GPIO Configuration:

In the given code “statemachine.c” the first three lines of code under the systemInit() function enables the GPIO clock for port A, B, C by configuring the advanced peripheral bus 2 (APB2) register. The individual configuration for these registers was already done and I made no additional changes. Then I inserted a function called “timer_ms_init();” this function configures the timer but does not directly tie in with GPIO configurations and will be discussed later.

Afterwards I saw the initial setup for the LEDs which needed GPIO E to be turned on, which was already given. All I needed to do was to set the LED mask which I copied from my lab 6 code, which is a constant long set of left shifts of 1 bits the exact code is “{1<<15, 1<<14, 1<<13, 1<<12, 1<<11, 1<<10, 1<<9, 1<<8}”. This number was then put above main and set as a global variable to not create any errors while passing variables. Then I also set up the clock for the analog to digital convertor(ADC) by configuring the APB2 register by shifting 1 to the 10^{th} bit using “RCC->APB2ENR |= 1<<9”. The configuration of the ADC was taken from lab 6 which was taken straight from the lecture.

I wrote a blink(); function to use in Gear 4 which set the output data register of GPIOE to the LED mask all within a for loop that repeated 8 times once for each of the 8 LEDs with a secondary delay loop inside the main fore loop to delay the on and off each LED.

Timer Configuration:

First, I set up the timer by enabling the clock to timer 4 by setting 1 to the 3^{rd} bit in APB1 register. I picked this register because it was a standard, general purpose timer as stated in the lecture slides. Then I enabled the GPIO clock and alternate functions by setting 1 to bits 1 and 4 to the APB2 register. I followed Johnknee’s microcontrollers website for exact calibrations of the

timer. First by setting the timer to default by setting 0 to TIM->CR1. Then set the Base clock frequency which operates at 8Mhz as the source with TIM4-> SMCR &= !7. Since the base frequency is 8Mhz I needed a way to limit this clock to only pulse at every 1 ms to do so I needed a pre-scalar at 8000 since 8,000,000 Hz /8,000 = 1000hz which equals 1 millisecond to do this I set the TIM4->PSC = 7999 since 0 counts as 1 therefore the pre-scalar had to be one less than the actual pre-scalar value. Finally TIM4->CR1 = 0x0001; was used to enable the timer after it was configured.

With the timer counting every 1ms I had each function wait a fixed amount of milliseconds, for example in Gear 1 the program called for a wait of 3 seconds, so I had the program loop in a while not loop. The condition for exiting this loop was if the TIM4->CNT was lower then the desired amount of repetitions then the Gear would not change. Naturally for these counters to work the timer had to be reset whenever entering a new count or Gear. I used TIM4->EGR to reset the timer by setting it to 1 whenever I needed a reset.

Functionality Claims:

My program successfully executed what was needed to complete the lab however, I had to make some improvised changes to my timers. When I originally tried to set the timer I wanted to use a Wait_ms(int) function to loop, however due to my inexperience with C code my function was not working so I just implemented the loop to each state manually. Furthermore, when I originally tested the timer it ran too fast. I didn't want to mess with the timer configuration since just changing the scale of each counter was a more simplistic solution I increased each counter by 10 fold so in the case of Gear 1 the count was increased from 3000 to 30,000 which when I tested the program again, it seemed to count at 3 second intervals before moving to the next gear.

Conclusion:

While my program successfully performed the tasks, I don't think the timers are precisely accurate to exactly 3 seconds. I learned that the timer pre-scalar configuration starts with 0 which counts as a pre-scalar as a 1. Perhaps my code has a faulty base clock frequency setup. Since I'm sure if the base clock had been 8Mhz then the Pre-scalar would have worked as intended without needing a patchwork solution.