Spring 2017
Semester Report
for
Cislunar Explorers
Power Subsystem

Daniel Kim
dsk252

Table of Contents

Current Status and Semester Overview

Much of this semester involved working on testing each component of the flatsat and assembling the flatsat itself. The flatsat was tested at a system level in preparation for Ground Tournament 4 for general reliability of the system as well as operation in environmental conditions. Higher-level functions were also implemented (e.g. turn the electrolyzers on, fire the spark plug) using the lower-level power board functions.

At this point, all components of the flatsat has been tested and confirmed to be working. All components for the flight units are also gathered, but not get assembled. Code for the power board is also complete, but needs to be integrated with the rest of the flight software. What needs to be done next is assembly of the flight units as well as some tests that could not be completed since the flight units were not built at the time of testing. This will involve construction of 2 more flatsats that are identical to the EDU unit that is already completed, and integrating them into the structure of the spacecraft.

Flatsat Overview

Below is an overview of the entire flatsat system as well as a description of each component and its functions. A photo of the actual flatsat system can also be seen. More information on each of the components can be found by doing a quick search of the corresponding data sheet using the listed part number.

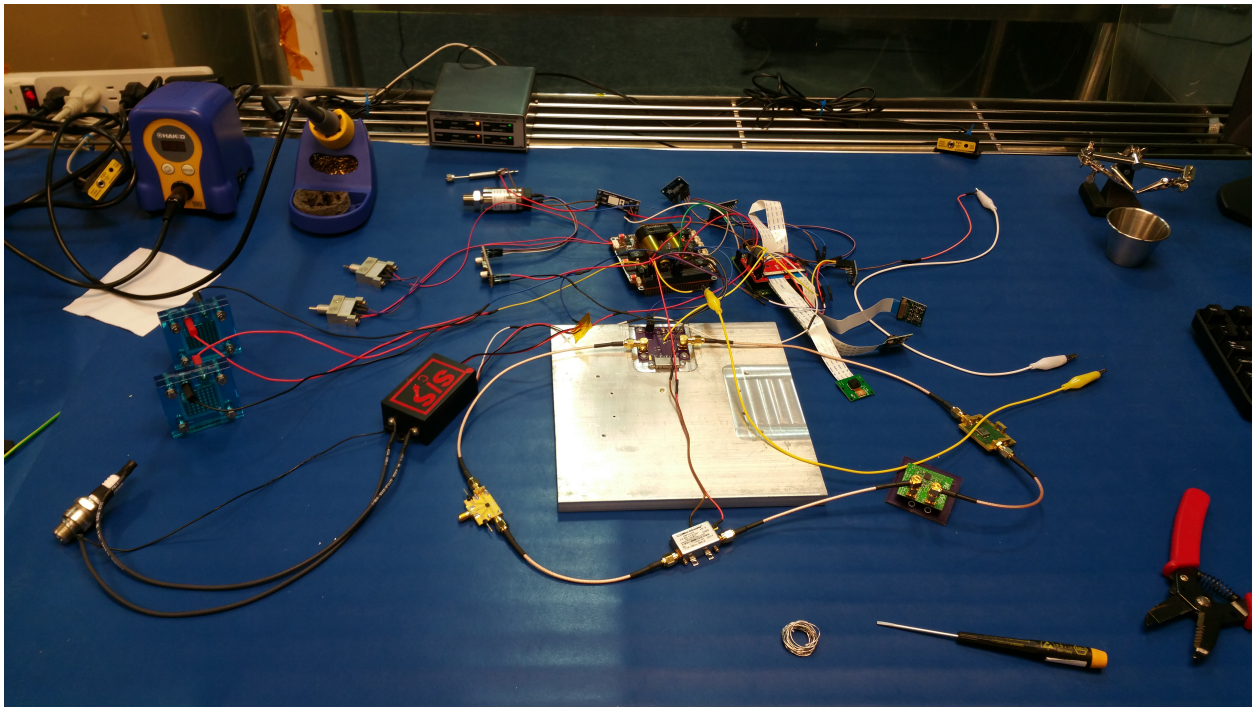

Figure 1 - Completed Flatsat

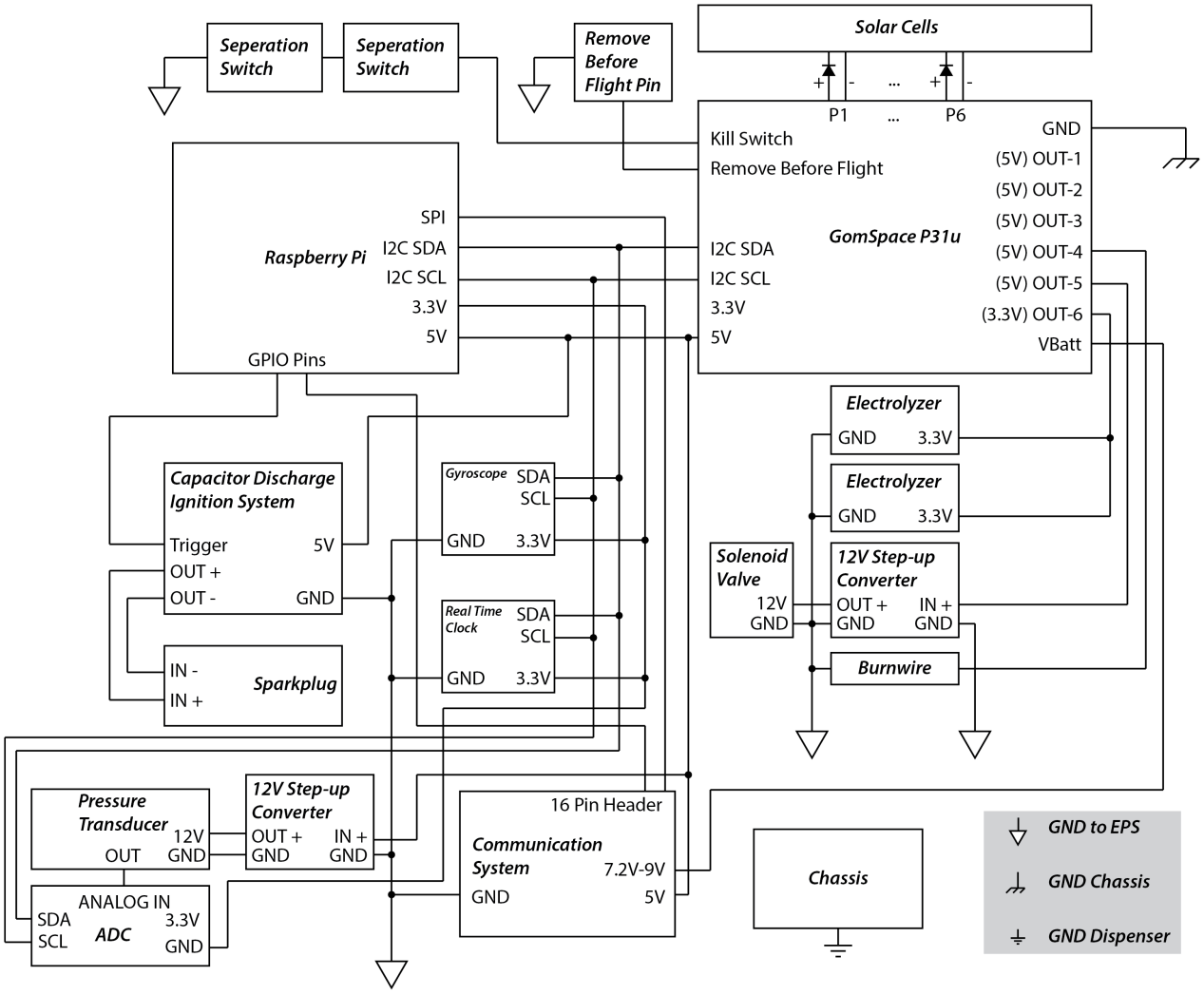

Figure 2 – Flatsat system-level view

Component	***Part Number***	***How it works***
Power Board	GomSpace P31U	The power board supports I2C communication, which can be controlled through the Raspberry Pi. More details can be seen in the following section on the various functions of the power board.
		Purpose
		Provide power to the whole system as well as store energy from solar panels. Primary uses include switching on and off certain outputs to turn components on and off.
Raspberry Pi	Model A+	At this point, controlling the Raspberry Pi is done through SSH. Python code can then be run, which can send various commands to the power board.
		Run flight software as well as control the power board via I2C.

Electrolyzer	Horizon Mini PEM Electrolyzer	Our setup with the electrolyzers currently involve first injecting water into the oxygen side (side with the red terminal) and then submersing them completely in water contained in a solo cup. Precautions must be taken in the clean room, so we also have a larger container in which the cup is placed. Then, 3.3V is applied to the terminals and bubbles should start to float up. There is a minor problem in which sometimes, bubbles do not immediately start to form, and the power board latches up. This is due to the fact that there is not enough water in the electrolyzer to begin with. Once the power board is reset and 3.3V is applied for a second time, the electrolyzers should run without a problem.
		The purpose of the electrolyzers is to convert the water in the tank into hydrogen and oxygen, which will be used as fuel.
Step-up Converter	Pololu U3V50F12	The voltage regulator takes a certain voltage on the input terminals (in our case, 5V) and boosts it to 12V. It also has an enable pin that, when pulled low, disables the converter meaning that the input voltage will simply be passed along to the output. The enable pin is set high by default via a pull-up resistor.
		The boost converters are used to provide 12V to the pressure transducer and the solenoid valve.
Capacitor Discharge System (CDI)	S/S Machine and Engineering CDI Module	The CDI requires an input voltage of 5V to charge the internal capacitor. The trigger line determines when the capacitor is discharged to the output terminals, and is activated when the trigger is pulled to ground and deactivated when pulled high. The red LED will light up once the capacitor is charged (which happens almost instantaneously).
		The output terminals of the CDI are connected to the sparkplug generate a large enough potential difference across its output terminals that it causes a spark to jump the gap of the sparkplug.
Sparkplug	Autolite Sparkplug	The sparkplug has a carbon core and a metal section with screw threads. The output of the CDI should be connected to these terminals. When the CDI is activated, there should be a visible spark that jumps the gap from the metal to the carbon.
		The sparkplug will be used to ignite the fuel produced by the electolyzers.
Solenoid Valve	IEPA1221141H	The solenoid has 2 terminals and is bipolar. 12V should be applied across these terminals and there should be an audible click indicating that the valve has opened. The valve should not be open for more than a couple milliseconds at a time due to the heat generated by holding the valve open, which can damage it. We have been able to use a spike-and-hold method to hold it open for longer periods of time, making use of the enable pin of the boost converter. First, the valve is opened with 12V for a few milliseconds. Then, the enable pin is driven low, causing the boost converter to switch off and pass on its input voltage (5V) to the output. This voltage can then safely hold the valve open for a few hundreds of milliseconds at a time.

		The solenoid is used to control the flow of gasses in the spacecraft. Refer to the propulsion team for more information on this as I did not work extensively with this component.
Burnwire	Nichrome Burnwire	The burnwire is basically a resistor that is designed to heat up when a current is sent through it. Currently, it is connected to 5V, but will more or less function at any voltage since it is just a resistor. We have found that due to the low resistance of the wire, it is hard to find the correct length of wire that allows the ideal amount of current to flow. Too much current results in the power board going into over-current protection mode while too little current is not enough to heat up the wire enough to cut the fishing wire. The best way to get the correct length of burnwire is to start with a long length, then shorten the wire little by little until it is able to cut through the fishing wire.
		The purpose of the burnwire is to separate the 3U cubesats once it is released from the dispenser and in outer space. The general setup is such that one of the cubesats will have a loop of fishing wire, which the burn wire will loop through. The burnwire is connected to the other 3U cubesat. When activated, the burnwire melts through the fishing wire and the two cubesats will be freed.
Communication System	Various Boards	The communication system is made up of several components including an amplifier, an antennae board, an RF switch, a PA board, and a board that interfaces with the Raspberry Pi. The amplifier requires 5V to operate and the PA board requires between 7.2V and 9V to operate. Currently, VBatt is connected to this component. The communication is a subsystem of itself; refer to the communications subteam for more information.
		The general purpose of the communications system is to transmit and receive to and from the ground station.
Separation Switch	2002204A	The separation switches have two terminals that are normally closed and two that are normally open. Depressing the pin on the switch will cause the normally closed terminals to open and the normally open terminals to close. Currently, two switches are wired in series and connect the kill switch of the power board to ground.
		These switches are used to ensure that the spacecraft is turned off from the time they are loaded onto the dispenser of the spacecraft to when they are released from the dispenser into space. The switches are depressed when they are loaded into the dispenser.
Pressure Transducer	IPSU-GP300-6	The pressure transducer has 4 terminals: ground, supply voltage (12V), positive output voltage, and negative output voltage. In our setup, negative output voltage and ground are shorted together. The positive output voltage ranges from 0V to 5V and is proportional to the pressure it senses. This output voltage is connected to the analog terminal of the ADC.
		The pressure transducer will take measurements of the pressure in the gas storage units of the spacecraft. Refer to the propulsion team for more information on this.
Analog-Digital Converter (ADC)	ADS1115	The ADC requires 3.3V to operate, and has an analog input terminal and is I2C compatible. The analog data can then be read by the Raspberry Pi.

		The purpose of this is to read the voltage from the pressure transducer and convert it into a digital value that can be accessed by the Raspberry Pi via I2C and converted into an actual pressure measurement.
Remove Before Flight (RBF) pin	SS-5GL13T	The RBF pin is not currently integrated into the flatsat. However, it is simply a contact that can connect the RBF pin of the power board to ground. When this occurs, the power board is prevented from turning on, even when the separation switches are activated.
		The purpose of this is to prevent the spacecraft from turning on prior to being placed in the dispenser. Once the spacecraft is inside the dispenser and the separation switches are depressed, the RBF pin will be removed.
Gyroscope	Adafruit 9-DOF IMU Breakout - L3GD20H + LSM303	The gyroscope requires a 3.3V supply voltage and connecting the SDA and SCL lines to the Raspberry Pi.
		The gyroscope will be used to track the orientation of the spacecraft as well as used in calculations for image tracking to determine the orientation of the spacecraft using perceived positions of the sun, moon, and earth. Refer to the optical navigation subteam for more information on this.
Real Time Clock	Adafruit DS3231 Precision RTC Breakout	The RTC requires 3.3V to operate as well as connecting the SDA and SCL lines to the Raspberry Pi for I2C communication.
		The real time clock will be used in flight software for scheduling purposes.

Power Board Operation

For the following descriptions, refer to the GomSpace P31U data sheet titled “gs-ds-nanopower-p31u-13" which can be found on the Cornell Box.

Turning it on

To turn the power board on, the orange colored jumpers (connector P7) must be switched from the parallel position to perpendicular with respect to the side of the board. This connects the batteries to the rest of the power board circuitry. The kill switch (connector P10) must be then shorted to ground to turn the system on. After the switch has been shorted, the system will remain on regardless of whether the kill switch has been shorted to ground or not.

Charging Batteries Through the Charge Pin

To charge the batteries using the charge pin, use the lab power supply and set the voltage at 5V with a current limit of 1A. Make sure the power board is turned off (jumpers are connected, but kill switch has not been shorted to ground). Connect the 5V output to pin 6 on connector P8 using a molex connector. Ground can be found on the same connector (P8) on pin 1. Once connected, the power supply should read 5V with a current draw of around 0.7A. When the batteries finish charging, current draw will automatically drop to 0A. Battery voltage should be checked periodically

Charging Batteries through Solar Cell Inputs

To charge batteries using the solar cell inputs, again use the lab power supply and set the voltage to 5V with a limit of 1A. Make sure the power board is turned on (jumpers connected and kill switch has been shorted to ground). Connect this output to connector P1, pin 4 and ground to pin 1 using a molex connector. The power supply should read around 3.7V at 1A. The system will stop drawing current once the batteries are fully charged.

Currently Assigned Output Pins

Several pins of the power board have been assigned to specific components of the flatsat and it is important that none of these are switched at any point since the power board software relies on this configuration (this mainly only applies to the switchable outputs). Below is a table listing the pins, and their assigned components. The power board has 6 outputs, 5 of which are 5V and 1 of which is 3.3V, which can be switched on and off. Additionally, it has 4 permanent outputs, 2 of which are 5V and 2 of which are 3.3V. When the power board is off, all of these outputs read 0V.

Pin Number	***Description***	***Function***	***Component***
H2-25	+5V	Permanent output , Raspberry Pi power	Raspberry Pi
H2-26	+5V	Permanent output, pressure transducer power	Pressure Transducer
H1-47	OUT-1	Switchable output, 5V	Amplifier
H1-48	OUT-4	Switchable output, 5V	Burnwire
H1-50	OUT-5	Switchable output, 5V	Solenoid
H1-52	OUT-6	Switchable output, 3.3V	Electrolyzer

Power Software Operation

Software for the power board can be found on my Github repository, at: https://github.com/danielkim802/SSDS

Useful Tips and Getting Started

To start using the Raspberry Pi to control the power board, we must first SSH into the Pi. To do this, first power on the Raspberry Pi. To SSH into the Pi, its IP address must be known. The Pi is currently set up to send an email with the IP information upon startup. To get the Pi to send the information to an email that is not the current email, modify the python file that is in the desktop of the Pi, titled "sendip.py" and replace the email and password variables with the appropriate strings. Once this is completed, the gmail account privacy settings may need to be changed to accommodate for this. Once the IP address is known, SSH into the Pi using Putty, or some other SSH client.

Before running the power software, the pigpiod daemon must be run. To do this, run the command `sudo pigpiod` from the Pi. Then, navigate to the directory titled SSDS. From there, python can be started and the power code can be run.

A useful command to know is one that detects currently connected devices. To check which I2C devices are connected to the Raspberry Pi, run `sudo i2cdetect -y 0` to display a list of addresses of the current I2C devices. Additionally, it may be helpful to

become familiar with a text editor in the terminal such as vim or nano as well as becoming familiar with basic terminal commands.

Currently Assigned Pins

Like the power board, there are certain pins that are assigned to the Raspberry Pi that the power software relies on. These should not be changed, and if they are, the corresponding pin definitions in the code should be changed as well. Below is a table showing which GPIO pins are assigned to which component. Also, note that GPIO pin numbers are not the same as the pin number on the Raspberry Pi header (e.g. pin 7 on the header corresponds to GPIO 4).

Pin Number	***Description***	***Component***
Pin 7	GPIO 4	Sparkplug, CDI trigger
Pin 11	GPIO 17	Communication system, PA board
Pin 38	GPIO 20	12V boost converter enable pin for the solenoid valve

Function Descriptions

The power software allows the Raspberry Pi to control the power board by sending commands through I2C communication. The following table contains descriptions of each of the functions, what they do, and how they can be used. To use these functions, a power object must first be created. All of these functions can then be called with respect to that object. The code for these functions is contained in the file titled "power_controller.py."

Function Name	***Inputs***	***Output***	***Description***
displayAll	void	void	Prints out all housekeeping data on the power board system.
write	cmd = 1 byte values = byte list	void	Writes the given data [values] to the specified command register [cmd].
read	bytes = int	byte list	Reads the specified number of bytes [bytes] from the current command register. To read from a specific command register, the command register must be written using the write function, then the read function must be called.
ping	value = 1 byte	1 byte	Writes a value [value] to the power board reads the same value back. This function is useful for sanity checks to test whether I2C communication is working.
get_hk_1	void	struct hkparam_t	Returns the corresponding housekeeping struct.
get_hk_2	void	struct eps_hk_t	Returns the corresponding housekeeping struct.
get_hk_2_vi	void	struct eps_hk_vi_t	Returns the corresponding housekeeping struct.
get_hk_out	void	struct eps_hk_out_t	Returns the corresponding housekeeping struct.
get_hk_wdt	void	struct	Returns the corresponding

		eps_hk_wdt_t	housekeeping struct.
get_hk_2_basic	void	struct eps_hk_basic_t	Returns the corresponding housekeeping struct.
set_output	byte = 1 byte	void	Sets the corresponding switchable outputs on or off depending on which bits are 1 and which are 0 in the given byte [byte].
set_single_output	channel = 1 byte value = 1 byte (1 or 0) delay = 2 bytes (seconds)	void	Sets a single switchable output on or off. [channel] selects which output to change. For switchable outputs, use 0~5 for outputs 0 through 5. For the BP4 heater, use 6. For the BP4 switch, use 7. [value] is either 1 or 0, indicating on or off. [delay] is in seconds and indicates the delay, after which point the command is executed.
set_pv_volt	volt1 = 2 bytes (millivolts) volt2 = 2 bytes (millivolts) volt3 = 2 bytes (millivolts)	void	Sets the voltages on the corresponding photovoltaic inputs.
set_pv_auto	mode = 1 byte (0, 1, or 2)	void	Sets the solar cell power tracking mode: 0 = hardware default power point, 1 = maximum power point tracking, 2 = fixed software powerpoint
set_heater	command = 1 byte (1 or 0) heater = 1 byte (0, 1, or 2) mode = 1 byte (1 or 0)	2 bytes	Returns a byte array with heater modes. [command] sets the heater on or off. For [heater] values: 0 = BP4, 1 = onboard, 2 = both. For [mode] values: 0 = off, 1 = on.
get_heater	void	2 bytes	Returns a byte array with the heater modes.
reset_counters	void	void	Resets the boot counter and WDT counters.
reset_wdt	void	void	Resets (kicks) the dedicated WDT.
config_cmd	command = 1 byte (1)	void	Use this command to control the config system: set [cmd] to 1 to restore the default configuration.
config_get	void	struct eps_config_t	Gets the current configuration struct.
config_set	struct = struct eps_config_t	void	Sets the current configuration as the input struct [struct].
hard_reset	void	void	Send this command to perform a hard reset of the P31, including cycling permanent 5V and 3.3V and battery outputs.
config2_cmd	command = 1 byte (1 or 2)	void	Use this command to control the config2 system: for [cmd], 1 = restore default configuration, 2 = confirm current configuration
config2_get	void	struct eps_config2_t	Gets the current configuration of the system.
config2_set	struct = struct eps_config2_t	void	Use this command to send config 2 to the P31 and save it (remember to also confirm it)
pulse	output = 0~5 duration = milliseconds	void	Pulses output [output] high on the power board for some amount of

	delay = seconds		milliseconds [duration]; called after a delay of [delay] seconds. *Caution: the output in question must be low before the function is called.
pulse_pi	output = 0~5 duration = milliseconds delay = seconds	void	Pulses output [output] high on the Raspberry Pi for some amount of milliseconds [duration]; called after a delay of [delay] seconds. *Caution: the output in question must be low before the function is called.
electrolyzer	switch = bool delay = seconds	void	Switches on the electrolyzers if [switch] is true, and off otherwise after a delay of [delay] seconds.
solenoid	spike = milliseconds hold = milliseconds delay = seconds	void	Spikes the solenoid for some number of milliseconds [spike] and holds at 5v for [hold] milliseconds with delay of [delay] seconds. *Caution: output must be off before the function is called.
sparkplug	duration = milliseconds delay = seconds	void	Pulses sparkplug for some number of milliseconds [duration] with delay of [delay] seconds. *Caution: output must be off before the function is called.
burnwire	duration = seconds delay = seconds	void	Turns burnwire on for [duration] seconds, with a delay of [delay] seconds.
comms	transmit = bool	void	Turns on the PA board if [transmit] is true, and off otherwise.
amplifier	on = bool	void	Turns on the amplifier if [on] is true, and off otherwise.
adc	t = seconds n = int gain = int	int list	Reads [n] values from the ADC with a delay of [t] seconds in between readings with a gain of [gain] and outputs the result in a list.
rtc	t = seconds n = int	int list	Reads [n] values from the RTC with a delay of [t] seconds in between readings and outputs the result in a list.
gyro	dt = seconds	void	Prints out readings from the gyroscope with a delay of [dt] seconds in between. *Hint: this function uses an infinite while loop. Use Ctrl+C to cancel the current process.
display_sensors	t = seconds	void	Displays readings from the gyroscope, pressure transducer, and real time clock with a delay of [dt] in between readings. Make the terminal window as wide as possible for this to display correctly. *Hint: this function uses an infinite while loop. Use Ctrl+C to cancel the current process.
nasa_demo	void	void	Used for the NASA demo. Demos each of the components in the following

			order: electrolyzers, sparkplug, solenoid, burnwire, and showing the sensor data. *Hint: this function uses an infinite while loop. Use Ctrl+C to cancel the current process.
capture	void	void	Captures pictures using the multplexer and the three cameras.
chamber	name = string t = seconds	void	Records housekeeping data from the power board every [t] seconds and stores the data in a text file titled [name]. *Hint: this function uses an infinite while loop. Use Ctrl+C to cancel the current process.

General Checks

There are a few general checks that should be completed when working with the power board. The first is to get and save the house keeping data. Housekeeping data should be saved when first starting to work with the power board as well as right before finishing work with the power board. To do this, call the function *displayAll* and copy/paste the result into a text file, with the date as the title. The housekeeping data should look like something along these lines:

```
***************-HOUSEKEEPING-***************
Photo-voltaic inputs:        1-348mV 2-357mV 3-359mV
Total photo current:         13mA
Battery voltage:             7888mV
Total system current:        190mA
Temp of boost converters:    1-22degC 2-23degC 3-22degC batt-22degC
External batt temp:          1-0degC 2-0degC
Latchups:                    1-[0] 2-[0] 3-[0] 4-[0] 5-[0] 6-[0]
Cause of last reset:         Brownout or power-on reset
Number of reboots:           43008
Number of software errors:   2
PPT mode:                    0
Channel output:              00000000
***************-CONFIG-***************
PPT mode:                    FIXED[2]
Battheater mode:             MANUAL[0]
Battheater low:              0degC
Battheater high:             5degC
Nominal mode output value:   1-[0] 2-[0] 3-[0] 4-[0] 5-[0] 6-[0] 7-[0] 8-[0]
Safe mode output value:      1-[0] 2-[0] 3-[0] 4-[0] 5-[0] 6-[0] 7-[0] 8-[0]
Output initial on:           1-[0s] 2-[0s] 3-[0s] 4-[0s] 5-[0s] 6-[0s] 7-[0s] 8-[0s]
Output initial off:          1-[0s] 2-[0s] 3-[0s] 4-[0s] 5-[0s] 6-[0s] 7-[0s] 8-[0s]
PPT point for boost conv:    1-3700mV 2-3700mV 3-3700mV
***************-CONFIG2-***************
Batt Max Voltage:            8300mV
Batt Safe Voltage:           6700mV
Batt Critical Voltage:       6500mV
Batt Normal Voltage:         7000mV
```

Check that the photo-voltaic inputs read around 350mV. Check the battery voltage. Although it is safe as low as 7.2V, batteries should ideally be charged once the voltage reaches around 7.5V. The max voltage for batteries is 8.3V. Total system current can vary, depending on which components are turned on, but nominally, with nothing turned on, it should read around 200mA. Check that the temperature of the boost converters is around room temperature. Check that there are no latchups (all of the readings are 0). If any of the outputs are nonzero, that means the corresponding output tried to supply too much current and the system went into overcurrent protection mode. Calling the function *reboot* resets the system and fixes this problem, given that the source of overcurrent is removed. Previously saved housekeeping data can be found on the Cornell Box.

When finished working with the power board, make sure that the orange jumpers are placed back in the disconnected orientation (parallel with the side of the board).

Tests to be Completed

Certain tests could not be fully completed due to the fact that the structure was not available at the time of testing. The following tests should be completed once the flight units are completed: CQP-TQR-047, CQP-TQR-048, and CQP-TQR-051, which can be found on Cornell Box.