

TinyRTOS51应用手册

- 简介
- 创建Keil C51 项目
- 任务堆栈的估算
- 中断函数
- 用户服务函数API
 - 内核服务函数
 - 临界保护(宏)
 - 二值信号量服务函数
 - 信号量服务函数
 - 事件标志服务函数
 - 消息队列服务函数
 - 消息服务函数
- 配置
- 移植
- 非屏蔽中断NMI
- CPU使用率统计
- 堆栈检查

TinyRTOS51应用手册

简介

TinyRTOS51是一款专用于51单片机的极简RTOS，需要极少的ROM和RAM，采用small编译模式，无需 reentrant可重入函数，可以用于资源很小的51单片机。

任务调度模式：支持协作式和抢先式任务调度模式。

协作式：只在任务主动放弃控制权时才进行任务切换，需要较少的ROM和RAM。

抢先式：高优先级任务可以抢先低优先级任务，相同优先级的任务采用协作式调度，不支持时间片轮流调度。

任务优先级：0-15，数值越大优先级越低，不同任务可以有相同的优先级，最低优先级15固定分配给空闲任务使用，用户可以使用0-14共15个优先级。

系统服务：二值信号量、信号量、事件标志和消息队列。

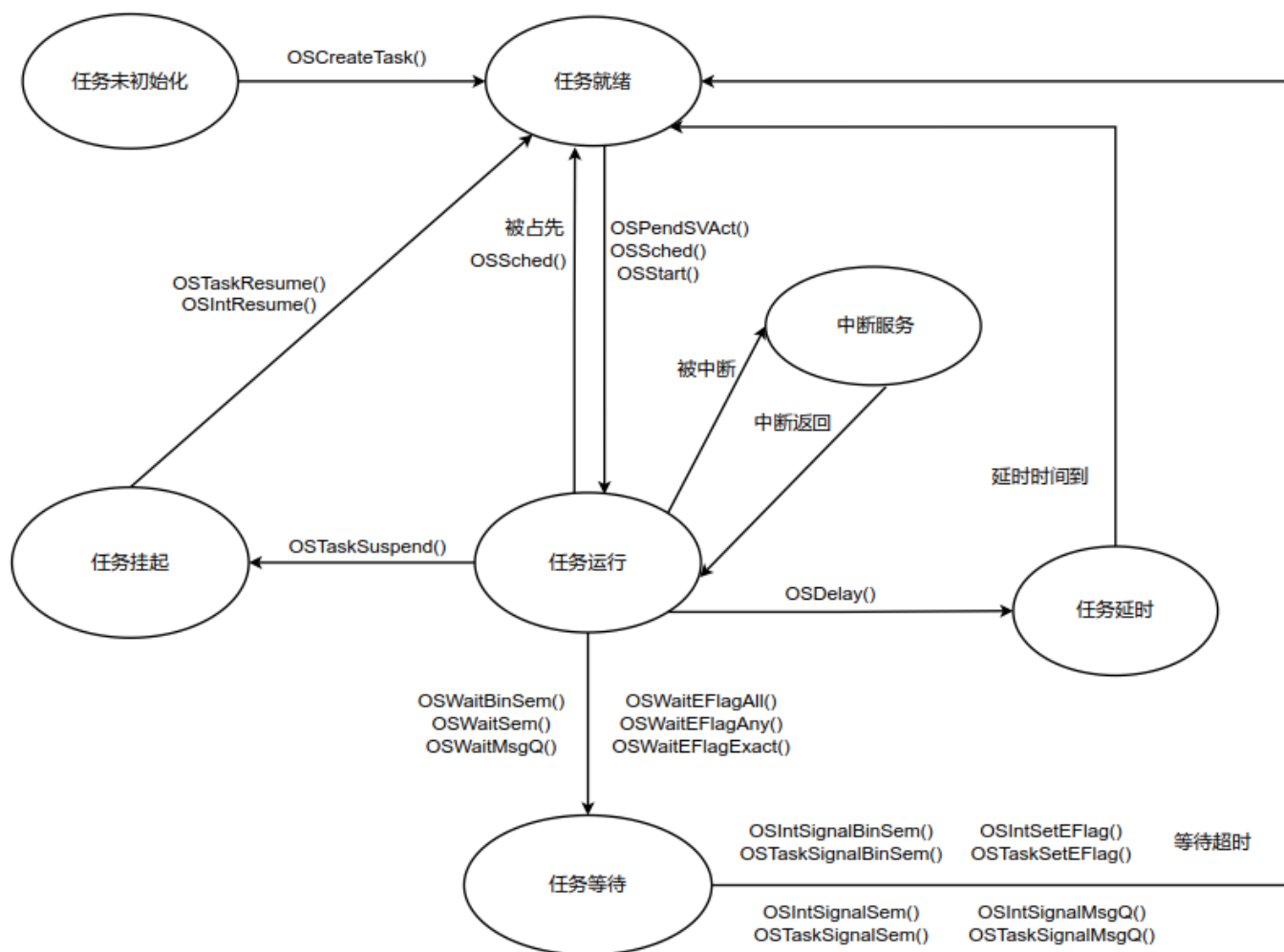
最大任务数限制：任务数 + 二值信号量个数 + 信号量个数 + 事件标志个数 + 消息队列个数 < 256 。

支持reentrant可重入函数：支持small、large模式，不支持compact模式。

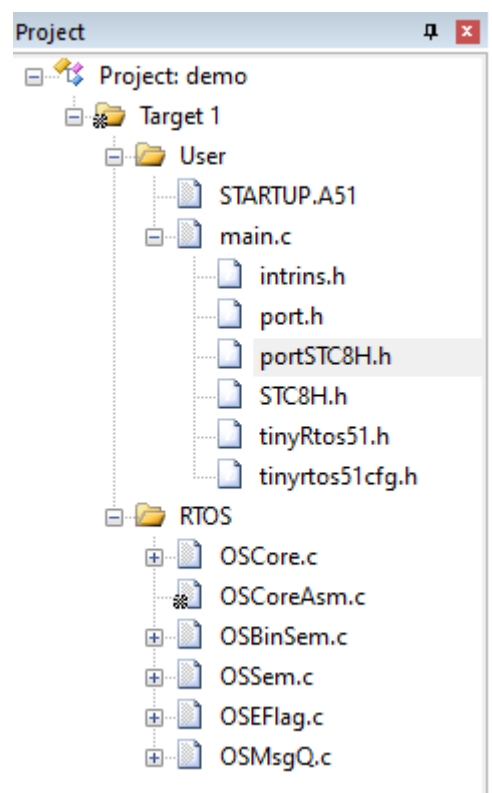
支持动态优先级。

支持非屏蔽中断NMI。

任务状态：



创建Keil C51 项目

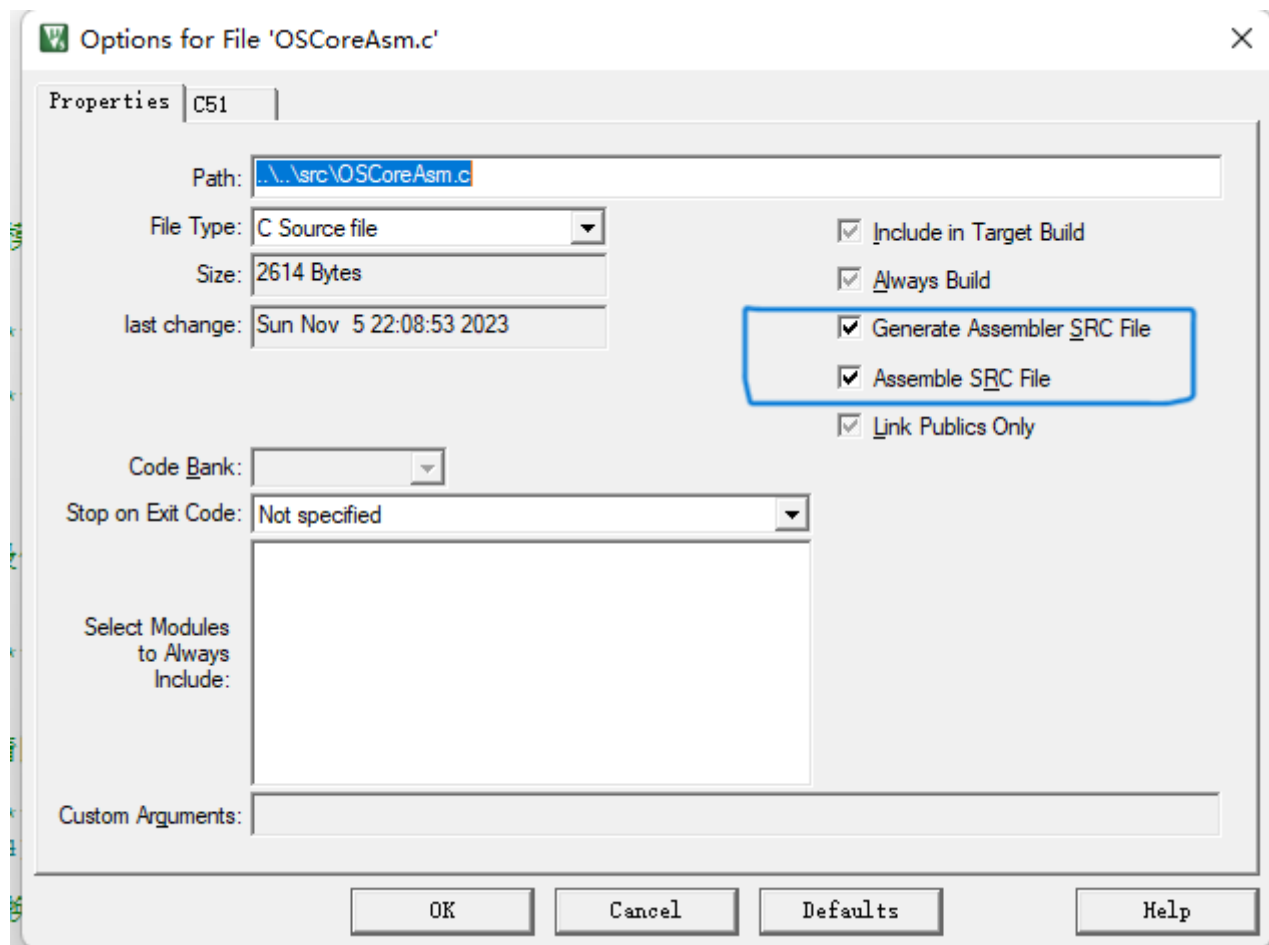


STARTUP.A51配置

- 正确设置IDATALEN和XDATALEN，确保所有变量初始值为0。
- 使能small reentrant支持，IBPSTACK 设为1，IBPSTACKTOP设为 0xFF+1。
- 使能large reentrant支持，XBPSTACK设为1，XBPSTACKTOP设为XRAM的总长度。

OSCore.c：内核C文件。

OSCoreAsm.c：内核C文件，包含有嵌入汇编代码。

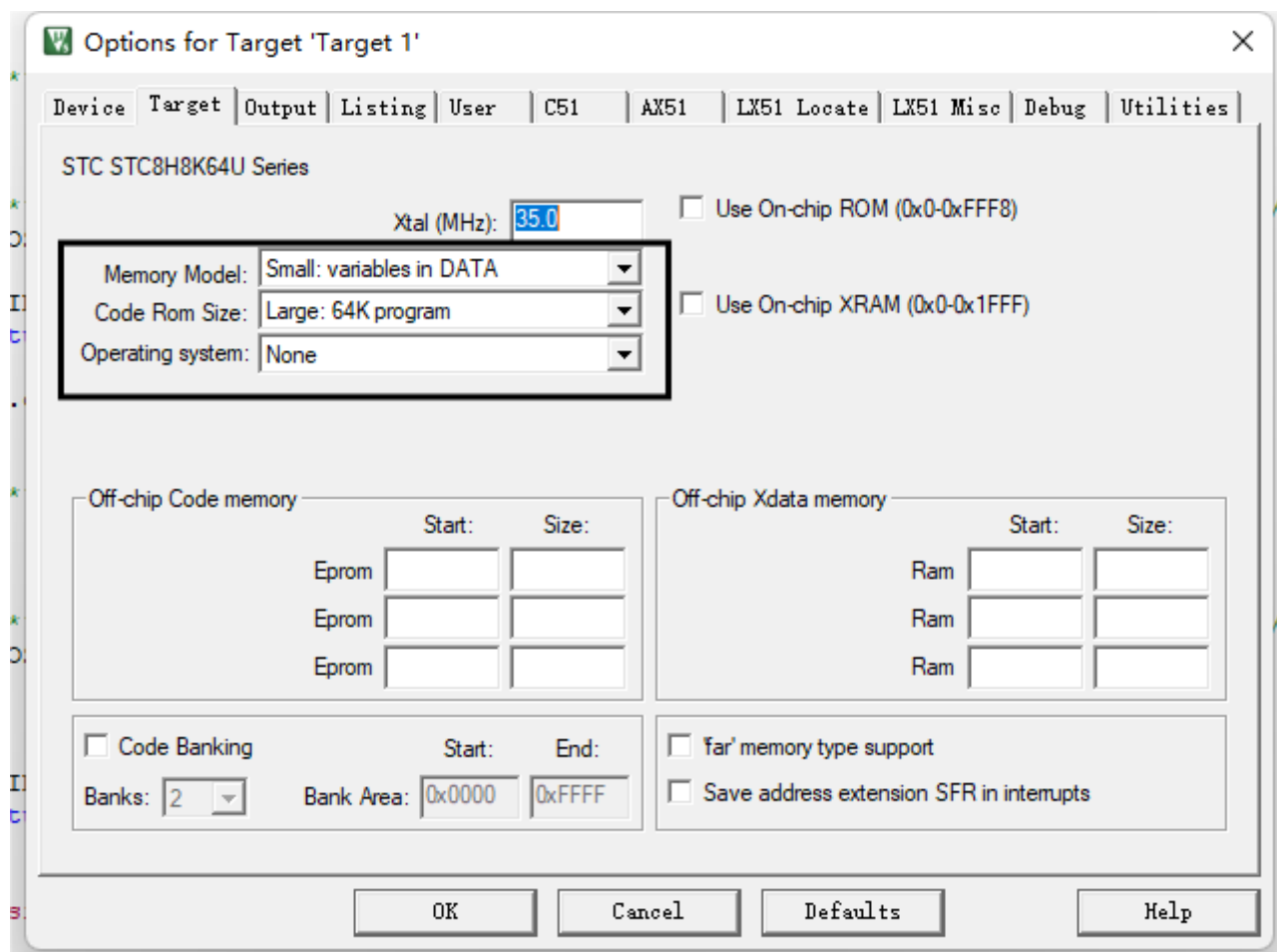
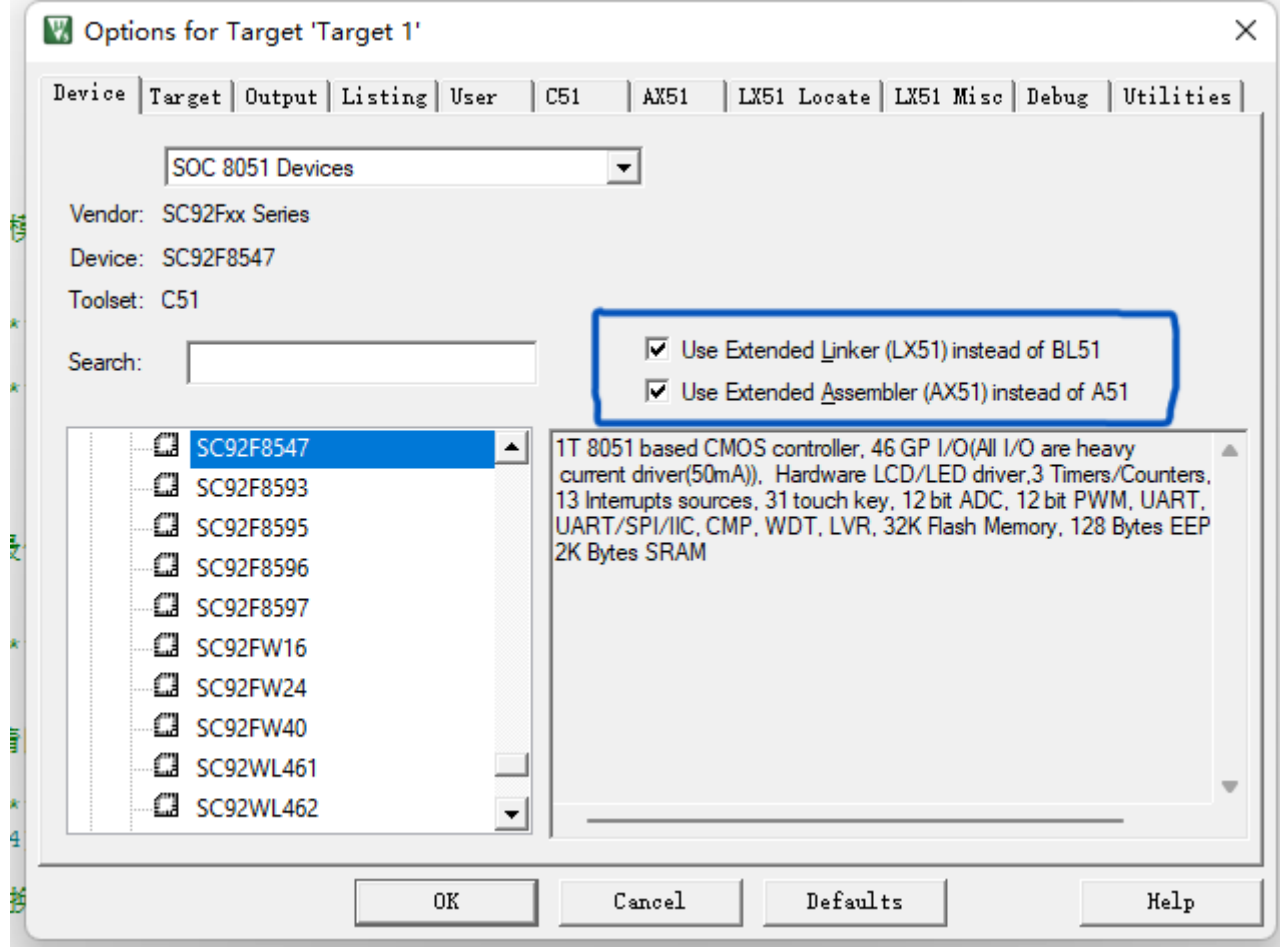


OSBinSem.c：二值信号量

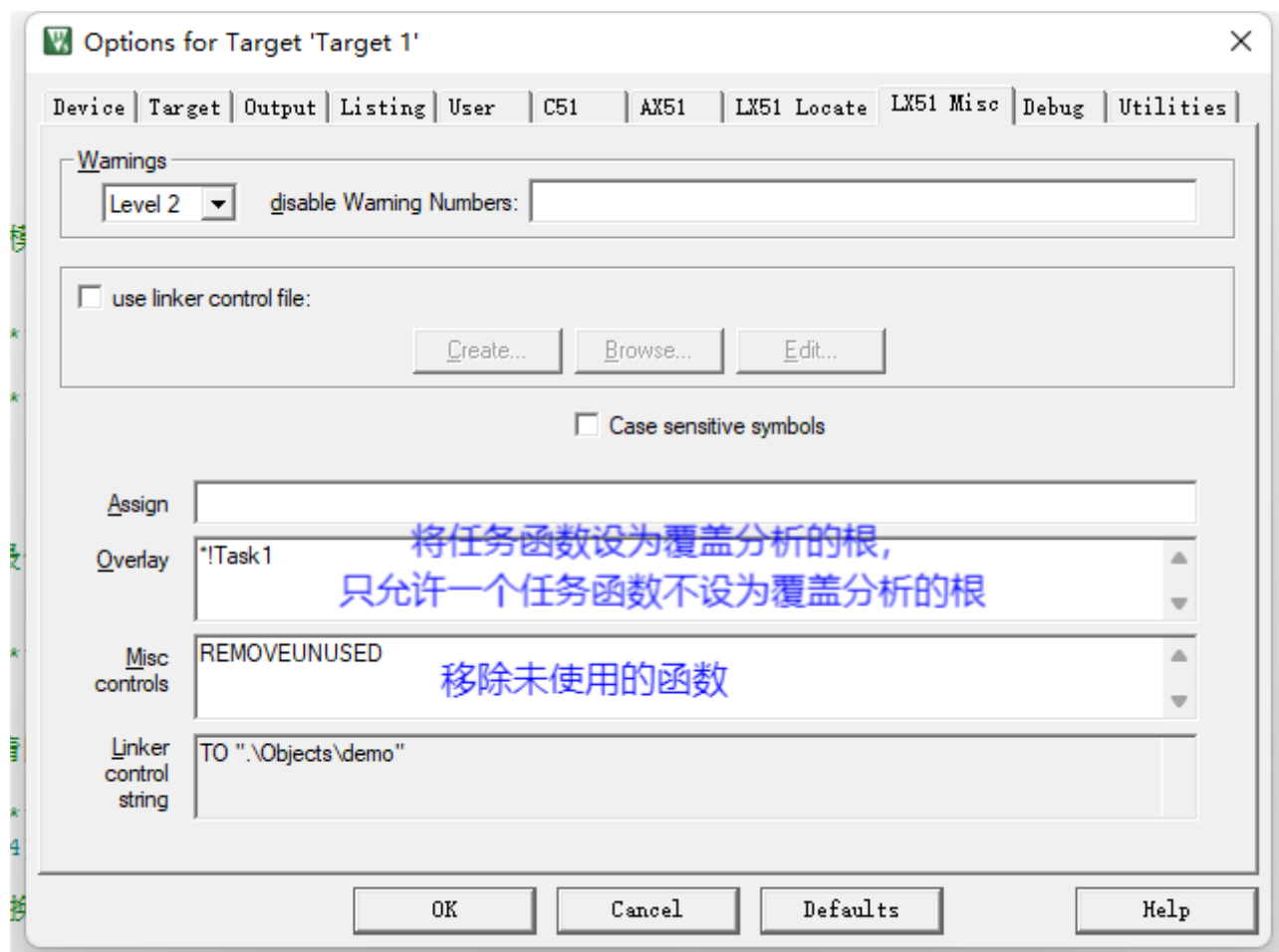
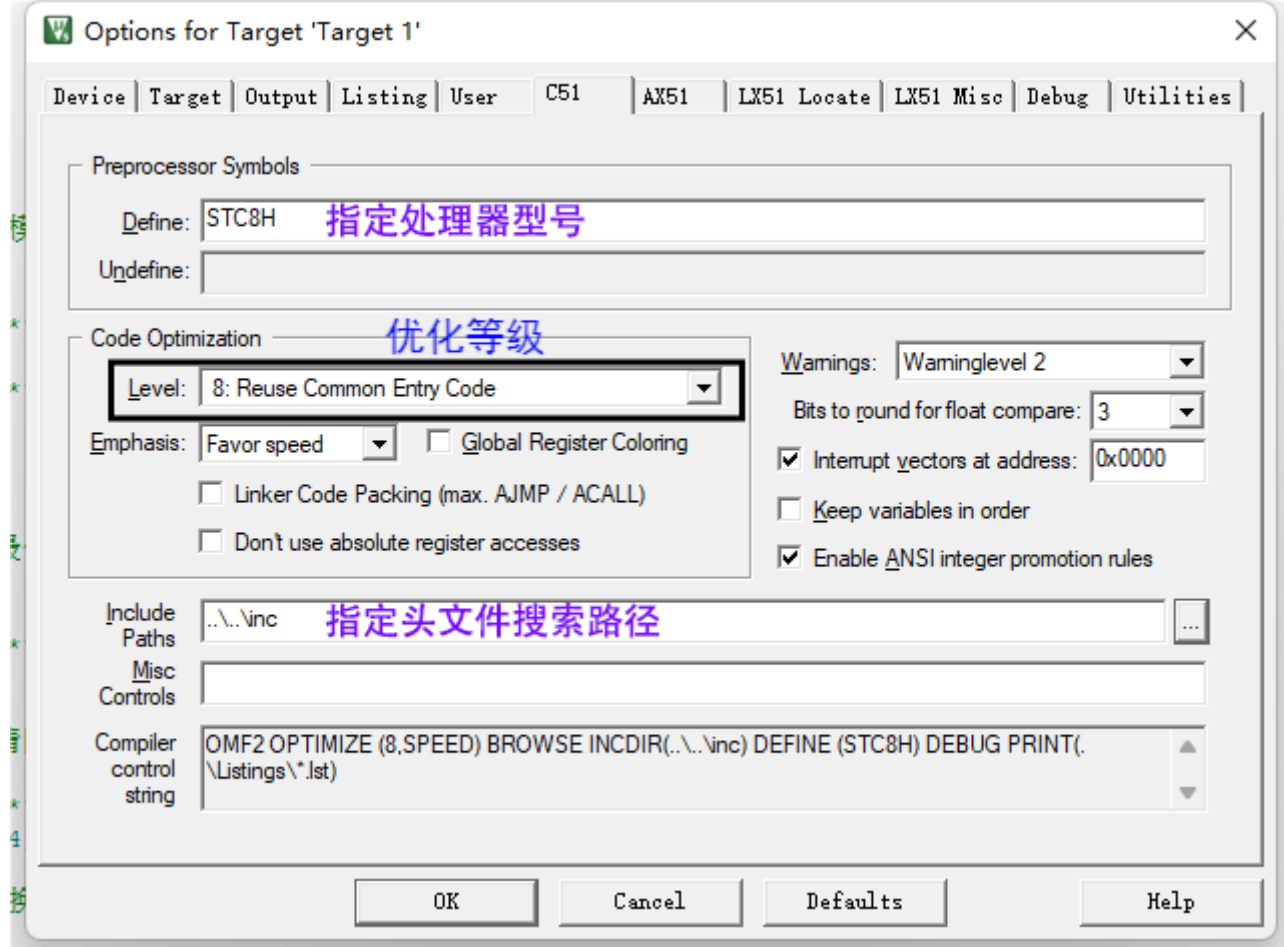
OSSem.c：信号量

OSEFlag.c：事件标志

OSMsgQ.c：消息队列



Memory Model 不要求一定是Small，用户可以根据需要自由选择。



任务堆栈的估算

协作模式：任务调用深度*2+7

抢先模式：

最高优先级任务：任务调用深度*2+7

其它任务：任务调用深度*2+16

任务控制块：3~6 bytes，使能small reentrant 增加2字节，使能large reentrant 增加1~2字节。

任务堆栈，可先设一个较大值，然后再调整。

如果配置为支持reentrant函数，有调用reentrant函数的任务，必须为reentrant函数分配额外的任务堆栈空间，small和large模式的reentrant函数所需的堆栈空间是相互独立的。没有调用reentrant函数的任务，无需额外的堆栈空间。

中断函数

没有调用large reentrant函数的中断服务程序没有特殊要求。

有调用large reentrant函数的中断服务程序，必须跟踪中断嵌套、切换模拟软堆栈。

```
#if OSENABLE_REENTRANT_LARGE > 0
#pragma disable
#endif
void ISR_xxx interrupt xxx
{
    #if OSENABLE_REENTRANT_LARGE > 0
        OSEnterInt() ;
    #endif
    ...
    ...
    #if OSENABLE_REENTRANT_LARGE > 0
        OSExitInt() ;
    #endif
}
```

用户服务函数API

宏定义 **OSSELF**：当前正在运行的任务ID。

以OSTask开头的API只能在任务级调用，不能在中断中调用。

以OSInt开头的API只能在中断中调用，不能在任务级调用。

内核服务函数

void OSInit(void)：系统初始化

void OSStart(void)：系统启动

void OSCreateTask(OSTypeInt8u tID,OSTypeTFP fp,OSTypeInt8u OSLOC_STACKS * stack,OSTypeInt8 prio)：创建任务并加入就绪表

参数：tID - 任务ID fp - 任务函数指针 stack - 任务堆栈指针 prio - 任务优先级
一个任务函数，只能创建一个任务实例。
不支持reentrant函数时使用。

void OSCreateTask(OSTypeInt8u tID,OSTypeTFP fp,OSTypeInt8u OSLOC_STACKS* stack,OSTypeInt8u stackSize,OSTypeInt8u largeStackSize,OSTypeInt8u prio)：创建任务并加入就绪表

参数：tID - 任务ID fp - 任务函数指针 stack - 任务堆栈指针 stackSize - 任务堆栈总长度
largeStackSize - 分配给large reentrant函数的堆栈长度（不支持large reentrant 设为0） prio - 任务优先级
一个任务函数，只能创建一个任务实例。
支持small reentrant函数时使用。

void OSCreateTask(OSTypeInt8u tID,OSTypeTFP fp,OSTypeInt8u OSLOC_STACKS* stack,OSTypeInt8u stackSize,OSTypeInt8u prio)

创建任务并加入就绪表

参数：tID - 任务ID fp - 任务函数指针 stack - 任务堆栈指针 stackSize - 任务堆栈总长度 prio - 任务优先级

一个任务函数，只能创建一个任务实例。

不支持small reentrant函数，只支持large reentrant函数时使用。

void OSSched(void): 任务级任务调度，只能在任务级调用，不能在中断中调用。

void OSTaskSuspend(void): 挂起当前任务,只能在任务级调用。

void OSTaskSuspendID(OSTypeInt8u tID): 挂起指定的任务,只能在任务级调用。

不支持在中断挂起任务，因为在中断中挂起当前任务有可能导致意外的后果。原因分析如下：在中断中挂起当前任务时，会将当前任务的TCB从就绪表中删除，由于是采用模拟pendSV中断来切换任务，程序必须先返回当前任务，然后响应模拟pendSV中断进行任务切换，由于中断响应的时序问题，当前任务有可能先运行一段程序后才响应pendSV中断，如当前任务在这段程序中操作就绪表，就会导致不可预知的意外后果。虽然在中断中挂起其它任务，不会导致这个问题，为了避免意外，不支持在中断挂起任务。

void OSIntResume(OSTypeInt8u tID): 恢复挂起的任务（协作式调度方式不进行任务切换，抢先式调度进行任务切换），只能在中断中调用。

参数：tID - 任务ID

void OSTaskResume(OSTypeInt8u tID) : 恢复挂起的任务（协作式调度方式不进行任务切换，抢先式调度进行任务切换），只能在任务级调用。

参数：tID - 任务ID

void OSDelay(OSTypeOfDelays ticks): 任务延时指定的节拍数，t=0 立即返回，任务进入延时状态(任务阻塞)并进行任务切换，只能在任务级调用。

参数：ticks - 任务延时的节拍数

OSDelayS(t) : 宏, 延时指定的秒数，只能在任务级调用。

void OSTimeTick(): 系统节拍处理函数，在定时器中断中调用。

OSTypeInt8u OSTimeOut(void) : 在任务等待服务函数之后调用，用于测试是否为等待超时。

返回：0 - 不是等待超时 1 - 等待超时

OSTypeInt8u OSTaskStateQuery(OSTypeInt8u tID) : 查询任务状态，只能在任务级调用。

参数：tID: 任务ID

返回值：0(OSTCB_ELIGIBLE)- 就绪 1(OSTCB_SUSPENDED)-挂起 2(OSTCB_DELAYING)-延时 3(OSTCB_WAITING)-等待事件

OSTypeInt8u OSTaskPrioQuery(OSTypeInt8u tID) : 查询任务优先级，只能在任务级调用。

参数：tID: 任务ID

返回值：任务优先级

void OSTaskSetPrio(OSTypeInt8u tID,OSTypeInt8u prio): 设置指定任务的优先级，只能在任务级调用。

参数：tID - 任务ID prio - 任务优先级

void OSIntSetPrio(OSTypeInt8u tID,OSTypeInt8u prio): 设置指定任务的优先级，只能在中断中调用。

参数：tID - 任务ID prio - 任务优先级

void OSTaskSchedLock(void): 在任务中给调度器上锁，只能在任务级调用。调用之后不能调用含有挂起自己操作的API，否则可能导致不可预知的后果。

不能调用的API：OSSched() OSDelay() OSDelayS() OSTaskSuspend() OSSuspendID(OSSELF) OSWaitxxx()

void OSTaskSchedUnLock(void): 在任务中给调度器解锁，并进行任务调度，只能在任务级调用。。

OSTypeInt32u OSTaskGetRunTime(void): 读取运行时间(ticks)，只能在任务级调用。

void OSTaskSetRunTime(OSTypeInt32u t): 设置运行时间(ticks)，只能在任务级调用。

void OSTaskStat(void) : CPU使用率统计。

void OSTaskIRunStackChk() : idata运行堆栈检查(包含small模式reentrant函数的运行堆栈)，将运行堆栈的数据交给自定义堆栈检查函数处理，自定义堆栈检查函数将数据通过串口发往上位机或显示、保存到介质等操作。

void OSTaskXRunStackChk(OSTypeInt8u size): Large reentrant函数模拟软堆栈检查(中断程序使用的软堆栈，任务级的软堆栈在任务堆栈中)，将堆栈的数据交给自定义堆栈检查函数处理，自定义堆栈检查函数将数据通过串口发往上位机或显示、保存到介质等操作。

void OSTaskStackChk(OSTypeInt8u tID,OSTypeInt8u size): 任务堆栈检查(包含small、larges模式reentrant函数任务级的模拟软堆栈)，将堆栈的数据交给自定义堆栈检查函数处理，自定义堆栈检查函数将数据通过串口发往上位机或显示、保存到介质等操作。

临界保护(宏)

OSEnterCritical() 进入临界区

OSExitCritical() 退出临界区

这两个宏，必须严格配对使用，否则可能产生不可预知的后果。

二值信号量服务函数

void OSInitBinSem(OSTypeInt8u eID,OSTypeInt8u state) ： 设置二值信号量初始值。

参数：eID - 二值信号量ID state - 二值信号量初始值。

void OSWaitBinSem(OSTypeInt8u eID): 等待二值信号量，二值信号量为FALSE时任务进入等待状态(任务阻塞)并进行任务切换，禁止延时服务时使用这个函数。

参数：eID - 二值信号量ID

void OSWaitBinSem(OSTypeInt8u eID,OSTypeOfDelays ticks) : 等待二值信号量，有超时服务，二值信号量为FALSE时任务进入等待状态(任务阻塞)并进行任务切换，使能延时服务时使用这个函数。

参数：eID - 二值信号量ID ticks - 超时的节拍数 =0 永久等待

OSTypeInt8u OSTryBinSem(OSTypeInt8u eID) ： 测试二值信号量的状态，并清除二值信号量，不会进入等待状态，可以在中断程序中调用。

参数：eID - 二值信号量ID

返回：二值信号量的状态

OSTypeInt8u OSReadBinSem(OSTypeInt8u eID) ： 读取二值信号量的状态，不改变二值信号量的状态，不会进入等待状态，可以在中断程序中调用。

参数：eID - 二值信号量ID

返回：二值信号量的状态

void OSTaskSignalBinSem(OSTypeInt8u eID) ： 在任务中发送二值信号量，等待二值信号量的最高优先级的任务就绪，协作式调度方式不进行任务切换，抢先式调度则立即进行任务切换。

参数：eID - 二值信号量ID

void OSIntSignalBinSem(OSTypeInt8u eID) ： 在中断中发送二值信号量，等待二值信号量的最高优先级的任务就绪，协作式调度方式不进行任务切换，抢先式调度则立即进行任务切换。

参数：eID - 二值信号量ID

信号量服务函数

void OSInitSem(OSTypeInt8u eID,OSTypeInt8u value) ： 设置信号量初始值。

参数：eID - 信号量ID value - 信号量初始值

void OSWaitSem(OSTypeInt8u eID) ： 等待信号量，信号量为0则任务进入等待状态(任务阻塞)并进行任务切换，禁止延时服务时使用这个函数。

参数：eID - 信号量ID

void OSWaitSem(OSTypeInt8u eID,OSTypeOfDelays ticks) ： 等待信号量，有超时服务，信号量为0则任务进入等待状态(任务阻塞)并进行任务切换，使能延时服务时使用这个函数。

参数：eID - 信号量ID ticks - 超时的节拍数 =0 永久等待

OSTypeInt8u OSTrySem(OSTypeInt8u eID) : 测试信号量, 计数信号大于0返回TRUE, 并使信号量减1, 不会进入等待状态, 可以在中断中调用。

参数: eID - 信号量ID

返回: 0 - 信号量为0 1 - 信号量非0

OSTypeInt8u OSReadSem(OSTypeInt8u eID) : 读取计数信号数值, 不改变信号量数值, 可以在中断中调用。

参数: eID - 信号量ID

返回: 信号量数值

void OSIntSignalSem(OSTypeInt8u eID) : 中断中发送信号量, 等待信号量的最高优先级任就绪, 协作式调度方式不进行任务切换, 抢先式调度则立即进行任务切换。

参数: eID - 信号量ID

void OSTaskSignalSem(OSTypeInt8u eID) : 任务中发送信号量, 等待信号量的最高优先级任就绪, 协作式调度方式不进行任务切换, 抢先式调度则立即进行任务切换。

参数: eID - 信号量ID

事件标志服务函数

void OSInitEFlag(OSTypeInt8u eID,OSTypeInt8u value) : 设置事件标志初始值。

参数: eID - 事件标志ID value - 初始值

void OSWaitEFlagAny(OSTypeInt8u eID,OSTypeInt8u mask) : 等待事件标志(掩码指定的任意位), 掩码指定的标志位都为0则任务进入等待状态(任务阻塞)并进行任务切换, 禁止延时服务时使用这个函数。

参数: eID - 事件标志ID mask - 掩码

void OSWaitEFlagAny(OSTypeInt8u eID,OSTypeInt8u mask,OSTypeOfDelays ticks) : 等待事件标志(掩码指定的任意位), 有超时服务, 掩码指定的标志位都为0则任务进入等待状态(任务阻塞)并进行任务切换, 禁止延时服务时使用这个函数。

参数: eID - 事件标志ID mask - 掩码 ticks - 超时的节拍数 =0 永久等待

void OSWaitEFlagAll(OSTypeInt8u eID,OSTypeInt8u mask) : 等待事件标志(掩码指定的所有位), 掩码指定的标志位中的任意位为0则任务进入等待状态(任务阻塞)并进行任务切换, 禁止延时服务时使用这个函数。

参数: eID - 事件标志ID mask - 掩码

void OSWaitEFlagAll(OSTypeInt8u eID,OSTypeInt8u mask,OSTypeOfDelays ticks) : 等待事件标志(掩码指定的所有位), 有超时服务, 掩码指定的标志位中的任意位为0则任务进入等待状态(任务阻塞)并进行任务切换, 禁止延时服务时使用这个函数。

参数: eID - 事件标志ID mask - 掩码 ticks - 超时的节拍数 =0 永久等待

void OSWaitEFlagExact(OSTypeInt8u eID,OSTypeInt8u mask) : 等待事件标志(和掩码完全匹配), 事件标志和掩码不完全匹配则任务进入等待状态(任务阻塞)并进行任务切换, 禁止延时服务时使用这个函数。

参数: eID - 事件标志ID mask - 掩码

void OSWaitEFlagExact(OSTypeInt8u eID,OSTypeInt8u mask,OSTypeOfDelays ticks) : 等待事件标志(和掩码完全匹配), 有超时服务, 事件标志和掩码不完全匹配则任务进入等待状态(任务阻塞)并进行任务切换, 禁止延时服务时使用这个函数。

参数: eID - 事件标志ID mask - 掩码 ticks - 超时的节拍数 =0 永久等待

OSTypeInt8u OSReadEFlag(OSTypeInt8u eID) : 读事件标志, 不改变事件标志状态, 可以在中断中调用。

参数: eID - 事件标志ID

返回: 事件标志值

void OSIntSetEFlag(OSTypeInt8u eID,OSTypeInt8u mask) : 在中断中设置事件标志, 所有等待事件标志的任务就绪。

参数: eID - 事件标志ID mask - 掩码

void OSTaskSetEFlag(OSTypeInt8u eID, OSTypeInt8u mask) : 在任务中设置事件标志, 所有等待事件标志的任务就绪。

参数: eID - 事件标志ID mask - 掩码

void OSClrEFlag(OSTypeInt8u eID, OSTypeInt8u mask) : 清除事件标志, 不改变任务状态, 事件标志不会自动清除, 用户负责在适当位置清除事件标志。

参数: eID - 事件标志ID mask - 掩码

消息队列服务函数

void OSInitMsgQ(OSTypeInt8u eID,OSTypeInt8u OSLOC_MSGQ_BUF *p,OSTypeInt8u size) : 初始化消息队列

参数: eID - 消息队列ID p - 队列指针 size - 队列长度

void OSWaitMsgQ(OSTypeInt8u eID,OSTypeInt8u idata *msg) : 等待消息队列, 消息队列为空则任务进入等待状态(任务阻塞)并进行任务切换, 禁止延时服务时使用这个函数。

参数: eID - 消息队列ID msg - 消息指针 存储类型必须为idata

void OSWaitMsgQ(OSTypeInt8u eID,OSTypeInt8u idata *msg,OSTypeOfDelays ticks) : 等待消息队列, 有超时服务, 消息队列为空则任务进入等待状态(任务阻塞)并进行任务切换, 使能延时服务时使用这个函数。

参数: eID - 消息队列ID msg - 消息指针 存储类型必须为idata ticks - 超时的节拍数 =0 永久等待

OSTypeInt8u OSTryMsgQ(OSTypeInt8u eID,OSTypeInt8u idata *msg) : 测试消息队列, 消息出列, 不会进入等待状态, 可以在中断中调用。

参数: eID - 消息队列ID msg - 消息指针 存储类型必须为idata

返回: FALSE - 消息队列空 TRUE - 消息队列非空

OSTypeInt8u OSReadMsgQ(OSTypeInt8u eID,OSTypeInt8u idata *msg) : 读取消息队列, 消息不出列, 不会进入等待状态, 可以在中断中调用。

参数: eID - 消息队列ID msg - 消息指针 存储类型必须为idata

返回: FALSE - 消息队列 TRUE - 消息队列非空

void OSTaskSignalMsgQ(OSTypeInt8u eID,OSTypeInt8u msg) : 在任务中发送消息, 消息入列, 等待消息队列的最高优先级任务就绪, 协作式调度方式不进行任务切换, 抢先式调度则立即进行任务切换。

参数: eID - 消息队列ID msg - 消息

void OSIntSignalMsgQ(OSTypeInt8u eID,OSTypeInt8u msg) : 在中断中发送消息, 消息入列, 等待消息队列的最高优先级任务就绪, 协作式调度方式不进行任务切换, 抢先式调度则立即进行任务切换。

参数: eID - 消息队列ID msg - 消息

OSTypeInt8u OSMsgQCount(OSTypeInt8u eID) : 返回消息队列中的消息个数。

参数: eID - 消息队列ID

返回: 消息队列中的消息个数

OSTypeInt8u OSMsgQEmpty(OSTypeInt8u eID) : 返回消息队列剩余可用空间个数。

参数: eID - 消息队列ID

返回: 消息队列剩余可用空间个数

消息服务函数

void OSInitMsg(OSTypeInt8u eID,OSTypeInt8u value): 设置消息初始值 默认为0(没有消息) 在OSStart()前调用

参数: eID: 消息ID value:消息初始值

void OSWaitMsg(OSTypeInt8u eID,OSTypeInt8u idata *msg,OSTypeOfDelays ticks): 等待消息 在任务中调用 不能在中断中调用 有超时服务

eID: 消息ID msg: 消息指针,用于接收消息,存储类型必须为idata,发生超时则消息指针指向的数据是不可预知的 ticks: 超时时间, =0 为永久等待

void OSWaitMsg(OSTypeInt8u eID,OSTypeInt8u idata *msg): 等待消息 在任务中调用 不能在中断中调用 没有超时服务

eID: 消息ID msg: 消息指针,用于接收消息, 存储类型必须为idata

OSTypeInt8u OSTryMsg(OSTypeInt8u eID): 测试并清除消息 不进入等待状态 可以在中断中调用

参数: eID: 消息ID 返回: 消息值,0-表示没有消息

OSTypeInt8u OSReadMsg(OSTypeInt8u eID): 读取但不清除消息、不进入等待状态, 可以在中断中调用

参数: eID: 消息ID 返回: 消息值,0-表示没有消息

void OSTaskSignalMsg(OSTypeInt8u eID,OSTypeInt8u msg): 在任务中发送消息

参数: eID: 消息队列ID msg: 消息 0-表示没有消息

void OSIntSignalMsg(OSTypeInt8u eID,OSTypeInt8u msg): 在中断中发送消息

参数: eID: 消息队列ID msg: 消息 0-表示没有消息

配置

在tinyrtos51cfg.h文件中配置RTOS

OSTASKS: 任务数量

OSMETHOD: 任务调度模, 0 - 协作式调度 1 - 抢先式调度

OSBYTES_OF_DELAYS: 延时计数器字节数(0,1,2) 默认为1 0-禁止延时及超时服务

OSTIMER_PRESCALAR: 配置OSTimer()的预分频(0 - 255)) 0,1-禁用预分频

OSTICKS_PER_SEC: 每秒的tick数

OSENABLE_RUNTIME: 使能运行时间(ticks)

OSLOC_LIST: 链表存储位置, idata 或 xdata

OSLOC_TCB: TCB 任务控制块存储位置, idata 或 xdata

OSLOC_STACKS: 任务堆栈存储位置, pdata 或 xdata

OSLOC_BINSEM: BinSem ECB存储位置, idata 或 xdata

OSLOC_SEM: Sem ECB存储位置, idata 或 xdata

OSLOC_EFLAG: EFlag ECB存储位置, idata 或 xdata

OSLOC_MSGQ: MsgQ ECB存储位置, idata 或 xdata

OSLOC_MSGQ_BUF: MsgQ 队列存储位置, idata 或 xdata

OSLOC_MSG: Msg ECB存储位置, idata 或 xdata

OSLOC_RUNTIME: 运行时间(ticks)变量存储位置, idata或xdata

OSLOC_STAT: 统计变量的存储位置, idata或xdata

OSLOC_STACK_CHK: 堆栈检查变量存储位置, idata或xdata

OSENABLE_REENTRANT_SMALL: 使能small reentrant可重入函数

OSENABLE_REENTRANT_LARGE: 使能large reentrant可重入函数

OSENABLE_SUSPENDID: 使能挂起其它任务(OSTaskSuspendID())

OSENABLE_SETPRIO: 使能动态优先级(OSTaskSetPrio(),OSIntSetPrio())

OSENABLE_SCHEDLOCK: 使能调度器锁

OSENABLE_STAT: 使能CPU使用率统计

OSENABLE_STAT_HOOK: 使能CPU使用率统计钩子函数

OSENABLE_STAT_MAX: 使能CPU最大使用率统计

OSENABLE_STACK_CHK: 使能堆栈检查

OSBINSEMS: 二值信号量个数 0 - 禁用二值信号量

OSSEMS: 信号量个数 0 - 禁用信号量

OSEFLAGS: 事件标志个数 0 - 禁用事件标志

OSMSGQS: 消息队列个数 0 - 禁用消息队列

OSMSGs: 消息个数 0 - 禁用消息

OSENABLE_ERROR_CHECKING: 0 - 禁止运行时错误检查 1 - 使能运行时错误检查

注意: OSTASKS+OSBINSEMS+OSSEMS+OSEFLAGS+OSMSGQS+OSMSGSGS < 256

移植

TinyRtos51是专用于51架构的MCU，移植到其它架构的MCU，改动较大。

以下移植是指移植到不同厂家、不同型号的51架构的MCU。

在移植文件portxxx.h中定义模拟pendSV指令的中断号，初始化及触发方式。

在port.h中指定不同型号MCU对应用portxxx.h文件。

非屏蔽中断NMI

TinyRTOS51,支持非屏蔽中断，也就是说临界区不用关闭总中断EA，只需要关闭有调用API的中断，没有调用API的中断可以不关闭，这些没有调用API的中断就称为非屏蔽中断，它们在临界区可以不关中断。对响应时间要求高，无法容忍临界区关中断造成的响应延迟的信号，可以做为非屏蔽中断处理。

使用方法：

1. 在portxxx.h中重新定义OSDi()、OSEi()宏，不关闭总中断EA，只关闭和恢复有调用API的中断。
2. 在调用OSStart()之前，必须开总中断EA，但要关闭有调用API的中断。
3. 非屏蔽中断不能调用API，同时要注意共享变量的保护。

CPU使用率统计

CPU使用率统计需要用到最高优先级0和次低优先级14，其它任务使用优先级1~13。

CPU使用率统计必须使能挂起指定任务、动态优先级。

可以选择统计当前使用率或最大使用率。

使用方法：

1. 创建一个运行时间统计任务，优先级为最高优先级0，在统计任务中调用API OSTaskStat()。
2. 运行足够长的时间后，在仿真时暂停程序运行，直接查看CPU使用率OSUsageRate变量或者使能钩子函数，系统将调用用户自定义的钩子函数，并将OSUsageRate传递给钩子函数，钩子函数将以某种形式显示OSUsageRate的数值或通过串口发送到上位机。

堆栈检查

由于任务堆栈是静态分配的，可以用以下两种方法检查堆栈。

1. 不使用堆栈检查API，在仿真运行足够长的时间后，暂停程序运行，直接查看堆栈的数据，未使用到存储器为连续的0。
2. 使用堆栈检查API，在任务中调用堆栈检查API，将堆栈数据交给自定义堆栈检查函数处理，自定义堆栈检查函数将数据通过串口发往上位机或显示、保存到介质等操作。堆栈检查API，不含临界保护，因此在调用前要先进入临界区，调用后退出临界区。