**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”. **FAIL**
2. Holding the board with the LCD closest to my chest, parallel to the ground the LCD, rotating counterclockwise (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. Holding the board with the LCD closest to my chest, parallel to the ground the LCD, rotating counterclockwise (fast), should read “speed: 0” and “direction: Grad Left”. **FAIL**
5. Holding the board with the LCD closest to my chest, parallel to the ground, rotating clockwise (fast), should read on the LCD “speed: 0” and “direction: Grad Right”. **FAIL**
6. Pressing the button will increase the speed by 5. **PASS**
7. Incrementing the speed beyond 75, the green LED will come on. **PASS**
8. Holding the button down for one second will decrease the speed by 5. **PASS**
9. If the speed is currently 0, holding the button down for one second will NOT decrease the speed by 5 (negative speed doesn’t make sense). **PASS**
10. Incrementing the speed beyond 45 and turning the board will result in a speed violation and the green LED will turn on. **PASS**

NOTE: It is worth noting that, when the output to the LCD is removed from the functional tests (just the function that displays the character data; the task itself works otherwise), the data does make it way through the tasks (see figure 3) and functional tests: 1 - 5 **ALL PASS.** The reason for this is mentioned below.

A screenshot of a computer

Description automatically generated**Why Functional Tests Fail:**

When the LCD Display Task’s updateLCD() function is added into the mix, everything goes to hell-in-a-handbasket (Figure 1). As an shown, this function will cause the LCD display task to block all other tasks, despite having a lower task priority than the sampling tasks, and the mutexes it pends on having *osMutexPrioInherit* attributes to prevent deadlock on them. The code seems to spend all of it’s time in the LCD\_DrawChar() function inside nested for loops. Not sure why yet.

A screenshot of a computer

Description automatically generatedHowever, when the LCD Update function is commented out, everything works swimmingly! Figure 2 shows that 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

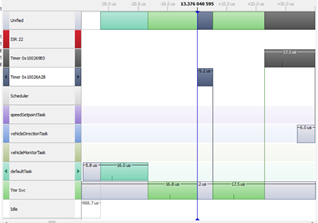
The direction alert timer callback function (0x10026A28, I couldn’t figure out how to get the names to show in SystemView) shows that the code CAN detect a direction violation (figure 8). However, it’s not very reliable due to the fact that my direction logic is flawed in that if you stop rotating, but aren’t parallel with the ground, the gyro will read it as “driving forward”, since it is no longer moving. This is because I reused my gyro code from previous labs and didn’t modify it at all. It’s worth noting that figure 8 was taken BEFORE integrating the LED Output Task, figure 9 does, however, show it working with speed violations.

Figure 8: Direction Alert timer callback triggering

A screenshot of a computer

Description automatically generatedFigure 9 shows the button ISR being triggered as I increment the speed beyond 75. You can track the system at work here:  
button ISR --> wakes up speed setpoint task --> signals to the vehicle monitor task to wake up and read in the new speed data --> signals the led output task that a speed violation has occurred.

Figure 9: Speed violation detected!