3 定时器中断实验

3.1. 背景介绍

定时器是一个非常重要的外围设备，它可以作为系统的周期性时钟源(Tick)，也可以作为一个计时器，测定事件发生的时间，还可以对外输出周期性脉冲或作为一条监管系统正常运行的“看门狗”(Watchdog)。

本章实验将定时器作为一个计时器，当计时时间到达之后，去改变 LED 灯的状态。在使用定时器之前，我们先来了解一下 CM3 拥有的定时器及其寄存器映射。

3.1.1. Timer 介绍

CM3 中有两个通用定时器：Timer1 和 Timer0，都是 32 位递减计数器，具有以下几种特性：

1. 当计数值达到 0 的时候，产生中断请求信号，中断请求会一直保持，直到写入 INTCLEAR 寄存器来清除。
2. 可以使用外部输入信号 EXTIN 的 0 到 1（上升沿）的转换来启用定时器。
3. 当 APB 定时器的计数值达到 0 时，软件清除前一个中断状态，中断状态将被设置为 1。
4. 外部时钟 EXTIN 必须比外围时钟的一半要慢，因为它是通过双触发器进行采样的。
5. 拥有组件 ID 和外围 ID 寄存器。
6. 外围 ID 寄存器 3 中的 eco 信号与 ECOREVNUM 输入信号相连。

3.1.2. Timer 寄存器映射

下表 3- 1 中显示了定时器的内存映射，需要注意的是，寄存器地址=基地址+基址偏移量，Timer0 的基地址为 0x4000_0000，Timer1 的基地址为 0x4000_1000。

表 3- 1 定时器寄存器映射表

寄存器名称	基址偏移量	类型	宽度	重置值	描述
CTRL	0x000	RW	4	0x0	[3]Timer interrupt enable [2]Select external input as clock [1]Select external input as enable [0] enable
VALUE	0x004	RW	32	0x00000000	[31:0]Current value
RELOAD	0x008	RW	32	0x00000000	[31:0]Reload value.A write to this register sets

					the current value after it reaches 0.
INTSTATUS、INTCLEAR	0x00C	RW	1	0x0	[0]Timer interrupt.Write one to clear.
PID4	0xFD0	RO	8	0x04	Peripheral ID Register 4: value=8'h4
PID5	0xFD4	RO	8	0x00	Peripheral ID Register 5: value=8'h0
PID6	0xFD8	RO	8	0x00	Peripheral ID Register 6: value=8'h0
PID7	0xFDC	RO	8	0x00	Peripheral ID Register 7: value=8'h0
PID0	0xFE0	RO	8	0x22	Peripheral ID Register 0: value=8'h22
PID1	0xFE4	RO	8	0xB8	Peripheral ID Register 1: value=8'hb8
PID2	0xFE8	RO	8	0x1B	Peripheral ID Register 2: value=8'h1b
PID3	0xFEC	RO	8	0x00	Peripheral ID Register 3: value=8'h0
CID0	0xFF0	RO	8	0x0D	Component ID Register 0: value=8'hd
CID1	0xFF4	RO	8	0xF0	Component ID Register 1: value=8'hf0
CID2	0xFF8	RO	8	0x05	Component ID Register 2: value=8'h5
CID3	0xFFC	RO	8	0xB1	Component ID Register 3: value=8'hb1

3.1.3. Dual Timer

Dual Timer 是挂在 APB 总线上的外设，有如下特性：

1. 可配置 16 位或 32 位计数器，采用向下计数的方式；
2. 支持 1/16/256 分频；
3. 每个定时器有独立的使能信号；
4. 三种定时模式：单次计时（one-shot），周期计时（periodic），自由运行模式（free-running）；
5. 拥有周期装载寄存器（period load register）和后台装载寄存器（background load register）。

3.1.4. Dual Timer 特性

1. Dual Timer 有三种计时模式：

自由运行模式：从最大值向下计数，每计到 0 就产生一次中断，不断循环。

周期计时模式：从重装载值向下计数，每计到 0 就产生一次中断，不断循环。重装载值可在定时器初始化时进行配置。如果在 counter 正常计数时配置重装载值，那么 counter 会立即从重装载值重新向下计数。

单次计时模式：计数器只产生一次中断，计数器向下计到 0 时就会停止。

2. 16bit 或 32bit 的 counter 计数器
3. 定时器分频：支持 1 分频、16 分频和 256 分频。一般微秒、毫秒、秒级的延时，完全可以使用 1 分频作为分频因子，且更加准确。举个例子，如果使用的系统时钟是 50MHz，希望产生 1 秒的时间间隔，只需要将分频模式设为 1 分频，装载值设为 50M 就可以了。
4. 装载值和当前值：定时器采用的是向下计数的方式，因此配置装载值后，计时器从装载值向下计数直至计到 0 并产生中断。在周期定时模式下，计到 0 之

后会重新从装载值向下计数并循环往复。对装载寄存器 Load Register 写值时，能够直接将向下计数的 counter 重设到新装载值，也就是说如果上次计数还没有计到 0，这一修改就使得上次计数还没有进入中断就重新计数。如果对后台装载寄存器 Background Load Register 写值，则不会立即修改 counter，而是 counter 向下计数至 0 后开始下一次计数时，从后台装载寄存器的值开始计时。

3.1.5. Dual Timer 寄存器映射

下表 3- 2 显示了双输入定时器的寄存器映射。请注意，寄存器地址=基地址+地址偏移，双输入定时器基地址为 0x4000_2000。

表 3- 2 Dual Timer 寄存器映射

寄存器名称	地址偏移量	类型	位宽	重置值	描述
TIMER1LOAD	0x00	RW	32	0x00000000	Timer1 的装载寄存器，向这一寄存器中写入装载值。
TIMER1VALUE	0x04	RO	32	0xFFFFFFFF	Timer1 的当前值寄存器，可以从这一寄存器中将 counter 当前的值读出来。
TIMER1CONTROL	0x08	RW	8	0x20	Timer1 的控制寄存器，用于初始化过程中的定时器配置。
TIMER1INTCLR	0x0C	WO	-	-	Timer1 的中断清除寄存器，向这一寄存器中写入任何值都将清除中断。
TIMER1RIS	0x10	RO	1	0x0	Timer1 的原始中断寄存器，未经过屏蔽的原始中断
TIMER1MIS	0x14	RO	1	0x0	Timer1 的中断状态寄存器，判断中断是否产生
TIMER1BGLOAD	0x18	RW	32	0x00000000	Timer1 的后台装载寄存器，区别于装载寄存器，向这一寄存器写装载值，在 counter 下次向下计数时进行装载，而不会立即打断本次计数。
TIMER2LOAD	0x20	RW	32	0x00000000	Timer2 的装载寄存器，向这一寄存器中写入装载值。
TIMER2VALUE	0x24	RO	32	0xFFFFFFFF	Timer2 的当前值寄存器，可以从这一寄存器中将 counter 当前的值读出来。
TIMER2CONTROL	0x28	RW	8	0x20	Timer2 的控制寄存器，用于初始化过程中的定时器配置。
TIMER2INTCLR	0x2C	WO	-	-	Timer2 的中断清除寄存器，向这一寄存器中写入任何值都将清除中断。
TIMER2RIS	0x30	RO	1	0x0	Timer2 的原始中断寄存器，未经过屏蔽的原始中断
TIMER2MIS	0x34	RO	1	0x0	Timer2 的中断状态寄存器，判断中断是否产生
TIMER2BGLOAD	0x38	RW	32	0x00000000	Timer2 的后台装载寄存器，区别于装载寄存器，向这一寄存器写装载值，在 counter 下

					次向下计数时进行装载，而不会立即打断本次计数。
TIMERITCR	0xF00	RW	1	0x0	启用集成测试模式，在这种模式下，集成测试输出寄存器直接控制掩码中断输出。
TIMERITOP	0xF04	WO	2	0x0	在集成测试模式下，该寄存器中的值直接启用中断输出。
TIMERPERIPHID4	0xFD0	RO	8	0x04	Peripheral ID Register 4: value = 8'h4
TIMERPERIPHID5	0xFD4	RO	8	0x00	Peripheral ID Register 5: value = 8'h0
TIMERPERIPHID6	0xFD8	RO	8	0x00	Peripheral ID Register 6: value = 8'h0
TIMERPERIPHID7	0xFDC	RO	8	0x00	Peripheral ID Register 7: value = 8'h0
TIMERPERIPHID0	0xFE0	RO	8	0x23	Peripheral ID Register 0: value = 8'h23
TIMERPERIPHID1	0xFE4	RO	8	0xB8	Peripheral ID Register 1: value = 8'hb8
TIMERPERIPHID2	0xFE8	RO	8	0x1B	Peripheral ID Register 2: value = 8'h1b
TIMERPERIPHID3	0xFEC	RO	8	0x00	Peripheral ID Register 3: value = 8'h0
TIMERPCELLID0	0xFF0	RO	8	0x0D	Prime Cell ID Register0: value = 8'h0d
TIMERPCELLID1	0xFF4	RO	8	0xF0	Prime Cell ID Register1: value = 8'hf0
TIMERPCELLID2	0xFF8	RO	8	0x05	Prime Cell ID Register2: value = 8'h05
TIMERPCELLID3	0xFFC	RO	8	0xB1	Prime Cell ID Register3: value = 8'hb1

3.1.5.1. 控制寄存器

配置三种定时模式的寄存器为控制寄存器，下表 3- 3 对于控制寄存器的每一位做出了介绍。

表 3- 3 控制寄存器描述表

Bits	名称	描述
[31:8]	-	保留，读为 0
[7]	Timer Enable 定时器使能	0：屏蔽；1：使能
[6]	Timer Mode 定时器模式	0：free-running 自由运行模式；1：periodic 周期定时模式
[5]	Interrupt Enable 中断使能	0：中断屏蔽；1：中断使能
[4]	保留位	保留，不修改，读时忽略
[3:2]	TimerPre 定时器预分频	00：1 分频；01：16 分频；10：256 分频
[1]	Timer Size 定时器的大小	0：16-bit 宽度计数器；1：32-bit 宽度计数器
[0]	One-shot Count 单次计数模式	0：循环模式；1：单次计数模式

3.2. 实验介绍

通过定时器中断的方式控制 LED 灯的闪烁，其中 Timer0 和 Timer1 控制 LED0 和 LED2 的闪烁，Dual Timer 控制 LED1 和 LED3 的闪烁。

3.3. 建立 HQFPGA 工程

本次实验将不再重新建立工程，而是对已有的工程进行修改，修改方式参考实验二中 2.3 节的内容。

3.4. 物理管脚约束

本章实验无需添加新的管脚约束，具体的管脚分配参考实验二。

3.5. 编译设计

点击“全部运行”按钮，对设计进行全编译并生成 bin 文件。操作步骤如下图 3. 1 所示。

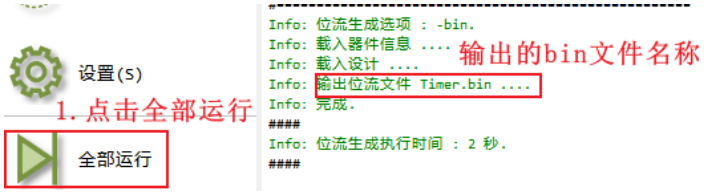

图 3. 1 编译运行整个文件

3.6. 建立 MDK 工程

重新建立一个新的 MDK 工程非常复杂，我们可以通过对已有的 MDK 工程进行修改，从而减少建立工程所需的时间，修改步骤参考实验 2 中 2.8 节的内容。工程建立完成之后，根据实验需求添加所需要的库文件。

3.6.1. 添加实验所需库文件

本次实验需要使用到官方提供的定时器库函数“CM3DS_timer”和“CM3DS_dualtimer”，首先将对应的库文件复制至 CMSIS 文件夹的 StdPeriph_Drive 文件夹之下，然后打开工程，在工程中进行添加，添加步骤如下图 3. 2 所示。

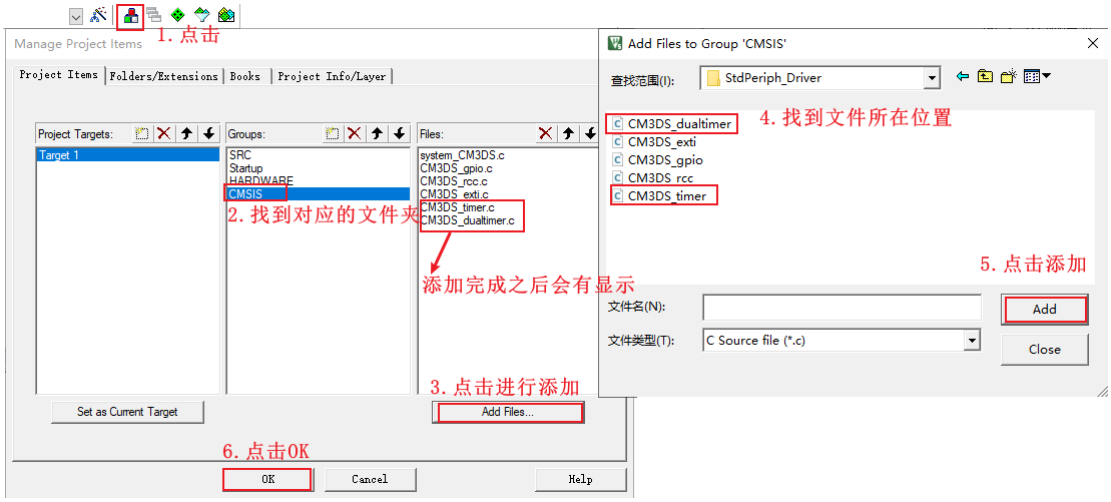

图 3. 2 添加所需的库文件

3.6.2. 调试软件设置

打开工程之后，还需要对调试软件设置，否则将无法下载程序，操作步骤参考 1.7.3 节中的内容。

3.7. 软件设计

3.7.1. Timer 库

对于定时器的控制，使用到的函数都在“CM3DS_timer.c”这个文件当中，下面对于该库中的函数做一下简单的介绍。

3.7.1.1. TIM_DeInit

将 TIM_DeInit 的外设时钟进行设置，该函数只有一个参数可以设置：CM3DS_MPS2_TIMERx，因为一共只有两个定时器，故可以设置为 CM3DS_MPS2_TIMER0 或者 CM3DS_MPS2_TIMER1。

函数内容：根据输入参数的不同，分别对定时器 0 和定时器 1 的复位进行设置，需要操作的寄存器是 REG_PERI_RST，该寄存器的基地址为 0x4000_3000，偏移地址位 0x00，对于寄存器中的每一位的说明如下表 3- 4 所示。

表 3- 4 REG_PERI_RST 寄存器说明表

Bits	名称	描述
[31:8]	-	Read as 0
[9]	SSP1_RSTn	SSP1 soft reset bit: 0: Reset SSP1 module;1: Release SSP1 module
[8]	SSP0_RSTn	SSP0 soft reset bit: 0: Reset SSP0 module;1: Release SSP0 module
[7]	Wdog_RSTn	Watchdog soft reset bit: 0: Reset Watchdog module;1: Release Watchdog module
[6]	ADC_RSTn	ADC soft reset bit: 0: Reset ADC module;1: Release ADC module
[5]	I2C_RSTn	I2C soft reset bit: 0: Reset I2C module;1: Release I2C module
[4]	UART1_RSTn	UART1 soft reset bit: 0: Reset UART1 module;1: Release UART1 module
[3]	UART0_RSTn	UART0 soft reset bit: 0: Reset UART0 module;1: Release UART0 module
[2]	Dtimer_RSTn	Dual-timer soft reset bit: 0: Reset Dual-timer module;1: Release Dual-timer module
[1]	Timer1_RSTn	Timer1 soft reset bit: 0:Reset Timer1 module;1:Release Timer1 module
[0]	Timer0_RSTn	Timer0 soft reset bit: 0:Reset Timer0 module;1:Release Timer0 module

由上表可以看出，REG_PERI_RST 寄存器的第 0 位和第 1 位分别控制着定时器 0 和定时器 1。通过 RCC_APB1PeriphResetCmd 函数向对应位写 0 和 1，对定时器实现复位和释放的功能。TIM_DeInit 函数的部分代码如下所示：

```
if (CM3DS_MPS2_TIMERx == CM3DS_MPS2_TIMER0)
{
    RCC_APB1PeriphResetCmd( RCC_APB1Periph_TIM0, ENABLE);
```

```
    RCC_APB1PeriphResetCmd( RCC_APB1Periph_TIM0, DISABLE);
} else if (CM3DS_MPS2_TIMERx == CM3DS_MPS2_TIMER1) {
    RCC_APB1PeriphResetCmd( RCC_APB1Periph_TIM1, ENABLE);
    RCC_APB1PeriphResetCmd( RCC_APB1Periph_TIM1, DISABLE);
}
```

3.7.1.2. TIM_TimeBaseInit

对于 Timer 进行初始化设置，函数主要进行的操作就是给定时器设置初始加载值和重新加载值，函数的代码如下所示：

```
void TIM_TimeBaseInit(CM3DS_MPS2_TIMER_TypeDef*CM3DS_MPS2_TIMERx,
                      TIM_TimeBaseInitTypeDef*TIM_TimeBaseInitStruct)
{
    /*Check the parameters*/
    assert_param(IS_TIM_ALL_PERIPH(CM3DS_MPS2_TIMERx));

    CM3DS_MPS2_TIMERx->VALUE  = TIM_TimeBaseInitStruct->TIM_Value;
    CM3DS_MPS2_TIMERx->RELOAD = TIM_TimeBaseInitStruct->TIM_Reload;
}
```

使用本函数的时候需要设置一个结构体变量 CM3DS_MPS2_TIMERx（CM3DS_MPS2_TIMER0 或者 CM3DS_MPS2_TIMER1），结构体变量包括初始加载值和重新加载值，函数的使用参考方式如下所示：

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
TIM_TimeBaseInitStruct.TIM_Reload = 12500000;
TIM_TimeBaseInitStruct.TIM_Value  = 12500000;
TIM_TimeBaseInit(CM3DS_MPS2_TIMER0, &TIM_TimeBaseInitStruct);
```

Timer0 和 Timer1 的时钟是 PCLK，在 cm3_system 这个模块中可以设置分频系数，本次实验中将其设置为 0，那么定时器的时钟信号就是系统时钟 100M。定时器的计数器的值将经过 1/100000000 秒减 1，如果需要定时 500ms，则装载值=0.5/(1/100000000)=50000000。需要注意的是，如果只给结构体变量 Value 赋值，没有给 Reload 赋值，这就意味着定时器只会运行一次，然后停止工作，如果要定时器循环工作就需要给 Reload 赋值，定时器第一次工作计数为 Value 的值，后面每次计数均为 Reload 的值。

3.7.1.3. TIM_CtrlCmd

对于定时器的控制寄存器进行设置，该函数主要使用到了三个变量，对于变量的说明如下表 3- 5 所示。

表 3- 5 TIM_CtrlCmd 函数变量说明表

变量名称	变量意义
CM3DS_MPS2_TIMERx	对应哪一个寄存器：CM3DS_MPS2_TIMER0 或者 CM3DS_MPS2_TIMER1

mode	设置控制寄存器的类型：使能定时器工作；选择外部输入作为定时器工作使能；选择外部输入作为时钟；选择中断使能。其对应的值在“CM3DS.timer.h”中可以找到，如下所示：
	`#define TIM_ENABLE (0x1)`
	`#define TIM_EXTERNAL_INPUT_AS_ENABLE (0x2)`
	`#define TIM_EXTERNAL_INPUT_AS_CLOCK (0x4)`
	`#define TIM_INTERRUPT_ENABLE (0x8)`

FunctionalState NewState	确定是否使能设置的类型：ENABLE：使能；DISABLE：不使能

函数内容：根据 FunctionalState NewState 的值确定是否使能设置的类型，如果使能的话，则通过向控制寄存器中写入相应的值来实现具体的功能，对于控制寄存器的说明参考 3.1.2 节中的表 3- 1 中的内容，函数的部分代码如下所示：

```
if (NewState != DISABLE)
{
    CM3DS_MPS2_TIMERx->CTRL |= mode;
}
else
{
    CM3DS_MPS2_TIMERx->CTRL &= 0x0;
}
```

函数的使用参考如下所示：

```
TIM_CtrlCmd(CM3DS_MPS2_TIMER0,TIM_ENABLE,ENABLE);//使能定时器 0 工作
```

3.7.1.4. TIM_GetITStatus

查看定时器中断的状态，通过读取中断状态寄存器（INTSTATUS）中的值，来判断是否产生定时器中断，如果为 1 则产生了中断，为 0 则没有。该函数只有一个变量 CM3DS_MPS2_TIMERx 需要设置。函数的具体代码如下所示：

```
ITStatus TIM_GetITStatus(CM3DS_MPS2_TIMER_TypeDef* CM3DS_MPS2_TIMERx)
{
    ITStatus bitstatus = RESET;
    uint16_t itstatus = 0x0;
    /*Check the parameters*/
    assert_param(IS_TIM_ALL_PERIPH(CM3DS_MPS2_TIMERx));
    itstatus =(uint16_t) CM3DS_MPS2_TIMERx->INTSTATUS;
    if (itstatus != (uint16_t) RESET)
    { bitstatus = SET; }
    else
    { bitstatus = RESET;}
    return bitstatus;
}
```

函数的使用方式如下所示：

```
status = TIM_GetITStatus(CM3DS_MPS2_TIMER0);
```

3.7.1.5. TIM_ClearIT

清除定时器中断，向中断清除寄存器中写入 1，该函数只有一个参数 CM3DS_MPS2_TIMERx 需要设置，函数代码如下所示：

```
void TIM_ClearIT(CM3DS_MPS2_TIMER_TypeDef*CM3DS_MPS2_TIMERx)
{
    /*Check the parameters*/
    assert_param(IS_TIM_ALL_PERIPH(CM3DS_MPS2_TIMERx));
    /*clear the IT pendint bit*/
    CM3DS_MPS2_TIMERx->INTCLEAR |= 0x1;
}
```

当定时器中断处理事件完成之后，都需要清除定时器中断，函数的使用参考方式如下所示。

```
TIM_ClearIT(CM3DS_MPS2_TIMER0);//清除定时器 0 的中断
```

3.7.2. Dual Timer 库

Dual Timer 定时器使用到的函数都在“CM3DS_dualtimer.c”这个文件之中。下面对于使用到函数进行一下简单的介绍。

3.7.2.1. DTIM_DeInit

将 Dual Timer 的外设时钟进行设置，函数的主要输入参数就是 CM3DS_MPS2_DUALTIMERx，该参数只能设置为 CM3DS_MPS2_DUALTIMER0。

函数的主要内容：对 Dual Timer 进行复位和释放，主要操作的寄存器就是 REG_PERI_RST，参考 3.7.1.1 节中表 3- 4 的内容可知，REG_PERI_RST 寄存器的第 2 位代表的是对 Dual Timer 的控制，对该位写入 1 和 0 即可实现释放和复位 Dual Timer 的功能。函数的部分代码如下所示：

```
if (CM3DS_MPS2_DUALTIMERx == CM3DS_MPS2_DUALTIMER0)
{
    RCC_APB1PeriphResetCmd(RCC_APB1Periph_DTIMER0, ENABLE);
    RCC_APB1PeriphResetCmd(RCC_APB1Periph_DTIMER0, DISABLE);
}
```

函数的使用参考方式如下所示：

```
DTIM_DeInit(CM3DS_MPS2_DUALTIMER0);
```

3.7.2.2. DTIM_Init

对 Dual Timer 进行初始化设置，函数输入参数一共有 5 个，对于函数参数变量说明如下表 3- 6 所示：

表 3- 6 DTIM_Init 函数变量说明表

变量名称	变量意义

CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0

DTIM	选择双定时器下的哪个定时器，一共有两个定时器可以选择，在“CM3DS_dualtimer.h”中定义如下：
	`#define DTIM_DUALTIMER0_1 ((uint32_t)0x0000FFFF)`
	`#define DTIM_DUALTIMER0_2 ((uint32_t)0xFFFFFFFF)`

DTIM_SIZE	选择 16 位或者 32 位的计数器，在“CM3DS_dualtimer.h”中定义如下：
	`#define DTIM_SIZE_32bit (0x1ul<<CM3DS_MPS2_DUALTIMER1_CTRL_SIZE_Pos)`
	`#define DTIM_SIZE_16bit (0x0ul<<CM3DS_MPS2_DUALTIMER1_CTRL_SIZE_Pos)`

DTIM_TIMERPRE	设置时钟分频系数，可以设置为 1、16、256，在“CM3DS_dualtimer.h”中定义如下：
	`#define DTIM_TIMERPRE_1 (0x00ul<<CM3DS_MPS2_DUALTIMER1_CTRL_PRESCALE_Pos)`
	`#define DTIM_TIMERPRE_16 (0x01ul<<CM3DS_MPS2_DUALTIMER1_CTRL_PRESCALE_Pos)`
	`#define DTIM_TIMERPRE_256 (0x10ul<<CM3DS_MPS2_DUALTIMER1_CTRL_PRESCALE_Pos)`

LOAD	设置定时器初始加载值，在没有对 Dual Timer 进行分频的时候，时钟为系统时钟 25M

函数内容：根据选择的定时器的不同，设置对应的控制寄存器的值，然后输入装载值。控制寄存器的第 1 位代表着选择多少位的计数器，如果是 0 则代表的是 16 位的计数器，为 1 则代表 32 位的计数器；控制寄存器的第[3:2]位代表着分频系数的选择：00：1 分频；01：16 分频；10：256 分频。根据输入的参数的选择，将对应的值进行或操作，然后将值写入控制寄存器中就可以了。函数的部分代码如下所示：

```
if(DTIM == DTIM_DUALTIMER0_1)
{
    CM3DS_MPS2_DUALTIMERx->Timer1Control = 0;
    read0 = CM3DS_MPS2_DUALTIMERx->Timer1Control;
    CM3DS_MPS2_DUALTIMERx->Timer1Control = read0 | DTIM_SIZE | DTIM_TIMERPRE;
    CM3DS_MPS2_DUALTIMERx->Timer1Load    = LOAD;
}
else if(DTIM == DTIM_DUALTIMER0_2)
{
    CM3DS_MPS2_DUALTIMERx->Timer2Control = 0;
    read1 = CM3DS_MPS2_DUALTIMERx->Timer2Control;
    CM3DS_MPS2_DUALTIMERx->Timer2Control = read1 | DTIM_SIZE | DTIM_TIMERPRE;
    CM3DS_MPS2_DUALTIMERx->Timer2Load    = LOAD;
}
```

函数的使用参考方式如下所示：

```
DTIM_Init(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1,DTIM_SIZE_32bit,DTIM_TIMERPRE_1,25000000);
```

3.7.2.3. DTIM_MODE

设置 Dual Timer 的工作模式，函数一共有 3 个变量，对其说明如下表 3-7 所示。

表 3- 7 DTIM_MODE 函数变量说明表

变量名称	变量意义
CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0
DTIM	选择使用哪个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 和 DTIM_DUALTIMER0_2

DTIM_MODE	工作模式的选择，Dual Timer 提供有三种工作模式：自由运行模式；周期计时模式；单次计时模式。在“CM3DS_dualtimer.h”中定义如下：
	`#define DTIM_MODE_ONE_SHOT_COUNT (0x1ul<<CM3DS_MPS2_DUALTIMER1_CTRL_ONESHOOT_Pos)`
	`#define DTIM_MODE_FREE_RUNNING (0x0ul<<CM3DS_MPS2_DUALTIMER1_CTRL_MODE_Pos)`
	`#define DTIM_MODE_PERIODIC (0x1ul<<CM3DS_MPS2_DUALTIMER1_CTRL_MODE_Pos)`

函数内容：根据设置的模式不同，向控制寄存器中写入不同的值，当设置为自由工作模式时，需要向控制寄存器的第 6 位写入 0；当选择为周期计时模式时，需要向控制寄存器的第 6 位写入 1；当选择为单次计时模式时，需要向控制寄存器的第 0 位写入 1。函数的部分代码如下所示：

```
if(DTIM == DTIM_DUALTIMER0_1)
{
    read2 = CM3DS_MPS2_DUALTIMERx->Timer1Control;
    CM3DS_MPS2_DUALTIMERx->Timer1Control = read2 | DTIM_MODE;
}
else if(DTIM == DTIM_DUALTIMER0_2)
{
    read3 = CM3DS_MPS2_DUALTIMERx->Timer2Control;
    CM3DS_MPS2_DUALTIMERx->Timer2Control = read3 | DTIM_MODE;
}
```

函数的使用方法如下所示：

```
DTIM_MODE(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1,DTIM_MODE_PERIODIC);
```

3.7.2.4. DTIM_START

是否使能定时器，函数的参数说明如下表 3- 8 所示：

表 3- 8 DTIM_START 函数参数说明表

变量名称	变量意义
CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0
DTIM	选择使用哪个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 和 DTIM_DUALTIMER0_2
FunctionalState NewState	是否使能定时器：ENABLE：使能，DISABLE：屏蔽

函数内容：是否使能定时器，通过控制寄存器的第 7 位进行控制，写入 0 则屏蔽中断，写入 1 则启用中断，函数的部分代码如下所示：

```
if ((DTIM == DTIM_DUALTIMER0_1)&&(NewState != DISABLE))
{
```

```
    read4 = CM3DS_MPS2_DUALTIMERx->Timer1Control;
    CM3DS_MPS2_DUALTIMERx->Timer1Control = read4 | (0x1ul << CM3DS_MPS2_DUALTIMER1_CTRL_EN_Pos);
}
else if((DTIM == DTIM_DUALTIMER0_2)&&(NewState != DISABLE))
{
    read5 = CM3DS_MPS2_DUALTIMERx->Timer2Control;
    CM3DS_MPS2_DUALTIMERx->Timer2Control = read5 | (0x1ul << CM3DS_MPS2_DUALTIMER1_CTRL_EN_Pos);
}
else
{
    CM3DS_MPS2_DUALTIMERx->Timer1Control &= ~CM3DS_MPS2_DUALTIMER_CTRL_EN_Msk;
    CM3DS_MPS2_DUALTIMERx->Timer2Control &= ~CM3DS_MPS2_DUALTIMER_CTRL_EN_Msk;
}
```

函数的使用方式如下所示：

```
DTIM_START(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1,ENABLE);
```

3.7.2.5. DTIM_SetBGLOAD

设置 Dual Timer 的重新加载值，函数的参数说明如下表 3- 9 所示。

表 3- 9 DTIM_SetBGLOAD 函数变量说明表

变量名称	变量意义
CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0
DTIM	选择使用哪个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 和 DTIM_DUALTIMER0_2
BGLOAD	设置重新加载值

函数内容：向后台加载寄存器中写入重新加载值，函数的部分代码如下所示：

```
if(DTIM == DTIM_DUALTIMER0_1)
{
    CM3DS_MPS2_DUALTIMERx->Timer1BGLoad  = BGLOAD;
}
else if(DTIM == DTIM_DUALTIMER0_2)
{
    CM3DS_MPS2_DUALTIMERx->Timer2BGLoad  = BGLOAD;
}
```

函数的使用方式如下所示：

```
DTIM_SetBGLOAD(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_2,0x0fffffff);
```

3.7.2.6. DTIM_ITConfig

设置 DualTimer 的中断触发类型，函数的参数说明如下表 3- 10 所示。

表 3- 10 DTIM_ITConfig 函数变量说明表

变量名称	变量意义
CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0

DTIM	选择使用哪个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 和 DTIM_DUALTIMER0_2
FunctionalState NewState	是否使能中断：ENABLE：使能；DISABLE：屏蔽

函数内容：启用中断的方式就是向对应的控制器的第 5 位写入 1，写入 0 则代表屏蔽中断，函数的部分代码如下所示：

```
if((DTIM == DTIM_DUALTIMER0_1)&&(NewState != DISABLE))
{
    temp0 = CM3DS_MPS2_DUALTIMERx->Timer1Control;
    CM3DS_MPS2_DUALTIMERx->Timer1Control = temp0 | (0x1ul << CM3DS_MPS2_DUALTIMER1
_CTRL_INTEN_Pos);
}
else if((DTIM == DTIM_DUALTIMER0_2)&&(NewState != DISABLE))
{
    temp1 = CM3DS_MPS2_DUALTIMERx->Timer2Control;
    CM3DS_MPS2_DUALTIMERx->Timer2Control = temp1 |  (0x1ul << CM3DS_MPS2_DUALTIMER
2_CTRL_INTEN_Pos) ;
}
else
{
    temp0 = CM3DS_MPS2_DUALTIMER0->Timer1Control;
    temp1 = CM3DS_MPS2_DUALTIMER0->Timer2Control;
    CM3DS_MPS2_DUALTIMERx->Timer1Control = temp0 | (0x0ul << CM3DS_MPS2_DUALTIMER1
_CTRL_INTEN_Pos);
    CM3DS_MPS2_DUALTIMERx->Timer2Control = temp1 | (0x0ul << CM3DS_MPS2_DUALTIMER2
_CTRL_INTEN_Pos) ;
}
```

函数的使用方式如下：

```
DTIM_ITConfig(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1, ENABLE);
```

3.7.2.7. DTIM_ClearIT

清除中断，函数的参数说明如下表 3- 11 所示。

表 3- 11 DTIM_ClearIT 函数参数说明表

变量名称	变量意义
CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0
DTIM	选择使用哪个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 和 DTIM_DUALTIMER0_2
FunctionalState NewState	是否使能中断：ENABLE：使能；DISABLE：屏蔽

函数内容：清除中断的方式就是向中断清除寄存器写值，写入任何值都可以清除中断，这里我们写入 1，函数的部分代码如下所示：

```
temp2 = CM3DS_MPS2_DUALTIMERx->Timer1IntClr;
temp3 = CM3DS_MPS2_DUALTIMERx->Timer2IntClr;
```

```
if((DTIM == DTIM_DUALTIMER0_1)&&(NewState != DISABLE))
{
    CM3DS_MPS2_DUALTIMERx->Timer1IntClr = temp2 | (0x1ul << CM3DS_MPS2
_DUALTIMER1_INTCLR_Pos);
}
else if((DTIM == DTIM_DUALTIMER0_2)&&(NewState != DISABLE))
{
    CM3DS_MPS2_DUALTIMERx->Timer2IntClr = temp3 | (0x1ul << CM3DS_MPS2
_DUALTIMER1_INTCLR_Pos);
}
else
{
    CM3DS_MPS2_DUALTIMERx->Timer1IntClr = temp2;
    CM3DS_MPS2_DUALTIMERx->Timer2IntClr = temp3;
}
```

函数的使用方式如下所示：

```
DTIM_ClearIT(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1,ENABLE);
```

3.7.2.8. DTIM_Timer0CurrentValue

查看定时器当前的值，函数的参数说明如下表 3- 12 所示：

表 3- 12 DTIM_Timer0CurrentValue 函数参数说明表

变量名称	变量意义
CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0
DTIM	选择使用哪个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 和 DTIM_DUALTIMER0_2

函数内容：查看定时器当前值的方式就是读取当前值寄存器中的值，然后将读到的值返回，函数的部分代码如下所示：

```
if(DTIM == DTIM_DUALTIMER0_1)
{
    value0 = CM3DS_MPS2_DUALTIMERx -> Timer1Value;
    return value0;
}
else    if(DTIM == DTIM_DUALTIMER0_2)
{
    value1 = CM3DS_MPS2_DUALTIMERx -> Timer2Value;
    return value1;
}
```

3.7.2.9. DTIM_Timer0RISValue

查看原始的中断状态，函数的参数说明如下表 3- 13 所示。

表 3- 13 DTIM_Timer0RISValue 函数参数说明表

变量名称	变量意义
CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0
DTIM	选择使用哪个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 和 DTIM_DUALTIMER0_2

函数内容：查看原始中断状态，就是读取原始中断寄存器中的值，然后将读取到的值返回，函数的部分代码如下所示：

```
if(DTIM == DTIM_DUALTIMER0_1)
{
    RIS1 = CM3DS_MPS2_DUALTIMERx->Timer1RIS;
    return RIS1;
}
else if (DTIM == DTIM_DUALTIMER0_2)
{
    RIS1 = CM3DS_MPS2_DUALTIMERx->Timer2RIS;
    return RIS2;
}
```

3.7.2.10. DTIM_Timer0MISValue

查看中断寄存器的状态，函数的参数说明如下表 3- 14 所示。

表 3- 14DTIM_Timer0MISValue 函数参数说明表

变量名称	变量意义
CM3DS_MPS2_DUALTIMERx	Dual Timer 结构体变量，设置为 CM3DS_MPS2_DUALTIMER0
DTIM	选择使用哪个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 和 DTIM_DUALTIMER0_2

函数内容：读取中断状态寄存器中的值，将读取到值返回。函数的部分代码如下所示：

```
if(DTIM == DTIM_DUALTIMER0_1)
{
    MIS1 = CM3DS_MPS2_DUALTIMERx->Timer1MIS;
    return MIS1;
}
else if (DTIM == DTIM_DUALTIMER0_2)
{
    MIS2 = CM3DS_MPS2_DUALTIMERx->Timer2MIS;
    return MIS2;
}
```

3.7.3. 应用程序编写

通过上述对库函数的讲解，编写定时器的应用程序。在这里我们提供有对应的应用库文件“Timer”，用户在需要使用定时器的时候将其添加至 HARDWARE 文件夹之中，添加方式参考实验 1 中 1.8.2 节中的内容，接下来将简单的介绍一

下编写的对应的应用程序。

3.7.3.1. Timer 中断应用

下面对添加的 timer.c 文件中的函数做一下简单的介绍，用户可根据自己的需求进行修改。

1. Timer_Init

初始化定时器，用来设置定时器的初始加载值和重新加载值，并使能需要使用的定时器。代码如下所示。

```
void Timer_Init(CM3DS_MPS2_TIMER_TypeDef*CM3DS_MPS2_TIMERx,uint32_t Value,uint32_
t Reload)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
    TIM_TimeBaseInitStruct.TIM_Reload = Reload;
    TIM_TimeBaseInitStruct.TIM_Value  = Value;
    if(CM3DS_MPS2_TIMERx == CM3DS_MPS2_TIMER0){
      TIM_TimeBaseInit(CM3DS_MPS2_TIMER0, &TIM_TimeBaseInitStruct);
      TIM_CtrlCmd(CM3DS_MPS2_TIMER0,TIM_ENABLE,ENABLE);//使能定时器 0 工作
    }
    else if(CM3DS_MPS2_TIMERx == CM3DS_MPS2_TIMER1){
      TIM_TimeBaseInit(CM3DS_MPS2_TIMER1, &TIM_TimeBaseInitStruct);
      TIM_CtrlCmd(CM3DS_MPS2_TIMER1,TIM_ENABLE,ENABLE);//使能定时器 1 工作
    }
}
```

函数主要有三个参数需要设置，其说明如下表表 3- 15 所示。

表 3- 15 Timer_Init 函数参数说明表

参数名称	参数意义
CM3DS_MPS2_TIMERx	使用哪一个寄存器，一共有两个寄存器可供选择
Value	初始加载值，定时器第一次工作的计数值，因为定时器的时钟频率为 25M，那么延时 1S，所需的计数值为 25000000。
Reload	重新加载值，如果要定时器循环工作就需要给 Reload 赋值，定时器第一次工作计数为 Value 的值，后面每次计数均为 Reload 的值

函数的参考使用方式如下：

```
Timer_Init(CM3DS_MPS2_TIMER1,100000000,100000000); //timer1 定时 1S
```

2. TIM_EXTI_Init

初始化定时器的中断，函数的主要内容：根据不同的定时器通过 TIM_CtrlCmd 函数启用对应定时器中断，然后清除挂起的中断，启用外部中断，函数的代码如下所示：

```
void TIM_EXTI_Init(CM3DS_MPS2_TIMER_TypeDef*CM3DS_MPS2_TIMERx,uint8_t en)
{
```

```
    if(CM3DS_MPS2_TIMERx == CM3DS_MPS2_TIMER0){
        //使能定时器 0 的中断
        TIM_CtrlCmd(CM3DS_MPS2_TIMER0,TIM_INTERRUPT_ENABLE,ENABLE);
        //清除挂起的中断
        NVIC_ClearPendingIRQ(TIMER0_IRQn);
        if(en == ENABLE){
            //启用外部中断
            NVIC_EnableIRQ(TIMER0_IRQn);
        }
        else
            NVIC_DisableIRQ(TIMER0_IRQn);
    }
    else if(CM3DS_MPS2_TIMERx == CM3DS_MPS2_TIMER1){
        //使能定时器 1 的中断
        TIM_CtrlCmd(CM3DS_MPS2_TIMER1,TIM_INTERRUPT_ENABLE,ENABLE);
        //清除挂起的中断
        NVIC_ClearPendingIRQ(TIMER1_IRQn);
        if(en == ENABLE){
            //启用外部中断
            NVIC_EnableIRQ(TIMER1_IRQn);
        }
        else
            NVIC_DisableIRQ(TIMER1_IRQn);
    }
}
```

函数一共有两个参数需要设置：CM3DS_MPS2_TIMERx：选择哪一个定时器，en：是否使能定时器中断。函数的使用参考方式如下所示：

```
TIM_EXTI_Init(CM3DS_MPS2_TIMER0,ENABLE); //启用定时器 0 的中断
```

3. TIMER0_Handler & TIMER1_Handler

定时器 0 和定时器 1 的中断服务函数，添加计数器减到 0 之后需要处理的事务，事物处理完成之后，清除中断，函数的代码如下所示：

```
void TIMER0_Handler(void)
{
    /*********添加定时器 0 的中断服务函数**********/

    /*****************************************/
    TIM_ClearIT(CM3DS_MPS2_TIMER0);
}
void TIMER1_Handler(void)
{
    /*********添加定时器 1 的中断服务函数**********/
```

```
    /*****************************************/
    TIM_ClearIT(CM3DS_MPS2_TIMER0);
}
```

3.7.3.2. Dual Timer 中断应用

下面对 Dualtimer.c 文件中的中的函数做一下简单的介绍，用户可根据自己的需求进行修改。

1. DTMI_Init

初始化 Dualtimer，函数的主要内容：设置计数器的位数、分频系数、初始加载值、工作模式、重新加载值，最后使能定时器和定时器中断。函数代码如下所示。

```
void DTMI_Init(uint32_t DTIM_DUALTIMER0_x,uint32_t SIZE,uint32_t MODE,uint32_t
TIMERPRE,uint32_t Value,uint32_t Reload)
{
    RCC_APB1PeriphResetCmd(RCC_APB1Periph_DTIMER0, DISABLE);
    //初始化定时器，设置计数器位数，设置分频系数，设置初始加载值
    DTIM_Init(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_x,SIZE,TIMERPRE,Value);
    //设置定时器的工作模式
    DTIM_MODE(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_x,MODE);
    //使能定时器的中断
    DTIM_ITConfig(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_x, ENABLE);
    //设置重新加载值
    DTIM_SetBGLOAD(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_x,Reload);
    //使能定时器
    DTIM_START(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_x,ENABLE);
}
```

函数的参数说明如下表表 3- 16 所示：

表 3- 16 DTMI_Init 函数参数说明表

参数名称	参数意义
DTIM_DUALTIMER0_x	使用 Dualtimer 中的哪一个定时器，一共有两个定时器可以选择：DTIM_DUALTIMER0_1 或者 DTIM_DUALTIMER0_2
SIZE	设置定时器的位数，32 位或者 16 位。
MODE	定时器工作模式选择，一共有三个工作模式可以选择：自由运行模式；周期计时模式；单次计时模式。
TIMERPRE	设置时钟分频系数，可以设置为 1、16、256，
Value	设置初始加载值，根据 CM3 的时钟频率进行修改，当时钟频率为 100M 的时候，定时 1S，value 需要写入的值为 100000000。
Reload	设置重复加载值

函数使用的时候，填入需要设置的参数就可以了。举例如下：

```
//初始化 dual timer 中的 time1，32 位计数器，周期运行模式，1 分频,定时 0.5S
```

```
DTMI_Init(DTIM_DUALTIMER0_1,DTIM_SIZE_32bit,DTIM_MODE_PERIODIC,DTIM_TIMERPRE_1,50000000,50000000);
```

2. DTMI_EXTI_Init

初始化 Dual Timer 的中断，清除启用的定时器中断，然后判断是否使能中断，代码如下所示。

```
void DTMI_EXTI_Init(uint32_t DTIM_DUALTIMER0_x,uint8_t en)
{
    //清除中断
    NVIC_DisableIRQ(DUALTIMER_IRQn);
    //清除定时器的中断
    DTIM_ClearIT(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_x,ENABLE);
    if(en == ENABLE){
    //使能中断
        NVIC_EnableIRQ(DUALTIMER_IRQn);
    }
    else
        NVIC_DisableIRQ(DUALTIMER_IRQn);
}
```

函数一共有两个参数可以选择：DTIM_DUALTIMER0_x：选择哪一个定时器，这里需要与初始化的定时器相对应；en：是否使能中断。函数的使用参考方式如下：

```
DTMI_EXTI_Init(DTIM_DUALTIMER0_1,ENABLE);//启用 dual timer 的 time1 中断
```

3. DUALTIMER_HANDLER

Dual Timer 中断服务函数，函数主要内容：使用 DTIM_Timer0MISValue 函数读取定时器的中断状态寄存器中的值，然后根据该值判断进入了哪一个定时器的中断，然后添加中断服务函数，最后清除对应的中断。

```
void DUALTIMER_HANDLER (void)
{
    uint32_t status1,status2;
    status1 = DTIM_Timer0MISValue(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1);
    status2 = DTIM_Timer0MISValue(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_2);
    if((status1 & 0x01) == 1){ //dual timer1 中定时器 1 产生了中断
    /*********添加定时器 1 的中断服务函数**********/

    /*****************************************/
    DTIM_ClearIT(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1,ENABLE);
    }
    if((status2 & 0x01) == 1){//dual timer1 中定时器 2 产生了中断
    /*********添加定时器 2 的中断服务函数**********/

```

```
    /****************************************/
    DTIM_ClearIT(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_2,ENABLE);
    }
}
```

3.7.4. 添加用户代码

实验功能：Timer0 和 Timer1 分别控制 LED0 和 LED2 的闪烁，翻转时间设置为 1S（每隔 2S 闪烁一次）；Dual Timer 中的 Timer1 和 Timer2 分别控制 LED1 和 LED3 的闪烁，翻转时间设置为 0.5S（每隔 1S 闪烁一次）。

1. 添加 Timer0 的中断服务函数，翻转 LED0 的状态。

```
void TIMER0_Handler(void)
{
    /*********添加定时器 0 的中断服务函数**********/
    led0_status =~led0_status;
    GPIO_WriteBit(CM3DS_MPS2_GPIO0, LED0,led0_status);

    /****************************************/
    TIM_ClearIT(CM3DS_MPS2_TIMER0);
}
```

2. 添加 Timer1 的中断服务函数，翻转 LED2 的状态。

```
void TIMER1_Handler(void)
{
    /*********添加定时器 1 的中断服务函数**********/
    led2_status =~led2_status;
    GPIO_WriteBit(CM3DS_MPS2_GPIO0, LED2,led2_status);

    /****************************************/
    TIM_ClearIT(CM3DS_MPS2_TIMER1);
}
```

3. 添加 Dual Timer 的中断服务函数，Timer1 和 Timer2 分别控制 LED1 和 LED3 的翻转。

```
void DUALTIMER_HANDLER (void)
{
    uint32_t status1,status2;
    status1 = DTIM_Timer0MISValue(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1);
    status2 = DTIM_Timer0MISValue(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_2);
    if((status1 & 0x01) == 1){ //dual timer1 中定时器 1 产生了中断
    /*********添加定时器 1 的中断服务函数**********/
    led1_status =~led1_status;
    GPIO_WriteBit(CM3DS_MPS2_GPIO0, LED1,led1_status);
    /****************************************/
```

```
    DTIM_ClearIT(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_1,ENABLE);
    }
    if((status2 & 0x01) == 1){//dual timer1 中定时器 2 产生了中断
    /*********添加定时器 2 的中断服务函数**********/
    led3_status =~led3_status;
    GPIO_WriteBit(CM3DS_MPS2_GPIO0, LED3,led3_status);
    /*****************************************/
    DTIM_ClearIT(CM3DS_MPS2_DUALTIMER0,DTIM_DUALTIMER0_2,ENABLE);
    }
}
```

主函数内容：点亮四个 LED 灯，启用 Timer 和 Dual Timer 的中断，代码如下所示：

```
#include "CM3DS_gpio.h"
#include "CM3DS_timer.h"
#include "CM3DS_dualtimer.h"
#include "timer.h"
#include "dualtimer.h"
#include "key_exit.h"
int main(void)
{
    //点亮 LED 灯
    GPIO_WriteBit(CM3DS_MPS2_GPIO0, LED0,0);
    GPIO_WriteBit(CM3DS_MPS2_GPIO0, LED1,0);
    GPIO_WriteBit(CM3DS_MPS2_GPIO0, LED2,0);
    GPIO_WriteBit(CM3DS_MPS2_GPIO0, LED3,0);
    //启用 Timer1 的中断,定时 1S
    Timer_Init(CM3DS_MPS2_TIMER1,SystemCoreClock,SystemCoreClock);
    TIM_EXTI_Init(CM3DS_MPS2_TIMER1,ENABLE);
    //启用 Timer0 的中断,定时 1S
    Timer_Init(CM3DS_MPS2_TIMER0,SystemCoreClock,SystemCoreClock);
    TIM_EXTI_Init(CM3DS_MPS2_TIMER0,ENABLE);
    //启用 dual timer 中的 time1，32 位计数器，周期运行模式，1 分频,定时 0.5S
    DTMI_Init(DTIM_DUALTIMER0_1,DTIM_SIZE_32bit,DTIM_MODE_PERIODIC,
                        DTIM_TIMERPRE_1,SystemCoreClock/2,SystemCoreClock/2);
    DTMI_EXTI_Init(DTIM_DUALTIMER0_1,ENABLE);
    //启用 dual timer 中的 time2，32 位计数器，周期运行模式，1 分频,定时 0.5S
    DTMI_Init(DTIM_DUALTIMER0_2,DTIM_SIZE_32bit,DTIM_MODE_PERIODIC,
                        DTIM_TIMERPRE_1,SystemCoreClock/2,SystemCoreClock/2);
    DTMI_EXTI_Init(DTIM_DUALTIMER0_2,ENABLE);
    while(1){
    }
}
```

代码中定时器的加载值调用了“system_CM3DS.c”文件中的 SystemCoreClock，该值代表的就是 CM3 的时钟频率的数值。

3.8. 板级验证

3.8.1. 实验所需硬件

(1) AC208 开发板
(2) FPGA 下载器：XIST USB Cable
(3) CM3 仿真器：DAP Link
(4) 电源线一根

3.8.2. 硬件连接

硬件连接参考实验一。

3.8.3. 下载文件至目标板

下载文件的方式参考实验一。

3.8.4. 功能演示

将程序下载之后，开发板上的四个 LED 灯都在不断的闪烁，LED1 和 LED3 闪烁一致（每隔 2S 闪烁一次），LED0 和 LED2 闪烁一致（每隔 1S 闪烁一致）。

3.9. 思考与总结

本次实验通过定时器中断的方式实现了对 LED 灯闪烁的控制。这里需要提醒的一点的是：添加中断服务子函数中的内容的时候，不易进行复杂的操作，否则会导致定时器溢出，导致定时时间不准。