实验目的

1．学习FreeRTOS操作系统的内核原理；

2．掌握stm32F1系列单片机的使用；

3．学习嵌入式实时操作系统内核的移植；

4．掌握FreeRTOS下基本多任务应用程序的编写，加深对嵌入式实时操作系统的理解。

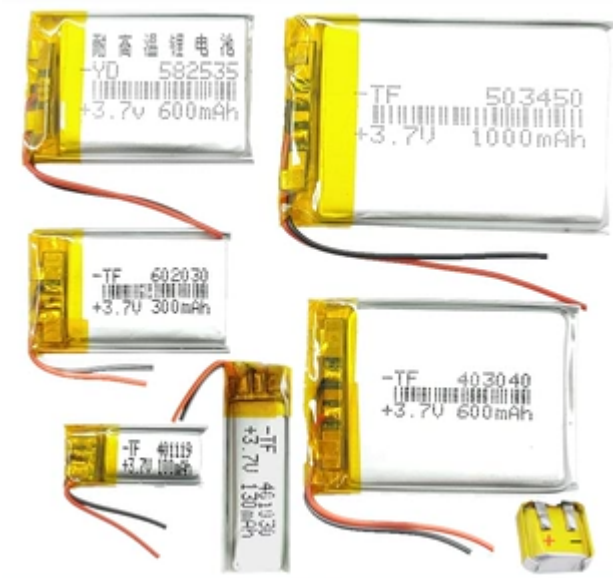
项目介绍

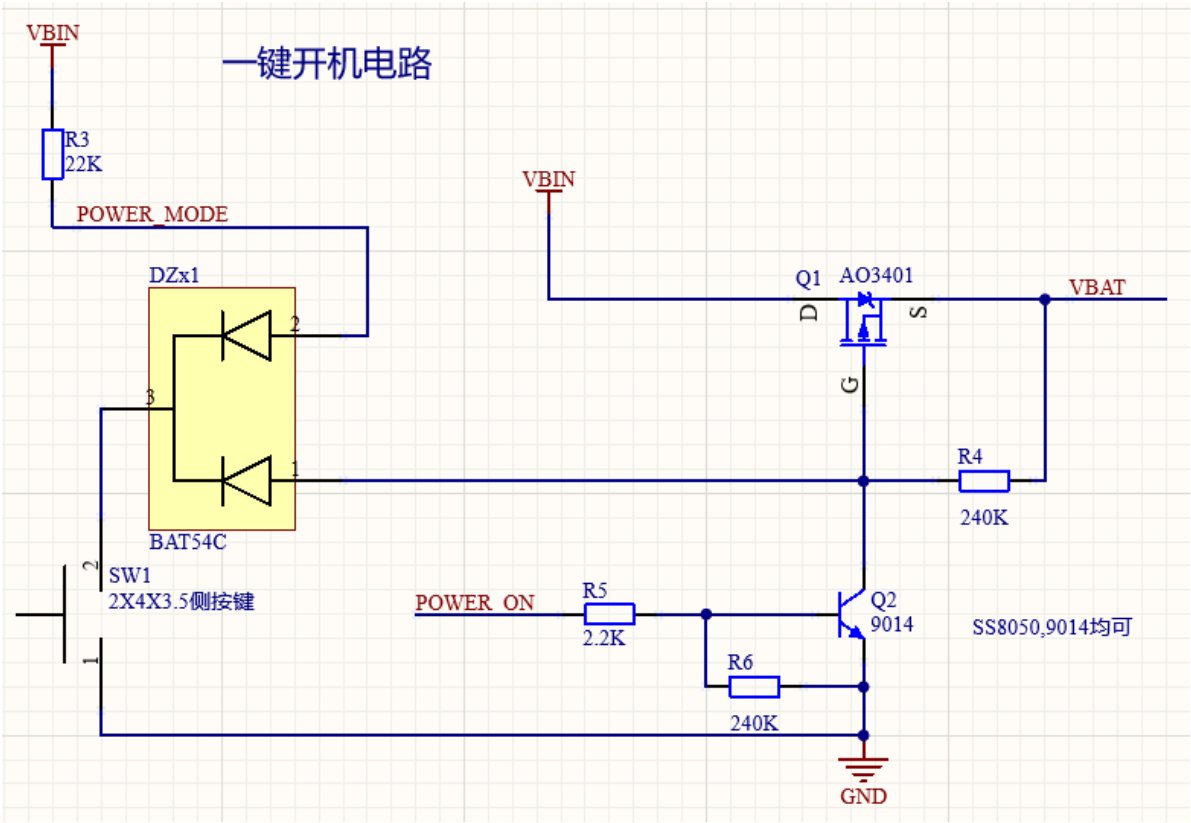
做一个基于stm32的智能手表，实现读取RTC芯片时间、改变时间、多级菜单目录等功能。MCU芯片采用STM32F103C8T6，价格便宜，扩展外设齐全，满足项目需求。OLED选取1.3寸I2C协议的OLED，RTC时钟芯片采用DS3231，陀螺仪芯片采用MPU6050，均为I2C通信协议，总线式通信，易于设计与布线。

硬件设计

电源部分

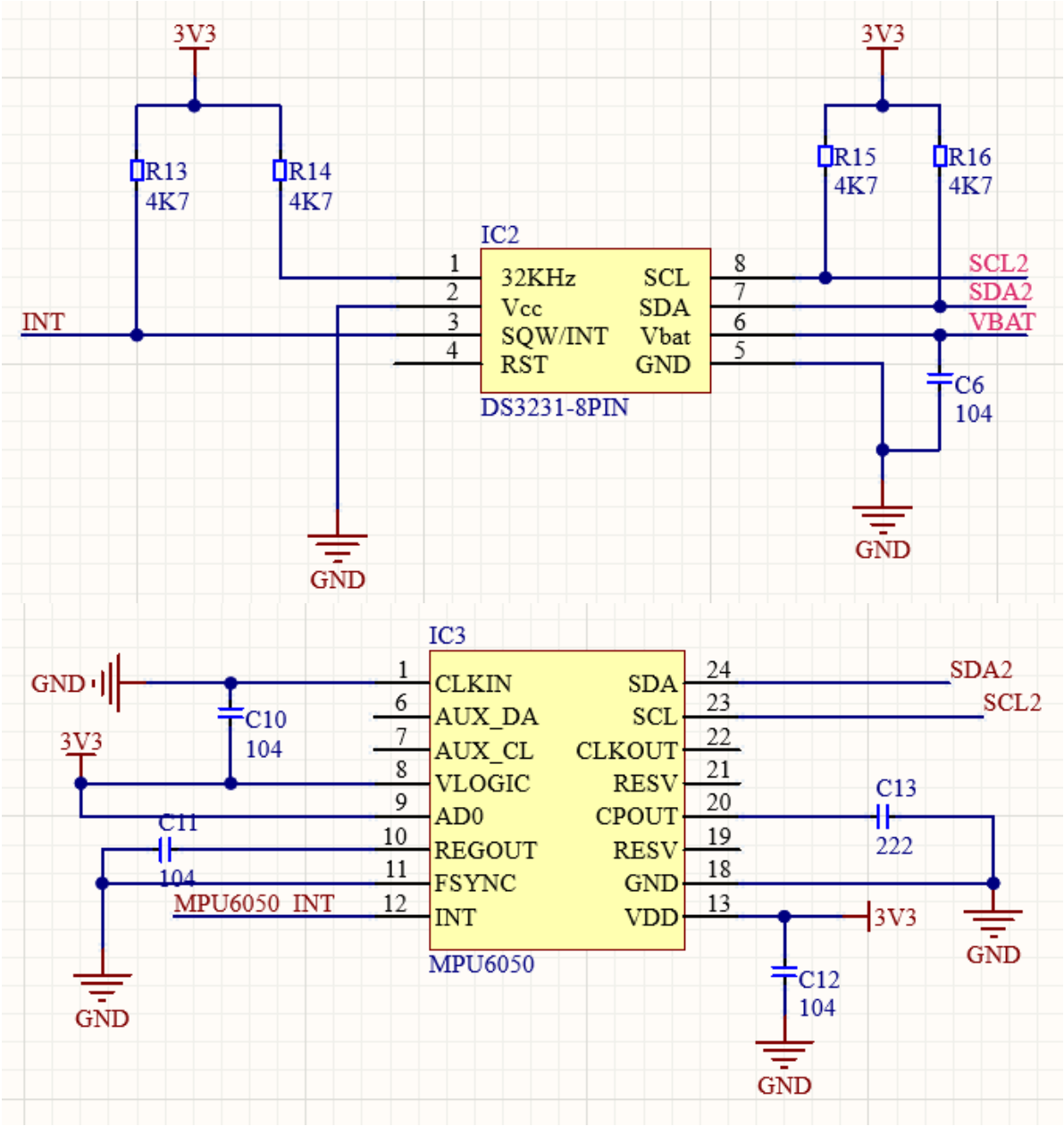
采用低压差LDO电源芯片利用type-c为电池供电，电池采用3.7V锂电池进行储存电能，为单片机供电。除此之外，还设计了一个一键开机电路，电池连接VBIN脚，VBAT为为单片机供电的脚，当SW1按下时，莫斯管导通，进而实现系统上电运转。





功能部分

OLED选取1.3寸I2C协议的OLED，RTC时钟芯片采用DS3231，陀螺仪芯片采用MPU6050，因为它们均是I2C总线形式通信，所以节省引脚。因为芯片自带的I2C驱动有不稳定的问题，所以软件方面采用模拟I2C。



软件部分

FREERTOS介绍

RTOS的好处

RTOS，real-time Operate System。有很多成熟的技术可以在不使用内核的情况下编写好的嵌入式软件，但是在复杂情况下，RTOS有如下的好处：

1）用户无需关心时间信息

内核负责计时，并由相关的API完成，从而使得用户的应用程序代码结构更简单。

2）模块化、可拓展性强

也正是由于第一点的原因，程序性能不易受底层硬件更改的影响。并且，各个任务是独立的模块，每个模块都有明确的目的，降低了代码的耦合性。

3）效率高

内核可以让软件完全由事件驱动，因次，轮询未发生的事件是不浪费时间的。相当于用中断来进行任务切换。

4）中断进程更短

通过把中断的处理推迟到用户创建的任务中，可以使得中断处理程序非常短。

FREERTOS

FreeRTOS是一个迷你的实时操作系统内核。作为一个轻量级的操作系统，功能包括：任务管理、时间管理、信号量、消息队列、内存管理、记录功能、软件定时器、协程等，可基本满足较小系统的需要。由于RTOS需占用一定的系统资源(尤其是RAM资源)，只有μC/OS-II、embOS、salvo、FreeRTOS等少数实时操作系统能在小RAM单片机上运行。相对μC/OS-II、embOS等商业操作系统，FreeRTOS操作系统是**完全免费的操作系统**，具有**源码公开、可移植、可裁减、调度策略灵活**的特点，可以方便地移植到各种单片机上运行。

FreeRTOS的特性

- 具有抢占式或者合作式的实时操作系统内核
- 功能可裁剪，最小占用10kB左右rom空间，0.5kB ram空间
- 灵活的任务优先级分配
- 具有低功耗模式
- 有互斥锁、信号量、消息队列等功能
- 运行过程可追踪
- 支持中断嵌套

FREERTOS功能

- 任务管理
- 时间管理
- 内存管理
- 通信管理

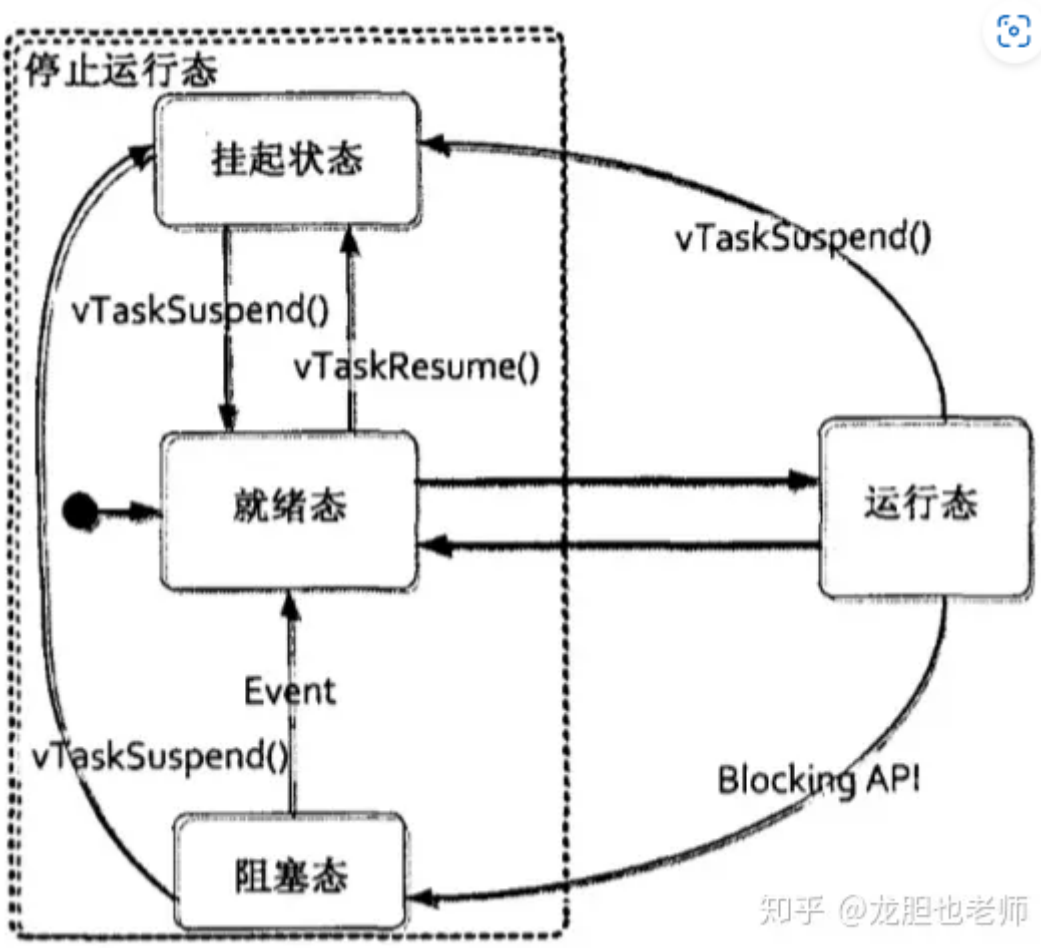
任务管理

有抢占式和合作式两种任务调度方式，其中抢占式用的比较多。

任务状态有：挂起状态、就绪状态、阻塞状态、运行状态

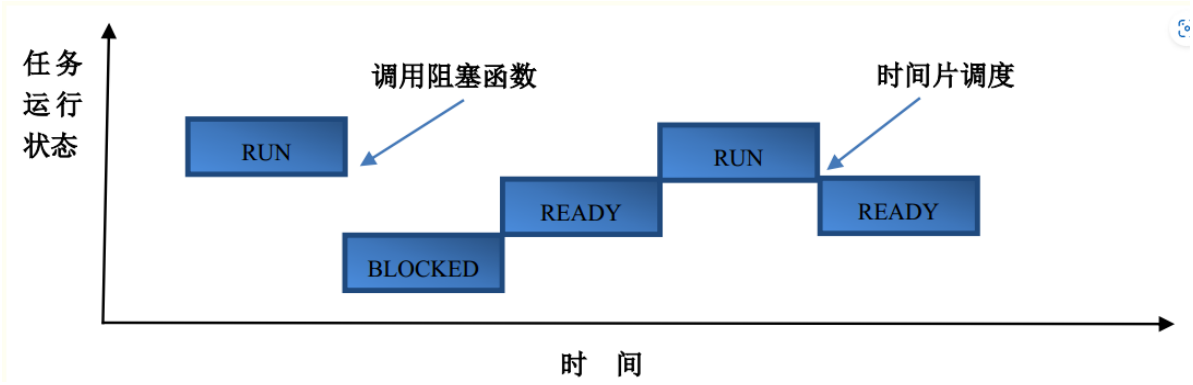
- 挂起状态：任务被挂起，不在执行，直到被唤醒，才可以进入就绪状态
- 就绪状态：任务等待执行，目前CPU等资源被分给的更高优先级的任务
- 阻塞状态：在任务运行到延时函数或者等待某个信号量时，任务这种状态就称为阻塞状态
- 运行状态：任务正在运行，占用CPU等资源

高优先级的任务会打断低优先级的任务执行，抢占资源，使得低优先级退回就绪状态，等高优先级任务退回阻塞状态时才会执行低优先的任务。



时间管理

在任务执行到延时函数或者等待信号量时，OS会让其进入到阻塞状态，空出资源执行其他任务。



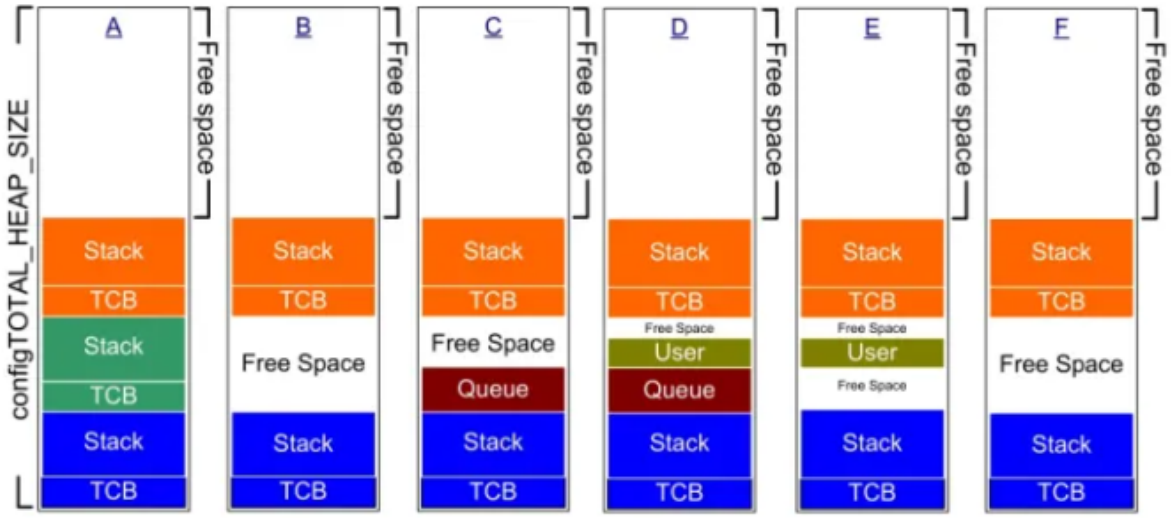
OS的时钟节拍在stm32上是由滴答时钟实现的，每次会产生中断实现OS的时钟节拍。

内存管理

表 1　FreeRTOS 四种内存分配方案比较

C 文件	优　点	缺　点
heap1.c	分配简单、时间确定	只分配、不回收
heap2.c	动态分配、最佳匹配	碎片、时间不定
heap3.c	调用标准库函数	速度慢、时间不定
heap4.c	相邻空闲内存可合并	碎片、合并效率低

下图为heap4内存分配方式



其中，heap4：我们主要用到的是动态内存分配的heap4。相比于heap2的链表式内存块结构，heap4是按照物理地址来进行排序。这样设计的目的是方便合并**相邻物理地址**中空闲的内存块。但是，当在嵌入式系统中，频繁的创建与释放内存，还是会导致空闲块物理地址相对分散，依然会产生较多内存碎片。

通信管理

OS中，为了防止任务对公共资源的访问冲突，即同一时间访问同一公共资源，设计了信号量，消息队列，邮箱等通信方式。

信号量是操作系统中重要的一部分，信号量一般用来进行资源管理和任务同步，FreeRTOS中信号量又分为二值信号量、计数型信号量、互斥信号量和递归互斥信号量。不同的信号量其应用场景不同，但有些应用场景是可以互换着使用的。

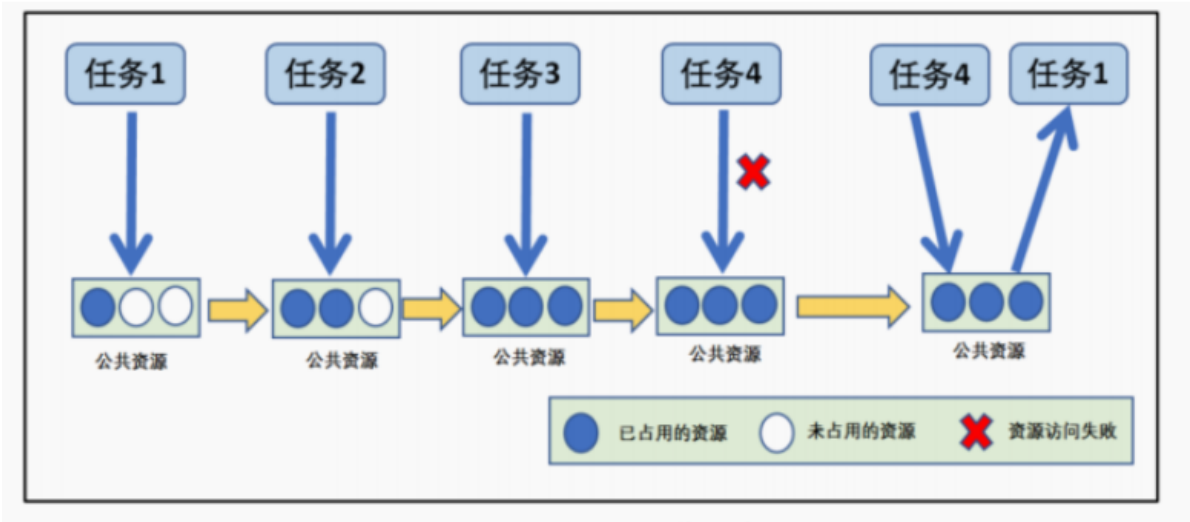
二值信号量

二值信号量其实就是一个只有一个队列项的队列，这个特殊的队列要么是满的，要么是空的。二值信号量通常用于互斥访问或任务同步，二值信号量和互斥信号量非常类似，但是还是有一些细微的差别，互斥信号量拥有优先级继承机制，二值信号量没有优先级继承。因此二值信号另更适合用于同步(任务与任务或任务与中断的同步)，而互斥信号量适合用于简单的互斥访问。

计数信号量

计数型信号量用于事件计数和资源管理。

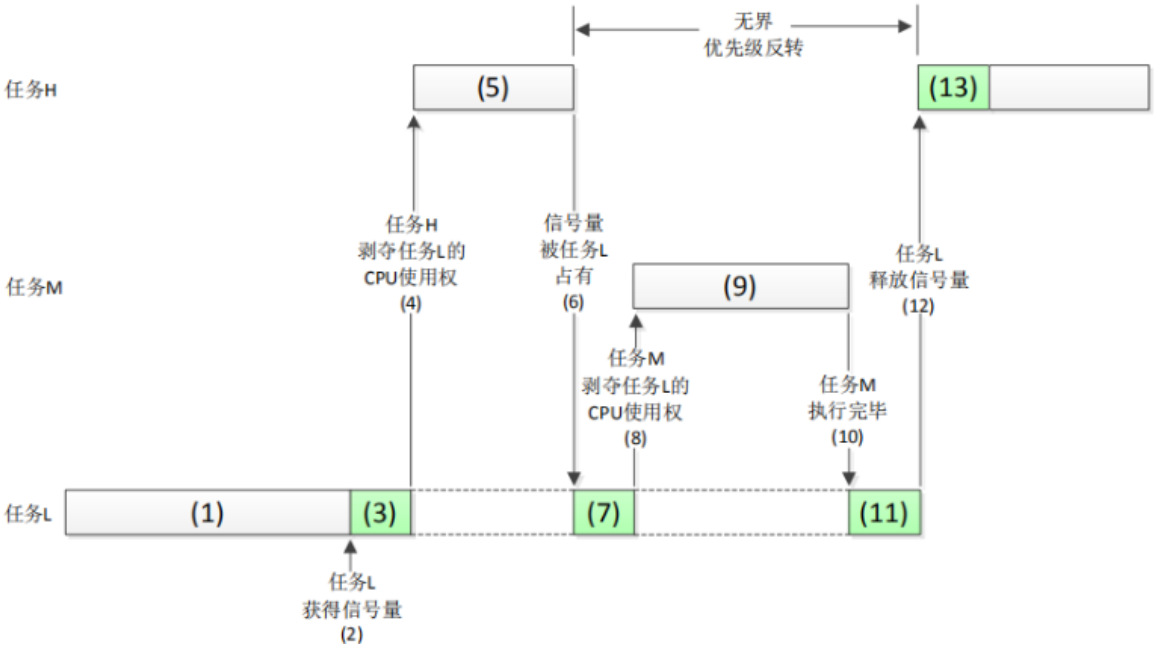
计数信号量可以用于资源管理，允许多个任务获取信号量访问共享资源，但会限制任务的最大数目。



互斥信号量

互斥信号量的应用场景---------->>优先级翻转

高优先级任务被低优先级任务阻塞，导致高优先级任务迟迟得不到调度。但其他中等优先级的任务却能抢到CPU资源。-- 从现象上来看，好像是中优先级的任务比高优先级任务具有更高的优先权。



优先级继承就是为了解决优先级反转问题而提出的一种优化机制。其大致原理是让低优先级任务在获得互斥信号量的时候(如果有高优先级的线程也需要使用该互斥信号量时)，临时提升其优先级。以前其能更快的执行并释放同步资源。释放同步资源后再恢复其原来的优先级。

互斥信号量为特殊的二值信号量，由于其特有的优先级继承机制，从而使它更适用于简单互锁，即保护临界资源。

递归信号量

递归，即可以重复获取调用。

即对于已经获取递归互斥量的任务可以重复获取该递归互斥量，该任务拥有递归信号量的所有权。

但任务成功获取多少次递归互斥量，就要返还几次，在此前递归互斥量都处于无效状态，其他任务无法获取，只有持有递归信号量的任务才能获取和释放。

OS配置

选择抢占式调度器

禁能tickless低功耗模式

OS节拍选择1000Hz，选择TIM1滴答时钟实现

最大优先级数为5

因为芯片选用的STM32F103，RAM只有20K，OS堆大小选择7K

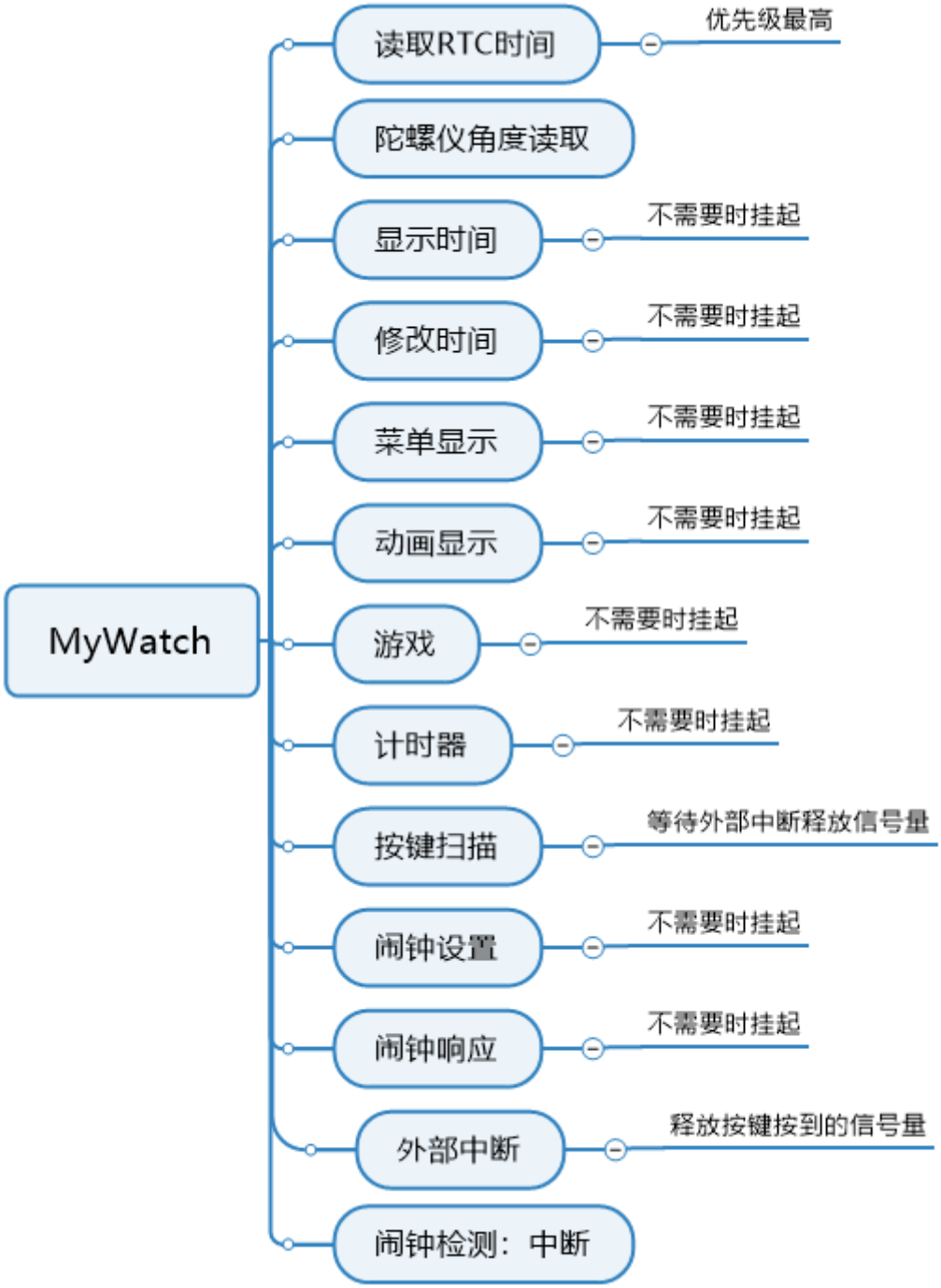
使能任务间直接的消息传递，包含信号量，事件标志组和消息邮箱

OS配置如下图所示：

∨ API	
FreeRTOS API	CMSIS v1
∨ Versions	
FreeRTOS version	10.0.1
CMSIS-RTOS version	1.02
∨ Kernel settings	
USE_PREEMPTION	Enabled
CPU_CLOCK_HZ	SystemCoreClock
TICK_RATE_HZ	1000
MAX_PRIORITIES	5
MINIMAL_STACK_SIZE	64 Words
MAX_TASK_NAME_LEN	16
USE_16_BIT_TICKS	Disabled
IDLE_SHOULD_YIELD	Disabled
USE_MUTEXES	Enabled
USE_RECURSIVE_MUTEXES	Disabled
USE_COUNTING_SEMAPHORES	Disabled
QUEUE_REGISTRY_SIZE	8
USE_APPLICATION_TASK_TAG	Enabled
ENABLE_BACKWARD_COMPATIBILITY	Enabled
USE_PORT_OPTIMISED_TASK_SELECTION	Enabled
USE_TICKLESS_IDLE	Disabled
USE_TASK_NOTIFICATIONS	Enabled
RECORD_STACK_HIGH_ADDRESS	Disabled
∨ Memory management settings	
Memory Allocation	Dynamic / Static
TOTAL_HEAP_SIZE	7168 Bytes
Memory Management scheme	heap_4

任务创建

创建多个任务，每个功能创建为一个任务，任务不需要时就挂起，需要运行时再运行，节省CPU等资源



任务的配置如下图：

Tasks

Task Name	Priority	Stack Size (Words)	Entry Function	Code Generation O...	Parameter	Allocation
defaultTask	osPriorityNormal	128	StartDefaultTask	Default	NULL	Dynamic
RTC_Task	osPriorityAboveNor...	64	Read_Time	As external	NULL	Dynamic
Root_Tas	osPriorityBelowNor...	128	Display_Root	As external	NULL	Dynamic
Change_Task	osPriorityBelowNor...	128	Display_ChangeTime	As external	NULL	Dynamic
Menu_Task	osPriorityBelowNor...	128	Display_Menu	As external	NULL	Dynamic
Cartoon_Task	osPriorityBelowNor...	128	Display_Cartoon	As external	NULL	Dynamic
Gyro_Task	osPriorityNormal	128	Read_Gyro	As external	NULL	Dynamic
AlarmSet_Task_	osPriorityBelowNor...	128	Alarm_Set	As external	NULL	Dynamic
AlarmGo_Task	osPriorityBelowNor...	64	Alarm_Go	As external	NULL	Dynamic
Timing_Task	osPriorityBelowNor...	128	Timing	As external	NULL	Dynamic
Game_Task	osPriorityBelowNor...	128	Game	As external	NULL	Dynamic
KEY_Task	osPriorityAboveNor...	64	Key_Scan	As external	NULL	Dynamic

采用信号量进行通信，其中互斥信号量进行任务间的通信，保证不同任务在访问共享资源时不会冲突；二值信号量为中断与任务的通信，任务请求信号量时进入阻塞状态，节省资源，中断释放信号量。

二值信号量（用于外部中断和按键扫描之间的通信）：

Binary Semaphores

Semaphore Name	Allocation
Key_State	Dynamic

互斥信号量：

Mutexes

Mutex Name	Allocation
Time_Signal	Dynamic
Gyro_Signal	Dynamic
SW_Signal	Dynamic

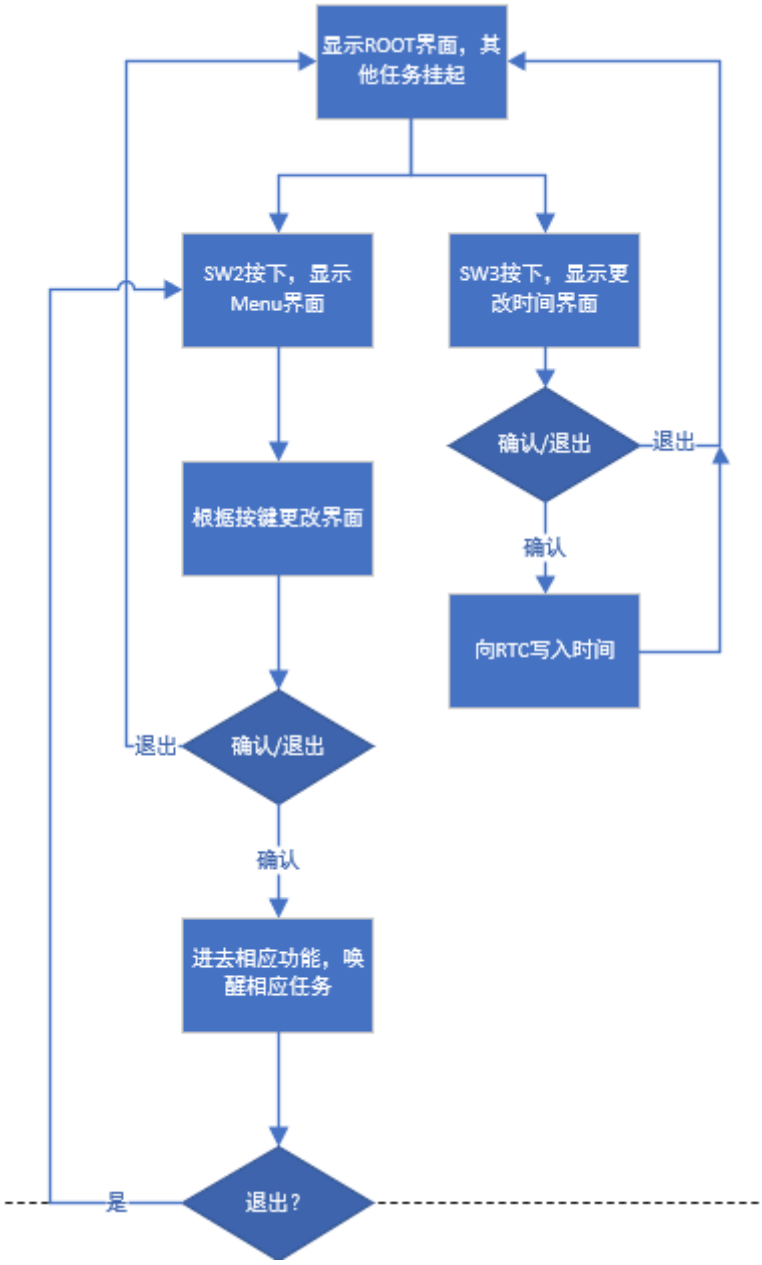
Time_Signal负责保护当前时间变量，主要有读取RTC时间、显示时间这两个任务需要访问。

Gyro_Signal负责保护陀螺仪的角度数据，主要有陀螺仪角度读取、游戏这两个任务需要访问。

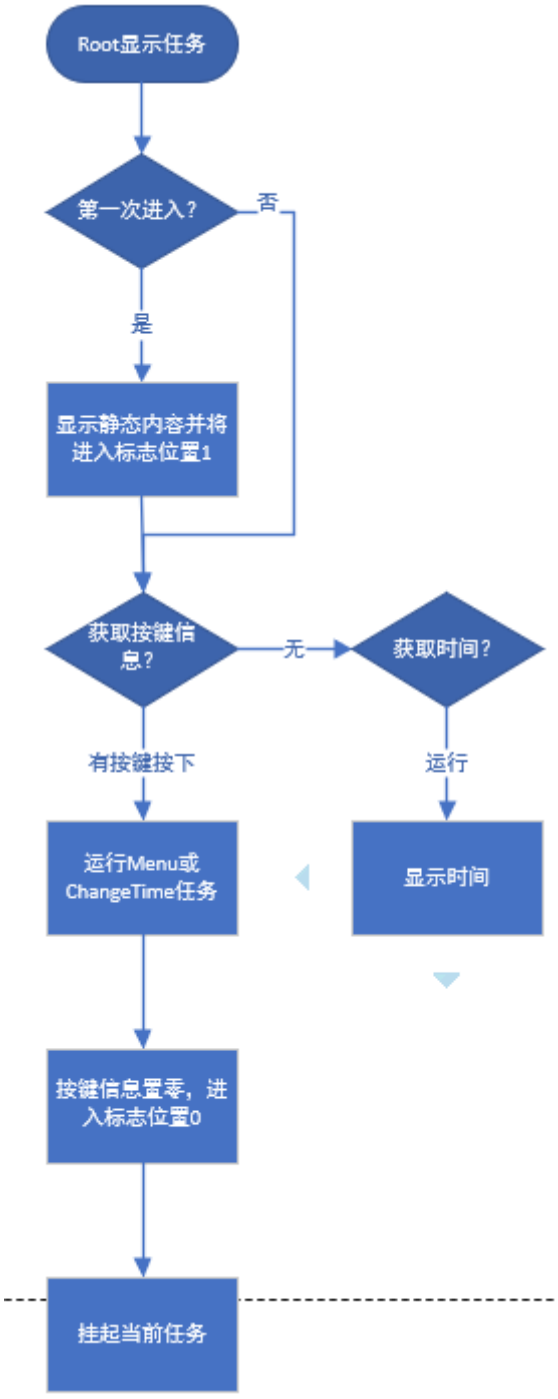
SW_Signal负责保护按键信息，其中显示时间、修改时间、菜单显示、动画显示、游戏、计时器、按键扫描、闹钟设置均需要访问。

各个任务的流程图

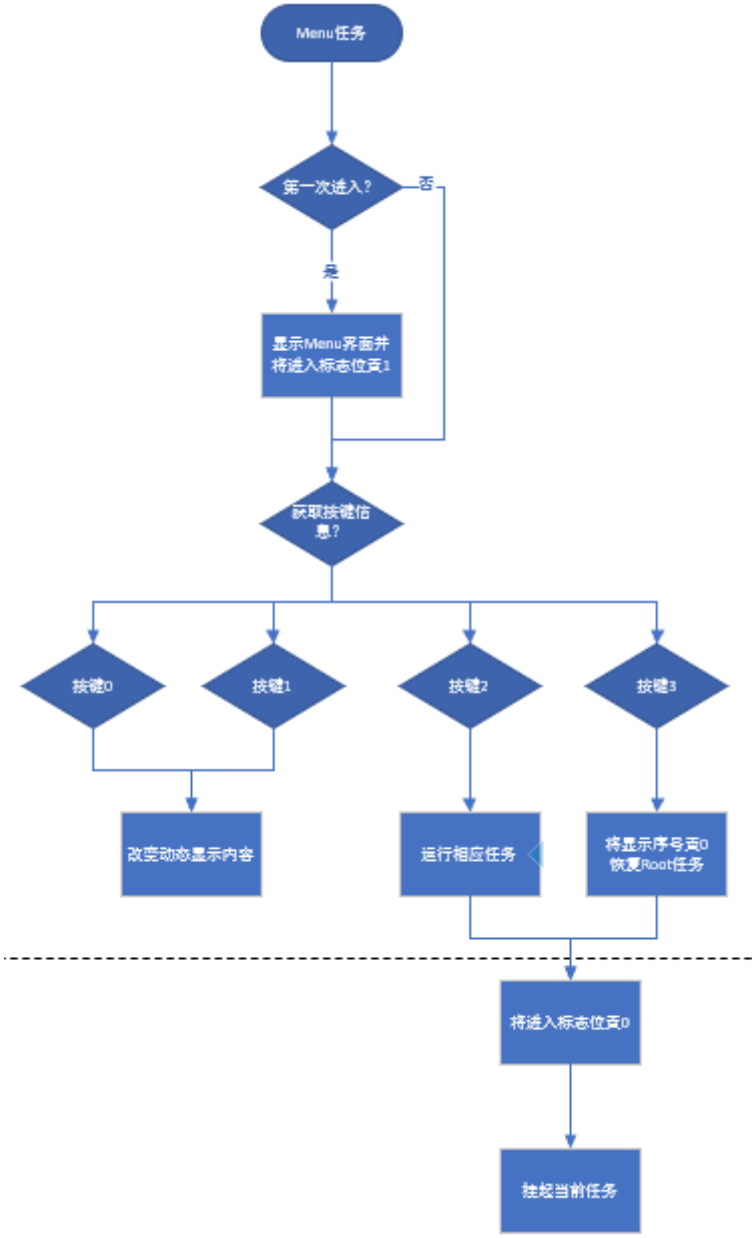
整体任务调度流程：

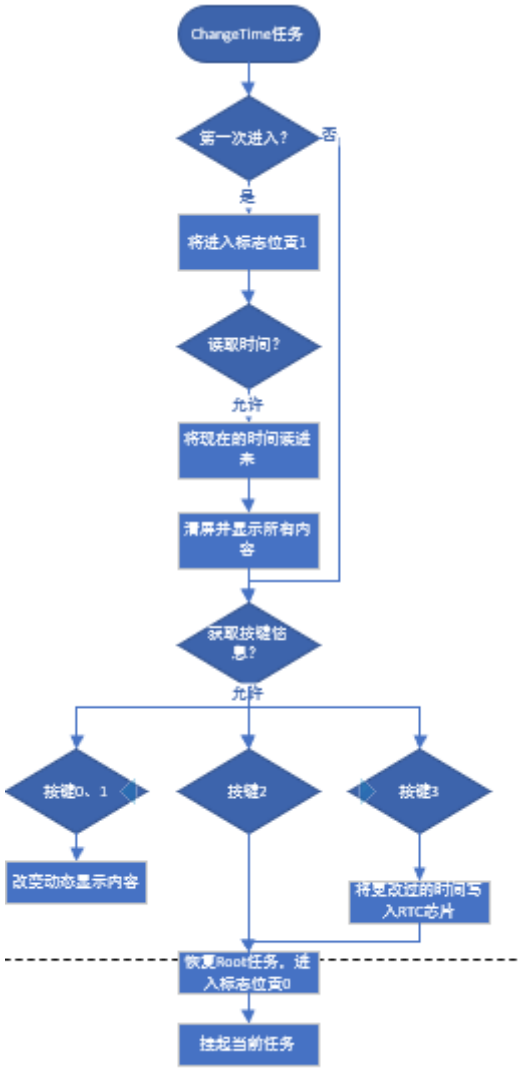


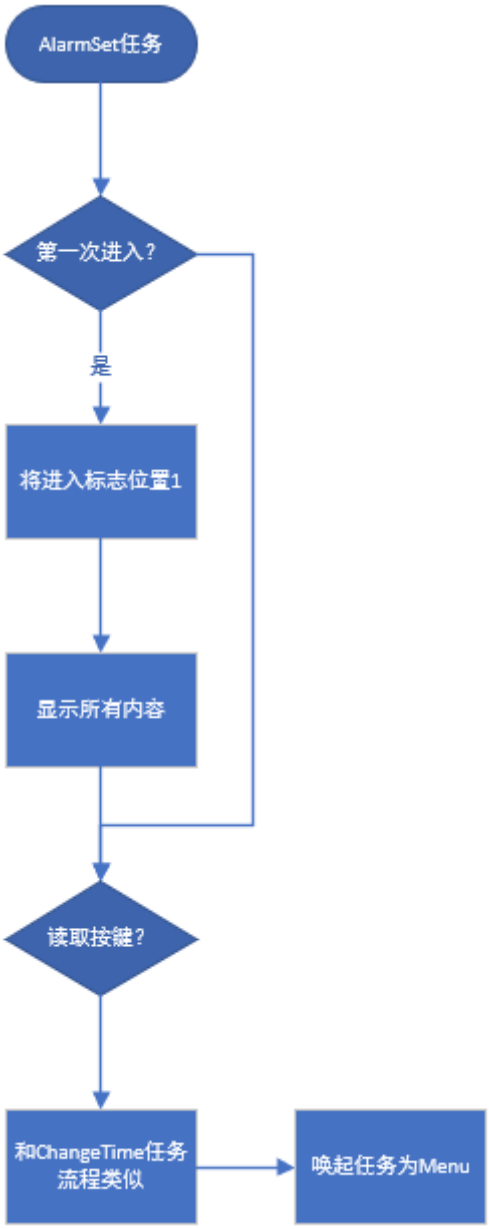
每个任务的流程图：


Root显示任务
第一次进入?
否
是
显示静态内容并将进入标志位置1
获取按键信息?
无
获取时间?
有按键按下
运行
运行Menu或ChangeTime任务
显示时间
按键信息置零，进入标志位置0
挂起当前任务

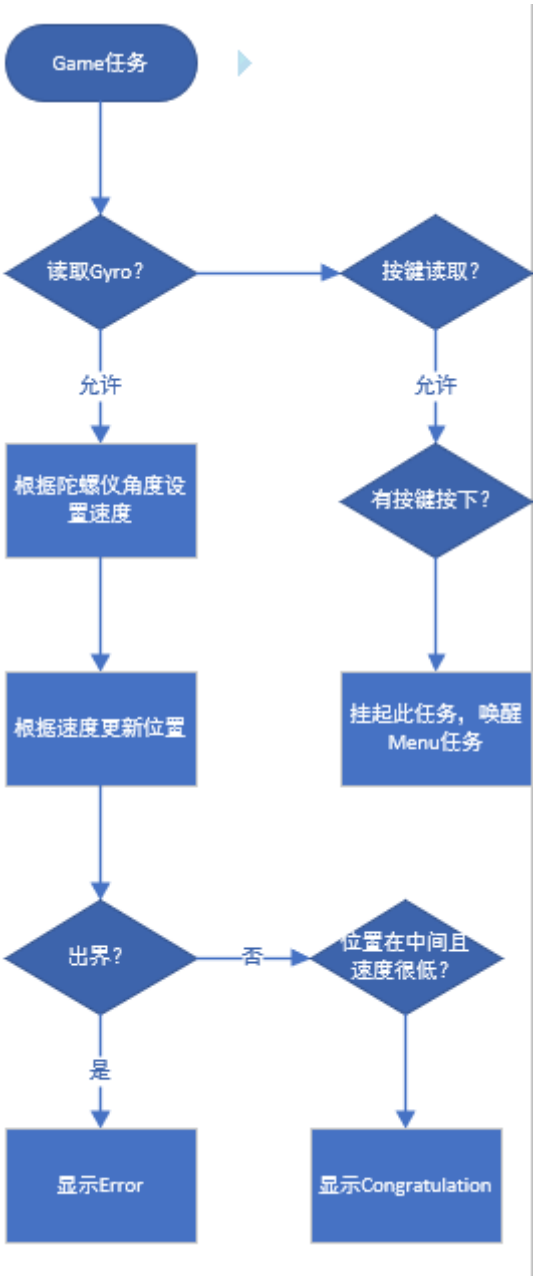

陀螺仪读取任务
确认在ROOT界面
长时间无操作?
是
挂起显示任务，不更新按键
屏幕熄灭且Gy过大?
是
唤醒ROOT显示界面

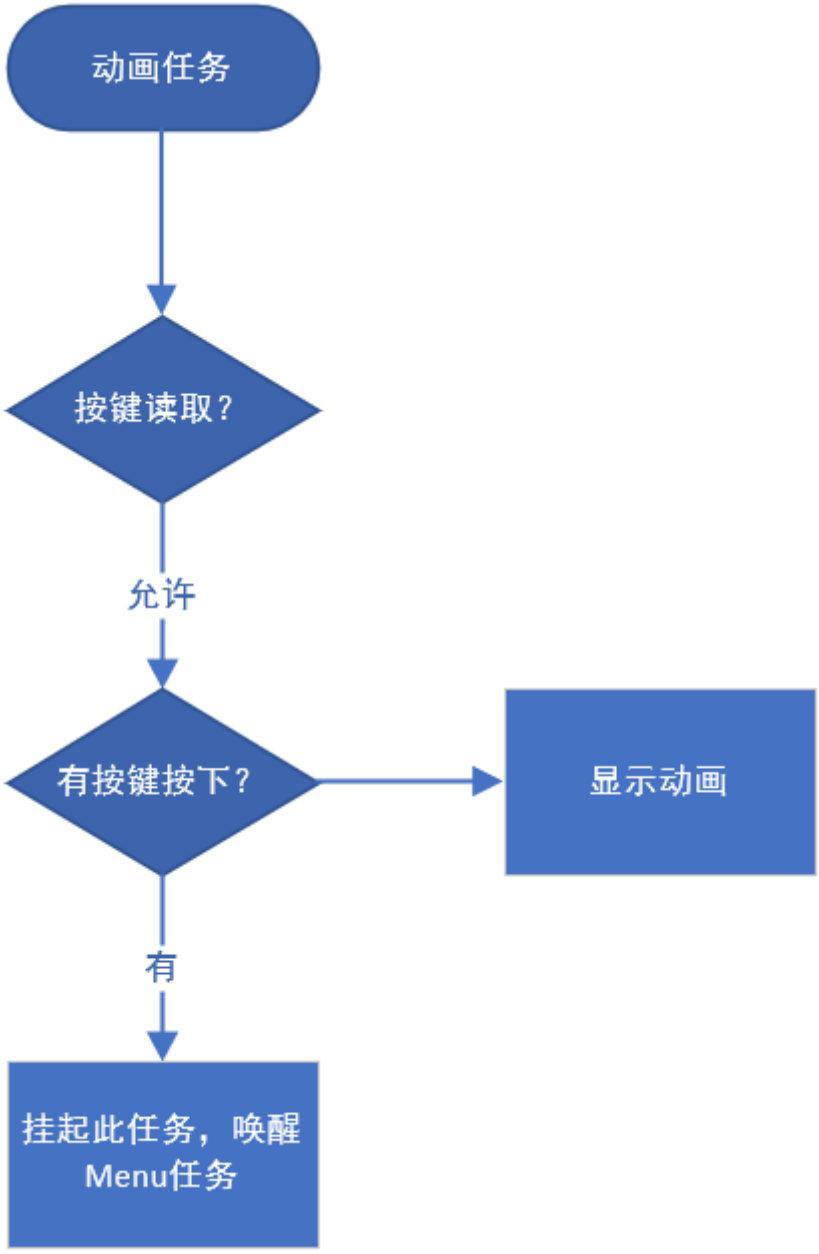

Menu任务
第一次进入?
否
是
显示Menu界面并
将进入标志位置1
获取按键信
息?
按键0
按键1
按键2
按键3
改变动态显示内容
运行相应任务
将显示序号页0
恢复Root任务
将进入标志位置0
挂起当前任务


ChangeTime任务
第一次进入?
否
是
将进入标志位置1
读取时间?
允许
将现在的时间读进来
清屏并显示所有内容
获取按键信息?
允许
按键0、1
按键2
按键3
改变动态显示内容
将更改过的时间写入RTC芯片
恢复Root任务，进入标志位置0
挂起当前任务

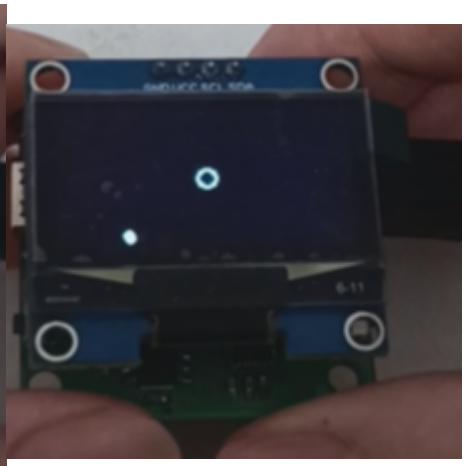
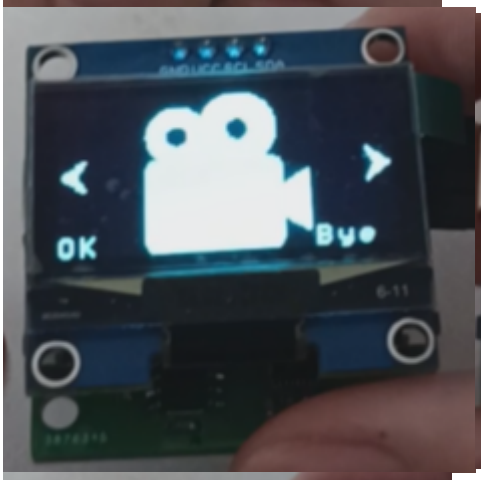

AlarmSet任务
第一次进入？
是
将进入标志位置1
显示所有内容
读取按键？
和ChangeTime任务流程类似
唤起任务为Menu


AlarmGo任务
中断检测到Alarm时间后，唤醒此任务
LED闪烁30s，之后挂起任务


Game任务
读取Gyro？
按键读取？
允许
允许
根据陀螺仪角度设置速度
有按键按下？
根据速度更新位置
挂起此任务，唤醒Menu任务
出界？
否
位置在中间且速度很低？
是
显示Error
显示Congratulation



最终情况预览

可改进空间

1. MAX30102心跳传感器没焊好，引脚在芯片下面，用热风枪吹坏了。
2. 电流不太够，3.7V转3.3V的LDO电流较小，无法驱动起OLED，需要改进。
3. 空间可以再优化一下，然后将SW的烧录排针与充电用的type-c复用一下。

4. 按键检测上可以加个电容消抖，感觉软件延时消抖的作用有限。

总结（个人感受）

这个项目中，之前我是在裸机上采用的前后台系统；在了解了RTOS之后，转而应用了freertos。

在裸机上的感觉：因为多级目录的原因，我需要思考每个按键按下后如何，OLED显示哪些东西，有什么改变，实现什么功能等等；感觉非常复杂。

之后我应用了RTOS，我觉得多级目录变得简单了很多很多，因为我可以专注于当前的显示任务，不需要过多考虑切换的逻辑和按键标志位的有效性更新；当要改变OLED的显示时，直接挂起当前任务然后唤醒目标任务即可。在任务中，我只要专心于当前的和下一级目录即可，检测到什么按键按下时，实现相应功能即可。

同时操作系统移植也是非常简单的，只要改好接口函数或者接口协议即可移植过去，再底层的模块驱动函数不需要更改，系统兼容性和可移植性都变高了，这一特性尤其适合于团队合作一起做一个很大的项目时。

附录

资料汇总