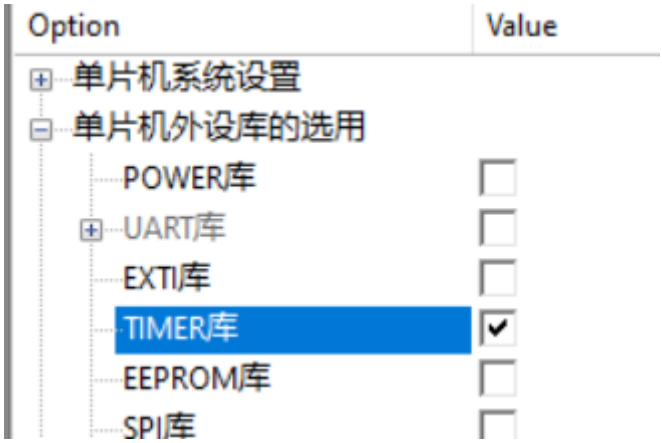
TIMER库

TIMER就是常说的定时器，TIMER库就是关于单片机定时器的操作库。在使用本库之前先到ecbm_core.h里使能。双击打开ecbm_core.h文件，然后进入图形化配置界面，使能TIMER库。



定时器的本质就是计数器，因此在接下来的文章中，你会看到定时/计数两个方向的应用。他们的区别在于：计数器是对外部脉冲进行计数，由于外部脉冲的周期和个数都不确定，所以只能统计脉冲的个数；定时器时对内部系统时钟进行计数，由于内部系统时钟的脉冲周期是确定的，根据公式“脉冲周期值x脉冲个数=经过的时间”就能算出时间。也就是说只要外部脉冲是连续不断的、周期恒定的，也可以用外部脉冲来做定时应用。

API

timer_start

函数原型：void timer_start(u8 id);

描述

定时器开启函数。

输入

- id：要开启的定时器编号，从0开始。

输出

无

返回值

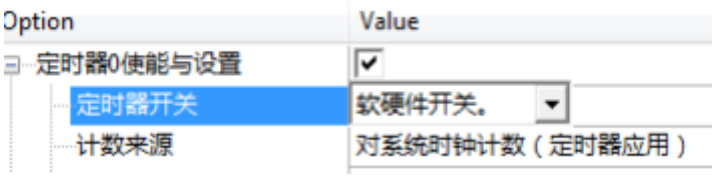
无

调用例程

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    timer_init();//先初始化定时器。
    timer_start(0);//再打开定时器0。
    while(1){
    }
}
```

注意事项

1. 使用本函数之前，要先初始化定时器。
2. 在【定时器开关】选项中选择了"软硬件开关"之后，除了执行本函数之外还需要对应的引脚输入高电平才可开启定时器。



timer_stop

函数原型：void timer_stop(u8 id);

描述

定时器关闭函数。

输入

- id：要关闭的定时器编号，从0开始。

输出

无

返回值

无

调用例程

```
if(key_stop==0){//当停止按键被按下的时候，
    timer_stop(0);//关闭定时器0。
}
```

注意事项

无

timer_out

函数原型：void timer_out(u8 id,u8 en);

描述

定时器输出控制函数。

输入

- id：要设置输出的定时器编号，从0开始。
- en：1代表开启时钟输出，0代表关闭时钟输出。

输出

无

返回值

无

输出引脚

- **定时器0**：P35脚。
- **定时器1**：P34脚。
- **定时器2**：P13脚。
- **定时器3**：P05脚。
- **定时器4**：P07脚。

调用例程

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    timer_init();//先初始化定时器。
    timer_out(0,1);//然后打开定时器0的时钟输出。
    timer_start(0);//最后打开定时器0。
    while(1){
    }
}
```

注意事项

1. 时钟输出的频率和定时器的溢出率有关，原理上来说定时器每溢出一次，对应的输出脚取反一次。这个过程是硬件全自动的，比软件取反要快很多，特别是几MHz的时钟输出。

timer_init

函数原型：void timer_init(void);

描述

定时器初始化函数。

输入

无

输出

无

返回值

无

参数配置

在图形化配置界面设置的参数，将会在执行本函数的时候写到定时器寄存器中。因此要保证选项选择正确无误。

设置说明如下：

定时器0

- **定时器0使能与设置**：勾选之后，定时器0的代码才会参与编译。
- **定时器开关**：【软件开关】是指用代码开启关闭定时器，【软硬件开关】是指在代码开启之后，还需要P32为高电平才能开启定时器。
- **计数来源**：用作定时器的时候选择【对系统时钟计数（定时器应用）】，用作计数器的时候选择【对外部T0（P34）脚的脉冲信号计数（计数器应用）】。
- **工作模式**：以前性能低没得选，现在只推荐选【16位自动重载模式】。
- **时钟分频**：有【系统时钟不分频】和【系统时钟12分频（Fosc/12）】可以选择，根据实际情况来选。但是注意这个分频只能影响到定时器，不会影响定时器以外的外设的工作频率（串口用到了定时器，就还会受影响）。
- **对外输出时钟**：使能之后，P35脚就会输出时钟。
- **定时时间/计数数量**：在定时模式下，这个参数决定了定时时间；在计数模式下，这个参数决定了计数的个数。
- **定时器0的中断使能**：使能之后，定时器0溢出就触发中断。

定时器1

- **定时器1使能与设置**：勾选之后，定时器1的代码才会参与编译。
- **定时器开关**：【软件开关】是指用代码开启关闭定时器，【软硬件开关】是指在代码开启之后，还需要P33为高电平才能开启定时器。
- **计数来源**：用作定时器的时候选择【对系统时钟计数（定时器应用）】，用作计数器的时候选择【对外部T1（P35）脚的脉冲信号计数（计数器应用）】。
- **工作模式**：以前性能低没得选，现在只推荐选【16位自动重载模式】。
- **时钟分频**：有【系统时钟不分频】和【系统时钟12分频（Fosc/12）】可以选择，根据实际情况来选。但是注意这个分频只能影响到定时器，不会影响定时器以外的外设的工作频率（串口用到了定时器，就还会受影响）。
- **对外输出时钟**：使能之后，P34脚就会输出时钟。
- **定时时间/计数数量**：在定时模式下，这个参数决定了定时时间；在计数模式下，这个参数决定了计数的个数。
- **定时器1的中断使能**：使能之后，定时器1溢出就触发中断。

定时器2

- **定时器2使能与设置**：勾选之后，定时器2的代码才会参与编译。
- **计数来源**：用作定时器的时候选择【对系统时钟计数（定时器应用）】，用作计数器的时候选择【对外部T2（P12）脚的脉冲信号计数（计数器应用）】。
- **工作模式**：以前性能低没得选，现在只推荐选【16位自动重载模式】。
- **时钟分频**：有【系统时钟不分频】和【系统时钟12分频（Fosc/12）】可以选择，根据实际情况来选。但是注意这个分频只能影响到定时器，不会影响定时器以外的外设的工作频率（串口用到了定时器，就还会受影响）。
- **对外输出时钟**：使能之后，P13脚就会输出时钟。
- **定时时间/计数数量**：在定时模式下，这个参数决定了定时时间；在计数模式下，这个参数决定了计数的个数。
- **定时器2的中断使能**：使能之后，定时器2溢出就触发中断。

定时器3

- **定时器3使能与设置**：勾选之后，定时器3的代码才会参与编译。
- **计数来源**：用作定时器的时候选择【对系统时钟计数（定时器应用）】，用作计数器的时候选择【对外部T3（P04）脚的脉冲信号计数（计数器应用）】。
- **工作模式**：以前性能低没得选，现在只推荐选【16位自动重载模式】。
- **时钟分频**：有【系统时钟不分频】和【系统时钟12分频（Fosc/12）】可以选择，根据实际情况来选。但是注意这个分频只能影响到定时器，不会影响定时器以外的外设的工作频率（串口用到了定时器，就还会受影响）。
- **对外输出时钟**：使能之后，P05脚就会输出时钟。
- **定时时间/计数数量**：在定时模式下，这个参数决定了定时时间；在计数模式下，这个参数决定了计数的个数。
- **定时器3的中断使能**：使能之后，定时器3溢出就触发中断。

定时器4

- **定时器4使能与设置**：勾选之后，定时器4的代码才会参与编译。
- **计数来源**：用作定时器的时候选择【对系统时钟计数（定时器应用）】，用作计数器的时候选择【对外部T4（P06）脚的脉冲信号计数（计数器应用）】。
- **工作模式**：以前性能低没得选，现在只推荐选【16位自动重载模式】。
- **时钟分频**：有【系统时钟不分频】和【系统时钟12分频（Fosc/12）】可以选择，根据实际情况来选。但是注意这个分频只能影响到定时器，不会影响定时器以外的外设的工作频率（串口用到了定时器，就还会受影响）。
- **对外输出时钟**：使能之后，P07脚就会输出时钟。
- **定时时间/计数数量**：在定时模式下，这个参数决定了定时时间；在计数模式下，这个参数决定了计数的个数。
- **定时器4的中断使能**：使能之后，定时器4溢出就触发中断。

调用例程

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    timer_init();//初始化定时器。
    while(1){
    }
}
```

注意事项

1. 本函数将会初始化所有使能的定时器。

timer_set_timer_mode

函数原型：void timer_set_timer_mode(u8 id,u16 us);

描述

定时器设置定时模式函数。

输入

- id：要设置为定时模式的定时器编号，从0开始。
- us：定时的时间，范围参考下面的说明。

输出

无

返回值

无

定时时间预估

单片机主频（MHz）	最小时间（uS）	最大时间（uS）
5.5296	2	65535
6.000	1	65535
11.0592	1	65535
12.000	1	65535
18.432	2	42666
20.000	1	39321
22.1184	1	35555
24.000	1	32768
27.000	1	29127
30.000	1	26214
33.000	1	23831
33.1776	1	23703
35.000	1	22469
36.864	1	21333
40.000	1	19660
44.2368	1	17777
45.000	1	17476

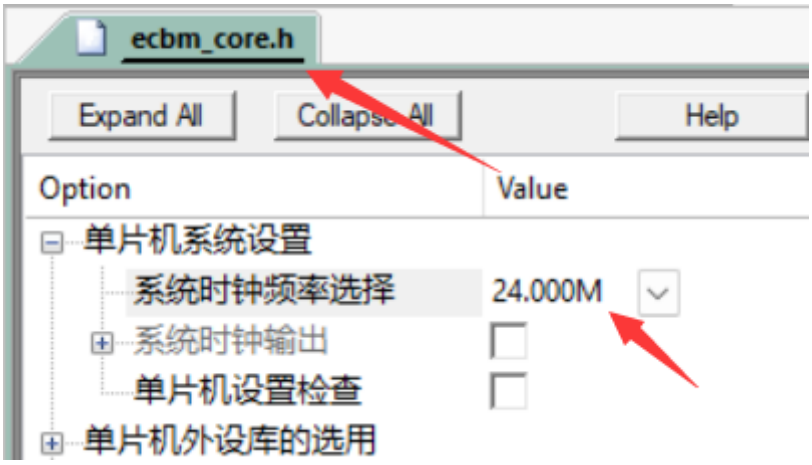
调用例程

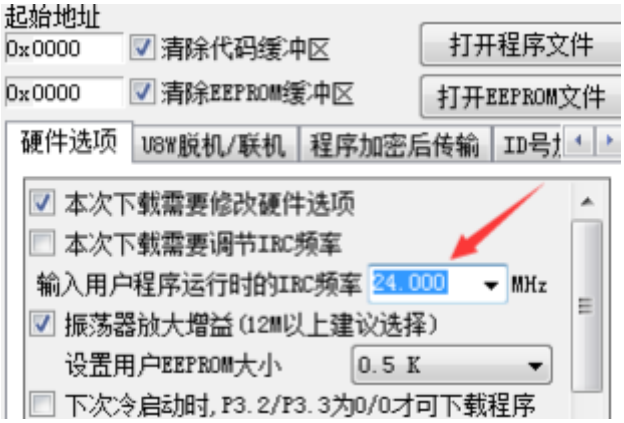
```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    timer_init();//初始化定时器。
    timer_set_timer_mode(1,10000);//设置定时器1的定时时间为10000uS也就是10mS。
    timer_start(1);//打开定时器1。
    while(1){
    }
}
```

注意事项

1. 运行本函数之后，会将指定的定时器的模式换成定时模式，同时根据参数计算出分频和初值。届时在图形化配置界面设置的这3个参数将会被本函数的计算值覆盖掉。

2. 基于上一点，当你发现定时器的运行效果和你在图形化配置界面设置的效果相差甚远的时候，可以找找是否在某处调用了本函数。
3. 本函数的计算依赖于系统主频的设置，部分时间会有少许计算误差，但如果定时严重不准的话，请注意这两处的设置是否一致：





timer_set_value

函数原型：void timer_set_value(u8 id,u16 value);

描述

定时器设定计数值函数。

输入

- id：要设定计数值的定时器编号，从0开始。
- value：设置的计数值，范围是0~65535。

输出

无

返回值

无

调用例程

```
timer_init();//初始化定时器。
timer_set_value(0,65530);//设置定时器0的计数值。
timer_start(0);//打开定时器0。
```

注意事项

1. 51单片机的定时器是从初值向上计数，计满65536溢出重载初值，所以你想计数20个的时候，需要给65536-20=65516的初值。

timer_get_value

函数原型：u16 timer_get_value(u8 id);

描述

定时器计数值获取函数。

输入

- id：要获取计数值的定时器编号，从0开始。

输出

无

返回值

- 指定编号的定时器计数值。

调用例程

```
u16 val;//定义的变量。
timer_stop(0);//先停止定时器0。
val=timer_get_value(0);//读取定时器0的计数值。
timer_start(0);//重新打开定时器0。
```

注意事项

1. 最好在读取之前把定时器关闭，以防在读取的过程中，计数值又发生了改变。

回调函数

在串口库那一章里有提到过回调函数的执行原理。目前为了统一管理中断，ECBM库已经定义好了中断处理函数，并开放了回调函数供大家使用。只要按着回调函数的名字定义一个一模一样的函数就行。

- timer0_it_callback
- timer1_it_callback

- timer2_it_callback
- timer3_it_callback
- timer4_it_callback

调用例程

```
void timer0_it_callback(void){//这是定时器0的中断处理函数。
    LED=!LED; //当定时器0中断时，取反LED的亮灭状态。
}
```

注意事项

1. 回调函数的名字一定得正确，因为中断里面会调用这些回调函数，如果名字不正确就调用不了。
2. 本函数不需要也不能在其他地方调用！只需要定义了即可。因为在定时器中断触发的时候，单片机的硬件会自动去调用的。

优化建议

用不到的定时器就不去开使能，如果定时时间固定，可以删掉timer_set_timer_mode来节约空间。