// This is designed for the Advanced Category of the United Kingdom CanSat

// Competition 2016, as part of Cyclone, a team from St Paul's School,

// Barnes, London.

//

// For more information about the entry, including the Electronics Design

// Take a look on the GitHub Page - http://github.com/cyclonecansat or our

// website at: http://teamcycl.one

//

// Any questions, contact me at: ashwin.ahuja@gmail.com

/\*

\* INCLUDES

\*

\* Sensor Library - Library for the reading and manipulation of the MS5637

\* and HYT271 - the pressure, internal temperature (MS5637) and humidity

\* and external temperature (HYT271) sensors

\*

\* RFM98W Library - Library for the RFM98W Transceiver Board, using the LoRa

\* transmission protocol - including dealing with creation of the Packet and

\* the transmission and receiving of data.

\*

\* Servo Library - Library used for the release of the parachute, currently

\* unused, since it will likely not be used.

\*

\* Wire Library - Library required for all I2C communication.

\*

\* TinyGPS++ Library - Library used for receiving and interpreting data from

\* the GP-2106 module (available from Sparkfun) and used alongside the board

\* produced by Sparkfun - the GP-2106 Evaluation Board

\*

\* SPI - Library used for all SPI communications - currently unused, but could

\* be used for the 9DOF sensor.

\*/

#include <sensor\_library.h>

#include <RFM98W\_library.h>

#include <**Servo**.h>

#include <Wire.h>

#include <**TinyGPS**++.h>

#include <**SPI**.h>

#include <LSM9DS1\_Registers.h>

#include <LSM9DS1\_Types.h>

#include <**SparkFunLSM9DS1**.h>

/\*

\* DEFINITIONS

\*

\* Pins for the RFM98W - as implied

\* Hardware Serial declarations for the GPS module and the OpenLog

\* Number of bytes implied by each command

\* Parachute Release Servo Pin

\* Boolean - parachute released

\* Boolean - parachute armed

\* Parachute Servo - move from (minimum)

\* Parachute Servo - move to (maximum)

\* Baud Rate for GPS communication

\* Baud Rate for Computer Serial communications

\* Verbosity - 0 for none, 1 for all

\* Boolean - Longitude

\* Boolean - Latitude

\* Time between transmissions (during other times, the system will be receiving data from the RFM98W

\* Various pieces of information about the program

\*/

//LoRa Pins

#define nss 20

#define dio0 7

#define dio5 16

#define rfm\_rst 21

// Serial Declarations

#define gpsSerial **Serial1**

#define openlog **Serial2**

//radio command lengths

const byte cmd\_lengths[8] = {0,8,2,1,1,2,2,2};

// Servo Info

const int ServoPin = 0; // NOT CORRECT

bool parachuteReleased = false;

bool parachuteArmed = false;

const int servoMin = 0;

const int servoMax = 180;

// Baud Rate Declarations

#define GPS\_BaudRate 4800

#define Computer\_BaudRate 115200

#define OpenLogBaudRate 9600

//Verbosity

#define verbosity 1

//LSM9DS1 definitions

#define LSM9DS1\_M  0x1E // Would be 0x1C if SDO\_M is LOW

#define LSM9DS1\_AG  0x6B // Would be 0x6A if SDO\_AG is LOW

#define PRINT\_CALCULATED

//GPS Declarations

bool latitudePositive = false;

bool longitudePositive = true;

//Time between transmissions

#define timeBetweenTransmissions 1000

#define sensorReadingPeriod 500

//Various other info

#define SoftwareVersionNumber "1.2.1"

#define Author "Ashwin Ahuja"

/\*

\* OBJECT DECLARATIONS

\*

\* TinyGPSPlus Object - receives and decodes GPS data

\* SensLib - Communication with MS5637 and HYT271

\* RFMLib - Communication with Hope RFM98W

\* LSM9DS1 -  Communication with 9DOF

\* Servo - Parachite release

\* Radio transmission timer.

\* Sensor reading timer

\* Packet received boolean

\* Has MS5637 been read?

\* Is it time to read?

\*/

**TinyGPSPlus** gps; // GPS

SensLib sns; // Sensor Object

RFMLib radio = RFMLib(nss, dio0, dio5, rfm\_rst); // Radio Object

**LSM9DS1** imu; // LSM9DS1 Object

**Servo** ParaRelease; // Parachute Release Servo Object

uint32\_t radioTransmitTimer;

uint32\_t sensorReadTimer;

boolean pkt\_rx = false;

boolean msRead = false;

boolean readSens = false;

/\*

\* Miscellaneous Declarations

\*/

int gyx;

int gyy;

int gyz;

int acx;

int acy;

int acz;

int magx;

int magy;

int magz;

float roll;

float pitch;

float heading;

int sampleNumber = 0;

void assemblePacket (RFMLib:: Packet &pkt);

void decodePacket (RFMLib:: Packet pkt);

void setup() {

 ParaRelease.attach(ServoPin);

 ParaRelease.write(servoMin);

**Serial**.begin(Computer\_BaudRate);

 gpsSerial.begin(GPS\_BaudRate);

 openlog.begin(OpenLogBaudRate);

**SPI**.begin();

 byte my\_config[5] = {0x64,0x74,0xFA,0xAC,0xCD};//radio settings

 radio.configure(my\_config);//Radio configuration

 Wire.begin();//join the I2C bus

 sns.initialise();//initialise the sensors connected over I2C

 imu.settings.device.commInterface = IMU\_MODE\_I2C;

 imu.settings.device.mAddress = LSM9DS1\_M;

 imu.settings.device.agAddress = LSM9DS1\_AG;

 if (imu.begin())

 {

   #if verbosity == 1

**Serial**.println("9DOF Initialised");

   #endif

 }

 radioTransmitTimer = millis();

 #if verbosity != 0

**Serial**.print("Cyclone CanSat Firmware Version Number: ");

**Serial**.print(SoftwareVersionNumber);

**Serial**.print(" (Cassini). Verbosity is equal to ");

**Serial**.println(verbosity);

 #endif

}

void loop() {

 // READ FROM IMU

 readIMU();

 if(millis() - sensorReadTimer >= sensorReadingPeriod)

   readSens = true;

 // READ FROM SENSORS

 if(readSens)

 {

    if(msRead)

    {

    sns.pollHYT271();

**Serial**.print("HUMIDITY: ");

**Serial**.println(sns.humidity);

    readSens = false;

    sensorReadTimer = millis();

    msRead = false;

    }

    else

    {

     sns.pollMS5637();

     msRead = true;

    }

 }

 if(radio.rfm\_done) finishRFM();

 while(gpsSerial.available())gps.encode(gpsSerial.read());

 if ((millis() - radioTransmitTimer) > timeBetweenTransmissions && radio.rfm\_status != 1)

 {

   transmission();

 }

}

void transmission()

{

#if verbosity != 0

**Serial**.println("TX");

#endif

if (radio.rfm\_status == 2)

{

 RFMLib::Packet p;

 radio.endRX(p);

}

RFMLib::Packet p;

assemblePacket(p);

radio.beginTX(p);

attachInterrupt(7, RFFinished, RISING);

radioTransmitTimer = millis();

sensorReadTimer = millis();

}

void RFFinished()

{

 radio.rfm\_done = true;

}

void finishRFM()

{

   switch(radio.rfm\_status){

 case 1:

    #if verbosity != 0

**Serial**.println("Ending transmission.");

    #endif

    radio.endTX();

    #if verbosity != 0

**Serial**.println("Beginning reception.");

    #endif

    radio.beginRX();

    radio.rfm\_done = false;

        attachInterrupt(7,RFFinished,RISING);

    break;

    case 2:

    #if verbosity != 0

**Serial**.println("Ending reception.");

    #endif

    RFMLib::Packet rx;

    radio.endRX(rx);

    decodePacket(rx);

    break;

  }

}

void decodePacket(RFMLib::Packet pkt)

{

 byte j = 0;

 if (verbosity != 0)

 {

**Serial**.println("Decode: ");

   for (int i = 0; i <= pkt.len; i++)

   {

**Serial**.print(pkt.data[i]);

   }

**Serial**.println();

 }

 while (j < pkt.len)

 {

   switch(pkt.data[j]){

     case 7:

       if (pkt.data[j+1] == 255 && pkt.data[j+2] == 255)

       {

**Serial**.println("RELEASE");

         ParaRelease.write(servoMax);

       }

       j = j + 2;

       break;

   }

 }

}

void assemblePacket(RFMLib::Packet &pkt)

{

 pkt.len = 41;

 int32\_t pr\_calc = sns.pressure;

 byte round\_byte = ((pr\_calc % 10)>4)?1:0;

 pr\_calc /= 10;

 pr\_calc += (int16\_t) round\_byte;

 uint16\_t small\_pressure = (uint16\_t) pr\_calc;

 sampleNumber++;

 pkt.data[0] = (byte)(sampleNumber >> 8);

 pkt.data[1] = (byte)(sampleNumber & 255);

 pkt.data[2] = (byte)(sns.internal\_temperature >> 8);

 pkt.data[3] = sns.internal\_temperature & 255;

 pkt.data[4] = (byte)(small\_pressure >> 8);

 pkt.data[5] = small\_pressure & 255;

 pkt.data[6] = (byte)(sns.external\_temperature >> 8);

 pkt.data[7] = sns.external\_temperature & 255;

 pkt.data[8] = (byte)(sns.humidity >> 8);

 pkt.data[9] = sns.humidity & 255;

 int time2 = gps.time.hour() \* 3600 + gps.time.minute() \* 60 + gps.time.second();

 pkt.data[10] = (byte)(time2 >> 8);

 pkt.data[11] = time2 & 255;

 pkt.data[12] = (byte)gps.satellites.value();

 uint32\_t raw\_pos = (uint32\_t)(gps.location.lng()\*1000000);

 pkt.data[13] = (byte)(raw\_pos >> 24);

 pkt.data[14] = (byte)(raw\_pos >> 16);

 pkt.data[15] = (byte)(raw\_pos >> 8);

 pkt.data[16] = raw\_pos & 255;

 raw\_pos = (uint32\_t)(gps.location.lat()\*1000000);

 pkt.data[17] = (byte)(raw\_pos >> 24);

 pkt.data[18] = (byte)(raw\_pos >> 16);

 pkt.data[19] = (byte)(raw\_pos >> 8);

 pkt.data[20] = raw\_pos & 255;

 uint32\_t raw\_alt = gps.altitude.meters();

 pkt.data[21] = (byte)(raw\_alt >> 8);

 pkt.data[22] = raw\_alt & 255;

 pkt.data[23] = 0;

 pkt.data[24] = 0;

 pkt.data[25] = 0;

 pkt.data[26] = 0;

 pkt.data[27] = 0;

 pkt.data[28] = 0;

 pkt.data[29] = 0;

 pkt.data[30] = 0;

 pkt.data[31] = 0;

 pkt.data[32] = 0;

 pkt.data[33] = 0;

 pkt.data[34] = 0;

 pkt.data[35] = 0;

 pkt.data[36] = 0;

 pkt.data[37] = 0;

 pkt.data[38] = 0;

 pkt.data[39] = 0;

 pkt.data[40] = 0;

 if (verbosity > 0)

 {

**Serial**.print("Sample Number = ");

**Serial**.println(sampleNumber);

**Serial**.print("Internal Temp = ");

**Serial**.println(sns.internal\_temperature);

**Serial**.print("Pressure = ");

**Serial**.println(sns.pressure);

**Serial**.print("External Temp = ");

**Serial**.println(sns.external\_temperature);

**Serial**.print("Humidity = ");

**Serial**.println(sns.humidity);

   int time3 = gps.time.hour() \* 3600 + gps.time.minute() \* 60 + gps.time.second();

**Serial**.print("Second = ");

**Serial**.println(time3);

**Serial**.print("GPS Fix = ");

**Serial**.println(gps.satellites.value());

**Serial**.print("Longitude = ");

**Serial**.println(gps.location.lng());

**Serial**.print("Latitude = ");

**Serial**.println(gps.location.lat());

**Serial**.print("Altitude = ");

**Serial**.println(gps.altitude.meters());

**Serial**.print("Acceleration in x = ");

**Serial**.println(acx);

**Serial**.print("Acceleration in y = ");

**Serial**.println(acy);

**Serial**.print("Acceleration in z = ");

**Serial**.println(acz);

**Serial**.print("Rotation in x = ");

**Serial**.println(gyx);

**Serial**.print("Rotation in y = ");

**Serial**.println(gyy);

**Serial**.print("Rotation in z = ");

**Serial**.println(gyz);

**Serial**.print("Heading = ");

**Serial**.println(heading);

**Serial**.print("Pitch = ");

**Serial**.println(pitch);

**Serial**.print("Roll = ");

**Serial**.println(roll);

 }

}

void readIMU()

{

 imu.readGyro();

 imu.readAccel();

 imu.readMag();

 gyx = imu.calcGyro(imu.gx);

 gyy = imu.calcGyro(imu.gy);

 gyz = imu.calcGyro(imu.gz);

 acx = imu.calcAccel(imu.ax);

 acy = imu.calcAccel(imu.ay);

 acz = imu.calcAccel(imu.az);

 magx = imu.calcMag(imu.mx);

 magy = imu.calcMag(imu.my);

 magz = imu.calcMag(imu.mz);

 calculateHeadingAndRoll();

}

void calculateHeadingAndRoll()

{

 #define DECLINATION -8.58

 roll = atan2(imu.ay, imu.az);

 pitch = atan2(-imu.ax, sqrt(imu.ay \* imu.ay + imu.az \* imu.az));

 heading;

 if (imu.my == 0)

   heading = (imu.mx < 0) ? 180.0 : 0;

 else

   heading = atan2(imu.mx, imu.my);

 heading -= DECLINATION \* PI / 180;

 if (heading > PI) heading -= (2 \* PI);

 else if (heading < -PI) heading += (2 \* PI);

 else if (heading < 0) heading += 2 \* PI;

 // Convert everything from radians to degrees:

 heading \*= 180.0 / PI;

 pitch \*= 180.0 / PI;

 roll  \*= 180.0 / PI;

}