**RTOS Final Project – Final Submission**

**Unit Testing Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Number** | **Unit Under Test** | **Description** | **Status** |
| 1 | Fuel Control Task  (Btn 1 pressed) | * Verify proper fuel burn rate update upon button 1 press | Pass |
| 2 | Fuel Control Task  (Btn 0 pressed) | * Verify proper fuel burn rate update upon button 0 press | Pass |
| 3 | Fuel Control Task  (Multiple pressed) | * Verify proper fuel burn rate update after multiple button presses | Pass |
| 4 | Angle Task  (Far left position) | * Verify proper angle update upon far left position press for X amount of time | Pass |
| 5 | Angle Task  (Left position) | * Verify proper angle update upon left position press for X amount of time | Pass |
| 6 | Angle Task  (Right position) | * Verify proper angle update upon right position press for X amount of time | Pass |
| 7 | Angle Task  (Far right position) | * Verify proper angle update upon far right position press for X amount of time | Pass |
| 8 | Angle Task  (No Press) | * Verify that no update occurs when no position is pressed | Pass |
| 9 | LED Task  (LED0 & LED1 healthy) | * Verify LED0 duty cycle is equal to current thrust as a percentage of max thrust * Verify that LED1 duty cycle is equal to the current acceleration as a percentage of blackout acceleration | Pass |
| 10 | LED Task  (LED0 & LED1 blackout) | * Verify LED0 duty cycle is equal to current thrust as a percentage of max thrust * Verify that LED1 duty cycle is equal to 50% and has a frequency of 3Hz during blackout | Pass |
| 11 | LED Task  (LED0 & LED1 crashed) | * Verify LED0 duty cycle is equal to current thrust as a percentage of max thrust * Verify that LED1 duty cycle is equal to 50% and has a frequency of 1Hz after crashing | Pass |

**Functional Testing Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Number** | **Function Under Test** | **Description** | **Status** |
| 1 | Btn0 Operation | * Verify decrease in descent acceleration upon Btn0 press | Pass |
| 2 | Btn1 Operation | * Verify decrease in descent acceleration upon Btn1 press | Pass |
| 3 | LED0 Operation | * Verify LED0 blinks at duty cycle equal to current thrust as a percentage of max thrust | Hard to verify exactly, but blink rate does change upon thrust change |
| 4 | LED1 Operation #1 | * Verify LED1 blinks at duty cycle equal to current acceleration as a percentage of blackout acceleration in healthy flight conditions | Hard to verify exactly, but blink rate does change upon thrust change |
| 5 | LED1 Operation #2 | * Verify that LED1 blinks at duty cycle equal to 50% and frequency of 3Hz during blackout | Pass |
| 6 | LED1 Operation #3 | * Verify that LED1 blinks at duty cycle equal to 50% and frequency of 1Hz during blackout | Pass |
| 7 | Capsense Operation | * Verify proper change in angle upon capsense press | Pass |
| 8 | LCD Operation #1 | * Verify proper LCD display during a safe landing condition | Pass |
| 9 | LCD Operation #2 | * Verify proper LCD display during a blackout condition | Pass |
| 10 | LCD Operation #3 | * Verify proper LCD display during a crash condition | Pass |

**Project Status**

I have completed all baseline deliverables required from the project description for option A. One thing that I did change about the requirements is that I changed the units used for most of the parameters in the physics configuration data structure. For example, I changed the angle change quanta from mrad to rad.

Overall, the game is definitely playable. It is possible to land successfully, crash, as well as reach blackout and recover from blackout. Upon reaching a game ending condition (land success or crash), the game waits about 10 seconds before starting up again.

**Summary of Effort & Progress Estimates**

I have completed 100% of my currently-scoped, estimated work (18/18hr) in 111% of the initially-estimated time (20/18hr). My best guess of my say/do ratio is 94%, so to unbias my estimates after this class, I may want to multiply my estimates by 1.06 (100%/94%).

My Latest scope is 100% of my original scope (35hrs, vs. 35hrs).

**List of In-Scope Work Items**

* Completed this week
  + Physics Task Implementation (6hr)
    - I completed the physics task. This definitely took much longer than expected, as it was very difficult to determine what physics configuration data would make the game playable. I ended up determining which values to use by plugging in values at random and using the debugger to make educated guesses on what values could possibly make the game work.
  + Graphics Task Implementation (6hr)
    - The graphics task was pretty simple to implement. I just took values from the physics task to determine where the spacecraft should be positioned. The hardest part was figuring out how to draw the spacecraft tilted at an angle.
  + LED Task Implementation (5hr)
    - The LED task was also pretty simple to implement. I utilized two different timers in order to produce PWM blinking from the LEDs. I also used data produced by the physics task to determine how the LEDs should be blinking.
  + Functional Test Development (5hr)
    - The functional test development was much less time consuming than I originally thought. Mainly due to a misunderstanding of what the functional tests are.
  + Functional Testing (2hr)
    - Functional testing is complete, and the game passed every test. The LED, LCD, physics, angle, and fuel control tasks all work as desired based on the functional test results.
* Complete
  + Design Planning (2.5hr)
  + Angle Task Implementation (2hr)
  + Fuel Control Task Implementation (2hr)
  + Unit Test Development (4hr)
  + Unit Testing (0.5hr)

**Analysis of Solution**

RT Tasks

(In order to view Segger, I need to add OS\_CFG\_TRACE\_EN to the os\_cfg.h file. However, adding this results in include errors within my project so I have decided not to include the Segger screenshot.)

*Task Priorities*

* Physics Task (1st priority)
* LCD Task (2nd priority)
* Fuel Control Task (3rd priority)
* Angle Task (4th priority)
* LED Task (5th priority)

Using the above priority settings, I did not notice any conflicts between tasks that affect the required operation of the game.

Code Space

Flash used: 69792 / 1024000 (6%)

RAM used: 32120 / 256000 (12%)

Physics Task Approach

For the physics task, I did not actually notice many limitations other than the fact that it needs to update at a certain frequency in order for the game to play smoothly. This was not a difficult challenge to overcome, as I simply made the physics task have highest priority, and woke it up periodically using a timer.

In order to actually calculate correct physics values, I just used a data structure to hold important values and used physics equations to update them. For example, in order to calculate velocity, I used the velocity from the previous physics update, and added it to the acceleration multiplied by the time since last physics update. I determined the delta time by utilizing a systick handler to count milliseconds since the last physics update.

Overall, the main time consuming portion of implementing the physics task was determining valid configuration values for the physics data structure. Other than that, performing the actual calculations and updating the values was not too difficult.

Scaling of Variable Spaces

In order to determine playable variable spaces, I ended up randomly selecting values and testing them out. The debugger helped me out a lot, and allowed me to make educated guesses on which values could possibly work out. Backtracking was a useful method, and allowed me to figure out which values to choose by having an idea of what outcome I wanted. For example, if I wanted the vertical position to increase by one pixel every time the physics task updated, I would think about what velocity, acceleration, and thrust I would need in order to get that result.

Overall, my physics configuration values are pretty randomly selected, and not based on values from a real lunar lander. However, the values I selected do make the game playable.

Future Development

If I had another 2 weeks to work on the project, I would likely add randomness to the initial position and velocities of the spacecraft in order to make the game a little more difficult. I would also add a score indicator in order to show how many successful landings have been made in a row. Lastly, I would tweak the physics configuration values to make the game slightly more smooth.