

μ C/OS 在 CH563 上的应用说明文档

版本: V1.0

<http://wch.cn>

1、 概述

μ C/OS 是一种可移植的, 可植入 ROM 的, 可裁剪的, 抢占式的, 实时多任务操作系统内核。它被广泛应用于微处理器、微控制器和数字信号处理器。

当前移植到 CH563 上的 μ C/OS 版本为 μ C/OS-II V2.51, 根据 CH563 的系统架构对代码进行了适当修改, 以满足 CH563 平台的使用, 可以实现任务的创建、任务的调度、IRQ 中断、FIQ 中断等功能。

本文档主要说明 μ C/OS 移植到 CH563 平台后的使用方法以及在使用过程中需要注意的事项, 并结合几个例程对工程进行简单的分析, 以帮助读者快速掌握 μ C/OS 在 CH563 平台上的使用。

本资料以及软件例程仅供学习交流使用, 您可以将其用于教学和私下研究; 但是如果你将其用于商业用途, 那么你必须通过 Micrium 获得商业许可。

2、 使用方法

以例程中的 Example1 为例来具体说明使用方法。

- A. 首先确定整个软件系统需要划分成多少个用户任务。
- B. 将每个用户任务中需要重复执行的代码按照 Example1 例程中的 Task1 函数的格式进行编写, 并为每个任务中使用到的硬件资源编写相关的驱动函数。
- C. 根据任务中所定义的变量多少来确定开辟给每个任务对应栈的大小, 并将 TASK_STK_SIZE 宏定义修改为该值; 并为每个任务定义一个数组作为该任务的栈, 数组大小为 TASK_STK_SIZE。
- D. 为系统创建一个起始任务 TaskStart, 首先在该函数中调用每个任务中使用到的硬件的驱动函数, 接着调用 BSP_init 函数以初始化系统时钟; 然后在该函数中通过调用系统函数 OSTaskCreate 来创建用户任务, 用户任务的优先级暂时不要设置为 0, 这个优先级是优先给起始任务使用的; 最后调用 OSTaskDel 将起始任务进行删除。
- E. 在 main 函数中调用 OSInit 实现对系统环境的初始化; 接着调用 OSTaskCreate 来创建起始任务, 该任务的优先级设置为 0; 最后调用 OSStart 开始操作系统的调度。

Note: 如果程序涉及到 FIQ、IRQ 中断, 可以参考例程 Example2、Example3; 其他 μ C/OS 的功能, 如信号量、互斥信号量、邮箱、队列等请自行根据 μ C/OS 教程进行学习。

3、 使用注意事项

以例程中的 Example1 工程为例, 对工程中需要注意的问题进行说明。

在移植系统的工程里, BSP.C 文件中包含的是有关硬件初始化的代码, 其中最重要的就是对 TIM0 定时器的初始化, 该初始化代码如非必要不要修改, TIM0 的中断采用了 IRQ 的向量中断, 请勿将其修改成 IRQ 非向量中断。在实际项目中如果需要用定时器, 请尽量使用除 TIM0 以外的其他定时器。

在无操作系统的 CH563 工程代码中, FIQ 以及 IRQ 的非向量中断函数分别为 __irq void FIQ_Handler(void) 和 __irq void IRQ_Handler(void); 添加操作系统之后, 每次进入

中断和退出中断都需要进行相关的任务切换等操作，因此将中断函数通过汇编宏定义进行间接调用，该汇编代码段放置在文件 `Os_int_a.s` 中；修改之后的 FIQ 以及 IRQ 的非向量中断函数分别为 `void FIQ_C_Handler (void)` 和 `void IRQ_C_Handler (void)`，这两个函数前面一定不要使用 `__irq` 进行修饰，如果添加 `__irq` 将会导致中断函数异常重复执行。

如果使用 CH563 的其他 IRQ 向量中断，如 SPI0 或 PB 中断，首先根据 CH563 向量中断配置的方法将 SPI0 和 PB 的 IRQ 中断配置成向量中断的方式，接着在 `Os_int_a.s` 文件中按照 TIMO 向量中断的格式分别定义 SPI0 和 PB 对应的中断，并在 `MAIN.C` 文件中实现对应的中断处理 C 语言函数。如下表：

Os_int_a.s 中汇编宏定义代码	MAIN.C 文件中对应的函数名	状态
IRQASMTimer0 HANDLER IRQ_C_Timer0	void IRQ_C_Timer0(void)	已添加
IRQ_Handler HANDLER IRQ_C_Handler	void IRQ_C_Handler(void)	已添加
FIQ_Handler HANDLER FIQ_C_Handler	void FIQ_C_Handler(void)	已添加
IRQASMSPI0 HANDLER IRQ_C_SPI0	void IRQ_C_SPI0(void)	未添加
IRQASMPB HANDLER IRQ_C_PB	void IRQ_C_PB(void)	未添加

μ C/OS 属于可裁剪操作系统，可以根据自身工程的需要对操作系统进行裁剪，具体方式为修改 `os_cfg.h` 文件中对应功能的使能宏定义，也可以修改部分参数的值，比如最大任务数量 `OS_MAX_EVENTS`、最低优先级 `OS_LOWEST_PRI0` 等参数来减小 RAM 和 ROM 的消耗。

4、 例程解析

4.1 Example 1

该例程是一个最简单的任务切换例程，在 `main` 函数中通过调用系统初始化函数 `OSInit` 函数来初始化操作系统环境，接着通过调用 `OSTaskCreate` 函数来创建起始任务 `TaskStart`，最后调用 `OSStart` 函数进行任务的调度，至此整个单片机交由操作系统进行调度。

在起始任务对应的函数 `TaskStart` 中调用了 `BSP_init` 函数，`BSP_init` 函数中主要调用了芯片各个模块的初始化代码，其中最重要的就是 `TIMO_INIT` 函数，因为采用 TIMO 作为系统的 `system tick` 时钟，因此该函数必须被调用，并且参数必须和 `os_cfg.h` 中的 `OS_TICKS_PER_SEC` 保持一致，否则将会导致系统中很多和时间相关的功能不准确，另外需要注意一点 `OS_TICKS_PER_SEC` 表示每秒钟系统进行调度的次数，按照一般程序来说，每秒调度 100 次左右即可满足大部分要求，不要将该值设置过大，因为操作系统调度的过程也是需要消耗一定的时间的，设置过大，用户任务将得不到有效执行，大部分时间消耗在任务调度上。

`TaskStart` 函数后续调用了 `OSTaskCreate` 分别创建两个任务 `Task1`、`Task2`，这两个任务就是用户任务，接下来自己调用了 `OSTaskDel` 函数将启动任务删除，因为启动任务也就在启动的时候进行系统相关的初始化以及用户任务的创建工作，至此其所有工作已经完成，没有必要再占用资源。官方给出的资料建议：不要在 `main` 函数中直接调用 `BSP_init` 等板级初始化函数，也不要再在 `main` 函数起始同时创建多个任务，而是先创建一个起始任务，在这个起始任务中完成这些工作。

在 `Task1` 和 `Task2` 任务中分别有两个无限循环，循环中分别进行打印信息，每次打印完后 `Task` 进行 4S、`Task` 进行 2S 的延时，让出 CPU 的控制权。

将代码写入 CH563 芯片，可以观察到串口 0 的输出如下。



```
stdio init ok.
task1 print CPU Usage:0
task2 OSTime:1000000000
task2 OSTime:1000000200
task1 print CPU Usage:1
task2 OSTime:1000000400
task2 OSTime:1000000600
task1 print CPU Usage:1
task2 OSTime:1000000800
task2 OSTime:1000001000
task1 print CPU Usage:1
```

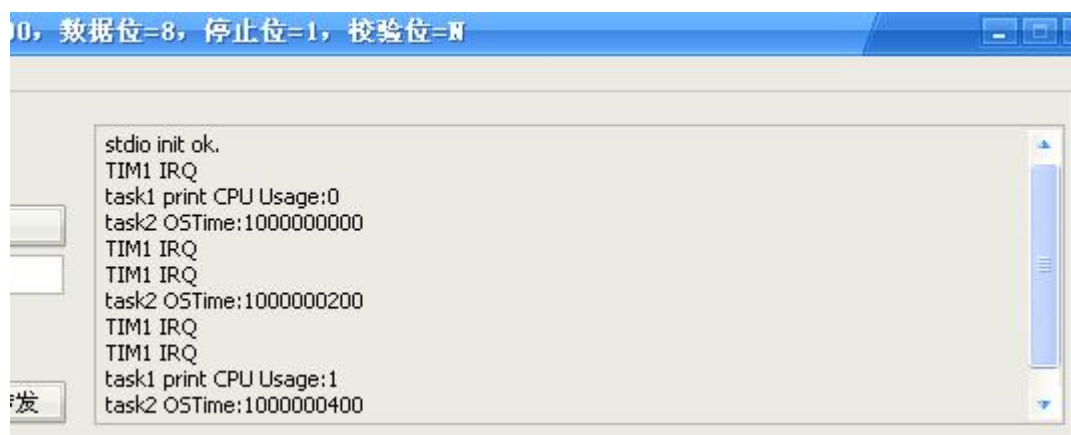
4.2 Example 2

该例程是在例程 1 的基础上添加了一个 TIM1 的 IRQ 非向量中断，以测试 IRQ 非向量中断在系统下是否正常使用。

前面的代码和 Example1 中相同，不在赘述。

在启动任务中对 TIM1 进行初始化，将 TIM1 设置为 1S 触发的溢出中断，在 IRQ 非向量中断函数中通过判断 IRQ 中断标志位知道是不是 TIM1 的中断，并在中断中进行打印（此处打印只是为了演示，在实际代码中不要在中断中加入耗时过长的代码）。

将代码写入 CH563 芯片，可以观察到串口 0 的输出如下。



```
0, 数据位=8, 停止位=1, 校验位=N
stdio init ok.
TIM1 IRQ
task1 print CPU Usage:0
task2 OSTime:1000000000
TIM1 IRQ
TIM1 IRQ
task2 OSTime:1000000200
TIM1 IRQ
TIM1 IRQ
task1 print CPU Usage:1
task2 OSTime:1000000400
```

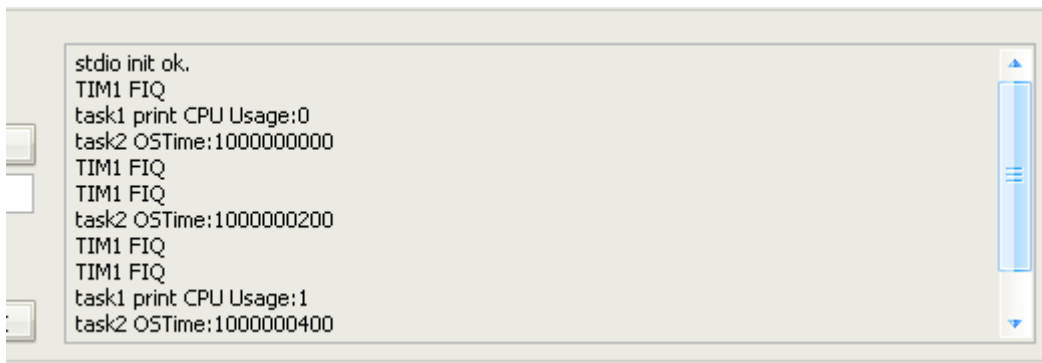
4.3 Example 3

该例程是在例程 1 的基础上添加了一个 TIM1 的 FIQ 非向量中断，以测试 FIQ 非向量中断在系统下是否正常使用。

前面的代码和 Example1 中相同，不在赘述。

在启动任务中对 TIM1 进行初始化，将 TIM1 设置为 1S 触发溢出中断，在 FIQ 非向量中断函数中通过判断 FIQ 中断标志位知道是不是 TIM1 的中断，并在中断中进行打印（此处打印只是为了演示，在实际代码中不要在中断中加入耗时过长的代码）。

将代码写入 CH563 芯片，可以观察到串口 0 的输出如下。



```
stdio init ok.  
TIM1 FIQ  
task1 print CPU Usage:0  
task2 OSTime:1000000000  
TIM1 FIQ  
TIM1 FIQ  
task2 OSTime:1000000200  
TIM1 FIQ  
TIM1 FIQ  
task1 print CPU Usage:1  
task2 OSTime:1000000400
```

5、 例程文件夹组织

ucosii 文件夹里为官方提供的有关 ucosII 的 C 语言源码，包括任务调度、内存分配、信号量、邮箱、队列等功能实现源文件；如非必要不要随便修改。

arm 文件夹里为根据 CH563 平台进行修改的有关 ucosII 任务切换以及中断宏定义的汇编文件，一般情况下不用修改此文件夹下文件；如果需要使用 CH563 的 SPI0 和 PB 的 IRQ 向量中断，需要修改部分代码。