- 注:RT的硬件定时器与HAL库的用法不太一样
  上一节讲了RT的软件定时器,这一节主要介绍硬件定时器

# HWTIMER设备

## 常用函数

### 函数归纳

| 函数名 | 说明 |
| --- | --- |
| rt_device_find() | 查找定时器设备 |
| rt_device_open() | 以读写方式打开定时器设备 |
| rt_device_set_rx_indicate() | 设置超时回调函数 |
| rt_device_control() | 控制定时器设备,可以设置定时模式(单次/周期)/计数频率,或者停止定时器 |
| rt_device_write() | 设置定时器超时值,定时器跟着启动 |
| rt_device_read() | 获取定时器的当前值 |
| rt_device_close() | 关闭定时器设备 |

### 操作步骤

1. 配置CubeMx，选择硬件定时器
   ↓
2. 修改并添加 board 中的宏定义
   ↓
3. 创建定时器设备
   ↓
4. 查找定时器设备
   ↓
5. 以读写方式打开定时器设备
   ↓
6. 书写超时回调函数
   ↓
7. 控制定时器设备（将配置参数写入）
   ↓
8. 设置定时器超时值
   ↓
9. 获取定时器当前值（可选）
   ↓
10. 关闭定时器（可选）

## 函数详解

- 查找定时器设备
  rt_device_find(const char* name)
  name 为创建定时器的名称
  如果找到定时器,则返回定时器的句柄
  如果返回RT_NULL则没有找到该设备
- 打开定时器设备
  rt_err_t rt_device_open(rt_device_t dev, rt_uint16_t oflags);
  dev 为找到的定时器的句柄
  oflas 为设备打开的模式,一般为读写模式:RT_DEVICE_OFLAG_RDWR
  若返回RT_EOK则设备打开成功
- 设置超时回调函数
  rt_err_t rt_device_set_rx_indicate(rt_device_t dev, rt_err_t (*rx_ind)(rt_device_t dev,rt_size_t size))
  dev 设备句柄
  rx_ind 超时回调函数名
  返回RT_EOK则为成功
- 控制定时器设备
  rt_err_t rt_device_control(rt_device_t dev,rt_uint8_t cmd, void* arg);
  dev 设备句柄
  cmd 命令控制字
  arg 控制的参数
  返回RT_EOK则为成功
- 注常用控制字:

| 控制字 | 描述 |
| --- | --- |
| HWTIMER_CTRL_FREQ_SET | 设置计数频率（若未设置该项，默认为1Mhz 或支持的最小计数频率） |
| HWTIMER_CTRL_STOP | 停止定时器 |
| HWTIMER_CTRL_INFO_GET | 获取定时器特征信息 |
| HWTIMER_CTRL_MODE_SET | 设置定时器模式（若未设置，默认是HWTIMER_MODE_ONESHOT） |

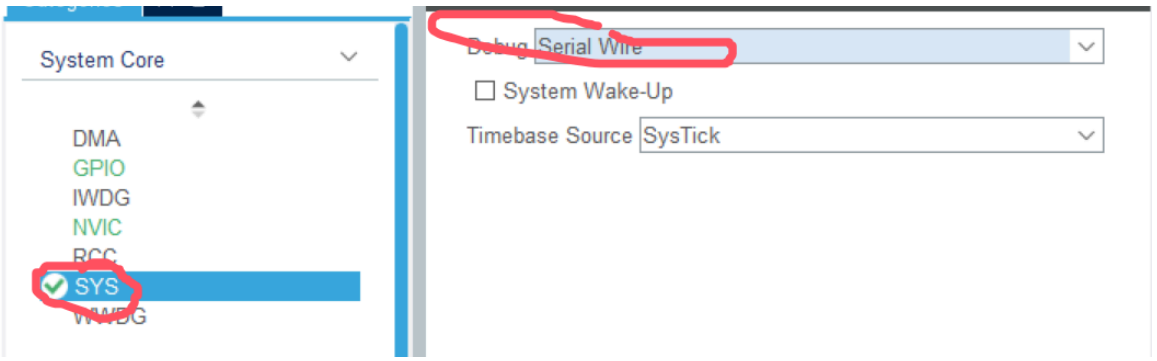
*设置定时器超时值*
*rt_size_t rt_device_write(rt_device_t dev, rt_off_t pos, const void* buffer, rt_size_t size);
dev 设备句柄
pos 指向定时器超时时间结构体的指针
buffer 指向定时器超时时间结构体的指针
size 超时时间结构体大小
返回写入数据的实际大小,否则失败.
超时值的单位为秒和微秒

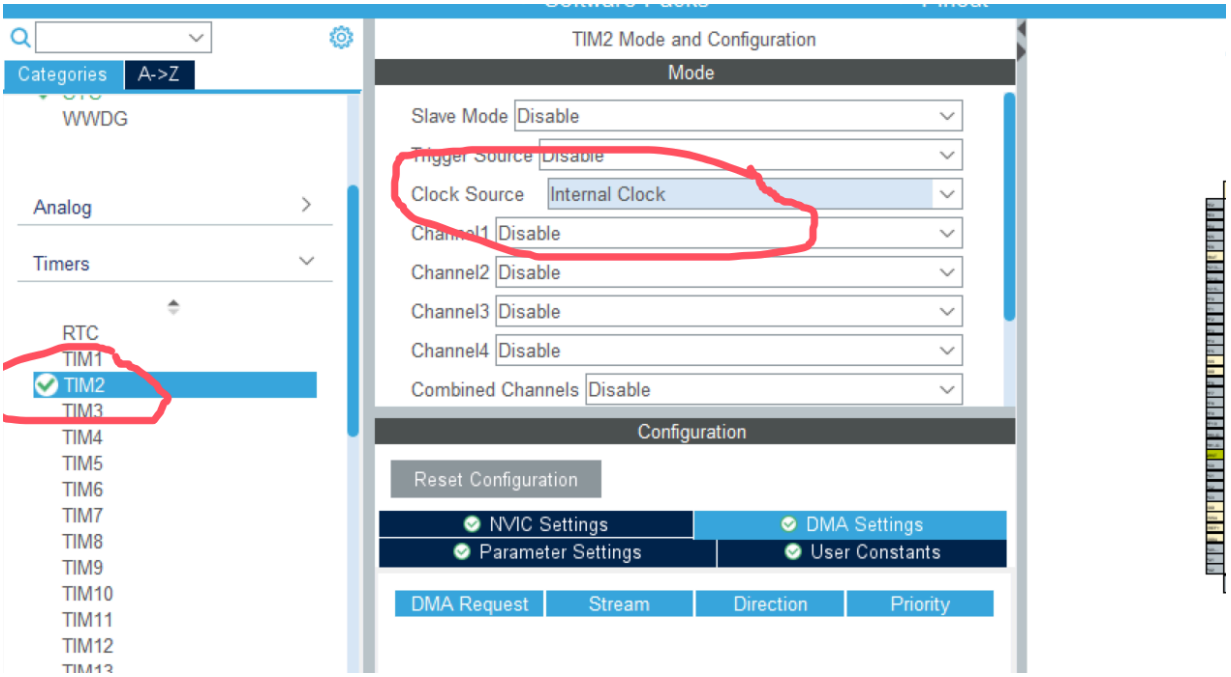
*获取定时器当前值*
*rt_size_t rt_device_read(rt_device_t dev, rt_off_t pos, void* buffer, rt_size_t size);
dev 设备句柄
pos 指向定时器超时时间结构体的指针
buffer 指向定时器超时时间结构体的指针
size 超时时间结构体大小
返回超时时间结构体的大小,否则失败

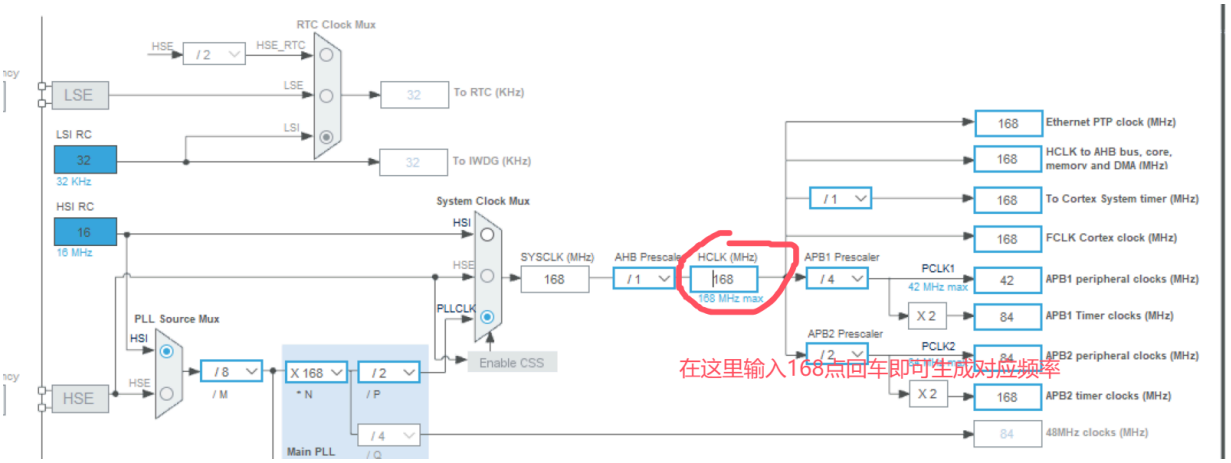
*关闭定时器设备
rt_err_t rt_device_close(rt_device_t dev);
dev 设备句柄
返回RT_EOK则关闭设备成功

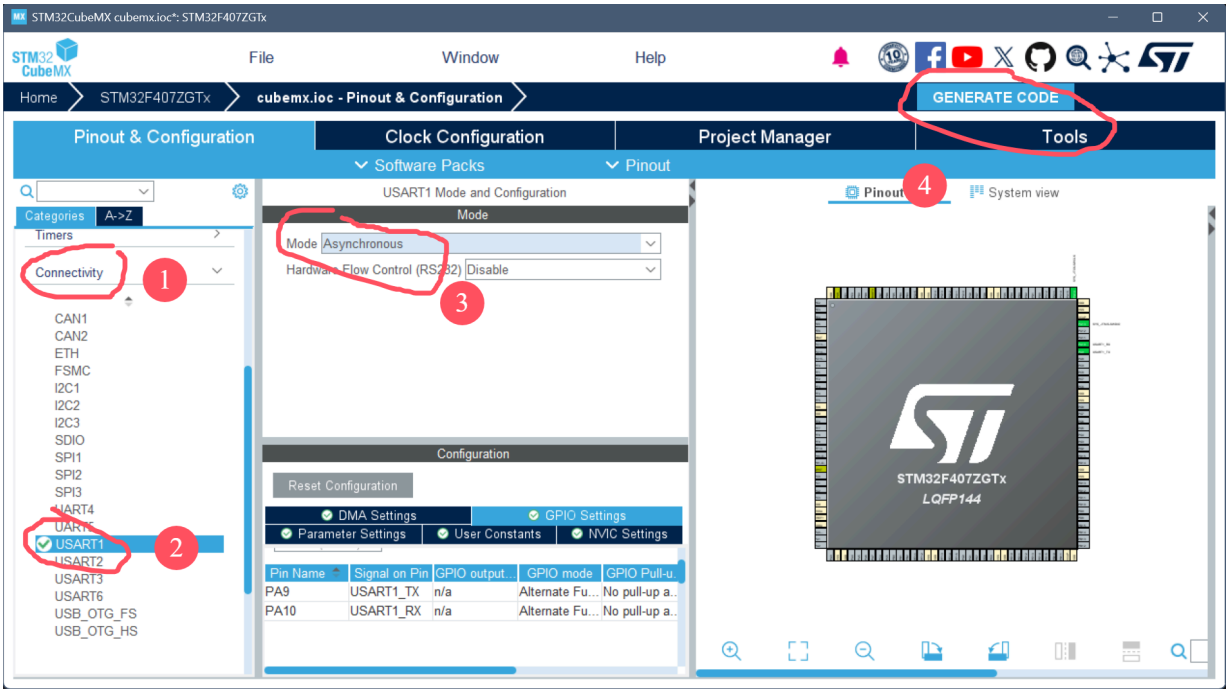
# 完整流程

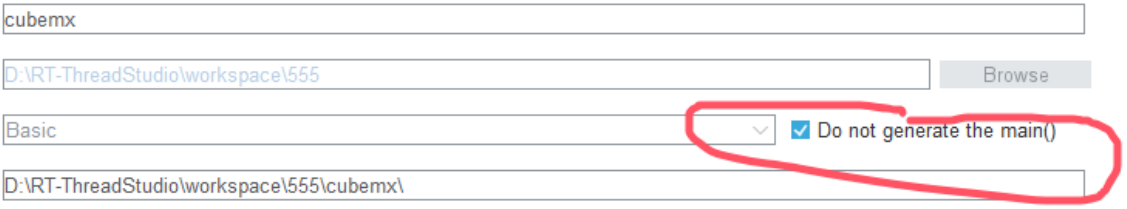
- 1.设置CubeMx



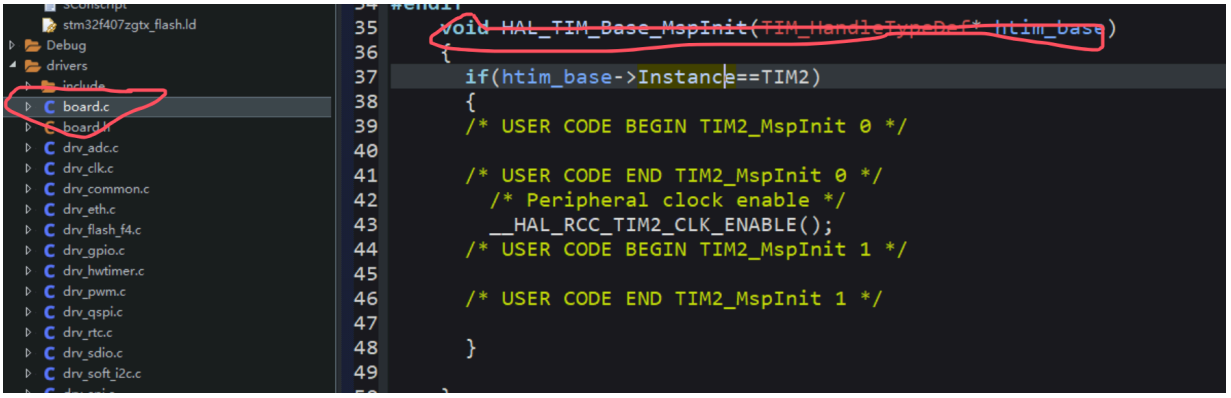



* 生成完记得关闭CubeMx否则会有报错

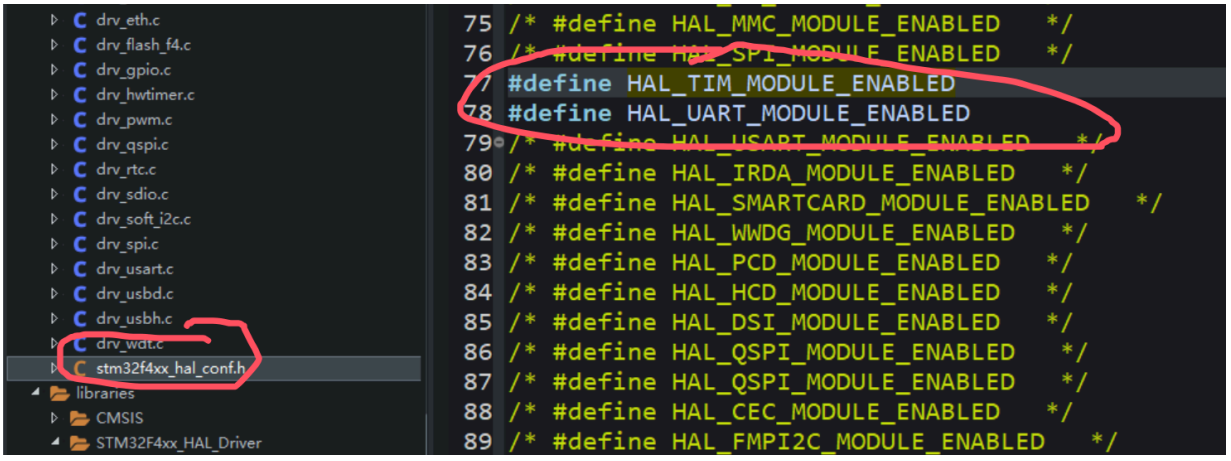

- 2.配置硬件定时器环境


做完记得Ctrl+s保存

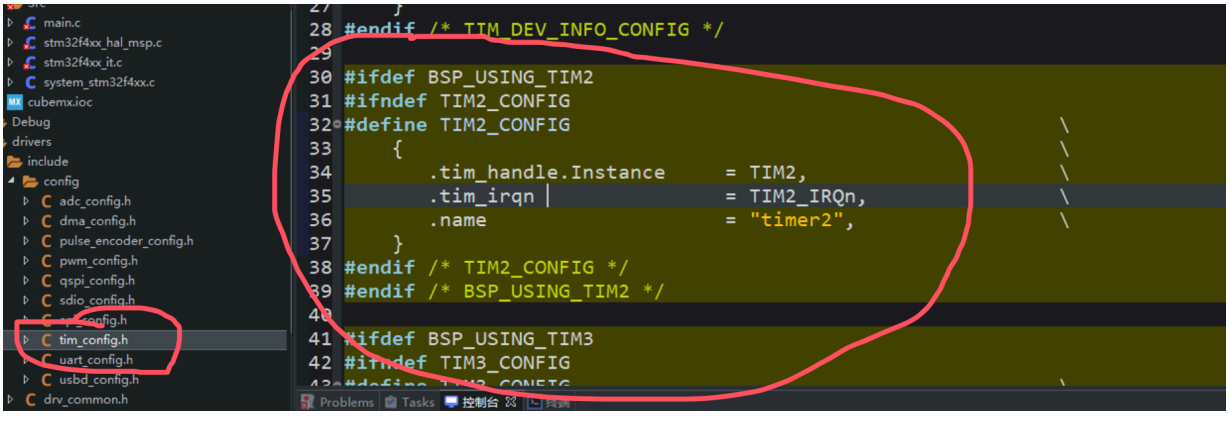



将CubeMx生成的HAL_TIM_Base_MspInit放在board.c中

将stm32f4xx_hal_conf.h中定时器的使能打开




在tim_config中添加tim2的配置(根据下面的注释复制粘贴后修改即可)




若点击编译没有报错,则配置成功

# 书写main.c代码

```c
/*
 * 程序清单：这是一个 hwtimer 设备使用例程
 * 例程导出了 hwtimer_sample 命令到控制终端
 * 命令调用格式：hwtimer_sample
 * 程序功能：硬件定时器超时回调函数周期性的打印当前tick值，2次tick值之差换算为时间等同于定时时间值。
*/

#include <rtthread.h>
#include <rtdevice.h>
/*注册定时器设备*/
#define HWTIMER_DEV_NAME   "timer2"     /* 定时器名称 */

/* 定时器超时回调函数 */
static rt_err_t timeout_cb(rt_device_t dev, rt_size_t size)
{
    rt_kprintf("this is hwtimer timeout callback fucntion!\n");
    rt_kprintf("tick is :%d !\n", rt_tick_get());

    return 0;
}

static int hwtimer_sample(int argc, char *argv[])
{
    rt_err_t ret = RT_EOK;
    rt_hwtimerval_t timeout_s;      /* 定时器超时值 */
    rt_device_t hw_dev = RT_NULL;   /* 定时器设备句柄 */
    rt_hwtimer_mode_t mode;         /* 定时器模式 */
    rt_uint32_t freq = 10000;               /* 计数频率 */

    /* 查找定时器设备 */
    hw_dev = rt_device_find(HWTIMER_DEV_NAME);
    if (hw_dev == RT_NULL)
    {
        rt_kprintf("hwtimer sample run failed! can't find %s device!\n", HWTIMER_DEV_NAME);
        return RT_ERROR;
    }

    /* 以读写方式打开设备 */
    ret = rt_device_open(hw_dev, RT_DEVICE_OFLAG_RDWR);
    if (ret != RT_EOK)
    {
        rt_kprintf("open %s device failed!\n", HWTIMER_DEV_NAME);
        return ret;
    }

    /* 设置超时回调函数 */
    rt_device_set_rx_indicate(hw_dev, timeout_cb);/*设备名+函数名*/

    /* 设置计数频率(若未设置该项，默认为1Mhz 或 支持的最小计数频率) */
    rt_device_control(hw_dev, HWTIMER_CTRL_FREQ_SET, &freq);
    /* 设置模式为周期性定时器（若未设置，默认是HWTIMER_MODE_ONESHOT）*/
    mode = HWTIMER_MODE_PERIOD;//设置为持续定时器中断
```

```c
    ret = rt_device_control(hw_dev, HWTIMER_CTRL_MODE_SET, &mode);//(启动配置)
    if (ret != RT_EOK)
    {
        rt_kprintf("set mode failed! ret is :%d\n", ret);
        return ret;

    }

    /* 设置定时器超时值为5s并启动定时器 */
    timeout_s.sec = 1;      /* 秒 */
    timeout_s.usec = 0;     /* 微秒 */
    if (rt_device_write(hw_dev, 0, &timeout_s, sizeof(timeout_s)) != sizeof(timeout_s))
    {
        rt_kprintf("set timeout value failed\n");
        return RT_ERROR;
    }

    /* 延时3500ms */
    rt_thread_mdelay(3500);

    /* 读取定时器当前值 */
    rt_device_read(hw_dev, 0, &timeout_s, sizeof(timeout_s));
    rt_kprintf("Read: Sec = %d, Usec = %d\n", timeout_s.sec, timeout_s.usec);

    return ret;
}
/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(hwtimer_sample, hwtimer sample);

int main(void)
{
    rt_kprintf("hello");
}
```

## 在终端输入hwtimer_samlpe,开启定时器功能

当前定时器实现功能

```
this is hwtimer timeout callback fucntion!
tick is :25345 !
this is hwtimer timeout callback fucntion!
tick is :26340 !
Read: Sec = 3, Usec = 514600
msh >this is hwtimer timeout callback fucntion!
tick is :27335 !
this is hwtimer timeout callback fucntion!
tick is :28330 !
this is hwtimer timeout callback fucntion!
tick is :29325 !
this is hwtimer timeout callback fucntion!
tick is :30320 !
this is hwtimer timeout callback fucntion!
tick is :31315 !
this is hwtimer timeout callback fucntion!
tick is :32310 !
this is hwtimer timeout callback fucntion!
tick is :33305 !
this is hwtimer timeout callback fucntion!
tick is :34300 !
this is hwtimer timeout callback fucntion!
tick is :35295 !
this is hwtimer timeout callback fucntion!
tick is :36290 !
this is hwtimer timeout callback fucntion!
tick is :37285 !
```