**Functional Testing**

1. Holding the board with the LCD closest to my chest, parallel to the ground the LCD should read “speed: 0” and “direction: straight”. **PASS**
2. Holding the board with the LCD closest to my chest, parallel to the ground the LCD, rotating counter clockwise (fast), should read “speed: 0” and “direction: hard left”. **FAIL**
3. Holding the board with the LCD closest to my chest, parallel to the ground the LCD, rotating clockwise (fast), should read “speed: 0” and “direction: hard right”. **FAIL**
4. Pressing the button should increment the speed by 5. **FAIL**
5. Incrementing the speed (pressing the button 16 times), the LCD should read “speed 80” and the green LED should come on. **FAIL**
6. Speed can be incremented/decremented based on how long the button is held. **PASS**

NOTE: It is worth noting that, when the LCD is taken out of the picture, the data does make it way through the tasks and tasks: 1, 2, 3, and 4 **ALL PASS.** There reason for this is mentioned below.

**Why Functional Tests Fail:**

A screenshot of a computer

Description automatically generatedWhen the LCD Display Task is added into the mix, everything goes to hell-in-a-handbasket (Figure 1). As can be shown, the LCD display task blocks all other tasks, despite having the lower task priority than the sampling tasks, and the mutexes it pends on having *osMutexPrioInherit* attributes to prevent deadlock on them.

Figure 1: LCD Display task blocking

The code seems to spend all of it’s time in the LCD\_DrawChar() function inside nested for loops. Not sure why.

A screenshot of a computer

Description automatically generatedHowever, when the LCD Update function is commented out, everything works swimmingly (Figure 2)! Shown are the wakeup timers (I couldn’t figure out how to get their names to appear in SystemView), the button ISR, four of the five tasks, and most importantly – LCD Display Task not blocking!

Figure 2: lcdUpdate() commented out

*A screenshot of a computer program

Description automatically generated*To show the code functioning properly, without the LEDs and LCD, I added some global variables (that only the LCD Display Task accesses, so no data race worries) to show the code working in unison (Figure 3). The LEDs don’t function because there is an error with the way the vehicle monitor task is starting and stopping the direction alert timer. As a result, I threw the LED task code behind a debugging compiler switch so it doesn’t throw compiler warnings while I work on other things.

Figure 3: Debugger showing realtime updating

A computer screen shot of a program

Description automatically generatedFigure4 shows where the \*\_LCD global variables are being updated. The updateLCD() function is commented out because it blocks every other task (see figure 1). I suspect that there is something going on with the driver (or maybe how I’m using it) that is causing it to block every other task. I know it isn’t the mutexes or semaphore causing the blocking because, as figure 3 shows, there is no blocking behavior occurring without the updateLCD() function being active.

Figure 4: Code showing variable placement

**Questions:**

**Q: Check the scheduling of each task. Is each task scheduled as expected? Explain.:**

Yes! The button ISR releases a semaphore, which then wakes up the Speed Setpoint Task. Once the task wakes up it promptly starts a timer to time how long the button is pressed. The task promptly pends on the semaphore again, waiting for the ISR to release it once the button is released. Behavior can be see on the next page (figure 5).

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Figure 5: Button ISR waking up Speed Setpoint Task

Figure 6: Button Hold timerelapsing after 1 second

If the button is held for more than 1 second, it well then decrement the speed by 5 (paying attention to not allow the speed to go negative). Figure 6 shows the second ISR event occurs at -1.5 second (button press), and the button is released slightly after -500ms. The timer callback is then called at -500ms, showing that the task is working properly. The event flag then signals to the vehicle monitor task that speed data is available (see figure 2).

A screenshot of a bar code

Description automatically generatedFigure 7 show that the Vehicle Direction Task is being woken up every 100ms as a result of a periodic timer. This is exactly what is expected. The vehicle direction task samples the gyro and then signals to the vehicle monitor task that direction data is available (see figure 2).

Finally, the LCD pends on a pair of semaphores for the speed and direction data and is woken up periodically by a wakeup timer. The task working (minus the output to the LCD) can be shown in the debugger (figure 3). The LEDs pend on event flags, but I can’t show this working at this time.  
Figure 7: Vehicle Direction Task waking up every 100ms

A screenshot of a computer

Description automatically generatedLastly, the direction alert timer callback function (0x10026A28, I couldn’t figure out how to get the names to show in SystemView) shows that the code CAN detect a direction violation (figure 8). However, it’s not very reliable due to the fact that my direction logic is flawed in that if you stop rotating, but aren’t parallel with the ground, the gyro will read it as “driving forward”. This is because I reused my gyro code from previous labs and didn’t modify it at all. So, in theory, I should be able to drive the LEDs but I didn’t get it working fully.  
Figure 8: Direction Alert Timer Callback