```
/*MODEL ROCKET FLIGHT COMPUTER
 * Datalogging program
 * Written by Matt Minogue, also known as @LabRatMatt on YouTube and Instructables
 * Note that much of the IMU Code derives from Jeff Rowberg's MPU6050 Demo Code, the copyright
statement and liscense are included
 *
 */
 //Changelog
 //12-29-2019 Created Program to test buzzer on Pin 3, tested library inclusion
 //01-04-2020 Finished hardware, wrote datalogging capability for temp and pressure
 //01-05-2020 Created datalogging program that includes modes and IMU data
 //01-07-2020 Implemented linear acceleration datalogging, ran into FIFO overflow issues
 //01-09-2020 Implemented angular position datalogging, yet to resolve FIFO overflow issues
 //01-18-2020 Added dataRate variable to set ideal number of data points per second
 //02-08-2020 Modified code to make better use of preprocessor directives in place of variables,
effect on memory is minimal
 //05-21-2020 Added additional comments and clarifications in preparation for publication on
Instructables
/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

#include <SD.h>           //includes SD card library for datalogging
#include <Wire.h>         //includes Wire.h library for I2C interface
#include <SFE_BMP180.h>   //includes SparkFun's BMP 180 library

// For the IMU
// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h" //#include "MPU6050.h" not necessary if using MotionApps
include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation is used in
I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

//define relevant pins
#define INTERRUPT_PIN 2  //I2C Interrupt Pin for MPU6050, use pin 2 on Arduino Uno & most boards
#define BUZZER_PIN 3     //set buzzer pin
#define BLUE_LED_PIN 8   //set blue LED pin
#define YELLOW_LED_PIN 9 //set yellow LED pin
#define RED_LED_PIN 10   //set red LED pin
#define BUTTON_PIN 7     //set button pin
#define chipSelect 4     //set chip select pin for MicroSD Card Adapter (CS pin)
```

```
//declare general use variables
int buttonState = 0;
int MODE = 0;       //initialize mode to zero
int t = 0;          //create timestamp value
int dataRate = 10;  //set specified sampling rate (data points per second) (somewhere between 10-
200 is ideal)

//declare MPU control/status vars
bool blinkState = false;
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

//declare orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

//create objects
File file;
SFE_BMP180 BMP;
MPU6050 mpu;

//Interrupt Detection Routine
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

void setup() {
  //--- Serial Debugging ---
  Serial.begin(9600);
  //--- Establish Pin Modes and turn off all LEDs ---
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(BLUE_LED_PIN, OUTPUT);
  pinMode(YELLOW_LED_PIN, OUTPUT);
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(chipSelect, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
  pinMode(INTERRUPT_PIN, INPUT);
  digitalWrite(YELLOW_LED_PIN, LOW);
  digitalWrite(RED_LED_PIN, LOW);
  digitalWrite(BLUE_LED_PIN, LOW);
  tone(BUZZER_PIN, 500, 250);
  //initialize SD Card
  if(!SD.begin(chipSelect)){
    //Serial debugging
    Serial.println("Could not initialize SD card");
  }
  //clear SD data
  if(SD.exists("file.txt")){
    if(SD.remove("file.txt") == true){
      Serial.println("removed data");
    }
  }
  //initialize BMP sensor
  if(BMP.begin()){
    Serial.println("BMP init success");
  }
  //initialize IMU and I2C clock
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE  // join I2C bus (I2Cdev library doesn't do
this automatically)
```

```
      Wire.begin();
      Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation
difficulties
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
      Fastwire::setup(400, true);
  #endif
  mpu.initialize(); //start MPU
  Serial.println(F("Testing device connections...")); //debugging serial statement
  Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection
failed")); //debugging serial statement
  devStatus = mpu.dmpInitialize();
  // supply your own gyro offsets here, scaled for min sensitivity
  mpu.setXGyroOffset(0);
  mpu.setYGyroOffset(0);
  mpu.setZGyroOffset(0);
  mpu.setZAccelOffset(1688); // 1688 factory default for my test chip
  if (devStatus == 0) {
      // turn on the DMP, now that it's ready
      Serial.println(F("Enabling DMP..."));
      mpu.setDMPEnabled(true);
      // enable Arduino interrupt detection
      Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
      Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
      Serial.println(F(")..."));
      attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
      mpuIntStatus = mpu.getIntStatus();
      // set our DMP Ready flag so the main loop() function knows it's okay to use it
      Serial.println(F("DMP ready! Waiting for first interrupt..."));
      dmpReady = true;
      // get expected DMP packet size for later comparison
      packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
      // ERROR!
      // 1 = initial memory load failed
      // 2 = DMP configuration updates failed
      // (if it's going to break, usually the code will be 1)
      Serial.print(F("DMP Initialization failed (code "));
      Serial.print(devStatus);
      Serial.println(F(")"));
  }
  //set mode
  MODE = 1; //set to PAD IDLE mode - initialize sensors and SD card
  //MODE = 2; //set to FLIGHT mode - log data
  //MODE = 3; //set to RECOVERY mode - close file
}

void loop() {
  // put your main code here, to run repeatedly:
  if(MODE == 1){  //PAD IDLE MODE
    digitalWrite(RED_LED_PIN, HIGH);
    file = SD.open("file.txt", FILE_WRITE); //Open SD card file
    if(file) {
      Serial.println("t,T,P,ax,ay,az,rx,ry,rz"); //print first line with data labels
      file.println("t,T,P,ax,ay,az,rx,ry,rz");
      MODE = 2;
    }
    else{
      Serial.println("Error opening file");
      delay(5000); //just chill for 5 seconds before trying again
    }
  }
  if(MODE == 2){ //ACTIVE FLIGHT mode
    digitalWrite(YELLOW_LED_PIN, HIGH);
    digitalWrite(RED_LED_PIN, LOW);
    digitalWrite(BLUE_LED_PIN, LOW);
    //print timestamp and comma to separate data
    Serial.print(t);
    Serial.print(",");
    file.print(t);
```

```
  file.print(",");
  //set up temperature and pressure sensor
  char status;
  double T,P,a;
  //get BMP status and read temperature
  status = BMP.startTemperature();
  if(status != 0){
    delay(status);
    status = BMP.getTemperature(T);
    if(status != 0){
      Serial.print(T); //print temperature values
      file.print(T);
    }
  }
  Serial.print(","); //print commas to separate values
  file.print(",");
  status = BMP.startPressure(3); //delay 3 ms to get temperature value
  if(status != 0){
    delay(status);
    status = BMP.getPressure(P,T);
    if(status != 0){
      Serial.print(P); //print pressure values
      file.print(P);
    }
  }
  Serial.print(",");
  file.print(",");
  //get IMU data
  if (!dmpReady) return;
  while (!mpuInterrupt && fifoCount < packetSize){
    if (mpuInterrupt && fifoCount < packetSize){
      fifoCount = mpu.getFIFOCount();
    }
  }
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();
  fifoCount = mpu.getFIFOCount();
  if((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount >=1024){
    mpu.resetFIFO();
    fifoCount = mpu.getFIFOCount();
    Serial.println("FIFO Overflow!");
  }
  else if(mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)){
    while(fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
    mpu.getFIFOBytes(fifoBuffer, packetSize);
    fifoCount -= packetSize;
    //get real-world acceleration
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    //print real-world acceleration
    Serial.print(aaReal.x);
    file.print(aaReal.x);
    Serial.print(",");
    file.print(",");
    Serial.print(aaReal.y);
    file.print(aaReal.y);
    Serial.print(",");
    file.print(",");
    Serial.print(aaReal.z);
    file.print(aaReal.z);
    Serial.print(",");
    file.print(",");
    //get Euler angles
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    //print Euler angles
    Serial.print(euler[0]*180/M_PI);
```

```
        file.print(euler[0]*180/M_PI);
        Serial.print(",");
        file.print(",");
        Serial.print(euler[1]*180/M_PI);
        file.print(euler[1]*180/M_PI);
        Serial.print(",");
        file.print(",");
        Serial.print(euler[2]*180/M_PI);
        file.print(euler[2]*180/M_PI);
      }
      //end data entry line
      Serial.println(); //ends line
      file.println(); //ends line
      //check for mode switch
      buttonState = digitalRead(BUTTON_PIN);
      if(buttonState == LOW){
        MODE = 3;
        tone(BUZZER_PIN, 1000, 250);
        delay(100);
      }
    }
    if(MODE == 3){ //RECOVERY MODE
      file.close();
      digitalWrite(YELLOW_LED_PIN, LOW);
      digitalWrite(RED_LED_PIN, LOW);
      digitalWrite(BLUE_LED_PIN, HIGH);
      delay(1000);
    }
    t = t + 1;              //increment t value
    delay(1000/dataRate);   //pause so that data output corresponds to data rate
    if(t > 32765){          //prevents issues related to integers rolling over at 32767
      t = 1;
    }
  }
```