# Mooresilicon Cygnus SDK FreeRTOS移植设计

© 2022 Mooresilicon All rights reserved.

本文档版权归TCL摩星半导体所有，受相关法律法规的保护。未经书面许可不得复制传播。

# 修订历史

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 作者 | 说明 |
| draft | 2022-03-24 | 区伟权 | 起草 |
| 1.0 | 2022-03-25 | 区伟权 | 按评审意见修改，完成初稿 |
| 1.1 | 2022-04-06 | 区伟权 | 1. 增加版权声明 2. 增加修订历史和目录 3. 更新时间管理描述 4. 增加内存管理描述 5. 增加设备管理描述 |
| 1.2 | 2022-04-07 | 区伟权 | 1. 增加中断优先级描述 2. 增加ASIC芯片的时钟方案描述 3. 增加内存划分描述 4. 增加UART驱动初始化描述 |
| 1.3 | 2022-04-08 | 区伟权 | 1. 增加目的和范围描述 2. 修改正硬件约束条件的RAM描述 3. 增加任务运行时间统计描述 |

目录

[Mooresilicon Cygnus SDK FreeRTOS移植设计 1](#_Toc100300489)

[修订历史 2](#_Toc100300490)

[目的 4](#_Toc100300491)

[范围 4](#_Toc100300492)

[概述 4](#_Toc100300493)

[设计约束 4](#_Toc100300494)

[零层设计 5](#_Toc100300495)

[设计规格 5](#_Toc100300496)

[CPU管理 8](#_Toc100300497)

[中断管理 9](#_Toc100300498)

[时间管理 10](#_Toc100300499)

[任务管理 10](#_Toc100300500)

[内存管理 11](#_Toc100300501)

[功耗管理 13](#_Toc100300502)

[设备管理 14](#_Toc100300503)

[配置项 15](#_Toc100300504)

# 目的

多任务并发系统，能很好地支持业务多样性。同时考虑MCU轻量级芯片嵌入式系统，Cygnus SDK集成RTOS是最优的选择。本文描述RTOS的设计思路，主要是FreeRTOS的移植设计。

# 范围

本文用于指导FreeRTOS的移植实现，和基于FreeRTOS的应用开发和设备驱动开发。读者是相关的软件开发工程师。

# 概述

本文主要描述FreeRTOS在RISC-V芯来Soc和Cygnus SDK的移植设计。FreeRTOS是目前主流的RTOS，主要有以下优点：

* 免费
* 开源
* MIT License，商务法律风险低
* 十分小巧，适用于小型MCU系统
* 与主流半导体厂商合作，在30种以上芯片架构上进行了移植
* 社区活跃，版本持续演进
* 社会占有量高，是主流的RTOS

同时也有以下缺点：

* 不支持文件系统和网络系统管理，不利于构建大型系统；
* 不支持设备管理，不利于集成PCIE和USB等复杂设备驱动；
* 内存管理算法简单，大型复杂业务下，不利于内存碎片管理，导致内存利用率下降；

# 设计约束

硬件约束：

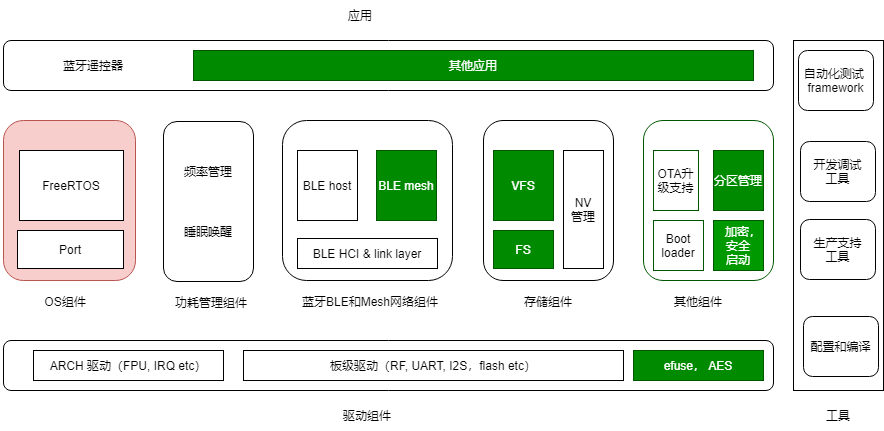
* 芯来N300系列RISC-V体系架构32位CPU Core，144MHz频率；
* 系统SRAM，大小96KB + 128KB SRAM用作rom备案 + I Cache 16KB；
* 4KB Ret\_RAM +（28KB + 4KB）EX\_RAM

软件约束：

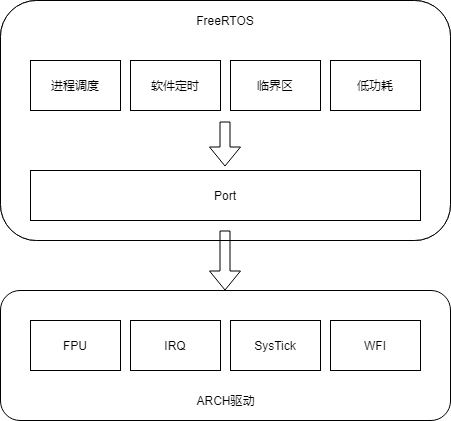
* 芯来RISC-V GCC编译器及工具链；
* 基于芯来SDK v0.3.4的RISC-V arch部分和FreeRTOS v10.4.6开源包部分进行移植；

# 零层设计

FreeRTOS在系统中的位置如下图所示。FreeRTOS是系统的基础组件，向其它组件提供：任务管理、任务间通信、内存管理、时间控制、定时器、临界区，等基础功能。实现多任务并发的实时系统。



OS组件包括两部分：FreeRTOS和Port 。Port是FreeRTOS的CPU适配层，对下调用ARCH驱动的接口控制CPU运算单元和控制单元，包括：各运算指令、中断控制、系统时钟控制、低功耗等。对上屏蔽CPU的差异，给FreeRTOS提供标准化的功能接口，包括：任务调度、上下文切换、软件定时、临界区、低功耗等，如下图所示。



# 设计规格

FreeRTOS需要支持的功能规格如下表所示。

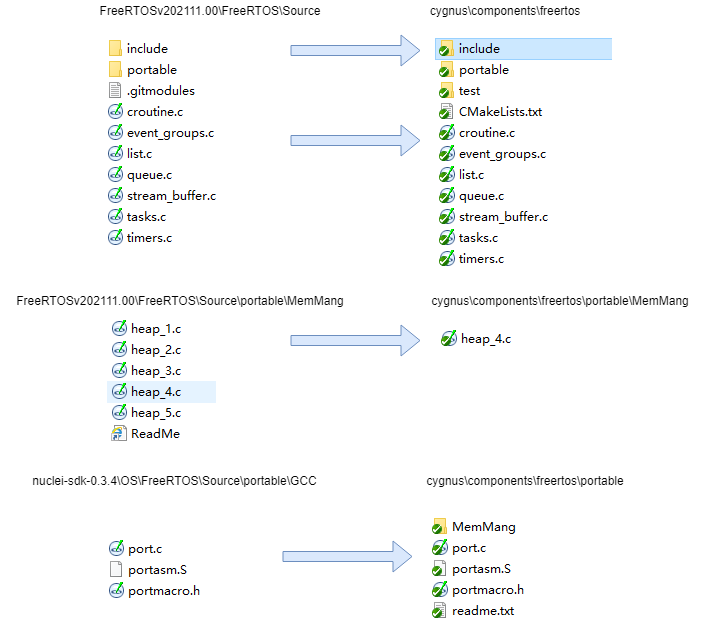
|  |  |  |  |
| --- | --- | --- | --- |
| DR | AR | 描述 | 支持 |
| CPU管理 | RISC-V | 支持RISC-V体系架构，芯来N300系列SoC | Y |
| 任务管理 | 任务的创建和删除 |  | Y |
|  | 任务的挂起和恢复 |  | Y |
|  | 任务获取 | 支持： 1.根据任务名获取任务句柄 2.获取当前任务句柄 | Y |
|  | 任务状态机 | 支持状态机：运行态/就绪态/阻塞态/挂起态 支持状态获取 | Y |
|  | 任务优先级 | 支持任务优先级的配置获取。优先级范围0~7，0表示最低优先级，7表示最高 | Y |
|  | 任务堆栈 | 每个任务有独立堆栈，任务切换时保持CPU寄存器等任务现场 | Y |
|  | 任务TRACE | 支持可视化输出： 1.所有任务的状态 2.每个任务的运行时间 | Y |
| 任务调度 | IDLE任务 | 启动任务调度，内核自动创建IDLE任务 | Y |
|  | 抢占式调度 | 在系统时钟中断处理进行任务切换 | N |
|  | 非抢占调度 | 任务阻塞或者主动放弃CPU，触发任务切换 | Y |
|  | 优先级调度 | 触发任务调度时，选择就绪任务中，优先级最高的任务调度 | Y |
|  | 时间片调度 | 任务运行一个时间片，即一个时钟节拍后让出CPU。需支持抢占式调度 | N |
|  | 任务主动切换 |  | Y |
|  | 开启关闭任务调度器 |  | Y |
|  | 挂起恢复任务调度器 |  | Y |
| 时间管理 | 系统时钟节拍 | 响应Machine timer interrupt，SysTick中断处理，累计系统Tick | Y |
|  | 系统软中断 | SysTick中断处理中，软件触发SWI 软中断，产生抢占式时间片调度 | N |
|  | 时间延迟 |  | Y |
| 中断管理 | 打开关闭中断 | 支持： 1.打开关闭单个中断 2.打开关闭所有中断，即临界区操作 | Y |
| 临界区 | 进入退出临界区 | 支持： 1.任务进出临界区 2.中断服务程序进出临界区 | Y |
| 队列 | 队列创建删除 |  | Y |
|  | 队列复位 |  | Y |
|  | 消息入队 | 支持任务上下文和中断上下文入队 | Y |
|  | 消息出队 |  | Y |
|  | 队列锁 | 支持 1.入队锁，上锁解锁 2.出队锁，上锁解锁 | Y |
| 信号量 | 二值信号量 | 二值信号量的创建/获取/释放 | Y |
|  | 计数型信号量 | 计数型信号量的创建/获取/释放 | Y |
|  | 互斥信号量 | 互斥信号量的创建/获取/释放 | Y |
|  | 递归信号量 | 递归信号量的创建/获取/释放 | Y |
| 软件定时器 | 软件定时器的创建删除 |  | Y |
|  | 软件定时器的开启停止 |  | Y |
|  | 软件定时器的复位 |  | Y |
| 事件标志组 | 事件标记组的创建删除 |  | Y |
|  | 设置获取事件位 |  | Y |
|  | 等待事件位 |  | Y |
| 任务通知 | 发送获取任务通知 |  | Y |
| 低功耗 | CPU低功耗 | 支持WFI指令进入睡眠模式 支持NMI输入信号/中断/事件推出睡眠模式 | Y |
|  | Tickless模式 | IDLE任务关闭系统时钟节拍 获取当前还有多长时间运行，作为系统时钟的补偿 系统时钟计数到补偿时间，产生中断唤醒CPU | Y |
|  | 外设低功耗 | CPU进入睡眠模式前，配置外设低功耗 CPU被唤醒后，配置外设退出低功耗 | Y |
| 内存管理 | heap\_1 | 应用场景： 1.系统资源如：任务/信号量/队列一旦创建好就不会释放 2.具有可确定性，不会产生内存碎片 3.实现简单，内存从静态数组分配，适用于不需要动态内存的应用 | N |
|  | heap\_2 | 应用场景： 1.应用会重复创建删除任务/队列/信号量等系统资源，会产生内存碎片 2.应用每次申请的内存大小都是一样的 3.内存碎片可以简单合并 | N |
|  | heap\_3 | 编译器提供内存堆，和malloc/free函数 | N |
|  | heap\_4 | 1.应用会重复创建删除任务/队列/信号量等系统资源，会产生内存碎片 2.申请的内存大小可以是随机的 3.可管理内存碎片和空闲内存 | Y |
|  | heap\_5 | 在heap\_4的基础上，允许内存跨越多个不连续的内存段。可支持片外扩展内存。 | N |

# CPU管理

FreeRTOS文件目录如下所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 目录文件 | 二级目录文件 | 描述 | 移植 |
| include |  | FreeRTOS的头文件 | FreeRTOS开发包  \FreeRTOS\Source\include  增加FreeRTOSConfig.h |
| portable |  | CPU ARCH相关 |  |
|  | MemMang | 内存管理相关 | \FreeRTOS\Source\portable\MemMang  保留heap\_4.c，其它删除 |
|  | port.c | Port适配层 | 芯来SDK开发包  \OS\FreeRTOS\Source\portable\GCC |
|  | portasm.S | 创建首任务；  软件中断处理； |
|  | portmacro.h | Port适配层头文件 |
| croutine.c |  | 协程相关，可忽略 | FreeRTOS开发包  \FreeRTOS\Source |
| event\_groups.c |  | 事件管理接口 |
| list.c |  | 链表接口 |
| queue.c |  | 队列管理接口 |
| stream\_buffer.c |  | 流缓冲区接口 |
| tasks.c |  | 任务管理接口 |
| timers.c |  | 软件定时器接口 |

移植关系如下图所示。



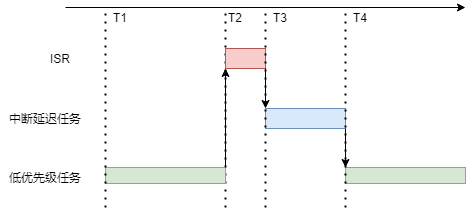
# 中断管理

**中断向量表**，在ARCH驱动（\components\plantform\arch\ startup\_NUCLEI\_N.S）管理。7号中断是系统时钟中断，ISR指向FreeRTOS Port层的处理函数xPortSysTickHandler 。

**中断优先级**，取值范围0~7，7表示最高优先级，0表示最低。使用ECLIC的中断源级别实现。系统时钟中断和软件中断（SWI）优先级最低。

**ISR（Interrupt Service Routine）**，应尽量降低时间开销。尽可能提升多任务实时系统的响应能力，系统均衡运行，减少抖动。对涉及大量业务的中断处理，可使用延迟服务任务。任务阻塞等待业务到来处理。ISR负责获取中断来源，清除中断，唤醒任务。为避免中断嵌套，ISR中应禁止中断，结束前打开中断。

**中断延迟任务**，阻塞等待ISR唤醒，进行业务处理。在任务中，可以执行较为冗长的业务操作，能使用OS的API，能执行IO访问等不确定性的操作。运行过程如下图所示，T2时刻产生中断，低优先级任务被ISR抢占；ISR T3时刻唤醒中断延迟服务；T4时刻，中断延迟任务执行完成，重新阻塞等待业务。低优先级任务重新被调度运行。



# 时间管理

系统时钟节拍设置为1毫秒。时钟节拍计数变量定义为64位整型变量，与芯片SysTimer 64位设计同步，且避免计数器翻转。

FGPA的SysTimer和CPU时钟频率都是48MHz 。因此配置SysTimer的计数器为48000 。

ASIC的CPU时钟频率是144MHz。SysTimer配置为CPU时钟的16分频（可配置为2~16分频），即9MHz。因此ASIC的SysTimer计数器配置为9000。

# 任务管理

**IDLE任务**，调度器启动时默认创建的任务，优先级最低。无业务时，CPU执行的任务，在其中执行低功耗相关处理。

**喂狗任务**，软件定时器喂狗。在狗复位周期前喂狗，无需频繁喂狗。优先级最高。如果业务任务长时间占用CPU不释放，定时器得不到调度，则系统被狗复位。

**业务任务**，APP任务、OTA任务、蓝牙业务任务。中等优先级。

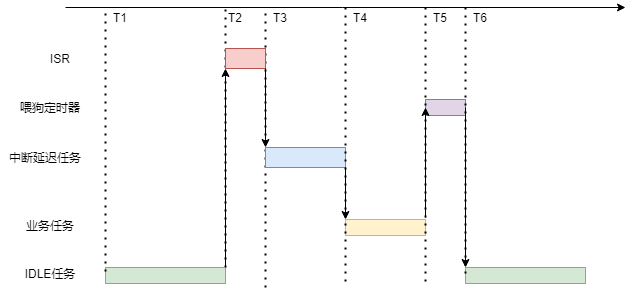
**中断延迟任务**，蓝牙数据收发中断、串口输入中断、DMA中断、外设接口按键中断，都可以使用延迟任务。各延迟任务优先级相同，使用次高优先级。

**各任务优先级**，分配如下表所示：

|  |  |  |
| --- | --- | --- |
| 任务 | 描述 | 优先级，范围0~7  0最低，7最高 |
| IDLE任务 | 空闲任务 | 0 |
| 喂狗任务 | 定时器喂狗 | 7 |
| 业务任务 |  | 2 |
| 中断延迟任务 |  | 6 |

**非抢占调度**，适合嵌入式系统。嵌入式系统是专用设备，业务固定，任务不多；且任务多与外设交互，频繁被IO阻塞，主动让出CPU调度。如果是抢占式调度，每个时钟Tick都做时间片检查，发起任务调度，则系统开销大。

**各任务运行关系**，如下图所示：T2时刻产生业务中断，进入ISR处理中断；T3时刻ISR唤醒中断延迟任务，并退出；中断延迟任务，处理中断相关业务，在T4时刻重新阻塞等待下一次业务，主动让出CPU调度；业务任务，因等待条件满足，进入就绪状态，在T4被调度器选择运行；T5时刻，业务任务等待下一次业务条件，主动让出CPU调度；喂狗定时器到时，任务就绪，被调度运行，喂狗。T6时刻，喂狗结束，重新计时，主动让出CPU，当前无就绪业务任务，调度IDLE任务。



**任务运行时间统计**，使用外设Timer，周期设置成50微秒。在Port层实现FreeRTOS接口：

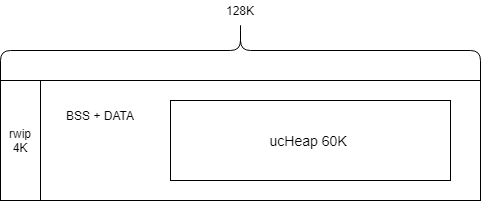
portCONFIGURE\_TIMER\_FOR\_RUN\_TIME\_STATS，任务调度器启动，时初始化Timer；

portGET\_RUN\_TIME\_COUNTER\_VALUE，任务切换时，获取Timer时间计数，记录任务运行时间；任务调度器结束时，停止并释放Timer。

FreeRTOS提供vTaskGetRunTimeStats接口。命令行等前台任务可用此接口实时获取各任务运行时间统计，格式化输出。

# 内存管理

支持heap\_4内存管理方式。操作系统的堆内存使用全局变量字符数组ucHeap 管理。芯片内存支持128Kbytes/ 128kbytes+96kbytes 。考虑芯片两种RAM模式，ucHeap数组长度配置为60Kbytes 。RAM划分如下图所示，rwip 4K预留给BT固化在 ROM的部分使用的变量，BSS+DATA使用剩余的124K/220K，其中包括操作系统的60K堆内存。

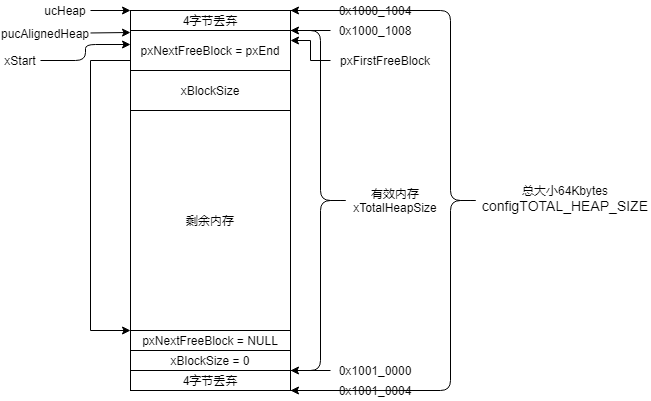


**内存块描述符**，定义为BlockLink\_t类型。存储于内存块的起始位置。成员xBlockSize最高位表示内存是否被使用，低位表示内存大小。因此最大支持内存块0x7FFF\_FFFF减去内存块描述符大小，再减去字节对齐大小。

**初始化**，函数prvHeapInit 实现步骤如下：

1. 内存堆起始地址做字节对齐；
2. 起始地址对齐后，有及字节被抛弃，因此内存堆总的可用大小要重新计算；
3. pucAlignedHeap 是对齐后的可用起始地址；
4. 初始化xStart为可用内存块链表头；
5. 初始化pxEnd为可用内存链表尾，pxEnd放到内存堆末尾；
6. 每个内存块有描述符，定义为BlockLink\_t变量类型，此处初始化首个内存块，size是整块堆内存；
7. xMinimumEverFreeBytesRemaining记录最小空闲内存块的大小，xFreeBytesRemaining表示内存堆剩余大小；
8. 初始化静态变量xBlockAllocatedBit 最高位置1，对于32位CPU即为0x8000\_0000；此变量用于检查内存块是否被使用；

内存配置64Kbytes，8字节对齐。字节对齐，可通过portBYTE\_ALIGNMENT编译宏控制。假设ucHeap数组地址为0x1000\_0004，初始化后，得出内存结构如下图所示。



**申请内存**，函数pvPortMalloc处理步骤如下：

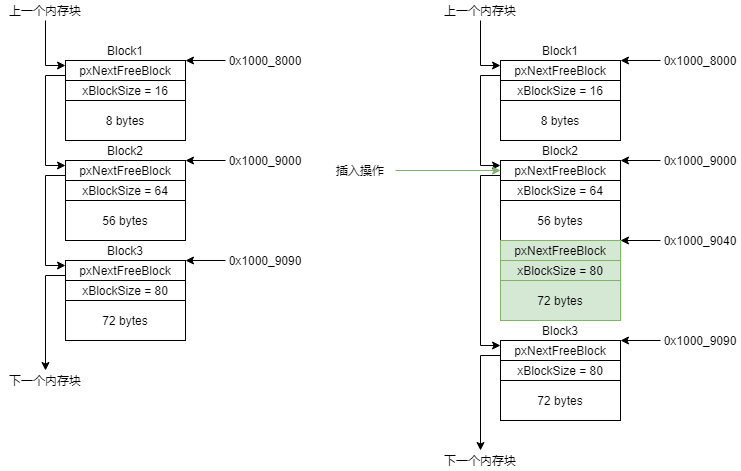
1. pxEnd如果为空，说明内存堆未初始化，因此要先执行初始化prvHeapInit；
2. 申请的size，最高位不能为1；
3. 实际所需的内存要加上内存块描述符的大小；还要对最终的size做字节对齐处理。
4. 从空闲链表头xStart开始，查找满足所需内存大小的内存块，pxPreviousBlock的下一个内存块就是找到的可用内存块；
5. 找到的内存块不能是pxEnd；
6. 找到内存块后，将内存首地址保存在pvReturn，用作函数返回值；
7. 此时内存块已经被申请了，要将其从空闲内存块链表中移除；
8. 如果申请到的内存块大于所需的大小，则需要把多余的内存重新组合成一个新的可用空闲内存块；
9. 将新的空闲内存块插入链表；
10. 更新全局状态变量xFreeBytesRemaining和xMinimumEverFreeBytesRemaining；
11. 已申请内存块xBlockSize最高位置1，表示已被使用；

释放内存，函数vPortFree处理步骤如下：

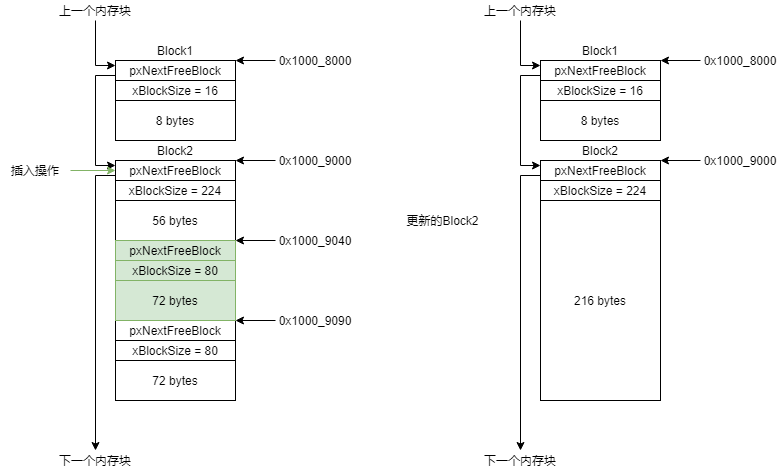
1. 内存指针移位，获取内存块描述符指针；
2. 内存块的xBlockSize最高位必须为1；
3. 内存块的xBlockSize最高位清零；
4. 将内存块插入空闲内存块链表中；

**内存块插入链表**，函数prvInsertBlockIntoFreeList处理步骤如下：

1. 遍历链表找出内存块的插入点；内存块是按照地址从低到高的顺序链接在一起；
2. 找到插入点后，判断是否可以和插入的内存块合并；如果可以则合并，如下图所示；插入的内存块起始地址是0x2000\_9040，正好与Block2内存的结束地址一样，因此将两个内存块合并，合并后Block2的xBlockSize更新为144bytes ；



1. 检查新内存块是否可以与下一个内存块Block3合并；如果可以则合并，如图所示；Block2/Block3合并成一个内存块，大小是224字节；



1. 如果不能和Block3合并，则将两块内存连接在一起；
2. pxIterator不等于pxBlockToInsert，表示内存插入过程中，没有进行过合并，此时只做链表处理。pxIterator的内存块在前，pxBlockToInsert的在后，把两个内存块链接起来；

# 功耗管理

**CPU WFI指令**，让CPU进入睡眠模式。配置中断可唤醒，配置系统时钟睡眠时间。可唤醒CPU睡眠的中断有：蓝牙业务中断、外设按键中断、睡眠时间到产生的时钟中断。

**Tickless模式**，IDLE任务被调度，表明系统可以进入低功耗模式，但要避免系统频繁进入退出低功耗模式，导致冗余消耗。避免的方法有两个：1. IDLE任务的时间片不能太短；2. 要关闭系统时钟，避免时钟中断频繁唤醒CPU，给系统时钟配置睡眠时间，到时后再产生时钟中断。

**IDLE任务处理**，计算时间片，如果大于配置的阈值，则配置系统时钟睡眠时间，指令CPU进入睡眠模式。

**外设功耗**，是低功耗的关键。在CPU进入睡眠前，配置外设进入低功耗模式；在唤醒后，配置外设退出低功耗模式。FreeRTOS使用宏configPRE\_SLEEP\_PROCESSING()配置外设进入低功耗，configPOST\_SLEEP\_PROCESSING()配置外设退出低功耗。这两个宏需要在Port层实现，调用外设驱动的低功耗控制接口。

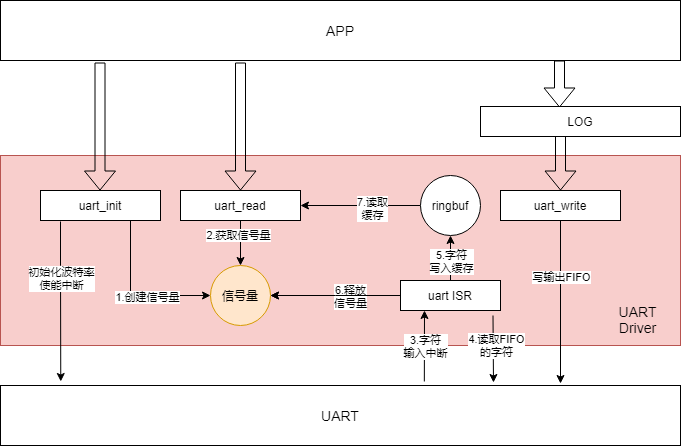
# 设备管理

FreeRTOS没有设备管理。此处描述的设备管理，是设备驱动程序基于FreeRTOS任务管理，实现的多任务并发。

在单任务系统中，APP持续占用CPU操作，无法实现多任务并发。在多任务并发系统中，多个APP任务，通过调用设备驱动的IO阻塞API接口，或者FreeRTOS的进程管理/队列管理/信号量/时间管理等阻塞API接口，实现主动释放CPU，触发任务调度操作。APP任务不感知被阻塞被调度。

以UART驱动为例，加入信号量处理。APP调用UART接口获取输入字符，处理流程如下图所示：

1. 调用UART初始化接口，创建信号量；
2. 调用UART读接口，获取信号量，如果此时没有输入字符，APP任务被信号量阻塞，释放CPU；如果已有输入字符，APP任务能获取到信号量，直接读取ringbuf字符；
3. UART有字符输入，产生中断，触发ISR处理；
4. ISR读取UART FIFO的字符；
5. ISR把字符写入软件缓存ringbuf；
6. ISR释放信号量；
7. APP任务被任务调度唤醒，继续执行UART读取程序，读取ringbuf字符；



各任务间并发运行关系，如下图所示：

T1时刻，系统调度初始化，IDLE任务运行；

T2时刻，APP任务运行；

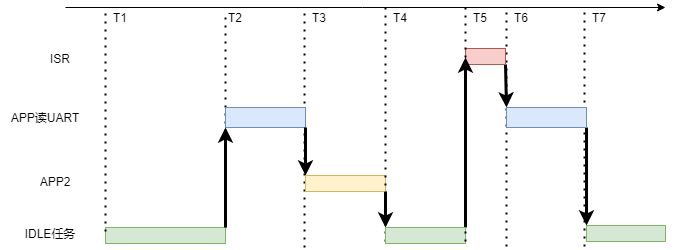
T3时刻，APP任务调用UART读接口获取字符；因没有硬件输入，被信号量阻塞，释放CPU；APP2任务得到调度；

T4时刻，APP2任务，因其它硬件资源被阻塞，释放CPU；因为未有就绪的任务，所以调度IDLE任务；

T5时刻，UART有输入，产生中断，调用UART ISR处理中断；

T6时刻，UART ISR获取了硬件输入字符并缓存，释放信号量，触发任务调度；APP任务获取到信号量，进入就绪状态，被调度运行；

T7时刻，APP完成处理输入字符，从新进入下一次读UART操作，被信号量再次阻塞，释放CPU；因为APP2阻塞条件未满足，状态未就绪，所以调度IDLE任务；



# API接口

参考FreeRTOS官网：www.freertos.org/features.html

# 配置项

|  |  |  |
| --- | --- | --- |
| 编译宏 | 默认值 | 描述 |
|  |  |  |
| 系统配置相关 |  |  |
| configUSE\_PREEMPTION | 0 | 启动抢占调度模式。取值： 1：内核在每个时钟节拍中断中进行任务调度 0：非抢占调度模式，有以下情况产生调度： a. 任务调用函数taskYIELD(); b. 任务调用了阻塞API函数； c. 应用程序定义了在中断执行上下文切换 |
| configUSE\_PORT\_OPTIMISED\_TASK\_SELECTION | 1 | 设置选择下一个要运行任务的方法： 0：软件通用方法； 1：硬件方法，使用MCU自带的硬件指令实现；如CLZ 计算前导零指令 |
| configUSE\_TICKLESS\_IDLE | 1 | 使能低功耗tickless模式 |
| configCPU\_CLOCK\_HZ | SystemCoreClock | 设置CPU频率 |
| configTICK\_RATE\_HZ | 1000 | 设置FreeRTOS系统时钟节拍，单位是Hz，此频率是滴答定时器的中断频率。1000表示1ms定时精度。 |
| configMAX\_PRIORITIES | 8 | 设置任务优先级数量，范围是0 ~（configMAX\_PRIORITIES - 1），0是最低优先级 |
| configMINIMAL\_STACK\_SIZE | 256 | IDLE任务的堆栈大小，以字为单位。实际是configMINIMAL\_STACK\_SIZE \* 4 bytes |
| configMAX\_TASK\_NAME\_LEN | 16 | 任务名最大长度 |
| configUSE\_16\_BIT\_TICKS | 0 | 设置系统节拍计数器变量的数据类型。系统节拍计数器的变量类型是TickType\_t 。当此宏为1，TickType\_t是16位；为0，是32位。 |
| configIDLE\_SHOULD\_YIELD | 0 | IDLE任务是否为同优先级的其他任务让出CPU；一般建议关闭 |
| configUSE\_TASK\_NOTIFICATIONS | 1 | 启动任务通知功能。相关API函数会被编译。 |
| configUSE\_MUTEXES | 1 | 使用互斥信号量。相关API会被编译。 |
| configUSE\_RECURSIVE\_MUTEXES | 0 | 启用递归互斥信号量。相关API会被编译。 |
| configUSE\_COUNTING\_SEMAPHORES | 1 | 启用计数型信号量，相关API会被编译 |
| configQUEUE\_REGISTRY\_SIZE | 10 | 设置可以注册的队列和信号量的最大数量。使用内核调试器查看队列和信号量，需要设置此宏。否则设置为0 。 |
| configUSE\_QUEUE\_SETS | 0 | 启用队列集功能 |
| configUSE\_TIME\_SLICING | 0 | 抢占式调度，在时钟节拍中断进行任务调度切换。当此宏设置为0，不会在时钟节拍中断中进行任务切换。 |
| configUSE\_NEWLIB\_REENTRANT | 0 | 支持newlib C库 |
| configENABLE\_BACKWARD\_COMPATIBILITY | 0 | 兼容老版本的数据类型 |
| configNUM\_THREAD\_LOCAL\_STORAGE\_POINTERS | 5 | 设置每个任务的本地存储指针数组大小。任务控制块中有本地储存数组指针，应用程序可以在其中存入数据。 |
|  |  |  |
| 内存分配相关 |  |  |
| configSUPPORT\_STATIC\_ALLOCATION | 0 | 为1，创建内核对象需要用户指定堆；为0，则内核动态分配 |
| configSUPPORT\_DYNAMIC\_ALLOCATION | 1 | 内核从堆动态获取内存 |
| configTOTAL\_HEAP\_SIZE | 60\*1024 | 设置堆大小。如果使用动态内存管理，则FreeRTOS在创建任务/信号量/队列等的时候，会用heap\_x.c的内存申请函数申请内存。这些内存是从堆ucHeap[configTOTAL\_HEAP\_SIZE]中申请。 |
| configAPPLICATION\_ALLOCATED\_HEAP | 0 | 堆内存是编译器分配。若打开此宏，堆内存ucHeap可以由用户设置。 |
|  |  |  |
| 回调函数相关 |  |  |
| configUSE\_IDLE\_HOOK | 1 | 使用IDLE任务回调函数。 函数原型 void vApplicationIdleHook( void ); |
| configUSE\_TICK\_HOOK | 0 | 启用时间片回调函数。 函数原型：void vApplicationTickHook(void) |
| configCHECK\_FOR\_STACK\_OVERFLOW | 1 | 设置堆栈溢出检查。每个任务都有堆栈，可以是FreeRTOS自动分配，也可以是用户指定。堆栈溢出检查有两种方法，都需要注册回调函数vApplicationStackOverflowHook。 宏定义为1：上下文切换时保存现场，现场保持在堆栈中。此时任务堆栈使用率达到最大值。此时不断检测任务堆栈指针是否指向有效空间。如果指向了无效空间，则调用回调函数。 宏定义大于1：创建任务时，向堆栈填充一个已知的标记值。一直检测堆栈的后几个bytes，标记值是否被改写。几乎能检测所有堆栈溢出 |
| configUSE\_MALLOC\_FAILED\_HOOK | 1 | 内存申请失败回调函数。 函数原型void vApplicationMallocFailedHook( void ); |
| configUSE\_DAEMON\_TASK\_STARTUP\_HOOK | 0 | 软件定时器第一次运行，执行一次回调函数。需要FreeRTOS运行时初始化的程序，可以放在这个回调函数执行。回调函数原型 void vApplicationDaemonTaskStartupHook( void ); |
|  |  |  |
| 任务状态相关 |  |  |
| configGENERATE\_RUN\_TIME\_STATS | 0 | 开启时间统计功能，需要定义： portCONFIGURE\_TIMER\_FOR\_RUN\_TIME\_STATS 初始化外设作为时间统计的基准时钟 |
| configUSE\_TRACE\_FACILITY | 0 | 支持执行可视化和跟踪 |
| configUSE\_STATS\_FORMATTING\_FUNCTIONS | 0 | configUSE\_TRACE\_FACILITY和configUSE\_STATS\_FORMATTING\_FUNCTIONS都是1，编译函数vTaskList和vTaskGetRunTimeStats ，可视化输出各任务状态 |
|  |  |  |
| Co-routine相关，可忽略 |  |  |
| configUSE\_CO\_ROUTINES | 0 | 启用协程 |
| configMAX\_CO\_ROUTINE\_PRIORITIES | 1 |  |
|  |  |  |
| 软时钟相关 |  |  |
| configUSE\_TIMERS | 1 | 使能软件定时器 |
| configTIMER\_TASK\_PRIORITY | 7 | 设置软件定时器任务的优先级 |
| configTIMER\_QUEUE\_LENGTH | 5 | 软件定时器API函数通过命令队列向软件定时器任务发送消息。此宏用于设置这个命令队列的长度。 |
| configTIMER\_TASK\_STACK\_DEPTH | configMINIMAL\_STACK\_SIZE | 设置软件定时器任务的堆栈大小 |
|  |  |  |
| 中断优先级相关 |  |  |
| configKERNEL\_INTERRUPT\_PRIORITY | 0 | 系统时钟中断优先级，芯来提供的代码设置为最低优先级 |
| configMAX\_SYSCALL\_INTERRUPT\_PRIORITY | 7 | 最大中断优先级 |
|  |  |  |
| 断言 |  |  |
| configASSERT |  | 设置断言宏定义 |
|  |  |  |
| 函数编译相关 |  |  |
| INCLUDE\_vTaskPrioritySet | 1 | 定义函数vTaskPrioritySet，设置任务优先级 |
| INCLUDE\_uxTaskPriorityGet | 1 | 定义函数uxTaskPriorityGet，获取任务优先级 |
| INCLUDE\_vTaskDelete | 1 | 定义函数vTaskDelete，删除任务 |
| INCLUDE\_vTaskSuspend | 1 | 定义函数vTaskSuspend和vTaskResume，挂起/恢复任务 |
| INCLUDE\_xResumeFromISR | 1 | 无相关条件编译 |
| INCLUDE\_vTaskDelayUntil | 1 | 定义函数vTaskDelayUntil，绝对时延 |
| INCLUDE\_vTaskDelay | 1 | 定义函数vTaskDelay，普通时延 |
| INCLUDE\_xTaskGetSchedulerState | 1 | 定义函数xTaskGetSchedulerState，获取调度器状态 |
| INCLUDE\_xTaskGetCurrentTaskHandle | 1 | 定义函数xTaskGetCurrentTaskHandle，获取当前任务句柄 |
| INCLUDE\_uxTaskGetStackHighWaterMark | 1 | 定义函数uxTaskGetStackHighWaterMark，获取任务执行以来堆栈最少剩余空间 |
| INCLUDE\_xTaskGetIdleTaskHandle | 1 | 定义函数xTaskGetIdleTaskHandle，获取IDLE任务句柄 |
| INCLUDE\_eTaskGetState | 0 | 定义函数eTaskGetState，获取任务状态 |
| INCLUDE\_xEventGroupSetBitFromISR | 1 | 无相关条件编译 |
| INCLUDE\_xTimerPendFunctionCall | 1 | 定义函数xTimerPendFunctionCall和xTimerPendFunctionCallFromISR，可以将函数的执行挂起到RTOS守护任务 |
| INCLUDE\_xTaskAbortDelay | 0 | 定义函数xTaskAbortDelay，强制任务离开阻塞状态，进入就绪状态 |
| INCLUDE\_xTaskGetHandle | 1 | 定义函数xTaskGetHandle，用任务名获取任务句柄 |
| INCLUDE\_xTaskResumeFromISR | 1 | 定义函数xTaskResumeFromISR，在中断上下文恢复任务 |