Final Project Report

This week, we finished the entire final project and successfully demonstrated it in the lab session. During the demo, we performed several functional tests and they all passed:

1. We only have a single chance of using the laser button, which will destroy the ball and the next ball will come out at the same place as the previous ball did. If the current ball is the last ball, then we automatically win the game.

2. If the ball misses the platform, we lose the game.

3. If the ball hits the platform but the platform is not boostered, then the ball should bounce back but it should bounce lower and lower as time passes.

4. If the ball hits the platform and the platform is boostered, then it should bounce higher until it hits the ceiling.

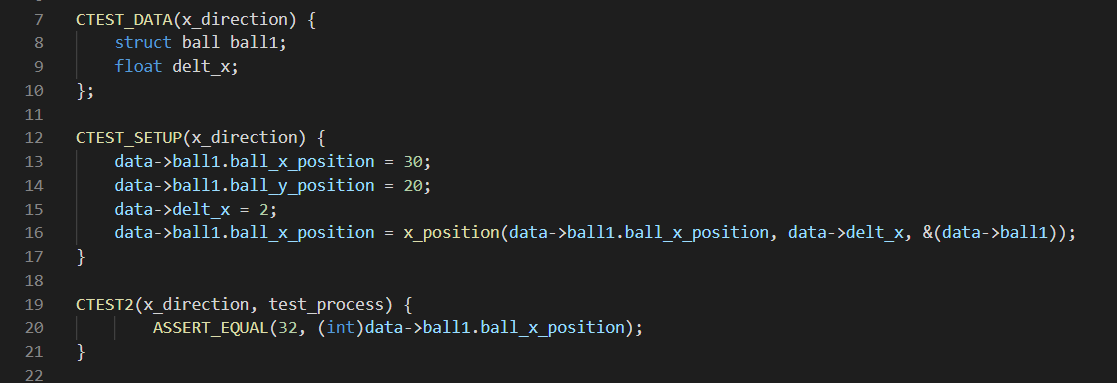
5. If the ball hits the ceiling, the next ball will come out at the same place as the previous ball did unless this is the last ball.

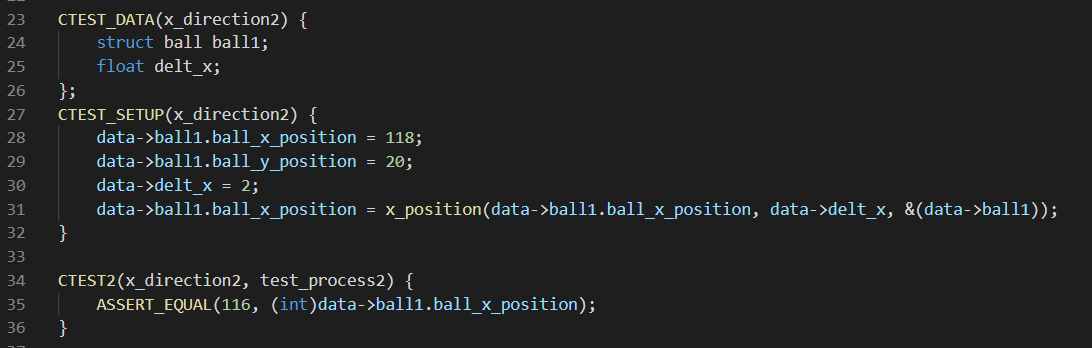
6. When the ball hits either side of the wall, then it should bounce back.

7. If we press the outer side of the slider, we apply the max force to the platform. So the LED should light up constantly.

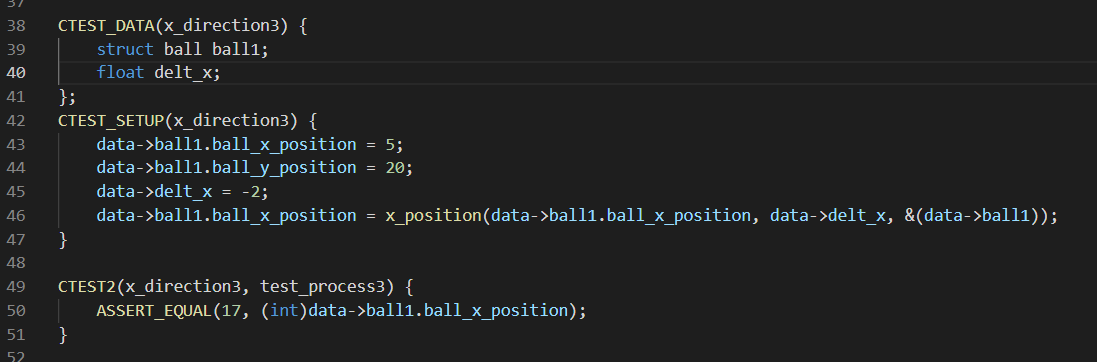
8. If we press the inner side of the slider, we apply the half force to the platform. So the LED should light up half of the period and turn down half of the period.

In addition to the functional test, we also did some unit tests:



Here is a unit test example of the physics task. We were assuming that the current x position of the ball is 30 and the current y position of the ball is 20, which is inside the bound of the left and right wall. So in this case, the ball should move from its current location to x + delt\_x, which should be 30 + 2 = 32. The below image shows another example:

In this example, assuming the current ball location is (118, 20), which is to the right of the right wall. According to the functional test, the ball should bounce back, so the x location should end up being 118 - 2(delt\_x) which is 116.

Here is another example:

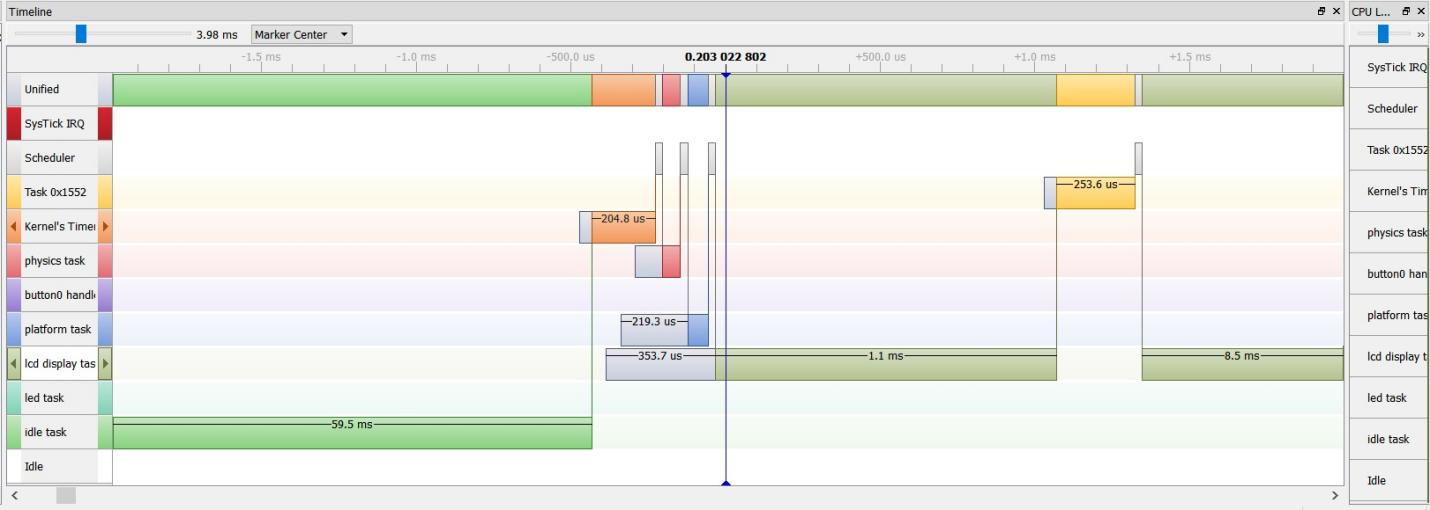
In this case, the current x position is to the left of the left wall. By default, it should bounce back. Since 5 is already outside of the boundary, the ball will first bounce back to 15(10 + 5) since the x position of the left wall is at x = 10. Then there is another delt\_x applied to the ball in the opposite direction. So it should be 15 + 2 = 17.

This week I did a final check of my codings and made sure they worked correctly for the presentation. I also added more balls so that the game won’t be very easy to win.

So far I have completed 99% of the work. The only task left is the right LED, where we should predict the amount of force we need to apply to the platform so that the ball can hit the center of the platform at their first intersection. Other than that, all the other functional tests work just as expected.

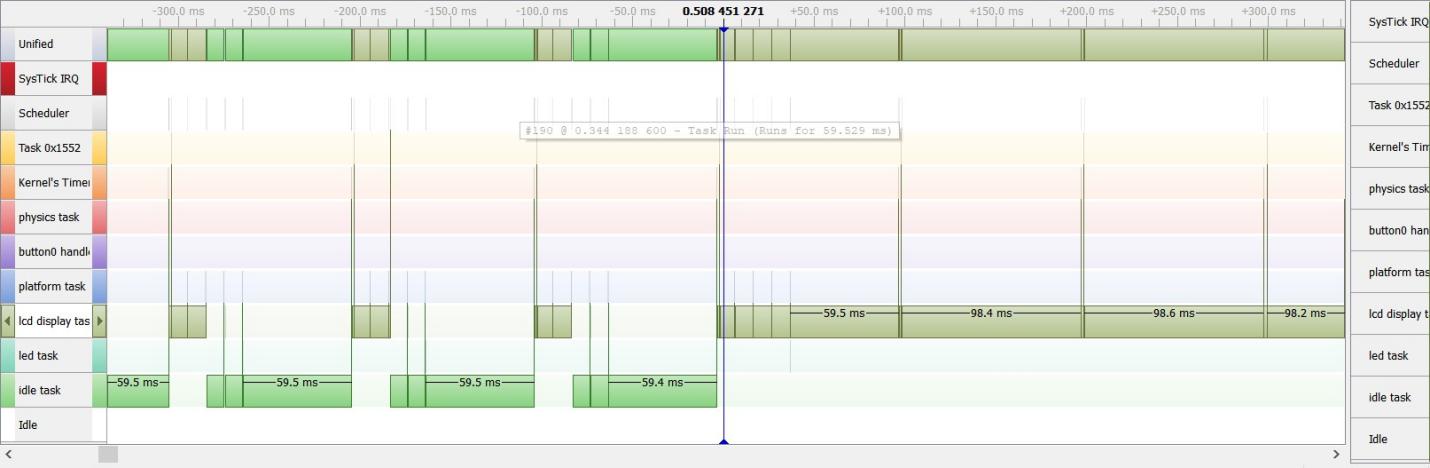
In order to make the game more playable, we changed the default settings of tau physics and tau lcd from 3 ticks to 1 tck. Which means physics task and lcd task will update every 10 ms instead of 30 ms. When I upload that onto the board, it works just as we expected. The ball and platform were moving more smoothly than before. I have also changed some of the other parameters. For example, originally I used 9.8 as the value for gravity. But once I actually play with it, the ball fall to fast, so I changed that to 1.1 to make it fall slower.

Although almost all of the tasks were completed, there were still lots of challenges. One of the most challenging errors that I have encountered was the semaphore post and pend from the btn1 irq handler. For some reason it just shows me some error. I still cannot fix that even with TA’s help. So in order to mitigate it, I just did everything related to btn inside the irq handler. Luckily there are not very many things to do so I won’t worry about the priority blocking too much.

For the priority, we set the physics task and button task both to be 10, platform task to be 20, LED and LCD task to be 30, and idle task to be 50, which is what it supposed to be because we want to handle the button task as soon as it’s pressed. So we want it to have the highest priority. 

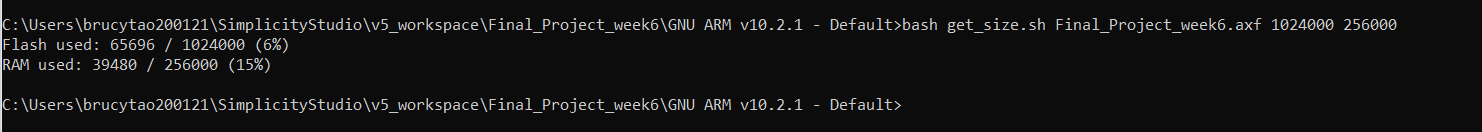
Here is a case where we haven’t pressed the buttons yet. We used tau physics to trigger both the physics task and platform task. As we can see as the timer counts down, the physics task and platform task has been called sequentially. Then as tau lcd counts down, lcd task has been called. 

Here is another image where we pressed the button. As we can see, since the button task has the highest priority, it’s been called as soon as it’s pressed and it’s before other tasks.



Here is an image where it’s game over. If the game is over, it should continually show “game over”. So lcd tasks should constantly be called.

Once we run the get size simulation, we can see that it’s taking 6% of the flash and 15% of the RAM. The reason for that is probably because I used too many variables so that it will take more space to calculate it. The other possible reason is that we included the math library, which is necessary when we were calculating the square root. When we have included the math library, that means we included all the contents inside, that could be another reason for the percentage of flash and RAM.



If I have another 2 weeks to work on the project, I will continue work on the last part of the LED task and make sure the project is 100% complete.