1 定时器中断实验

1.1 实验目的

STM32F429 的定时器功能十分强大，有 TIME1 和 TIME8 等高级定时器，也有 TIME2~TIME5，TIM9~TIM14 等通用定时器，还有 TIME6 和 TIME7 等基本定时器，总共达 14 个定时器之多。在本章中，我们将使用 TIM3 的定时器中断来控制 DS1 的翻转，在主函数用 DS0 的翻转来提示程序正在运行。本章实验将介绍定时器中断的配置，以及定时器的使用。

1.2 实验原理

STM32F429 的通用定时器包含一个 16 位或 32 位自动重载计数器（CNT），该计数器由可编程预分频器（PSC）驱动。STM32F429 的通用定时器可以被用于：测量输入信号的脉冲长度(输入捕获)或者产生输出波形(输出比较和 PWM)等。使用定时器预分频器和 RCC 时钟控制器预分频器，脉冲长度和波形周期可以在几个微秒到几个毫秒间调整。STM3 的通用 TIMx(TIM2~TIM5 和 TIM9~TIM14)定时器功能包括：16 位/32 位(仅 TIM2 和 TIM5)向上、向下、向上/向下自动装载计数器（TIMx_CNT）、16 位可编程(可以实时修改)预分频器(TIMx_PSC)、4 个独立通道（TIMx_CH1~4，TIM9~TIM14 最多可以用 1 个定时器控制另外一个定时器）的同步电路、如下事件发生时产生中断/DMA（TIM9~TIM14 不支持 DMA）。实验使用定时器产生中断，然后在中断服务函数里面翻转 DS1 上的电平，来指示定时器中断的产生。

1.3 代码描述

timer.c 片段

```c
// 回调函数，定时器中断服务函数调用
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  if (htim == (&TIM3_Handler)) {
    LED1 = !LED1; // 发生中断时 LED1 闪烁
    if (!LED1)
      LED0 = 1; // 并且打断 LED0 的点亮状态
  }
}
```

main.c 片段

```c
int main(void) {
  HAL_Init();                     // 初始化 HAL 库
  Stm32_Clock_Init(360, 25, 2, 8); // 设置时钟,180Mhz
  delay_init(180);                // 初始化延时函数
  uart_init(115200);              // 初始化 USART
  LED_Init();                     // 初始化 LED
  KEY_Init();                     // 初始化按键
  TIM3_Init(9000 - 1, 9000 - 1);
  // 定时器 3 初始化，定时器时钟为 90M，分频系数为 9000-1，
  // 所以定时器 3 的频率为 90M/9000=10K，自动重装载为 9000-1，那么定时器周期就是 900ms
  delay_ms(10);
  while (1) {
    LED0 = !LED0; // LED0 翻转
    delay_ms(500);
  }
}
```

1.4 实验结果

DS0 与 DS1 以不同频率闪烁。当 DS1 亮起时，若 DS0 已亮起，将会强制熄灭 DS0。在可观测的一段时间后，DS0 将再次亮起，表明中断程序结束，系统恢复到中断前的状态。

1.5 心得体会

在本次实验中，我了解了定时器中断的原理，以及如何编写中断服务，使用定时器中断来控制 LED 的亮灭。并且我还学习了调整定时器初始化的自动重装载参数，从而调整定时器中断的触发频率的方法。

定时器中断程序能够在固定的一段时间间隔后执行中断程序，也可以用于系统维护、资源释放，方便进行资源的调度调整。

2 PWM 输出实验

2.1 实验目的

在本次实验中，将介绍如何使用 STM32F429 的 TIM3 产生 PWM 输出，利用 TIM3 的通道 4 来产生 PWM 来控制 DS0 的亮度。

2.2 实验原理

脉冲宽度调制（PWM），是英文 `Pulse Width Modulation` 的缩写，简称脉宽调制，是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术。简单一点，就是对脉冲宽度的控制，PWM 原理如图所示：

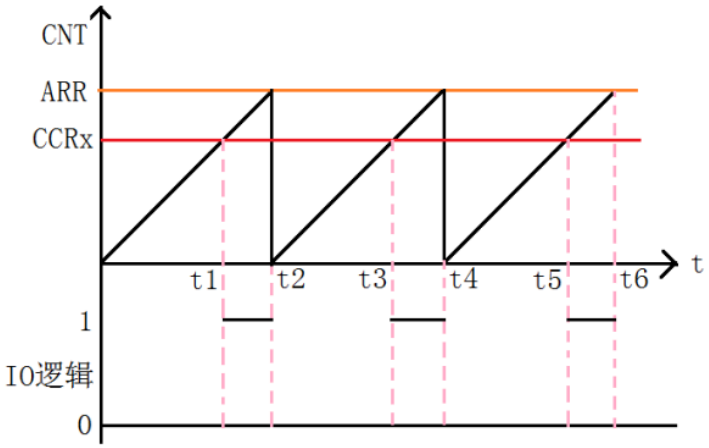

图 **1** **PWM** 原理示意图

图中，我们假定定时器工作在向上计数 PWM 模式，且当 CNT<CCRx 时，输出 0，当 CNT>=CCRx 时输出 1。那么就可以得到如上的 PWM 示意图：当 CNT 值小于 CCRx 的时候，IO 输出低电平(0)，当 CNT 值大于等于 CCRx 的时候，IO 输出高电平(1)，当 CNT 达到 ARR 值的时候，重新归零，然后重新向上计数，依次循环。改变 CCRx 的值，就可以改变 PWM 输出的占空比，改变 ARR 的值，就可以改变 PWM 输出的频率，这就是 PWM 输出的原理。

2.3 代码描述

main.c 片段

```c
int main(void) {
  u16 led0pwmval = 0, maxn = 100;
  u8 dir = 1, delay = 10, key;
  HAL_Init();                          // 初始化 HAL 库
  Stm32_Clock_Init(360, 25, 2, 8); // 设置时钟,180Mhz
  delay_init(180);                     // 初始化延时函数
  uart_init(115200);
  KEY_Init();
  LED_Init(); // 初始化 LED
  TIM3_PWM_Init(500 - 1, 90 - 1);
  // 90M/90=1M 的计数频率，自动重装载为 500，那么 PWM 频率为 1M/500=2kHZ
  while (1) {
    key = KEY_Scan(0); // 调整 delay 的值，从而控制闪烁频率
    switch (key) {
    case WKUP_PRES:
      delay = 10;
      break;
    case KEY2_PRES:
      delay = 5;
      break;
    case KEY1_PRES:
      delay = 2.5;
      break;
    case KEY0_PRES:
      delay = 1.25;
      break;
    }
    delay_ms(delay);
    if (dir)
      led0pwmval++; // dir==1 led0pwmval 递增
    else
      led0pwmval--; // dir==0 led0pwmval 递减
    if (led0pwmval > maxn)
      dir = 0; // led0pwmval 到达 maxn 后，方向为递减
    if (led0pwmval == 0)
      dir = 1; // led0pwmval 递减到 0 后，方向改为递增
    TIM_SetTIM3Compare4(led0pwmval); // 修改比较值，修改占空比
  }
}
```

2.4 实验结果

观察到DS0通过PWM调整占空比实现亮度渐变的闪烁，并且可以通过四个按键来控制其不同的闪烁频率。

下图是CH4通道的电平输出波形的瞬时值。实际上。此矩形波的占空比会不断改变，并且随着不同按键的按下，占空比的改变速率也会发生变化。

图 **2**　**PWM** 输出

2.5 心得体会

本次实验最开始是想做同时控制 DS0 和 DS1 随着占空比的变化而改变亮度，并让两个 PWM 频率通过不同的通道输出在示波器上。但是这涉及到比较复杂的底层代码修改。调试了很久，都无法达到预期效果，才放弃了这个想法，选择了按键控制频率的改进。按键控制频率只需要修改主程序代码，相对来说比较简单。