

## 目 录

|       |                   |    |
|-------|-------------------|----|
| 第 1 章 | 通用定时器(Timer)..... | 1  |
| 1.1   | Timer 总体特性 .....  | 1  |
| 1.2   | Timer 功能概述 .....  | 2  |
| 1.3   | Timer 库函数 .....   | 3  |
| 1.4   | Timer 例程 .....    | 11 |

## 第1章 通用定时器(Timer)

表 1.1 Timer 函数索引

| 函 数 原 型                                                                                              | 页码 |
|------------------------------------------------------------------------------------------------------|----|
| void <b>TimerConfigure</b> (unsigned long ulBase, unsigned long ulConfig)                            | 4  |
| void <b>TimerControlStall</b> (unsigned long ulBase, unsigned long ulTimer, tBoolean bStall)         | 5  |
| void <b>TimerControlTrigger</b> (unsigned long ulBase, unsigned long ulTimer, tBoolean bEnable)      | 6  |
| void <b>TimerControlEvent</b> (unsigned long ulBase, unsigned long ulTimer, unsigned long ulEvent)   | 6  |
| void <b>TimerControlLevel</b> (unsigned long ulBase, unsigned long ulTimer, tBoolean bInvert)        | 6  |
| void <b>TimerLoadSet</b> (unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)        | 7  |
| unsigned long <b>TimerLoadGet</b> (unsigned long ulBase, unsigned long ulTimer)                      | 7  |
| unsigned long <b>TimerValueGet</b> (unsigned long ulBase, unsigned long ulTimer)                     | 7  |
| void <b>TimerEnable</b> (unsigned long ulBase, unsigned long ulTimer)                                | 7  |
| void <b>TimerDisable</b> (unsigned long ulBase, unsigned long ulTimer)                               | 8  |
| void <b>TimerRTCEnable</b> (unsigned long ulBase)                                                    | 8  |
| void <b>TimerRTCDisable</b> (unsigned long ulBase)                                                   | 8  |
| void <b>TimerQuiesce</b> (unsigned long ulBase)                                                      | 8  |
| void <b>TimerMatchSet</b> (unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)       | 8  |
| unsigned long <b>TimerMatchGet</b> (unsigned long ulBase, unsigned long ulTimer)                     | 9  |
| void <b>TimerPrescaleSet</b> (unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)    | 9  |
| unsigned long <b>TimerPrescaleGet</b> (unsigned long ulBase, unsigned long ulTimer)                  | 9  |
| void <b>TimerIntEnable</b> (unsigned long ulBase, unsigned long ulIntFlags)                          | 9  |
| void <b>TimerIntDisable</b> (unsigned long ulBase, unsigned long ulIntFlags)                         | 10 |
| void <b>TimerIntClear</b> (unsigned long ulBase, unsigned long ulIntFlags)                           | 10 |
| unsigned long <b>TimerIntStatus</b> (unsigned long ulBase, tBoolean bMasked)                         | 10 |
| void <b>TimerIntRegister</b> (unsigned long ulBase, unsigned long ulTimer, void (*pfnHandler)(void)) | 10 |
| void <b>TimerIntUnregister</b> (unsigned long ulBase, unsigned long ulTimer)                         | 11 |

### 1.1 Timer 总体特性

在 Stellaris 系列 ARM 内部通常集成有 2 ~ 4 个通用定时器模块 ( General-Purpose Timer Module , GPTM ), 分别称为 Timer0、Timer1、Timer2 和 Timer3。它们的用法是相同的：每个 Timer 模块都可以配置为一个 32 位定时器或一个 32 位 RTC 定时器；也可以拆分为两个 16 位的定时/计数器 TimerA 和 TimerB，它们可以被配置为独立运行的定时器、事件计数器或 PWM。

Timer 模块具有非常丰富的功能：

- 32 位定时器模式：
  - 可编程单次触发 ( one-shot ) 定时器
  - 可编程周期 ( periodic ) 定时器
  - 实时时钟 RTC ( Real Time Clock )
  - 软件可控的事件暂停 ( 用于单步调试时暂停计数，RTC 模式除外 )
- 16 位定时器模式：

- 带 8 位预分频器的通用定时器功能
- 可编程单次触发 (one-shot) 定时器
- 可编程周期 (periodic) 定时器
- 软件可控的事件暂停
- 16 位输入捕获模式：
  - 输入边沿计数捕获
  - 输入边沿定时捕获
- 16 位 PWM 模式：
  - 用法简单的 PWM (Pulse-Width Modulation, 脉宽调制) 模式
  - 可通过软件实现 PWM 信号周期、占空比、输出反相等的控制

## 1.2 Timer 功能概述

Timer 模块的功能在总体上可以分成 32 位模式和 16 位模式两大类。在 32 位模式下, TimerA 和 TimerB 被连在一起形成一个完整的 32 位计数器, 对 Timer 的各项操作, 如装载初值、运行控制、中断控制等, 都用对 TimerA 的操作作为总体上的 32 位控制, 而对 TimerB 的操作无任何效果。在 16 位模式下, 对 TimerA 的操作仅对 TimerA 有效, 对 TimerB 的操作仅对 TimerB 有效, 即对两者的操控是完全独立进行的。

每一个 Timer 模块对应两个 CCP 管脚。CCP 是 “Capture Compare PWM” 的缩写, 意为 “捕获/比较/脉宽调制”。在 32 位单次触发和周期定时模式下, CCP 功能无效 (与之复用的 GPIO 管脚功能仍然正常)。在 32 位 RTC 模式下, 偶数 CCP 管脚 (CCP0、CCP2、CCP4 等) 作为 RTC 时钟源的输入, 而奇数 CCP 管脚 (CCP1、CCP3、CCP5 等) 无效。在 16 位模式下, 计数捕获、定时捕获、PWM 功能都会用到 CCP 管脚, 对应关系是: Timer0A 对应 CCP0、Timer0B 对应 CCP1, Timer1A 对应 CCP2、Timer1B 对应 CCP3, 依此类推。

### 1. 32 位单次触发/周期定时器

在这两种模式中, Timer 都被配置成一个 32 位的递减计数器, 用法类似, 只是单次触发模式只能定时一次, 如果需要再次定时则必须重新配置, 而周期模式则可以周而复始地定时, 除非被关闭。在计数到 0x00000000 时, 可以在软件的控制下触发中断或输出一个内部单时钟周期脉冲信号, 该信号可以用来触发 ADC 采样。

### 2. 32 位 RTC 定时器

在该模式中, Timer 被配置成一个 32 位的递增计数器。

RTC 功能的时钟源来自偶数 CCP 管脚的输入。在 LM3S101/102 里, RTC 时钟信号从专门的 “32KHz” 管脚输入。输入的时钟频率应当为精准的 32.768KHz, 在芯片内部有一个 RTC 专用的预分频器, 固定为 32768 分频。因此最终输入到 RTC 计数器的时钟频率正好是 1Hz, 即每过 1 秒钟 RTC 计数器增 1。

RTC 计数器从 0x00000000 开始计满需要  $2^{32}$  秒, 这是个极长的时间, 有 136 年! 因此 RTC 真正的用法是: 初始化后不需要更改配置 (调整时间或日期时例外), 只需要修改匹配寄存器的值, 而且要保证匹配值总是超前于当前计数值。每次匹配时可产生中断 (如果中断已被使能), 据此可以计算出当前的年月日、时分秒以及星期。在中断服务函数里应当重新设置匹配值, 并且匹配值仍要超前于当前的计数值。

注意: 在实际应用当中一般不会真正采用 Timer 模块的 RTC 功能来实现一个低功耗万年历系统, 因为芯片一旦出现复位或断电的情况就会清除 RTC 计数值。取而代之的是冬眠模块 (Hibernation Module) 的 RTC 功能, 由于采用了后备电池, 因此不怕复位和 VDD 断

电，并且功耗很低。

### 3. 16 位单次触发/周期定时器

一个 32 位的 Timer 可以被拆分为两个单独运行的 16 位定时/计数器，每一个都可以被配置成带 8 位预分频（可选功能）的 16 位递减计数器。如果使用 8 位预分频功能，则相当于 24 位定时器。具体用法跟 32 位单次触发/周期定时模式类似，不同的是对 TimerA 和 TimerB 的操作是分别独立进行的。

### 4. 16 位输入边沿计数捕获

在该模式中，TimerA 或 TimerB 被配置为能够捕获外部输入脉冲边沿事件的递减计数器。共有 3 种边沿事件类型：正边沿、负边沿、双边沿。

该模式的工作过程是：设置装载值，并预设一个匹配值（应当小于装载值）；计数使能后，在特定的 CCP 管脚每输入 1 个脉冲（正边沿、负边沿或双边沿有效），计数值就减 1；当计数值与匹配值相等时停止运行并触发中断（如果中断已被使能）。如果需要再次捕获外部脉冲，则要重新进行配置。

### 5. 16 位输入边沿定时捕获

在该模式中，TimerA 或 TimerB 被配置为自由运行的 16 位递减计数器，允许在输入信号的上升沿或下降沿捕获事件。

该模式的工作过程是：设置装载值（默认为 0xFFFF）捕获边沿类型；计数器被使能后开始自由运行，从装载值开始递减计数，计数到 0 时重装初值，继续计数；如果从 CCP 管脚上出现有效的输入脉冲边沿事件，则当前计数值被自动复制到一个特定的寄存器里，该值会一直保存不变，直至遇到下一个有效输入边沿时被刷新。为了能够及时读取捕获到的计数值，应当使能边沿事件捕获中断，并在中断服务函数里读取。

### 6. 16 位 PWM

Timer 模块还可以用来产生简单的 PWM 信号。在 Stellaris 系列 ARM 众多型号当中，对于片内未集成专用 PWM 模块的，可以利用 Timer 模块的 16 位 PWM 功能来产生 PWM 信号，只不过功能较为简单。对于片内已集成专用 PWM 模块的，但仍然不够用时，则可以从 Timer 模块借用。

在 PWM 模式中，TimerA 或 TimerB 被配置为 16 位的递减计数器，通过设置适当的装载值（决定 PWM 周期）和匹配值（决定 PWM 占空比）来自动地产生 PWM 方波信号从相应的 CCP 管脚输出。在软件上，还可以控制输出反相，参见函数 TimerControlLevel()。

## 1.3 Timer 库函数

在使用某个 Timer 模块之前，应当首先将其使能，方法为：

```
#define SysCtlPeriEnable      SysCtlPeripheralEnable
SysCtlPeriEnable(SYSCTL_PERIPH_TIMERn);           // 末尾的 n 取 0、1、2 或 3
```

对于 RTC、计数捕获、定时捕获、PWM 等功能，需要用到相应的 CCP 管脚作为信号的输入或输出。因此还必须对 CCP 所在的 GPIO 端口进行配置。以 CCP2 为例，假设在 PD5 管脚上，则配置方法为：

```
#define CCP2_PERIPH          SYSCTL_PERIPH_GPIOD
#define CCP2_PORT            GPIO_PORTD_BASE
```

```
#define CCP2_PIN          GPIO_PIN_5
SysCtlPeripheralEnable(CCP2_PERIPH);           // 使能 CCP2 管脚所在的 GPIO
GPIOPinTypeTimer(CCP2_PORT, CCP2_PIN);         // 配置 CCP2 管脚为 Timer 功能
```

## 1. 配置与控制

函数 TimerConfigure( )用来配置 Timer 的工作模式，这些模式包括：32 位单次触发定时器、32 位周期定时器、32 位 RTC 定时器、16 位输入边沿计数捕获、16 位输入边沿定时捕获和 16 位 PWM。对 16 位模式，Timer 被拆分为两个独立的定时/计数器 TimerA 和 TimerB，该函数能够分别对它们进行配置。详见表 1.2 的描述。

表 1.2 函数 TimerConfigure( )

| 功能                    | 配置 Timer 模块的工作模式                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|----------------|----------------------|--------------|------------------|-----------------|----------------------|--------------------|----------------------|------------------|-----------------------|--------------------|----------------------|--------------------|-----------------|--------------------|----------------------|--------------------|----------------------|------------------|-----------------------|--------------------|----------------------|--------------------|-----------------|--------------------|
| 原型                    | void TimerConfigure(unsigned long ulBase, unsigned long ulConfig)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| 参数                    | <p>ulBase：Timer 模块的基址，取值 TIMERN_BASE (n 为 0、1、2 或 3)</p> <p>ulConfig：Timer 模块的配置</p> <p>在 32 位模式下应当取下列值之一：</p> <table> <tr> <td>TIMER_CFG_32_BIT_OS</td><td>// 32 位单次触发定时器</td></tr> <tr> <td>TIMER_CFG_32_BIT_PER</td><td>// 32 位周期定时器</td></tr> <tr> <td>TIMER_CFG_32_RTC</td><td>// 32 位 RTC 定时器</td></tr> </table> <p>在 16 位模式下，一个 32 位的 Timer 被拆分成两个独立运行的子定时器 TimerA 和 TimerB。</p> <p>配置 TimerA 的方法是参数 ulConfig 先取值 TIMER_CFG_16_BIT_PAIR 再与下列值之一进行“或运算”的组合形式：</p> <table> <tr> <td>TIMER_CFG_A_ONE_SHOT</td><td>// TimerA 为单次触发定时器</td></tr> <tr> <td>TIMER_CFG_A_PERIODIC</td><td>// TimerA 为周期定时器</td></tr> <tr> <td>TIMER_CFG_A_CAP_COUNT</td><td>// TimerA 为边沿事件计数器</td></tr> <tr> <td>TIMER_CFG_A_CAP_TIME</td><td>// TimerA 为边沿事件定时器</td></tr> <tr> <td>TIMER_CFG_A_PWM</td><td>// TimerA 为 PWM 输出</td></tr> </table> <p>配置 TimerB 的方法是参数 ulConfig 先取值 TIMER_CFG_16_BIT_PAIR 再与下列值之一进行“或运算”的组合形式：</p> <table> <tr> <td>TIMER_CFG_B_ONE_SHOT</td><td>// TimerB 为单次触发定时器</td></tr> <tr> <td>TIMER_CFG_B_PERIODIC</td><td>// TimerB 为周期定时器</td></tr> <tr> <td>TIMER_CFG_B_CAP_COUNT</td><td>// TimerB 为边沿事件计数器</td></tr> <tr> <td>TIMER_CFG_B_CAP_TIME</td><td>// TimerB 为边沿事件定时器</td></tr> <tr> <td>TIMER_CFG_B_PWM</td><td>// TimerB 为 PWM 输出</td></tr> </table> | TIMER_CFG_32_BIT_OS | // 32 位单次触发定时器 | TIMER_CFG_32_BIT_PER | // 32 位周期定时器 | TIMER_CFG_32_RTC | // 32 位 RTC 定时器 | TIMER_CFG_A_ONE_SHOT | // TimerA 为单次触发定时器 | TIMER_CFG_A_PERIODIC | // TimerA 为周期定时器 | TIMER_CFG_A_CAP_COUNT | // TimerA 为边沿事件计数器 | TIMER_CFG_A_CAP_TIME | // TimerA 为边沿事件定时器 | TIMER_CFG_A_PWM | // TimerA 为 PWM 输出 | TIMER_CFG_B_ONE_SHOT | // TimerB 为单次触发定时器 | TIMER_CFG_B_PERIODIC | // TimerB 为周期定时器 | TIMER_CFG_B_CAP_COUNT | // TimerB 为边沿事件计数器 | TIMER_CFG_B_CAP_TIME | // TimerB 为边沿事件定时器 | TIMER_CFG_B_PWM | // TimerB 为 PWM 输出 |
| TIMER_CFG_32_BIT_OS   | // 32 位单次触发定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_32_BIT_PER  | // 32 位周期定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_32_RTC      | // 32 位 RTC 定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_A_ONE_SHOT  | // TimerA 为单次触发定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_A_PERIODIC  | // TimerA 为周期定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_A_CAP_COUNT | // TimerA 为边沿事件计数器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_A_CAP_TIME  | // TimerA 为边沿事件定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_A_PWM       | // TimerA 为 PWM 输出                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_B_ONE_SHOT  | // TimerB 为单次触发定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_B_PERIODIC  | // TimerB 为周期定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_B_CAP_COUNT | // TimerB 为边沿事件计数器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_B_CAP_TIME  | // TimerB 为边沿事件定时器                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| TIMER_CFG_B_PWM       | // TimerB 为 PWM 输出                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| 返回                    | 无                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |
| 示例                    | <pre>// 配置 Timer0 为 32 位单次触发定时器 TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_OS);  // 配置 Timer1 为 32 位周期定时器 TimerConfigure(TIMER1_BASE, TIMER_CFG_32_BIT_PER);  // 配置 Timer2 为 32 位 RTC 定时器 TimerConfigure(TIMER2_BASE, TIMER_CFG_32_RTC);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                     |                |                      |              |                  |                 |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |                      |                    |                      |                  |                       |                    |                      |                    |                 |                    |

|  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <pre>// 在 Timer0 当中，配置 TimerA 为单次触发定时器（不配置 TimerB） TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR   TIMER_CFG_A_ONE_SHOT);  // 在 Timer0 当中，配置 TimerB 为周期定时器（不配置 TimerA） TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR   TIMER_CFG_B_PERIODIC);  // 在 Timer0 当中，配置 TimerA 为单次触发定时器，同时配置 TimerB 为周期定时器 TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR                   TIMER_CFG_A_ONE_SHOT                   TIMER_CFG_B_PERIODIC);  // 在 Timer1 当中，配置 TimerA 为边沿事件计数器、TimerB 为边沿事件定时器 TimerConfigure(TIMER1_BASE, TIMER_CFG_16_BIT_PAIR                   TIMER_CFG_A_CAP_COUNT                   TIMER_CFG_B_CAP_TIME);  // 在 Timer2 当中，TimerA、TimerB 都配置为 PWM 输出 TimerConfigure(TIMER2_BASE, TIMER_CFG_16_BIT_PAIR                   TIMER_CFG_A_PWM                   TIMER_CFG_B_PWM);</pre> |
|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

函数 TimerControlStall( )可以控制 Timer 在程序在单步调试时暂停运行，这为用户随时观察相关寄存器的内容提供了方便，否则在单步调试时 Timer 可能还在飞速运行，从而影响互动的调试效果。但是该函数对 32 位 RTC 定时器模式无效，即 RTC 定时器一旦使能就会独立地运行，除非被禁止计数。参见表 1.3 的描述。

函数 TimerControlTrigger( )可以控制 Timer 在单次触发/周期定时器溢出时产生一个内部的单时钟周期脉冲信号，该信号可以用来触发 ADC 采样。参见表 1.4 的描述。

函数 TimerControlEvent( )用于两种 16 位输入边沿捕获模式，可以控制有效的输入边沿，输入边沿有 3 种情况：正边沿、负边沿和双边沿。详见表 1.5 的描述。

函数 TimerControlLevel( )可以控制 Timer 在 16 位 PWM 模式下的方波有效输出电平是高电平还是低电平，即可以控制 PWM 方波反相输出。参见表 1.6 的描述。

表 1.3 函数 TimerControlStall( )

|    |                                                                                                                                                                                                                                                                                                                                                          |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 功能 | 控制 Timer 暂停运行（对 32 位 RTC 模式无效）                                                                                                                                                                                                                                                                                                                           |
| 原型 | void TimerControlStall(unsigned long ulBase, unsigned long ulTimer, tBoolean bStall)                                                                                                                                                                                                                                                                     |
| 参数 | <p>ulBase：Timer 模块的基址，取值 TIMERN_BASE（n 为 0、1、2 或 3）</p> <p>ulTimer：指定的 Timer，取值 TIMER_A、TIMER_B 或 TIMER_BOTH</p> <p>在 32 位模式下只能取值 TIMER_A，作为总体上的控制，而取值 TIMER_B 或 TIMER_BOTH 都无效。在 16 位模式下取值 TIMER_A 只对 TimerA 有效，取值 TIMER_B 只对 TimerB 有效，取值 TIMER_BOTH 同时对 TimerA 和 TimerB 有效。</p> <p>bStall：如果取值 true，则在单步调试模式下暂停计数</p> <p>如果取值 false，则在单步调试模式下继续计数</p> |
| 返回 | 无                                                                                                                                                                                                                                                                                                                                                        |

表 1.4 函数 TimerControlTrigger()

|    |                                                                                                                                                                     |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 功能 | 控制 Timer 的输出触发功能使能或禁止                                                                                                                                               |
| 原型 | void TimerControlTrigger(unsigned long ulBase, unsigned long ulTimer, tBoolean bEnable)                                                                             |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH<br>bEnable : 如果取值 true, 则使能输出触发<br>如果取值 false, 则禁止输出触发 |
| 返回 | 无                                                                                                                                                                   |

表 1.5 函数 TimerControlEvent()

|    |                                                                                                                                                                                                                                                                                                                                                                   |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 功能 | 控制 Timer 在捕获模式中的边沿事件类型                                                                                                                                                                                                                                                                                                                                            |
| 原型 | void TimerControlEvent(unsigned long ulBase, unsigned long ulTimer, unsigned long ulEvent)                                                                                                                                                                                                                                                                        |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH<br>ulEvent : 指定的边沿事件类型, 应当取下列值之一:<br>TIMER_EVENT_POS_EDGE       // 正边沿事件<br>TIMER_EVENT_NEG_EDGE       // 负边沿事件<br>TIMER_EVENT_BOTH_EDGES     // 双边沿事件 (正边沿和负边沿都有效)<br>注: 在 16 位输入边沿计数捕获模式下, 可以取值 3 种边沿事件的任何一种, 但在 16 位输入边沿定时模式下仅支持正边沿和负边沿, 不能支持双边沿。 |
| 返回 | 无                                                                                                                                                                                                                                                                                                                                                                 |

表 1.6 函数 TimerControlLevel()

|    |                                                                                                                                                                                      |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 功能 | 控制 Timer 在 PWM 模式下的有效输出电平                                                                                                                                                            |
| 原型 | void TimerControlLevel(unsigned long ulBase, unsigned long ulTimer, tBoolean bInvert)                                                                                                |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH<br>bInvert : 当取值 false 时 PWM 输出为高电平有效 (默认)<br>当取值 true 时输出低电平有效 (即输出反相) |
| 返回 | 无                                                                                                                                                                                    |

## 2. 计数值的装载与获取

函数 TimerLoadSet() 用来设置 Timer 的装载值。装载寄存器与计数器不同, 它是独立存在的。在调用 TimerEnable() 时会自动把装载值加载到计数器里, 以后每输入一个脉冲计数器值就加 1 或减 1 (取决于配置的工作模式), 而装载寄存器不变。另外, 除了单次触发定时器模式以外, 在计数器溢出时会自动重新加载装载值。函数 TimerLoadGet() 用来获取装载寄存器的值。参见表 1.7 和表 1.8。

函数 TimerValueGet() 用来获取当前 Timer 计数器的值。但在 16 位输入边沿定时捕获模式里, 获取的是捕获寄存器的值, 而非计数器值。参见表 1.9 的描述。

表 1.7 函数 TimerLoadSet()

|    |                                                                                                                                                               |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 功能 | 设置 Timer 的装载值                                                                                                                                                 |
| 原型 | void TimerLoadSet(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)                                                                         |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH<br>ulValue : 32 位装载值 (32 位模式) 或 16 位装载值 (16 位模式) |
| 返回 | 无                                                                                                                                                             |

表 1.8 函数 TimerLoadGet()

|    |                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------|
| 功能 | 获取 Timer 的装载值                                                                                                |
| 原型 | unsigned long TimerLoadGet(unsigned long ulBase, unsigned long ulTimer)                                      |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH |
| 返回 | 32 位装载值 (32 位模式) 或 16 位装载值 (16 位模式)                                                                          |

表 1.9 函数 TimerValueGet()

|    |                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------|
| 功能 | 获取当前的 Timer 计数值 (在 16 位输入边沿定时捕获模式下, 获取的是捕获值)                                                                 |
| 原型 | unsigned long TimerValueGet(unsigned long ulBase, unsigned long ulTimer)                                     |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH |
| 返回 | 当前 Timer 计数值 (在 16 位输入边沿定时捕获模式下, 返回的是捕获值)                                                                    |

### 3. 运行控制

函数 TimerEnable() 用来使能 Timer 计数器开始计数, 而函数 TimerDisable() 用来禁止计数。参见表 1.10 和表 1.11 的描述。

在 32 位 RTC 定时器模式下, 为了能够使 RTC 开始计数, 需要同时调用函数 TimerEnable() 和 TimerRTCEnable()。函数 TimerRTCEnable() 用于禁止 RTC 计数时。参见表 1.12 和表 1.13 的描述。

调用函数 TimerQuiesce() 可以复位 Timer 模块的所有配置。这为快速停止 Timer 工作或重新配置 Timer 为另外的工作模式提供了一种简便的手段。参见表 1.14 的描述。

表 1.10 函数 TimerEnable()

|    |                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------|
| 功能 | 使能 Timer 计数 (即启动 Timer)                                                                                      |
| 原型 | void TimerEnable(unsigned long ulBase, unsigned long ulTimer)                                                |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH |
| 返回 | 无                                                                                                            |

表 1.11 函数 TimerDisable()

|    |                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------|
| 功能 | 禁止 Timer 计数 (即关闭 Timer)                                                                                      |
| 原型 | void TimerDisable(unsigned long ulBase, unsigned long ulTimer)                                               |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH |
| 返回 | 无                                                                                                            |

表 1.12 函数 TimerRTCEnable()

|    |                                                      |
|----|------------------------------------------------------|
| 功能 | 使能 RTC 计数                                            |
| 原型 | void TimerRTCEnable(unsigned long ulBase)            |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3) |
| 返回 | 无                                                    |
| 说明 | 启动 RTC 时, 除了要调用本函数外, 还必须要调用函数 TimerEnable()          |

表 1.13 函数 TimerRTCDisable()

|    |                                                      |
|----|------------------------------------------------------|
| 功能 | 禁止 RTC 计数                                            |
| 原型 | void TimerRTCDisable(unsigned long ulBase)           |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3) |
| 返回 | 无                                                    |

表 1.14 函数 TimerQuiesce()

|    |                                                      |
|----|------------------------------------------------------|
| 功能 | 使 Timer 进入它的复位状态                                     |
| 原型 | void TimerQuiesce(unsigned long ulBase)              |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3) |
| 返回 | 无                                                    |

#### 4. 匹配与预分频

函数 TimerMatchSet() 和 TimerMatchGet() 用来设置和获取 Timer 匹配寄存器的值。Timer 开始运行后, 当计数器的值与预设的匹配值相等时可以触发某种动作, 如中断、捕获、PWM 等。参见表 1.15 和表 1.16 的描述。

在 Timer 的 16 位单次触发/周期定时器模式下, 输入到计数器的脉冲可以先经 8 位预分频器进行 1~256 分频, 这样, 16 位的定时器就被扩展成了 24 位。该功能是可选的, 预分频器默认值是 0, 即不分频。函数 TimerPrescaleSet() 和 TimerPrescaleGet() 用来设置和获取 8 位预分频器的值。参见表 1.17 和表 1.18 的描述。

表 1.15 函数 TimerMatchSet()

|    |                                                                                        |
|----|----------------------------------------------------------------------------------------|
| 功能 | 设置 Timer 的匹配值                                                                          |
| 原型 | void TimerMatchSet(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue) |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)                                   |

|    |                                                                                                    |
|----|----------------------------------------------------------------------------------------------------|
|    | ulTimer：指定的 Timer，取值 TIMER_A、TIMER_B 或 TIMER_BOTH<br>ulValue：32 位匹配值（32 位 RTC 模式）或 16 位匹配值（16 位模式） |
| 返回 | 无                                                                                                  |

表 1.16 函数 TimerMatchGet()

|    |                                                                                                       |
|----|-------------------------------------------------------------------------------------------------------|
| 功能 | 获取 Timer 的匹配值                                                                                         |
| 原型 | unsigned long TimerMatchGet(unsigned long ulBase, unsigned long ulTimer)                              |
| 参数 | ulBase：Timer 模块的基址，取值 TIMERN_BASE（n 为 0、1、2 或 3）<br>ulTimer：指定的 Timer，取值 TIMER_A、TIMER_B 或 TIMER_BOTH |
| 返回 | 32 位匹配值（32 位 RTC 模式）或 16 位匹配值（16 位模式）                                                                 |

表 1.17 函数 TimerPrescaleSet()

|    |                                                                                                                                                               |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 功能 | 设置 Timer 预分频值（仅对 16 位单次触发/周期定时模式有效）                                                                                                                           |
| 原型 | void TimerPrescaleSet(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)                                                                     |
| 参数 | ulBase：Timer 模块的基址，取值 TIMERN_BASE（n 为 0、1、2 或 3）<br>ulTimer：指定的 Timer，取值 TIMER_A、TIMER_B 或 TIMER_BOTH<br>ulValue：8 位预分频值（高 24 位无效），取值 0 ~ 255，对应的分频数是 1 ~ 256 |
| 返回 | 无                                                                                                                                                             |

表 1.18 函数 TimerPrescaleGet()

|    |                                                                                                       |
|----|-------------------------------------------------------------------------------------------------------|
| 功能 | 获取 Timer 预分频值（仅对 16 位单次触发/周期定时模式有效）                                                                   |
| 原型 | unsigned long TimerPrescaleGet(unsigned long ulBase, unsigned long ulTimer)                           |
| 参数 | ulBase：Timer 模块的基址，取值 TIMERN_BASE（n 为 0、1、2 或 3）<br>ulTimer：指定的 Timer，取值 TIMER_A、TIMER_B 或 TIMER_BOTH |
| 返回 | 8 位预分频值（高 24 位总是为 0）                                                                                  |

## 5. 中断控制

Timer 模块有多个中断源，有超时中断、匹配中断和捕获中断 3 大类，又细分为 7 种。函数 TimerIntEnable() 和 TimerIntDisable() 用来使能或禁止一个或多个 Timer 中断源。详见表 1.19 和表 1.20 的描述。

函数 TimerIntClear() 用来清除一个或多个 Timer 中断状态，函数 TimerIntStatus() 用来获取 Timer 的全部中断状态。在 Timer 中断服务函数里，这两个函数通常要配合使用。参见表 1.21 和表 1.22 的描述。

函数 TimerIntRegister() 和 TimerIntUnregister() 用来注册和注销 Timer 的中断服务函数，参见表 1.23 和表 1.24 的描述。

表 1.19 函数 TimerIntEnable()

|    |                                                                     |
|----|---------------------------------------------------------------------|
| 功能 | 使能 Timer 的中断                                                        |
| 原型 | void TimerIntEnable(unsigned long ulBase, unsigned long ulIntFlags) |

|    |                                                      |
|----|------------------------------------------------------|
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3) |
|    | ulIntFlags : 被使能的中断源, 应当取下列值之一或者它们之间的任意“或运算”组合形式:    |
|    | TIMER_TIMA_TIMEOUT // TimerA 超时中断                    |
|    | TIMER_CAPA_MATCH // TimerA 捕获模式匹配中断                  |
|    | TIMER_CAPA_EVENT // TimerA 捕获模式边沿事件中断                |
|    | TIMER_TIMB_TIMEOUT // TimerB 超时中断                    |
|    | TIMER_CAPB_MATCH // TimerB 捕获模式匹配中断                  |
| 返回 | TIMER_CAPB_EVENT // TimerB 捕获模式边沿事件中断                |
|    | TIMER_RTC_MATCH // RTC 匹配中断                          |
| 返回 | 无                                                    |

表 1.20 函数 TimerIntDisable()

|    |                                                                                                             |
|----|-------------------------------------------------------------------------------------------------------------|
| 功能 | 禁止 Timer 的中断                                                                                                |
| 原型 | void TimerIntDisable(unsigned long ulBase, unsigned long ulIntFlags)                                        |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulIntFlags : 被禁止的中断源, 取值与表 1.19 当中的参数 ulIntFlags 相同 |
| 返回 | 无                                                                                                           |

表 1.21 函数 TimerIntClear()

|    |                                                                                                             |
|----|-------------------------------------------------------------------------------------------------------------|
| 功能 | 清除 Timer 的中断                                                                                                |
| 原型 | void TimerIntClear(unsigned long ulBase, unsigned long ulIntFlags)                                          |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulIntFlags : 被清除的中断源, 取值与表 1.19 当中的参数 ulIntFlags 相同 |
| 返回 | 无                                                                                                           |

表 1.22 函数 TimerIntStatus()

|    |                                                                                                                           |
|----|---------------------------------------------------------------------------------------------------------------------------|
| 功能 | 获取当前 Timer 的中断状态                                                                                                          |
| 原型 | unsigned long TimerIntStatus(unsigned long ulBase, tBoolean bMasked)                                                      |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>bMasked : 如果需要获取的是原始的中断状态, 则取值 false<br>如果需要获取的是屏蔽的中断状态, 则取值 true |
| 返回 | 中断状态, 数值与表 1.19 当中的参数 ulIntFlags 相同                                                                                       |

表 1.23 函数 TimerIntRegister()

|    |                                                                                                                                                        |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| 功能 | 注册一个 Timer 的中断服务函数                                                                                                                                     |
| 原型 | void TimerIntRegister(unsigned long ulBase, unsigned long ulTimer, void (*pfnHandler)(void))                                                           |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH<br>pfnHandler : 函数指针, 指向 Timer 中断出现时调用的函数 |

|    |   |
|----|---|
| 返回 | 无 |
|----|---|

表 1.24 函数 TimerIntUnregister()

|    |                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------|
| 功能 | 注销 Timer 中断服务函数                                                                                              |
| 原型 | void TimerIntUnregister(unsigned long ulBase, unsigned long ulTimer)                                         |
| 参数 | ulBase : Timer 模块的基址, 取值 TIMERN_BASE (n 为 0、1、2 或 3)<br>ulTimer : 指定的 Timer, 取值 TIMER_A、TIMER_B 或 TIMER_BOTH |
| 返回 | 无                                                                                                            |

## 1.4 Timer 例程

表 1.25 Timer 例程索引

| 序号 | Timer 例程         | 页码                 |
|----|------------------|--------------------|
| 1  | 32 位单次触发定时       | <a href="#">11</a> |
| 2  | 32 位周期定时         | <a href="#">13</a> |
| 3  | 32 位 RTC 定时      | <a href="#">14</a> |
| 4  | 16 位单次触发定时 (预分频) | <a href="#">17</a> |
| 5  | 16 位周期定时 (预分频)   | <a href="#">19</a> |
| 6  | 16 位输入边沿计数捕获     | <a href="#">20</a> |
| 7  | 16 位输入边沿定时捕获     | <a href="#">22</a> |
| 8  | 16 位 PWM         | <a href="#">25</a> |
| 9  | PWM 应用: 蜂鸣器发声    | <a href="#">26</a> |
| 10 | PWM 应用: 蜂鸣器演奏乐曲  | <a href="#">29</a> |

### 1. 32 位单次触发定时

[程序清单 1.1](#) 是 Timer 模块 32 位单次触发定时器模式的例子。程序运行后, Timer 初始化为 32 位单次触发定时器, 并使能超时中断。在主循环里, 当检测到 KEY 按下时, 启动 Timer 定时 1.5 秒, 同时点亮 LED 以表示单次定时开始。当 Timer 倒计时到 0 时自动停止, 并触发超时中断。在中断服务函数里翻转 LED 亮灭状态, 由于原先 LED 是点亮的, 因此结果是 LED 熄灭。如果再次按下 KEY, 则再次点亮 LED 并定时 1.5 秒, 如此反复循环。如果不再按 KEY, 则 LED 一直保持熄灭状态, 说明 Timer 是单次触发的, 中断不会被触发, 因此在中断服务函数里的翻转 LED 操作也不会被执行到。

程序清单 1.1 Timer 例程: 32 位单次触发定时

文件: main.c

```
#include "systemInit.h"
#include <timer.h>

// 定义 LED
#define LED_PERIPH      SYSCTL_PERIPH_GPIOG
#define LED_PORT        GPIO_PORTG_BASE
```

```
#define LED_PIN                GPIO_PIN_2

// 定义 KEY
#define KEY_PERIPH             SYSCTL_PERIPH_GPIOD
#define KEY_PORT                GPIO_PORTD_BASE
#define KEY_PIN                GPIO_PIN_1

// 主函数（程序入口）
int main(void)
{
    jtagWait( );                // 防止 JTAG 失效，重要！
    clockInit( );               // 时钟初始化：晶振，6MHz

    SysCtlPeriEnable(LED_PERIPH); // 使能 LED 所在的 GPIO 端口
    GPIOPinTypeOut(LED_PORT, LED_PIN); // 设置 LED 所在的管脚为输出
    GPIOPinWrite(LED_PORT, LED_PIN, 1 << 2); // 熄灭 LED
    SysCtlPeriEnable(KEY_PERIPH); // 使能 KEY 所在的 GPIO 端口
    GPIOPinTypeIn(KEY_PORT, KEY_PIN); // 设置 KEY 所在管脚为输入

    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER0); // 使能 Timer 模块
    TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_OS); // 配置 Timer 为 32 位单次触发
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // 使能 Timer 超时中断
    IntEnable(INT_TIMER0A); // 使能 Timer 中断
    IntMasterEnable( ); // 使能处理器中断

    for (;;)
    {
        if (GPIOPinRead(KEY_PORT, KEY_PIN) == 0x00) // 如果复位时按下 KEY
        {
            SysCtlDelay(10 * (TheSysClock / 3000)); // 延时，消除按键抖动
            while (GPIOPinRead(KEY_PORT, KEY_PIN) == 0x00); // 等待按键抬起
            SysCtlDelay(10 * (TheSysClock / 3000)); // 延时，消除松键抖动

            TimerLoadSet(TIMER0_BASE, TIMER_A, 9000000); // 设置 Timer 初值，定时 1.5s
            TimerEnable(TIMER0_BASE, TIMER_A); // 使能 Timer 计数
            GPIOPinWrite(LED_PORT, LED_PIN, 0x00); // 点亮 LED，定时开始
        }
    }
}

// TimerA 的中断服务函数
void Timer0A_ISR(void)
{
    unsigned char ucVal;
```

```

unsigned long ulStatus;

ulStatus = TimerIntStatus(TIMER0_BASE, true);           // 获取当前中断状态
TimerIntClear(TIMER0_BASE, ulStatus);                   // 清除全部中断状态

if (ulStatus & TIMER_TIMA_TIMEOUT)                       // 如果是超时中断
{
    ucVal = GPIOPinRead(LED_PORT, LED_PIN);             // 反转 LED
    GPIOPinWrite(LED_PORT, LED_PIN, ~ucVal);
}
}

```

## 2. 32 位周期定时

**程序清单 1.2** 是 Timer 模块 32 位周期定时器模式的例子。程序运行后，配置 Timer 为 32 位周期定时器，定时 0.5 秒，并使能超时中断。当 Timer 倒计时到 0 时，自动重装初值，继续运行，并触发超时中断。在中断服务函数里翻转 LED 亮灭状态，因此程序运行的最后结果是 LED 指示灯每秒钟就会闪亮一次。

程序清单 1.2 Timer 例程：32 位周期定时

文件：main.c

```

#include "systemInit.h"
#include <timer.h>

// 定义 LED
#define LED_PERIPH          SYSCTL_PERIPH_GPIOG
#define LED_PORT            GPIO_PORTG_BASE
#define LED_PIN             GPIO_PIN_2

// 主函数（程序入口）
int main(void)
{
    jtagWait( );           // 防止 JTAG 失效，重要！
    clockInit( );          // 时钟初始化：晶振，6MHz

    SysCtlPeriEnable(LED_PERIPH);           // 使能 LED 所在的 GPIO 端口
    GPIOPinTypeOut(LED_PORT, LED_PIN);      // 设置 LED 所在管脚为输出

    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER0); // 使能 Timer 模块
    TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER); // 配置 Timer 为 32 位周期定时器
    TimerLoadSet(TIMER0_BASE, TIMER_A, 3000000UL); // 设置 Timer 初值，定时 500ms

    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // 使能 Timer 超时中断
    IntEnable(INT_TIMER0A);                       // 使能 Timer 中断
}

```

```

    IntMasterEnable( );                // 使能处理器中断
    TimerEnable(TIMER0_BASE, TIMER_A); // 使能 Timer 计数

    for (;;)
    {
    }
}

// 定时器的中断服务函数
void Timer0A_ISR(void)
{
    unsigned char ucVal;
    unsigned long ulStatus;

    ulStatus = TimerIntStatus(TIMER0_BASE, true); // 读取中断状态
    TimerIntClear(TIMER0_BASE, ulStatus);        // 清除中断状态，重要！

    if (ulStatus & TIMER_TIMA_TIMEOUT)           // 如果是 Timer 超时中断
    {
        ucVal = GPIOPinRead(LED_PORT, LED_PIN); // 反转 LED
        GPIOPinWrite(LED_PORT, LED_PIN, ~ucVal);
    }
}

```

### 3. 32 位 RTC 定时

**程序清单 1.3** 是 Timer 模块 32 位 RTC 定时器模式的例子。程序运行后，要求输入 RTC 初始时间，格式为“hh:mm:ss”，即“时:分:秒”。比如我们输入 9:59:45，然后回车<Enter>。在程序里，会把这个初始时间转换为以“秒”为单位的整数 ulVal，接着初始化 Timer 为 32 位 RTC 定时器，装载值为 ulVal 的数值，匹配值超前 1 秒钟。以后每经过 1 秒钟，RTC 就产生一次匹配中断。在中断服务函数里，重新设置匹配值，仍然超前 1 秒钟，计算并显示当前时间。最终我们通过 UART 会看到不断显示 09:59:45、09:59:46、……、09:59:59，接着会进位到 10:00:00，并继续运行。

**程序清单 1.3** 已在 EasyARM1138 开发板上调试通过。由于在该开发板上并没有提供 32.768KHz 时钟源，因此我们特意在程序里安排了一个 pulseInit() 函数，利用 6MHz 晶振提供的系统时钟在 PF6/CCP1 管脚产生一个接近于 32.768KHz 的 PWM 方波信号，标称频率为 32786.885Hz。实际做实验时，需要短接 PF6/CCP1 和 PF7/CCP4 管脚，CCP4 是 RTC 时钟源输入管脚。在其它开发板上，如 EasyARM615 或 EasyARM8962，提供有 32.768KHz 时钟源，则可以不必使用函数 pulseInit() 来产生 RTC 时钟源。

在**程序清单 1.3** 里，用到了一个新的头文件<pin\_map.h>，作用是提高程序的可移植性。在 Stellaris 系列 ARM 里，同一个外设的特定功能管脚在不同的型号里所对应的 GPIO 管脚位置可能并不相同，例如在 LM3S1138 里 CCP4 所在的 GPIO 是 PF7，而在 LM3S615 里 CCP4 在 PE2 上。为了能够使一个程序在不同型号上方便地进行移植，在《Stellaris 外设驱动库》里特意编排了一个<pin\_map.h>头文件，包含有全部型号的外设特定功能管脚的定义。如果

要在 LM3S1138 上运行程序，则首先#define PART\_LM3S1138，然后#include <pin\_map.h>，以后就可以采用 CCP4\_PERIPH、CCP4\_PORT、CCP4\_PIN 之类的定义。如果要程序移植到 LM3S615 上，则将宏定义 PART\_LM3S1138 改成 PART\_LM3S615 即可，而程序的其它部分不需要改动。

### 程序清单 1.3 Timer 例程：32 位 RTC 定时

文件：main.c

```
#include "systemInit.h"
#include "uartGetPut.h"
#include <timer.h>
#include <stdio.h>

#define PART_LM3S1138
#include <pin_map.h>

// 在 PF6/CCP1 管脚产生 32786.885Hz 方波，为 Timer2 的 RTC 功能提供时钟源
void pulseInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER0);           // 使能 TIMER0 模块
    SysCtlPeriEnable(CCP1_PERIPH);                     // 使能 CCP1 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP1_PORT, CCP1_PIN);             // 配置相关管脚为 Timer 功能

    TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR | // 配置 TimerB 为 16 位 PWM
                  TIMER_CFG_B_PWM);

    TimerLoadSet(TIMER0_BASE, TIMER_B, 183);           // 设置 TimerB 初值
    TimerMatchSet(TIMER0_BASE, TIMER_B, 92);           // 设置 TimerB 匹配值
    TimerEnable(TIMER0_BASE, TIMER_B);
}

// 定时器 RTC 功能初始化
void timerInitRTC(unsigned long ulVal)
{
    SysCtlPeriEnable(CCP4_PERIPH);                     // 使能 CCP4 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP4_PORT, CCP4_PIN);             // 配置 CCP4 管脚为 RTC 时钟输入

    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER2);           // 使能 Timer 模块
    TimerConfigure(TIMER2_BASE, TIMER_CFG_32_RTC);     // 配置 Timer 为 32 位 RTC 模式
    TimerLoadSet(TIMER2_BASE, TIMER_A, ulVal);         // 设置 RTC 计数器初值
    TimerMatchSet(TIMER2_BASE, TIMER_A, 1 + ulVal);    // 设置 RTC 匹配值

    TimerIntEnable(TIMER2_BASE, TIMER_RTC_MATCH);     // 使能 RTC 匹配中断
    IntEnable(INT_TIMER2A);                             // 使能 Timer 中断
    IntMasterEnable();                                  // 使能处理器中断
}
```

```

    TimerRTCEnable(TIMER2_BASE);                // 使能 RTC 计数
    TimerEnable(TIMER2_BASE, TIMER_A);           // 使能 Timer 计数
}

// 计算并显示 RTC 时钟
void timerDispRTC(unsigned long ulVal)
{
    char s[40];

    // 计算并显示小时
    if (ulVal / 3600 < 10) uartPutc('0');
    sprintf(s, "%ld:", ulVal / 3600);
    ulVal %= 3600;
    uartPuts(s);

    // 计算并显示分钟
    if (ulVal / 60 < 10) uartPutc('0');
    sprintf(s, "%ld:", ulVal / 60);
    ulVal %= 60;
    uartPuts(s);

    // 显示秒钟，并回车换行
    if (ulVal < 10) uartPutc('0');
    sprintf(s, "%ld\r\n", ulVal);
    uartPuts(s);
}

// 主函数（程序入口）
int main(void)
{
    unsigned long ulH, ulM, ulS;
    unsigned long ulVal;
    char s[40];

    jtagWait( );                                // 防止 JTAG 失效，重要！
    clockInit( );                               // 时钟初始化：晶振，6MHz
    uartInit( );                                // UART 初始化

    uartPuts("Please input the time (hh:mm:ss)\r\n"); // 提示输入时、分、秒
    uartGets(s, sizeof(s));                     // 从 UART 读取 RTC 初始时间
    sscanf(s, "%ld:%ld:%ld", &ulH, &ulM, &ulS); // 扫描输入时、分、秒
    ulVal = 3600 * ulH + 60 * ulM + ulS;         // 时分秒转换为 1 个整数
    ulVal %= 24 * 60 * 60;                       // 去掉“天”
    timerInitRTC(ulVal);                         // RTC 初始化
}

```

```

pulseInit( );                                // 开始提供 RTC 时钟源

ulVal = TimerValueGet(TIMER2_BASE, TIMER_A);    // 读取当前 RTC 计时器值
timerDispRTC(ulVal);                            // 显示初始时间

for (;;)
{
}

// Timer2 的中断服务函数
void Timer2A_ISR(void)
{
    unsigned long ulStatus;
    unsigned long ulVal;

    ulStatus = TimerIntStatus(TIMER2_BASE, true);
    TimerIntClear(TIMER2_BASE, ulStatus);

    if (ulStatus & TIMER_RTC_MATCH)
    {
        ulVal = TimerValueGet(TIMER2_BASE, TIMER_A);    // 读取当前 RTC 计时器值

        if (ulVal >= 24 * 60 * 60)                        // 若超过一天，则从 0 开始
        {
            ulVal = 0;
            TimerLoadSet(TIMER2_BASE, TIMER_A, 0);        // 重新设置 RTC 计数器初值
        }

        TimerMatchSet(TIMER2_BASE, TIMER_A, 1 + ulVal);    // 重新设置 RTC 匹配值
        timerDispRTC(ulVal);                            // 显示当前时间
    }
}

```

#### 4. 16 位单次触发定时

[程序清单 1.4](#) 是 Timer 模块 16 位单次触发定时器模式的例子。该例程实现的功能与 32 位单次触发定时的例程 ( 参见[程序清单 1.1](#) ) 基本相同，只是多了一个 8 位预分频器的运用。

程序清单 1.4 Timer 例程：16 位单次触发定时

文件：main.c

```

#include "systemInit.h"
#include <timer.h>

```

```
// 定义 LED
#define LED_PERIPH          SYSCTL_PERIPH_GPIOG
#define LED_PORT            GPIO_PORTG_BASE
#define LED_PIN             GPIO_PIN_2

// 定义 KEY
#define KEY_PERIPH          SYSCTL_PERIPH_GPIOD
#define KEY_PORT            GPIO_PORTD_BASE
#define KEY_PIN             GPIO_PIN_1

// 主函数（程序入口）
int main(void)
{
    jtagWait();                // 防止 JTAG 失效，重要！
    clockInit();              // 时钟初始化：晶振，6MHz

    SysCtlPeriEnable(LED_PERIPH);        // 使能 LED 所在的 GPIO 端口
    GPIOPinTypeOut(LED_PORT, LED_PIN);   // 设置 LED 所在的管脚为输出
    GPIOPinWrite(LED_PORT, LED_PIN, 1 << 2); // 熄灭 LED

    SysCtlPeriEnable(KEY_PERIPH);        // 使能 KEY 所在的 GPIO 端口
    GPIOPinTypeIn(KEY_PORT, KEY_PIN);    // 设置 KEY 所在管脚为输入

    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER0); // 使能 Timer 模块
    TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR | // 配置 Timer 为 16 位单次触发
                  TIMER_CFG_A_ONE_SHOT);

    TimerPrescaleSet(TIMER0_BASE, TIMER_A, 199); // 预先进行 200 分频
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // 使能 Timer 超时中断
    IntEnable(INT_TIMER0A); // 使能 Timer 中断
    IntMasterEnable(); // 使能处理器中断

    for (;;)
    {
        if (GPIOPinRead(KEY_PORT, KEY_PIN) == 0x00) // 如果复位时按下 KEY
        {
            SysCtlDelay(10 * (TheSysClock / 3000)); // 延时，消除按键抖动
            while (GPIOPinRead(KEY_PORT, KEY_PIN) == 0x00); // 等待按键抬起
            SysCtlDelay(10 * (TheSysClock / 3000)); // 延时，消除松键抖动

            TimerLoadSet(TIMER0_BASE, TIMER_A, 45000); // 设置 Timer 初值，定时 1.5s
            TimerEnable(TIMER0_BASE, TIMER_A); // 使能 Timer 计数
            GPIOPinWrite(LED_PORT, LED_PIN, 0x00); // 点亮 LED，定时开始
        }
    }
}
```

```

    }
}

// TimerA 的中断服务函数
void Timer0A_ISR(void)
{
    unsigned char ucVal;
    unsigned long ulStatus;

    ulStatus = TimerIntStatus(TIMER0_BASE, true);           // 获取当前中断状态
    TimerIntClear(TIMER0_BASE, ulStatus);                   // 清除中断状态，重要！

    if (ulStatus & TIMER_TIMA_TIMEOUT)                       // 如果是超时中断
    {
        ucVal = GPIOPinRead(LED_PORT, LED_PIN);            // 反转 LED
        GPIOPinWrite(LED_PORT, LED_PIN, ~ucVal);
    }
}

```

## 5. 16 位周期定时

[程序清单 1.5](#) 是 Timer 模块 16 位周期定时器模式的例子。该例程实现的功能与 32 位周期定时的例程（参见[程序清单 1.2](#)）基本相同，只是多了一个 8 位预分频器的运用。

程序清单 1.5 Timer 例程：16 位周期定时

文件：main.c

```

#include "systemInit.h"
#include <timer.h>

// 定义 LED
#define LED_PERIPH      SYSCTL_PERIPH_GPIOG
#define LED_PORT        GPIO_PORTG_BASE
#define LED_PIN         GPIO_PIN_2

// 主函数（程序入口）
int main(void)
{
    jtagWait( );           // 防止 JTAG 失效，重要！
    clockInit( );          // 时钟初始化：晶振，6MHz

    SysCtlPeriEnable(LED_PERIPH);           // 使能 LED 所在的 GPIO 端口
    GPIOPinTypeOut(LED_PORT, LED_PIN);      // 设置 LED 所在管脚为输出
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER0); // 使能 Timer 模块
}

```

```

    TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR |    // 配置 Timer 为 16 位周期定时器
                  TIMER_CFG_A_PERIODIC);

    TimerPrescaleSet(TIMER0_BASE, TIMER_A, 99);           // 预先进行 100 分频
    TimerLoadSet(TIMER0_BASE, TIMER_A, 30000);            // 设置 Timer 初值，定时 500ms

    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);      // 使能 Timer 超时中断
    IntEnable(INT_TIMER0A);                                // 使能 Timer 中断
    IntMasterEnable( );                                    // 使能处理器中断
    TimerEnable(TIMER0_BASE, TIMER_A);                    // 使能 Timer 计数

    for (;;)
    {
    }
}

// 定时器的中断服务函数
void Timer0A_ISR(void)
{
    unsigned char ucVal;
    unsigned long ulStatus;

    ulStatus = TimerIntStatus(TIMER0_BASE, true);          // 读取中断状态
    TimerIntClear(TIMER0_BASE, ulStatus);                  // 清除中断状态，重要！

    if (ulStatus & TIMER_TIMA_TIMEOUT)                     // 如果是 Timer 超时中断
    {
        ucVal = GPIOPinRead(LED_PORT, LED_PIN);           // 反转 LED
        GPIOPinWrite(LED_PORT, LED_PIN, ~ucVal);
    }
}

```

## 6. 16 位输入边沿计数捕获

**程序清单 1.6** 是 Timer 模块 16 位输入边沿计数捕获模式的例子。在程序中，利用函数 `pulseInit()` 产生 10KHz 的 PWM 方波，为边沿计数捕获模式提供时钟源。在 EasyARM1138 开发板上做实验时，需要短接 PF6/CCP1 和 PF7/CCP4 管脚。程序移植到其它开发板，如 EasyARM615 或 EasyARM8962，可以利用 32.768KHz 振荡器作为输入时钟源。

程序运行后，配置 Timer 为 16 位输入边沿计数捕获模式，设置计数初值和匹配值。启动 Timer 计数后，每从 CCP4 管脚输入一个脉冲，计数值就减 1，直到与匹配值相当时停止计数，并触发计数捕获中断。在中断服务函数里重新配置 Timer，并翻转 LED。

程序清单 1.6 Timer 例程：16 位输入边沿计数捕获

文件：main.c

```
#include "systemInit.h"
#include <timer.h>
#include <stdio.h>

// 定义 LED
#define LED_PERIPH          SYSCTL_PERIPH_GPIOG
#define LED_PORT            GPIO_PORTG_BASE
#define LED_PIN             GPIO_PIN_2

#define PART_LM3S1138
#include <pin_map.h>

// 在 CCP1 管脚产生 10KHz 方波，为 Timer2 的 16 位输入边沿计数捕获功能提供时钟源
void pulseInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER0);           // 使能 TIMER0 模块
    SysCtlPeriEnable(CCP1_PERIPH);                   // 使能 CCP1 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP1_PORT, CCP1_PIN);           // 配置相关管脚为 Timer 功能

    TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR | // 配置 TimerB 为 16 位 PWM
                  TIMER_CFG_B_PWM);

    TimerLoadSet(TIMER0_BASE, TIMER_B, 600);         // 设置 TimerB 初值
    TimerMatchSet(TIMER0_BASE, TIMER_B, 300);        // 设置 TimerB 匹配值
    TimerEnable(TIMER0_BASE, TIMER_B);
}

// 定时器 16 位输入边沿计数捕获功能初始化
void timerInitCapCount(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER2);           // 使能 Timer 模块
    SysCtlPeriEnable(CCP4_PERIPH);                   // 使能 CCP4 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP4_PORT, CCP4_PIN);           // 配置 CCP4 管脚为脉冲输入

    TimerConfigure(TIMER2_BASE, TIMER_CFG_16_BIT_PAIR | // 配置 Timer 为 16 位事件计数器
                  TIMER_CFG_A_CAP_COUNT);

    TimerControlEvent(TIMER2_BASE,                    // 控制 TimerA 捕获 CCP 负边沿
                     TIMER_A,
                     TIMER_EVENT_NEG_EDGE);

    TimerLoadSet(TIMER2_BASE, TIMER_A, 40000);        // 设置计数器初值
    TimerMatchSet(TIMER2_BASE, TIMER_A, 35000);       // 设置事件计数匹配值
    TimerIntEnable(TIMER2_BASE, TIMER_CAPA_MATCH);    // 使能 TimerA 捕获匹配中断
}
```

```

    IntEnable(INT_TIMER2A);                // 使能 Timer 中断
    IntMasterEnable();                     // 使能处理器中断
    TimerEnable(TIMER2_BASE, TIMER_A);     // 使能 Timer 计数
}

// 主函数（程序入口）
int main(void)
{
    jtagWait();                           // 防止 JTAG 失效，重要！
    clockInit();                           // 时钟初始化：晶振，6MHz

    SysCtlPeriEnable(LED_PERIPH);          // 使能 LED 所在的 GPIO 端口
    GPIOPinTypeOut(LED_PORT, LED_PIN);     // 设置 LED 所在管脚为输出

    pulseInit();

    timerInitCapCount();                   // Timer 初始化：16 位计数捕获

    for (;;)
    {
    }
}

// Timer2 的中断服务函数
void Timer2A_ISR(void)
{
    unsigned long ulStatus;
    unsigned char ucVal;

    ulStatus = TimerIntStatus(TIMER2_BASE, true); // 读取当前中断状态
    TimerIntClear(TIMER2_BASE, ulStatus);         // 清除中断状态，重要！

    if (ulStatus & TIMER_CAPA_MATCH)               // 若是 TimerA 捕获匹配中断
    {
        TimerLoadSet(TIMER2_BASE, TIMER_A, 40000); // 重新设置计数器初值
        TimerEnable(TIMER2_BASE, TIMER_A);         // TimerA 已停止，重新使能

        ucVal = GPIOPinRead(LED_PORT, LED_PIN);   // 反转 LED
        GPIOPinWrite(LED_PORT, LED_PIN, ~ucVal);

    }
}

```

## 7. 16 位输入边沿定时捕获

程序清单 1.7 是 Timer 模块 16 位输入边沿定时捕获模式的例子。同样采用 pulseInit( )

函数来产生捕获用的时钟源，频率为 1KHz。在 EasyARM1138 开发板上做实验时，需要短接 PF6/CCP1 和 PF7/CCP4 管脚。

程序运行后，配置 Timer 模块为 16 位输入边沿定时捕获模式。函数 pulseMeasure( ) 利用捕获功能测量输入到 CCP4 管脚的脉冲频率，结果通过 UART 显示。

程序清单 1.7 Timer 例程：16 位输入边沿定时捕获

文件：main.c

```
#include "systemInit.h"
#include "uartGetPut.h"
#include <timer.h>
#include <stdio.h>

#define PART_LM3S1138
#include <pin_map.h>

// 在 CCP1 管脚产生 1KHz 方波，为 Timer2 的 16 位输入边沿定时捕获功能提供时钟源
void pulseInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER0);           // 使能 TIMER0 模块
    SysCtlPeriEnable(CCP1_PERIPH);                     // 使能 CCP1 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP1_PORT, CCP1_PIN);             // 配置相关管脚为 Timer 功能

    TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR | // 配置 TimerB 为 16 位 PWM
                  TIMER_CFG_B_PWM);

    TimerLoadSet(TIMER0_BASE, TIMER_B, 6000);          // 设置 TimerB 初值
    TimerMatchSet(TIMER0_BASE, TIMER_B, 3000);          // 设置 TimerB 匹配值
    TimerEnable(TIMER0_BASE, TIMER_B);
}

// 定时器 16 位输入边沿定时捕获功能初始化
void timerInitCapTime(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER2);           // 使能 Timer 模块
    SysCtlPeriEnable(CCP4_PERIPH);                     // 使能 CCP4 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP4_PORT, CCP4_PIN);             // 配置 CCP4 管脚为脉冲输入

    TimerConfigure(TIMER2_BASE, TIMER_CFG_16_BIT_PAIR | // 配置 Timer 为 16 位事件定时器
                  TIMER_CFG_A_CAP_TIME);

    TimerControlEvent(TIMER2_BASE,                     // 控制 TimerA 捕获 CCP 正边沿
                      TIMER_A,
                      TIMER_EVENT_POS_EDGE);
}
```

```

    TimerControlStall(TIMER2_BASE, TIMER_A, true);           // 允许在调试时暂停定时器计数
    TimerIntEnable(TIMER2_BASE, TIMER_CAPA_EVENT);           // 使能 TimerA 事件捕获中断
    IntEnable(INT_TIMER2A);                                   // 使能 TimerA 中断
    IntMasterEnable( );                                       // 使能处理器中断
}

// 定义捕获标志
volatile tBoolean CAP_Flag = false;

// 测量输入脉冲频率并显示
void pulseMeasure(void)
{
    unsigned short i;
    unsigned short usVal[2];
    char s[40];

    TimerLoadSet(TIMER2_BASE, TIMER_A, 0xFFFF);           // 设置计数器初值
    TimerEnable(TIMER2_BASE, TIMER_A);                     // 使能 Timer 计数

    for (i = 0; i < 2; i++)
    {
        while (!CAP_Flag);                                   // 等待捕获输入脉冲
        CAP_Flag = false;                                    // 清除捕获标志
        usVal[i] = TimerValueGet(TIMER2_BASE, TIMER_A);     // 读取捕获值
    }

    TimerDisable(TIMER2_BASE, TIMER_A);                     // 禁止 Timer 计数
    sprintf(s, "%d Hz\r\n", (usVal[0] - usVal[1]) / 6);      // 输出测定的脉冲频率
    uartPuts(s);
}

// 主函数（程序入口）
int main(void)
{
    jtagWait( );                                             // 防止 JTAG 失效，重要！
    clockInit( );                                           // 时钟初始化：晶振，6MHz
    uartInit( );                                             // UART 初始化
    pulseInit( );
    timerInitCapTime( );                                     // Timer 初始化：16 位定时捕获

    for (;;)
    {
        pulseMeasure( );
        SysCtlDelay(1500 * (TheSysClock / 3000));
    }
}

```

```

    }
}

// Timer2 的中断服务函数
void Timer2A_ISR(void)
{
    unsigned long ulStatus;

    ulStatus = TimerIntStatus(TIMER2_BASE, true);           // 读取当前中断状态
    TimerIntClear(TIMER2_BASE, ulStatus);                   // 清除中断状态，重要！

    if (ulStatus & TIMER_CAPA_EVENT)                         // 若是 TimerA 事件捕获中断
    {
        CAP_Flag = true;                                     // 置位捕获标志
    }
}

```

## 8 . 16 位 PWM

**程序清单 1.8** 是 Timer 模块 16 位 PWM 模式的例子。程序运行后，配置 Timer 工作在双 16 位 PWM 模式下，设置的装载值决定 PWM 周期，设置的匹配值决定 PWM 占空比。最终 PWM 方波信号从 TimerA 和 TimerB 对应的两个 CCP 管脚输出。

程序清单 1.8 Timer 例程：16 位 PWM

文件：main.c

```

#include "systemInit.h"
#include <timer.h>

#define PART_LM3S1138
#include <pin_map.h>

// Timer 初始化为 16 位 PWM 模式
void timerInitPWM(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER1);                // 使能 Timer 模块
    SysCtlPeriEnable(CCP2_PERIPH);                          // 使能 CCP2 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP2_PORT, CCP2_PIN);                 // 配置 CCP2 管脚为 PWM 输出
    SysCtlPeriEnable(CCP3_PERIPH);                          // 使能 CCP3 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP3_PORT, CCP3_PIN);                 // 配置 CCP3 管脚为 PWM 输出

    TimerConfigure(TIMER1_BASE, TIMER_CFG_16_BIT_PAIR |    // 配置 Timer 为双 16 位 PWM
                  TIMER_CFG_A_PWM |
                  TIMER_CFG_B_PWM);
}

```

```

    TimerControlLevel(TIMER1_BASE, TIMER_BOTH, true);           // 控制 PWM 输出反相
    TimerLoadSet(TIMER1_BASE, TIMER_BOTH, 6000);                // 设置 TimerBoth 初值
    TimerMatchSet(TIMER1_BASE, TIMER_A, 3000);                  // 设置 TimerA 的 PWM 匹配值
    TimerMatchSet(TIMER1_BASE, TIMER_B, 2000);                  // 设置 TimerB 的 PWM 匹配值
    TimerEnable(TIMER1_BASE, TIMER_BOTH);                       // 使能 Timer 计数 ,PWM 开始输出
}

// 主函数（程序入口）
int main(void)
{
    jtagWait();                                                  // 防止 JTAG 失效，重要！
    clockInit();                                                  // 时钟初始化：晶振，6MHz
    timerInitPWM();                                              // Timer 的 PWM 功能初始化

    for (;;)
    {
    }
}

```

## 9. Timer PWM 应用：蜂鸣器发声

如图 1.1 所示，为 EasyARM1138 开发板上的蜂鸣器驱动电路。蜂鸣器类型是交流蜂鸣器，也称无源蜂鸣器，需要输入一系列方波才能鸣响，发声频率等于驱动方波的频率。

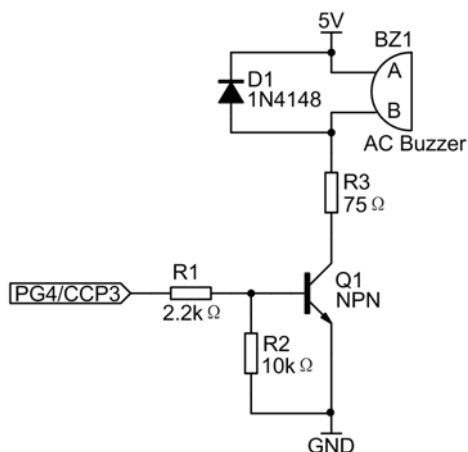


图 1.1 蜂鸣器驱动电路

程序清单 1.9 是 Timer 模块 16 位 PWM 模式的一个应用，可以驱动交流蜂鸣器发声，运行后蜂鸣器以不同的频率叫两声。其中"buzzer.h"和"buzzer.c"是蜂鸣器的驱动程序，仅有 3 个驱动函数，用起来很简捷。

程序清单 1.9 Timer PWM 应用：蜂鸣器发声

文件：main.c

```

#include "systemInit.h"
#include "buzzer.h"

```

```
// 主函数（程序入口）
int main(void)
{
    jtagWait( );                // 防止 JTAG 失效，重要！
    clockInit( );              // 时钟初始化：晶振，6MHz
    buzzerInit( );             // 蜂鸣器初始化

    buzzerSound(1500);          // 蜂鸣器发出 1500Hz 声音
    SysCtlDelay(400 * (TheSysClock / 3000)); // 延时约 400ms

    buzzerSound(2000);          // 蜂鸣器发出 2000Hz 声音
    SysCtlDelay(800 * (TheSysClock / 3000)); // 延时约 800ms

    buzzerQuiet( );            // 蜂鸣器静音

    for (;;)
    {
    }
}
```

文件：buzzer.h

```
#ifndef __BUZZER_H__
#define __BUZZER_H__

// 蜂鸣器初始化
extern void buzzerInit(void);

// 蜂鸣器发出指定频率的声音
extern void buzzerSound(unsigned short usFreq);

// 蜂鸣器停止发声
extern void buzzerQuiet(void);

#endif // __BUZZER_H__
```

文件：buzzer.c

```
#include "buzzer.h"
#include <hw_types.h>
#include <hw_memmap.h>
#include <sysctl.h>
#include <gpio.h>
#include <timer.h>
```

```
#define PART_LM3S1138
#include <pin_map.h>

#define SysCtlPeriEnable          SysCtlPeripheralEnable
#define GPIOPinTypeOut           GPIOPinTypeGPIOOutput

// 声明全局的系统时钟变量
extern unsigned long TheSysClock;

// 蜂鸣器初始化
void buzzerInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER1);           // 使能 TIMER1 模块
    SysCtlPeriEnable(CCP3_PERIPH);                     // 使能 CCP3 所在的 GPIO 端口
    GPIOPinTypeTimer(CCP3_PORT, CCP3_PIN);             // 设置相关管脚为 Timer 功能

    TimerConfigure(TIMER1_BASE, TIMER_CFG_16_BIT_PAIR | // 配置 TimerB 为 16 位 PWM
                  TIMER_CFG_B_PWM);
}

// 蜂鸣器发出指定频率的声音
// usFreq 是发声频率，取值 (系统时钟/65536)+1 ~ 20000，单位：Hz
void buzzerSound(unsigned short usFreq)
{
    unsigned long ulVal;

    if ((usFreq <= TheSysClock / 65536UL) || (usFreq > 20000))
    {
        buzzerQuiet( );
    }
    else
    {
        GPIOPinTypeTimer(CCP3_PORT, CCP3_PIN);         // 设置相关管脚为 Timer 功能
        ulVal = TheSysClock / usFreq;
        TimerLoadSet(TIMER1_BASE, TIMER_B, ulVal);     // 设置 TimerB 初值
        TimerMatchSet(TIMER1_BASE, TIMER_B, ulVal / 2); // 设置 TimerB 匹配值
        TimerEnable(TIMER1_BASE, TIMER_B);             // 使能 TimerB 计数
    }
}

// 蜂鸣器停止发声
void buzzerQuiet(void)
{
    TimerDisable(TIMER1_BASE, TIMER_B);                // 禁止 TimerB 计数
}
```

```
GPIOPinTypeOut(CCP3_PORT, CCP3_PIN);           // 配置 CCP3 管脚为 GPIO 输出
GPIOPinWrite(CCP3_PORT, CCP3_PIN, 0x00);        // 使 CCP3 管脚输出低电平
}
```

10 . Timer PWM 应用：蜂鸣器演奏乐曲

程序清单 1.10 是 Timer 模块 16 位 PWM 模式的一个应用，能驱动交流蜂鸣器演奏一首动听的乐曲《化蝶》(乐谱参见图 1.2)。“music.h”和“music.c”是演奏乐曲的驱动程序。

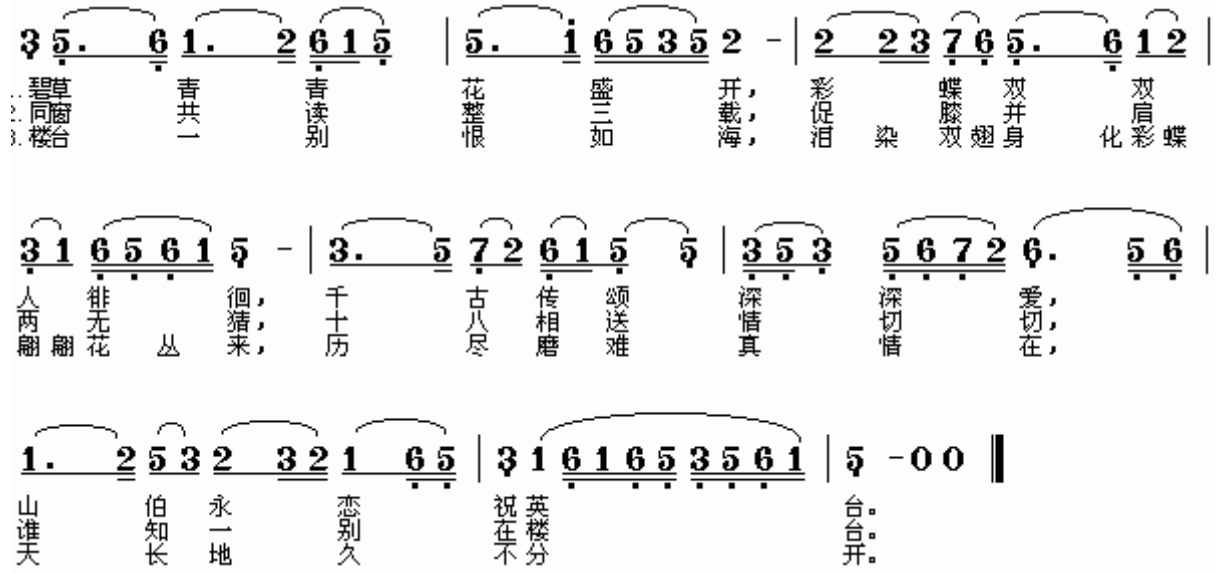


图 1.2 乐谱《化蝶》

简谱是大众化的音乐记谱方式，比较容易理解和掌握。我们可以把一首乐谱（score）看成是由若干个基本的音符（note）单元组成。一个音符由音名和时值组成。音名就是低音、中音、高音的 1234567（唱作 do re mi fa sol la si），其本质是音符的发声频率。在头文件“music.h”里，用 L1~L7、M1~M7、H1~H7 定义了低音、中音、高音所对应的发声频率。时值是音符的发声时间长短，有全音符、二分音符、四分音符……等等。音符可以后缀一个“符点”，表示时值增加 1/2，特殊地，二分音符加符点时用“-”代替圆点。参见表 1.26 的描述。

表 1.26 常见简谱音符示例

| 音 符             | 名 称    | 相 对 时 值  |
|-----------------|--------|----------|
| 5——             | 全音符    | T        |
| 5 -             | 二分音符   | T/2      |
| 5               | 四分音符   | T/4      |
| <u>5</u>        | 八分音符   | T/8      |
| <u><u>5</u></u> | 十六分音符  | T/16     |
| 5——.            | 符点二分音符 | T/2+T/4  |
| 5 .             | 符点四分音符 | T/4+T/8  |
| <u>5 .</u>      | 符点八分音符 | T/8+T/16 |

在头文件"music.h"里定义有一个音符结构体 tNote，有两个数据成员：音名 mName 和时值 mTime。在 C 文件"music.c"里定义有一个 tNote 型常量数表 MyScore[]，用来保存实际乐谱转换成 tNote 格式的数据。有了上述一点点乐谱基础知识，我们就可以很方便地编辑这个数表了。比如音符“3.”转换为“{L3, T/4}”，音符“3\_”转换为“{M3, T/4+T/8}”，等等。在"music.c"里，已经在数表 MyScore[]里给出了乐谱《化蝶》开头一部分音符转换结果，其余部分请感兴趣的读者补充完整。

#### 程序清单 1.10 Timer PWM 应用：蜂鸣器演奏乐曲

文件：main.c

```
#include "systemInit.h"
#include "buzzer.h"
#include "music.h"

// 主函数（程序入口）
int main(void)
{
    jtagWait();                // 防止 JTAG 失效，重要！
    clockInit();               // 时钟初始化：晶振，6MHz
    buzzerInit();              // 蜂鸣器初始化

    for (;;)
    {
        musicPlay();
        SysCtlDelay(4000 * (TheSysClock / 3000));
    }
}
```

文件：music.h

```
#ifndef __MUSIC_H__
#define __MUSIC_H__

// 定义低音音名（数值单位：Hz）
#define L1      262        // c
#define L2      294        // d
#define L3      330        // e
#define L4      349        // f
#define L5      392        // g
#define L6      440        // a1
#define L7      494        // b1

// 定义中音音名
#define M1      523        // c1
#define M2      587        // d1
#define M3      659        // e1
#define M4      698        // f1
```

```
#define M5      784      // g1
#define M6      880      // a2
#define M7      988      // b2
// 定义高音音名
#define H1      1047     // c2
#define H2      1175     // d2
#define H3      1319     // e2
#define H4      1397     // f2
#define H5      1568     // g2
#define H6      1760     // a3
#define H7      1976     // b3

// 定义时值单位，决定演奏速度（数值单位：ms）
#define T        3600

// 定义音符结构
typedef struct
{
    short mName;        // 音名：取值 L1 ~ L7、M1 ~ M7、H1 ~ H7 分别表示低音、中音、高音的
                        //      1234567，取值 0 表示休止符
    short mTime;        // 时值：取值 T、T/2、T/4、T/8、T/16、T/32 分别表示全音符、
                        //      二分音符、四分音符、八分音符...，取值 0 表示演奏结束
}tNote;

// 演奏乐曲
extern void musicPlay(void);

#endif // __MUSIC_H__
```

文件：music.c

```
#include "music.h"
#include "buzzer.h"
#include "systemInit.h"

// 定义乐曲：《化蝶》（梁祝）
const tNote MyScore[ ] =
{
    {L3, T/4},
    {L5, T/8+T/16},
    {L6, T/16},
    {M1, T/8+T/16},
    {M2, T/16},
    {L6, T/16},
    {M1, T/16},
```

```
{L5, T/8},

{M5, T/8+T/16},
{H1, T/16},
{M6, T/16},
{M5, T/16},
{M3, T/16},
{M5, T/16},
{M2, T/2},

// 省略后续乐曲数据，请感兴趣的读者补充完整

{ 0, 0}          // 结束
};

// 演奏乐曲
void musicPlay(void)
{
    short i = 0;

    for (;;)
    {
        if (MyScore[i].mTime == 0) break;

        buzzerSound(MyScore[i].mName);
        SysCtlDelay(MyScore[i].mTime * (TheSysClock / 3000));
        i++;

        buzzerQuiet( );
        SysCtlDelay(10 * (TheSysClock / 3000));
    }
}
```