```
/*****************************************************************************
   RockOn 2016 Workshop Flight Code
   Authors:
   Emily Logon, Dylan Herron, David Baird, Jared Stanley, Zak Hall
   Tess Geiger, Rafael Macias, Chris Koehler

   Revision 44.0.0
   -Remove flash
   -Rework SD card code to be simpler
   -Ground Mode uses serial print and writes to SD Card
   -Flight Mode only writes to SD card and does not serial print to increase
   Revision by: Jesse Austin
   Date: 3/18/16, 4/29/16, 5/2/16
 *****************************************************************************/

// [SENSOR LIST]
/*
   Geiger    (Interrupt 5 on Pin 18)
   SD Card   (Chip Select Pin: 53)
   Pressure  (I2C Bus Address: 0x77)
   Gyro      (I2C Bus Address: 0x69)
   3g Accelerometer
   16g Accelerometer
   50g Accelerometer
   Humidity
*/

// MAIN PROGRAM ////////////////////////////////////////////////////////////
// Libraries //
// The libraries below are needed so
// that we can use functions/variable definitions defined in them.

// The included Arduino libraries installed in main Arduino library folder
   #include <SPI.h>   // Serial Peripheral Interface (SPI) communication libr
   #include <Wire.h>  // Inter Integrated Circuit (I2C) communcation libary f
   #include <SD.h>    // Secure Digital (SD) function library for the SD card

// Global Variables //
   const int LEDA = 48; //defined in blink_pattern_code.h
   const int LEDB = 49; //defined in blink_pattern_code.h
   const int PROG = 47;
   boolean groundMode = 0;

// The included Space Shield libraries installed in sketch folder
   #include "gyro.h"        //Gyroscope sensor library
```

```
   #include "bmp.h"         //Barometric Pressure Sensor library
   #include "startTasks.h"
   #include "SDCard.h"

// [ACCELEROMETERS]
   // Low Accelerometer (3g) Pin Definitions
   const int ACCL_X = A3;  // X-axis on analog pin A3
   const int ACCL_Y = A4;  // Y-axis on analog pin A4
   const int ACCL_Z = A5;  // Z-axis on analog pin A5
   // Medium Accelerometer (16g) Pin Definitions
   const int ACCM_X = A0;  // X-axis on analog pin A0
   const int ACCM_Y = A1;  // Y-axis on analog pin A1
   const int ACCM_Z = A2;  // Z-axis on analog pin A2
// High Accelerometer (50g) Pin Definition
   const int ACCH_Z = A6;    // Z-axis on analog pin A6

// [HUMIDITY_SENSOR]
   // Humidity sensor on analog pin A8
   const int HUM_SENS = A8;

// [GYROSCOPE]
   // variables for Gryo X,Y,Z axes
   int gyX;
   int gyY;
   int gyZ;
   const int gyX_offset = 0; //set to X offset value for your board
   const int gyY_offset = 0; //set to Y offset value for your board
   const int gyZ_offset = 0; //set to Z offset value for your board

// [PRESSURE-TEMPERATURE]
   // variables for Pressure and Temperature
   short tempur;
   long presur;

// [GEIGER]
   // Geiger counter interrupts detected on pin 18
   const unsigned int gc_pin = 18;
   //  Attach the the 5th interrupt of the Arduino Mega.
   const unsigned int gc_intnumber = 5;
   // gc_cnt to store the number of counts
   // from gc_counts no counts are missed
   // while writing to buffer
   unsigned int gc_cnt;
   // Declared volatile because two threads of execution are using it.
   // Value is initially set to zero because there are no counts.
```

```
    volatile unsigned int gc_counts = 0;

    // Function for interrupt
    // gc_counts is increased by 1
    // every time function called.
    // void loop will reset when written memory
    void gc_interrupt() {
         gc_counts++;
    }

void setup() {
// [LED_MODES]
    // The built in "pinMode" function tells Arduino whether it should expect
    // to receive inputs or send outputs over different pins
    // LEDA, LEDB, and PROG defined in blink_pattern_code.h
    pinMode(LEDA, OUTPUT); // LEDA variable set to output
    pinMode(LEDB, OUTPUT); // LEDB variable set to output
    pinMode(PROG, INPUT); // PROG variable set to input

// [GROUND/FLIGHT_MODES]
    // If PROG is high, board is in ground mode, else flight mode
    if (digitalRead(PROG)) {
       // In Ground Mode
       // [SERIAL]
       // Starts serial (USB) communication at a 9600 baud rate
       Serial.begin(9600);
       Serial.println("Arduino Started in ground mode");
       // Indicates we are in Ground Mode
       groundMode = 1;
       digitalWrite(LEDA, HIGH);
    }
    else {
       //In flight mode. No Serial printing.
       groundMode = 0;
    }

// [ACCELEROMETERS_MODES]
    // Tells Arduino we expect to received data from these pins
    pinMode(ACCL_X, INPUT);
    pinMode(ACCL_Y, INPUT);
    pinMode(ACCL_Z, INPUT);
    pinMode(ACCM_X, INPUT);
    pinMode(ACCM_Y, INPUT);
    pinMode(ACCM_Z, INPUT);
    pinMode(ACCH_Z, INPUT);
```

```
// [HUMIDITY_MODE]
   pinMode(HUM_SENS, INPUT);

// [START_I2C]
   //References "Wire.begin" function in the wire.h library above.
   if (groundMode) Serial.println("START WIRE");
   // Go to StartTasks Namespace and start I2C communication for the Gyroscop
   // and Barometric Pressure sensors
   StartTasks::startI2C();

// [START_BMP]
   // Starts I2C Barometric Pressure Sensor
   if (groundMode) Serial.println("START BMP");
   // The "::" tells Arduino to go to the Bmp name-space,
   // call the "bmp085GetError" function,
   // and then return the current error recorded by the sensor.
   // Once we get the value for this error, we print it to the serial monitor
   if (groundMode) Serial.println(Bmp::bmp085GetError());
   // Again, we go to the Bmp namespace, call the "bmp085Calibration" functio
   // This tells the sensor to create variables for use in
   // pressure and temperature calculations (see bmp.h)
   Bmp::bmp085Calibration();

// [START_GYROSCOPE]
   // Starts I2C Gyroscope Sensor
   if (groundMode) Serial.println("Start Gyro");
   // The "::" tells Arduino to go to the Gyro name-space,
   // and call the "setupGyroITG" function.
   // This will configure the Gyroscope to sample at a rate of 100 Hz
   // and scale the outputs to +/- 2000 degrees per second
   Gyro::setupGyroITG();
   // Again, we go to the Gyro namespace, call the "itgRead" function.
   // This tells the sensor to send us its current I2C address
   // Print this address to the Serial Monitor
   if (groundMode) Serial.println(Gyro::itgRead(Gyro::itgAddress, 0x00));

// [START_SPI]
   // Start SPI communication for SD and Flash
   if (groundMode) Serial.println("START SPI");
   // Go to StartTasks Namespace and start SPI communication for the SD
   // and Flash sensors
   StartTasks::startSPI();

// [START_SD]
```

```
   // Start SD (Requires SPI to be started)]
   if (groundMode) Serial.println("START SD");
   pinMode(chipSelect, OUTPUT); // Set the Chip Select pin for the SD card to

// This function will set up the SD card so we can write to it.
   // Also the header of the log file will be written to the file.
   if (SDCardInit()) {
      if (groundMode) Serial.println("SD Card Initialized");
   }
   else {
      if (groundMode) Serial.println("SD Card Init Error");
   }

// [START_GEIGER]
   // Start Gieger Interrupt
   // Tells Arduino we expect to receive data from gc_pin
   pinMode(gc_pin, INPUT);
   // Sets up the interrupt to trigger for a rising edge
   // gc_pin (18) corresponds to the 5th interrupt gc_intnumber
   // gc_interrupt is the function we want to call once we detect an interrup
   attachInterrupt(gc_intnumber, gc_interrupt, RISING);

// Startup Complete
   if (groundMode) Serial.println("Startup Complete->Start Loop");
   if (groundMode) Serial.println(sensorNames);
}

void loop() {
// [LOG INTERVAL]
   //   Set your sample rate in SDCard.h
   //   This code does not use a log interval
   //   It logs data as fast as it can
   //   Longer flights might want to use a log interval
   //   delay(LOG_INTERVAL);

// [TIME STAMP]
   // millis returns number of milliseconds since program
   // started; writes this data to buffer
   // Clear data string before logging the time
   dataString = "";
   timeStamp = millis();
   dataString = String(timeStamp);

// [ACCELEROMETER_READ]
   // Read 3G X,Y,Z accels, store in buffer, update cursor locations
```

```
    dataString = dataString + ", " + String(analogRead(ACCL_X));
    dataString = dataString + ", " + String(analogRead(ACCL_Y));
    dataString = dataString + ", " + String(analogRead(ACCL_Z));
    // Read 16G X,Y,Z accels, store in buffer, update cursor locations
    dataString = dataString + ", " + String(analogRead(ACCM_X));
    dataString = dataString + ", " + String(analogRead(ACCM_Y));
    dataString = dataString + ", " + String(analogRead(ACCM_Z));
    // Read 50G Z accel, store in buffer, update cursor location
    dataString = dataString + ", " + String(analogRead(ACCH_Z));

// [PRESSURE_SENSOR_READ]
    if (Bmp::bmp085GetError() > 3) {
    // If error count is greater than 3, indicate sensor is malfunctioning.
    // Print 6 error bytes to replace the 6 Barometric Pressure data bytes
    // 2 bytes from temperature (data type short), 4 bytes from pressure
    // (data type long)
    // This allows us to detect errors and maintains data array format
    // for parsing
    // Print "TE" for temperature error instead of temperature output
    // from bmp085ReadUT() for a total of 2 bytes
    dataString = dataString + ", ERROR "; //temperature read error
    dataString = dataString + ", ERROR "; //pressure read error
    }
    else {
    // If error count is not greater than 3,
    // Print temperature reading from bmp085ReadUT()
    // Print pressure reading from bmp085ReadUP()
    tempur = Bmp::bmp085GetTemperature(Bmp::bmp085ReadUT());
    presur = Bmp::bmp085GetPressure(Bmp::bmp085ReadUP());
    dataString = dataString + ", " + String(tempur * 0.1);
    dataString = dataString + ", " + String(presur);
    }

  // [GIEGER_COUNTER_READ]
  // Create a variable gc_cnt to store the number of Geiger
  // hits from gc_counts so that if an interrupt is triggered, hits are
  // not missed while writing to the buffer
  gc_cnt = gc_counts;
  // Reset gc_counts to 0
  gc_counts = 0;
  // Print the number of counts detected during the duration of 1 loop
  dataString = dataString + ", " + String(gc_cnt);

  // [HUMIDITY_SENSOR_READ]
  // Read humidity sensor data, store in buffer, update cursor location
```

```
  dataString = dataString + ", " + String(analogRead(HUM_SENS));

  // [GYROSCOPE_SENSOR_READ]
  if (Gyro::gyroGetError() > 3) {
    // If error count is greater than 3, indicate sensor is malfunctioning.
    // Print 6 error bytes to replace the 6 Gyroscope data bytes
    //  2 bytes from X-axis, 2 bytes from Y-axis, 2 bytes from Z-axis
    //  (data type for each axis is an int)
    // This allows us to detect errors and maintains data array format for pa
    // Store "GE" for gyro error instead of gyro reading from the X-axis
    //  for a total of 2 bytes

    //write out error to data
    dataString = dataString + ", ERROR "; //gyX read error
    dataString = dataString + ", ERROR "; //gyY read error
    dataString = dataString + ", ERROR "; //gyZ read error
  }

  else {
    // If error count is not greater than 3, read gyro values
    gyX = Gyro::readX() + gyX_offset; //enter your team's initialization valu
    gyY = Gyro::readY() + gyY_offset; //enter your team's initialization valu
    gyZ = Gyro::readZ() + gyZ_offset; //enter your team's initialization valu
    // store in buffer, update cursor location
    dataString = dataString + ", " + String(gyX);
    dataString = dataString + ", " + String(gyY);
    dataString = dataString + ", " + String(gyZ);
  }

  // [SD_WRITE]
  // Write data to SD file prepared by dataSD::setupMemory().
  // This appends this data to the end of the file
  if (writeDataToSD()) {
//   sdError = false; // If TRUE, write to SDCard and Blink LED B
    if (!groundMode) digitalWrite(LEDA, LOW);
    if (groundMode) digitalWrite(LEDA, HIGH);
    //  Change the State of the LED from OFF->ON, or ON->OFF
       if (ledState == 0) {
          digitalWrite(LEDB, HIGH);
          ledState = 1;
       }
    //  Handle the case where the LED state is now 2 and set it back to a val
       else {
           digitalWrite(LEDB, LOW);
           ledState = 0;
```

```
      }
    }
    else {
      // IF False SDCard error, NO write to SDCard and Blink LED A
  //   sdError = true;
      digitalWrite(LEDB, LOW);
      if (ledState == 0) {
        digitalWrite(LEDA, HIGH);
        ledState = 1;
      }
      //  Handle the case where the LED state is now 2 and set it back to a val
      else {
        digitalWrite(LEDA, LOW);
        ledState = 0;
      }
    }
    // Print data to Serial monitor
    if (groundMode) Serial.println(dataString);
}  // End of main LOOP
```