EE/CSE 474 Lab Report
Project 2

Radleigh Ang

Abdul Katani

Daniel Snitkovskiy

TABLE OF CONTENTS

ABSTRACT

The objective of this lab was to design and develop the overall control flow of a Satellite Management and Control System, which will be used to communicate with both the Earth and a distant asteroid to control an extraterrestrial mining unit in search of rare metals. The major components of this project included the development of the drivers for mining, the basic satellite operation, as well as the displaying and annunciation of status, warning, and alarm.

To meet these objectives, we started by building diagrams to get a better understanding of the expected output and flow of control. After this, we started to develop our task queue, populating it with the 5 main functions driving the system:

1. **Warning alarm:** Signals low/critical battery/fuel level by flashing LEDs.
2. **Console display:** Prints satellite status and alarm information.
3. **Satellite comms:** Manages communication with the Earth.
4. **Thruster subsystem:** Interprets commands from Earth and controls the thrusters.
5. **Power subsystem:** Manages the battery and solar panels.

After outlining the 5 functions and the task queue, the scheduler was drafted in order to manage the sequence and frequency of execution of the various tasks. Functions were set to run according to a “major” and a “minor” cycle. Every task will execute on the “major” cycle (every 5 minutes), while only the Warning alarm and Satellite comms will execute on the “minor” cycle (every 10 milliseconds).

Through this lab, we have gained a deeper understanding of the C language (e.g. pointers, structs, etc), used formal design tools to model system behavior, and used task queues and scheduling to construct a real-time embedded systems application.

1. INTRODUCTION

In this lab, our main goal was to get started on our main project of creating a Satellite Management and Control System for managing communication with a vehicle mining asteroids for rare metals. To this end, we designed and developed the overall control flow within the system using the rapid prototyping design methodology.

Purpose:
Design and develop a Satellite Management and Control System by breaking it into five separate functions:

1. warningAlarm()
2. consoleDisplay()
3. satlliteComms()
4. thrustersSubsystem()
5. powerSubsystem()

Methods/Tools:

Methods:
- Paired programming.
- Splitting the work across different files

Tools:
- AM3358BZCZ100 BeagleBone Black
- Ubuntu 16.04.2 LTS Work Station
- GDB 7.4.1 Debian debugger
- GCC 4.6.3 Debian compiler
- Scope Probe

2. DESIGN SPECIFICATIONS

This section shows the design specifications for Project 2.

2.1 System Requirements

2.1.1 General Summary of *Satellite Management and Control System*

- Main purpose of *Satellite Management and Control System*:
    - Control surface-based mining equipment from an orbiting command center and communicate with various Earth stations.
- Specific functions include:
    - Displaying status, warning, and alarm information on the Satellite console (modeled with a terminal display in the Linux environment).
    - Sending pertinent mission information and receiving commands from Earth station.
    - Managing the batteries, fuel, and solar panels.
    - *For an external view of system functionality, see the "Use Case Diagram" (Figure 1) in the General UML Diagram Section.*

2.1.2 User Interface of the Satellite Terminal

- Inputs: Thruster Command from the Earth Station
- Outputs: Satellite Status which is printed on the satellite terminal and communicated to the earth station.
- Side Effects: During the mining process, the system consumes power (in turn decreasing the battery level), generates power from the solar panel, and uses fuel to control the thruster.

2.1.3 Subsystems Overview

The purpose of the system is to display alarm information, satellite status, and support basic command/control signaling. The following is a general overview for the subsystems *(See the "Functional Decomposition" (Figure 2) in the General UML Diagram section for a high level summary)*:

- Power Management Control
    - Tracks these variables:
        - Power Consumption of Satellite
        - Battery Level
    - If Battery Level is low, deploy solar panels, else, retract solar panels.
- Thruster Management and Control
    - Control duration and magnitude of thruster burn to move in desired direction.
    - Tracks direction:

        - Left
        - Right
        - Up
        - Down
    - Tracks Thruster Control:
        - Thrust (Full OFF or Full ON)
    - Thrust Duration
- Communications Requirements
    - Supports communication to earth through...
        - Status
        - Annunciation
        - Warning information
    - Currently, incoming commands are limited to thruster control.
- Status and Annunciation Subsystem Requirements
    - Displays the information in the form of status, annunciation, and alarm through the console and the lights on the front panel.
    - Tracks these variables for Status:
        - Solar Panel State (retracted or deployed)
        - Battery Level
        - Power Consumption and Generation
        - Fuel Level
    - Tracks these variables for Warning and Alarm:
        - Fuel Low
        - Battery Low
    - Warning Alarm controls the function of the LEDs based on the battery level and fuel level. The LEDs will flash at a rate corresponding to the amount of battery (LED2) and fuel (LED1) left in the system:
        - Flashes every two seconds if between 10% and 50%
        - Flashes every one second if less than 10%
        - If both battery and fuel are > 50%, turn on LED3.

2.1.3 Scheduler for these Tasks

The schedule task manages the execution order and period of the tasks in the system.
The scheduler executes tasks in a round robin fashion, utilizing a major cycle and a series of minor cycles. The period of the major cycle is 5 seconds, while the duration of the minor cycle is 10 milliseconds.

Following each major cycle through the task queue, the scheduler will cause the suspension of all task activity, except for the operation of the warning and error annunciation, for five seconds. In between major cycles, there shall be a number of minor cycles to support functionality that must execute on a shorter period. (i.e. the Warning Alarm and Satellite Comms). *(For a graphical view of the scheduling algorithm, see the "State Chart Diagram" (Figure 3) in the General UML Diagram Section).*

2.2 Specific Details and Requirements

2.2.1 Tasks and Task Control Blocks

The Task Control Block (TCB) is implemented as a C struct; utilizing a separate struct for each task. Each TCB has two members:

A. The first member is a pointer to a function taking a void* argument and returning a void.
B. The second member is a pointer to void used to reference the shared data for the task.

The TCB member, taskDataPtr, will reference a struct containing references to all data utilized by task. Each data struct will contain pointers to data required/modified by the target task.

2.2.2 Initial State for Shared Variables

The following variables are defined to hold the status and alarm information and command and control information. *(For a graphical view of relationship between the shared variables and the various TCB functions, see the "Shared Variables vs. TCB Methods Diagram" (Figure 4))*

Thruster Control

Global - Type unsigned int

Thruster Command	initial value	0

Local - Type unsigned short

Left	initial value	0
Right	initial value	0
Up	initial value	0

Down initial value 0

Power Management

Global - Type unsigned short
- Battery Level initial value 100
- Fuel Level initial value 100
- Power Consumption initial value 0
- Power Generation initial value 0

Solar Panel Control

Global - Type Bool
- Solar Panel State initial value FALSE

Local - Type unsigned short
- Motor Drive initial value 0
- Range
  - Full OFF
  - Full ON

Status Management and Annunciation

Global - Type Bool
- Solar Panel State initial value FALSE

Global - Type unsigned short
- Battery Level initial value 100
- Fuel Level initial value 100
- Power Consumption initial value 0
- Power Generation initial value 0

warningAlarm

Global - Type Bool
- Fuel Low initial value FALSE
- Battery Low initial value FALSE

2.2.3 Requirements for PowerSubsystem

The following operations are to be performed on each of the data variables referenced in powerSubsystemData (the struct held by the taskDataPtr).

powerConsumption:

Increment the variable by 2 every even numbered time the function is called and decrement by 1 every odd numbered time the function is called until the value of the variable exceeds 10. The number 0 is considered to be even. Thereafter, reverse the process until the value of the variable falls below 5. Then, once again reverse the process.

powerGeneration:

If the solar panel is deployed
- If the battery level is greater than 95%
  - Issue the command to retract the solar panel
- Else
  - Increment the variable by 2 every even numbered time the function is called and by 1 every odd numbered time the function is called until the value of the battery level exceeds 50%. Thereafter, only increment the variable by 2 every even numbered time the function is called until the value of the battery level exceeds 95%.

If the solar panel is not deployed
- If the battery level is less than or equal to 10%
  - Issue the command to deploy the solar panel
- Else
  - Do nothing.

batteryLevel

If the solar panel is not deployed
- Each time the function is called, compute the current battery level as follows,
  - batteryLevel = batteryLevel - 3*powerConsumption

If the solar panel is deployed
- Each time the function is called, compute the current battery level as follows,
  - batteryLevel = batteryLevel – powerConsumption + powerGeneration

2.2.4 Requirements for ThrusterSubsystem

The thrusterSubsystem task will interpret each of the fields within the thruster command and generate a control signal of the specified magnitude and duration to the designated thruster. The thruster control signals, for a specified duration, have magnitudes of 0%, 50%, and 100% of full scale. If fuel is expended at a continuous 5% rate, the satellite will have a mission life of 6 months. The thrusterSubsystem will update the state of the fuel level based upon the use of the thrusters.

2.2.5 Requirements for SatelliteComs

Data transfer from the satellite to the earth shall be the following status information:

- Fuel Low
- Battery Low
- Solar Panel State
- Battery Level
- Fuel Level
- Power Consumption
- Power Generation

The thrust command (a randomly generated 16-bit number) shall be interpreted as follows:

Thruster ON	Bits 3-0
Left	0
Right	1
Up	2
Down	3
Magnitude	Bits 7-3
OFF	0000
Max	1111
Duration - sec	Bits 15-8
0	0000
255	1111 1111

2.2.6 Requirements for ConsoleDisplay

The *ConsoleDisplay* task will support two modes: *Satellite Status* and *Annunciation*.
In the *Satellite Status* mode, the following will be displayed for the satellite

- Solar Panel State
- Battery Level
- Fuel Level
- Power Consumption

In the *Annunciation* mode, the following will be displayed

- Fuel Low
- Battery Low

2.2.7 Requirements for WarningAlarm

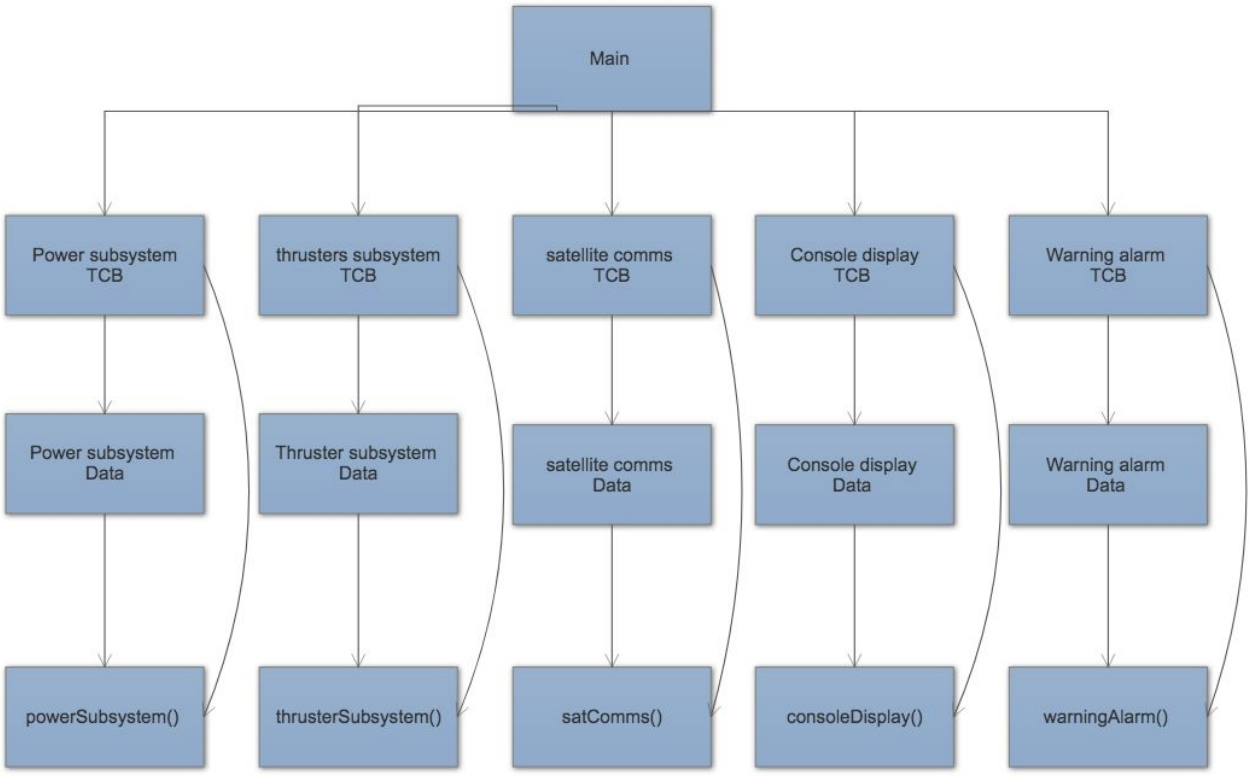
The *warningAlarm* task interrogates the state of the battery and fuel level to determine if they have reached a critical level.

- If both are within range, the LED3 on the annunciation panel shall be illuminated and on solid.
- If the state of the battery level reaches 50%, LED2 on the annunciation panel shall flash at a 2 second rate.
- If the state of the fuel level reaches 50%, LED1 on the annunciation panel shall flash at a 2 second rate.
- If the state of the battery level reaches 10%, LED2 on the annunciation panel shall flash at a 1 second rate.
- If the state of the fuel level reaches 10%, LED1 on the annunciation panel shall flash at a 1 second rate.

3. SOFTWARE IMPLEMENTATION

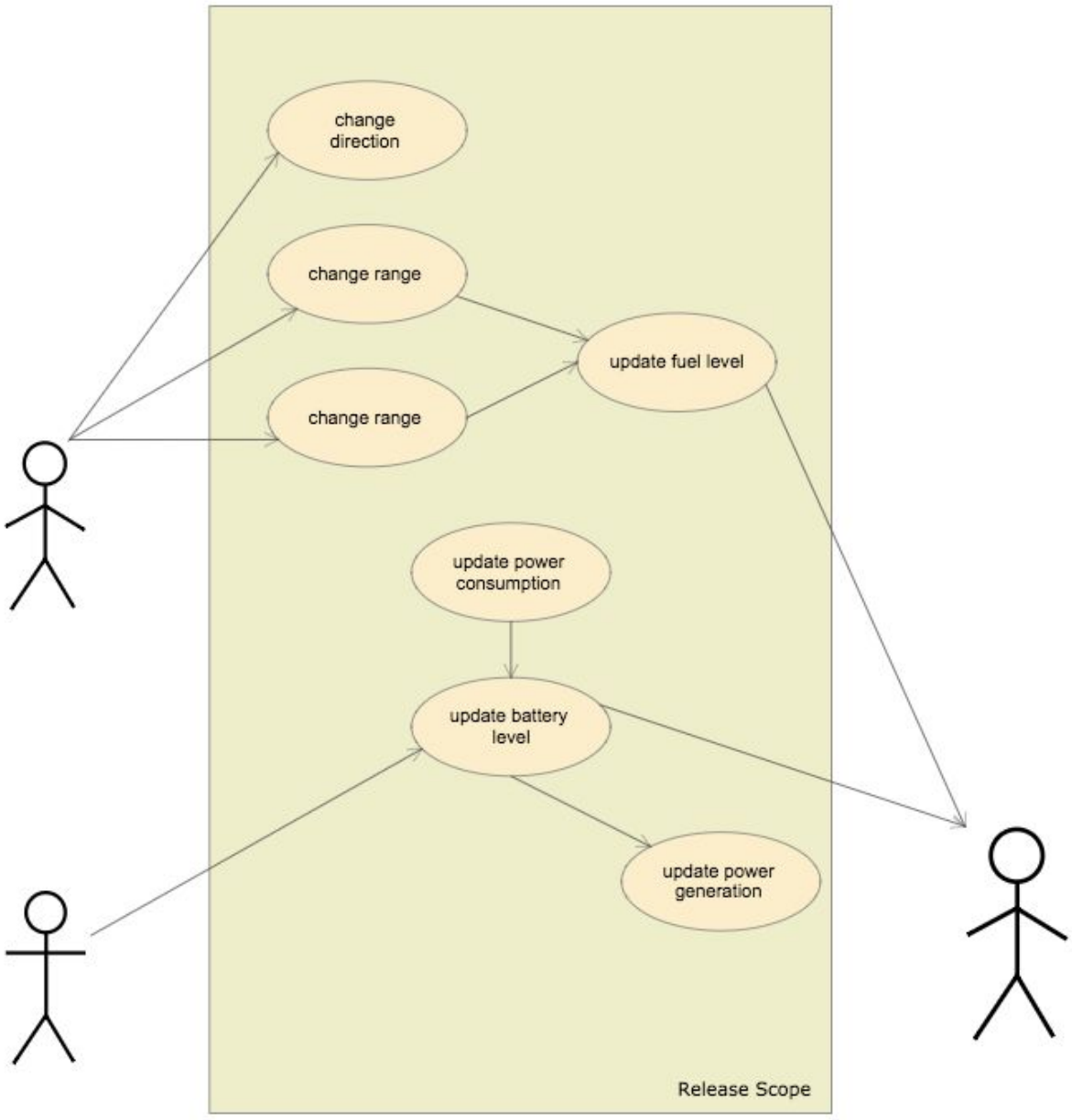
The following section describes in detail how the specifications was implemented in this project. First, a top level view of the code will be explored through a high-level block diagram, and the individual TCB methods will be modeled via UML Activity Diagrams. Then, the relationship and sequence of all the TCBs will be modeled via UML Sequence Diagram. Finally, the function prototypes, data structs, and pseudocode implementation of the project will be examined.

3.1 High-Level Block Diagram

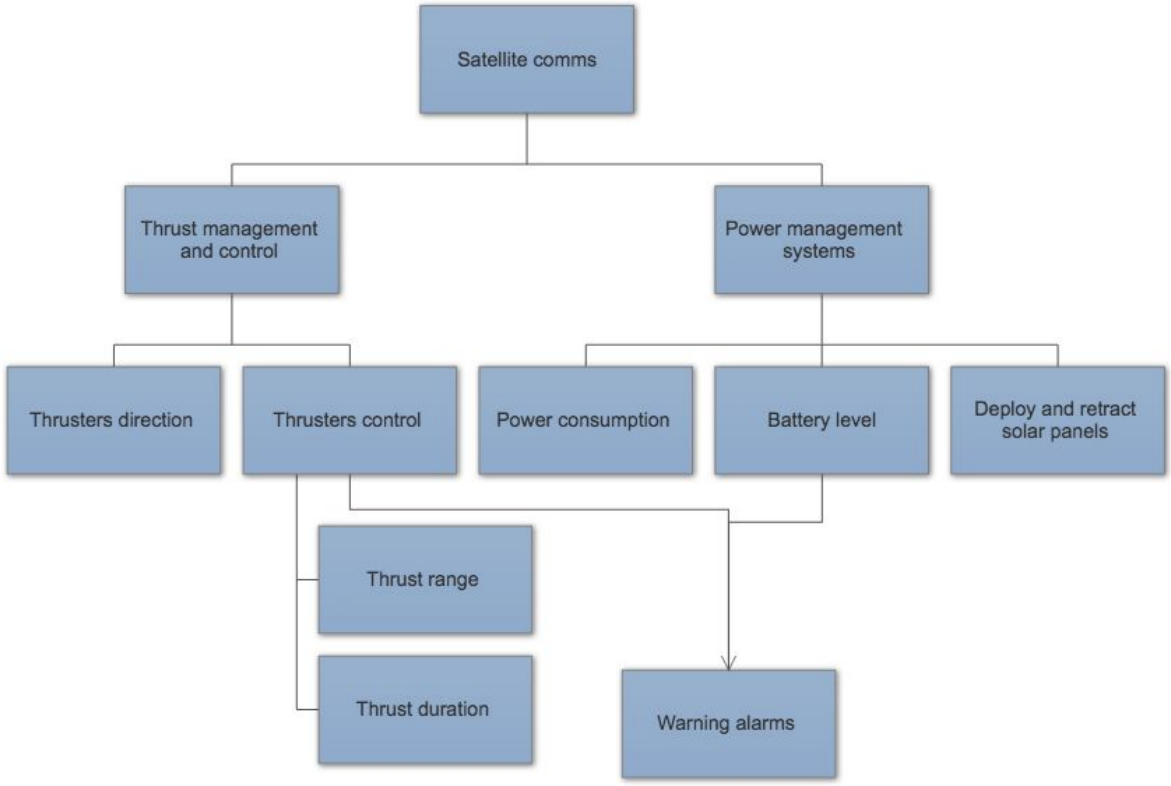


From this diagram, we see that the main function coordinates the five different TCBs. Within each TCB is a reference to a struct with the shared variables and a function that uses them.

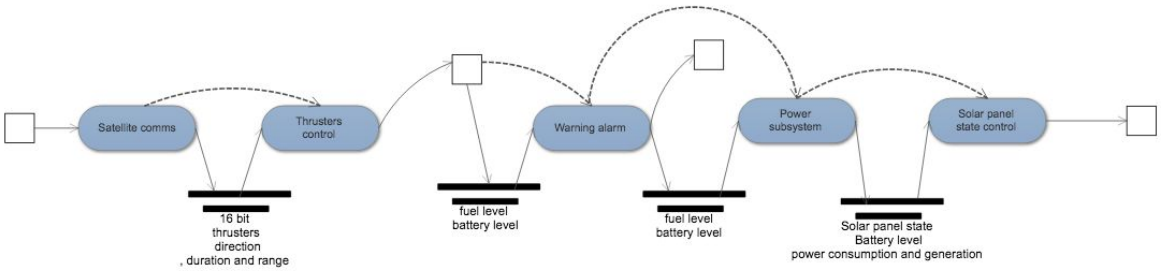
3.2 General UML Diagrams



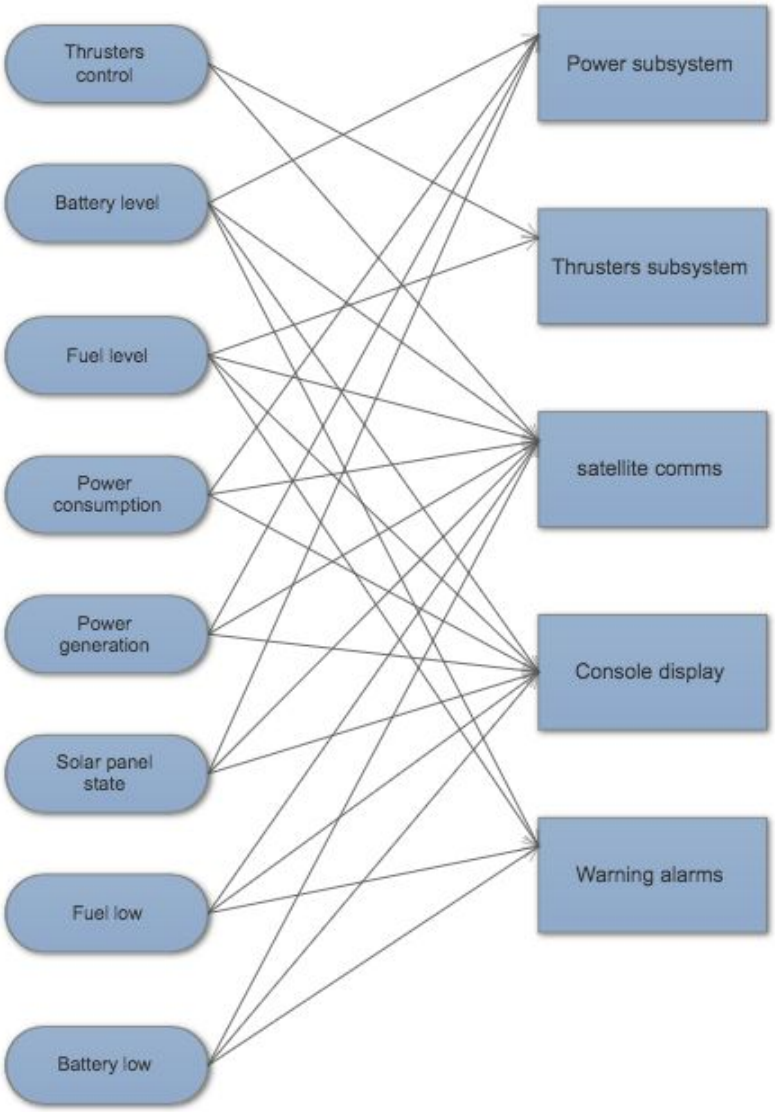
(Figure 1: Use Case Diagram)



(Figure 2: Functional Decomposition)



(Figure 3: State Chart Diagram)

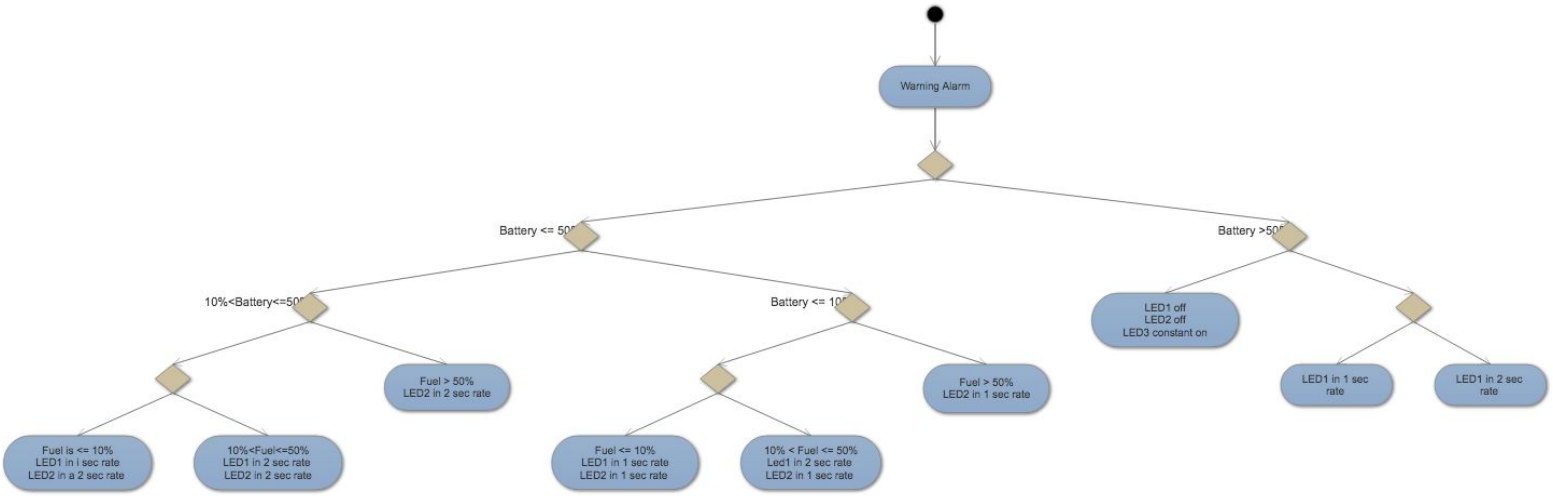


(Figure 4: Shared Variables vs. TCB Methods Diagram)

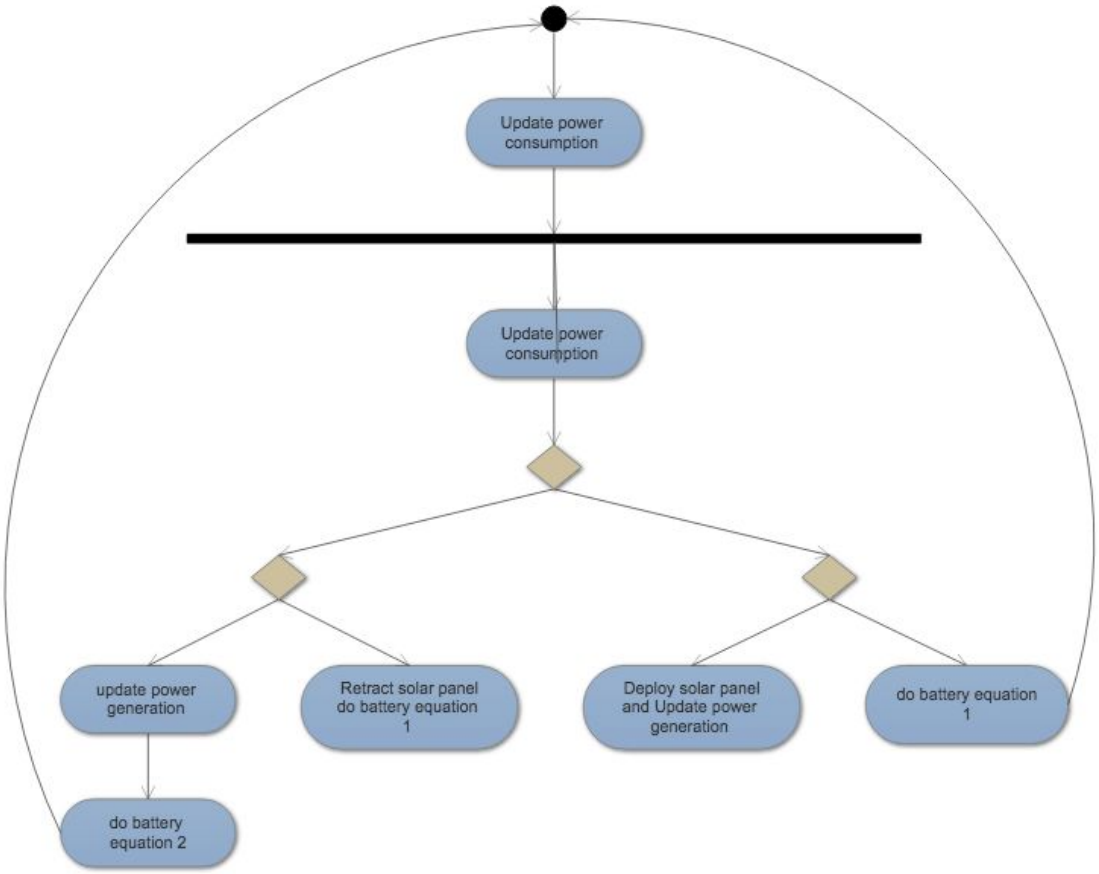
3.3 UML Activity Diagrams

The following section will show the UML Activity Diagrams depicting each of the 5 TCBs in action. (note that “Satellite Comms” is reduced to an activity within the “Thruster Subsystem”)

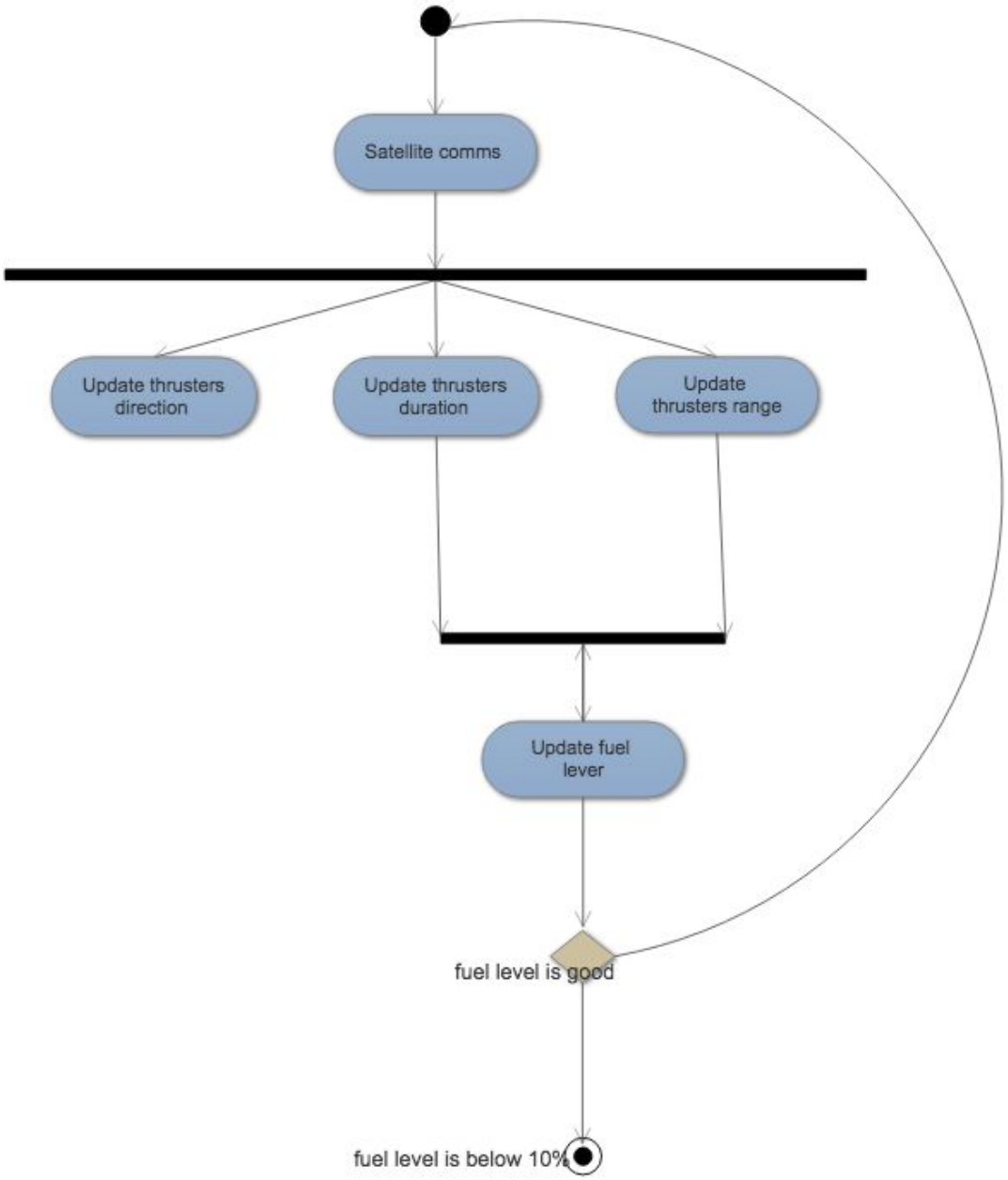
3.3.1. Warning Alarm:



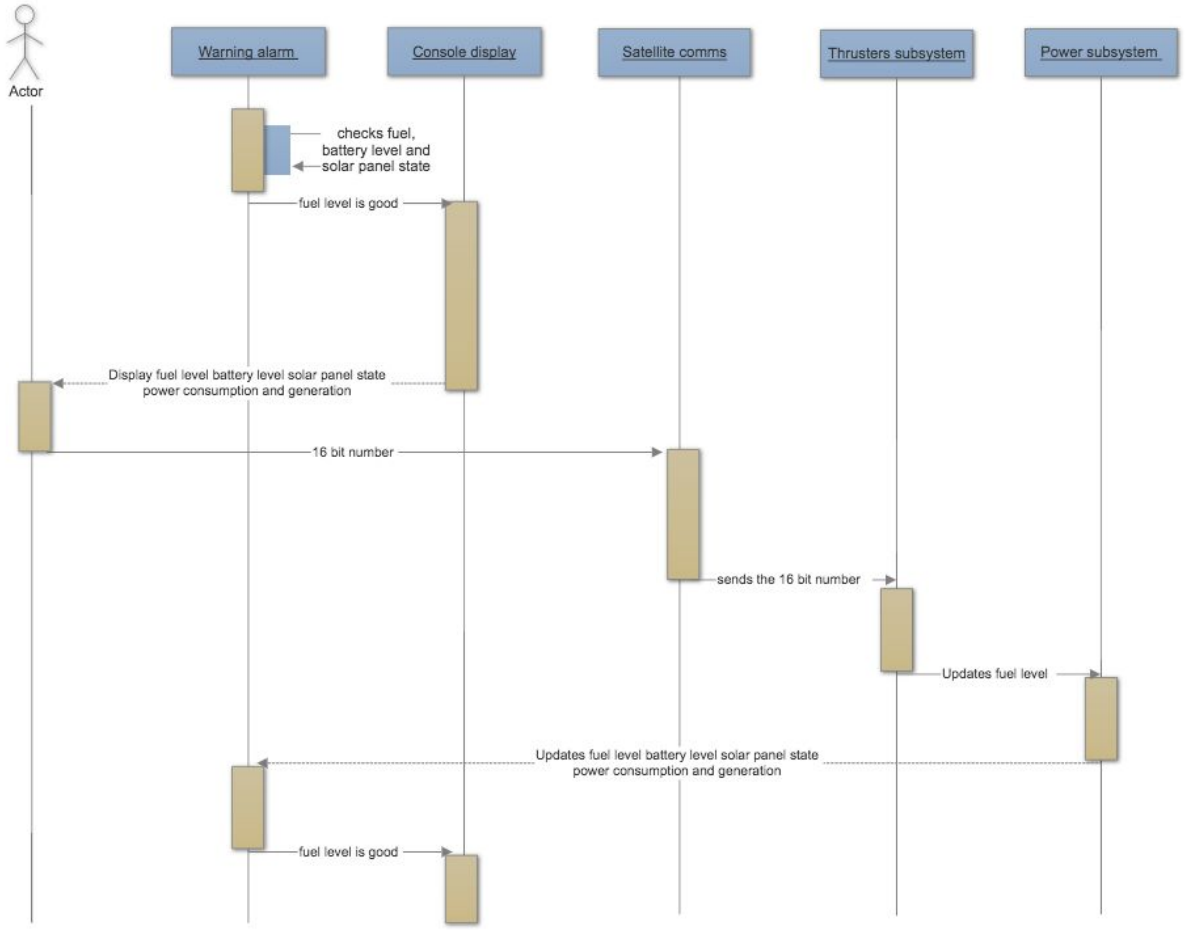
3.3.2. Power Subsystem:



3.3.3. Thruster Subsystem



3.3 UML Sequence Diagram



From this diagram, we can see that the flow of control between the TCBs passes between each other via status checks, conditionally flowing from one TCB to another based on variables such as the “16-bit” command, “fuel level”, etc.

3.4 Function Prototypes for Tasks

The following function prototypes are given for the tasks defined for the application:

- void powerSubsystem(void *powerStruct);
    - Controls the power of the satellite.
- void satelliteComs(void *satStruct);

    - Establishes the comms link between the satellite status and the earth station.
- void consoleDisplay(void *consoleStruct);
    - Displays satellite status and annunciation on the console panel of the satellite.
- void thrusterSubsystem(void *thrustStruct);
    - Controls the magnitude, duration, direction of the thruster, and the fuel usage.
- void warningAlarm(void *warnStruct);
    - Displays the LEDs on the console which warn when the fuel and/or battery is low.

The pointers passed into these functions are structs which contain shared variables that the satellite system.

3.5 Data Structs Defined for Application

The following is the coding for all the data structs used and their associated shared variables.

```
typedef struct powerSubsystemData {
  bool *solarPanelStatePtr;
  unsigned short *batteryLvlPtr;
  unsigned short *pConsumePtr;
  unsigned short *pGeneratePtr;
} powerData;
```

(The power system struct tracks pointers to the solar panel state, battery level, power consumed and power generated by the solar panels when deployed).

```
typedef struct thrusterSubsystemData {
  unsigned int *thrusterCommandPtr;
  unsigned short *fuelLvlPtr;
} thrustData;
```

(The thruster system struct tracks pointers to the thruster command from the earth station and fuel level).

```
typedef struct satelliteComsData {
  bool *fuelLowPtr;
  bool *batteryLowPtr;
  bool *solarPanelStatePtr;
  unsigned short *batteryLvlPtr;
  unsigned short *fuelLvlPtr;
  unsigned short *pConsumePtr;
  unsigned short *pGeneratePtr;
  unsigned int *thrusterCommandPtr;
} satData;
```

(The satellite communication system struct tracks pointers to the satellite status and thruster command. This allows information to be shared with the earth station).

```
typedef struct consoleDisplayData {
  bool *fuelLowPtr;
  bool *batteryLowPtr;
  bool *solarPanelStatePtr;
  unsigned short *batteryLvlPtr;
  unsigned short *fuelLvlPtr;
  unsigned short *pConsumePtr;
  unsigned short *pGeneratePtr;
} consoleData;
```

(The console display struct tracks pointers to the satellite status and annunciation (fuel low and battery low).

```
typedef struct warningAlarmData {
  bool *fuelLowPtr;
  bool *batteryLowPtr;
  unsigned short *batteryLvlPtr;
  unsigned short *fuelLvlPtr;
} warnData;
```

(The warning alarm struct tracks pointers to annunciation (fuel low and battery low), fuel level and battery level).

3.6 Pseudocode

The following sections provide pseudocode implementations of the 5 TCB functions. For a complete implentation in C, see the attached “a2.zip” file.

3.6.1 WarningAlarm

```
warningAlarm:
        fopen leds
        check if fopened successfully
        int battery region = LOW, MED or HIGH
        int fuel region = LOW, MED or HIGH
        bool batteryLow = (battery region == LOW);
        bool fuelLow = (fuel region == LOW);
        if battery and fuel are HIGH
                turn on led 3
                turn off other leds
```

```
else // (either battery or fuel is not HIGH)
        turn off led 3
        if battery is at MED
                if (GLOBALCOUNTER - prev) % 2 seconds
                        flip state of led 2
        else if battery is LOW
                if (GLOBALCOUNTER - prev) % 1 seconds
                        flip state of led 2
        else //(battery is HIGH)
                turn led 2 off
```

Note: The same steps from above are repeated for fuel level, but led 1's state is flipped instead. Also, "prev" gets the value of GLOBALCOUNTER during the start of a minor cycle and holds that value until the next minor cycle.

3.6.2 SatelliteComs

```
satelliteComs:
        Store all shared variables locally;
        thrusterCommand = random 16 bit int;

        print(string storage, all shared variables);
        terminalComs(string storage);
```

Note: SatelliteComs was changed since the demo to more accurately reflect the specifications. See conclusion for more details.

3.6.3 ConsoleDisplay

```
consoleDisplay:
        Store all shared variables locally;
        print(shared variables);
```

Note: ConsoleDisplay was changed since the demo to more accurately reflect the specifications. See conclusion for more details.

3.6.4 Power Subsystem

```
powerSubsystem:

        init local vars for batteryLvl, powerConsumption, powerGeneration, and solarPanelState;
```

```
updatePowerConsumption(powerConsumption);
if (solarPanels_on_currently(powerConsumption, powerGeneration, solarPanelState)) {
        if (batteryLvl could go negative) {
                batteryLvl & powerConsumption = 0;
        } else{
                batteryLvl to min(batteryLvl+powerGeneration-powerConsumption, 100)
        }
} else {
        if(batteryLvl could go negative) {
                batteryLvl & powerConsumption = 0;
        } else {
                batteryLvl = batteryLvl - 3*powerConsumption;
        }
}
```

```
updatePowerConsumption(powerConsumption):

        if(was reversed condition) {  // powerConsumption was > 10
                if(not currently in reversed condition) {  // powerConsumption is < 5
                        flip(condition);
                }
        } else {  // powerConsumption was <= 10
                if(not currently in unreversed condition) {  // powerConsumption is > 10
                        flip(condition);
                }
        }
        if(even function call) {
                powerConsumption = (reversed_condition) -> -2 : (otherwise) 2;
        } else {  // odd function call
                powerConsumption = (reversed_condition) -> 1 : (otherwise) -1;
        }
```

```
solarPanels_on_currently(powerConsumption, powerGeneration, solarPanelState):

        if(solar panel deployed) {
                if(batteryLvl > 95) {
                        retract solar panel;
                        0 out powerGeneration;
                } else {
                        updatePowerGeneration(powerGeneration, powerConsumption);
                }
        } else {  // solar panels retracted
                if(batteryLvl <= 10) {
                        deploy solar panel;
```

```
            }
        }
        return true if solar panels deployed, and false otherwise;


updatePowerGeneration(powerGeneration, powerConsumption):

        if(batteryLvl <= 95) {
            if(batteryLvl <= 50) {
                if(even number of calls) {
                    powerGeneration = powerGeneration + 2;
                }
                else {
                    powerGeneration = powerGeneration + 1;
                }
            } else { // batteryLvl > 50
                if(even number of calls) {
                    powerGeneration = powerGeneration + 2;

                }
            }
        }
```

3.6.5 Thruster Subsystem

thrusterSubsystem:

```
        init local vars for thrusterCommand and fuelLvl;
        init preciseFuelCost;
        parsedComand = parseCommands(thrusterCommand);
        preciseFuelCost = preciseFuelCost + getFuelCost(parsedCommand);
        if(fuelLvl = 0) {
            terminate_the_program();
        } else {
            fuelLvl = fuelLvl - (coerce_to_int) preciseFuelCost;
        }
        preciseFuelCost = preciseFuelCost - (coerce_to_int) preciseFuelCost;

parseCommands(thrusterCommand):

        duration = mask_out_bits_8_to_1(thrusterCommand);
        magnitude = duration = mask_out_bits_16_to_9_and_4_to_1(thrusterCommand);
        thruster_diration = mask_out_bits_16_to_5(thrusterCommand);
        return { duration, magnitude, thruster_duration };
```

```
getFuelCost(parsedCommand):

        cost = 0. 0001284522 * parsedCommand.magnitude * parsedCommand.duration;
        return cost;
```

Note: The "preciseFuelCost" is a decimal representation of the cost are any given function call, and is used to keep provide more precise truncation when setting the integer "fuelLvl" variable. Also, note that the constant "0. 0001284522" was obtained by taking the ratio (100 / (6 months * 0.05 magnitude)).

3.6.6 Scheduler Pseudocode

```
main:

        initialize the globalcounter;
        initialize the majorcycle;
        initialize the majorDelay;
        initialize the minorDelay;

        if (globalcounter % majorcycle == 0){
                delay with majorDelay;
        }else{
                delay with minorDelay;
        }
        globalcounter ++;

non-critical TCB functions:

    static unsigned long start = 0;

    if ((globalcounter - start) % majorcycle != 0  ){
            do not run this function;
    }
    assign start to globalcounter;
```

Note: "Non-critical TCB functions" include every TCB excluding "Warning Alarm" and "Satellite Comms".

4. TEST PLAN

In order to verify that the implementation meets the specification, several questions must be answered: What is the expected state of each data item? How does the system respond in extreme conditions? Does the execution meet the time-constraints?

To answer the first question, the set of valid inputs/outputs must be outlined based on the specification. Once this has been defined, test cases can be formulated to verify the system responds as expected within the average cases.

To answer the second question, the set of invalid input/outputs must be pinpointed based on possible boundary conditions. Once this has been defined, test cases can be formulated to test the robustness of the system within the worst cases.

To answer the final question, the system must be prototyped in order to observe and verify the program meets the specified time constraints. Once this has been defined, performance bottlenecks and mismatched execution can be discovered.

Sections 4 and 5 outline the test specification and test cases, which provide both the “what” and the “how” of this project’s approach to testing.

5. TEST SPECIFICATION

The following table outlines the expected values of the data items within each TCB, outlining the invariants that must be preserved during execution.

TCB	Primary Data	Valid State	Boundary Values
Power Subsystem	1. Power consumption 2. Power generation 3. Solar panel state 4. Battery level	1. 0 <= x <= 12 2. x >= 0 3. ON/OFF 4. 0 <= x <= 100	1. x = 0, x= 12 2. x = 0 3. N/A 4. x = 0, x= 100
Thruster Subsystem	1. Thruster Command 2. Fuel Level	1. 16-bit int 2. 0 <= x <= 100	1. N/A 2. x = 0, x = 100
Satellite Comms	1. Thruster Command	1. 16-bit int	1. N/A
Warning Subsystem	1. Fuel Level 2. Battery Level 3. Fuel Low 4. Battery Low	1. 0 <= x <= 100 2. 0 <= x <= 100 3. TRUE/FALSE 4. TRUE/FALSE	1. x = 0, x= 100 2. x = 0, x= 100 3. N/A 4. N/A
Console Display	1. Fuel Level 2. Battery Level 3. Fuel Low 4. Battery Low 5. Power consumption 6. Power generation	1. 0 <= x <= 100 2. 0 <= x <= 100 3. TRUE/FALSE 4. TRUE/FALSE 5. 0 <= x <= 12 6. x >= 0	1. x = 0, x= 100 2. x = 0, x= 100 3. N/A 4. N/A 5. x = 0, x = 12 6. x = 0

6. TEST CASES

The following table outlines the test cases for each TCB and the subsequent methodology for testing said cases.

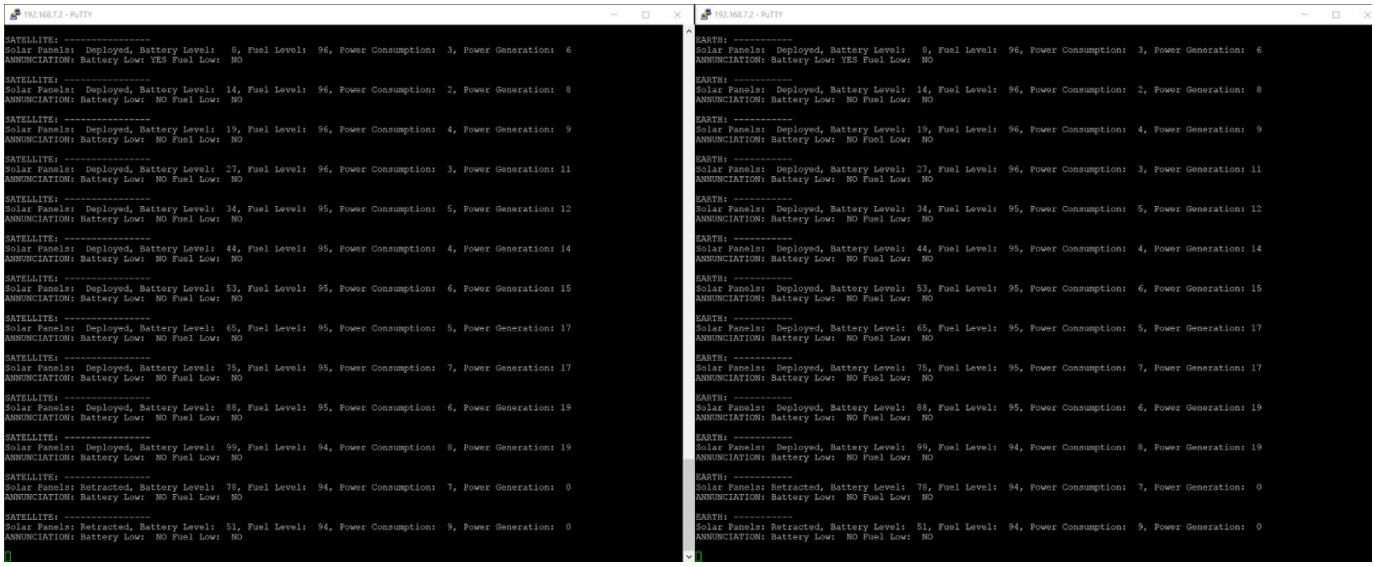
TCB	Testing Method	Test Cases	Tools (if applicable)
Power Subsystem	Unit Testing	*Power Consumption* 1. Reversed condition & even number of function calls. 2. Reversed condition & odd number of function calls. 3. Non-reversed condition & even number of function calls. 4. Non-reversed condition & even number of function calls. *Power Generation* 1. BatteryLvl <= 50 & even number of function calls. 2. BatteryLvl <= 50 & odd number of function calls. 3. 50 < BatteryLvl <= 95 & even number of function calls. 4. 50 < BatteryLvl <= 95 & odd number of function calls 5. BatteryLvl > 95 *powerSubsystem:* 1. Solar panel deployed. 2. Solar panel retracted.	Google Test C++ Testing Framework

Thruster Subsystem	Unit Testing	*Parse Commands:* 1. All permutations of thruster_dirations (up, down,left, right), magnitudes (0-100), and durations (0-255). *Thruster Subsystem:* 1. Test known command and verify cost.	Google Test C++ Testing Framework
Satellite Comms	Unit Testing Printf statements	1. Random integer is no greater than maximum for a 16-bit number (2^16 -1) 2. Integer's third and fourth bit are masked to 0 correctly	Google Test C++ Testing Framework
Warning Subsystem	Paired timing & Observation Printf statements	1. 10 < BatteryLvl <= 50 2. BatteryLvl <= 10 3. 10 < FuelLvl <= 50 4. FuelLvl <= 10 5. BatteryLvl and FuelLvl > 50	Metronome + Stopwatch
Console Display	Observation Printf statements	1. All commands are printed correctly on the console	N/A

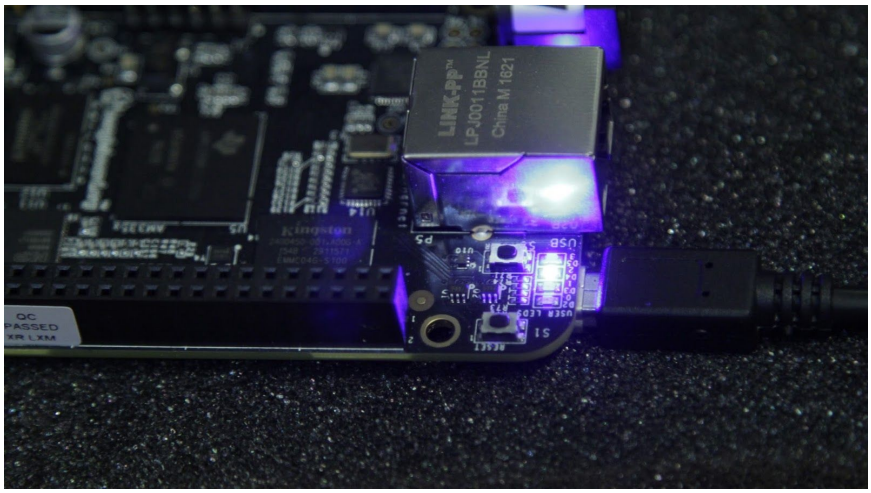
Note: The unit tests can be found within the ".cc" files, within the attached "a2.zip".

7. PRESENTATION AND RESULTS

7.1 Output of the System

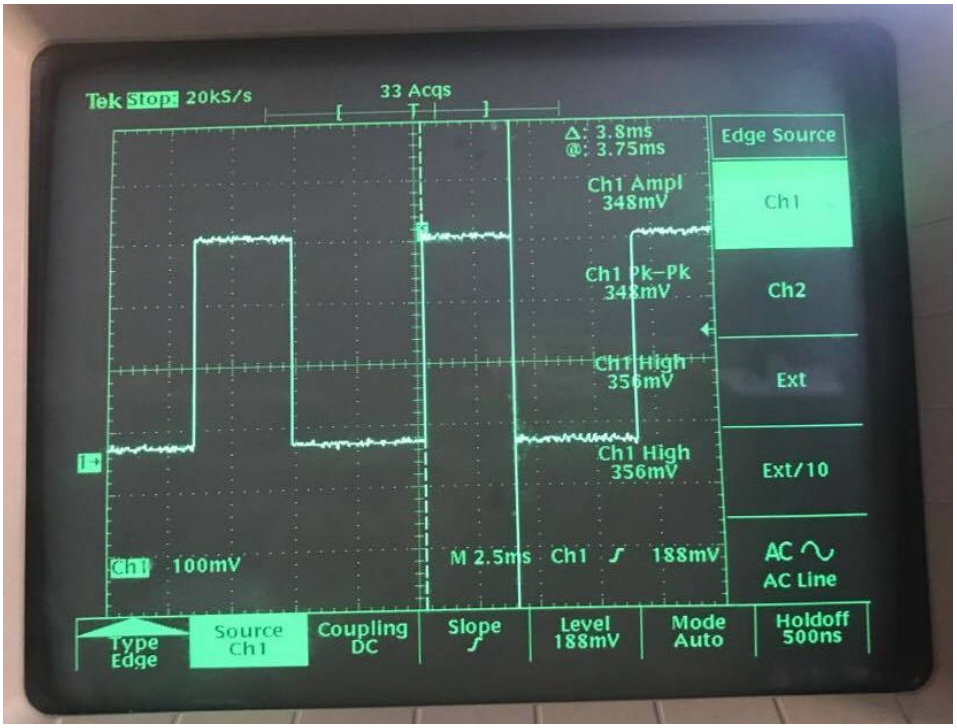


This screenshot shows the information from the shared variables printed on both the satellite terminal and earth terminal. Note that the information for the satellite terminal was displayed by consoleDisplay and that of the earth terminal was sent by satelliteComs. In the screenshot, we see the variables change correctly with each other.



Another output of the system is the functionality of the LEDs to annunciate when fuel and battery are low. The picture shows the LED2 light up and flash when battery level reaches between 50% and 10%.

7.2 Results for Task Execution Time



The snapshot shows a square wave which indicates the length of the task. When the voltage measured by the GPIO pin was high, the task was initiated. When the voltage goes back low, the task ended. Thus we can measure the time interval for how long the voltage of the GPIO pin was high.

Note: The picture of this oscilloscope was taken before we realized that using system calls rather than opening GPIO streams caused extra overhang. This is why the scale incorrectly shows milliseconds instead of the actual microsecond measurements we got. More details are revealed in the Resolved Errors Section.

	How Long Each Task Takes (µs)				
	Power Subsystem	Thruster Subsystem	Satellite Comms	Console Display	Warning Alarm
Trial 1	10	8	10	488	75
Trial 2	6	6	10	480	78
Trial 3	6	8	8	478	80
Trial 4	8	8	10	418	52

Trial 5	6	6	6	456	70
Average	7.2	7.2	8.8	464	71
Std Dev	1.789	1.095	1.789	28.320	11.269
Major Cycle	558.2		Delay Major Cycle	9441.8	
Minor Cycle	79.8		Delay Minor Cycle	9920.2	

The table shows our trial measurements for how long each task runs. Then, we calculated the standard deviation for each task and determined the major and minor cycles. In this case, we considered warning alarm and satellite comms as part of the minor cycle.

7.3 Lab Spec Questions

8. If a stealth submersible sinks, how do they find it?
Sheer willpower and luck.

9. Does a helium filled balloon fall or rise south of the equator?
Rise.

10. If you fly faster than the speed of sound, do you have to slow down every now and then to let the sound catch up?
Probably.

11. If you fly really really fast around the world, can you catch up to the sound before it catches up to you and answer a question before you hear it?
Sure.

12. If you don't answer a cell phone call, where does it go? Is it just sitting there waiting for you?
Were you drunk when you wrote these questions? No beers until after you're done.

8. ANALYSIS OF ANY RESOLVED ERRORS

In this section, we discuss the errors that we encountered and how we resolved them.

8.1 Measuring Errors for files

Filename	File Description	Error (s)	Cause (s)	Resolution
Console Display	Displays onto satellite terminal	Printed to earth terminal when should only printing to the satellite terminal.	Calling terminal Coms when we shouldn't have been	Moved terminal Coms call to satelliteComs and replaced it with printf statements.
Terminal Coms	Sets up another terminal, reads a file and dprints the contents	Terminal was printing the same phrase over and over	File stream that was being printed to didn't overwrite the old phrase	Use fseek twice, one for fwrite and one for fread
Satellite Coms	Sends information to terminal Coms and obtains thruster command from random number generator	Random number generator range of values were wrong.	Call to random number generator had incorrect values	Added a 2^16 mod and changed range to reflect a 16 bit range.
Warning Alarm	Flashes corresponding LEDs based on battery level and fuel level	LEDs were not turning on and off correctly	Didn't save the initial values every cycle of the global counter correctly	Used additional static variables to indicate whenever the state of an LED changed
Power Subsystem	Contains power subsystem	Didn't increment Power Consumption correctly	Wrong constant	Added "3*"

8.2 GPIO Pin Error in measurement

This section is a followup to the error for the empirical task measurement in the presentation and results section. The following pseudocode demonstrates our measurement tactic in *gpio.c*:

```
main:
        open GPIO pin stream
        intitalize an int called "timeTask" // based off the index of the task in the queue
        while(true)
                if timeTask == task index
                        write 1 to gpio stream // task has started

                run task we want to measure

                if timeTask == task index
                        write 0 to gpio stream // task has ended
```

We used the oscilloscope probes at 10x to measure the voltage difference when the gpio is set to 1 and then at 0. We then paused the waveform and used the scope's vertical cursors to determine the time interval, which estimates the task's execution time.

The issue with the image in the presentation and results section was that the GPIO was controlled using a direct system call. We later realized that a direct system call took much longer, giving us an inflated time range to milliseconds. We have since changed to opening a GPIO file stream, which has yielded more accurate results.

9. ANALYSIS OF UNRESOLVED ERRORS

The main unresolved error was the slight imprecision when measuring the task length. Referring back to the presentation and result, an abridged version of the table is posted for convenience:

	How Long Each Task Takes (µs)				
	Power Subsystem	Thruster Subsystem	Satellite Comms	Console Display	Warning Alarm
Average	7.2	7.2	8.8	464	71
Std Dev	1.789	1.095	1.789	28.320	11.269

Ideally, we want the standard deviation to be perfectly 0, meaning the five trials we did for each task would yield the exact same number. To further minimize the error, by the Law of Large Numbers, more trials would have given results that approached the "true" execution time.

10. SUMMARY

The following list summarizes the goals and tasks for the project:

1. Build UML diagrams to get the basic understanding of what should be done to start the programming process of the functions
2. Build a task queue that would run each task in a cycle using structs
3. Share data between each task
4. Set a schedule that selects which tasks to run at specific major and minor cycles based on their importance
5. Use the GPIO pins and the oscilloscope to figure out the time it requires to run each function and chaing that time so it can satisfy the specs of the report.

We were able to achieve these tasks as outlined by the previous sections.

11. CONCLUSION

In conclusion, this lab helped us better develop an understanding of pointers, of passing pointers to subroutines, and of manipulating them in these subroutines. We also learned the formal design cycle and methodologies such as building UML diagrams and marking the flow of control. Lastly, we got a better understanding of simple task queues and data shared between the queues, as well as software delay functions and how to use basic output operations.

Overall, we gained a clearer picture of the general project and are prepared to get started on the next phase of the project.

Also, as mentioned previously, we have fixed certain aspects that did not quite match the specification. For example, satelliteComs should have been the function to implement terminalComs and send the earth terminal the relevant information, not consoleDisplay. Also, we have since included the motor drive measurement with regards to the solar panels.

12. APPENDICES

We adapted certain supplementary programs written by Jim Peckol and other authors to aid us in getting our system to work.

- We adapted Peckol's random number generator *rand2.c* to generate a 16 bit number for the thruster command.
- We adapted Peckol's *terminalComs0.c* to allow communication between two open Linux terminals.
- We referred to files like *blinkusr.c* by Gavin Strunk, Kernel3.c, and other example code to build our project.

These files can be found under Peckol's Archive of Code here.

13. CONTRIBUTION

Name	Contribution
Abdul Katani	created the skeleton for the schedule and implemented also contributed to warningAlarm.c, powerSubsystem.c. and the lab report. Worked with the team on the GPIO and sis the measurement that is required
Radleigh Ang	Created skeleton for overall system, contributed to main.c, implemented satelliteComs.c, warningAlarm.c, consoleDisplay.c, dataStructs.h and other auxiliary functions.. Contributed to report.
Daniel Snitkovskiy	Implemented powerSubsytem.c and thrusterSubsystem.c (with associated unit tests), set up the Google test suite, and helped execute the timing experiments.

Time Spent with Project

Estimate on Time Spent	**Hours**
Design	7
Coding	40
Test/Debug	80
Documentation	15

Signature

The undersigned testify to the best of their knowledge that this report and its contents are solely their own work and that any outside references used are cited.

Radleigh Ang

_____

Author 1

Daniel Snitkovskiy

_____

Author 2

Abdul Katani

_____

Author 3