Cover Page

Lab 4 Report

Implementation of PWM
**And achievement**
**Light change**
ECE4436, DMU
Spring 2024
**Wankang Zhai**
2232923

## contents

# 1. Overview

In this experiment, we completed the configuration of PWM and successfully completed the practice of breathing lights through PWM configuration. The specific knowledge I mastered in this experiment and the tasks I completed are as follows:

1. Successfully understand the meaning of Duty and complete the setting of light on and off by changing Duty.
2. Successfully pass two while to ensure continuous lighting and off.
3. Have an in-depth understanding of PWM and know the principles of PWM
4. Complete different rate settings by setting Commands

# 2. Description of Tasks and Results

## Task1: Set the initialization of PWM.

### 2.1.1 Set PWM1 PeripheralEnable first

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);

while(!SysCtlPeripheralReady(SYSCTL_PERIPH_PWM1));

while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));
```

Figure 1

In figure 1, we first enable the Peripheral of the PWM1 interface and then wait for it to complete the configuration. This step is the beginning of all steps. For any serial port setting, we should first Enable the Peripheral. At the same time, I also wait for PortF interface, because later I need to use the GPIO port in portF to light up the LED light.

### 2.1.2 Get correct PWM system clock

```
SysCtlPWMClockSet(SYSCTL_PWMDIV_64);
GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);

GPIOPinConfigure(GPIO_PF1_M1PWM5);

float PWM_clock = SysCtlClockGet() / 64;
```

Figure 2

· Set the correct PWM clock and get GPIOPinType. In this experiment, we re-adjusted the central clock and set the corresponding PWM clock, which is sixty-fourth of the corresponding clock. The clock we set has a great relationship with the frequency of the breathing light below us. The duty we set later changes based on the clock.

### 2.1.3 Setting the correct load and PWMGenPeriodSet

·

```
load = PWM_clock / PWM_freq - 1;
PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, load);
PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
width = load * duty;
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, width);

PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);

PWMGenEnable(PWM1_BASE, PWM_GEN_2);
```

Figure 3

In Figure 3，我们设置了 PWM_GEN_2 ，并通过查找 Table 1 设置 PWM_OUT_5. Table 1 在下面给出。然后我们这只了 pulse width。我们通过调整 duty 的大小来改变这个 width。以实现不同频率的呼吸的目的。我们将 duty 和 load width 设置为全局变量，通过在下方循环中改变不同的 duty 来实现不同速率的呼吸。

| Pin Name | Pin Number | Pin Mux / Pin Assignment | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| M0FAULT0 | 30<br>53<br>63 | PF2 (4)<br>PD6 (4)<br>PD2 (4) | I | TTL | Motion Control Module 0 PWM Fault 0. |
| M0PWM0 | 1 | PB6 (4) | O | TTL | Motion Control Module 0 PWM 0. This signal is controlled by Module 0 PWM Generator 0. |
| M0PWM1 | 4 | PB7 (4) | O | TTL | Motion Control Module 0 PWM 1. This signal is controlled by Module 0 PWM Generator 0. |
| M0PWM2 | 58 | PB4 (4) | O | TTL | Motion Control Module 0 PWM 2. This signal is controlled by Module 0 PWM Generator 1. |
| M0PWM3 | 57 | PB5 (4) | O | TTL | Motion Control Module 0 PWM 3. This signal is controlled by Module 0 PWM Generator 1. |
| M0PWM4 | 59 | PE4 (4) | O | TTL | Motion Control Module 0 PWM 4. This signal is controlled by Module 0 PWM Generator 2. |
| M0PWM5 | 60 | PE5 (4) | O | TTL | Motion Control Module 0 PWM 5. This signal is controlled by Module 0 PWM Generator 2. |
| M0PWM6 | 16<br>61 | PC4 (4)<br>PD0 (4) | O | TTL | Motion Control Module 0 PWM 6. This signal is controlled by Module 0 PWM Generator 3. |
| M0PWM7 | 15<br>62 | PC5 (4)<br>PD1 (4) | O | TTL | Motion Control Module 0 PWM 7. This signal is controlled by Module 0 PWM Generator 3. |
| M1FAULT0 | 5 | PF4 (5) | I | TTL | Motion Control Module 1 PWM Fault 0. |
| M1PWM0 | 61 | PD0 (5) | O | TTL | Motion Control Module 1 PWM 0. This signal is controlled by Module 1 PWM Generator 0. |
| M1PWM1 | 62 | PD1 (5) | O | TTL | Motion Control Module 1 PWM 1. This signal is controlled by Module 1 PWM Generator 0. |
| M1PWM2 | 23<br>59 | PA6 (5)<br>PE4 (5) | O | TTL | Motion Control Module 1 PWM 2. This signal is controlled by Module 1 PWM Generator 1. |
| M1PWM3 | 24<br>60 | PA7 (5)<br>PE5 (5) | O | TTL | Motion Control Module 1 PWM 3. This signal is controlled by Module 1 PWM Generator 1. |
| M1PWM4 | 28 | PF0 (5) | O | TTL | Motion Control Module 1 PWM 4. This signal is controlled by Module 1 PWM Generator 2. |
| M1PWM5 | 29 | PF1 (5) | O | TTL | Motion Control Module 1 PWM 5. This signal is controlled by Module 1 PWM Generator 2. |
| M1PWM6 | 30 | PF2 (5) | O | TTL | Motion Control Module 1 PWM 6. This signal is controlled by Module 1 PWM Generator 3. |
| M1PWM7 | 31 | PF3 (5) | O | TTL | Motion Control Module 1 PWM 7. This signal is controlled by Module 1 PWM Generator 3. |

Table 1

M1PWM5 in my using Pin Name. I will write my code according to this Pin name.

### 2.1.4 Setting CTI commands

In this experiment, we also need to use different command lines to control different rates. Therefore, this experiment is the same as Lab4. We also used CTI technology to achieve control on the TivaC terminal. Figure 4 shows the process of our resume Struct.

```c
typedef struct{
    char cmdd[4];
    void(*function)(void);
}lookup;

lookup commands[] = {
        {"FWD",gogogogo},
        {"RRR",mmm},
        {"LLL",ppp},
        {"TTT",ooo},
};
```

Figure 4

Task2: Set the initialization of PWM.

## 2.2.1 Setting the main while loop

```
    while(1)
    {

        for (x= 0; x< sizeof(commands)/sizeof(commands[0]); ++x){
                if (flag ==1){
                    ans = strncmp(cmd, commands[x].cmdd,3);
                    if(ans==0){
                        commands[x].function();
                        while(1)
                        {
                            while(duty<=1)
                            {
                                width = load * duty;
                                PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, width);
                                PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);
                                PWMGenEnable(PWM1_BASE, PWM_GEN_2);
                                duty = duty + step;

                            }
                            while(duty>=0.05)
                            {
                                width = load * duty;
                                PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, width);
                                PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);
                                PWMGenEnable(PWM1_BASE, PWM_GEN_2);
                                duty = duty - step;

                            }
                        }


                        }

                    if(ans==1){
                        err();
                        flag = 0;}//
//                  if(ans!=0){
//                      err();
//                      flag = 0;}
                }
            }
    }
}
```

In the main loop, I set a infinite loop to make it continue work. As you can see the code. I different each loop via duty value. When the value is different, it will go much ligher or much darker. In this way, we change step to make it much convenient to calculate. When duty reach much lower, it went to the next while loop, the duty will be add and be higher. When we can' t find the correct commands, it will jump into err function, which indicates this is not a right commands.

# 3 Overview

In this experiment, we successfully configured the PWM module and understand the meaning of PWM. I can change width via change duty value. So in the end, I successfully realize the breathing light using PWM model of Tiva C.