telemetryHornet

// Joseph Laub
// 2/21/15
// Telemetry System Prototype 2
// Code name Hornet
// Prototype system to be used on the Aircraft Murcielago 2

//------------------------------------------------------------------------------------------------------------------------------

// Arduino Library Setup
#include <Wire.h>            // For the i2c bus
#include "Arduino.h"         // For communicating with the EagleTree Air Speed Sensor V3
#include "MPL3115A2.h"       // For communicating with the MPL3115A2
#include <EEPROM.h>          // Load the EEPROM library
#include <VirtualWire.h>     // Load Virtual Wire Library so that RF ground link can be established.
#include <SFE_VL6180X.h>     // Load the VL6180 Library for Time of Flight sensor
#include <Average.h>

MPL3115A2 myPressure;        //Create an instance of the object for altitude sensor

const int onLEDPin   = 4;   // Red LED
const int infoLEDPin = 3;   // Yellow LED
const int buttonPin  = 2;   // Button on the User Interface Board

#define VL6180X_ADDRESS 0x29

VL6180xIdentification identification;
VL6180x sensor(VL6180X_ADDRESS);

void setup() {
  Serial.begin(115200);
  Serial.println("Telemetry System v2.0: Hornet");

  Wire.begin();  // Join i2c bus

  pinMode(onLEDPin, OUTPUT);
  pinMode(infoLEDPin, OUTPUT);
  pinMode(buttonPin, INPUT);

  if(sensor.VL6180xInit() != 0){
    Serial.println("FAILED TO INITIALIZE"); //Initialize device and check for errors
  }

  sensor.VL6180xDefaultSettings(); //Load default settings to get started.

  // vw_set_ptt_inverted(true);  // Required for RF Link Module
  // vw_setup(1200);             // Bits per second
// vw_set_tx_pin(1);           // pin 1 used a as the transmit data out into the TX link module

myPressure.begin();              // Get sensor online
myPressure.setModeBarometer();   // Measure pressure in Pascals from 20 to 110 kPa
myPressure.setOversampleRate(7); // Set Oversample to the recommended 128
myPressure.enableEventFlags();   // Enable all three pressure and temp event flags

delay(1000);

Serial.println("System Setup Successfully");
digitalWrite(onLEDPin, HIGH);
Serial.println("Enter 1 for Data Acquisition Mode");
Serial.println("Enter 2 for Read EEPROM Mode");
Serial.println("Enter 3 for Clear EEPROM Mode");

int buttonState = 0; // variable for reading the pushbutton status
int byteRead = 0;
boolean waiting = true;
void loop() {

waiting = true;
delay(1000);
while (waiting == true) {

buttonState = digitalRead(buttonPin);

if (Serial.available()) {
    byteRead = Serial.read();
}

if (buttonState == HIGH) {
    Serial.println("DATA ACQUISITION MODE SELECTED");
dataAcquisitionLoop();
    waiting = false;
delay(2000);
}
if (byteRead == 49) {
    Serial.println("DATA ACQUISITION MODE SELECTED");
dataAcquisitionLoop();
    waiting = false;
delay(2000);
}
if (byteRead == 50) {
    Serial.println("READ EEPROM MODE SELECTED");
readEEPROM();
    Serial.println("EEPROM READ");
    byteRead = 0;
}
waiting = false;
delay(2000);
}
if (byteRead == 51) {
    Serial.println("CLEAR EEPROM MODE SELECTED");
clearEEPROM();
    Serial.println("EEPROM CLEARED");
    byteRead = 0;
    waiting = false;
    delay(2000);
}
Serial.println("STAND BY MODE");
while (waiting == false) {
    digitalWrite(infoLEDPin, HIGH);
delay(500);
    digitalWrite(infoLEDPin, LOW);
delay(500);
}
}

 altitudeZero
 // Joseph Laub
 // Altitude Zeroing
 // 2/19/15

 int g = 1;
 int zeroing = 0;
 int avgAmount = 2;
 float tempAlt = 0;
 int tempAltitude[200];

 float altitudeZero()
{
    digitalWrite(infoLEDPin, HIGH);
delay(250);
    digitalWrite(infoLEDPin, LOW);
    Serial.println("ZEROING ALTITUDE");
    // Ignore the first 50 samples
    for (g = 0; g <= 100; g++) {
        tempAlt = pressureToAltitude(findPressure())/100;
delay(100);
    }
    Serial.println(tempAlt);
clearEEPROM

// Joseph Laub
// 2/16/15
// Clears the EEPROM addresses within the I2C Chip
// Address for the I2C Chip is 0x50

unsigned int clearAddr;
int clearData;
void clearEEPROM() {
  clearData = 255;
  digitalWrite(infoLEDPin, HIGH);
  // write 255 to all 32768 bytes of the EEPROM Chip

  for (clearAddr = 0; clearAddr < 32768; clearAddr++) {
    Wire.beginTransmission(0x50);
    Wire.write((int)(clearAddr >> 8)); // MSB
    Wire.write((int)(clearAddr & 0xFF)); // LSB
    Wire.write(clearData);
    Wire.endTransmission();
    delay(5);
  }
  digitalWrite(infoLEDPin, LOW);
}

dataAcquisitionLoop

/*
Joseph Laub
2/16/15
This is the Data Acquisition Loop for Hornet.
Hornet uses I2C protocol to communicate with various sensors.
The order of events in the Data Acquisition Loop are:
  Find Data
  Make Necessary Corrections to Data
  Store Data in EEPROM Chip
  Transmit Data
  Check for Stop Button
  Repeat
The following variables are recorded by the system:
  Air Speed
  Altitude
  Ground Speed
  GPS Altitude
  Latitude
Longitude
Pressure
Temperature
Time
Time of Flight

The following sensors are used in the Hornet system:
Eagle Tree Air Speed Sensor V3...........Air Speed
MPL3115A2................................Altitude, Pressure, Temperature
Navigatron v2............................Ground Speed, GPS Altitude, Latitude, Longitude
VL6180...................................Time of Flight

// Initializing variables
boolean dataLoop = true;  // If true then keep looping the data acquisition loop. If false, return.
byte timeLSB = 0;         // Holds the least significant 8 bits of the timeBin
byte timeMSB = 0;         // Holds the most significant 8 bits of the timeBin

float airSpeedData = 0;   // Set air speed value to zero for start
float altitudeData = 0;   // Set altitude value to zero for start
float pressureData = 0;   // Set pressure value to zero for start
float timeBeforeLoop = 0; // This variable holds the time value for the script
float timeDiff = 0;       // Hold the difference in time that has been calculated. Used for determining sampling times
float timeZero = 0;       // Used for zeroing time
float zed = 0;

long address = 0;         // This variable chooses which EEPROM address to write to for function writeEEPROM
int altDecData = 0;        // Variable holds the decimal places from the altitude
long maxAddress = 32768;   // The maximum address that will be reached by the system before ending
int sampleTime = 500;      // in milliseconds, time between measurements
int temperatureData = 0;   // Set temperature value to zero for start
int ToFData = 0;           // in mm, Time of Flight data for liftoff confirmation
int timeBin = 0;           // Holds the time as a binary value

void dataAcquisitionLoop() {

    zed = altitudeZero();
    timeZero = millis();   // Take a time stamp of when this loop starts so that data can be synced to this starting time.

    digitalWrite(infoLEDPin, HIGH);  // Turn on infoLED to indicate that data is being acquired.
    Serial.println("DATA ACQUISITION LOOP EXECUTED");

    while (dataLoop == true) {

        // Code to acquire data from sensors and do calculations
    
    
}
timeBeforeLoop = millis() - timeZero;  // in ms, time before data is acquired

airSpeedData = findAirSpeed();  // in mph, air speed is returned
pressureData = findPressure();  // in Pa, pressure is returned multiplied by 1000 to preserve decimal values

temperatureData = findTemperature();  // in F, temperature is returned
altitudeData = pressureToAltitude(pressureData);  // in ft, altitude is returned multiplied by 100 to preserve two decimal places
ToFData = sensor.getDistance();  // in mm, Time of Flight data measures distance from base of plane to ground (to about 25 cm)

altitudeData = altitudeData/100 - zed;  // Altitude was multiplied by 100 to preserve two decimal places. Correct this magnitude.
pressureData = pressureData/1000;  // Pressure was multiplied by 1000 to preserve accuracy for int variables. Correct the magnitude

altDecData = (altitudeData - floor(altitudeData))*100;  // This takes the decimal value of the altitude
altitudeData = floor(altitudeData);  // Rounds down the altitude so that we can add the decimal values back later

timeBin = timeBeforeLoop/10;  // Time is stored in units of centiseconds.
timeMSB = timeBin >> 8;  // The most significant bits of time in cs

Serial.println("HORNET TELEMETRY DATA");
Serial.print("TIME ELAPSED:	"); Serial.print(timeBeforeLoop/1000); Serial.println("(s)");
Serial.print("TIME MSB:	"); Serial.println(timeMSB);
Serial.print("TIME LSB:	"); Serial.println(timeLSB);
Serial.print("ADDRESS:	"); Serial.print(address); Serial.println("(EEPROM)");
Serial.print("AIR SPEED:	"); Serial.print(airSpeedData); Serial.println("(mph)");
Serial.print("TIME OF FLIGHT:	"); Serial.print(ToFData); Serial.println("(mm)");
Serial.print("PRESSURE:	"); Serial.println(pressureData/1000);
Serial.print("ALTITUDE:	"); Serial.println(altitudeData);
Serial.print("ALTITUDE DECMIAL:	"); Serial.println(altDecData);
Serial.print("ALTITUDE ZEROING:	"); Serial.println(zed);
Serial.print("TEMPERATURE:	"); Serial.println(temperatureData); Serial.println("(F)");

// address = writeEEPROM(address, timeMSB);  // Write the first part of the time
address = writeEEPROM(address, timeLSB);  // Write the second part of the time
address = writeEEPROM(address, airSpeedData);  // Write the air speed to address.
address = writeEEPROM(address, ToFData);  // Time of Flight Data
address = writeEEPROM(address, altitudeData);  // Write the change in altitude to address.
address = writeEEPROM(address, altDecData);  // Write the change in altitude decimal places to address.
address = writeEEPROM(address, temperatureData);  // Write the temperature to address.
//transmitData(stopType);

    while (timeDiff < sampleTime) {   // Waits until one entire sampleTime has passed from the start of dataAcquisitionLoop
        timeDiff = (millis() - timeZero) - timeBeforeLoop;   // This time difference is how much time has elapsed since this loop started
    }

    timeDiff = 0;                       // Resets timeDiff for next loop

    if (address >= maxAddress) {        // If the address is at its limit
        address = 0;       // Reset address to 0 so that data can loop
    }

    buttonState = digitalRead(buttonPin);

    if (buttonState == HIGH) {
        Serial.println("DATA ACQUISITION LOOP TERMINATED");
        dataLoop = false;
        delay(1000);
    }
}

digitalWrite(infoLEDPin, LOW);   // Turn on infoLED to indicate that data is being acquired.
return;

findAirSpeed

    // Joseph Laub
    // 12/3/14
    // Finds and returns the air speed using the EagleTree Air Speed Sensor V3

int addressAirSpeed = 0x75;       // Data sheet claims 0xEA, however, use the high 7 version (0x75)
int w = 0x07;                      // Not sure why 0x07, however, I think this puts the sensor in read mode
float r = 0;                       // Used to store the air speed

float findAirSpeed() {

    r = 0; // reset r to 0
    Wire.beginTransmission(addressAirSpeed);
    Wire.write(w);
    Wire.endTransmission();
    Wire.requestFrom(addressAirSpeed, 2); // request 2 bytes from device
Wire.available(); // check and see how many bytes are available, should be 2

r = Wire.read();

Wire.endTransmission();

return r;
}

findAltitude

// Joseph Laub
// 12/7/14
// Finds the pressure recorded on the MPL3115A2. This can later be converted to a change in altitude

float findAltitude() {
    float f = myPressure.readAltitudeFt();
    return f;
}

findPressure

// Joseph Laub
// 12/3/14
// Finds the pressure recorded on the MPL3115A2. This can later be converted to a change in altitude

float findPressure() {
    float p = myPressure.readPressure();
    return p*1000;
}

findTemperature

// Joseph Laub
// 12/3/14
// Finds the temperature recorded on the MPL3115A2.

float findTemperature() {
    float t = myPressure.readTempF();
    return t;
}
**pressureToAltitude**

```c
int pressureToAltitude(float pre) {
    // p = 101325(1 - 2.25577e-5 * h)^5.25588
    // p = z(x - c * h)^v
    // solve for h
    // h = (x-(p/z)^(1/d))/v

    const float z = 101325;
    const float x = 1;
    const float c = 0.0000225577;
    const float v = 5.25588;
    float h = 0; // Altitude to be calculated
    pre = pre/1000;
    float part1 = pow((pre/z), (1/v));
    h = (x-part1)/c;
    h = h*3.28084;
    return h*100;
}
```

**readEEPROM**

```c
// Joseph Laub
// 2/16/15
// Reads the EEPROM addresses within the I2C Chip
// Address for the I2C Chip is 0x50

void readEEPROM() {
    long j = 0;
    int flagEnd = 0; // flagEnd will determine if there is no more data. Ten 255's in a row means it is the end.
    digitalWrite(infoLEDPin, HIGH);

    while (j < 32768 && flagEnd < 10) {
        byte rdata = 0xFF;
        Wire.beginTransmission(0x50);
        Wire.write((int)(j >> 8)); // MSB
        Wire.write((int)(j & 0xFF)); // LSB
        Wire.endTransmission();
        Wire.requestFrom(0x50, 1);
        if (Wire.available()) rdata = Wire.read();
        Serial.print("Address:\t"); Serial.print(j); Serial.print("Data:\t"); Serial.println(rdata);
        j++;
    }
}```
if (rdata == 255) {
    flagEnd++;
} else {
    flagEnd = 0;
}

digitalWrite(infoLEDPin, LOW);

void transmitData(int data) {
    //
    char dataMsg[4];
    data = abs(data);

    vw_set_p1t_inverted(true);  // Required for RF Link Module
    vw_setup(1200);             // Bits per second
    vw_set_t1x_pin(1);          // pin 1 used a as the transmit data out into the TX link module
    // delay(200);

    itoa(data, dataMsg, 10);
    vw_send((uint8_t *)dataMsg, strlen(dataMsg));
    vw_wait_t1x();

    // Serial.println("SENT");
}

int writeEEPROM(unsigned int addr, byte val) {
    int writeData = val;
    Wire.beginTransmission(0x50);
    Wire.beginTransmission(0x50);
    Wire.write((int)(addr >> 8)); // MSB
    Wire.write((int)(addr >> 8)); // MSB
    Wire.write((int)(addr >> 8)); // MSB
    Wire.write((int)(addr >> 8)); // MSB
    Wire.write((int)(addr >> 8)); // MSB
    Wire.write((int)(addr >> 8)); // MSB
    Wire.endTransmission();
    return writeData;
}
Wire.write((int)(addr & 0xFF)); // LSB
Wire.write(writeData);
Wire.endTransmission();
delay(10);
addr++;
return addr;
}