上海电力大学

DSP 原理与应用实验报告

实验名称：CPU 定时器 0 中断实验

专业班级：集成电路设计与集成系统

姓　　名：

学　　号：

2024 年 1 月 6 日

一、实验内容

利用 CPU 定时器 0 的周期中断来控制 LED 灯的闪烁。CPU 定时器 0 每次定时时间到，实现指示灯的定时闪烁，这种方法比软件循环延时更能实现精确的定时。

二、实验原理

1. F2812 的 CPU 定时器

X281x 芯片内部具有 3 个 32 位的 CPU 定时器 – Timer0, Timer1 和 Timer2。其中，CPU 定时器 1 和 2 被系统保留，用于实时操作系统，例如 DSP BIOS；只有 CPU 定时器 0 可以供用户使用。

2. F2812 定时器的控制方法

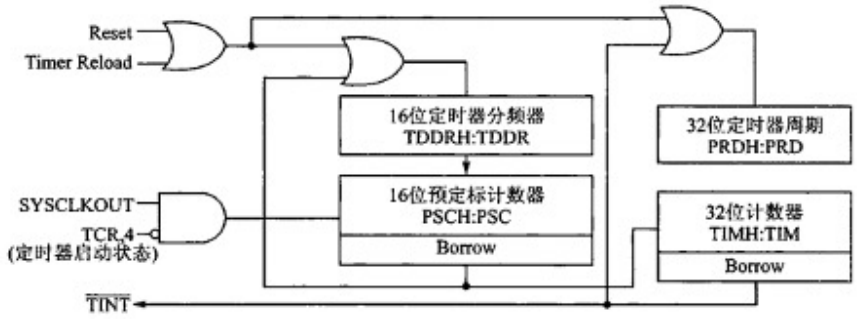

图 1: CPU 定时器内部结构

从图 1 可以看到 CPU 定时器的几个寄存器：32 位的定时器周期寄存器 PRDH:PRD, 32 位的计数器寄存器 TIMH:TIM, 16 位的定时器分频器寄存器 TD-DRH:TDDR, 16 位的预定标计数器寄存器 PSCH:PSC。这里第 1 次遇到“XH:X”形式表示存器的方式。因为 X28IxDSP 的寄存器都是 16 位的，但是 CPU 定时器是 32 位的，，例如定时器周期寄存器、定时器计数器寄存器。可以用 2 个 16 位的寄存器 XH 和又来表示 32 位的寄存器，其中 XH 表示高 16 位，而 X 表示低 16 位。

3. F2812 的中断结构和对中断的处理流程

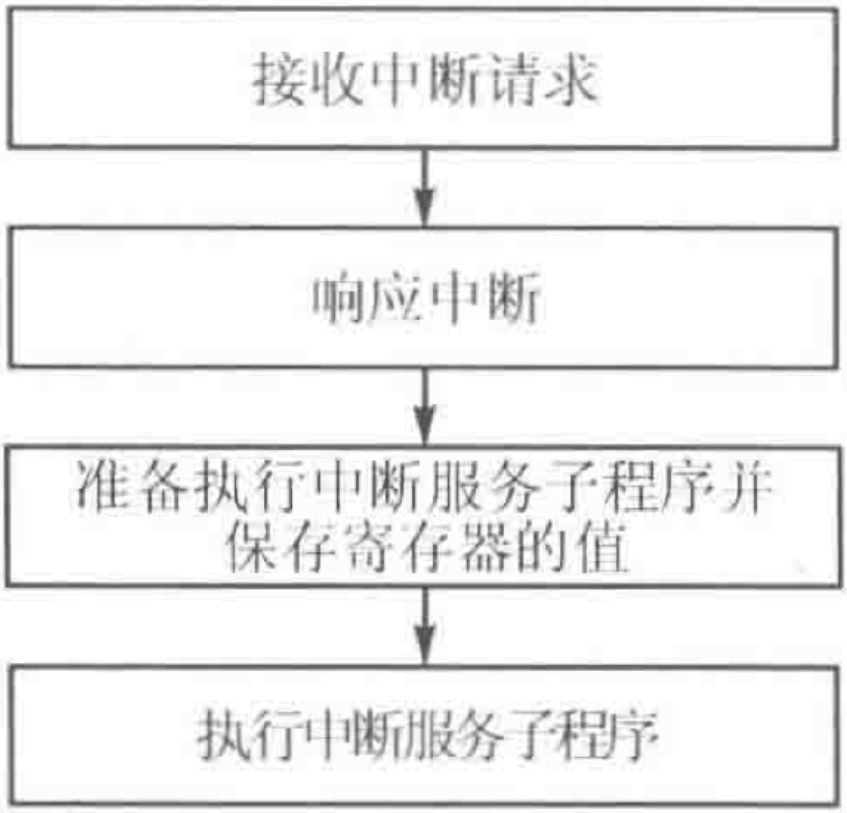

图 2: CPU 处理中断的 4 个步骤

X2812 的中断主要有两种方式触发：一种是在软件中写指令，例如 INTR、OR IFR 或者 TRAP 指令；另一种是硬件方式触发，例如来自片内外设或者外围设备的中断信号，表示某个事件已经发生。无论是软件中断还是硬件中断，都可以归结为可屏蔽中断和不可屏蔽中断。

所谓可屏蔽中断就是这些中断可以用软件加以屏蔽或者解除屏蔽。X2812 片内外设所产生的中断都是可屏蔽中断，每一个中断都可以通过相应寄存器的中断使能位来禁止或者使能该中断。

不可屏蔽中断就是这些中断是不可以被屏蔽的，一旦中断申请信号发出，CPU 必须无条件的立即去响应该中断并执行相应的中断服务子程序。X281x 的不可屏蔽中断主要包括软件中断（INTR 指令和 TRAP 指令等）、硬件中断 NMI、非法指令陷阱以及硬件复位中断。

X2812 的 CPU 按照图 2 所示的 4 个步骤来处理中断。首先由外设或者其他方式向 CPU 提出中断请求，然后如果这个中断是可屏蔽中断，CPU 便会去检查这个中断的使能情况，再决定是否响应该中断；如果这个中断是不可屏蔽中断，则 CPU 便会立即响应该中断。接着，CPU 会完整地执行完当前指令，为了记住当前主程序的状态，CPU 必须要做一些准备工作，例如将 ST0、T、AH、AL、PC 等寄存器的内容保存到堆栈中，以便自动保存主程序的大部分内容。在准备工作做完之后，CPU 就取回中断向量，开始执行中断服务子程序。当然，处理完相应的中断事件之后，CPU 就回到原来主程序暂停的地方，恢复各个寄存器的内容，继续执行主程序。

三、实验步骤

1. 编写代码

修改 main.c

```c
#include "My_Main_Declaration.h"

void main(void)
{
    InitSysCtrl(); // 初始化系统函数
    DINT;
    IER = 0x0000;
    IFR = 0x0000;
    InitPieCtrl();                          // 初始化 PIE 控制寄存器
    InitPieVectTable();                     // 初始化 PIE 中断向量表
    InitPeripherals();                      // 初始化 CPU 定时器模块
    Init_Gpio();                            // 初始化 GPIO
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;      // 使能 PIE 模块中的 CPU 定时器 0 的中断
    IER |= M_INT1;                          // 开 CPU 中断 1
    EINT;                                   // 使能全局中断
    ERTM;                                   // 使能实时中断
    ConfigCpuTimer(&CpuTimer0, 150, 1000000); // 配置 CPU 定时器 0 的周期为 1s
    StartCpuTimer0();                       // 启动 CPU 定时器 0
    for (;;)
    {
    }
}
```

修改 common_files/DSP2812_resources/DSP281x_DefaultIsr.c

```c
// 仅修改部分

// INT1.7
interrupt void TINT0_ISR(void) // CPU-Timer 0
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code

  CpuTimer0.InterruptCount++;
  if (CpuTimer0.InterruptCount == 1)
  {
    LED1_ON;
    LED4_OFF;
  }
  if (CpuTimer0.InterruptCount == 2)
```

```
  {
    LED1_OFF;
    LED2_ON;
  }
  if (CpuTimer0.InterruptCount == 3)
  {
    LED2_OFF;
    LED3_ON;
  }
  if (CpuTimer0.InterruptCount == 4)
  {
    LED3_OFF;
    LED4_ON;
    CpuTimer0.InterruptCount = 0;
  }

  CpuTimer0Regs.TCR.bit.TIF = 1;    // 清除定时器中断标志位
  PieCtrlRegs.PIEACK.bit.ACK1 = 1; // 响应同组其他中断
  EINT;                             // 开全局中断
}
```

为了简化开发过程，可以在 lab4 项目的基础上进行修改。但需要注意的是，请删除 lab4 项目中的 num.h 文件，否则会导致意想不到编译错误。

2. 下载运行、实验现象

编译下载运行，实验现象如下，可以看到 LED 灯每隔一秒闪烁一次，循环运行。

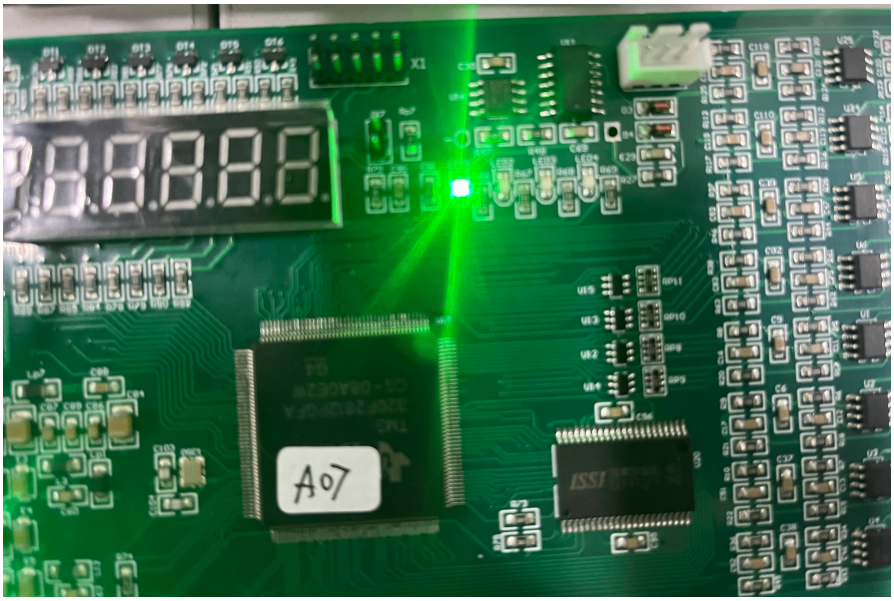

图 3: 实验现象 1

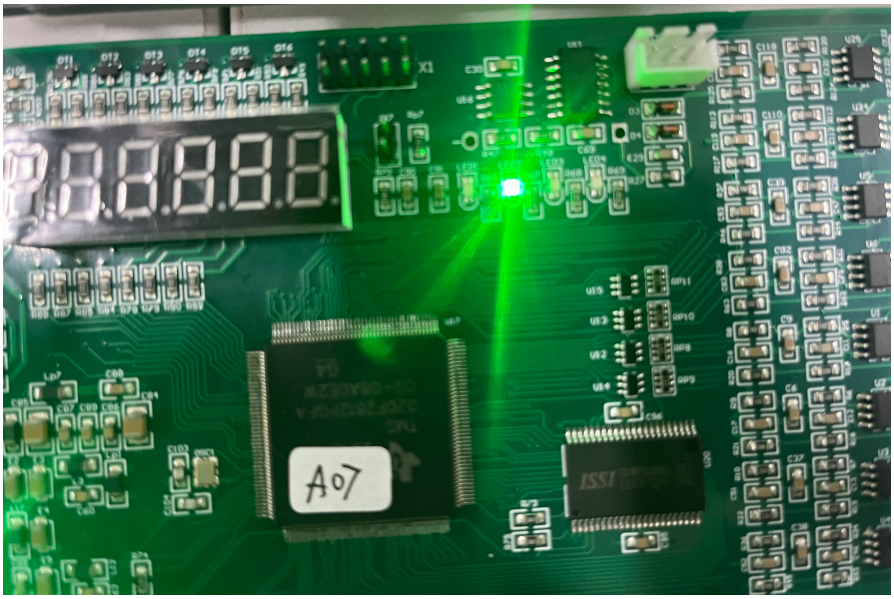

图 4: 实验现象 2

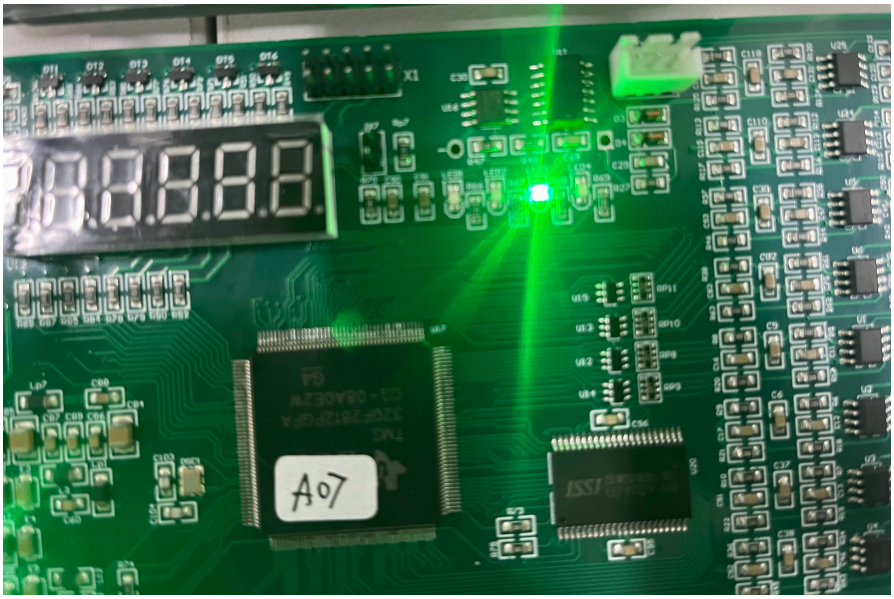

图 5: 实验现象 3

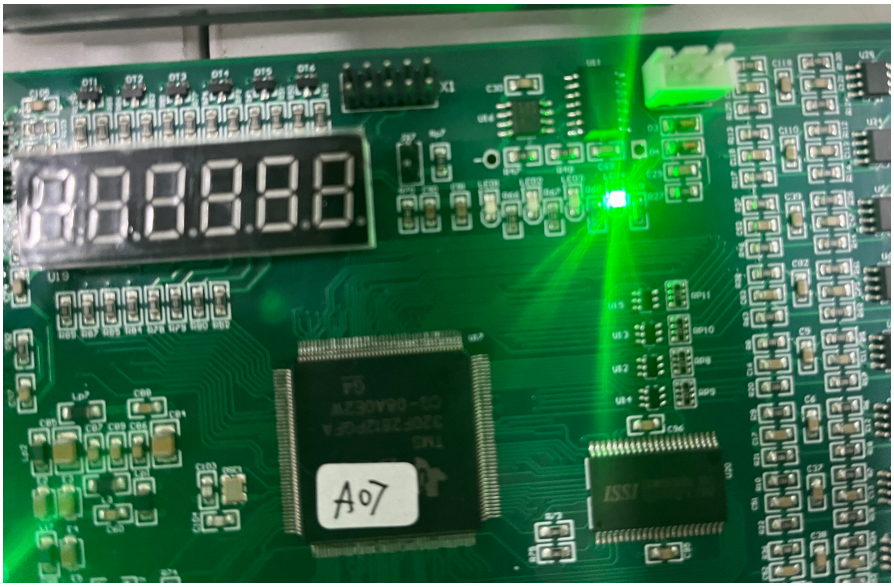

图 6: 实验现象 4

四、实验小结

本实验的目的是利用 CPU 定时器 0 的周期中断来控制 LED 灯的闪烁，实验的原理是通过设置 CPU 定时器的寄存器和中断向量来实现定时中断的响应和处理，实验的步骤是编写相应的 C 代码，修改 main.c 和 DSP281x_DefaultIsr.c 文件，实验的现象是观察 LED 灯每隔一秒闪烁一次，循环运行。