LABORATORY 2:

Printf Debugging, Pulse-width Modulation (PWM) and Motor Control

ECE 4240 B01

October 1, 2025

Dayaveer Gill – 7969885

4.0 - LABORATORY EXPERIMENTS

PART 1 – PRINTF AND STANDARD OUT REDIRECTION TO SWV ITM PORT 0

For the first part of this lab, we started off by formatting the board to as instructed in the lab 2 manual. After getting the project up we added the required files from UMLearn to their appropriate folders and then we began to write code for the first part using the printf statements. Figure 1 in the appendix shows the code we wrote for this first part. The way this code works is that initially the counter variable is set to 0 outside the while loop. Then inside the while loop the printf statement prints the words “ITM printf test #” with the value of the counter. Then the counter++ increases the value of counter by 1 and then HAL_Delay makes a 1 ms delay and the while loop happens again. Figure 2. In the appendix shows the results for part 1 which is just printing “ITM printf test #” per line and the test starts at 0 and increase by 1 each line we also showed these results to a TA in the lab.

PART 2 – CONTROL LED BRIGHTNESS VIA SOFTWARE PWM

For the second part we started of my enabling timer 1 and setting the prescaler to 79 and ARR to 65535. Then we started off my writing a separate function for a delay in microseconds. Figure 4 shows the code we wrote for the delay function, the way this delay function works is it sets the timer 1 counter to 0 and checks while that counter is less than the request delay it stays in that while loop and then returns out of the function. Then we wrote code in our loop for the software PWM and LED brightness ramping up and down. Figure 3 shows our code for this, the way this code works is every 1 second it checks if the duty is at 100 or 0, and if it has reached either 100 or 0 it flips the duty rate to either positive or negative, and until it has reached either 100 or 0 it adds the duty rate which is 10 to the current duty. Every while loop cycle the PWM is run and for the current duty the LED is on and for the reset of the PWM the LED is off. We then showed the working LED ramping up and down to the TA in the lab.

PART 3 – DC MOTOR SPEED CONTROL USING PWM AND A ROTARY ENCODER

For part 3 we started off by enabling pins for our rotary encoder, and the timer 2 channel 1. Then we wrote code for the functions as instructed in lab manual, figure 6 shows both these functions, the functions PWMStart starts the PWM for timer 2 and starts to output it on channel 1. PWMAdjust takes in a duty in percentage and converts it to the actual duty and updates it on the timer 2 channel 1. We did try to fix our code from lab 1, so the rotary encoder would check 4 states before making a decision on whether its CW or CCW like instructed in the lab 1 feed back, but due to our encoder being buggy it was not working as expected and we switched back to the lab 1 code and updated it to work with this lab and made it a separate function which is shown in figure 6. Figure 5 shows the code for our while loop, the way this code works is that for every while loop cycle it gets input from the rotary encoder and check if there was a update, only if there was a update it enters the if statement where it updates the duty value by plus or minus 5 and then it send the value to the PWMAjust function, and outputs the duty value to the console as well as updating the LED accordingly. Updating the duty on Timer 2 channel 1 PWM updates the speed of the vibration motor. A video demonstration of the results was recorded and attached the submission of this lab report.

ADDITIONAL QUESTIONS

1. In Part 2, we simulated PWM output using a software implementation. Explain why this approach is not scalable in complex systems. (Hint: Consider the effect of interrupts.)

Answer 1: This is not scalable in complex systems since in a software implementation the CPU keeps track of the on / off, of the timer and having interrupts or multitasking system make cause disturbances and cause inaccurate duty cycles.

2. Perhaps you noticed that when increasing the PWM duty cycle from 0%, the motor does not begin to work immediately and instead requires some minimum duty cycle to be reached before it starts. Describe why you think this occurs.

Answer 2: The transistor causes a voltage drop between the motor and the voltage from the PWM, and at low duty cycles there isn't enough average voltage being supplied to overcome this voltage drop and have enough to power on the motor after the voltage drop, that's why at low duty cycles the motor doesn't turn on.

3. When switching inductive loads through a transistor with a PWM signal, an effect known as back-emf (also called inductive kickback) often destroys semiconductor devices.

3.1. In simple terms, describe what back-emf is and be sure to briefly describe the underlying physics.

Answer 3.1: Back-emf is the voltage generated in an electric motor that is opposing the supply voltage. This occurs when a motors rotating coils cut through the magnetic field producing a current, and voltage which is proportional to the motors speed and goes against the supply voltage. This reduces the motors speed since the net supply voltage is supplied voltage – voltage generated from kickback.

3.2. What simple components can be used to damp this unwanted effect and protect semiconductors.

Answer 3.2: The simplest way to protect against this would be to add a flyback diode across the vibration motor.

4. In Part 3, the use of the timer peripheral allows some flexibility to choose the effective
resolution of the PWM output. Suppose you wanted to approximate a 10-bit DAC using a PWM frequency of approximately 39kHz and have effective analog output bandwidth of 1kHz from your (approximated) DAC.
a. If the clock supplied to the timer is 40MHz, what should the Prescaler and ARR parameters be set to?

Answer 4a: The ARR would be 2^{10} -1 which is 1023.

$$f = \frac{f_{clock}}{(Prescaler + 1)(ARR + 1)}$$

$$39{,}000 = \frac{40{,}000{,}000}{(Prescaler + 1)(1023 + 1)} \Rightarrow Prescaler = 0$$

Prescaler is 0.
b. Describe a suitable low pass filter circuit with component values for R & C.

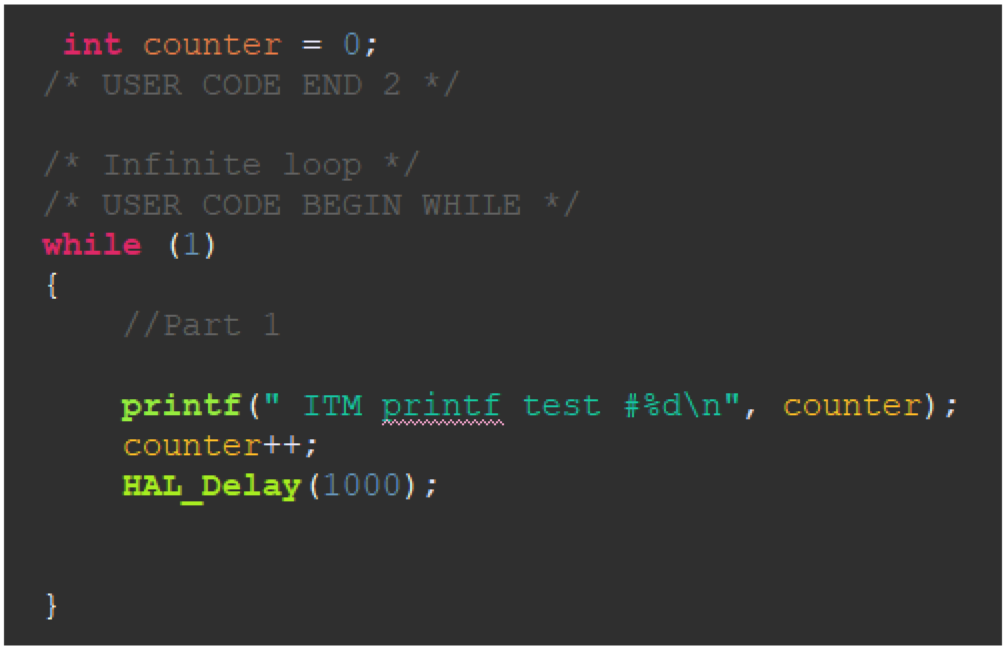
Answer 4b: $f = \frac{1}{2\pi RC}$ if we set R to 1k ohms then

$$1000 = \frac{1}{2\pi(1000)(C)} \Rightarrow C = 1.6 \times 10^{-7}$$

C = 1.6×10^{-7}F
5. Bit Angle Modulation (BAM) is an alternative to PWM for controlling power, look it up and then describe in your own words how this works.

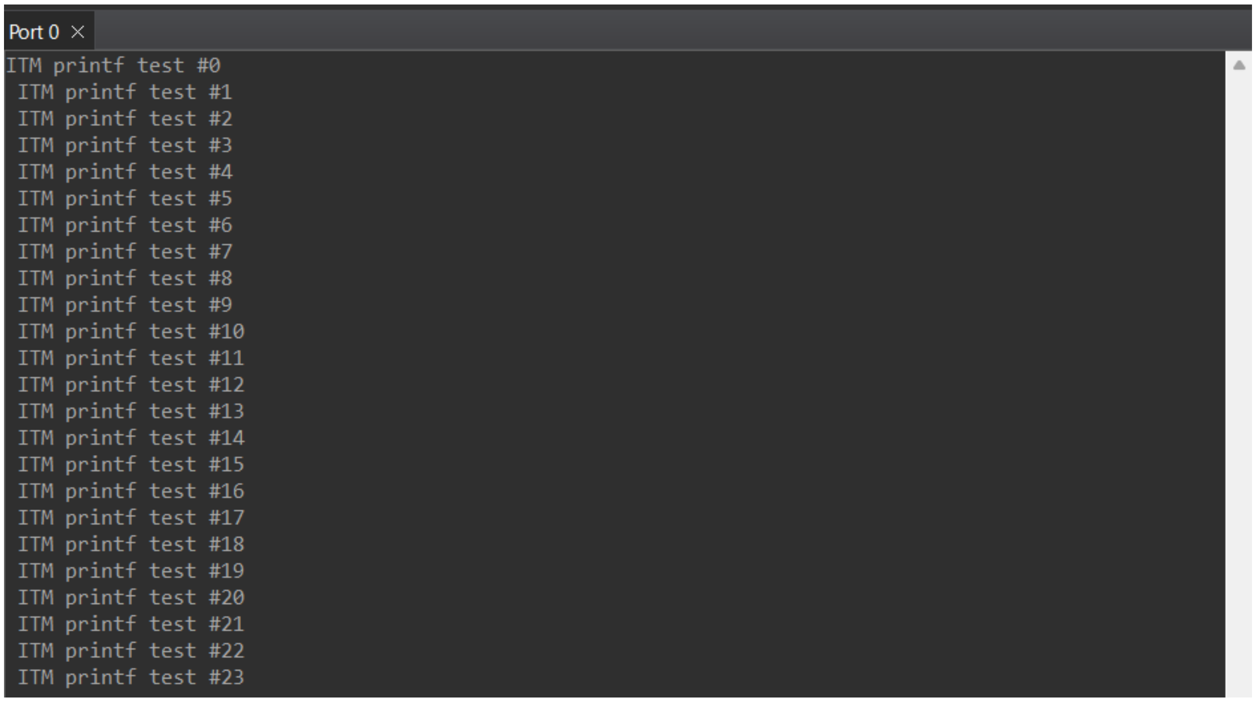
Answer 5: BAM is used to control the brightness od LED by assigning a specific time duration to each bit of a binary number. The way BAM does this is it turns the value into a bit number and dictates a fixed duration that doubles with each bit from least significant bit to most significant bit and the value of the bit decides if the LED is on or off, 1 being on and 0 being off. An example would be the value 13 which has a bit value of 1101, this would make the LED be on for 1 seconds, then off for 2 seconds, then on for 4 seconds and then on for 8 seconds.

Appendix.



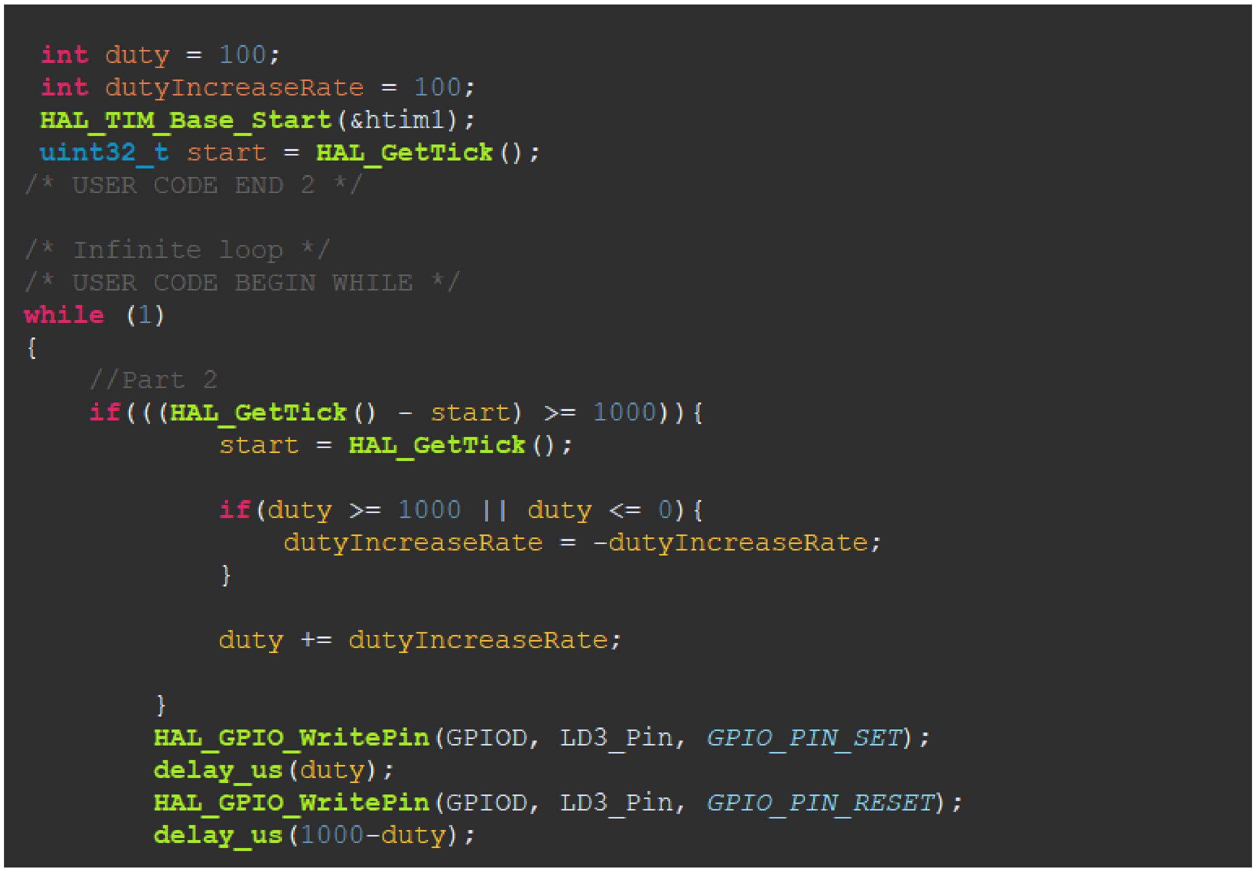
```
 int counter = 0;
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    //Part 1

    printf(" ITM printf test #%d\n", counter);
    counter++;
    HAL_Delay(1000);


}
```

Figure 1. Part 1 code.



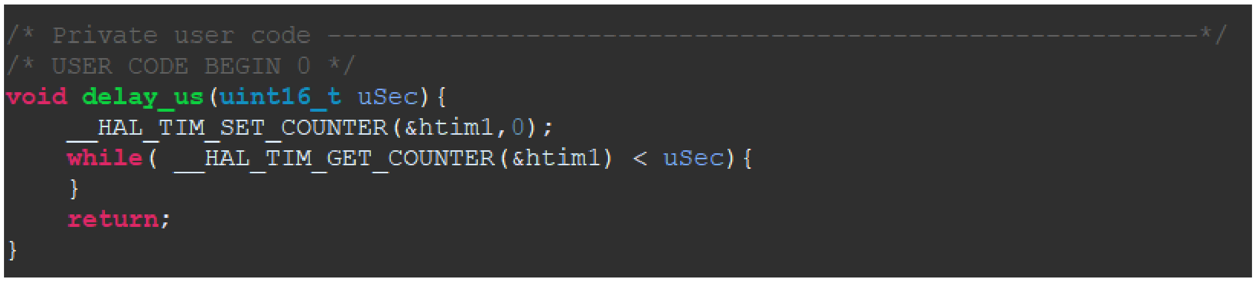
```
ITM printf test #0
 ITM printf test #1
 ITM printf test #2
 ITM printf test #3
 ITM printf test #4
 ITM printf test #5
 ITM printf test #6
 ITM printf test #7
 ITM printf test #8
 ITM printf test #9
 ITM printf test #10
 ITM printf test #11
 ITM printf test #12
 ITM printf test #13
 ITM printf test #14
 ITM printf test #15
 ITM printf test #16
 ITM printf test #17
 ITM printf test #18
 ITM printf test #19
 ITM printf test #20
 ITM printf test #21
 ITM printf test #22
 ITM printf test #23
```

Figure 2. Results for part 1.



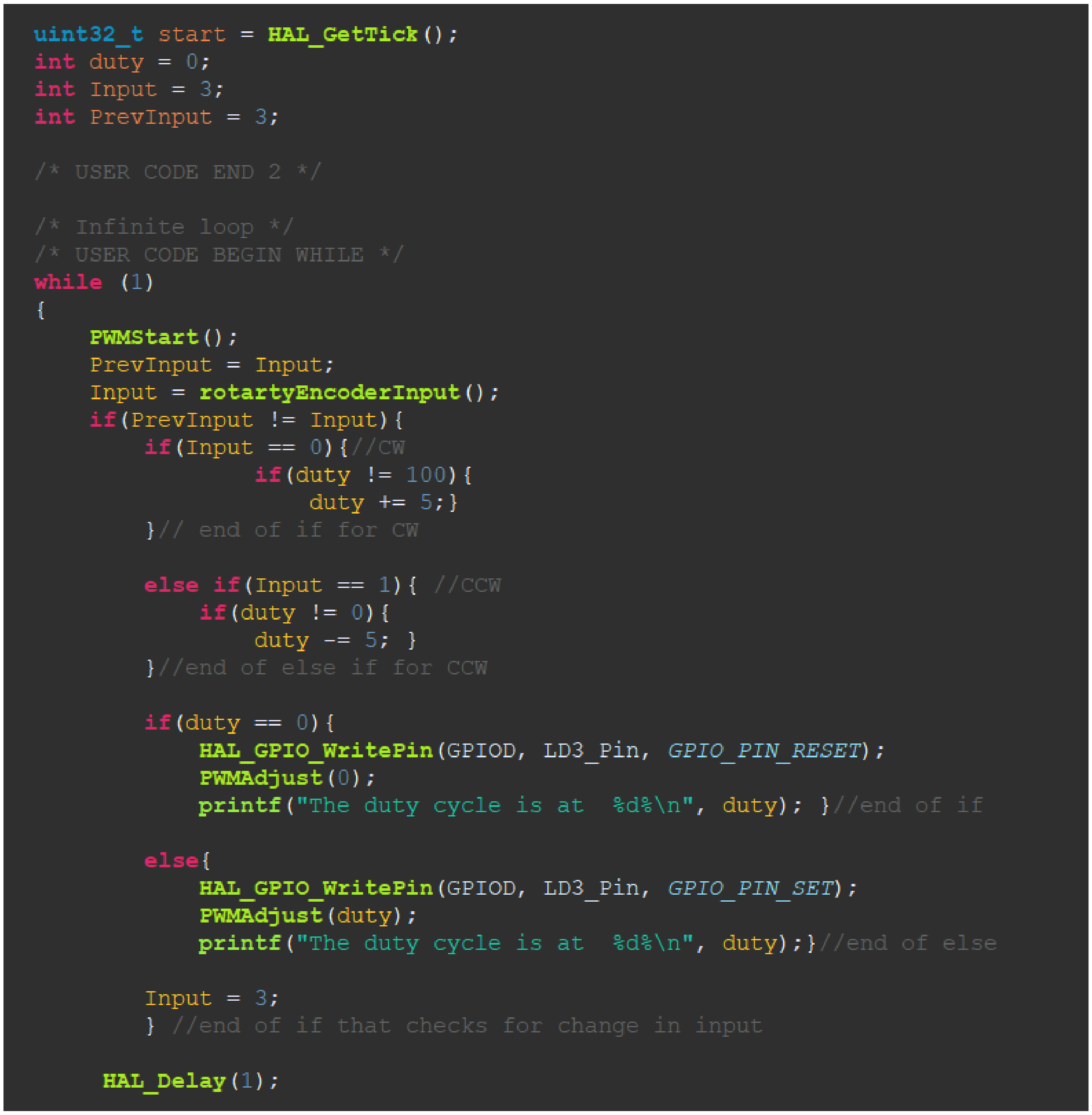
```
  int duty = 100;
  int dutyIncreaseRate = 100;
  HAL_TIM_Base_Start(&htim1);
  uint32_t start = HAL_GetTick();
 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
     //Part 2
     if(((HAL_GetTick() - start) >= 1000)){
             start = HAL_GetTick();

             if(duty >= 1000 || duty <= 0){
                 dutyIncreaseRate = -dutyIncreaseRate;
             }

             duty += dutyIncreaseRate;

         }
         HAL_GPIO_WritePin(GPIOD, LD3_Pin, GPIO_PIN_SET);
         delay_us(duty);
         HAL_GPIO_WritePin(GPIOD, LD3_Pin, GPIO_PIN_RESET);
         delay_us(1000-duty);
```

Figure 3. Part 2 code.



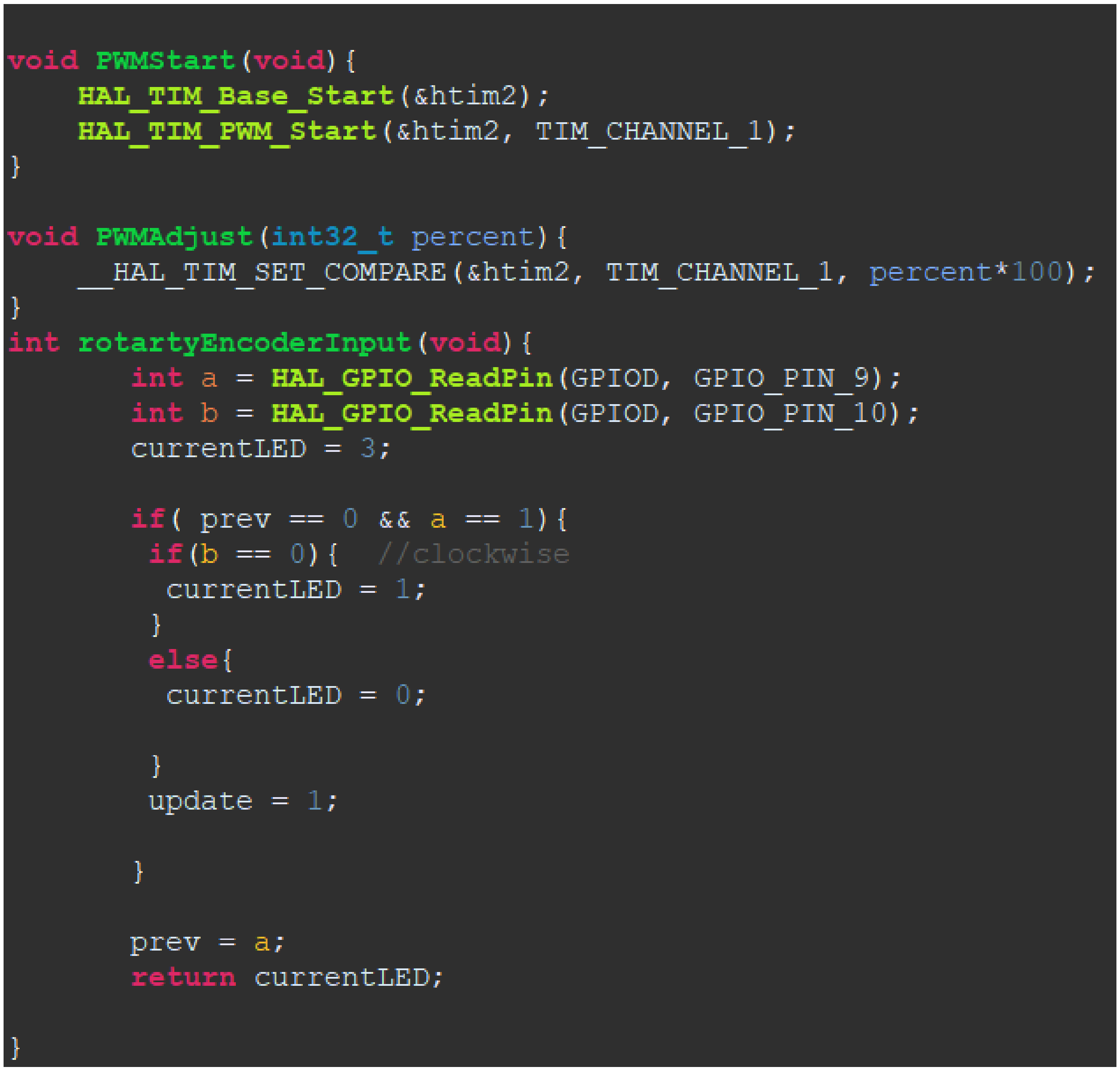
```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void delay_us(uint16_t uSec){
    __HAL_TIM_SET_COUNTER(&htim1,0);
    while( __HAL_TIM_GET_COUNTER(&htim1) < uSec){
    }
    return;
}
```

Figure 4. Part 2 code.



```c
uint32_t start = HAL_GetTick();
int duty = 0;
int Input = 3;
int PrevInput = 3;

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    PWMStart();
    PrevInput = Input;
    Input = rotartyEncoderInput();
    if(PrevInput != Input){
        if(Input == 0){//CW
                if(duty != 100){
                    duty += 5;}
        }// end of if for CW

        else if(Input == 1){ //CCW
            if(duty != 0){
                duty -= 5; }
        }//end of else if for CCW

        if(duty == 0){
            HAL_GPIO_WritePin(GPIOD, LD3_Pin, GPIO_PIN_RESET);
            PWMAdjust(0);
            printf("The duty cycle is at  %d%\n", duty); }//end of if

        else{
            HAL_GPIO_WritePin(GPIOD, LD3_Pin, GPIO_PIN_SET);
            PWMAdjust(duty);
            printf("The duty cycle is at  %d%\n", duty);}//end of else

        Input = 3;
        } //end of if that checks for change in input

     HAL_Delay(1);
```

Figure 5. Part 3 code.



```
void PWMStart(void){
    HAL_TIM_Base_Start(&htim2);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
}

void PWMAdjust(int32_t percent){
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, percent*100);
}
int rotartyEncoderInput(void){
       int a = HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_9);
       int b = HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_10);
       currentLED = 3;

       if( prev == 0 && a == 1){
        if(b == 0){  //clockwise
         currentLED = 1;
        }
        else{
         currentLED = 0;

        }
        update = 1;

       }

       prev = a;
       return currentLED;

}
```

Figure 6. Part 3 code.