Grover II

Project Specification Document

Created By

Sean Mayes
Yoni Gefen
Mike Comberiate

September 1, 2011

Table of Contents

1. Revision History

Date	Description	Contributor
08/16/2011	Initial Creation	Sean Mayes, Yoni Gefen
08/17/2011	Complete Document Update	Sean Mayes
09/01/2011	Complete Document Update	Sean Mayes
09/02/2011	Added 5.1 Functional Unit Description	Sean Mayes
09/03/2011	Added Highlighted Sections Throughout	Mike Comberiate
09/04/2011	Edited Maple Control	Yoni Gefen
09/04/2011	Added Python Maple Communication	Yoni Gefen
09/06/2011	Added Electronic Assembly	Sean Mayes

2. Introduction

Grover II is design to operate autonomously in Greenland for approximately three months. During this time, a group of scientist will attach a ground penetrating radar to the Grover to collect data for research.

2.1 Where Grover II stands

Currently Grover II is not fully operation for fieldwork.

- Mechanical
  Although the mechanical side could use some improvements, the Grover is mechanically complete.
- Electrical
  Although the electrical side could use some improvements, the Grover is electrically complete.
- Software
  The Grover can be driven through either a direct link to the onboard motherboard by attaching a monitor, keyboard, or mouse; or through the remote desktop using the wireless network GROVER. Grover has an IP address of 10.42.43.1. After turning Grover on, open the terminal on the laptop. Simply type in "vncviewer 10.42.43.1". When prompt for a password enter "Grover". Once you have established the remote desktop link, run "main" either through the terminal or eclipse.

    'w' – Forward
    's' – Backwards
    'a' – Turn Left
    'd' – Turn Right

  The Grover does NOT do zero point turns. If a zero point turn is attempted, the torque will rip apart the frame.

  Side Note:
  All the sensors attached to the electrical system are working properly. However, the code to read from the sensors and graph the outputs is not

working properly. I believe the bug to be small because the graphs were working perfectly when tested earlier. The bug is more than likely caused from reading the wrong pins for the graph.

2.2 What Next?

Grover should have four modes of control operation; Manual control, wifi control, iridium control, and autonomous control. All mechanical and electrical task should be complete.

- **Manual Control**: Physical buttons attached to the control-board that enables a user to move Grover. Forward, backwards, left, and right. When the user depresses one of the buttons, the robot moves in the corresponding direction. If no button is pressed, the Grover stops. This enable the user to easily maneuver the robot in tight spaces – like loading into the back of the truck.
- **Wifi Control**: The motherboard is equipped with a wireless card. Although, we can currently control the robot through remote desktop, this method is not preferred. Either a Web App or a client/server setup is required. A control GUI can be built to run on one laptop and transmit signals required to control the robot through the wireless signal. This is beneficial in that it is cheaper than iridium communication and will run anywhere there is a wireless signal. (wireless antenna needs to be adapted to be placed outside the electrical box)
- **Iridium Control**: The ability to send directional commands through the iridium is vital. Since the iridium delays range from 1 minute to 18 minutes for communication, it is not feasible to send immediate directional commands. However, you should be able to send it GPS coordinates. Have the vehicle move from GPS point A to GPS point B, or command the Grover to initiate a particular pattern. The protocol for using the Iridium system is quite different from the other communications methods described above. We have separate documentation on that system. Suffice it to say that this system has limited access to the GROVER but it is available from anywhere on Earth. The idea is that when Iridium is being used the GROVER will be autonomous like a spacecraft – executing commanded functions with limited monitoring.
- **Autonomous Control**: In autonomous mode, the Grover operates completely on its own – As in it follows a pattern or coordinate that it is programmed with. During normal functionality, it will monitor and record data from all the sensors every ‘x’ amount of time. If something were to go amiss, such as temperature to high, voltage to low, loss of internal or external communication, the Grover would initiate programmed protocol – that is, turn on fans, go into sleep mode, reset maples to reestablish communication, etc. During autonomous mode, everything will be logged and alerts sent back to base station through the iridium satellites.

3 Electronic Assembly

1. Step One: Verify all components and connection of empty electrical box
    a. The empty box contains:
    b. 6 Fan units (each fan unit contains two fans)
        i. Note: Blue and Yellow wires on fans are not used
    c. One terminal lug strip
    d. Connectors
        i. Motor Left
        ii. Motor Right
        iii. Solar panel Left
        iv. Solar panel right
        v. Wind turbine left
        vi. Wind turbine right
        vii. Battery left
        viii. Battery Right
        ix. 4 Light Sensor Connectors (3 wire each – black, white, red)
        x. Modem antenna
        xi. Four extra wires (2x black and 2x red)
        xii. Multiple extra connectors and ports for wires
    e. To Control Panel
        i. PC On/Off
        ii. PC Power
        iii. Maple power
        iv. 12 volt regulator input
        v. 24 volt line out (multiple 24V lines return in)
        vi. Sabertooth
        vii. CS in (Main battery Current Sensor)
        viii. CS out (Main battery Current Sensor)

2. Step 2:
    a. Clear space for large MPPT Plexiglas unit and drop down in electrical box like shown in photo.
    b. Note: current sensor 5 & 6 (CS 5 and CS 5) wires are wired beneath and to the side of the Plexiglas board.
    c. Confirm that the Plexiglas is not sitting on top of any wires.

3. Step 3 Connect Wires
    a. Connect large 2x5 Anderson block on right-side.
    b. Connect smaller 2x3 Anderson block on left-side.
    c. Connect yellow Anderson to other yellow Anderson (Supplies 24 Volts to 12 Volt regulator)
    d. Connect Left and Right motors (2x1 Anderson blocks)
        i. Note: motors are not polarized however for the sabertooth commands to work properly, they need to be plugged in correctly.
    e. Confirm all wires wrap around side of Plexiglas and not over the top.

4. Step 4

a. Plug Space Hut USB cable into the Maple board
b. Clear space for maple control board
c. drop in maple control board
d. confirm no wires are being crushed by Plexiglas
e. Plug in blue & black Anderson 12 volt regulator output (Green and red twisted wires)
f. Plug in Maple ground wire (Black anderson with green wire)
g. Plug in 2x2 Anderson block for Maple & PC power and PC on/off switch into 2x2 Anderson block (All red Anderson connectors, match solid black permanent marker line)
h. Screw in modem antenna. (there is only one gold connection for it to screw into)
i. Plug in 3x12 female header fan connectors to Cobra's 3x12 male headers. (green and white twisted wires)
    i. Note: row of green and white wires are closest to maple board. Two red wires are attached to the bottom row farthest from Cobra maple.
j. Plug in 3x5 female header current sensors into Watch Dog male headers.
    i. White signal wires are closest to Watch Dog
    ii. Use pins 1 – 5
    iii. There are only 5 current sensors being monitored at this time

k. Plug in 3x1 female header voltage sensor into Cobra male headers.
    i. White signal wire is closest to Cobra
    ii. Use pin 1
    iii. There is only 1 voltage sensors being monitored at this time
l. Plug in 2x1 female header TX & GND wires from Sabertooth into Shamrock 2x1 male header TX 2 and GND (Orange & white stripe TX wire and Gray GND wire)
m. Plug in 3x4 female header temperature sensor wires into Cobra's Male headers
    i. White signal wire is closest to Cobra
    ii. Use pins 1 – 4
    iii. There is only 4 temperature sensors being monitored at this time
n. Plug in the individual temperature sensors
    i. The front of the Grover is compartment A or 1
    ii. The back of the Grover is compartment D or 4
    iii. Note: the temperature sensors have both a humidity port and a temperature port. Plug into temperature port. One temperature sensor per compartment. (you may have to trace wire from Cobra maple board to correct compartment. i.e. Pin 1 goes to compartment 1)
o. Everything **except** for Temp 4, PC On/Off (blue and white wire at the end), and Python should be plugged in

5. Step 5

a. Plug in 2x1 female headers from python PC On/Off (blue and white wires) into motherboard 2x5 male headers.
    i. White wire is signal, blue wire is ground.
    ii. See diagram below

b. Clear space for Python motherboard to slid in
c. Connect Python power (red & black Andersons with green & red twisted wires) to Python motherboard power (red & black Andersons with black & white wires)
d. Plug Temperature sensor 4 into compartment 4's temperature sensor

6. Step 6
    a. Plug Shamrock, Watch Dog, and Cobra's USB's into the maples
    b. Plug Other end of USB into Python motherboard (Ports are dynamically assigned in the code – therefore, it does NOT matter which port the usb is plugged into on the motherboard.
7. Step 7
    a. Place all wires along the upper sides of the Plexiglas boards.
    b. Confirm no wires are draped over the top of the Plexiglas boards
    c. Verify there are no loose wires and that everything is connectd
8. Step 8
    a. Flip main power switch
        i. The fans should all turn on HIGH
        ii. Outter four fans should blow out of electrical box
        iii. Inner two fancs should blow into electrical box
        iv. Sabertooth should have a green light LED on
    b. Flip Maple switch
        i. Verify that all maples have a blue blinking light when turn on.
        ii. Note: the blue LED's only blink for a few seconds. You may have to turn off and on the maple switch to confirm all maples are working.
    c. Flip PC Power switch
        i. Motherboard fan will twitch for a second. (Fan will not turn on)
        ii. Green LED will turn on and remain on while power is supplied
    d. Press PC On/Off button
        i. Motherboard fan will turn on

4. Top Level Description
4.1 Hardware
4.1.1 Motherboard

- ASUS AT3GC-I
  - Integrated Dual-core Intel® Atom™ processor 330
  - Voltage(V): DC 12V
  - 1 x VGA
  - 1 x PS/2 Keyboard
  - 1 x PS/2 Mouse
  - 1 x LAN(RJ45) port
  - 8 x USB 2.0 (4 ports at mid-board, 4 ports at back panel)
  - 1 x COM port
  - 6 - Channel Audio I/O
  - 1 x Parallel port

  - http://www.asus.com/Motherboards/Intel_CPU_on_Board/AT3GCI/

4.1.2 Maple Board

- Maple - Core: ARM 32-bit Cortex™-M3 CPU
  - 128 Kbytes of Flash memory
  - 3 USARTs

- 1 USB 2.0 full-speed interface

4.2 Hardware Functions

4.2.1 Python (Motherboard) Functions

Short Description: When Grover II is operating at 100%, Python is in complete control over the entire system. All communications, obstacle avoidance systems, and data collections go through Python.

- **Communicates with**
  - Space Hut
  - Watch Dog
  - Shamrock
  - Cobra
  - Radar
- **USB GPS**
  - Longitude
  - Latitude
  - Altitude
  - Sun location
- **Obstacle Avoidance**
  - Ultrasonic Sensors
  - Bump Sensors
  - Encoder
- **Heartbeat**
  - Confirm all devices are working properly

4.2.2 Space Hut (Maple) Functions

Short Description: Enables communication between external location and Grover II through the iridium modem, passing commands to other devices in the robot. It is the only external communication to the Grover when no wireless signal is available. It gathers data from on board sensors and returns the data back to external location.

- **Communicates with**
  - Python
  - Watch Dog
  - Shamrock
  - Iridium modem

4.2.3 Shamrock (Maple) Functions

Short Description: Shamrock is the only microcontroller connected to the motor controller (Sabertooth). It controls all physical motion of Grover II, along with

basic obstacle avoidance. Basic obstacle avoidance should be able to run whether or not Python is running.

- **Communicates with**
  - Python
  - Space Hut
  - Watch Dog
- **Basic Obstacle Avoidance**
  - Ultrasonic Sensors
  - Bump Sensors
  - Cyclometer (encoder-like operation)
- **Navigation**
  - Saber tooth – Motor Controller
  - Control all motion of Grover II
  - Receive directional commands from Python or directly Space Hut

4.2.4 Watch Dog (Maple) Functions

Short Description: Monitors all diagnostic information onboard Grover II.

- **Communicates with**
  - Python
  - Space Hut
  - Shamrock
  - Cobra
- **Diagnostics**
  - Current sensors
    - MPPT Output
    - Front / Back – Turbines
    - Left / Right – Motors
    - System input (battery output to the system)
  - Light sensors
    - Sun tracking
- **Actuators**
  - 4x Servo control

4.2.5 Cobra (Maple) Functions

Short Description: Cobra is a diagnostic system for monitoring the condition of Grover II.

- **Communicates with**
  - Python
  - Watch Dog
- **Diagnostics**
  - Temperature sensors

        - 2x Left battery box (Not installed, although ports are Available)
        - 2x Right battery box (Not installed, although ports are Available)
        - 4x Main electrical box
    - Voltage Sensors
        - System input (both batteries)
        - Left / Right – Solar Panels (Not installed although ports are Available)
    - Cooling Fans
        - 2x Input to main electrical box
        - 4x Output to main electrical box
- **GPS** (Not installed although ports are Available)
    - Longitude
    - Latitude
    - Altitude
    - Sun location

5 Functional Block Descriptions

5.1 Functional Unit Description

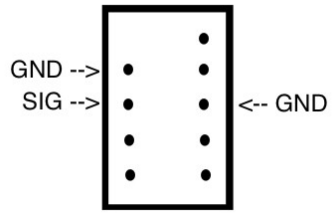
The Grover Power Control box is broken up into 4 switches/buttons. The **Main Power** switch is connected directly to the batteries. When this switch is in the off position, all power to the Grover is disconnected. The MPPT and the wind turbines are the only components attached to the battery at all time – regardless of switch positions. In this configuration, the solar panels and wind turbines will continue to charge the batteries in all states. Both the solar panels and wind turbines are self-regulated to protect over charging the batteries. When the **Main Power** switch is turned on, power is provided to the motor controller and the 12V 8.3A regulator. The 12V regulator is then connected to all six fans through an 8A fuse.

The **Maple Power** switch provides power to all maples, sensors, and the iridium modem through a 5V 4A regulator. When **Maple Power** switch is in the off position, no power is being supplied to any maple, sensor, modem, servos, and etc. When **Maple Power** switch is in the on position, all maples and sensor have power and sensor information will be provided to the maples.

NOTE: Space Hut controls the "always off" digital relay to turn on Python (the motherboard). Therefore, power to the maples MUST be supplied to turn on the motherboard. (Refer to Space Hut 5.4.3 Theory of Operation)

The **Motherboard Power** switch provides power to the motherboard through a 2A fuse. When the **Motherboard Power** switch is off, no power to the motherboard is supplied. When the **Motherboard power** switch is on, power is supplied to the motherboard. However, this does NOT turn on the motherboard. The **PC On/Off** button must be pressed to turn on the PC. Currently, there is no software switch to turn the motherboard on. Therefore, if the motherboard ever needs to be reset by cutting the power supply, the **PC On/Off** button must be pressed to turn the motherboard back on.

Electrical/Software task to add software switch: The motherboard has two columns; one column with 4 rows and one column with 5 rows.



The Signal pin is held high (5V). When the PC On/Off button is pressed, the signal pin is grounded. Currently the PC On/Off button is attached to both the Sig and GND pin. To add the ability to turn back on the PC remotely, we need to add a software switch. Pin 37 on Space Hut is both open for use and 5-volt tolerant. Simply splice a wire into the signal pin and solder the other end to Pin 37 on the Space Hut. In the Maple IDE, open the space hut code and add these lines

```
const int         PCswitch = 37;
void setup(){
        pinMode(PCswitch, OUTPUT);
        digitalWrite(PCswitch, HIGH);
}
void loop(){
        // To turn on motherboard (after digital relay supplies power)
        // simulate button press
        digitalWrite(PCswitch, LOW);
        delay(1000);
        digitalWrite(PCswitch, HIGH);

        // To perform a software reset (that is turn off the computer while it is running and
        // then turn it back on)
        // Turn off computer by doing a software FORCE OFF (press and hold button for
        // ~5 seconds)
        digitalWrite(PCswitch, LOW);
        delay(5000);    // may need to be greater than 5 seconds.  Try 6 if 5 does not work
        digitalWrite(PCswitch, HIGH);
        // wait a second or so
        delay(2000);
        // Then turn it back on
        digitalWrite(PCswitch, LOW);
        delay(1000);
        digitalWrite(PCswitch, HIGH);


}
```

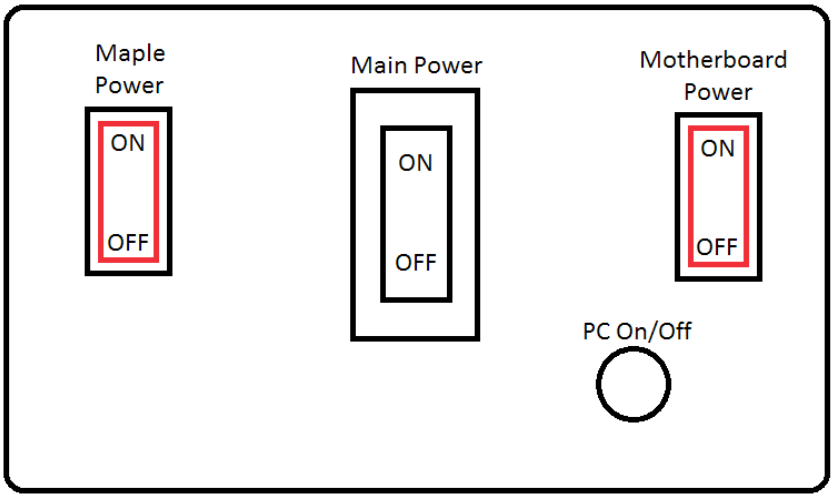


Figure 1: Grover Power Control Diagram

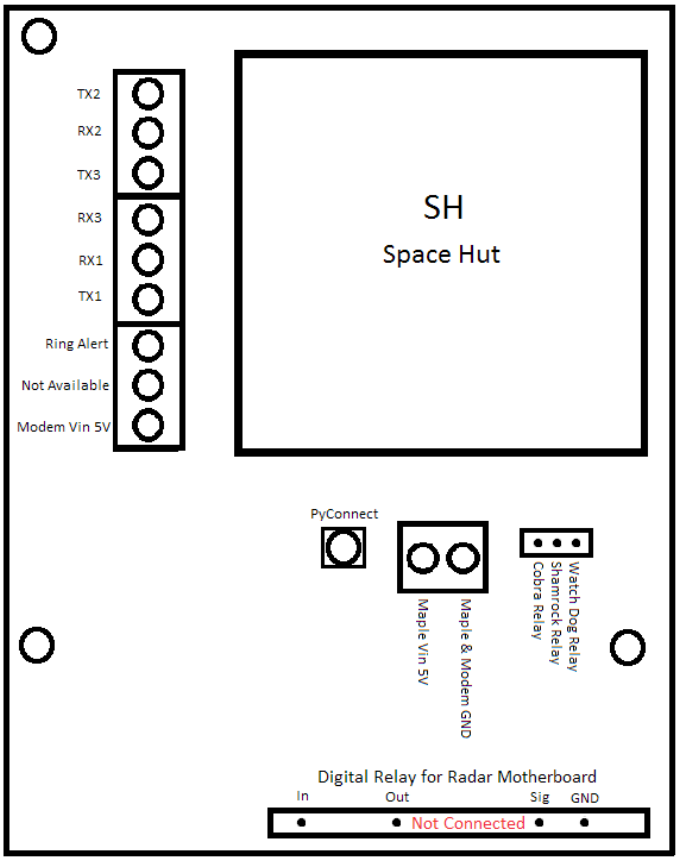


Figure 1: Space Hut Protoboard Diagram

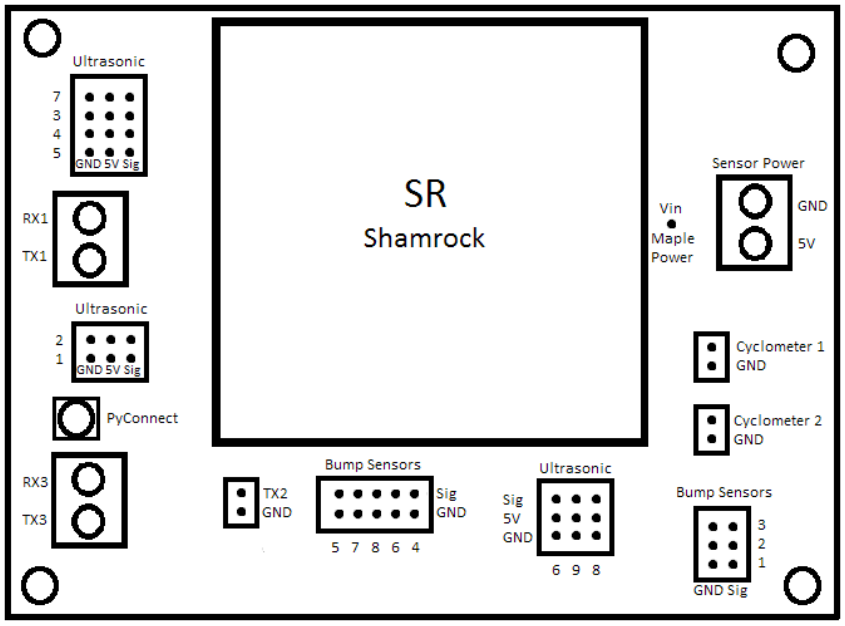


Figure 2: Shamrock Protoboard Diagram

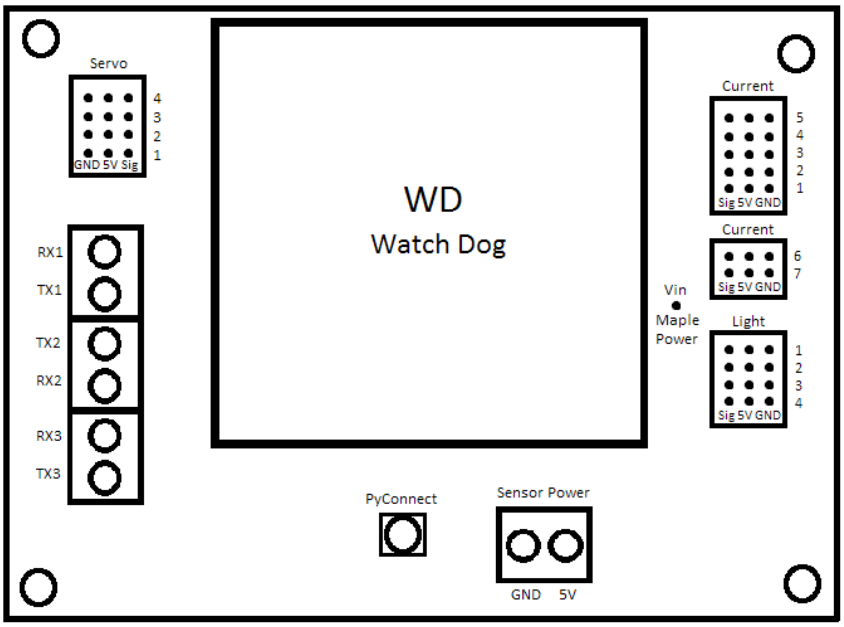


Figure 3: Watch Dog Protoboard Diagram

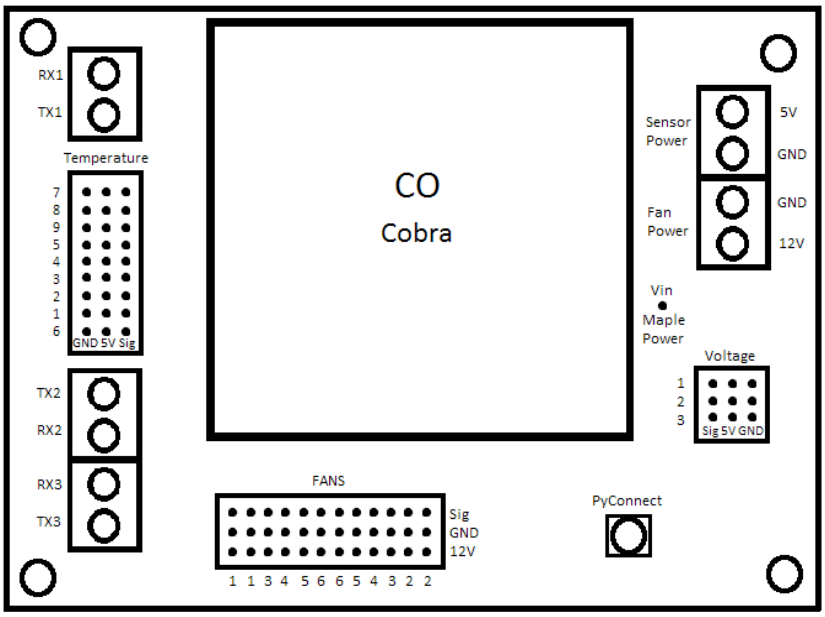


Figure 4: Cobra Protoboard Diagram

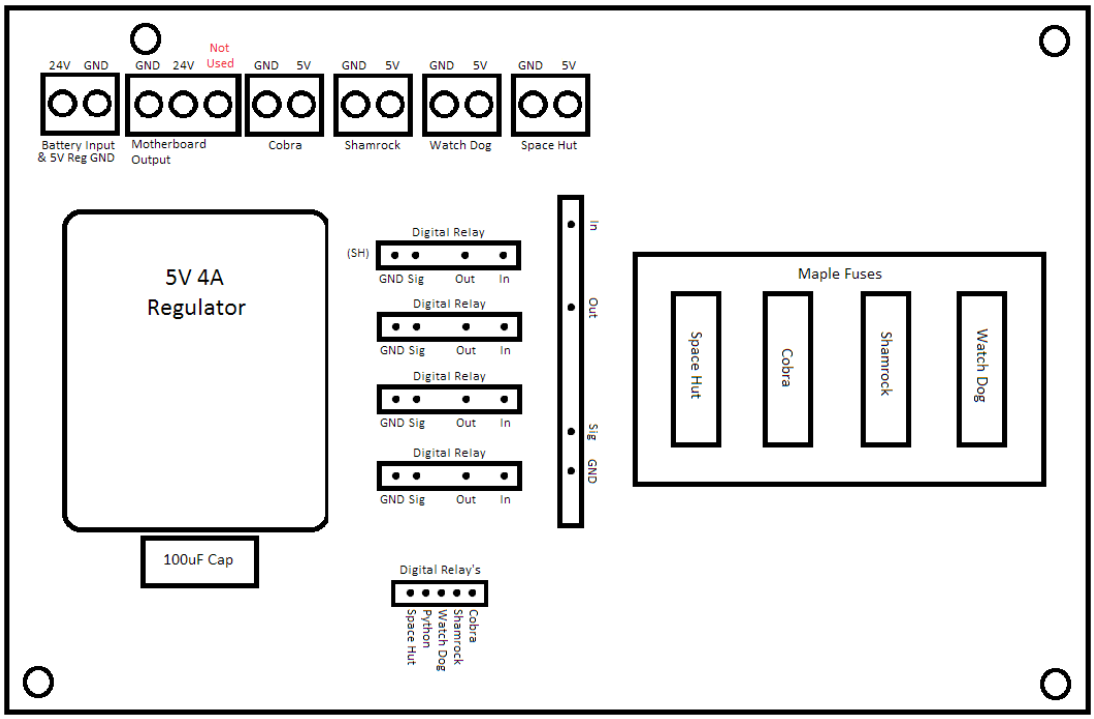
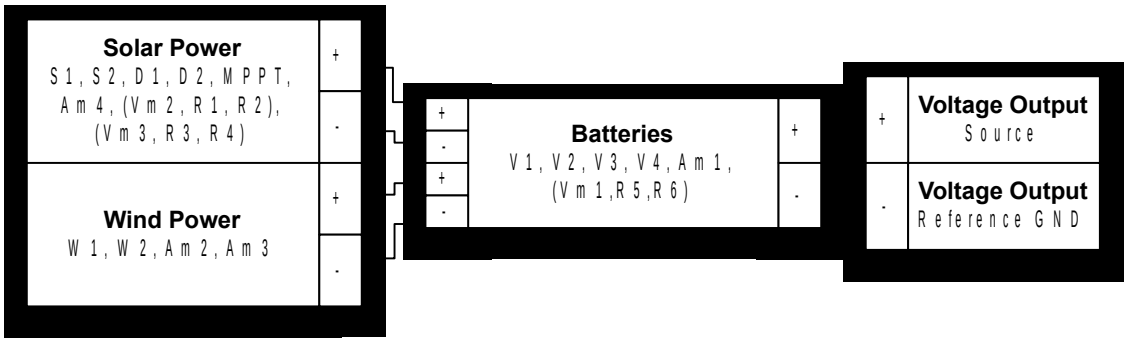


Figure 5: Power & Fuse Protoboard Diagram

5.2 Power Supply

5.2.1 Power Supply Block Diagram



5.2.2 Power Supply Interface Definition

Name	Type	Description
Solar Power:	Power Source	Charges Battery S1, S2: Solar Panels (SUNPOWER SPR-327NE-WHT-D) • Connected to MPPT through D1 & D2 • Provides a maximum output voltage of 54.7 V • Provides a maximum output current of 5.98 A D1, D2: (Phillips E8305 power diodes) • Connected between each solar panel and the MPPT • Provides reverse current protection MPPT: Maximum PowerPoint Tracker • Provide maximum power from solar panels to batteries Am4: 1122 Current Sensor • Monitors output current of MPPT Vm2, Vm3: 1135 Voltage Sensor • Monitors Voltage of S1 and S2
Wind Power	Power Source	Charges Battery W1, W2: Wind Turbines (Air Breeze BR0015454) • Connected directly to batteries • Provides a maximum output voltage of 24 V • Provides a maximum output current of 12.5 A

		Am2, Am3: 1122 Current Sensor • Monitors output current of W1 & W2
Batteries	Power Source	V1, V2, V3, V4: 12V Batteries -- Power Sonic PG-12V65-FR Lead Acid • Supplies 24V DC • V1||V2 in series with V3||V4

5.2.3 Power Supply Theory of Operation

The purpose of the Power Block is to supply power to all the subsystems of GROVER II while maintaining charge on all the batteries. At the center of the Power Block are four 12V, 65 Amp-Hour lead acid batteries. These batteries are configured to provide 24V at 130 Amp-Hours. GROVER II will be expected to operate autonomously for at least three months in Greenland. Connected to the batteries are two SUNPOWER SPR-327NE-WHT-D solar panels and two Air Breeze BR0015454 wind turbines. This configuration will maintain charge on GROVER II during continuous operation.

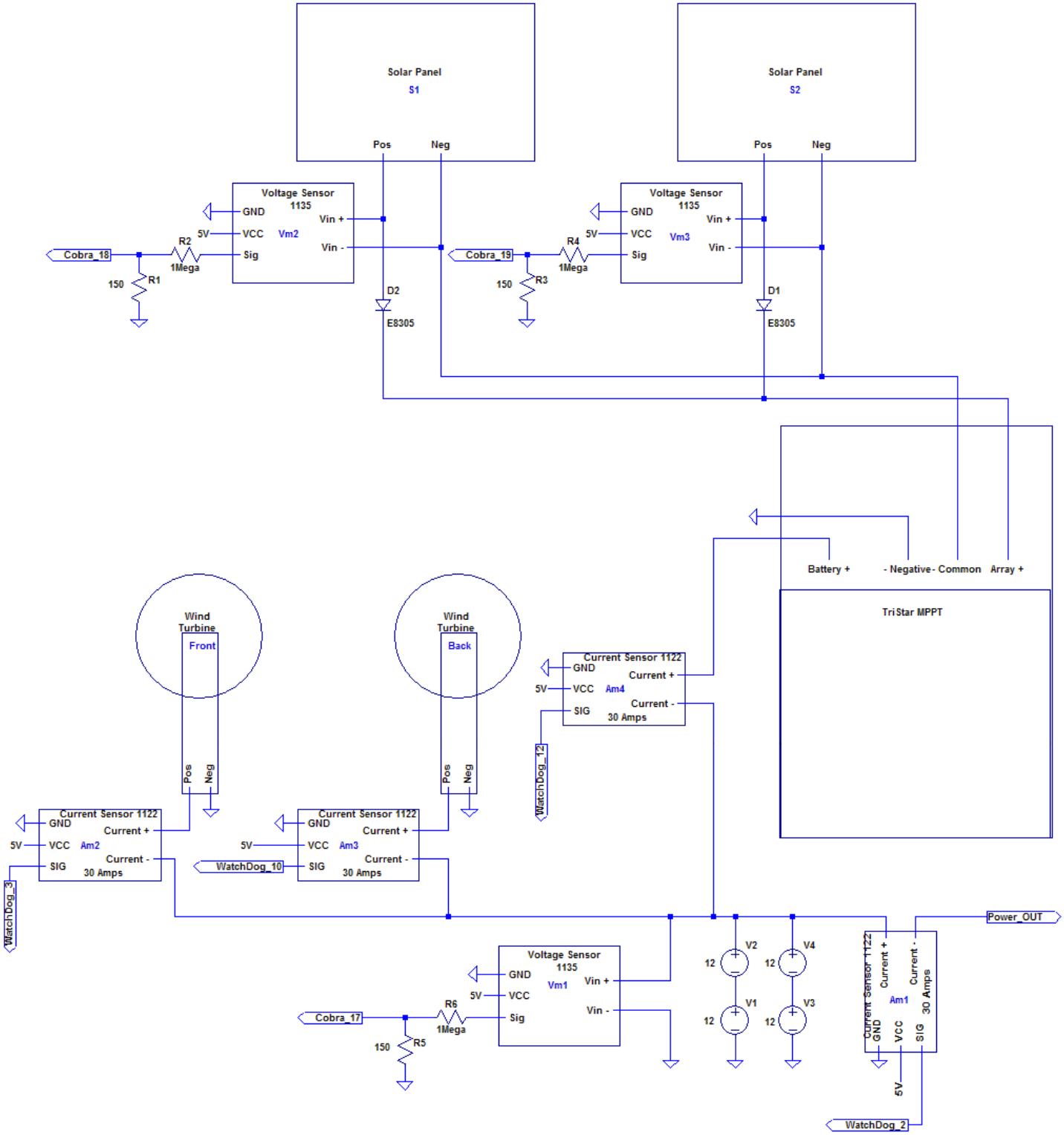
Each of the two SUNPOWER solar panels is connected to the battery bank through a Phillips E8305 power diode to a 45 Amp Maximum PowerPoint Tracker (MPPT) (TrakStar Version TS-MPPT-45). Each solar panel can supply a maximum of 54.7 Volts and 5.98 Amps. This voltage is dependent on the solar input and on the operating load. The diodes prevent current flow back into either solar panel. A panel will have a higher open circuit voltage than it does once it is loaded. If there is enough sunlight on it, that panel will produce enough current to power the load and to set the voltage at the input to the MPPT. Then the other panel must have enough sunlight on it to produce that same voltage at whatever current level it can support. So, the output voltages of each panel may be the same when they are contributing to the load, but the current contributions from each panel will be dependent on the sunlight on each of them. NOTE: if there is not enough sunlight on one panel to reach the loaded voltage at the MPPT input, then it will not produce any current. If you were to read it's open circuit voltage in that case, it could be higher than you'd expect, but you need other information to know how little sunlight is on it.

The take home message here is this. We need another device to know which panel is lit and how they compare relatively. We plan to use the Photo Diode detector array mounted on top of the Array structure to do this. This array has one photo diode pointed in each of the four quadrants: front, back, left, right. It needs to be mounted in a bowl of some sort to keep snow off it and painted to mask out stray light from the sides or ground. If it works, then by monitoring each of the four diodes, we will know how much sunlight is on each array. All we need is a relative voltage level (eg High, Medium, Low), not a precise value. We just need confirmation of the actual direction to the Sun from our vehicle. The masking is intended to ensure that there is a noticeable difference in the photo diode when it is in the sunlight. We can then couple this information with the calculated position of the Sun provided by the GPS (time and location) to verify where

we are actually pointed relative to where we want to be pointed. We would generally point according to the GPS predictions to optimize our solar array outputs, but the photo diode array will tell us what is actually happening, especially when there is a particular point source as opposed to a cloudy day.

The Air Breeze wind turbines supply 24V at 12.5A. They are self-regulated and protected against over-current, extreme weather conditions, and have an automatic brake to prevent damage from excessively high wind speeds. The turbines are directly connected to the batteries and supply power in parallel. The output current of each turbine is measured in order to monitor the operating status of this subsystem. If the buss voltage at the junction of the MPPT output and the two Wind turbines is too high for the level set at each turbine (set screw on the side of the unit), then that unit will not produce any power. It will brake itself and wait for the load to drain down the batteries to a level where that turbine can add in more current to raise the buss voltage. So, you can have a situation where there is plenty of wind, but you are not using it. This is okay, since you don't want to over charge the batteries and destroy them.

5.2.4 Power Supply Schematic (Voltage Sensors for solar panels not installed – Ports are available)



5.2.5 Power Supply Parts List

5.2.6 Power Supply Code

See Appendix A

5.2.7 Power Supply Additional Comments

The current sensors and voltage sensors used output a voltage that correlates with the actual voltage/current being detected. The graphs below show the data collected for these sensors

Current Sensor 1122		**Maple Input**			
Measured Current (A)	**Voltage Out (V)**	**Maple Voltage (V)**	**Digital Value (12-bit ADC)**	**Mapped Value**	**Trend Line (A)**
30	5				
19.8		3.3	4095	1365	-19.8
0	2.5				0
- 30	0	0	0	0	30

Figure 1: Current sensor ADC – Maple conversion information

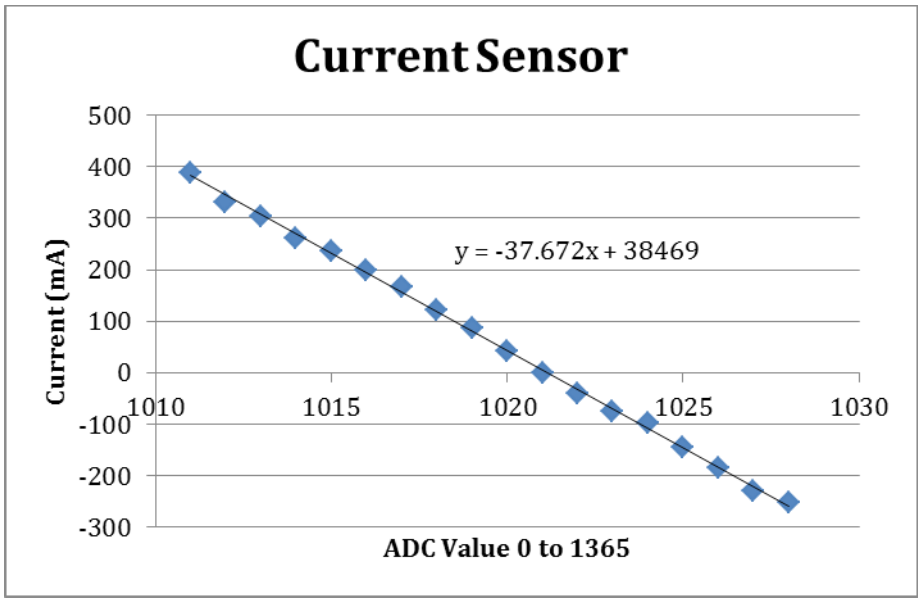


Figure 2: 1122 Current sensor data

Voltage Sensor 1135		Maple Input			
Measured Voltage (V)	**Voltage Out (V)**	**Maple Voltage (V)**	**Digital Value (12-bit ADC)**	**Mapped Value**	**Trend Line (V)**
30	4.5				
22		3.3	4095	1365	-22
0	2.5				0
-30	-0.5	0	0	0	30

Figure 3: Voltage sensor ADC – Maple conversion information

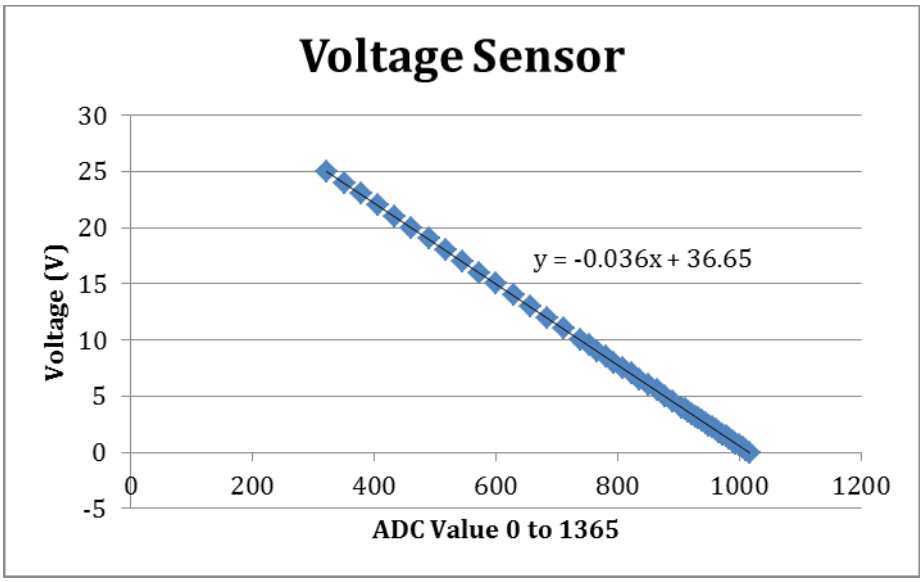
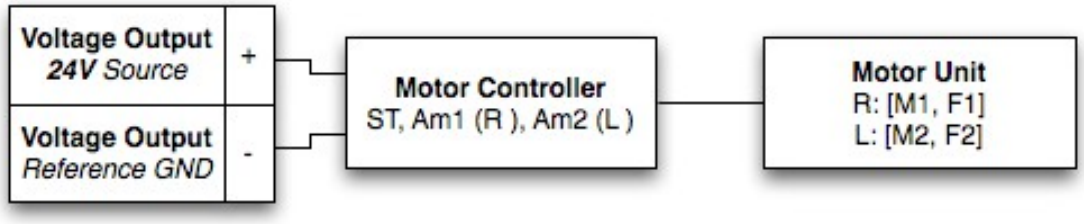


Figure 4: 1135 Voltage sensor data

5.2.8 Power Supply Testing

5.3 Motor Control

5.3.1 Motor Control Block Diagram



5.3.2 Motor Control Interface Definition

Name	Type	Description
Motor Controller:	DC Motor Driver	ST: Sabertooth Motor Controller • Dual 50A motor driver (Dimension Engineering Sabertooth 2x50HV) • Receives Motor commands from Shamrock Microcontroller and drives both left and right motors Am1 (R), Am2 (L): 1122 Current Sensor • Monitors output current of MPPT Am1 (R) & Am2 (L) are 30A current sensors (Phidgets, 1122), which measure the current to each motor
Motor Unit:	Motor	Powered by ST, the motor drive GROVER II M1&M2 are the right and left motors, respectively. (Leeson 1/4 HP 64RPM 12VDC gearmotor M1135245) F1&F2 are 20A fast fuses to prevent damage to gearbox from over-currenting should the software fail

5.3.3 Motor Control Theory of Operation

The purpose of the Motor Block is to control all the physical movements of GROVER II. The Motor Block receives 24V from the Power Block (see 5.2). The right and left motors, M1 and M2 respectively, are powered and controlled by Sabertooth. Sabertooth is a dual 50A motor controller (ST). Due to mechanical issues, the motors should not be given more than 12 volts at 20 amps. [NOTE: this limitation is still under consideration. The technical point of contact is the Vendor, Electric Motor Wholesale.com 302 6531844 .

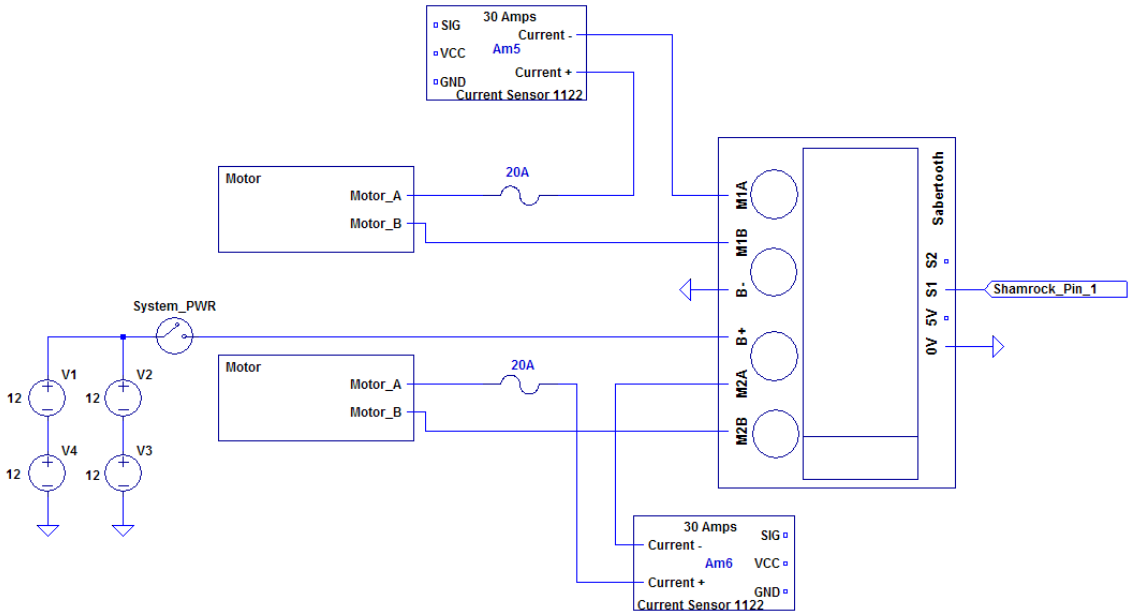
Edward Frye or Greg at:

Electric Motor Wholesale.com
829 Lion Hope Rd.
Clayton, DE 19938
(302) 653-1844
sales@electricmotorwholesale.com

Currently, the motors are current limited at 20A each. We experienced a fault when this was exceeded; which appeared to indicated that the mechanical fuse pin inside both motors broke at once. However the manufacturer recently examined both motors in detail and saw no damage at all. Hence, we need to work with them on this and then we can probably lift that constraint and go faster. To go faster we could just put the smaller gears on the motors in the chain drive assembly. Conversely, we could increase the voltage to the motors by commanding the Sabretooth Controller. This could be done in small steps. However, before doing either of these, be sure to understand what happened at GSFC when the fault occurred.

As such, a software current limitation has been implemented using the information from Am1 and Am2. In addition a 20A fast blow fuse is attached to each motor as a failsafe (F1 & F2) **(F1 and F2 are not yet connected)**. The Sabertooth[1] (ST) is set to "Standard Simplified Serial Mode" with a baud rate of 9600. Integers ranging from 0-255 are sent to the Sabertooth via serial communication (TX & RX).

5.3.4 Motor Control Schematic



1

5.3.5 Motor Control Parts List

5.3.6 Motor Control Code

See Appendix B

5.3.7 Motor Control Additional Comments

Due to mechanical issues, the motors should not be given more than 12 volts at 20 amps. Since the Sabertooth motor controller is capable of providing 24 volts, we never send it a signal would produce more than12 volts. If the motor mechanical concern is resolved then you can go faster by increasing this voltage or by putting on the smaller Gears mentioned above. Try the software mod first and do that in small steps. The motor ranges (character sent across TX line from maple to sabertooth) are shown below.

MOTOR RANGES
- M1 30 - 97
  - M1 FULL REV: 30 (-12V OUTPUT)
  - M1 STOP: 64
  - M1 FULL FWD: 97 (12V OUTPUT)
- M2 157 - 224
  - M2 FULL REV: 157 (-12V OUTPUT)
  - M2 STOP: 192
  - M2 FULL FWD: 224 (12V OUTPUT)

There must be at least a **50 ms delay** between transmitting motor values to the Sabertooth.

5.3.8 Motor Control Testing

5.4 Space Hut

5.4.1 Space Hut Block Diagram

5.4.2 Space Hut Interface Definitions

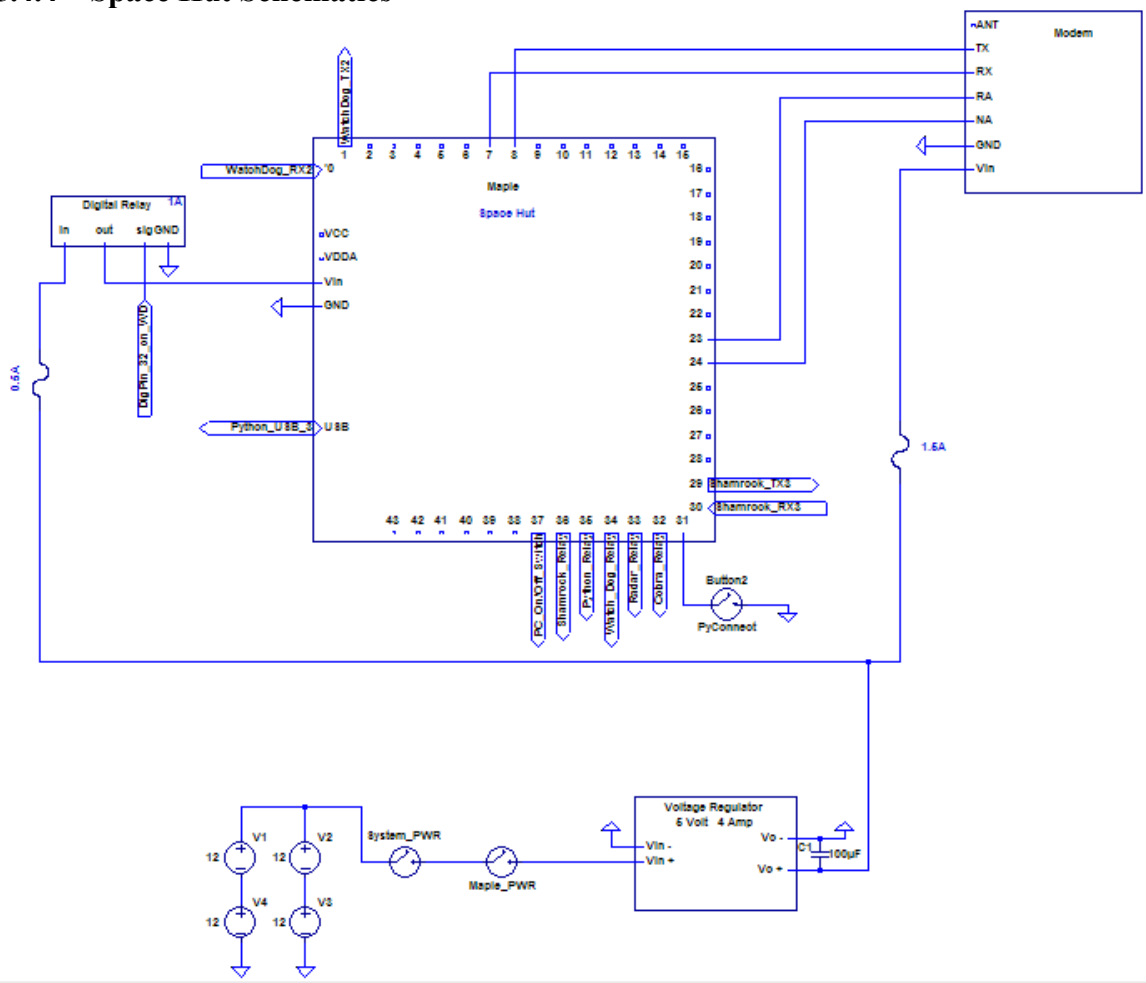
5.4.3 Space Hut Theory of Operation

Space Hut is connected to power through a 5-volt regulator. Between the voltage regulator and Vin on the Maple is a 500mA fuse and a digital relay. Watch Dog controls the digital relay. Watch Dog can reset power to Space Hut by writing LOW to Space Hut's relay. Space Hut is the main microcontroller that must remain on at all times. Nevertheless, if it ever needs to be reset, Watch Dog has the power to do so. During real-world operation, all communication to the Grover takes place through the iridium satellites into Space Hut. Space Hut can then process the command received. Space Hut as well controls the digital relays to all microcontrollers and to the motherboard.

Note: The 1A digital relays attached to all the micro-controllers are always on relays (digital LOW is on, digital HIGH is off). The 5A digital relay attached to the motherboard (Python) is an always off relay (digital LOW is off, and digital HIGH is on). Since Space Hut controls the power to Python then if Space Hut is reset, Python will reset as well. The Setup Code for Space Hut must have digital pin 35 set to HIGH for Python to turn on.

If Python is reset, how will the push-button be pushed to complete the Power up?

5.4.4 Space Hut Schematics



5.4.5 Space Hut Parts List

5.4.6 Space Hut Code

5.4.7 Space Hut Testing

5.4.8 Space Hut Additional Comments

5.5 Shamrock

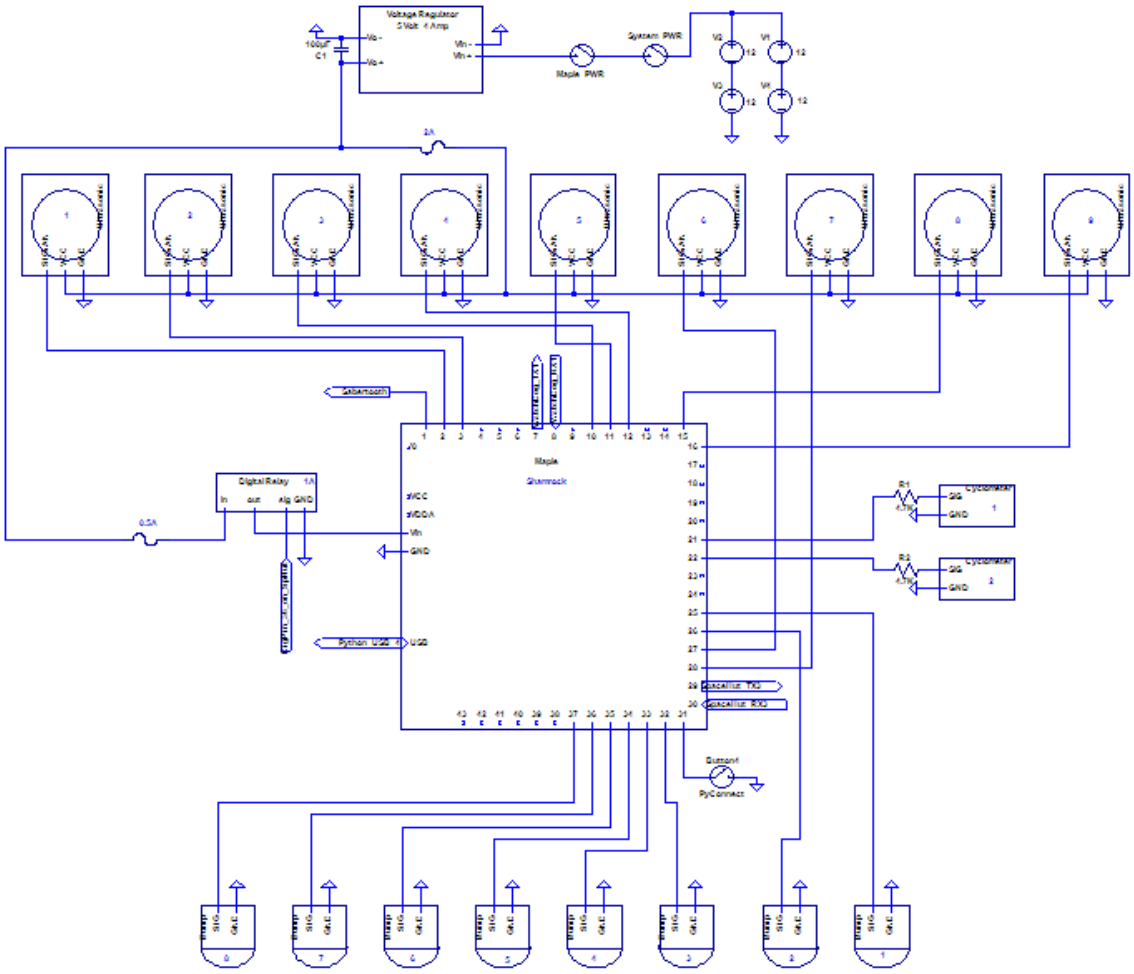
5.5.1 Shamrock Block Diagram

5.5.2 Shamrock Interface Definitions

5.5.3 Shamrock Theory of Operation

Shamrock is connected to power through a 5-volt regulator. Between the voltage regulator and Vin on the Maple is a 500mA fuse and a digital relay. Space Hut controls the digital relay. Space Hut can reset power to Shamrock by writing LOW to Shamrock's relay. The main purpose of Shamrock is to control the motors and implement basic obstacle avoidance. Attached to shamrock are 9 ultrasonic sensors, 8 bump sensors, and two cyclometers. The ultrasonic sensors are capable of detecting objects within a few meters. The bump sensors are setup to be active Low (The pins are set to digital HIGH. An interrupt is attached to detect the falling edge of the signal. When the bumpers are pressed, the signal goes low and triggers the interrupt). The cyclometers act as digital encoders and are used to detect the revolution of one of the wheels attached to the tracks (just like you see on bicycles for odometers). If the Grover goes into an obstacle avoidance mode, the GPS is not able to track minute movements. Therefore, the ability to measure distance travelled by revolutions is important when avoiding obstacles. Shamrock talks to the motor controller "Sabertooth" through a USART port. It transmits one character representing a number 0 – 255 (See Motor Controller Block for more information).

5.5.4 Shamrock Schematics



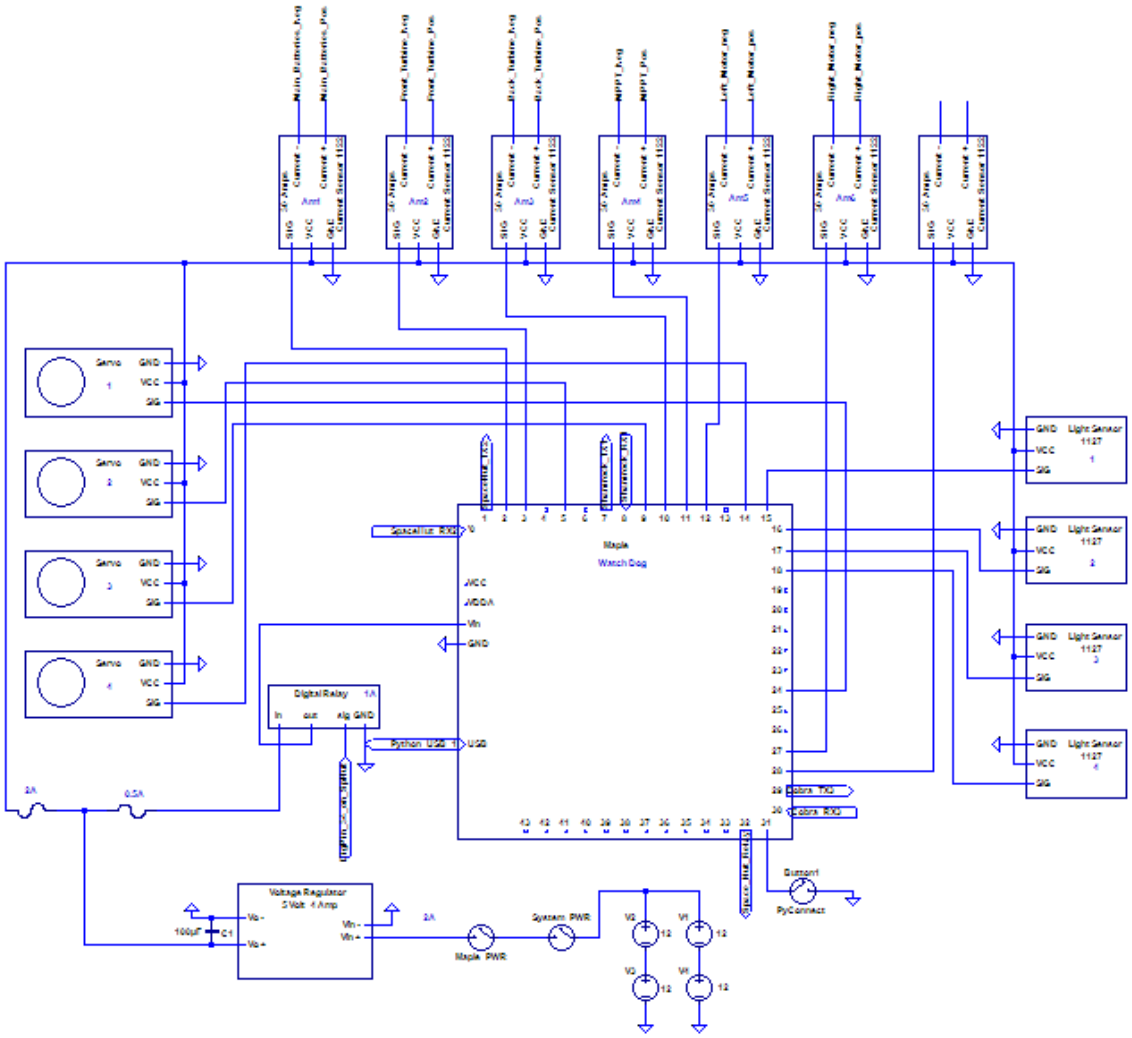
5.5.5 Shamrock Parts List

5.5.6 Shamrock Code
5.5.7 Shamrock Testing
5.5.8 Shamrock Additional Comments

5.6 Watch Dog
5.6.1 Watch Dog Block Diagram
5.6.2 Watch Dog Interface Definitions
5.6.3 Watch Dog Theory of Operation

Watch Dog is connected to power through a 5V regulator. Between the voltage regulator and Vin on the Maple is a 500mA fuse and a digital relay. Space Hut controls the digital relay. Space Hut can reset power to Watch Dog by writing LOW to Watch Dog's relay. Watch Dog has three USART ports that are used for communication to and from Space Hut, Shamrock, and Cobra, as well as one USB port for communication with Python. Connected to Watch Dog are 7 current sensors, 4 light sensors, and 4 servos. Watch Dog is responsible for monitoring all the sensors and controlling the servos. The current sensors monitor the output of the MPPT into the battery, both the wind turbines into the battery, the system input (battery output to the system), and both of the motors. The light sensors (photo diode array atop the solar arrays) act as a low level sun tracking system. They are set up to face 90 degrees adjacent form one another, covering all 360 degrees around the GROVER. The servos are design to open vents allowing airflow into the electrical box and both battery boxes from the outside environment. These vents will be set up to open when the temperature becomes too high. We are not implementing this function. We'll just leave the vents open and let the fans switch on or off.

5.6.4 Watch Dog Schematics



5.6.5 Watch Dog Parts List
5.6.6 Watch Dog Code
5.6.7 Watch Dog Testing
5.6.8 Watch Dog Additional Comments

5.7 Cobra
5.7.1 Cobra Block Diagram
5.7.2 Cobra Interface Definitions
5.7.3 Cobra Theory of Operation

Cobra is connected to power through a 5V regulator. Between the voltage regulator and Vin on the Maple is a 500mA fuse and a digital relay. Space Hut controls the digital relay. Space Hut can reset power to Cobra by writing LOW to Watch Dog's relay. The main objective of Cobra is simply an extension to Watch Dog. Cobra carries the addition sensor required by the Grover; voltage and temperature sensors, and fans. Cobra reads in the values for both the voltage and temperature sensors. Cobra has the ability to turn the fans on or off, according to the temperature as well as relay information about the

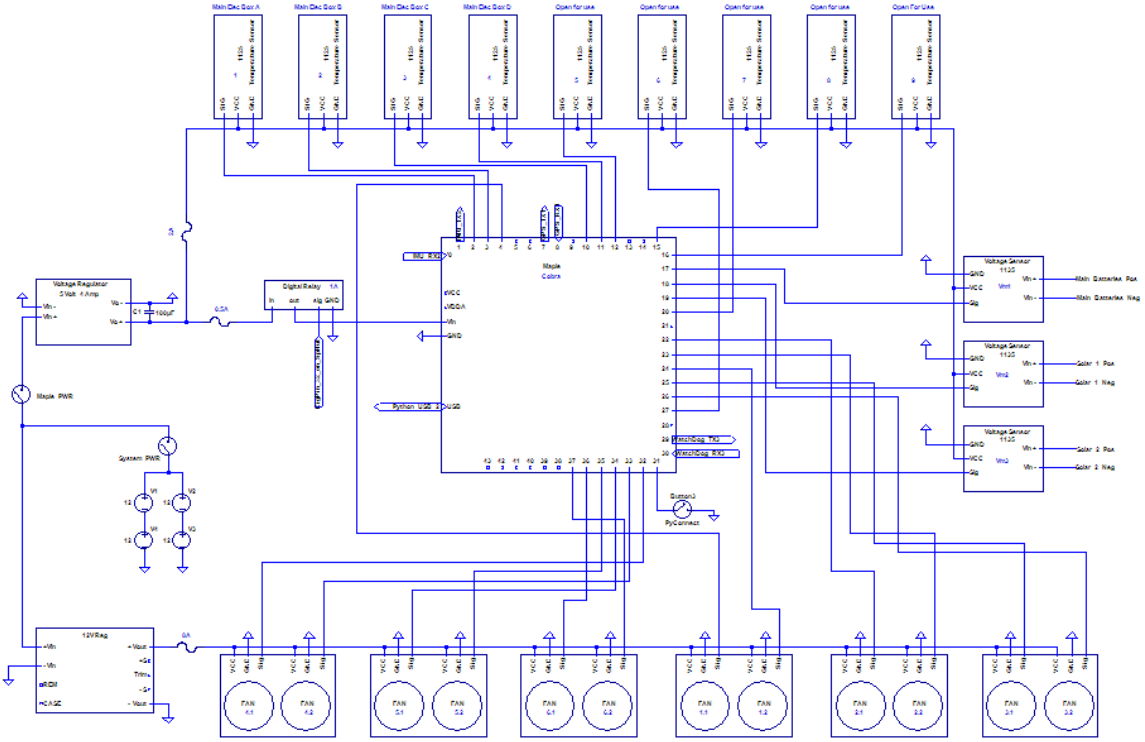
voltage. (aka, if the voltage is to low, Cobra can inform Space Hut and/or Python to begin operating in charging mode).

BATTERY BOX COMMENTS ON THINGS TO DO:

NOTE: The idea of the fans is to use the heat generated inside the Electronics Box to warm the Batteries inside their respective boxes within each track assembly. This would be more efficient than wasting that heat, but it is mainly effective when the GROVER motors are operating and generating most of the heat. Because we also need to keep the AGM (Gel cell type) Lead Acid Batteries near room temperatures for max performance, we need to use temp-controlled patch heaters. These are provided separately so the Idaho team needs to disassemble the track sections to access the batteries and install these heaters. There are four patch heaters. Put one under each battery. There are two 24Vdc Temp switches one for each track section. Put one in the center of the two contiguous batteries and wire one lead to each terminal to provide the +24 Vin to this switch. Figure out a good way to secure the switches. These switches are factory set to turn on between 10C and 15C; and to turn off between 20C and 25C. The other two temp switches are provided as spares.

NOTE: There is an extra connector hole in the back of each of the battery boxes. This was intended for the wires going to the battery temp sensors, which are not installed yet. You can do this, but you can probably use the extra pins on the Battery Power connector and then just cover up the unused hole. We are providing the amphenol connectors so you have either option.

5.7.4 Cobra Schematics



5.7.5 Cobra Parts List

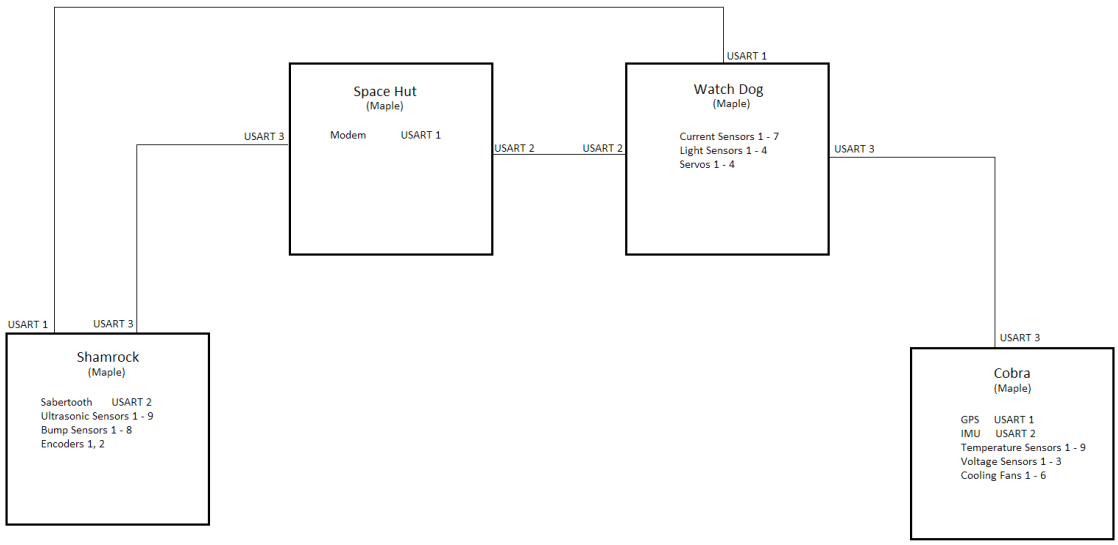
5.7.6 Cobra Code

5.7.7 Cobra Testing

5.7.8 Cobra Additional Comments

5.8 Maple Control

5.8.1 Maple Control Block Diagram



5.8.2 Maple Control Interface Definition

Name	Type	Description
Space Hut:	Microcontroller	Dedicated processor to communicate with modem for iridium satellite
Shamrock:	Microcontroller	Dedicated processor to drive motors, read wheel speed, and optional support for bump and distance sensors

Watch Dog:	Microcontroller	Dedicated processor to monitor current and directional ambient light readings
Cobra:	Microcontroller	Dedicated processor to monitor voltages and compartment temperatures

---

5.8.3 Maple Control Theory of Operation

In the current configuration the Maples do not communicate with eachother, though the physical setup allows for this expansion as a software fix. The idea behind having inter-Maple communication was primarily rooted in the desire to create redundancy in the system. One application of this redundancy was to allow the Maples to operate autonomously while the python computer was down for any number of reasons including low power and damage. In this case the Maples need a way to communicate messages back and forth to each other. We selected which Maples would communicate to eachother based on how we perceived information would need to flow to make decisions. The second application of the inter-Maple connectivity is if python lost communication with a Maple, then it could try to reach it via a second Maple.

None of these redundancy features have been implanted yet. We currently run under direct python control, as will happen under normal operation.

5.8.4 Maple Control Schematics

5.8.5 Maple Control Parts List

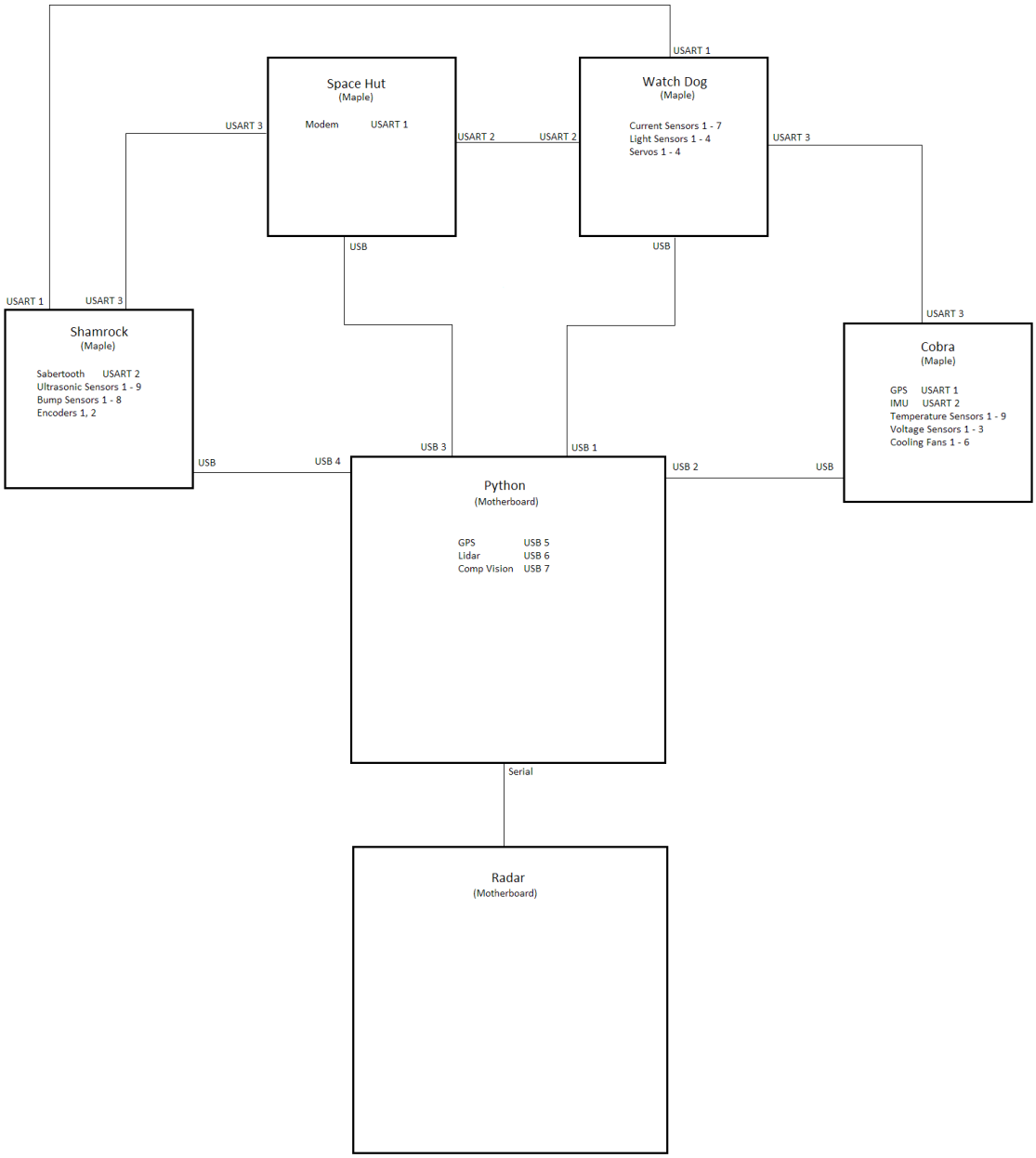
5.8.6 Maple Control Code

5.8.7 Maple Control Testing

5.8.8 Maple Control Additional Comments

5.9 Python Maple Communication

5.9.1 Python Maple Communication Block Diagram



5.9.2 Python Maple Communication Interface Definition

Name	Type	Description
Python	Motherboard	Under normal operation: Python does all the computation and sends READ/WRITE/PING commands to the maples. Other modes of operation have not yet been implemented.
Space Hut:	Microcontroller	Under normal operation: the microcontrollers act as DACs and ADCs for the Python computer simply taking queues from Python. Space Hut is dedicated to communication with the iridium satellite modem. Under non-normal conditions: all microcontrollers will have an elevated level of autonomy, which is not yet defined.
Shamrock:	Microcontroller	Dedicated processor to drive motors, read wheel speed, and optional support for bump and distance sensors
Watch Dog:	Microcontroller	Dedicated processor to monitor current and directional ambient light readings
Cobra:	Microcontroller	Dedicated processor to monitor voltages and compartment temperatures

5.9.3 Python Maple Communication Theory of Operation

The Python motherboard is a more powerful, easier to program development platform than the Maple microprocessors, albeit it draws more power and lacks ADCs and DACs. Under normal operating conditions, excepting, for example, low batteries, the Python computer does all the decision making, effectively treating the Maples as ADCs and DACs. The Maples simply listen for simple commands from Python and carry them out. Sometimes Python's request requires a little processing on the Maple side, for

instance when reading certain sensors, but for the most part computation on the Maples is avoided.

The communication protocol is described in detail in the dropbox code/Committed Maple Code/Mini\ Documentation.txt and is copied below for reference:

“ This project contains a unified software package to be deployed on every Maple Microcontroller on the Grover. Every Maple shares the same code with the exception of the .h files and config files differentiated by the fact that the files contain the names of the Maples they pertain to.

The only thing that the end user needs to change between maples is to uncomment the #define below pertaining to the name of the Maple they want to upload to. Only one #define should be uncommented at a time.

Code Structure:
General Compile Setup Page: "Comprehensive_GROVERII_Code"
In this file we select which files will be compiled and uploaded to the Maples. I.e., in this file you choose which Maple you have connected to the computer that you want to load code to by uncommenting the name of the Maple you are using.
For Example: if we were upload ing to Watchdog:

```
#define WATCHDOG
//#define COBRA
//#define SPACEHUT
//#define SHAMROCK
```

Pin Configuration Page: "XXX_Config" (where XXX is the name of the Maple)
In this page, for each maple, we:
1. define the pin numbers that the different sensors are attached to, and other variables pertaining globally to that Maple
2. call the setup loop which defines the pins as analog/digital input/outputs, opens the Serial ports, and defines two data structures to be used in resolving the python computers requests to the Maples

Function Definition Page : "XXX_Functions.h"
In this page we define functions pertaining to the sensors attached to the Maple XXX

Main loop page: "X_Loop"
In this page we define the main loop of the code which listens for a communication from the Python computer. The messages from the Python computer come in several flavors.
Note that words in "" are string literals, i.e., they are the exact letters sent. meanwhile words not enclosed in quotations are variables and have values that are transmitted instead of the variable name.
To write to a digital/analog output:

```
SENT:       "WRITE " channel_number " " value_to_write Command_ID \n
RETURNED:   "OK " Command_ID
```

To Read from a digital/analog input:

SENT: "READ " channel_number " " Command_ID \n
RETURNED: "SENSOR " sensorvalue " " Command_ID

To ping the Maples for their identity:

SENT: "PING " Command_ID \n
RETURNED: "PONG " Maple_ID " " Command_ID

To debug using fake data:

SENT: "FAKE " channel_number " " value_to_write Command_ID \n
RETURNED: "FAKE " Command_ID

To resolve what pins are referred to we use the "channel_to_pin_number[channel_number]", and to resolve whether to apply digital or analog methods to that pin we use the "channel_pin_type[channel_number]" data structure where:

pin type 1 = Analog I/O
pin type 2 = Digital I/O
pin type 3 = Serial line

In the X_Loop file we also implement a general analog read function called "Read(int channel)" because each sensor has a slightly different way it needs to be read and interpreted"

5.9.4 Python Maple Communication Schematic

5.9.5 Python Maple Communication Parts List

5.9.6 Python Maple Communication Code

5.9.7 Python Maple Communication Testing

5.9.8 Python Maple Communication Additional Comments

6. Python (Motherboard) and Maple Pin Assignments

6.1 Python (Motherboard)

1. Maple – Watch Dog — USB (Dynamically Assigned)
2. Maple – Cobra — USB (Dynamically Assigned)
3. Maple – Space Hut — USB (Dynamically Assigned)
4. Maple – Shamrock — USB (Dynamically Assigned)
5. Motherboard – Radar — Serial (Port _____)
6. GPS — USB (Dynamically Assigned)

NOTE: Schematics are the most up to date. If there is a conflict between the schematics and pin assignments below, the schematics are correct. Please update this sheet if any discrepancies are found.

6.2 Shamrock (Maple)

1. **Maple – Python** — USB (Dynamically Assigned)
2. **Maple – Watch Dog** — USART_1 (Pin 7&8)
3. **Maple – Space Hut** — USART_3 (Pin 29&30)
4. **Sabertooth** — USART_2 (Pin 0&1)
5. **Ultrasonic sensors 1** — ADC0 (Pin 2)
    a. Left-side sensor
6. **Ultrasonic sensors 2** — ADC1 (Pin 3)
    a. Left front corner sensor
7. **Ultrasonic sensors 3** — ADC4 (Pin 10)
    a. Left middle sensor
8. **Ultrasonic sensors 4** — ADC6 (Pin 12)
    a. Center sensor
9. **Ultrasonic sensors 5** — ADC7 (Pin 11)
    a. Right middle sensor
10. **Ultrasonic sensors 6** — ADC8 (Pin 27)
    a. Right front corner sensor
11. **Ultrasonic sensors 7** — ADC9 (Pin 28)
    a. Right-side sensor
12. **Ultrasonic sensors 8** — ADC10 (Pin 15)
    a. Rear sensor
13. **Ultrasonic sensors 9** — ADC11 (Pin 16)
    a. Extra analog saved for addition sensor
14. **Bump Sensors 1** — Digital (Pin 25)
    a. Front left corner sensor
15. **Bump Sensors 2** — Digital (Pin 26)
    a. Front left middle sensor
16. **Bump Sensors 3** — Digital (Pin 32)
    a. Front center sensor
17. **Bump Sensors 4** — Digital (Pin 33)
    a. Front right middle sensor
18. **Bump Sensors 5** — Digital (Pin 34)

    a. Front right corner sensor
19. **Bump Sensors 6** Digital (Pin 35)
    a. Back left corner sensor
20. **Bump Sensors 7** Digital (Pin 36)
    a. Back middle sensor
21. **Bump Sensors 8** Digital (Pin 37)
    a. Back right corner sensor
22. **Cyclometer 1** Digital (Pin 21)
    a. Left track
23. **Cyclometer 2** Digital (Pin 22)
    a. Right track

6.3 Watch Dog (Maple)

1. **Maple – Python** USB (Dynamically Assigned)
2. **Maple – Shamrock** USART_1 (Pin 7&8)
3. **Maple – Space Hut** USART_2 (Pin 0&1)
4. **Maple – Cobra** USART_3 (Pin 29&30)

5. **Phidget – Current Sensor 1** ADC0 (Pin 2)
    a. Left solar panel
6. **Phidget – Current Sensor 2** ADC1 (Pin 3)
    a. Right solar panel
7. **Phidget – Current Sensor 3** ADC4 (Pin 10)
    a. Front Turbine
8. **Phidget – Current Sensor 4** ADC6 (Pin 12)
    a. Back Turbine
9. **Phidget – Current Sensor 5** ADC7 (Pin 11)
    a. Left motor
10. **Phidget – Current Sensor 6** ADC8 (Pin 27)
    a. Right motor
11. **Phidget – Current Sensor 7** ADC9 (Pin 28)
    a. System input (one for both batteries)
12. **Phidget – Light Sensor 1** ADC10 (Pin 15)
    a. North sun tracking
13. **Phidget – Light Sensor 2** ADC11 (Pin 16)
    a. East sun tracking
14. **Phidget – Light Sensor 3** ADC12 (Pin 17)
    a. South sun tracking
15. **Phidget – Light Sensor 4** ADC13 (Pin 18)
    a. West sun tracking
16. **Servo 1** PWM5 (Pin 5)
    a. Left battery box
17. **Servo 2** PWM6 (Pin 6)
    a. Main electrical box – slot 1

18. **Servo 3** PWM9 (Pin 9)
    a. Main electrical box – slot 2
19. **Servo 4** PWM14 (Pin 14)
    a. Right battery box

6.4 Cobra (Maple)

1. **Maple – Python** USB (Dynamically Assigned)
2. **Maple – Watch Dog** USART_3 (Pin 29&30)
3. **GPS** USART_1 (Pin 7&8)
4. **Phidget – Temperature Sensor 1** ADC0 (Pin 2)
    a. Main electrical box A
5. **Phidget – Temperature Sensor 2** ADC1 (Pin 3)
    a. Main electrical box B
6. **Phidget – Temperature Sensor 3** ADC4 (Pin 10)
    a. Main electrical box C
7. **Phidget – Temperature Sensor 4** ADC6 (Pin 12)
    a. Main electronics box D
8. **Phidget – Temperature Sensor 5** ADC7 (Pin 11)
    a. Open for use
9. **Phidget – Temperature Sensor 6** ADC8 (Pin 27)
    a. Open for use
10. **Phidget – Temperature Sensor 7** ADC15 (Pin 20)
    a. Open for use
11. **Phidget – Temperature Sensor 8** ADC10 (Pin 15)
    a. Open for use
12. **Phidget – Temperature Sensor 9** ADC11 (Pin 16)
    a. Open for use
13. **Phidget – Voltage Sensor 1** ADC12 (Pin 17)
    a. System input (both batteries)
14. **Phidget – Voltage Sensor 2** ADC13 (Pin 18)
    a. Left Solar Panel
15. **Phidget – Voltage Sensor 3** ADC14 (Pin 19)
    a. Right Solar Panel
16. **Cooling Fan 1** PWM5 (Pin 5)
    a. Left battery box 1
17. **Cooling Fan 2** PWM6 (Pin 6)
    a. Left battery box 2
18. **Cooling Fan 3** PWM9 (Pin 9)
    a. Left battery box 3
19. **Cooling Fan 4** PWM14 (Pin 14)
    a. Right battery box 1
20. **Cooling Fan 5** PWM24 (Pin 24)
    a. Right battery box 2
21. **Cooling Fan 6** PWM28 (Pin 28)
    a. Right battery box 3

6.5 Space Hut (Maple)

1. **Maple – Python**	USB (Dynamically Assigned)
2. **Maple – Shamrock**	USART_3 (Pin 29&30)
3. **Maple – Watch Dog**	USART_2 (Pin 0&1)
4. **Modem/antenna**	USART_1 (Pin 7&8)
Modem/antenna	Digital (Pin 23)
Modem/antenna	Digital (Pin 25)

7. Maple Pin Layout

7.1 Space Hut Pin Layout

Pins	PWM
0	
1	
2	
3	
5	
7	
8	
9	
11	
12	
14	
24	
27	
28	

Pins	Analog
0	
1	
(ADC0) 2	
(ADC1) 3	
(ADC4) 10	
(ADC7) 11	
(ADC6) 12	
(ADC10) 15	
(ADC11) 16	
(ADC12) 17	
(ADC13) 18	
(ADC14) 19	
(ADC15) 20	
(ADC8) 27	
(ADC9) 28	

Pins	Digital
0	RX2 (WD)
1	TX2 (WD)
2	
3	
4	
5	
6	
7	TX1 (Modem)
8	RX1 (Modem)
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	Modem
24	Modem
25	
26	
27	
28	
29	TX3 (SR)
30	RX3 (SR)
31	PyComm Int
32	CO Relay
33	
34	WD Relay
35	
36	SR Relay
37	
	Maple button
39	
40	
41	
42	
43	

7.2 Shamrock Pin Layout

Pins	PWM
0	
1	
2	
3	
5	
7	
8	
9	
11	
12	
14	
24	
27	
28	

Pins	Analog
0	
1	
(ADC0) 2	
(ADC1) 3	
(ADC4) 10	
(ADC7) 11	
(ADC6) 12	
(ADC10) 15	
(ADC11) 16	
(ADC12) 17	
(ADC13) 18	
(ADC14) 19	
(ADC15) 20	
(ADC8) 27	
(ADC9) 28	

Pins	Digital
0	RX2 (WD)
1	TX2 (WD)
2	
3	
4	
5	
6	
7	TX1 (Modem)
8	RX1 (Modem)
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	Modem
24	Modem
25	
26	
27	
28	
29	TX3 (SR)
30	RX3 (SR)
31	PyComm Int
32	CO Relay
33	Radar Relay
34	WD Relay
35	PY Relay
36	SR Relay
37	PY On/Off
	Maple button
39	
40	
41	
42	
43	

7.3 Watch Dog Pin Layout

Pins	PWM
0	
1	
2	
3	
5	Servo2
7	
8	
9	Servo3
11	
12	
14	Servo4
24	Servo1
27	
28	

Pins	Analog
0	
1	
(ADC0) 2	CurrSens1
(ADC1) 3	CurrSens2
(ADC4) 10	CurrSens3
(ADC7) 11	CurrSens4
(ADC6) 12	CurrSens5
(ADC10) 15	LightSens1
(ADC11) 16	LightSens2
(ADC12) 17	LightSens3
(ADC13) 18	LightSens4
(ADC14) 19	
(ADC15)20	
(ADC8) 27	CurrSens6
(ADC9) 28	CurrSens7

Pins	Digital
0	RX2 (SH)
1	TX2 (SH)
2	
3	
4	
5	
6	
7	TX1 (SR)
8	RX1 (SR)
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	TX3 (CO)
30	RX3 (CO)
31	PyComm Int
32	
33	
34	
35	
36	
37	
	Maple Button
39	
40	
41	
42	
43	

7.4 Cobra Pin Layout

Pins	PWM
0	
1	
2	
3	
5	
7	
8	
9	
11	
12	
14	
24	
27	
28	

Pins	Analog
0	
1	
(ADC0) 2	temp1
(ADC1) 3	temp2
(ADC4) 10	temp3
(ADC7) 11	temp4
(ADC6) 12	temp5
(ADC10) 15	temp8
(ADC11) 16	temp9
(ADC12) 17	volt1
(ADC13) 18	volt2
(ADC14) 19	volt3
(ADC15) 20	temp7
(ADC8) 27	temp6
(ADC9) 28	

Pins	Digital
0	RX2 (IMU)
1	TX2 (IMU)
2	
3	
4	Fan 1.1
5	
6	
7	TX1 (GPS)
8	RX1 (GPS)
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	Fan 2.1
23	Fan 2.2
24	Fan 1.2
25	Fan 3.1
26	Fan 3.2
27	
28	
29	TX3 (WD)
30	RX3 (WD)
31	PyComm Int
32	Fan 4.1
33	Fan 4.2
34	Fan 5.1
35	Fan 5.2
36	Fan 6.1
37	Fan 6.2
	Maple Button
39	
40	
41	
42	
43	

8. Overall Electrical Parts List

Reference	Qty.	Common Name	Manufacturer Number	Supplier	Supplier Number	Unit Price (US dollars)	E (

Appendix A

Power Supply Block Test Code: The power supply code requires two Maple microcontroller – Cobra and Watch Dog. Cobra reads from the voltage sensors and Watch Dog reads from the current sensors.

```
/*
   POWER SUPPLY BLOCK TEST CODE

   Microcontroller: Cobra
*/

// voltage sensor to maple pin
const int VoltSense1 = 17;
const int VoltSense2 = 18;
const int VoltSense3 = 19;

void setup(){
  // Set voltage sensor pins as input
  pinMode(VoltSense1, INPUT);
  pinMode(VoltSense2, INPUT);
  pinMode(VoltSense3, INPUT);
}

void loop(){

   // Temp values hold the sum of 50 separate reading.
   // The average is then taken from these readings
   int temp1 = 0;
   int temp2 = 0;
   int temp3 = 0;

   // Take 50 readings from each sensor and sum the temp values
   int count = 0;
   int readings = 50;
   while (count < readings){
      int volt1 = analogRead(VoltSense1);  // Read pin
      // For greater stability -- Map 4095 (12-bit ADC) digital values down to 1365 digital values
      int volt1_Temp = map(volt1, 0, 4095, 0, 1365);

      int volt2 = analogRead(VoltSense2);
      int volt2_Temp = map(volt2, 0, 4095, 0, 1365);

      int volt3 = analogRead(VoltSense3);
      int volt3_Temp = map(volt3, 0, 4095, 0, 1365);

      // Sum previous readings with new readings
      temp1 += volt1_Temp;
      temp2 += volt2_Temp;
      temp3 += volt3_Temp;

      // increment count
      count++;
      delay(1);
   }


    //  Calculate the average value of all the readings
    float RawVoltVal1 = temp1 / readings;
    float RawVoltVal2 = temp2 / readings;
    float RawVoltVal3 = temp3 / readings;


    // Equation created from collecting data and creating a trend line in MS excel
```

```
    // Voltage sensors have a linear plot
    float   v1 = (RawVoltVal1 * -36 + 36650);
    float   v2 = (RawVoltVal2 * -36 + 36650);
    float   v3 = (RawVoltVal3 * -36 + 36650);

    // Print Values to USB connectio
    SerialUSB.print("Voltage Value 1:   ");
    SerialUSB.print(v1);
    SerialUSB.println(" mV");

    SerialUSB.print("Voltage Value 2:   ");
    SerialUSB.print(v2);
    SerialUSB.println(" mV");

    SerialUSB.print("Voltage Value 3:   ");
    SerialUSB.print(v3);
    SerialUSB.println(" mV");

    // Delay 1 second so program does not freeze ;)
    delay(1000);

}

/*
    POWER SUPPLY BLOCK TEST CODE

    Microcontroller: WatchDog
*/

// Setup current sensor pin connections
const int CurrSens1 = 2;
const int CurrSens2 = 3;
const int CurrSens3 = 10;
const int CurrSens4 = 11;
const int CurrSens5 = 12;
const int CurrSens6 = 27;
const int CurrSens7 = 28;

int CurrVal1, CurrVal2, CurrVal3, CurrVal4, CurrVal5, CurrVal6, CurrVal7;
float c1, c2, c3, c4, c5, c6, c7;

/*********************************************************/

void setup(){
  // Set current sensor pins as input
  pinMode(CurrSens1, INPUT);
  pinMode(CurrSens2, INPUT);
  pinMode(CurrSens3, INPUT);
  pinMode(CurrSens4, INPUT);
  pinMode(CurrSens5, INPUT);
  pinMode(CurrSens6, INPUT);
  pinMode(CurrSens7, INPUT);

}

void loop(){

    // Temp values hold the sum of 50 separate reading.
    // The average is then taken from these readings
    int temp1 = 0;
    int temp2 = 0;
    int temp3 = 0;
    int temp4 = 0;
    int temp5 = 0;
    int temp6 = 0;
    int temp7 = 0;
```

```
// Take 50 readings from each sensor and sum the temp values
int count = 0;
int readings = 50;
while (count < readings){
   int Cv1 = analogRead(CurrSens1);  // Read pin
   // For greater stability -- Map 4095 (12-bit ADC) digital values down to 1365 digital values
   int Cv1Temp = map(Cv1, 0, 4095, 0, 1365);

   int Cv2 = analogRead(CurrSens2);
   int Cv2Temp = map(Cv2, 0, 4095, 0, 1365);

   int Cv3 = analogRead(CurrSens3);
   int Cv3Temp = map(Cv3, 0, 4095, 0, 1365);

   int Cv4 = analogRead(CurrSens4);
   int Cv4Temp = map(Cv4, 0, 4095, 0, 1365);

   int Cv5 = analogRead(CurrSens5);
   int Cv5Temp = map(Cv5, 0, 4095, 0, 1365);

   int Cv6 = analogRead(CurrSens6);
   int Cv6Temp = map(Cv6, 0, 4095, 0, 1365);

   int Cv7 = analogRead(CurrSens7);
   int Cv7Temp = map(Cv7, 0, 4095, 0, 1365);


   // Sum previous readings with new readings
   temp1 += Cv1Temp;
   temp2 += Cv2Temp;
   temp3 += Cv3Temp;
   temp4 += Cv4Temp;
   temp5 += Cv5Temp;
   temp6 += Cv6Temp;
   temp7 += Cv7Temp;

   // increment count
   count++;
   delay(1);
}


//  Calculate the average value of all the readings
CurrVal1 = temp1 / readings;
CurrVal2 = temp2 / readings;
CurrVal3 = temp3 / readings;
CurrVal4 = temp4 / readings;
CurrVal5 = temp5 / readings;
CurrVal6 = temp6 / readings;
CurrVal7 = temp7 / readings;

// Equation created from collecting data and creating a trend line in MS excel
// Current sensors have a linear plot
c1 = (CurrVal1 * -37.672 + 38469);
c2 = (CurrVal2 * -37.672 + 38469);
c3 = (CurrVal3 * -37.672 + 38469);
c4 = (CurrVal4 * -37.672 + 38469);
c5 = (CurrVal5 * -37.672 + 38469);
c6 = (CurrVal6 * -37.672 + 38469);
c7 = (CurrVal7 * -37.672 + 38469);

// Print Values to USB connection
SerialUSB.print("Current Value 1:   ");
SerialUSB.print(c1);
SerialUSB.println(" mA");

SerialUSB.print("Current Value 2:   ");
SerialUSB.print(c2);
SerialUSB.println(" mA");
```

```
  SerialUSB.print("Current Value 3:   ");
  SerialUSB.print(c3);
  SerialUSB.println(" mA");

  SerialUSB.print("Current Value 4:   ");
  SerialUSB.print(c4);
  SerialUSB.println(" mA");

  SerialUSB.print("Current Value 5:   ");
  SerialUSB.print(c5);
  SerialUSB.println(" mA");

  SerialUSB.print("Current Value 6:   ");
  SerialUSB.print(c6);
  SerialUSB.println(" mA");

  SerialUSB.print("Current Value 7:   ");
  SerialUSB.print(c7);
  SerialUSB.println(" mA");

  // Delay 1 second so program does not freeze ;)
  delay(1000);

}
```

Appendix B

Motor Control Block Test Code: The motor control code requires two Maple microcontrollers – Shamrock and Watch Dog. Shamrock controls the motors and Watch Dog reads from the current sensors.

```
/*
  MOTOR CONTROL BLOCK TEST CODE

  Microcontroller: Shamrock
*/


// functional keys for "keyPressed"
// w -- Forward
// s -- Backwards
// a -- Left
// d -- Right
// spacebar -- All stop (kill switch)
// b -- both stop
char keyPressed;

void setup() {
  // Start USART 2
  Serial2.begin(9600);
  Serial2.flush();
  Serial2.print(char(0));   // Force the tracks to stop

  // Start USART 1
  Serial1.begin(9600);
  Serial1.flush();
}

void loop() {
```

```
// MOTOR RANGES
// M1 30 - 97
   // M1 FULL REV: 30   (-12V OUTPUT)
   // M1 STOP: 64
   // M1 FULL FWD: 97   (12V OUTPUT)
// M2 157 - 224
   // M2 FULL REV: 157  (-12V OUTPUT)
   // M2 STOP: 192
   // M2 FULL FWD: 224  (12V OUTPUT)


   //Function will wait until button is pressed on keyboard
   keyPressed = SerialUSB.read();

   if ( keyPressed == 'w') {
     // Both Forward
     // M1
     Serial2.print(char(97));  // Absolute Max Value for M1 FORWARD
     delay(100);
     // M2
     Serial2.print(char(224)); // Absolute Max Value for M2 == FORWARD
     SerialUSB.println("forward");
   }
   else if(keyPressed == 's'){
     // Both Reverse
     // M1
     Serial2.print(char(30));  // Absolute Min Value for M1 == REV
     delay(100);
     // M2
     Serial2.print(char(157)); // Absolute Min Value for M2 == REV
     SerialUSB.println("Reverse");
   }
   else if(keyPressed =='d'){
     // Turn Right
     // M1
     Serial2.print(char(30));  // Absolute Min Value for M1 == REV
     delay(100);
     // M2
     Serial2.print(char(224)); // Absolute Max Value for M2 == REV
     SerialUSB.println("Right");
   }
   else if(keyPressed =='a'){
     // Turn Left
     // M1
     Serial2.print(char(97));  // Absolute Max Value for M1 == REV
     delay(100);
     // M2
     Serial2.print(char(157)); // Absolute Min Value for M2 == REV
     SerialUSB.println("Left");
   }
   else if(keyPressed =='b'){
     // Both Stop
     // M1
     Serial2.print(char(64));  // Absolute Max Value for M1 == REV
     delay(100);
     // M2
     Serial2.print(char(192)); // Absolute Max Value for M2 == REV
     SerialUSB.println("both stop");
   }
  else if(keyPressed ==' '){
     // All STOP
     Serial2.print(char(0));  // all stop
     SerialUSB.println("all STOP");
   }
  else{
    SerialUSB.println("Wrong key pressed -- Do nothing");
    delay(500);
   }
```

```
    delay(1000);  // delay to allow time for motors to ramp up and pull current


    for (int i = 1; i <= 2; i++){
      // Request value of a current sensor (connected to Motor)
      // Use Current Sensor pins 1 & 2
      Serial1.flush();
      Serial1.print(char(i));  //Current Sensor

      // Wait for reply from shamrock
      while (!Serial1.available()) {  }

      // Read in both bytes of current sensors value
      int MSB = Serial1.read();
      delay(100);
      int LSB = Serial1.read();

      /*
      (Note: Bit-shifts do not change variable being shifted)
      */

      // Add bytes together
      int currentReading = (MSB << 8) + LSB;

      // Store sign byte (Note: Bit-shifts do not change variable being shifted)
      int sign = (currentReading >> 15);

      // Handles negative values
      if (sign == 1){
          // Note: int is a 32 bit number
          int x = (~currentReading) << 16;  //negate and drop excess 1's
          int y = (x >> 16) + 1;  // shift back into place and add 1 (Two's complement)
          currentReading = y * -1;
      }


       //Print current reading to screen
       SerialUSB.print("Motor Current ");
       SerialUSB.print(i);
       SerialUSB.print(": ");
       SerialUSB.print(currentReading);
       SerialUSB.println(" mA");
    }

}



/*
   MOTOR CONTROL BLOCK TEST CODE

   Microcontroller: Watch Dog
*/

// Setup current sensor pin connections
const int CurrSens1 = 2;
const int CurrSens2 = 3;
const int CurrSens3 = 10;
const int CurrSens4 = 11;
const int CurrSens5 = 12;
const int CurrSens6 = 27;
const int CurrSens7 = 28;

void setup(){
  // Set current sensor pins as input
```

```
  pinMode(CurrSens1, INPUT);
  pinMode(CurrSens2, INPUT);
  pinMode(CurrSens3, INPUT);
  pinMode(CurrSens4, INPUT);
  pinMode(CurrSens5, INPUT);
  pinMode(CurrSens6, INPUT);
  pinMode(CurrSens7, INPUT);

  // Start USART 1
  Serial1.begin(9600);
  Serial1.flush();

}

void loop(){

    // Enters loop when something is ready to transmit
    if (Serial1.available()){
        int sensorNumber = Serial1.read();  // read which current sensor to check
        getCurrent(sensorNumber);
    }

}

void getCurrent(int sensorNum){
    int cSensor;
    // assign sensor number to correct pin
    if (sensorNum == 1){
        cSensor = CurrSens1;
    }
    else if (sensorNum == 2){
        cSensor = CurrSens2;
    }
    else if (sensorNum == 3){
        cSensor = CurrSens3;
    }
    else if (sensorNum == 4){
        cSensor = CurrSens4;
    }
    else if (sensorNum == 5){
        cSensor = CurrSens5;
    }
    else if (sensorNum == 6){
        cSensor = CurrSens6;
    }
    else if (sensorNum == 7){
        cSensor = CurrSens7;
    }


    // Temp values hold the sum of 50 separate reading.
    // The average is then taken from these readings
    int temp = 0;

    // Take 50 readings from each sensor and sum the temp values
    int count = 0;
    int readings = 50;
    while (count < readings){
        int cValue = analogRead(cSensor);  // Read pin

        // For greater stability -- Map 4095 (12-bit ADC) digital values down to 1365 digital values
        int cValueMapped = map(cValue, 0, 4095, 0, 1365);

        // Sum previous readings with new readings
        temp += cValueMapped;

        count++;  // increment count
        delay(1);
    }
```

```
  //  Calculate the average value of all the readings
  float RawCurrVal = temp / readings;

  // Equation created from collecting data and creating a trend line in MS excel
  // Current sensors have a linear plot
  int currentValue = (RawCurrVal * -37.672 + 38469);

  // convert currentValue into binary to be transmitted
  // Only one byte at a time can be transmitted
  int LSB_8_bits = 0b0000000011111111 & currentValue;
  int MSB_8_bits = (0b1111111100000000 & currentValue) >> 8;

  // Print to screen the decimal version of what is being sent
  SerialUSB.print("Sending to Shamrock (in binary):  ");
  SerialUSB.println(currentValue);

  // transmit both bytes
  Serial1.flush();
  Serial1.print(char(MSB_8_bits));
  Serial1.print(char(LSB_8_bits));

}
```

Appendix C

Watch Dog Block Test Code: The Watch Dog block code requires one Maple microcontroller – Watch Dog. Watch Dog reads from the current sensors, light sensors and controls the servos.

```
/*
   WATCH DOG BLOCK TEST CODE

   Microcontroller: Watch Dog
*/


#include "usb.h" // gives us access to libmaple usb functions

// Setup current sensor pin connections
const int CurrSens1 = 2;
const int CurrSens2 = 3;
const int CurrSens3 = 10;
const int CurrSens4 = 11;
const int CurrSens5 = 12;
const int CurrSens6 = 27;
const int CurrSens7 = 28;

// Setup light sensor pin connections
const int lightSens1 = 15;
const int lightSens2 = 16;
const int lightSens3 = 17;
const int lightSens4 = 18;

// Setup servo pin connections
const int servo1 = 24;
const int servo2 = 5;
const int servo3 = 9;
const int servo4 = 14;

HardwareTimer timer(1);
```

```
void setup(){
  // Set current sensor pins as input
  pinMode(CurrSens1, INPUT);
  pinMode(CurrSens2, INPUT);
  pinMode(CurrSens3, INPUT);
  pinMode(CurrSens4, INPUT);
  pinMode(CurrSens5, INPUT);
  pinMode(CurrSens6, INPUT);
  pinMode(CurrSens7, INPUT);

  // Set light sensor pins as input
  pinMode(lightSens1, INPUT);
  pinMode(lightSens2, INPUT);
  pinMode(lightSens3, INPUT);
  pinMode(lightSens4, INPUT);

  // Set servo pins as output
  pinMode(servo1, PWM);
  pinMode(servo2, PWM);
  pinMode(servo3, PWM);
  pinMode(servo4, PWM);

}

char buf[65];
void loop(){


    int newbytes = 0;
    // Enters loop when something is ready to transmit
    if (newbytes = SerialUSB.available()){

        usbReceiveBytes((uint8*)buf, newbytes);
        buf[newbytes] = 0; // set the last char to 0 to create a null terminated string
        //SerialUSB.print(buf);

        char object = buf[0];  //object: l-light, c-current, s-servo
        char number = buf[1];  //number: which sensor number you want


        if (object == 'l'){
            getLight(number);
        }
        else if (object == 'c'){
            getCurrent(number);
        }
        else if (object == 's'){
            setServo(number);
        }
        else{
            SerialUSB.println("Watch Dog does not have that object");
        }
    }



}

void getCurrent(char sensorNum){
    int cSensor;
    boolean run = true;

    // assign sensor number to correct pin
    if (sensorNum == '1'){
        cSensor = CurrSens1;
    }
    else if (sensorNum == '2'){
        cSensor = CurrSens2;
    }
    else if (sensorNum == '3'){
```

```
    cSensor = CurrSens3;
  }
  else if (sensorNum == '4'){
    cSensor = CurrSens4;
  }
  else if (sensorNum == '5'){
    cSensor = CurrSens5;
  }
  else if (sensorNum == '6'){
    cSensor = CurrSens6;
  }
  else if (sensorNum == '7'){
    cSensor = CurrSens7;
  }
  else{
    run = false;
    SerialUSB.print("No current sensor at pin ");
    SerialUSB.println(sensorNum);
  }

  if (run == true){

    // Temp values hold the sum of 50 separate reading.
    // The average is then taken from these readings
    int temp = 0;

    // Take 50 readings from each sensor and sum the temp values
    int count = 0;
    int readings = 50;
    while (count < readings){
      int cValue = analogRead(cSensor);  // Read pin

      // For greater stability -- Map 4095 (12-bit ADC) digital values down to 1365 digital values
      int cValueMapped = map(cValue, 0, 4095, 0, 1365);

      // Sum previous readings with new readings
      temp += cValueMapped;

      count++;  // increment count
      delay(1);
    }


    //  Calculate the average value of all the readings
    float RawCurrVal = temp / readings;

    // Equation created from collecting data and creating a trend line in MS excel
    // Current sensors have a linear plot
    int currentValue = (RawCurrVal * -37.672 + 38469);

    // Print to screen the valu
    SerialUSB.print("Current Sensor ");
    SerialUSB.print(sensorNum);
    SerialUSB.print(":  ");
    SerialUSB.println(currentValue);
  }
}


void getLight(char sensorNum){
  int liSensor;
  boolean run = true;

  // assign sensor number to correct pin
  if (sensorNum == '1'){
    liSensor = lightSens1;
  }
  else if (sensorNum == '2'){
    liSensor = lightSens2;
  }
```

```
    else if (sensorNum == '3'){
      liSensor = lightSens3;
    }
    else if (sensorNum == '4'){
      liSensor = lightSens4;
    }
    else{
      run = false;
      SerialUSB.print("No light sensor at pin ");
      SerialUSB.println(sensorNum);
    }

    if (run == true){

      // Temp values hold the sum of 50 separate reading.
      // The average is then taken from these readings
      int temp = 0;

      // Take 50 readings from each sensor and sum the temp values
      int count = 0;
      int readings = 50;
      while (count < readings){
        int liValue = analogRead(liSensor);  // Read pin

        // For greater stability -- Map 4095 (12-bit ADC) digital values down to 1365 digital values
        int liValueMapped = map(liValue, 0, 4095, 0, 1365);

        // Sum previous readings with new readings
        temp += liValueMapped;

        count++;  // increment count
        delay(1);
      }


      //  Calculate the average value of all the readings
      float RawLightValue = temp / readings;

      // Equation created from collecting data and creating a trend line in MS excel
      // Current sensors have a linear plot
     // int lightValue = (RawLightValue * -37.672 + 38469);

      // Print to screen the valu
      SerialUSB.print("Light Sensor ");
      SerialUSB.print(sensorNum);
      SerialUSB.print(":  ");
      SerialUSB.println(RawLightValue);
    }
}

void setServo(char servoNum){
    int serSensor;
    boolean run = true;

    // assign sensor number to correct pin
    if (servoNum == '1'){
      serSensor = servo1;
    }
    else if (servoNum == '2'){
      serSensor = servo2;
    }
    else if (servoNum == '3'){
      serSensor = servo3;
    }
    else if (servoNum == '4'){
      serSensor = servo4;
    }
    else{
      run = false;
      SerialUSB.print("No servo attached to pin ");
```

```
      SerialUSB.println(servoNum);
    }

    if (run == true){
      SerialUSB.print("Test Servo: ");
      SerialUSB.println(servoNum);

      timer.pause();   // Pause timer to configure
      timer.setPrescaleFactor(21);  // Prescaler set for servo operation

      // Maple measures time in micro-seconds
      // This sets the period of the servo to repeat every 20 milli-seconds
      timer.setPeriod(20 * 1000);

      timer.refresh();  // Refresh the timer's count, prescale, and overflow
      timer.resume();  // Start the timer counting


      // pwmWrite values go from 0 to 65535, need to scale properly
      float dutyCycle = 0.20;    // Turn all the way left
      pwmWrite(serSensor, dutyCycle * 65536);
      delay(1000);

      dutyCycle = 0.40;         // Center
      pwmWrite(serSensor, dutyCycle * 65536);
      delay(1000);

      dutyCycle = 0.60;         // Turn all the way right
      pwmWrite(serSensor, dutyCycle * 65536);
      delay(1000);

      SerialUSB.print("Test Complete on Servo: ");
      SerialUSB.println(servoNum);

    }
}
```