东南大学自动化学院

实 验 报 告

课程名称：嵌入式及其系统设计

第 2 次实验

实验名称：定时器输出 PWM 波形控制产生呼吸灯

院 （系）：自动化学院 专 业：自动化、机器人工程

姓 名：08023313 朱斌荣，08123127 赵普为，08123113 张博轩 08023214 张韫译萱

实 验 室：无 实验组别：14

同组人员：无 实验时间：2025 年 8 月 29 日

评定成绩： 审阅教师：

目录

1 实验目的和要求

1.1 实验目的

本次实验旨在加深对 STM32 微控制器定时器外设的理解，并掌握其输出脉冲宽度调制（PWM）波形的应用。

1.2 实验要求

利用普中天马 STM32F407 开发板，设计一个呼吸灯程序。呼吸周期的时长控制在 2 秒。

1.3 实验分工

08123113 张博轩 负责搭建工程，配置开发板，处理 GPIO 初始化
08023313 朱斌荣 负责 PWM 功能的初始化、实现与封装
08123127 赵普为 负责实现呼吸灯业务逻辑
08023214 张韫译萱 负责整合代码、联调测试

2 实验原理

2.1 脉冲宽度调制（PWM）原理

脉冲宽度调制（Pulse Width Modulation, PWM）是一种对模拟信号电平进行数字化编码的技术。它通过改变周期性方波信号的占空比（Duty Cycle）来控制输出到负载的平均功率。占空比定义为一个周期内高电平时间与总周期时间的比值。对于 LED 亮度控制等应用，人眼对高频闪烁不敏感，因此可以利用 PWM 技术，通过快速开关 LED，在视觉上产生亮度连续变化的效果。当占空比从 0% 增加到 100% 时，LED 的平均驱动电流随之增加，其表观亮度也相应地从熄灭状态平滑过渡到最亮状态。

2.2 STM32 定时器 PWM 模式

STM32 系列微控制器的通用定时器和高级定时器内置了强大的 PWM 生成功能。其核心是定时器/计数器（CNT）、预分频器（PSC）、自动重装载寄存器（ARR）和捕获/比较寄存器（CCR）。

在 PWM 输出模式下，定时器的工作流程为：

1. **时钟源与预分频** 定时器的时钟源通常来自 APB 总线。该时钟首先经过预分频器（PSC）进行分频，得到计数器（CNT）的时钟频率 `CK_CNT`。计算公式为：`CK_CNT = CK_TIM / (PSC + 1)`。
2. **周期设定** 计数器（CNT）从 0 开始向上计数，直到其值等于自动重装载寄存器（ARR）中设定的值时，计数器复位为 0，并产生一个更新事件（UEV）。这个过程定义了 PWM 波的周期（T）。PWM 的频率 `f_PWM` 由下式决定：`f_PWM = CK_CNT / (ARR + 1)`。

3. **占空比设定** 捕获/比较寄存器（CCR）的值用于确定 PWM 的占空比。在 PWM 模式 1 下，当 `CNT < CCR` 时，输出为有效电平；当 `CNT >= CCR` 时，输出为无效电平。因此，占空比（Duty Cycle）由 `CCR` 和 `ARR` 的比值决定：`Duty Cycle = CCR / (ARR + 1)`。

通过在程序运行中修改 CCR 寄存器的值，我们就可以改变 PWM 的占空比，从而实现对 LED 亮度的控制。

2.3 STM32 HAL 库 PWM 相关配置

本实验采用硬件抽象层（HAL）库来简化外设配置。HAL 库将底层的寄存器操作封装成一系列标准化的 API 函数。配置 PWM 主要涉及以下几个结构体和函数：

- `TIM_HandleTypeDef`：定时器句柄，包含了定时器的所有配置信息和状态。
- `TIM_OC_InitTypeDef`：PWM 通道配置句柄，用于设置 PWM 模式、脉冲值（即 CCR 值）、输出极性等。
- `HAL_TIM_PWM_Init()`: 初始化定时器的基本参数，如预分频值、计数模式和周期。
- `HAL_TIM_PWM_ConfigChannel()`: 配置具体的 PWM 通道参数。
- `HAL_TIM_PWM_Start()`: 启动指定通道的 PWM 信号输出。
- `HAL_TIM_PWM_MspInit()`: 这是一个由 `HAL_TIM_PWM_Init()` 调用的回调函数，用于完成与 MCU 相关的硬件初始化，例如 GPIO 引脚的复用功能配置和时钟使能。

3 实验方案与实验步骤

3.1 硬件方案

本实验直接使用普中天马 STM32F407 开发板上的板载 LED。PWM 输出引脚为 `PF9`，该引脚连接到板载 LED。

3.2 软件方案

软件设计的核心在于初始化 TIM14 产生 PWM 信号，并在主循环中周期性地修改其占空比。

3.2.1 系统与时钟初始化

在主函数 `main()` 的起始部分，首先调用 `HAL_Init()` 对 HAL 库进行初始化。接着，通过 `SystemClock_Init(8, 336, 2, 7)` 将系统主时钟（HCLK）配置为 168MHz。根据 STM32F4 的时钟树，连接到 TIM14 的 APB1 总线时钟频率为 HCLK/4 = 42MHz，而 APB1 定时器时钟则为 APB1 总线时钟的 2 倍，即 84MHz。最后，调用 `SysTick_Init(168)` 初始化系统滴答定时器，为 `delay_ms()` 函数提供精确的延时基准。

3.2.2 PWM 初始化

PWM 的初始化由函数 `TIM14_CH1_PWM_Init(u16 per, u16 psc)` 完成。根据实验要求，我们需要一个较高的 PWM 频率。代码中通过传递参数 `per=500-1` 和 `psc=84-1` 来进行配置。

```c
void TIM14_CH1_PWM_Init(u16 per, u16 psc) {
    TIM14_Handler.Instance = TIM14;                     // 定时器14
    TIM14_Handler.Init.Prescaler = psc;                 // 定时器分频
    TIM14_Handler.Init.CounterMode = TIM_COUNTERMODE_UP; // 向上计数模式
    TIM14_Handler.Init.Period = per;                    // 自动重装载值
    TIM14_Handler.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    HAL_TIM_PWM_Init(&TIM14_Handler);  // 初始化PWM

    TIM14_CH1Handler.OCMode = TIM_OCMODE_PWM1;                                  // 模式选择PWM1
    TIM14_CH1Handler.Pulse = per / 2;                                           // 设置比较值,此值用来确定占空比
    TIM14_CH1Handler.OCPolarity = TIM_OCPOLARITY_LOW;                           // 输出比较极性为低
    HAL_TIM_PWM_ConfigChannel(&TIM14_Handler, &TIM14_CH1Handler, TIM_CHANNEL_1); // 配置TIM14通道1

    HAL_TIM_PWM_Start(&TIM14_Handler, TIM_CHANNEL_1);  // 开启PWM通道1
}

// 定时器底层驱动，时钟使能，引脚配置
// 此函数会被HAL_TIM_PWM_Init()调用
// htim:定时器句柄
void HAL_TIM_PWM_MspInit(TIM_HandleTypeDef *htim) {
    GPIO_InitTypeDef GPIO_Iniure;
    __HAL_RCC_TIM14_CLK_ENABLE();  // 使能定时器14
    __HAL_RCC_GPIOF_CLK_ENABLE();  // 开启GPIOF时钟

    GPIO_Iniure.Pin = GPIO_PIN_9;           // PF9
    GPIO_Iniure.Mode = GPIO_MODE_AF_PP;     // 复用推挽输出
    GPIO_Iniure.Pull = GPIO_PULLUP;         // 上拉
    GPIO_Iniure.Speed = GPIO_SPEED_HIGH;    // 高速
    GPIO_Iniure.Alternate = GPIO_AF9_TIM14; // PF9复用为TIM14_CH1
    HAL_GPIO_Init(GPIOF, &GPIO_Iniure);
}
```

图 1: 源代码：定时器初始化

- 定时器时钟：84 MHz
- 预分频值 `PSC`：83
- 自动重装载值 `ARR`：499

由此可计算出 PWM 信号的频率：

$$f_{PWM} = \frac{84,000,000}{(83+1)\times(499+1)} = \frac{84,000,000}{84\times500} = 2000\,\text{Hz} = 2\,\text{kHz}$$

2kHz 的频率可以保证对人眼来说 LED 亮度变化平滑无闪烁。

在 HAL 库的回调函数 `HAL_TIM_PWM_MspInit()` 中，完成了底层硬件的配置。该函数使能了 TIM14 和 GPIOF 的时钟，并将 `PF9` 引脚配置为复用推挽输出模式（`GPIO_MODE_AF_PP`），其复用功能选择为 `GPIO_AF9_TIM14`，从而将该引脚与 TIM14 的通道 1 输出连接起来。

```c
void LED_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;

    LED1_PORT_RCC_ENABLE;
    LED2_PORT_RCC_ENABLE;

    GPIO_InitStructure.Pin = LED1_PIN;
    GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;  // 推挽输出
    GPIO_InitStructure.Pull = GPIO_PULLUP;          // 上拉
    GPIO_InitStructure.Speed = GPIO_SPEED_HIGH;     // 高速
    HAL_GPIO_Init(LED1_PORT, &GPIO_InitStructure);

    GPIO_InitStructure.Pin = LED2_PIN;
    HAL_GPIO_Init(LED2_PORT, &GPIO_InitStructure);

    HAL_GPIO_WritePin(LED1_PORT, LED1_PIN, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED2_PORT, LED2_PIN, GPIO_PIN_SET);
}
```

图 2: 源代码：GPIO 初始化

3.2.3 主循环与呼吸效果实现

呼吸灯的效果在 `main()` 函数的 `while(1)` 循环中实现。该循环通过改变比较寄存器 `CCR1` 的值来控制占空比，从而控制 LED 亮度。

```c
int main() {
    u16 i;
    u8 fx = 0;

    HAL_Init();                     // 初始化HAL库
    SystemClock_Init(8, 336, 2, 7);  // 设置时钟,168Mhz
    SysTick_Init(168);
    LED_Init();                          // LED初始化
    TIM14_CH1_PWM_Init(500 - 1, 84 - 1);  // 频率是2Kh

    while (1) {
        if (fx == 0) {
            i++;
            if (i == 500) {
                fx = 1;
            }
        } else {
            i--;
            if (i == 0) {
                fx = 0;
            }
        }
        TIM14_SetCompare1(i);

        delay_ms(2);
    }
}
```

图 3: 源代码：呼吸灯逻辑

一个完整的呼吸周期包含一个渐亮过程和一个渐暗过程。

- **渐亮**：变量 `i`（CCR 值）从 0 递增到 499。此过程共 500 步。

- **渐暗**：变量 `i` 从 500 递减到 1。此过程共 500 步。

整个周期总计 1000 步。每一步之间有 2ms 的延时，由 `delay_ms(2)` 实现。因此，一个完整呼吸周期的总时长为：

$$T_{cycle} = 1000\,\text{steps} \times 2\,\text{ms/step} = 2000\,\text{ms} = 2\,\text{s}$$

这精确地满足了实验要求。占空比的更新通过调用 `TIM14_SetCompare1(i)` 函数实现，该函数将值 `i` 写入 TIM14 的比较寄存器 `TIM14->CCR1`。

3.3 实验步骤

1. 在 Keil uVision 5 中建立新工程，选择对应的 STM32F407 型号。引入 HAL 库以及厂商提供的 LCD、KEY、RTC 等外设的驱动文件。配置工程头文件路径，确保所有模块能正确引用。
2. 在 `main.c` 中实现呼吸灯逻辑。
3. 编译整个工程，将生成的目标文件下载到 STM32 开发板中。
4. 按下开发板上的复位按钮，启动程序，并观察 `PF9` 引脚对应的 LED 灯的状态。

4 实验设备与器材配置

1. 普中科技天马 STM32F407 开发板
2. ARM 仿真下载器
3. Keil uVision 5

5 实验记录

程序成功下载到开发板并复位后，观察到位于 `PF9` 引脚的 LED 灯开始呈现周期性的亮度变化。LED 从完全熄灭状态开始，亮度在约 1 秒的时间内平滑、线性地增加到最亮。达到最亮状态后，LED 的亮度随即在约 1 秒的时间内平滑、线性地减弱，直至完全熄灭。上述“亮-暗”变化过程不断循环，形成了稳定的“呼吸”效果。

图 4: LED 灯明亮

图 5: LED 灯暗亮

6 实验总结

本次实验成功地利用 STM32F407 微控制器的 TIM14 定时器实现了预期的 PWM 呼吸灯效果，达到了实验目的。

通过本次实践，我认识到 PWM 技术在嵌入式系统中的重要性。我对微控制器底层外设的工作原理，包括时钟树配置、定时器预分频器（PSC）、自动重装载寄存器（ARR）和捕获/比较寄存器（CCR）的配置和使用有了更具体、更深入的认识。

参考文献

[1] 普中,普中STM32F4xx开发攻略–HAL库版[M],深圳市普中科技有限公司,2020.

[2] 意法半导体,STM32F4xx参考手册（第四版）[M],意法半导体有限公司,2013.

[3] 意法半导体,STM32F405xx,STM32F407xx数据手册[M],意法半导体有限公司,2012.