开源协会蓝桥杯第四讲 硬件定时器学习

硬件定时器工作原理（了解即可，可以直接跳过到实操）

- 硬件定时器是一种独立于CPU的计时设备，通常集成在微控制器或处理器芯片中。***它通过内部的时钟信号或外部的脉冲信号来计数，当计数达到预设的值时，会触发一个中断或事件，从而实现定时功能。***
- 基于单片机内部时钟信号而触发称为“定时器”，基于外部脉冲信号而触发称为“计数器”。
- 在介绍具体的例子之前我们先举个栗子：）
  你可以把定时器比喻成一个装了水的瓶子，每一次计数理解成向瓶子里面丢一个石子，当丢的石子足够多时，瓶子里面的水就会溢出，产生中断请求。

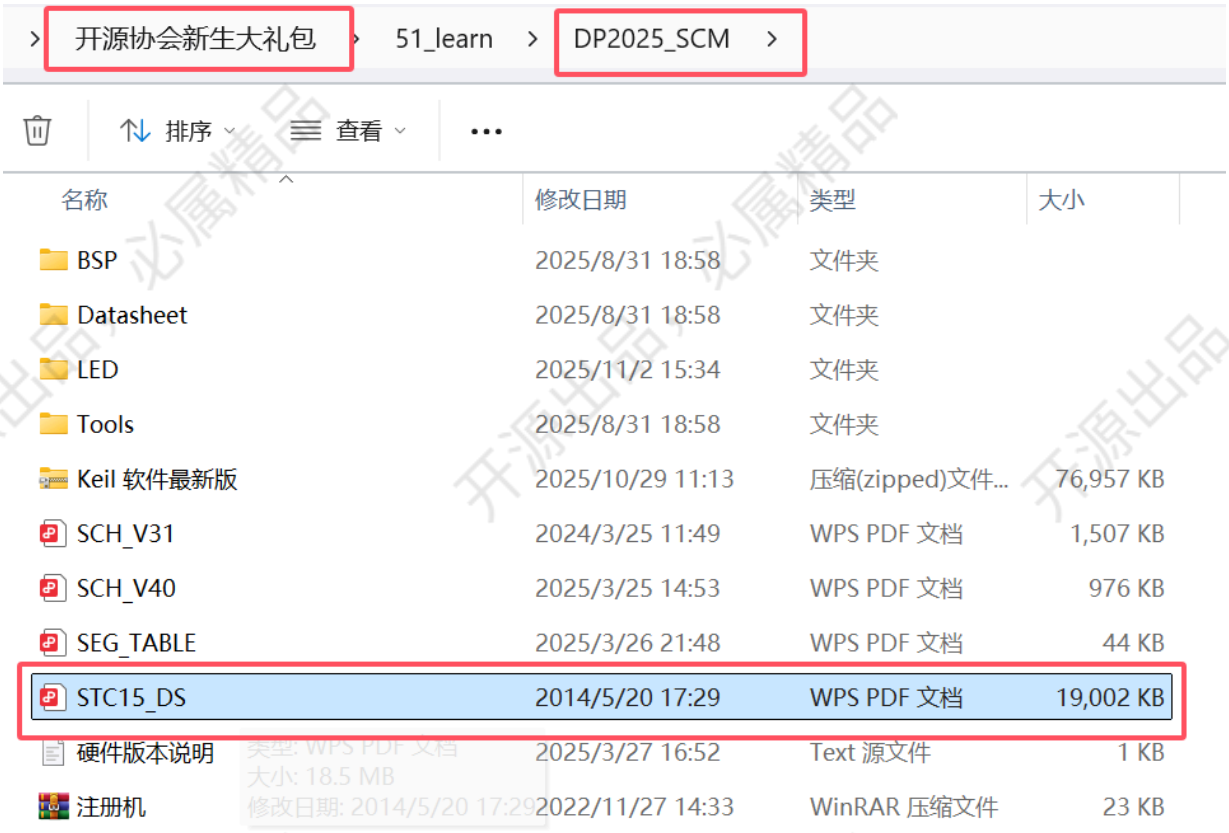
当作为定时器使用时，计数信号的来源是周期性的内部时钟频率，在单片机的内部，有一个频率为12MHZ的晶振，可以稳定的产生的产生一个周期为12MHZ的时钟信号，那么定时器就可以实现在每隔一段时间加一，实现定时。我们可以来简单的计算一下一个定时器的溢出周期：

- 单片机的时钟周期：T=1/f(sys) = 1/12MHZ = 0.083us 这是一个时钟周期。
- 单片机的指令周期：单片机的指令周期 = 12*时钟周期=12 * 0.083 = 1us;即定时器在工作的状态下每1us计数器就会加一（每1us向瓶子里面丢一个石子）。
- 那么最大可以加到多少呢？我们这里以定时器0为例：在单片机内部有两个寄存器TH0和TL0（每个寄存器8位，两个就是16位，每位有0何一两种状态），用来给定时器0计数，那么计数的最大值就是2^16-1 = 65535（最多丢65535个石子，瓶子里的水就会溢出），也就是说定时器可以对时钟信号从0开始计数一直到65535，然后溢出产生中断请求。那么最大的定时周期就是1us*65535 =65.535ms;当然：65.535ms的定时时间对于我们也许有一些鸡肋，但是我们可以认为的设置定时器初始的值（即可以设置这个瓶子里原来有多少石子），来控制定时的时间。
- eg: 假设我们希望定时器每隔1ms产生一个中断请求，即当我们丢入1000个石子（根据单片机的指令周期每1us丢入一个石子）时，瓶子里的水就会溢出，那么我们就可以设置定时器的初始值为65535-1000=64535,即TH0和TL0寄存器**经过一系列的转化后**的初始值为64535,那么当定时器开始工作时，每投1000个石子（隔1ms）就会产生一个中断请求。

转化示例：

```
TH0 = (65535 - 1000) / 256;  // 设置定时器 0 初值的高 8 位（0xFC=1111 1100）
TL0 = (65535 - 1000) % 256;  // 设置定时器 0 初值的低 8 位(0x18=0001 0111)
//65536-1000=64536=0xFC18
```

在蓝桥杯单片机的芯片手册中能找到定时器的寄存器介绍（了解即可可跳过）



1.定时器/计数器0/1控制寄存器TCON（可位寻址）手册第485页

1. 定时器/计数器0/1控制寄存器TCON

TCON为定时器/计数器T0、T1的控制寄存器，同时也锁存T0、T1溢出中断源和外部请求中断源等，TCON格式如下：

TCON：定时器/计数器中断控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件置“1”TF1，向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

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。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

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，TR1=1且INT0输入高电平时，才允许T0计数。

IE1：外部中断1请求源（INT1/P3.3）标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0：外部中断0请求源（INT0/P3.2）标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

IT0：外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

485

- TR0/TR1分别为定时器0/1的运行控制位，TR0/TR1为1时表示打开定时器0/1，为0时表示关闭

- TF0/TF1为定时器0/1溢出中断标志，当定时器0/1溢出时由硬件置1，在CPU处理中断请求后自动清0，也可以由软件清0。（知道就行，不用管）

2.定时器/计数器控制寄存器TMOD（不可位寻址）手册第486页

2. 定时器/计数器工作模式寄存器TMOD

定时和计数功能由特殊功能寄存器TMOD的控制位C/$\overline{T}$进行选择，TMOD寄存器的各位信息如下表所列。可以看出，2个定时/计数器有4种操作模式，通过TMOD的M1和M0选择。2个定时/计数器的模式0、1和2都相同，模式3不同，各模式下的功能如下所述。

寄存器TMOD各位的功能描述

TMOD　地址：89H　复位值：00H

不可位寻址

7	6	5	4	3	2	1	0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
定时器1				定时器0			

位	符号	功能
TMOD.7/	GATE	TMOD.7控制定时器1，置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。
TMOD.3/	GATE	TMOD.3控制定时器0，置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。
TMOD.6/	C/$\overline{T}$	TMOD.6控制定时器1用作定时器或计数器，清零则用作定时器(对内部系统时钟进行计数)，置1用作计数器(对引脚T1/P3.5外部脉冲进行计数)
TMOD.2/	C/$\overline{T}$	TMOD.2控制定时器0用作定时器或计数器，清零则用作定时器(对内部系统时钟进行计数)，置1用作计数器(对引脚T0/P3.4的外部脉冲进行计数)
TMOD.5/TMOD.4	M1、M0	定时器定时器/计数器1模式选择
	0　0	16位自动重装定时器，当溢出时将RL_TH1和RL_TL1存放的值自动重装入TH1和TL1中。
	0　1	16位不可重装载模式，TL1、TH1全用
	1　0	8位自动重装载定时器，当溢出时将TH1存放的值自动重装入TL1
	1　1	定时器/计数器1此时无效（停止计数）。
TMOD.1/TMOD.0	M1、M0	定时器/计数器0模式选择
	0　0	16位自动重装定时器，当溢出时将RL_TH0和RL_TL0存放的值自动重装入TH0和TL0中。
	0　1	16位不可重装载模式，TL0、TH0全用
	1　0	8位自动重装载定时器，当溢出时将TH0存放的值自动重装入TL0
	1　1	不可屏蔽中断的16位自动重装定时器

3.辅助寄存器AUXR（不可位寻址）

- 主要负责对定时器0/1的速度进行控制，以及定时器2的相关配置

3. 辅助寄存器AUXR

STC15系列**单片机是** 1T 的8051单片机，为兼容传统8051，定时器0、**定时器1, 和定时器2**复位后是传统8051的速度，即12分频，这是为了兼容传统8051。但也可不进行12分频，通过设置新增加的特殊功能寄存器AUXR, 将T0, T1, **T2设置为1T。**普通111条机器指令**执行速度是固定**的，快4到24倍，无法改变。

AUXR格式如下：

AUXR：辅助寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2

T0x12：定时器0速度控制位

0，定时器0是传统8051速度，12分频；

1，定时器0的速度是传统8051的12倍，不分频

T1x12：定时器1速度控制位

0，定时器1是传统8051速度，12分频；

1，定时器1的速度是传统8051的12倍，不分频

如果UART1/串口1用T1作为波特率发生器，则由T1x12决定UART1/串口1是12T还是1T

UART_M0x6：串口1模式0的通信速度设置位。

0，串口1模式0的速度是传统8051单片机串口的速度，12分频；

1，串口1模式0的速度是传统8051单片机串口速度的6倍，2分频

T2R：定时器2允许控制位

0，不允许定时器2运行；

1，允许定时器2运行

T2_C/$\overline{T}$：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/P3.1的外部脉冲进行计数)

T2x12：定时器2速度控制位

0，定时器2是传统8051速度，12分频；

1，定时器2的速度是传统8051的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

EXTRAM：内部/外部RAM存取控制位

0，允许使用逻辑上在片外、物理上在片内的扩展RAM；

1，禁止使用逻辑上在片外、物理上在片内的扩展RAM

S1ST2：串口1(UART1)选择定时器2作波特率发生器的控制位

0，选择定时器1作为串口1(UART1)的波特率发生器；

1，选择定时器2作为串口1(UART1)的波特率发生器，此时定时器1得到释放，可以作为独立定时器使用

4.定时器T0和T1的中断控制寄存器IE和IP（可位寻址）

5、定时器T0和T1的中断控制寄存器：IE和IP

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位，EA=1，CPU开放中断，EA=0，CPU屏蔽所有的中断申请。EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制；其次还受各中断源自己的中断允许控制位控制。

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

ET0：T0的溢出中断允许位，ET0=1允许T0中断，ET0=0禁止T0中断。

IP：中断优先级控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PT1:　定时器1中断优先级控制位。
当PT1=0时，定时器1中断为最低优先级中断(优先级0)
当PT1=1时，定时器1中断为最高优先级中断(优先级1)

PT0:　定时器0中断优先级控制位。
当PT0=0时，定时器0中断为最低优先级中断(优先级0)
当PT0=1时，定时器0中断为最高优先级中断(优先级1)

- ET0/ET1：定时器T0/T1的中断允许位，为1时表示允许中断，反之则不允许
- PT0/PT1：设置定时器T0/T1的中断优先级，为1设置为高优先级，为0设置低优先级

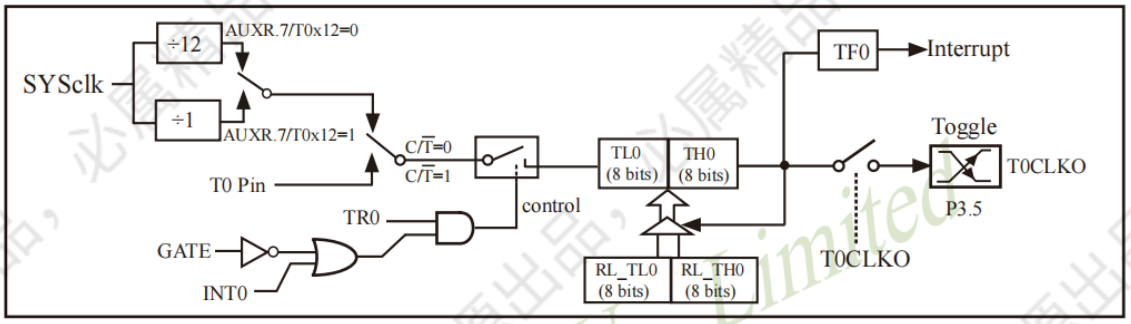
5.定时器0/1/2高八位/低八位寄存器TLH0/TL0/TH1/TL1/T2H/T2L（484页）

7.1 定时器/计数器的相关寄存器

符号	描述	地址	位地址及其符号 MSB LSB								复位值
TCON	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	Timer Mode	89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	0000 0000B
TL0	Timer Low 0	8AH									0000 0000B
TL1	Timer Low 1	8BH									0000 0000B
TH0	Timer High 0	8CH									0000 0000B
TH1	Timer High 1	8DH									0000 0000B
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	中断优先级寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
T2H	定时器2高8位寄存器	D6H									0000 0000B
T2L	定时器2低8位寄存器	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	外部中断允许和时钟输出寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000 x000B
T4T3M	T4和T3的控制寄存器	D1H	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO	0000 0000B
T4H	定时器4高8位寄存器	D2H									0000 0000B
T4L	定时器4低8位寄存器	D3H									0000 0000B
T3H	定时器3高8位寄存器	D4H									0000 0000B
T3L	定时器3低8位寄存器	D5H									0000 0000B
IE2	Interrupt Enable register	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

6.自动重装载模式

此模式下定时器/计数器0作为可自动重装载的16位计数器，如下图所示。



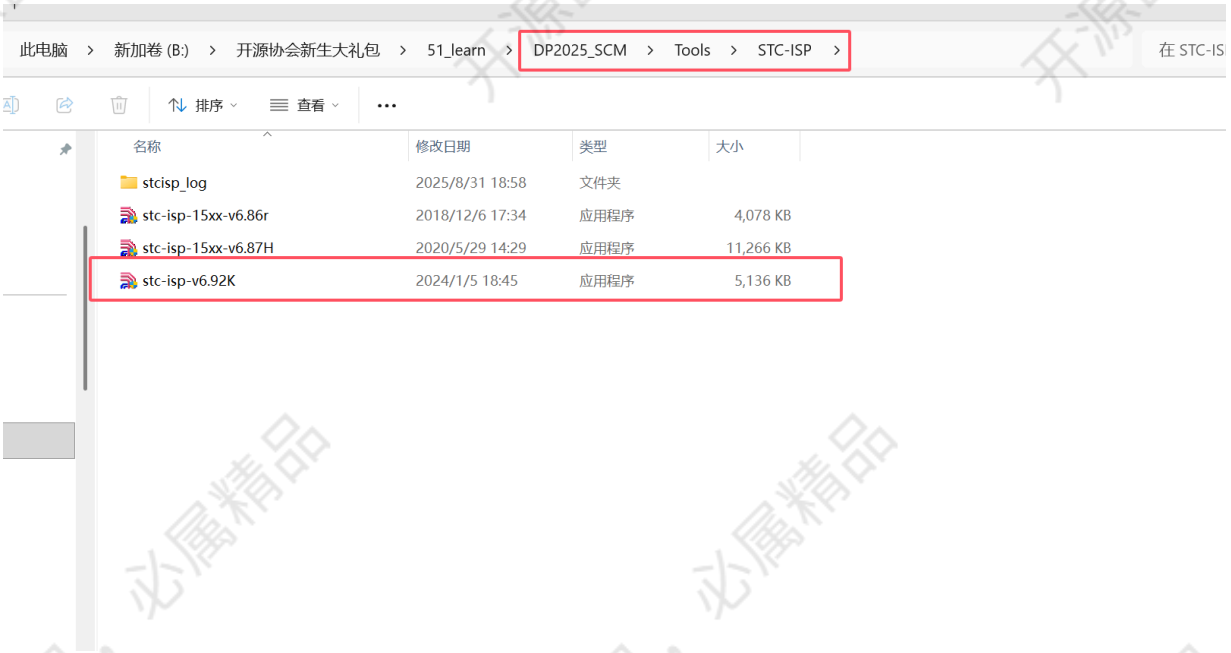
定时器/计数器0的模式 0: 16位自动重装

定时器中断原理

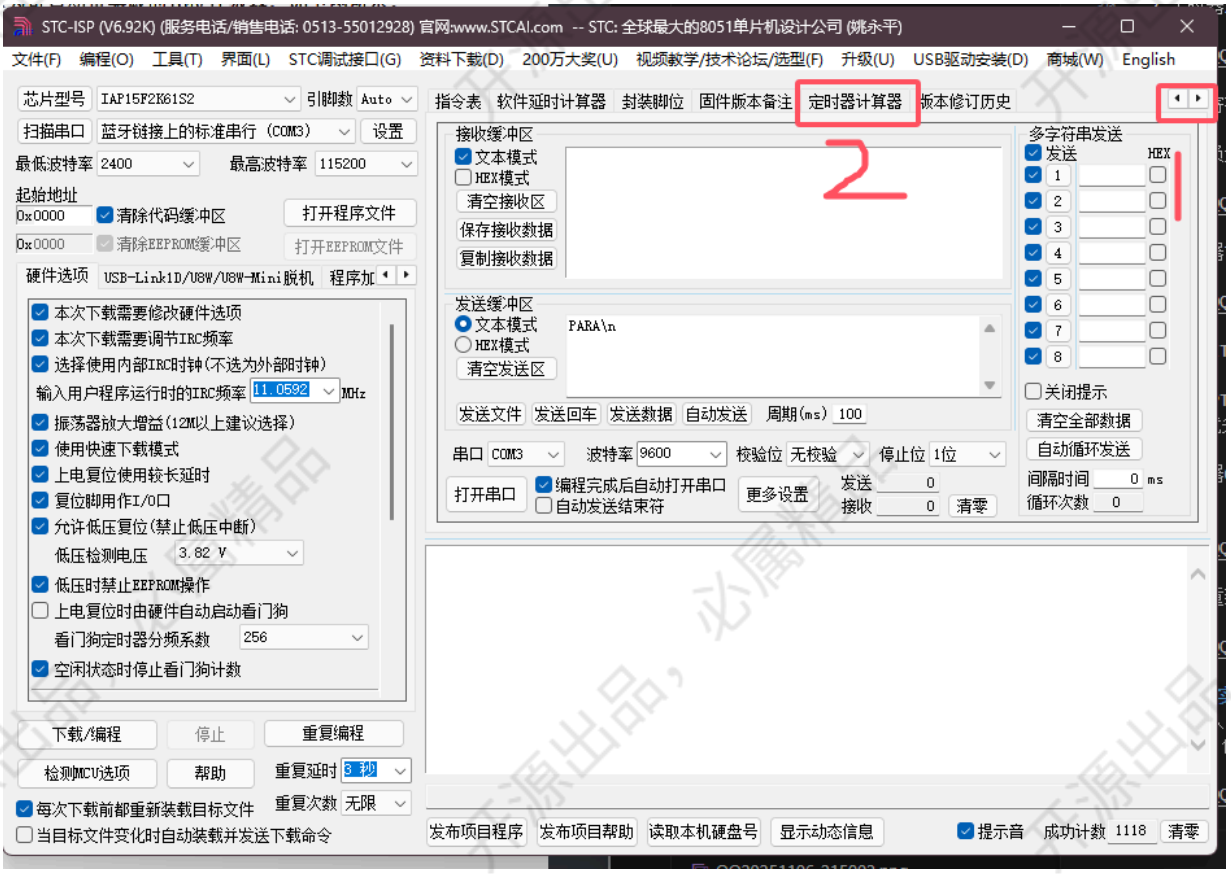
- 当定时器溢出时，会自动触发一个中断请求，CPU在完成当前指令后会跳转到对应的中断服务程序（ISR）执行预定义的任务。
  作用：定时器中断可以用于实现周期性任务，如定时采样、时间延迟、事件计数等。
  好处：使用定时器中断可以让CPU在等待定时事件时执行其他任务，提高系统的效率和响应能力。

代码实操

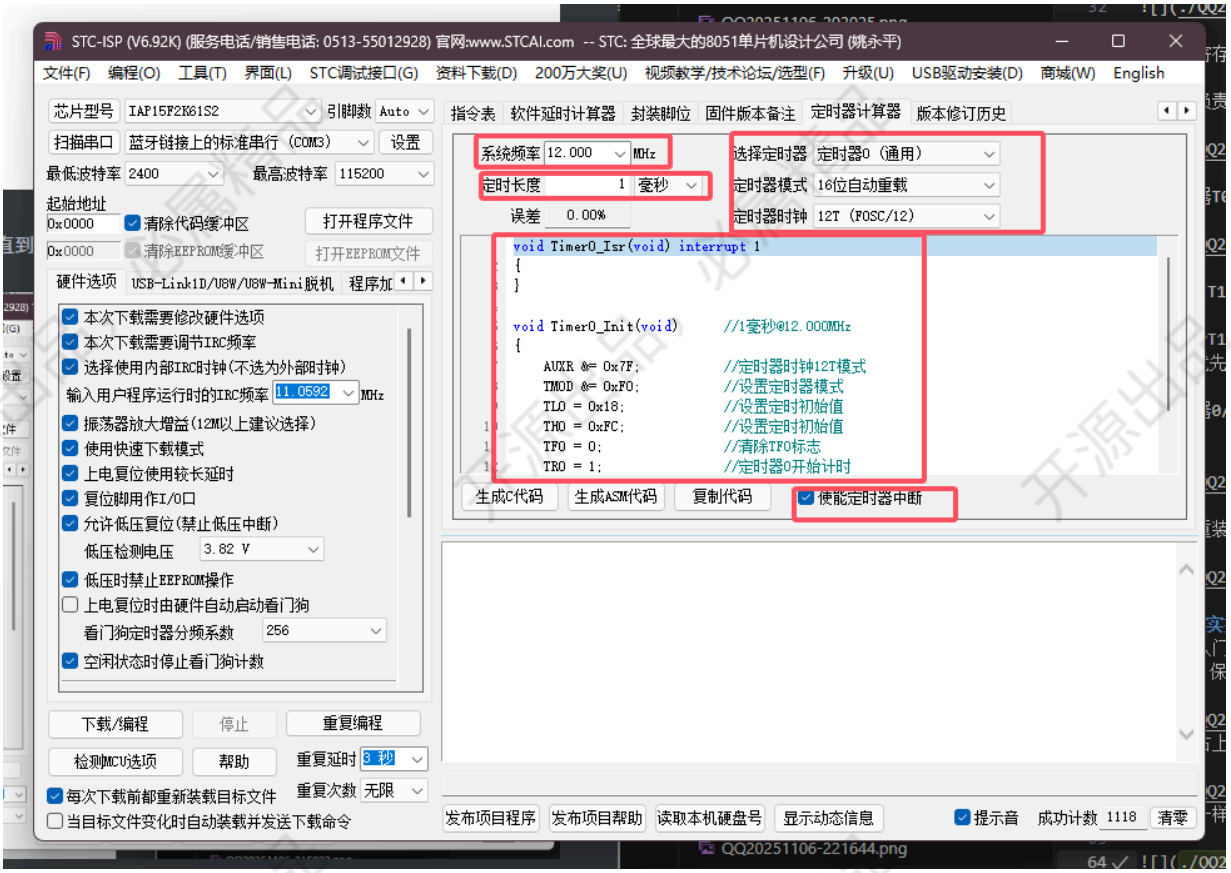
- 打开入门手册中做好的工程，打开STC-ISP软件(注意！一定要是V6.92的版本，保证定时器配置一样)



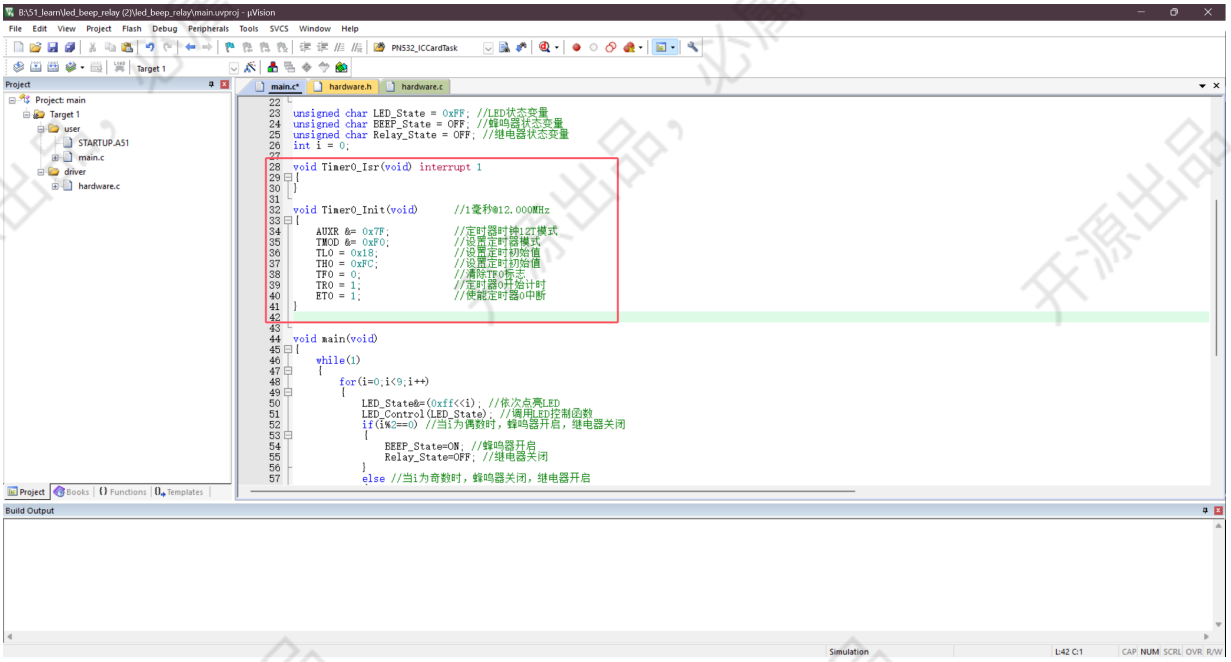
- 点击右上角的右箭头，直到看见定时器/计数器设置



- 像我一样设置定时器



点击生成C代码，复制代码后粘贴到keil中，如图所示




```c
void Timer0_Isr(void) interrupt 1
{
}

void Timer0_Init(void)      //1毫秒@12.000MHz
{
    AUXR &= 0x7F;           //定时器时钟12T模式
    TMOD &= 0xF0;           //设置定时器模式
    TL0 = 0x18;             //设置定时初始值
    TH0 = 0xFC;             //设置定时初始值
    TF0 = 0;                //清除TF0标志
    TR0 = 1;                //定时器0开始计时
    ET0 = 1;                //使能定时器0中断
```

```
}
```

- 代码改动
  添加EA=1;使能总中断

```
void Timer0_Isr(void) interrupt 1
{
}
void Timer0_Init(void)      //1毫秒@12.000MHz
{
    AUXR &= 0x7F;           //定时器时钟12T模式
    TMOD &= 0xF0;           //设置定时器模式
    TL0 = 0x18;             //设置定时初始值
    TH0 = 0xFC;             //设置定时初始值
    TF0 = 0;                //清除TF0标志
    TR0 = 1;                //定时器0开始计时
    ET0 = 1;                //使能定时器0中断
    EA = 1;                 //使能总中断
}
```

- 我们一般将中断函数放在main函数的后面，在中断函数中添加代码，每隔1s让继电器和蜂鸣器的状态取反

最后的总代码：(只改变了main.c的代码其他的文件保持不变)

- 复制粘贴到main.c中覆盖原有代码重新编译下载即可实现

```
#include <STC15F2K60S2.H>
#include "hardware.h"
#include "intrins.h"

void Delay500ms(void)   //@12.000MHz
{
    unsigned char data i, j, k;

    _nop_();
    _nop_();
    i = 23;
    j = 205;
    k = 120;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}

unsigned char LED_State = 0xFF; //LED状态变量
unsigned char BEEP_State = OFF; //蜂鸣器状态变量
unsigned char Relay_State = OFF; //继电器状态变量
int i = 0;

int time1s=0; //1秒计时变量不能用unsigned char否则会溢出

```

```c
void Timer0_Init(void)		//1毫秒@12.000MHz
{
	AUXR &= 0x7F;			//定时器时钟12T模式
	TMOD &= 0xF0;			//设置定时器模式
	TL0 = 0x18;				//设置定时初始值
	TH0 = 0xFC;				//设置定时初始值
	TF0 = 0;				//清除TF0标志
	TR0 = 1;				//定时器0开始计时
	EA=1;					//打开总中断
	ET0 = 1;				//使能定时器0中断
}



void main(void)
{
	Timer0_Init();
	while(1)
	{
		for(i=0;i<9;i++)
		{
			LED_State&=(0xff<<i); //依次点亮LED
			LED_Control(LED_State); //调用LED控制函数
//			if(i%2==0) //当i为偶数时，蜂鸣器开启，继电器关闭
//			{
//				BEEP_State=ON; //蜂鸣器开启
//				Relay_State=OFF; //继电器关闭
//			}
//			else //当i为奇数时，蜂鸣器关闭，继电器开启
//			{
//				BEEP_State=OFF; //蜂鸣器关闭
//				Relay_State=ON; //继电器开启
//			}
//			BEEP_Relay_Control(BEEP_State, Relay_State); //蜂鸣器开启，继电器关闭
			Delay500ms(); //延时500毫秒
		}
		for(i=0;i<9;i++)
		{
			LED_State|=~(0xff>>i); //依次熄灭LED
			LED_Control(LED_State); //调用LED控制函数
//			if(i%2==0) //当i为偶数时，蜂鸣器开启，继电器关闭
//			{
//				BEEP_State=OFF; //蜂鸣器开启
//				Relay_State=ON; //继电器关闭
//			}
//			else //当i为奇数时，蜂鸣器关闭，继电器开启
//			{
//				BEEP_State=ON; //蜂鸣器关闭
//				Relay_State=OFF; //继电器开启
//			}
//			BEEP_Relay_Control(BEEP_State, Relay_State); //蜂鸣器开启，继电器关闭
			Delay500ms(); //延时500毫秒
		}
	}
}
```

```
bit beep_relay_flag=0;
void Timer0_Isr(void) interrupt 1
{
    if(time1s++>=1000)
    {
        if(beep_relay_flag == 0)
        {
            BEEP_State=ON; //蜂鸣器开启
            Relay_State=OFF; //继电器关闭
            BEEP_Relay_Control(BEEP_State, Relay_State); //蜂鸣器开启，继电器关闭
            beep_relay_flag = 1;
        }
        else if(beep_relay_flag == 1)
        {
            BEEP_State=OFF; //蜂鸣器关闭
            Relay_State=ON; //继电器开启
            BEEP_Relay_Control(BEEP_State, Relay_State); //蜂鸣器关闭，继电器开启
            beep_relay_flag = 0;
        }
        time1s=0;
    }

}
```