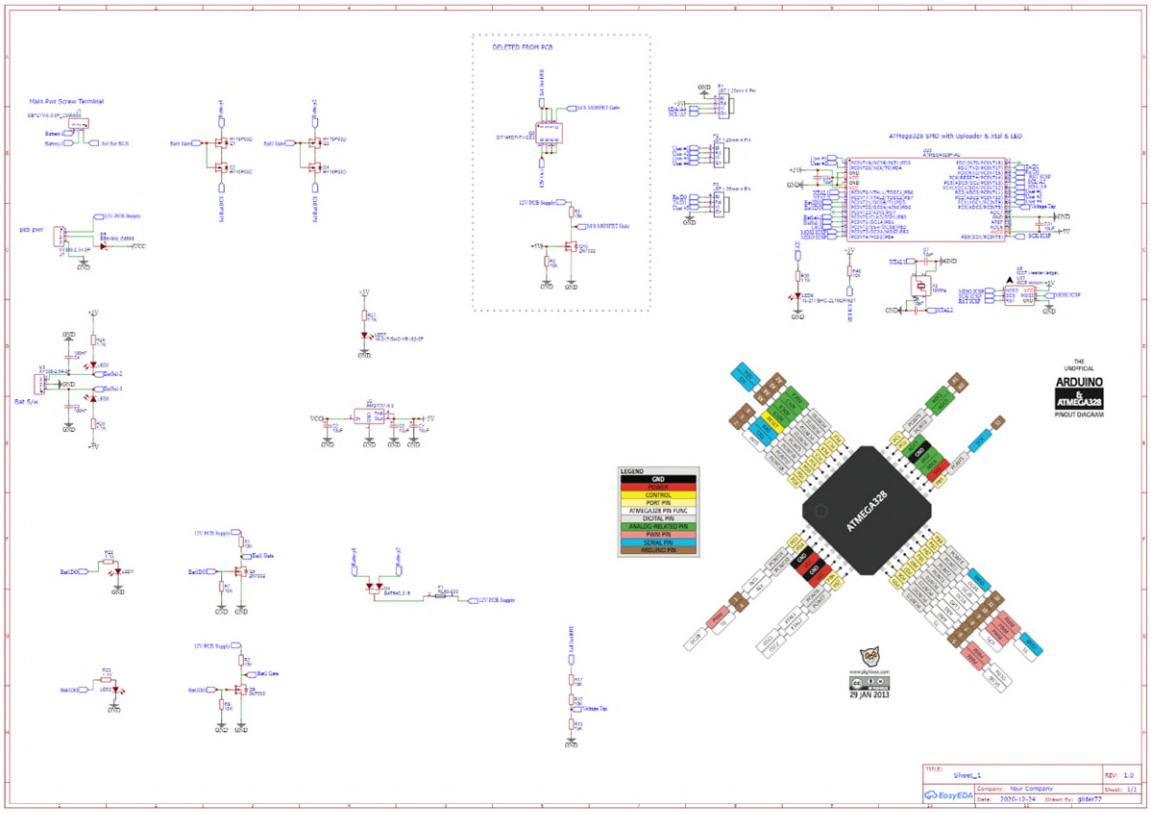
LEDS STATUS

The GBMS PCB has the following LEDs to indicate the status of:

- Master Switch – the GBMS has power at the mcu (micro control unit)
- Battery # (eg 1) ON – the MOSFET is powered ON (indicates the status of the mcu digital output is high – LED is ON).
- Battery Select # - which battery is selected (indicates the status of the mcu digital input is low – LED is ON.)
    - No LED status would indicate the battery selector switch is OFF (or in fault) and therefore the GBMS is in 'auto'.

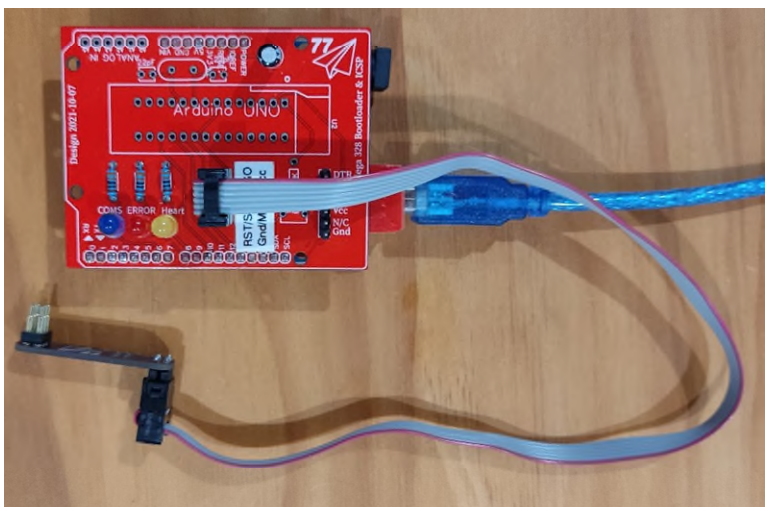
SCHEMATIC

The attached is the smc design for a 2 battery system. Other versions are in use



UPLOADING CODE

To change the on the GBMS, an ICSP (in circuit serial programmer) similar to the below is required. The ICSP interface is on the bottom of the pcb.



BOM

the below is the bill of materials for the included schematic

ID	Name	Designator	Footprint	Quantity	Manufacturer Part	Manufacturer	Supplier	Supplier Part
1	10uF	C1,C2,C5	CAP-SMD_L3.2-W1.6-R-RD	3	TAJA106K016RNJ	AVX	LCSC	C7171
2	100nF	C3,C4	C0402_NEW	2	CL05B104KO5NNNC	SAMSUNG	LCSC	C1525
3	10pF	C6,C7	C0402	2	CL05C100JB5NNNC	SAMSUNG	LCSC	C32949
4	10uF	C31,C32	C0805	2	CL21A106KAYNNNE	SAMSUNG	LCSC	C15850
5	BAT54C,215	D4	SOT-23-3_L2.9-W1.6-P1.90-LS2.8-BR	1	BAT54C,215	Nexperia	LCSC	C37704
6	B5819W_C8598	D5	SOD-123_L2.7-W1.6-LS3.7-RD-1	1	B5819W	CJ	LCSC	C8598
7	RL60-030	F1	FUSE-TH_L7.4-W3.0-P5.10-D0.6-S1.20	1	RL60-030	RUILON	LCSC	C21024
8	DB127V-5.0-3P_C395886	J1	CONN-TH_DB127V-5.0-3P	1	DB127V-5.0-3P	DIBO	LCSC	C395886
9	19-217/BHC-ZL1M2RY/3T	LED1,LED2,LED9	LED0603-R-RD	3	19-217/BHC-ZL1M2RY/3T	EVERLIGHT	LCSC	C72041
10	19-213/Y2C-CQ2R2L/3T(CY)	LED5,LED6	LED0603-R-RD	2	19-213/Y2C-CQ2R2L/3T(CY)	EVERLIGHT	LCSC	C72038
11	19-217/GHC-YR1S2/3T	LED7	LED0603-R-RD	1	19-217/GHC-YR1S2/3T	EVERLIGHT	LCSC	C72043
12	JST 1.25mm 4 Pin	P1,P2,P3	JST 1.25MM 4 PIN	3				AliExpress
13	HY19P03D	Q1,Q2,Q3,Q4	TO-252-2_L6.6-W6.1-P4.57-LS9.9-TL-CW	4	HY19P03D	HOOYI	LCSC	C122492
14	2N7002	Q5,Q6	SOT-23-3_L2.9-W1.3-P1.90-LS2.4-BR	2	2N7002	CJ	LCSC	C8545
15	10K	R1,R2,R7,R9,R40	R0603	5	0603WAF1002T5E	UniOhm	LCSC	C25804
16	15K	R12,R13,R17	R0603	3	0603WAF1502T5E	UniOhm	LCSC	C22809
17	1.1K	R22,R23,R24,R25,R26,R27	R0603	6	0603WAF1101T5E	UniOhm	LCSC	C22764
18	XY308-2.54-3P	U1,U3	CONN-TH_XY308-2.54-3P	2	XY308-2.54-3P	--	LCSC	C557686
19	AMS1117-5.0	U2	SOT-223_L6.5-W3.5-P2.30-LS7.0-BR	1	AMS1117-5.0	AMS	LCSC	C6187
20	ISCP Header (edge)	U8	ICSP (EDGE)	1				
21	ATMEGA328P-AU	U26	TQFP-32_L7.0-W7.0-P0.80-LS9.0-BL	1	ATMEGA328P-AU	MICROCHIP	LCSC	C14877
22	16MHz	X3	OSC-SMD_4P-L3.2-W2.5-BL	1	X322516MLB4SI	YXC	LCSC	C13738

CODE

The code for the 2 battery GBMS (as of 20220511) is as follows:

```
/*
* 2 Battery Changover switch using ATMega328 and other smd including MOSFETS. A few thru hole remain
* 3 position changeover switch with Auto Mode using ATMega328 smd
* derived for 3 battery code
 *


 * Includes battery Voltage measurement and this is used in the Auto mode.  Each battery is auto selected on startup (so a voltage measurement is stored), before Auto is activated.
 * pcb jumer enable auto mode
 *
 * Battery Changeover Switch for a glider with the following features
 *    - Make before Break ensuring continuity of power to the instruments on duty battery changeover
 *    - CODE for 2 batteries
 *
 *    NOTE:    External (to arduino) pullup resistors (10k ohm) are required to be wired into the MOSFET gate cicuit.  This ensures that the MOSFETS are OFF (N/O) before MCU is active (ie unpowered) - failsafe.
 *
 *    - A full current master/isolator MOSFET has been also installed thus eliminating the need for a separate full-load master switch (of coarse one can be installed if required).  The power supply for this board (12V) is switched by the Master via a dedicated circuit.
 *
 *    - LED heart-beat (blinking every 7 seconds).  Battery Selection confirmation via the LED at each input and output
```

```
 *
 *
 *
 * By David Nash Oct 2021
*/

#include <avr/wdt.h> //Watch dog timer functions

//    #define DEBUG   //Comment out this line to (AUTO) stop ALL Serial.print (aka DEBUGPRINT(x) etc)
 //    ************************************************************

#ifdef DEBUG
#define DEBUGPRINT(x) Serial.print(x)
#define DEBUGPRINTLN(x) Serial.println(x)
#define DEBUGDELAY(x) delay(x)
#else
#define DEBUGPRINT(x)
#define DEBUGPRINTLN(x)
#define DEBUGDELAY(x)
#endif
```

```
#define makeBreakDelay  75    // battery changeover make before break delay time (excludes MOSFET reaction) in milliseconds (ie 100 = 0.1 seconds)

#define switchDelay     500   // defines time the battery change switch needs to read steady-state before code actions (in milliseconds - eg prevents battery 2 being selected when swithing 1 to 3).  Excludes OFF - see 'offDelay'

//#define offDelay        2000  // defines time the battery change switch needs to read steady-state OFF before code actions (in milliseconds -NOT USED ANYMORE


#define LED             10  // heartbeat LED connection

#define select1         8  //Input Pin to use battery 1 - ie Battery Changeover Switch   Layout of PCB doesn't match schematic labelling

#define select2         9  //Input Pin to use battery 2


#define mosfetGate1     6  // Output Pin to switch battery 1 MOSFET - Layout of PCB doesn't match schematic labelling

#define mosfetGate2     5  // Output Pin to switch battery 2 MOSFET



#define VoltTap         A0   // via Voltage tap (R1 & R2 & R3) to ATMEGA328


#define User#1          A3  // SPARE via JST connector

#define User#2          2  // SPARE via JST connector

#define User#3          A2  // SPARE via JST connector

#define User#4          A1  // SPARE via JST connector
```

```
#define User#5         3  // SPARE via JST connector


volatile int battery = 0; // stores currently selected battery state

volatile int prevBattery = 0; // stores previous selected battery state

volatile int prevSwitch =0;  // stores previous selected switch state



volatile unsigned long lastBlinks; float blinkCount;  //for Blink fn



// Auto Battery Code

float minBatThreshold= 10500; //Voltage threshold to instantly do something in Auto (mV) - ie selected battery is U/S

int autoPeriod  = 10; //secounds between Auto switch function being executed

int Volts_PV_Period = 1; //secounds between auto voltage reads

float voltageDividerRatio = 3.009; // voltage measured to calc (manually) actual value .  Use 3.0 if unknown

float reverseCurrentDiode = 0.0;  // reverse current diode delta V.  Added to measured volts as the pcb & glider gnd differ by this value



volatile int BatVoltage[3]={0,15000,15000};  // use 'battery #' as array id -therefore need 3 (for 2 bat).  No battery 0 (so 0.0V).  Actual values obtained in setup();


volatile unsigned long autoNow;
```

```
volatile unsigned long SwitchNow;
volatile unsigned long voltsNow;



void setup() {

wdt_disable(); //Datasheet recommends disabling WDT right away in case of low probabibliy event

Serial.begin(9600);       // Serial print baudrate - debugging tool

pinMode (LED, OUTPUT);



pinMode (select1, INPUT);
pinMode (select2, INPUT);



pinMode (mosfetGate1, OUTPUT);
pinMode (mosfetGate2, OUTPUT);
```

```
pinMode (VoltTap, INPUT);



digitalWrite (mosfetGate1, LOW);  // Switches off Battery 1 - initialisation.  An external (to arduino) pullup resistors (10k ohm) is required to be wired into the cicuit.
digitalWrite (mosfetGate2, LOW);  // Switches off Battery 2 - initialisation.  An external (to arduino) pullup resistors (10k ohm) are required to be wired into the cicuit.



//Select 1 option below for WDT (watchdog)
  setWDT(0b01000000); //set for interrupt
 // setWDT(0b00001000); //set for reset
  //setWDT(0b01001000); //set for interrupt and then reset

    // initialise battery voltage readings
    battery = 2;  prevSwitch  =  2;          //start with battery 2 incase it isn't fitted.  Then pilot will see a short delay on start up rather than a short blackout
    MOSFET_switching();
    delay(500);

    wdt_reset(); //Reset the WDT
    VoltageCalcs();
    battery = 1;  prevSwitch  =  1;
    MOSFET_switching();
```

```
    delay(500);
    VoltageCalcs();

voltsNow  = millis(); SwitchNow = millis(); autoNow=millis();
}



void loop()
{
    wdt_reset(); //Reset the WDT

    if (prevSwitch  !=  battery)
    {
      VoltageCalcs();
      SwitchNow = millis();
      prevSwitch = battery;
        while(millis() - SwitchNow < switchDelay)
        {
          digitalWrite(LED,HIGH);
        }
```

```
  prevSwitch = battery;
  batterySwitchSelection();
  if (prevSwitch  ==  battery  && prevBattery  != battery)
    {
      MOSFET_switching();
      wdt_reset(); //Reset the WDT
      VoltageCalcs();
      digitalWrite(LED,LOW);
    }
}
else
{
  blinkLED(5,2,500);
  batterySwitchSelection();
}

if ( millis() - voltsNow  > (Volts_PV_Period *1000) &&  prevSwitch == battery)
{
                        DEBUGPRINTLN("called Volt Read Fn ");
  VoltageCalcs();
  voltsNow  = millis();
```

```
    //                  DEBUGDELAY(500);    //debug
   }



//   DEBUGPRINT("Select State  :"); DEBUGPRINT(selectState1); DEBUGPRINT("\t "); DEBUGPRINT(selectState2); DEBUGPRINT("\t "); DEBUGPRINT(selectState3);
DEBUGPRINTLN("\t ");
  DEBUGPRINT("battery  :"); DEBUGPRINTLN(battery); DEBUGPRINTLN("\t ");
  DEBUGPRINT("prevBattery  :"); DEBUGPRINTLN(prevBattery); DEBUGPRINTLN("\t ");
  DEBUGPRINT("prevSwitch  :"); DEBUGPRINTLN(prevSwitch); DEBUGPRINTLN("\t ");
  DEBUGDELAY(100); // debug  */



}



void blinkLED (int period, int blinkNumber, int pulseDelay)   //period in seconds, blinkNumber per period. pulseDelay between flashs (eg 100 milliseconds)
{
    // NEED TO GLOGALLY DECLARE "    unsigned long lastBlinks; float blinkCount;    "  PLUS LED pinMode GPIO etc
    // using these parameters is a nice 'strobe' blinkLED(5,2,250);
    // too many DEBUGPRINT commands will impact the 'timing' thus thhe ops of theis fn
```

```
if (millis()- lastBlinks > period*1000 ) // delay of 'period' sec (millisecs)
 {
    lastBlinks = millis();  // reset time and thus Blink sequence
    blinkCount  = 1.0;                    //counter reset
 }
 if (blinkNumber < blinkCount)     //resets counter after required blinkNumber achieved...
 {
 delay(1);     // blinks done- waiting for blink reset above
 }
 else
 {
     float i = (pulseDelay* (blinkCount  - 0.0));
     float j = (pulseDelay* (blinkCount  + 0.5));



     if (millis()-lastBlinks <  i)
     {
         digitalWrite(LED, LOW); // set the LED on
```

```
            }

            else if (millis()-lastBlinks <  j)
            {
               digitalWrite(LED, HIGH); // set the LED on

            }
            else
            {
              blinkCount++;
              digitalWrite(LED, LOW); // set the LED on

            }
          }

 return;
}



void MOSFET_switching()
```

```
{
 switch(battery)
 {
  /* case 0:

    digitalWrite (mosfetGate2, LOW);  // Switches off Battery 2
    digitalWrite (mosfetGate1, LOW);  // Switches off Battery 1
    delay (makeBreakDelay);              // short delay (milliseconds)

    prevBattery=0;               // stores the value

   break;  */


   case 2:
    digitalWrite (mosfetGate2, HIGH);  // Switches on Battery 2 - wired pullup resistors
    delay (makeBreakDelay);              // short delay (milliseconds)
    digitalWrite (mosfetGate1, LOW);  // Switches off Battery 1
    delay (makeBreakDelay);              // short delay (milliseconds)
```

```
    prevBattery=2;                // stores the value

   break;



   default:    // // Switches on Battery 1  (No Case 1) essentially and a catch-all for other inputs(??).

    digitalWrite (mosfetGate1, HIGH);  // Switches on Battery 1 - wired pullup resistors
    delay (makeBreakDelay);              // short delay (milliseconds)
    digitalWrite (mosfetGate2, LOW);  // Switches off Battery 2
    delay (makeBreakDelay);              // short delay (milliseconds)



    prevBattery=1;                // stores the value



   break;
  }
return;
}
```

```
void AutoMode()     // called in Selectedbattery() subrountine
{
  DEBUGPRINTLN("---> AutoMode Fn started ...");
  if( millis() - autoNow > autoPeriod*(1000) || BatVoltage[1]<minBatThreshold || BatVoltage[2]<minBatThreshold  )
  {
     //DEBUGPRINTLN("Inside 1st loop - Auto Bat if/else");

     if(BatVoltage[1]>BatVoltage[battery])
      {
        //switch to battery 1
        battery=1;

        DEBUGPRINTLN(F("Auto Bat 1"));
      }
     else if(BatVoltage[2]>BatVoltage[battery])
      {
        //switch to battery 2
        battery=2;
```

```
        DEBUGPRINTLN(F("Auto Bat 2"));
      }


     DEBUGPRINT("AutoMode fn returns Battery :"); DEBUGPRINTLN(battery);
    autoNow=millis();
 }
 return battery;
}



void VoltageCalcs()
 {
    for (int j=0; j<30; j++)    // initial analog read are inaccurate
    {
     int i = map(analogRead(VoltTap),0,1023,0,5000) + reverseCurrentDiode *1000;

       BatVoltage[battery] = i * voltageDividerRatio;    //converts AI to actual 'selected' battery volts
```

```
        DEBUGPRINT("Selected BatVolt (raw) V(mV) : ");DEBUGPRINTLN(i);
      }
      DEBUGPRINT("Selected BatVolt (corrected mV) : ");DEBUGPRINTLN(BatVoltage[battery]);
      DEBUGPRINT("all Bat V: ");  DEBUGPRINT(BatVoltage[1]); DEBUGPRINT(" ");  DEBUGPRINT(BatVoltage[2]); DEBUGPRINTLN(" ");
}



void batterySwitchSelection()
{

int switchbinary = 2*digitalRead(select1) + digitalRead(select2) ;

DEBUGPRINT("switchbinary  :");DEBUGPRINTLN(switchbinary);
  /*
   *        bat s/w (NOTE SELECTED bat is pulled LOW)
   *       #1 #2
   * --|----------
   *        1 1  = Auto   = 3 binary
   *        0 1  = #1     = 1 binary
   *        1 0  = #2     = 2 binary
   *
```

```
*/
 switch (switchbinary)
{
  case 1:   //manually switch to battery 1 with AutoBat OFF
    battery = 1;
  break;

  case 2:   //manually switch to battery 2  with AutoBat OFF
    battery = 2;
  break;


  case 3:   // switch off = Automode
    AutoMode();
   break;

  default:
    AutoMode();
  break;
}
```

```
return;
}

//this function setups up and starts the watchdog timer
void setWDT(byte sWDT) {
  WDTCSR |= 0b00011000;
  WDTCSR = sWDT |  WDTO_2S; //Set WDT based user setting and for 2 second interval
  wdt_reset();
}


//This is the interrupt service routine for the WDT. It is called when the WDT times out and is in interrupt mode.
//This ISR must be in your Arduino sketch or else the WDT will not work correctly
ISR (WDT_vect)
{
  ISR_LED_blink();      //this shouldn't run if things are working properly

  //wdt_disable();
}  // end of WDT_vect

void ISR_LED_blink()
```

```
{
 digitalWrite(LED, HIGH);
 delay(1000);
 digitalWrite(LED, LOW);
 delay(1000);
 digitalWrite(LED, HIGH);
 delay(1000);
 digitalWrite(LED, LOW);
 delay(1000);
}
```