# <u>COMPENG 3DQ5 - Tuesday Group 23 Lab Report 1 Prawin and Sava</u>

Our reasoning for our code mainly consisted of logic and similarities from the previous experiments 1-5 which were then used to help with the take-home exercise. Our take-home exercise was an extension of experiment 5 with the same push button functionalities where the counter either goes up with the press of button 1 or goes down with the press of button 2. For this, we added a variable named “up_count” and set its state to either 0 or 1 under the push button status conditional statements for simplicity. Then we added code logic under where the counter is incremented to either go up or down based on the state of up_count. The counter rollover conditions for the timer were similar to experiment 4, but it was modified so that we could account for both the counter going up and down. For example, once the counter reached 55 (instead of 59), it would roll over to 0. To prevent it from reaching hexadecimal numbers, every time the one's digit of the counter would go over 9, we would reset it to 0 and the ten’s digit column would get incremented by 1. Some more modifications made was that when the counter reached ‘ff’, which was the counter’s state after 00 when counting down, it would then set the counter to 55 to achieve the rollover effect as tasked with.

For the next task, when the counter is stopped and button 3 is pressed, we followed a similar method for loading 33 as we did for the counting direction. To successfully load 33, a variable “load_count” was initialized and it was set to 1 under the push button 3 conditional statement, while the conditional check for load_count == 1’b1 was set under the clock cycle. However, the initial problem faced with this was that after push button 3 was pressed and every time the counter was subsequently stopped, 33 would be loaded every time without push button 3 being pressed. This issue was eventually troubleshooted by resetting the “load_count” variable to 0 under push button 0 (the stop button), so that every time the counter is stopped, you would have to re-press button 3 in order for 33 to be loaded onto the counter.

For the last task, we used fundamentals learned from experiment 1. This was to test the different LED reactions to specific switches that were pressed. The eight different LEDs were lit up based on the different switches that we pressed. The logic for each of the eight LED’s was implemented in the last line of the code, where we called 8 different sets ofSWITCH_I for the LED initialization. For example, the LED 8 only lit up when the number of switches between 15 and 11 was odd, meaning we should make it an XOR gate. This meant that ^SWITCH_I[15:11] was the way to initialize this. We did a similar implementation for all the other LED logic, with the last one being a bit longer due to the complexity. LED 0 would only light up if three of the four switches from 0 to 3 were HI. Henceforth, we used SWITCH_I[3] + SWITCH_I[2] + SWITCH_I[1] + SWITCH_I[0] ==3 to present this, with the + signs representing an or gate that has to equal to 3 to be lit up.