Atmospheric Data Collection and Transmission via High-Altitude Weather Balloon

Wes Hileman, Cory Ross, Josh Connolly, Melinda Bradley
Pikes Peak Community College
Faculty Adviser: Liz Coelho (liz.coelho@ppcc.edu)
7 April 2015

Abstract

The Apogee mission’s primary goal is to provide future DemoSat teams with a precise view of the conditions a payload experiences during flight on high-altitude balloon in the lower stratosphere. We aim to enable future DemoSat teams to construct effective experiments by presenting and analyzing the atmospheric- and flight-data collected by the Apogee payload. The payload includes the following sensors: an altimeter, an accelerometer, a magnetometer, an ultraviolet light sensor, two Geiger counters, and eight thermistors. Sensor data is processed by a microcontroller and stored in onboard microSD memory. The addition of a radio downlink provides real-time flight-data to ground teams and a redundant data logging system. The collected data is analyzed and used to describe payload’s flight. The results of the data collection, analysis, and transmission provide information for use in the design of future DemoSat projects.

1 Introduction

The DemoSat program is a statewide engineering project in which teams of college students construct payloads for launch into the lower stratosphere on a high-altitude balloon. The payload built by Pikes Peak Community College’s Apogee team is a spacecraft that embodies sensors, actuators, and their supporting subsystems. The spacecraft imitates a cube satellite (CubeSat) and is designed to exploit the weather, radiation, and perspective conditions characteristic of the high-altitude environment. Specifically, the payload collects weather and ionizing-radiation data to characterize the flight and transmits the data via radio downlink in near real-time. Table 1 outlines key mission-parameters and the next section details the spacecraft’s design.

<table>
<thead>
<tr>
<th>Launch Mechanism</th>
<th>latex balloon with parachute and control payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch Interface</td>
<td>braided nylon flight cord (4.7 mm diameter)</td>
</tr>
<tr>
<td>Approximate Apogee</td>
<td>30 km</td>
</tr>
<tr>
<td>Mass Ceiling</td>
<td>1.1 kg</td>
</tr>
<tr>
<td>Approximate Duration</td>
<td>90 min ascent, 60 min descent</td>
</tr>
</tbody>
</table>

Table 1: Key mission-parameters.

2 Background

Six elements form the mission design, or mission architecture, of the Apogee project. Table 2 on the next page summarizes these six elements, including the mission concept, subject, payload, spacecraft bus, launch system, ground system, and communications architecture. A complete mission design encompasses a set of designs for each element [4].

A number of the elements are marked as non-tradable, indicating the designs of those elements are fixed by some constraining factor. For example, the subject is defined by the mission statement and the sponsor (COSGC) sets the launch and ground systems. Contrary to non-tradable elements are tradable elements,
<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
<th>Can be Traded</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mission Concept</strong></td>
<td>The fundamental statement of how the system will work (data delivery, tasking, communications, timeline)</td>
<td>yes</td>
<td>Open to alternate approaches</td>
</tr>
<tr>
<td><strong>Subject</strong></td>
<td>The items sensed by the payload</td>
<td>no</td>
<td>Subject is passive and well defined in the mission statement</td>
</tr>
<tr>
<td><strong>Payload</strong></td>
<td>Hardware and software that sense the subject</td>
<td>yes</td>
<td>Can select method of sensing</td>
</tr>
<tr>
<td><strong>Spacecraft Bus</strong></td>
<td>Houses the payload and supporting subsystems such as power, structure and rigidity, temperature control, data handling, and downlink</td>
<td>yes</td>
<td>Can choose structure and subsystem configuration</td>
</tr>
<tr>
<td><strong>Launch System</strong></td>
<td>Launch vehicle, interfaces, and ground-support equipment. Constrains size, shape, and mass of spacecraft</td>
<td>no</td>
<td>Fixed by sponsor (COSGC)</td>
</tr>
<tr>
<td><strong>Ground System</strong></td>
<td>The hardware and software governing ground-based command and control of the launch vehicle and spacecraft</td>
<td>no</td>
<td>Fixed by communications architecture, sponsor (COSGC)</td>
</tr>
</tbody>
</table>

Table 2: Mission elements. (Adapted from Space Mission Analysis and Design, 3rd ed. See [4]).

whose designs are open to change. The mission concept, payload, and spacecraft bus are tradable elements. The following sections detail the implementation of each tradable element in the Apogee spacecraft.

Figure 1 on the following page portrays the spacecraft’s electrical schematic and figure 2 on the next page depicts its block diagram. The components shown in the figures are also discussed in the following sections.

2.1 Mission Concept

The *mission concept* defines the elementary workings of the mission and is subdivided into the areas of data delivery and tasking [4]. Each area is discussed in the following sections.

2.1.1 Data Delivery

Data delivery addresses data generation, collection, and ultimate distribution to the Apogee team. Figure 3 on page 4 summarizes the high-level data flows of the mission. As shown, data is produced by sensors, processed into storable formats, and stored in onboard memory for later retrieval. Telemetry is also sent via radio downlink, providing a level of redundancy.

The data generated by the payload and transmitted by downlink is *mission data* that characterizes the subject directly. *Housekeeping data* is collected by sensors in the spacecraft bus and either influences the spacecraft’s operation (e.g. heater control) or is used in analysis of mission data (e.g. altitude data) [4].

2.1.2 Tasking

Tasking decides when and in what order operations are preformed by the spacecraft [4]. Tasking for the Apogee spacecraft be accomplished autonomously by an onboard computer. The computer is an *Arduino Micro*, a breakout of the ATmega32u4 microcontroller (see section 2.2.4 on page 6 for more details). Figure 4 on page 4 depicts the state diagram defining the computer’s operation at any instant. See appendix section 5 on page 11 for the flight software.

2.2 Spacecraft Bus

The *spacecraft bus* accommodates the payload’s supporting subsystems, including structural, power, thermal, structural, command and data handling (C&DH), and communications. Each subsystem is discussed in the
Figure 1: Spacecraft electrical schematic.

Figure 2: Spacecraft block diagram.
Figure 3: Top level data flows.

Figure 4: State diagram that determines the flight-computer’s programming.
next sections.

2.2.1 Structural

The structure is made from six pieces of rigid sheets of ¾”-thick expanded polystyrene Mylar-foil-lined foam, cut and assembled to form the shape of a hollow cube. Each piece is fastened together using 100% silicon. An approximately 1” thick layer of Great Stuff insulating spray foam covers the outside of the cube. The inside dimensions of the structure are 6.5” x 6.5” x 6.5” (274.625 in³).

The rigid foam core gives the spacecraft a uniform interior with straight walls and a level floor and ceiling, which allows each payload component to fit neatly inside. To facilitate component placement and organization, a system of shelves is used. This shelving system is comprised of two 6.5” x 6.5” shelves made of foam-core board, placed horizontally, and held in place at each corner by hollow copper rods. The shelving system divides the interior of the spacecraft into three compartments and is easily removed from the spacecraft, allowing the crew to quickly make repairs or modifications to interior components.

To satisfy the secondary function, insulation from extreme cold, the structural materials were carefully chosen. The ¾” expanded polystyrene foam lined with Mylar foil has an R-value of 2.9. The Great Stuff insulating spray foam has an R-value of approximately 6 at a thickness of 1”. These two materials were chosen for their thermal insulating characteristics, their superior durability and strength, and for their low weight.

The copper rods assist in each of the three structural functions. They are each anchored into the floor and ceiling of the spacecraft, supporting the two shelves and adding strength to the spacecraft structure as a whole. The rods are also connected to the power supply, which allows them to serve as power rails to the electrical and thermal components of the spacecraft. This satisfies the tertiary goals of the spacecraft structure.

The spacecraft’s electrical components are secured and connected in circuits with printed circuit boards (PCBs) to reduce wiring and promote intelligibly. For ease of removability, external components are wired to the boards with screw headers. Figure 5 portrays the circuit board created for the thermal subsystem.

![Thermal-subsystem PCB.](image)

Figure 5: Thermal-subsystem PCB.

2.2.2 Power

Eight Efest Purple 18650 3100mAh 3.7V 20A batteries supply spacecraft power. The power system provides, via the onboard voltage regulator, 3.3V and 5.0V electric potentials to the electrical components and 7.4V to the thermal components (heating pads) of the spacecraft. Each of these potentials is supplied individually through three of the four copper tube power rails used simultaneously as structural support, while the fourth copper tube goes to ground.

The extreme cold of the upper troposphere and lower stratosphere has the potential to significantly reduce battery life and threaten the success of the spacecraft. To protect the batteries from the cold, a single heating pad is used in the power subsystem compartment to keep the batteries at a temperature of 10° C with a variance of approximately ± 5° C. A thermistor in the power compartment detects the current temperature and relays that information to the flight computer, which controls the status of the heating pads.
By a wide margin, the power subsystem is the heaviest of the spacecraft subsystems. To keep the center of gravity as low as possible, the batteries are secured in plastic single- or double-battery holders and fastened to the floor of the structure.

2.2.3 Thermal

The thermal subsystem consists of four heating pads powered at 7.4 volts. Each heating pad is controlled by an electrical relay system managed by the ATmega32u4 microcontroller on the Arduino Micro board. The heating pads consist of a mesh of polyester and conductive metal. When supplied with a small current (~1.0 Amp), the materials act as a resistor and produce heat. The electrical relay system is assembled on a custom printed circuit board.

Thermistors inside each of the three compartments of the spacecraft detect the local temperature and send that information to the ATmega32u4 microcontroller. The microcontroller is programmed to activate each heater individually if the local temperature falls below 10°C. Temperature regulation of the power subsystem is the top priority of the thermal subsystem.

2.2.4 Command & Data Handling

Spacecraft timekeeping, FC health monitoring, and data collection, processing, and storage are encompassed by the command and data handling (C&DH) subsystem [4].

The SparkFun Real-Time Clock (RTC) Module realizes timekeeping. The device is a breakout of the DS1307 microchip, a RTC that provides I2C-accessible date (year, month, day) and time (hour, minute, second) data. The chip updates the stored date and time data with the output of an on-board crystal oscillator. A small lithium battery placed on the underside of the breakout runs the RTC in the event of power failure.

Data collection and processing is accomplished with a ATmega32u4 microcontroller (Arduino Micro board). The microcontroller implements a 10-bit ADC that is utilized in obtaining readings from analog sensors (such as the UV sensor and thermistors discussed in following sections). Data is obtained from digital sensors (such as the MPL3115A2 pressure sensor) with the supported I2C two-wire serial protocol. Last, the microcontrollers’ digital pins are employed to collect data from the Geiger counter. The FC health-monitoring implement is the ATmega32u4’s built-in watchdog timer; in the event the chip fails to reset the timer in a certain amount of time (e.g. due to a lock-up), the computer is reset.

2.2.5 Communications

The communication subsystem is responsible for passing sensor data to a ground station for analysis via a UHF radio transmitter. The transmitter chosen for the project is a Radiometrix NTX2 70cm band transmitter. Its operational frequency is 434.650MHz. The data mode chosen for encoding the data is audio Morse code to be modulated by an Arduino Uno microcontroller. A twin lead J-pole has been fashioned to increase the likelihood of the low power transmitter being detected by the ground station while in flight.

The ground station is comprised of a laptop to display and record the decoded data, a Coastal Chipworks TNC-X terminal node controller to demodulate the data, a Baofeng UV5R handheld UHF/VHF dual band radio to detect the radio transmission, and a custom built, high gain, directional antenna to increase the likelihood of detecting the transmitter while in flight. If possible, an omnidirectional antenna will be used while mobile.

2.3 Payload

The hardware and software that sense the mission’s subject comprise the payload [4]. Seven devices form this mission’s payload: (1) two Geiger counters, (2) an ultraviolet (UV) light sensor, (3) solar panels, (4) thermistors, (5) a pressure and altitude sensor, (6) a triple-axis magnetometer, and (7) a triple-axis accelerometer. These items are discussed below.

2.3.1 Geiger Counters

The GCK-01-01 analog Geiger counter is responsible for measuring radioactivity. The radioactivity collected is alpha (above 3.0 MeV), beta (above 50 KeV), and gamma (above 7 Ke). The Geiger counter clicks and blinks an LED with each detection along with an output to a data chart. Two counters will be on the
payload, a shielded device along with a control to measure the difference in incoming particles. They may
be powered with a DC or AC voltage between 6 and 12 volts.

2.3.2 Ultraviolet-Light Sensor

Our payload is equipped with a ML8511 microchip breakout from Sparkfun. The ML8511 is a UV sensor
capable of reading UV-A and UV-B radiation and is most effective at reading 280-390nm light. The sensor
sends data by way of an analog output that can be read with an analog-to-digital converter (ADC). The
potential across the output and ground is generated by an internal amplifier that converts photo-current to
voltage.

The UV sensor is utilized to read UV radiation from the sun. To obtain an accurate reading, the sensor
must be in direct contact with the sun or the radiation. For this reason, the UV sensor was attached to
the outside of the payload. Also, depending on temperature, the UV intensity received from the sensor will
vary slightly, as shown in Figure 3.8. Since the data received from the UV sensor is a voltage reading, it is
necessary to convert the readings to intensity units (mW/cm² in this case).

2.3.3 Solar Panels

Attached to the exterior of our payload are PowerFilm® MPT6-75 flexible solar panel cells. We have one
panel attached to each exterior side of our payload, excluding the top and bottom, and thus having four in
total. Each MPT6-75 solar panel produces approximately 6V 50mA in standard conditions. Portrayed in
figure 6 is the MPT6-75 solar panels.

Although solar panels can be used as a power source, the payload has an insufficient amount to charge
the batteries properly. The payload instead utilizes the solar panels as sun sensors. When connected to an
analog input, the voltage from the panels can be determined. As providing four analog pins directly from our
micro-controller is impossible, we have instead put the panels onto the same 16:1 multiplexer the thermistors
are on.

![Figure 6: PowerFilm® MPT-75 flexible solar panel.](image)

By having a solar panel on each side of our payload, we will be able to determine the rate of revolution,
direction the payload is spinning, which direction the payload is facing at any given point in time, and the
amount of energy being produced from the solar panels from solar energy. Although we have included other
sensors that determine revolution, the solar panels will help verify these results.

2.3.4 Thermistors

External and internal temperature measurement is accomplished with a set of thermistors—ceramic- or
polymer-based resistors whose resistances change considerably with temperature [3]. Due to their low cost,
precision, and predictable temperature-response, thermistors are commonly applied as temperature sensors
[2]. Figure 7 on the following page shows one of the eight negative temperature coefficient (NTC) thermistors
placed in our payload.

The resistance of a NTC thermistor decreases nonlinearly as temperature increases, roughly according to
the B-parameter equation

\[
\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \ln\left(\frac{R}{R_0}\right)
\]

where \( T \) is the thermistor’s temperature in kelvin, \( R \) is the thermistor’s resistance in ohms, \( B \) is a constant
that depends on the thermistor, and \( R_0 \) is the thermistor’s resistance at a reference temperature \( T_0 \) [2].
The thermistors utilized in this mission have a $B$-value of $B = 3950$ K. Table 3 displays the thermistors’ resistances $R_0$, as measured by a multimeter, at a reference temperature of $T_0 = 292.1$ K.

The final unknown in the preceding equation (1), $R_s$, is determined with a 10-bit analog-to-digital converter (ADC) and a voltage divider; each thermistor is placed in series with a 10kΩ resistor across a 3.3V potential and the voltage drop across the resistor is measured with the 10-bit ADC of the flight-computer (an ATmega32u4 microcontroller). It can be shown that

$$R = (\frac{1023}{ADC_{out}} - 1)R_s$$

where $R_s$ is the measured resistance of the series resistor (this value may differ from 10kΩ due to tolerances) and $ADC_{out}$ is the digital output of the 10-bit ADC—an integer between 0 and 1023 ($= 2^{10} - 1$) that is directly proportional to a voltage between zero and the ADC’s analog reference (AREF) voltage. Due to the stability of the 3.3V power supply on the Arduino Micro board [2], that supply is used as the AREF voltage. Also shown in table 3 are the measured series resistance values corresponding to each thermistor.

<table>
<thead>
<tr>
<th>Thermistor ID</th>
<th>$R_0$ (kΩ)</th>
<th>$R_s$ (kΩ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>13.63</td>
<td>9.65</td>
</tr>
<tr>
<td>T2</td>
<td>13.63</td>
<td>9.99</td>
</tr>
<tr>
<td>T3</td>
<td>13.43</td>
<td>9.88</td>
</tr>
<tr>
<td>T4</td>
<td>13.25</td>
<td>9.90</td>
</tr>
<tr>
<td>T5</td>
<td>13.57</td>
<td>9.94</td>
</tr>
<tr>
<td>T6</td>
<td>13.28</td>
<td>9.95</td>
</tr>
<tr>
<td>T7</td>
<td>13.35</td>
<td>9.96</td>
</tr>
<tr>
<td>T8</td>
<td>13.37</td>
<td>10.04</td>
</tr>
</tbody>
</table>

Table 3: Thermistor parameters ($T_0 = 292.1$ K).

### 2.3.5 Pressure & Altitude Sensor

Portrayed in figure 8 on the following page is the implement for measuring pressure and altitude. The device is a SparkFun breakout of the MPL3115A2 I2C Precision Altimeter, a microchip utilizing a pressure sensor and a 24-bit ADC to generate serial-accessible pressure and altitude data. The chip also provides 16-bit temperature data via an internal temperature sensor.

The MPL3115A2 derives altitude from the pressure measurement and the 1976 U.S. Standard Atmosphere. The relation

$$h = 44330.77(1 - (\frac{P}{P_0})^{0.1902632})$$

results in the altitude $h$ (in meters) given by the chip where $P$ is the measured pressure (in Pascals) and $P_0 = 101326$ Pa is sea-level pressure [1]. Communication with the chip is handled with a C++ class written for import into the Arduino software. The chip is rated to a minimum operational pressure of 20kPa (which corresponds to an altitude of 12km). Thus, readings below that pressure value may be inaccurate.
2.3.6 Triple-Axis Magnetometer

Direction measurement is accomplished with a digital compass, specifically a SparkFun breakout of the HMC5883L triple-axis magnetometer. The device measures an ambient magnetic field’s strength in three mutually-orthogonal \(x, y,\) and \(z\) directions and converts the measurements to I2C-accessible digital data with a 12-bit ADC. The magnetometer’s gain is programmable between 0.73\(mG/\text{output unit}\) and 4.35\(mG/\text{output unit}\) to allow measurement of both relatively weak and strong magnetic fields. A gain of 0.73\(mG/\text{output unit}\) is used in our application to sense the earth’s magnetic field.

Given magnetic-field strengths in the \(x, y,\) and \(z\) directions, \(m_x, m_y,\) and \(m_z,\) respectively, the angle the overall magnetic field vector \(B\) makes in the \(xy\)-plane is

\[
\theta = \begin{cases} 
\arctan\left(\frac{m_y}{m_x}\right) & m_x > 0 \\
\pi + \arctan\left(\frac{m_y}{m_x}\right) & m_x < 0 
\end{cases}
\]  

The angle the vector makes with the \(z\)-axis is

\[
\phi = \begin{cases} 
\arctan\left(\frac{\sqrt{m_x^2 + m_y^2}}{m_z}\right) & m_z > 0 \\
\pi + \arctan\left(\frac{\sqrt{m_x^2 + m_y^2}}{m_z}\right) & m_z < 0 
\end{cases}
\]  

Finally, the intensity of the magnetic field is

\[
|B| = \sqrt{m_x^2 + m_y^2 + m_z^2}
\]  

These calculated quantities determine the direction of the spacecraft relative to the earth’s magnetic field given the readings from the magnetometer.

2.3.7 Triple-Axis Accelerometer

The implement for measuring acceleration is a SparkFun breakout of the ADXL335 triple-axis accelerometer. The device produces three voltage-outputs between zero and 3.3V that are directly proportional to the acceleration experienced in three mutually-orthogonal \(x, y,\) and \(z\) directions. The flight-computer’s ADC is utilized to convert the voltage readings to digital form for processing.

3 Test Results

Three types of testing have been conducted: Structural, Battery, and Power/Thermal.

The structural tests are meant to simulate takeoff, flight, and landing conditions. To simulate takeoff and flight conditions, the spacecraft and payload are spun overhead with a violent change in direction. To simulate landing conditions, the spacecraft and payload are dropped onto concrete from a height of 20ft as well as pitched down several flights of stairs. In each of the structural tests, the spacecraft performed well
sustaining only light scratches to the exterior and only minor interior damage. This interior damage was noted and adjustments were made to the payload securement plans.

Battery testing involves the comparison of three types of batteries in controlled conditions. The three types of batteries tested are all type 18650 3.7V batteries: Ultrafire 5000mAh, Efest 2500 mAh, and Efest 3100 mAh. Two batteries of each type are connected in parallel and the resulting voltage is applied to one of the 5cm x 10cm heating pads used in the thermal subsystem. Each test is run at room temperature (~22°C) until either the battery power is exhausted or the test reaches approximately three hours in length. The test results are in favor of the Efest 3100mAh batteries, which sustains a sufficient voltage over nearly three hours of testing.

The power/thermal test involves testing the Efest 3100mAh batteries powering the same heating pad in a cold environment, approximately 0°C. Again, the batteries sustained a voltage sufficient to power the heating pads over the course of nearly 2.5 hours.

![Figure 9: Battery and heater cold test.](image)

4 Summary and Conclusion

This mission’s goal is the provision of flight data to future DEMOSAT teams for use in constructing experiments for launch into the stratosphere. The Apogee payload collects radiation and weather data for this purpose with two Geiger counters, an ultraviolet (UV) light sensor, solar panels, thermistors, a pressure and altitude sensor, a triple-axis magnetometer, and a triple-axis accelerometer. The retrieved information will provide a basis for the design of future DEMOSAT experiments.

References


5 Appendix: Flight-Computer Code

```c
// ApogeePayload.ino
/
// Written by Wes Hileman on 22 March 2015
/
// Purpose: To control the Apogee payload during flight.
/
// INCLUDES
/
// Spacecraft-bus libraries
#include <avr/wdt.h>  // watchdog timer
#include <RTC.h>     // real-time clock
#include <MUX.h>     // multiplexer
#include <Heater.h>  // heating pads
#include <SendOnlySoftwareSerial.h>  // serial Tx for OpenLogs and LCD
/
// Sensor libraries
#include <Wire.h>    // I2C communication library
#include <MPL3115A2.h>  // altimeter
#include <HMC5883L.h>  // triple-axis magnetometer
#include <ML8511.h>   // uv sensor
#include <ADXL335.h>  // triple-axis accelerometer
#include <Thermistor.h>  // thermistor temperature-sensors
/
// CONSTANTS
/
// States for the FSM implementation
enum State { IDLING, COLLECTING_DATA, STORING_DATA, DRIVING_HEATERS, RUNNING_DIAGNONSTICS
};
/
// Aliases for multiplexer pins
enum MuxPin { M0, M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15
};
/
#define ANALOG_REF 3.3
#define WATCHDOG_TIMEOUT WDTO_8S
#define LOG_BAUD 9600
#define LCD_BAUD 9600
/
// Temperature control
#define BATT_CENTER_TEMP 10
#define BATT_EPSILON 3
#define INT_CENTER_TEMP 0
#define INT_EPSILON 5
/
// Timing
#define CYCLE_TIMEOUT 1000 // milliseconds
#define SLEEP_TIME 10 // microseconds
/
#define ALTIMETER_oversample_rate 7
#define COMPASS_AV_SAMPLES AV8
#define COMPASS_SR F3HZ
#define COMPASS_GAIN G1370
#define COMPASS_MODE CONTINUOUS
/
// =========== PIN MAPPING ===========
/
#define MUX_IO_PIN A0
#define DATA_LOG_TX_PIN 12
#define BATT_HEAT1_PIN 4
#define BATT_HEAT2_PIN 5
#define INT_HEAT1_PIN 6
#define INT_HEAT2_PIN 7
#define TEST_BUTTON_PIN 11
#define LCD_PIN 10
/
#define NUM_MUX_SELECT_PINS 4
// Multiplexer select pins
int muxSelectPins[ NUM_MUX_SELECT_PINS ] = { A1, A2, A3, A4
};
/
#define UV_PIN A5
```

#define ACCEL_X_PIN M13
#define ACCEL_Y_PIN M14
#define ACCEL_Z_PIN M15
#define BATT_THERM1_PIN M0
#define BATT_THERM2_PIN M1
#define INT_THERM1_PIN M2
#define INT_THERM2_PIN M3
#define EXT_THERM1_PIN M4
#define EXT_THERM2_PIN M5
#define EXT_THERM3_PIN M6
#define EXT_THERM4_PIN M7

// =========== END PIN MAPPING ===========

// OBJECTS

// Spacecraft-bus objects
RTC clock;
MUX mux( muxSelectPins, NUM_MUX_SELECT_PINS );
SendOnlySoftwareSerial datalog( DATA_LOG_TX_PIN );
SendOnlySoftwareSerial lcd( LCD_PIN );

// Thermistors that control heaters
Thermistor battTherm1( MUX_IO_PIN, 9650.0, 3950.0, 292.1, 13630.0, &mux, BATT_THERM1_PIN );
Thermistor battTherm2( MUX_IO_PIN, 9990.0, 3950.0, 292.1, 13630.0, &mux, BATT_THERM2_PIN );
Thermistor intTherm1( MUX_IO_PIN, 9880.0, 3950.0, 292.1, 13430.0, &mux, INT_THERM1_PIN );
Thermistor intTherm2( MUX_IO_PIN, 9900.0, 3950.0, 292.1, 13250.0, &mux, INT_THERM2_PIN );

// Heater objects (require thermistor objects)
Heater battHeater1( BATT_HEAT1_PIN, BATT_CENTER_TEMP, BATT_EPSILON, &battTherm1 );
Heater battHeater2( BATT_HEAT2_PIN, BATT_CENTER_TEMP, BATT_EPSILON, &battTherm2 );
Heater intHeater1( INT_HEAT1_PIN, INT_CENTER_TEMP, INT_EPSILON, &intTherm1 );
Heater intHeater2( INT_HEAT2_PIN, INT_CENTER_TEMP, INT_EPSILON, &intTherm2 );

// Sensor objects
MPL3115A2 alt;
HMC5883L compass( COMPASS_AV_SAMPLES, COMPASS_SR, COMPASS_GAIN, COMPASS_MODE );
ML8511 uvSen( UV_PIN, ANALOG_REF );
ADXL335 accel( ANALOG_REF, ACCEL_X_PIN, ACCEL_Y_PIN, ACCEL_Z_PIN, &ux, MUX_IO_PIN );

// Thermistors
Thermistor extTherm1( MUX_IO_PIN, 9940.0, 3950.0, 292.1, 13570.0, &ux, EXT_THERM1_PIN );
Thermistor extTherm2( MUX_IO_PIN, 9950.0, 3950.0, 292.1, 13280.0, &ux, EXT_THERM2_PIN );
Thermistor extTherm3( MUX_IO_PIN, 9960.0, 3950.0, 292.1, 13350.0, &ux, EXT_THERM3_PIN );
Thermistor extTherm4( MUX_IO_PIN, 10040.0, 3950.0, 293.2, 11900.0, &ux, EXT_THERM4_PIN );

void setup()
{
  // Initialize I2C communication
  Wire.begin();
  digitalWrite( 9600 );

  // ========= SPACECRAFT BUS =========

  // Initialize real-time clock
  clock.begin();
  // Initialize MUX I/O pin as input
  pinMode( MUX_IO_PIN, INPUT );
  // Initialize data logging
  datalog.begin( LOG_BAUD );
  // Initialize heaters
  battHeater1.begin();
  battHeater2.begin();
  intHeater1.begin();
  intHeater2.begin();
  // Initialize test-button input
  pinMode( TEST_BUTTON_PIN, INPUT );
  // Initialize lcd
  lcd.begin( LCD_BAUD );
  // Send calibration code to lcd
  lcd.print("?f?c0?G420?f");
// Set analog reference voltage to external AREF (3.3V regulator)
analogReference( EXTERNAL );

// Enable watchdog timer
wdt_enable( WATCHDOG_TIMEOUT );

// Print data-file header
datalog.println( "Time (ms), Timestamp, Pressure (Pa), UV Int (mW/cm^2), Comp Theta (deg), Comp Phi (deg), Comp Mag (gauss), X accel (g), Y accel (g), Z accel (g), Batt T1 (C), Batt T2 (C), INT T1 (C), INT T2 (C), EXT T1 (C), EXT T2 (C), EXT T3 (C), EXT T4 (C), Heat1, Heat2, Heat3, Heat4" );

// -------------- PAYLOAD --------------

// Initialize altimeter
alt.begin( ALTIMETER_OVERSAMPLE_RATE );

// Initialize magnetometer
compass.begin();

// Initialize UV sensor
uvSen.begin();

// Initialize accelerometer
accel.begin();

// end setup

void loop()
{
  // Update the start time of the latest cycle
  unsigned long lastCycleTime = millis();

  // String to store timestamp data
  char timestamp[ TIMESTAMP_LENGTH ] = "";

  // Variables to store sensor data
  float pressure, uv, compassTheta, compassPhi, compassMag, accelX, accelY, accelZ;
  float battTemp1, battTemp2, intTemp1, intTemp2, extTemp1, extTemp2, extTemp3, extTemp4;
  bool heat1State, heat2State, heat3State, heat4State;

  // Reset watchdog timer
  wdt_reset();

  // Collect data
  clock.timestamp( &timestamp[ 0 ] );
  pressure = alt.pressure();
  uv = uvSen.intensity();
  compass.data( &compassTheta, &compassPhi, &compassMag );
  accelX = accel.accelerationX();
  accelZ = accel.accelerationZ();
  battTemp1 = battTherm1.temp();
  battTemp2 = battTherm2.temp();
  intTemp1 = intTherm1.temp();
  intTemp2 = intTherm2.temp();
  extTemp1 = extTherm1.temp();
  extTemp2 = extTherm2.temp();
  extTemp3 = extTherm3.temp();
  extTemp4 = extTherm4.temp();

  // Drive heaters
  heat1State = battHeater1.cycle();
  heat2State = battHeater2.cycle();
  heat3State = intHeater1.cycle();
  heat4State = intHeater2.cycle();

  datalog.print( lastCycleTime );
  datalog.print( timestamp );
  datalog.print( pressure );
  datalog.print( "", "");
  datalog.print( "", "");
  datalog.print( "", "");
  datalog.print( compassTheta );
204  datalog.print( "", "");
205  datalog.print( compassPhi );
206  datalog.print( "", "");
207  datalog.print( compassMag );
208  datalog.print( "", "");
209  datalog.print( accelX );
210  datalog.print( "", "");
211  datalog.print( battTemp1 );
212  datalog.print( "", "");
213  datalog.print( heat1State ? "ON" : "OFF" );
214  datalog.print( "", "");
215  datalog.print( heat2State ? "ON" : "OFF" );
216  datalog.print( "", "");
217  datalog.print( heat3State ? "ON" : "OFF" );
218  datalog.print( "", "");
219  datalog.println( heat4State ? "ON" : "OFF" );
220  // Wait until the cycle times out
221  while( millis() - lastCycleTime < CYCLE_TIMEOUT )
222  {
223    if( digitalRead( TEST_BUTTON_PIN ) == HIGH )
224    {
225      runDiagnostics();
226    } // end if
227    delayMicroseconds( SLEEP_TIME );
228  } // end while
229
230  // end loop
231
232  void runDiagnostics()
233  {
234    // Clear screen
235    lcd.print( "?f" );
236    // Output diagnostic data
237    lcd.print( "IT'S A GO!?n" );
238    lcd.print( "------------------?n" );
239  } // end runDiagnostics

--

2015 COSGC Space Research Symposium  APOGEE