第 2-10 讲：定时器/计数器

1. 学习目的

1. 了解定时器/计数器的概念和区别。
2. 掌握 STC8A8K64D4 定时器/计数器的应用流程及程序设计。

2. Timer 原理

定时器几乎是每个单片机必有的重要外设之一，可用于定时、精确延时、计数等等，在检测、控制领域有广泛应用。

Timer（本文中的 Timer 指的是定时/计数器）运行时不占用 CPU 时间，配置好之后，可以与 CPU 并行工作，实现精确的定时和计数，并且可以通过软件控制其是否产生中断，使用起来灵活方便。

讲解 STC8A8K64D4 的 Timer 之前，我们有必要先了解一下定时器和计数器的区别。

■ 定时器和计数器的区别：

定时器和计数器实际都是通过计数器来计数，定时器是对周期不变的脉冲计数（一般来自于系统时钟），由计数的个数和脉冲的周期即可计算出时间，同时，通过一个给定的预期值（即比较值，对应预期的计数值，也就是预期时间），当计数值达到预期值时产生中断，这样就实现了定时，应用程序通过设置不同的预期值实现不同时长的定时。

计数器是对某一事件进行计数，这个事件每发生一次，计数值加/减 1，而这个事件的产生可能是没有规律的。也就是计数器的用途是对事件的发生次数进行计数，由计数值来反映事件产生的次数。

从本质上来说，Timer 的核心部件是一个加法计数器，无论工作在定时器还是计数器模式，他们都是对脉冲信号进行计数。不同的是，定时器的脉冲信号来自于系统时钟，而计数器的脉冲信号来自于单片机特定外部输入引脚。

STC8A8K64D4 单片机片内共有 5 个 16 位定时器/计数器 T0~T4，这 5 个 16 位定时器均具有计数方式和定时方式两种工作方式。应用程序可通过特殊功能寄存器设置相应的 C/T 控制位，来选择 Timer 工作在哪种方式。

2.1. 定时器应用流程

Timer 工作于定时方式时的应用流程如下图所示，需要配置 Timer 为定时器并设置定时器速度为 1T 或 12T，这决定了定时器的计数速度。之后设置定时时间并启动定时器，当定时器溢出，即定时时间到达时，定时器溢出中断标志置位，如果开启了中断，则产生中断。

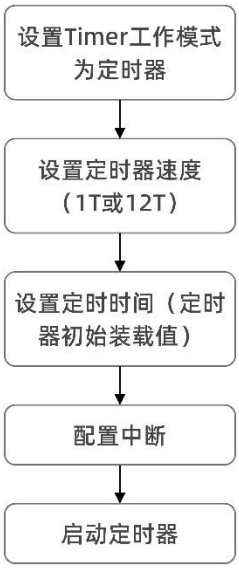

图 1：定时器应用流程

2.1.1. 设置 Timer 为定时器

5 个 Timer 可以通过相应的寄存器配置为定时器或计数器，配置方法如下表所示。

表 1：Timer 配置为定时器或计数器

Timer	配置方式
Timer0	由定时器 0/1 模式寄存器“TMOD”中的 T0_C/T 位配置。
Timer1	由定时器 0/1 模式寄存器“TMOD”中的 T1_C/T 位配置。
Timer2	由辅助寄存器 1“AUXR”中的 T2_C/T 位配置。
Timer3	由定时器 4/3 控制寄存器“T4T3M”中的 T3_C/T 位配置。
Timer4	由定时器 4/3 控制寄存器“T4T3M”中的 T4_C/T 位配置。

1. 配置定时器 0/1 为定时器或计数器：由“定时器 0/1 模式寄存器（TMOD）”寄存器中的 T0_C/T 和 T1_C/T 位控制。

定时器 0/1 模式寄存器（TMOD）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

- T1_C/T：控制定时器 1 用作定时器或计数器
  - 0：用作定时器（对内部系统时钟进行计数）。
  - 1：用作计数器（对引脚 T1/P3.5 外部脉冲进行计数）。
- T0_C/T：控制定时器 0 用作定时器或计数器
  - 0：用作定时器（对内部系统时钟进行计数）。
  - 1：用作计数器（对引脚 T0/P3.4 外部脉冲进行计数）。

2. 配置定时器 2 为定时器或计数器：由“辅助寄存器 1（AUXR）”寄存器中的 T2_C/T 位控制。

辅助寄存器 1（AUXR）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

- T2_C/T： 控制定时器 2 用作定时器或计数器
  - 0：用作定时器（对内部系统时钟进行计数）。
  - 1：用作计数器（对引脚 T2/P1.2 外部脉冲进行计数）。

3. 配置定时器 3/4 为定时器或计数器：由“定时器 4/3 控制寄存器（T4T3M）”寄存器中的 T3_C/T 和 T4_C/T 位控制。

定时器 4/3 控制寄存器（T4T3M）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

- T4_C/T：控制定时器 4 用作定时器或计数器
  - 0：用作定时器（对内部系统时钟进行计数）。
  - 1：用作计数器（对引脚 T4/P0.6 外部脉冲进行计数）。
- T3_C/T：控制定时器 3 用作定时器或计数器
  - 0：用作定时器（对内部系统时钟进行计数）。
  - 1：用作计数器（对引脚 T3/P0.4 外部脉冲进行计数）。

✍ **配置示例**：Timer0 配置为定时器

```
TMOD &= ~0x04;        //配置 Timer0 为定时器
```

✍ **配置示例**：Timer0 配置为计数器

```
TMOD |= 0x04;         //配置 Timer0 为计数器
```

2.1.2. 设置定时器的速度（1T/12T）

Timer 工作在定时器时，需要设置定时器的速度是 1T 还是 12T，12T 时定时器时钟是系统时钟/12，1T 时定时器时钟是系统时钟/1（不分频）。

5 个 Timer 的分频配置方式如下表所示。

表 2：Timer 分频配置（1T/12T）

Timer	配置方式
Timer0	工作在定时器时，由辅助寄存器 1“AUXR”中 T0x12 配置。
Timer1	工作在定时器时，由辅助寄存器 1“AUXR”中 T1x12 配置。
Timer2	工作在定时器时，由辅助寄存器 1“AUXR”中 T2x12 配置。
Timer3	工作在定时器时，由定时器 4/3 控制寄存器中 T3x12 配置。
Timer4	工作在定时器时，由定时器 4/3 控制寄存器中 T4x12 配置。

1. 定时器 0/1/2 的速度：由“辅助寄存器 1（AUXR）”寄存器中的 T0x12、T1x12 和

T2x12 位控制。

辅助寄存器 1（AUXR）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

- T0x12：定时器 0 速度控制位

  0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）。

  1：1T 模式，即 CPU 时钟不分频（FOSC/1）。

- T1x12：定时器 1 速度控制位

  0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）。

  1：1T 模式，即 CPU 时钟不分频 FOSC/1）。

- T2x12：定时器 2 速度控制位

  0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）。

  1：1T 模式，即 CPU 时钟不分频（FOSC/1）。

2. 定时器 3/4 的速度：由“定时器 4/3 控制寄存器（T4T3M）”寄存器中的 T0x12、T1x12 和 T2x12 位控制。

定时器 4/3 控制寄存器（T4T3M）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

- T3x12：定时器 0 速度控制位

  0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）。

  1：1T 模式，即 CPU 时钟不分频（FOSC/1）。

- T4x12：定时器 1 速度控制位

  0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）。

  1：1T 模式，即 CPU 时钟不分频 FOSC/1）。

✍ **配置示例**：Timer0 配置 1T

```
AUXR |= 0x80;          //定时器 0 时钟设置为 1T
```

2.1.3. 设置定时时间

对于定时器来说，最重要的部分是定时时间的设定。定时时间和系统时钟、定时器计数速度以及定时器工作模式密切相关。如下图所示，系统时钟、定时器计数速度和定时器工作模式这 3 项决定了定时器最大定时时间，我们配置的定时时间是不能大于这个时间的。

定时时间最终会换算为计数寄存器的初值，定时器启动后，从这个设置的初始值开始计数直到溢出这一段时间即为我们设置的定时时间。

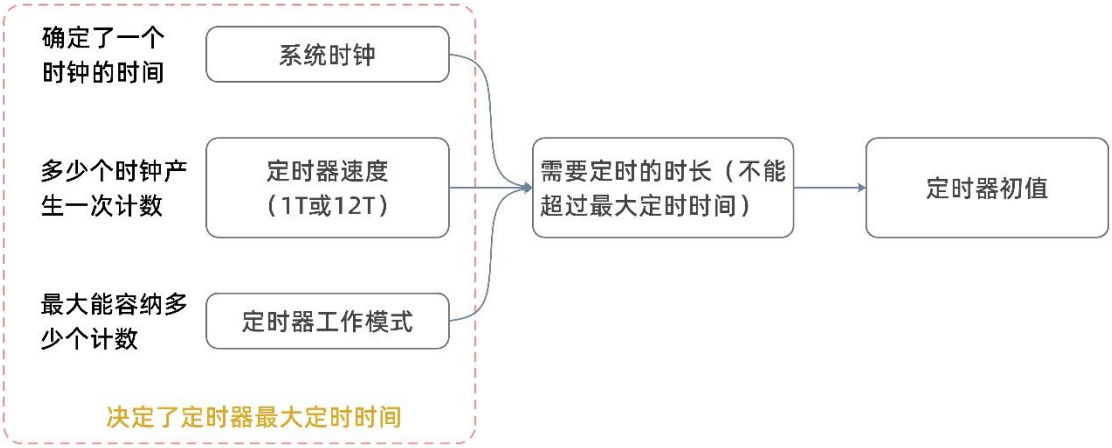

图 2：定时器定时时间

1) 系统时钟：系统时钟决定了一个时钟是多长时间。如系统时钟为 24MHz，那么一个时钟所需的时间是（1/24M）秒 = 1/24us。

2) 定时器计数速度（1T/12T）：计数速度决定了定时器多少个系统时钟产生一次计数，计数速度可以设置为 1T 或 12T 模式。

- 1T 模式下的计数速度和系统时钟一样，每个时钟计数一次，如系统时钟为 24MHz，则每个计数值对应的时间是（1/24M）秒 = 1/24us。
- 12T 模式下，每 12 个系统时钟，计数加 1，如系统时钟为 24MHz，则每个计数值对应的时间是（1/24M）×12 秒 = 1/2us。

3) 定时器工作模式

STC8A8K64D4 的 5 个定时器各自具有的模式如下表所示。这里面用的最多的是 16 位自动重装载模式，因此，推荐读者重点关注这种模式。本节以 Timer0 为例讲解 16 位自动重装载模式，至于其他几种模式，读者在熟悉了 16 位自动重装载模式的基础上很容易理解，如感兴趣，可阅读《STC8A8K64D4 系列单片机技术参考手册》中的“13：定时器/计数器”章节中相关的内容。

表 3：定时器工作模式

模式	T0	T1	T2	T3	T4
模式 0：16 位自动重装载模式	有	有	有	有	有
模式 1：16 位不可重装载模式	有	有			
模式 2：8 位自动重装载模式	有	有			
模式 3：不可屏蔽 16 位自动重装载模式	有				

定时器 0 的 16 位自动重装载模式原理框图如下所示。

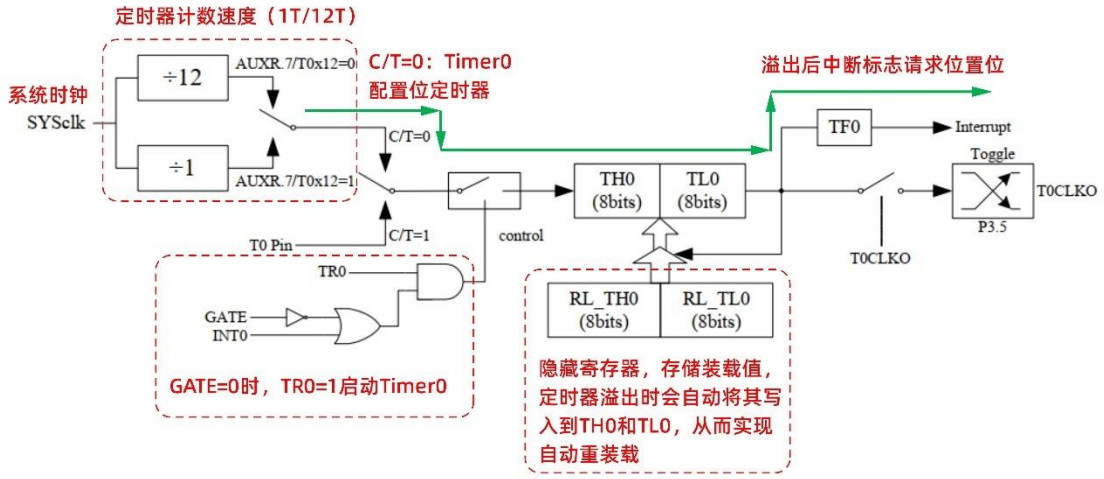

图 3：定时器 0 的 16 位自动重装载模式

1) 通过“辅助寄存器 1（AUXR）”寄存器中的 T0x12 位设置定时器计数速度（1T/12T）。
2) 设置定时器 0/1 模式寄存器“TMOD”中的 T0_C/T 为 0，将 Timer0 配置为定时器。
3) 当 GATE=0（TMOD.3）时，如 TR0=1，则启动定时器计数。（注：在程序中，经常会看到配置定时器时没有设置 GATE，不用奇怪，这是因为 GATE 复位值是 0，直接使用即可）。
4) 定时器 0 有两个隐藏的寄存器 RL_TH0 和 RL_TL0（程序员不可见），RL_TH0 与 TH0 共有同一个地址，RL_TL0 与 TL0 共有同一个地址。当 TR0=0 即 Timer0 未启动时，对 TL0 写入的内容会同时写入 RL_TL0，对 TH0 写入的内容也会同时写入 RL_TH0。当 TR0=1 即 Timer0 被允许工作时，对 TL0 写入内容，实际上不是写入当前寄存器 TL0 中，而是写入隐藏的寄存器 RL_TL0 中，对 TH0 写入内容，实际上也不是写入当前寄存器 TH0 中，而是写入隐藏的寄存器 RL_TH0，这样可以巧妙地实现 16 位重装载定时器。当读 TH0 和 TL0 的内容时，所读的内容就是 TH0 和 TL0 的内容，而不是 RL_TH0 和 RL_TL0 的内容。
5) 当定时器 0 溢出时不仅置位中断请求标志 TF0，而且会自动将隐藏寄存器 RL_TH0 和 RL_TL0 的内容重新装入 TH0 和 TL0。

定时器 0 在 16 位自动重装载模式下定时时间计算公式如下：

■ 当定时器 0 计数速度为 1T 时：

$$定时时间(S) = \frac{65536 - [TH0, TL0]}{SYSclk}$$

由此公式得到定时器 0 初值（自动装载的值）的计算公式为：

$$[TH0, TL0] = 65536 - 定时时间(S) \times SYSclk$$

✍ **计算示例**：在系统时钟为 24MHz 时，定时时间为 1ms，定时器 0 计数初值计算：

$$[TH0, TL0] = 65536 - 10^{-3} \times 24 \times 10^{6}$$

$$= 65536 - 24000$$

$$= 41536$$

41536 转换为 16 进制为：0xA240，因此 TH0=0xA2，TL0=0x40。

■ 当定时器 0 计数速度为 12T 时：

$$定时时间(S) = \frac{65536 - [TH0, TL0]}{SYSclk} \times 12$$

由此公式得到定时器 0 初值（自动装载的值）的计算公式为：

$$[TH0, TL0] = 65536 - \frac{定时时间(S) \times SYSclk}{12}$$

对于定时器定时时间的计算，在开发过程中使用宏晶科技发布的定时器计算工具更为方便，该工具具体的使用步骤如下。

1) 打开 STC-ISP 软件后，依次点击“工具→独立使用波特率计算工具(B)”，打开波特率计算工具。

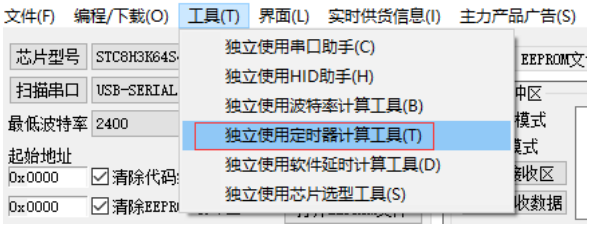

图 4：打开定时器计算工具

2) 选择相关参数后，点击“生成 C 代码”即可获取串口初始化代码，同时也可以看到该配置下定时器的误差。

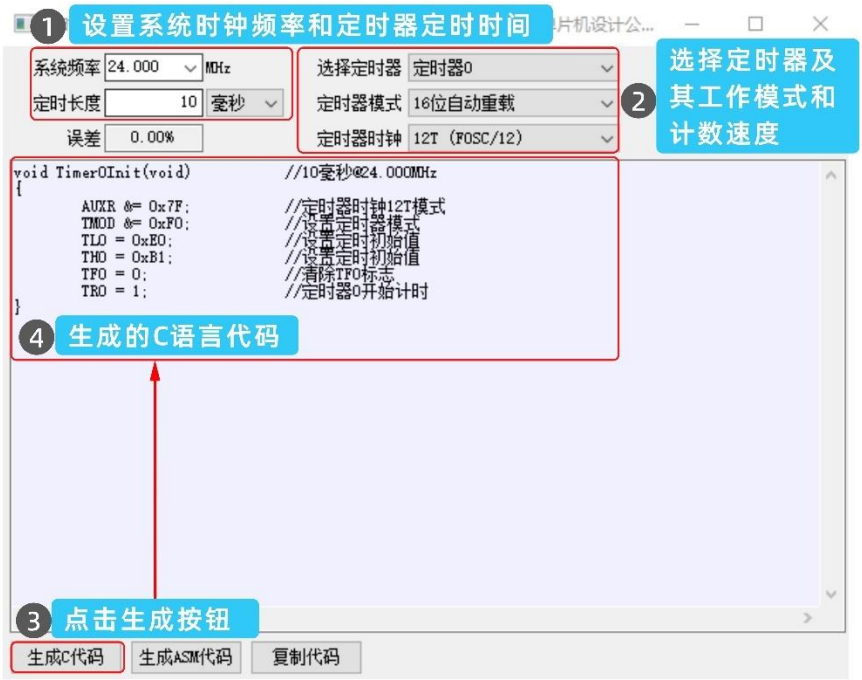

图 5：定时器计算

2.1.4. 中断配置

中断配置包括中断的开启/关闭和中断优先级的配置。

Timer0 和 Timer1 的中断的开启和关闭由中断使能寄存器 IE 的位 1（ET0）和位 3（ET1）控制，如下图所示。

IE（中断使能寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

总中断允许位　Timer1溢出中断允许位　Timer0溢出中断允许位

- ET1：定时/计数器 T1 的溢出中断允许位。
  - 0：禁止 T1 中断。
  - 1：允许 T1 中断。

- ET0：定时/计数器 T0 的溢出中断允许位。
  - 0：禁止 T0 中断。
  - 1：允许 T0 中断。

Timer2、Timer3 和 Timer4 中断的开启和关闭由中断使能寄存器 IE2 的位 2（ET2）、位 5（ET3）和位 6（ET4）控制，如下图所示。

IE2（中断使能寄存器 2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	EUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

Timer4溢出中断允许位　Timer3溢出中断允许位　Timer2溢出中断允许位

- ET4：定时/计数器 T4 的溢出中断允许位。
  - 0：禁止 T4 中断。
  - 1：允许 T4 中断。
- ET3：定时/计数器 T3 的溢出中断允许位。
  - 0：禁止 T3 中断。
  - 1：允许 T3 中断。
- ET2：定时/计数器 T2 的溢出中断允许位。
  - 0：禁止 T2 中断。
  - 1：允许 T3 中断。

5 个 Timer 中，定时器 0 和定时器 1 有 4 级中断优先级可设置，定时器 2、定时器 3 和定时器 4 中断优先级是固定的，只能是最低优先级 0。

表 4：Timer 中断优先级

Timer	中断号	说明
Timer 0	1	中断优先级可配置为 0、1、2、3。
Timer 1	3	中断优先级可配置为 0、1、2、3。
Timer 2	12	中断优先级只能为最低优先级 0。
Timer 3	19	中断优先级只能为最低优先级 0。
Timer 4	20	中断优先级只能为最低优先级 0。

定时器 0 的中断优先级由 IP 和 IPH 寄存器的位 1 的组合配置，定时器 1 的中断优先级由 IP 和 IPH 寄存器的位 3 的组合配置，如下图所示。

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000

Timer1中断优先级配置位　　Timer0中断优先级配置位

图 6：Timer0 和 Timer1 中断优先级配置

■ PT0H，PT0：定时器 0 中断优先级控制位
- 00：定时器 0 中断优先级为 0 级（最低级）。
- 01：定时器 0 中断优先级为 1 级（较低级）。
- 10：定时器 0 中断优先级为 2 级（较高级）。
- 11：定时器 0 中断优先级为 3 级（最高级）。

■ PT1H，PT1：定时器 1 中断优先级控制位
- 00：定时器 1 中断优先级为 0 级（最低级）。
- 01：定时器 1 中断优先级为 1 级（较低级）。
- 10：定时器 1 中断优先级为 2 级（较高级）。
- 11：定时器 1 中断优先级为 3 级（最高级）。

✍ **中断优先级配置示例**：配置 Timer0 中断优先级为最高优先级 3

```
IP  |= 0x02;  //中断优先级配置为最高优先级：PT0H = 1，PT0 = 1
IPH |= 0x02;
```

✧ **注意事项**：Timer 开启中断的情况下，还需要开启总中断“EA=1”，中断才能起作用。通常，我们会在主函数“main()”中开启总中断，这是因为我们开发的程序中可能会使用到多个中断，在主函数中开启一次即可。

2.1.5. 启动和停止定时器

Timer 可以配置为定时器或计数器，无论在哪种模式下，Timer0 都是通过置位“定时器 0/1 控制寄存器（TCON）”中的“TR0 位”启动，通过清零“TR0 位”停止，启动后，

定时器将继续从他之前被停止时的值继续计数。

1) 定时器 0/1 的启动和停止：由 TCON 寄存器中的 TR0 和 TR1 位控制。

定时器 0/1 控制寄存器（TCON）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- TR1：定时器 T1 的运行控制位，该位由软件置位和清零。当 GATE（TMOD.7）=0，TR1=1 时就允许 T1 开始计数，TR1=0 时禁止 T1 计数。当 GATE（TMOD.7）=1，TR1=1 且 INT1 输入高电平时，才允许 T1 计数。
- TR0：定时器 T0 的运行控制位，该位由软件置位和清零。当 GATE（TMOD.3）=0，TR0=1 时就允许 T0 开始计数，TR0=0 时禁止 T0 计数。当 GATE（TMOD.3）=1，TR0=1 且 INT0 输入高电平时，才允许 T0 计数，TR0=0 时禁止 T0 计数。

2) 定时器 2 的启动和停止：由“辅助寄存器 1（AUXR）”寄存器中的 T2R 位控制。

辅助寄存器 1（AUXR）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

- T2R：定时器 2 的运行控制位。

  0：定时器 2 停止计数。

  1：定时器 2 开始计数。

3) 定时器 3/4 的启动和停止：由“定时器 4/3 控制寄存器（T4T3M）”寄存器中的 T4R 和 T3R 位控制。

定时器 4/3 控制寄存器（T4T3M）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

- T4R：定时器 4 的运行控制位。

  0：定时器 4 停止计数。

  1：定时器 4 开始计数。

- T3R：定时器 3 的运行控制位

  0：定时器 3 停止计数。

  1：定时器 3 开始计数。

✍ **示例**：启动定时器 0 计数

```
TR0 = 1;        //启动定时器 0 计数
```

✍ **示例**：停止定时器 0 计数

```
TR0 = 0;        //停止定时器 0 计数
```

2.2. 计数器应用流程

Timer 工作于计数器时的应用流程如下图所示。Timer 配置为计数器之后，对应的引脚会连接到计数器，因此需要配置引脚的上拉电阻（如果没有外部上拉），之后设置计数寄存器的初值并根据需要开启中断，配置完成后，启动计数器即可。

这里要注意一下，计数器是对外部引脚的脉冲计数，他不会对外部脉冲分频的，所以不需要设置定时器的速度的（1T/12T）。

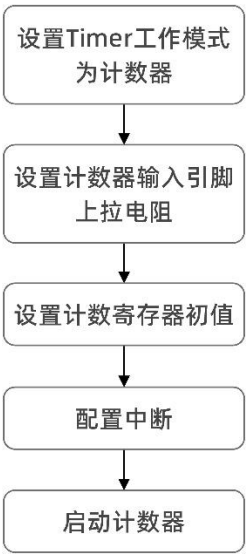

图 7：计数器应用流程

2.2.1. 设置 Timer 为计数器

Timer 设置为计数器和设置为定时器所用到的寄存器完全一样，读者参见《设置 Timer 为定时器》这一节即可，区别是设置为定时器是将对应寄存器的位清零（=0），设置为计数器是将对应寄存器的位置位（=1）。

Timer 设置为计数器后，不再对系统时钟进行计数，而是对特定的引脚的输入脉冲进行计数。Timer0~ Timer5 设置为计数器后，对应的外部输入引脚如下表所示，这些引脚都是固定的，无需软件配置的。如 Timer0 配置为计数器后，P3.4 即为其外部输入引脚，而且也只有 P3.4 能作为他的外部输入引脚。

表 5：5 个 Timer 的外部输入引脚

Timer	外部输入引脚
Timer0	P3.4
Timer1	P3.5
Timer2	P1.2
Timer3	P0.4
Timer4	P0.6

2.2.2. 设置计数器输入引脚上拉电阻

这一步是视具体情况而定的，如果 Timer 的外部输入引脚已经在片外设计了上拉电阻，或者连接到外部输入引脚的电路有足够驱动能力保证电平的稳定，则无需开启 IO 端口的内部上拉。否则，需要打开 IO 端口的内部上拉，以防止因电平抖动而引起计数器误计数。

开启 Timer 外部输入引脚的片内上拉电阻也就是开启对应的 GPIO 的片内上拉电阻，

GPIO 上拉电阻的配置方法我们在《第 2-3 讲：按键检测》中已经讲解过，读者可以回顾一下。下面的代码是开启 Timer3 外部输入引脚 P0.4 的片内上拉电阻。

代码清单：开启 P3.4 的内部上拉电阻

```
1.  P_SW2 |= 0x80;              //将 EAXFR 位置 1，允许访问扩展 RAM 区特殊功能寄存器(XFR)
2.  P0PU |= 0x10;               //开启 P0.4 的上拉电阻
3.  P_SW2 &= 0x7F;              //将 EAXFR 位置 0，禁止访问 XFR
```

2.2.3. 设置计数器初值和配置中断

Timer 在计数器模式下，对应的外部输入引脚每输入一个脉冲，计数器值加 1，如果开启了中断，当计数器溢出时产生中断。因此，计数器中断和初值配置是相关的，下面两种配置是常用的两种场景。

1) 只想得到外部脉冲的计数：这时可以设置计数器初值为 0，不用开启中断，当需要获取计数值时读取 Timer 的计数寄存器即可。
2) 每个外部脉冲都产生一次中断：这时可以配置计数器初值为 0xFFFF，开启中断，外部输入引脚，输入一个脉冲后计数器溢出产生中断。

2.2.4. 启动和停止计数器

Timer 工作在计数器时，启动和停止的方法和工作在定时器时完全一样，读者可以参见定时器的启动和停止这一部分。

3. 软件设计

本小节我们通过 Timer0 配置为定时器时的定时实验和配置为计数器时对外部脉冲计数的实验来讲解 Timer0 的软件编程。

3.1. Timer 定时实验

✧ 注：本节的实验是在“实验 2-6-1：串口 1 数据收发实验”的基础上修改，本节对应的实验源码是：“实验 2-10-1：Timer0 定时驱动 LED 闪烁”。

3.1.1. 实验内容

配置 Timer0 工作于定时器，定时时间为设置 200ms，即每 200ms 产生一次溢出中断，Timer0 中断服务函数中翻转指示灯 D1 的状态，从而达到驱动指示灯 D1 以 200ms 间隔闪烁的目的。

通过按键 KEY1 和 KEY2 启动和停止定时器，KEY1 按下后，Timer0 启动，指示灯 D1 开始以 200ms 的间隔闪烁。KEY2 按下后，Timer0 停止，指示灯 D1 停止闪烁。

3.1.2. 代码编写

1. 新建一个名称为“timer.c”的文件及其头文件“timer.h”并保存到工程的“Source”文件夹，并将“timer.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“timer.c”文件中的函数，所以需要引用下面的头文件“timer.h”。

代码清单：引用头文件

```
1.  //引用 timer 的头文件
2.  #include    " timer.h"
```

3. 定时器初始化

设置 Timer 为定时器、设置定时器速度、定时时间以及中断配置均属于定时器初始化部分，我们将这些代码封装到定时器初始化函数 timer0_init()中，代码清单如下，本例中设置的定时时间是 10ms。

代码清单：定时器 0 初始化函数

```
1.  /*********************************************************************************
2.  功能描述：初始化 Timer0（系统时钟使用 24MHz），定时时间 10ms,12T
3.  参    数：无
4.  返 回 值：无
5.  *********************************************************************************/
6.  void timer0_init(void)
7.  {
8.      AUXR &= 0x7F;        //定时器时钟 12T 模式
9.      TMOD &= 0xF0;        //配置 Timer0 为定时器
10.     TL0 = 0xE0;          //设置定时初始值
11.     TH0 = 0xB1;          //设置定时初始值
12.     IP |= 0x02;          //中断优先级配置为最高优先级
13.     IPH |= 0x02;
14.     //使能定时器 0 中断，注意：开启定时器 0 中断的情况下，还需要开启总中断“EA=1”，中断才能起作用
15.     ET0 = 1;
16. }
```

4. 启动和停止 Timer0

Timer0 通过置位“定时器 0/1 控制寄存器（TCON）”中的“TR0 位”启动，为了方便其他程序调用，我们将启动和停止封装为函数，代码清单如下。

代码清单：定时器 0 启动和停止函数

```
1.  void timer0_start(void)
2.  {
3.      TR0 = 1;             //启动定时器 0
4.  }
5.
6.  void timer0_stop(void)
7.  {
8.      TR0 = 0;             //停止定时器 0
9.  }
```

5. Timer0 中断服务函数

因为 Timer0 配置的定时时间是 10ms，为了达到 200ms 的定时，我们定义一个变量“count”来记录 Timer0 的溢出次数，当“count”的值等于 10 的时候，表示达到 200ms，此时，翻转指示灯 D1 的状态，从而实现驱动 D1 以 200ms 间隔闪烁的目的。

代码清单：Timer0 中断服务函数中

```
/*******************************************************************************
 * 描   述 : 定时器中断服务函数
 * 入   参 : 无
 * 返回值 : 无
 *******************************************************************************/
void timer0_isr() interrupt 1
{
  count++;
  if(count == 20)        //200ms
  {
     led_toggle(LED_1);  //翻转指示灯 D1 状态
     count = 0;          //计数清零
   }
}
```

6. 主函数

主函数中初始化指示灯、按键和定时器并开启总中断，之后在主循环里面读取按键 KEY1 和 KEY2 状态，若 KEY1 按下，启动 Timer0，若 KEY2 按下，停止 Timer0，代码清单如下。

代码清单：主函数

```
/*******************************************************************************
功能描述：主函数
入口参数：无
返回值：int 类型
 *******************************************************************************/
int main(void)
{
   u8 temp;

   P0M1 &= 0xF0;   P0M0 &= 0xF0;     //设置 P0.3~P0.0 为准双向口（LED）
   P3M1 &= 0x3F;   P3M0 &= 0x3F;     //设置 P3.6 和 P3.7 为准双向口（按键 S3、S4）
   P4M1 &= 0xEF;   P4M0 &= 0xEF;     //设置 P4.4 为准双向口（按键 S5）

   P_SW2 |= 0x80;                  //将 EAXFR 位置 0，禁止访问 XFR
   P4PU |= 0x10;                   //开启 P4.4 的上拉电阻（按键 S5）
   P3PU |= 0xC0;                   //开启 P3.6 和 P3.7 的上拉电阻（按键 S3、S4）
   P_SW2 &= 0x7F;                  //将 EAXFR 位置 0，禁止访问 XFR

   timer0_init();                //定时器 0 初始化
```

```
20.     EA = 1;                         //使能总中断
21.
22.     while(1)
23.     {
24.       temp=buttons_scan(0);             //获取开发板用户按键状态，不支持连按
25.       if(temp == BUTTON1_PRESSED)       //按键 KEY1 按下
26.       {
27.         timer0_start();                 //启动定时器 0
28.       }
29.       else if(temp == BUTTON2_PRESSED)  //按键 KEY2 按下
30.       {
31.         timer0_stop();                  //停止定时器 0
32.       }
33.     }
34. }
```

3.1.3. 硬件连接

本实验需要使用 LED 指示灯 D1 和按键 KEY1、KEY2，按照下图所示短接指示灯和按键的跳线帽。

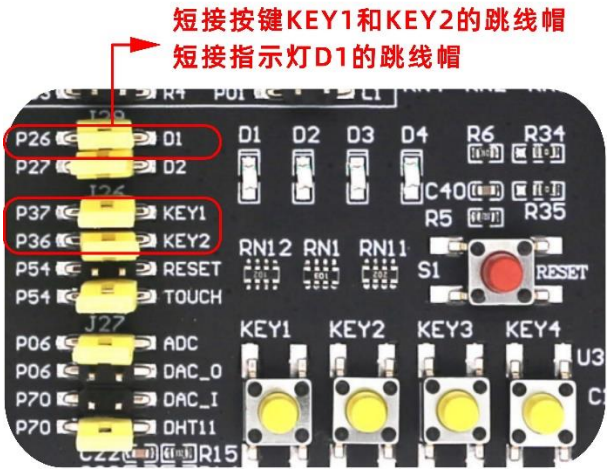

图 8：跳线帽短接

3.1.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-10-1：Timer0 定时驱动 LED 闪烁”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\timer0_timer\project”目录下的工程文件“timer.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“timer.hex”位于工程的“…\timer0_timer\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 程序运行后，按下按键 KEY1，可以观察到指示灯 D1 开始以 200ms 间隔闪烁，说明

此时 Timer0 已经启动并工作在定时器模式。按下按键 KEY2，可以观察到指示灯 D1 停止闪烁，说明此时 Timer0 已经停止。

✧ **说明：**Timer2、Timer3 和 Timer4，他们的操作和 Timer0 类似，读者可以尝试在学习了 Timer0 的基础上用他们来实现本例的功能。

我们也编写好了 Timer2、Timer3 和 Timer4 工作于定时器模式的例子，这些例子在资料的“…\第 3 部分：配套例程源码”目录下，他们的实验名称如下，读者在编写的过程中可以参考。

- 实验 2-10-2：Timer1 定时驱动 LED 闪烁。
- 实验 2-10-3：Timer2 定时驱动 LED 闪烁。
- 实验 2-10-4：Timer3 定时驱动 LED 闪烁。
- 实验 2-10-5：Timer4 定时驱动 LED 闪烁。

3.2. Timer 计数实验

✧ 注：本节的实验是在“实验 2-6-1：串口 1 数据收发实验”的基础上修改，本节对应的实验源码是：“实验 2-10-6：Timer0 对外部输入脉冲计数”。

3.2.1. 实验内容

配置 Timer0 工作于定时器，对 P3.4 输入的脉冲进行计数，程序中每秒读取一次计数值并通过串口输出。

为了方便测试，开发上用杜邦线将 P3.4 连接到按键 KEY1，通过按动按键 KEY1 产生脉冲，即每按动一次 KEY1，计数器的计数值加 1。

3.2.2. 代码编写

1. 新建一个名称为“timer.c”的文件及其头文件“timer.h”并保存到工程的“Source”文件夹，并将“timer.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“timer.c”文件中的函数，所以需要引用下面的头文件“timer.h”。

代码清单：引用头文件

```
1.  //引用 timer 的头文件
2.  #include    " timer.h"
```

3. 计数器初始化

将 Timer0 配置为计数器后，P3.4 即为其外部输入引脚。因为开发板上 P3.4 没有外部上拉电阻，为了保证 P3.4 上电平的稳定，需要开启 P3.4 的内部上拉电阻。

本例中，我们是对外部输入脉冲计数，因此，计数器初值设置为 0，并且不需要开启中断，计数器初始化代码清单如下。

代码清单：计数器初始化

```
1.  /*********************************************************************************
```

```
2.  功能描述：初始化 Timer0 为计数器，开启 P3.4 的内部上拉电阻
3.  参    数：无
4.  返 回 值：无
5.  ******************************************************************************/
6.  void timer0_counter_init(void)
7.  {
8.      P_SW2 |= 0x80;     //将 EAXFR 位置 0，禁止访问 XFR
9.      P3PU |= 0x10;      //开启 P3.4 的上拉电阻
10.     P_SW2 &= 0x7F;     //将 EAXFR 位置 0，禁止访问 XFR
11.
12.     TMOD = 0x04;       //配置 Timer0 为计数器
13.     TL0 = 0x00;        //计数器初始值设置为 0
14.     TH0 = 0x00;        //计数器初始值设置为 0
15.     TF0 = 0;           //清除 TF0 标志
16. }
```

4. 读取计数值

当我们需要读取计数值的时候，直接从“TH0”和“TL0”读取即可，为了方便其他程序模块调用，我们将读取计数值的代码封装为名称为“get_timer0_count”的函数，该函数返回读取的计数值，代码清单如下。

代码清单：读取计数值

```
1.  /******************************************************************************
2.   * 描   述 : 读取当前计数值
3.   * 入   参 : 无
4.   * 返回值 : 读取的计数值
5.   ******************************************************************************/
6.  u16 get_timer0_count(void)
7.  {
8.      u8 tempH,tempL;
9.
10.     tempH = TH0;              //读取计数值
11.     tempL = TL0;
12.     return (tempH<<8)+tempL; //返回读取的计数值
13. }
```

5. 主函数

主函数中调用 Timer0 初始化和启动函数，完成对 Timer0 的初始化和启动。之后在主循环里面每隔 1 秒读取一次计数值并通过串口输出计数值，代码清单如下。

代码清单：主函数

```
1.  /*********************************************************************
2.  功能描述：主函数
3.  入口参数：无
4.  返回值：int 类型
5.   *********************************************************************/
```

```
int main(void)
{
    u16 conut_number;
    u8 str_array[10];

    uart1_init();                   //串口 1 初始化
    timer0_counter_init();          //timer0 初始化
    timer0_start();                 //启动 timer0
    EA = 1;                         //使能总中断

    while(1)
    {
      conut_number = get_timer0_count();          //读取计数值
      uart1_send_string("connter value: ");       //串口输出读取的计数值
      sprintf(str_array, "%d", conut_number);
      uart1_send_string(str_array);
      uart1_send_string(" \r\n");
      delay_ms(1000);
    }
}
```

3.2.3. 硬件连接

本实验需要用杜邦线将 P3.4 连接到按键 KEY1 电路上，如下图所示。

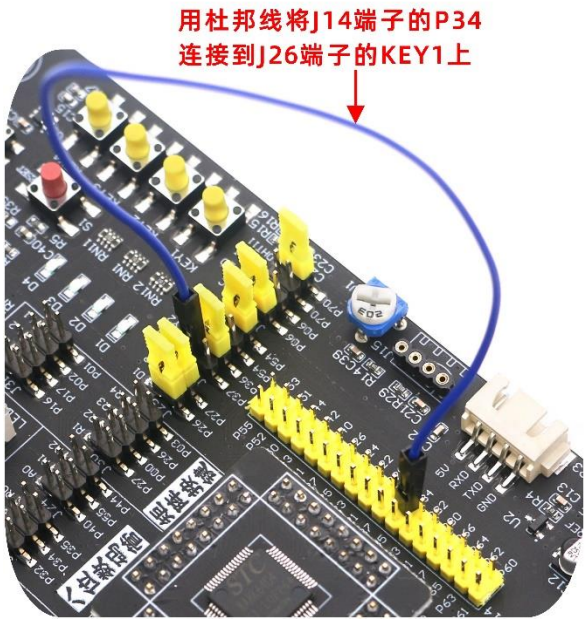

图 9：硬件连接

3.2.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-10-6：Timer0 对外部

输入脉冲计数”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。

2) 双击“…\timer0_counter\project”目录下的工程文件“counter.uvproj”。

3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“counter.hex”位于工程的“…\timer0_counter\Project\Object”目录下。

4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。

5) 程序运行后，按动 KEY1 按键并观察串口调试助手，可以在串口调试助手接收窗口观察到计数器的计数值随着按键按动增长。

✧ **说明：**Timer1、Timer2、Timer3 和 Timer4，他们工作于定时器模式时的操作和 Timer0 类似，读者可以尝试在学习了 Timer0 的基础上用他们来实现本例的功能。

我们也编写好了 Timer1、Timer2、Timer3 和 Timer4 工作于定时器模式的例子，这些例子在资料的“…\第 3 部分：配套例程源码目录下，他们的实验名称如下，读者在编写的过程中可以参考。

- 实验 2-10-7：Timer1 对外部输入脉冲计数。
- 实验 2-10-8：Timer2 对外部输入脉冲计数。
- 实验 2-10-9：Timer3 对外部输入脉冲计数。
- 实验 2-10-10：Timer4 对外部输入脉冲计数。

✧ 注意事项：

因为实验中用按键模拟脉冲，按键是存在抖动的，因此按动一次 KEY1 按键可能会产生多个脉冲，即计数器值会多次增加。