STM32 μC/OS-II 稳定性测试方案

2013 年 6 月 15 日

修改记录

制定日期	生效日期	制定 / 修订 内容摘要	页数	版本	拟稿	审查	批准
2013.06.20		μC/OS-II 稳定性测试方案		0.01	朱正晶		

目 录

1 简介

本节将简要说明手册的目的、范围。

1.1 手册目的

本手册的目的在于说明 STM32F207VG 运行移植好的 μC/OS-II 系统稳定性测试方案。

1.2 手册范围

本手册首先简要地介绍 μC/OS-II 在 STM32F207VG 芯片上稳定性测试的方案，然后说明测试的具体步骤。

本手册的使用者包括：

程序编写、维护者

…

2 µC/OS-II 稳定性

我们系统使用官网上最新的 µC/OS-II 内核，版本为 V2.92.07。移植原理在文档《µC/OS-II 在 STM32F207VGTb 上的移植及使用手册.pdf》中有详细说明。µC/OS-II 是基于 µC/OS 的，µC/OS 自 1992 年以来已经有成百上千的商业应用。µC/OS-II 与 µC/OS 的内核是一样的，只不过提供了更多的功能，系统稳定性与可靠性还是有保证的。

2.1 µC/OS-II 稳定性测试方案简介

为了测试 µC/OS-II 内核在我们的 STM32F207 芯片上运行的稳定性，本测试方案采用间接的方法，即调用 µC/OS-II 系统的 API，每个 API 都做好出错处理（一旦出错就使用 log 记录下来）。然后尽量让 CPU 多做一些测试任务，模拟实际使用的环境，并且使 CPU 资源的占用在 90%以上。编写完善的 log 记录系统，增加和系统的交互能力，在任何情况下都能自由查看系统当前状态。本测试方案不对任何 µC/OS-II 内核函数进行改写，µC/OS-II 系统保持原样。

为此，我们根据 µC/OS-II 内核提供的功能，编写测试任务，并且使测试系统长时间，并且在严酷环境下运行，这样有利于发现更多问题。本测试系统可以和项目开发并行。项目中实现的一些功能可以放到测试系统中进行测试。

有了 µC/OS-II 系统，我们每个开发人员只需要编写自己的模块（任务 task），模块设计时需要保持独立性，也就是各个模块之间的耦合度要低，这样在移植和调试时能很快的进行。

由于 µC/OS-II 内核只提供了基本的内核服务，当我们在具体工程中使用时可能需要加入很多外围系统，比如文件系统，调试系统，TCP/IP 协议栈等等。本测试系统会使用到这些子系统，在实现时尽量将这些外围系统模块化，方便以后的参考或使用。

2.2 µC/OS-II 系统 API 测试说明

µC/OS-II 提供了许多系统 API 供我们使用。本测试系统中尽量多调用系统 API，初步选用如下测试 API。

Service 系统 API	*测试系统调用情况*	*说明*
Miscellaneous		
OSInit()	✔	µC/OS-II 系统初始化
OSSchedLock()	✔	调度器上锁，禁止任务调度
OSSchedUnlock()	✔	调度器解锁，打开任务调度
OSStart()	✔	µC/OS-II 系统启动
OSStatInit()	✔	得系统状态信息初始化
OSVersion()	✔	系统版本号
Interrupt Management		
OSIntEnter()	✔	中断函数进入时调用
OSIntExit()	✔	中断函数退出时调用
Event Flags		
OSFlagAccept()	✖	无等待地获取事件标志组中的事件标志

OSFlagCreate()	✖	建立一个事件标志组
OSFlagDel()	✖	删除一个事件标志组
OSFlagPend()	✖	等待事件标志组的事件标志位
OSFlagPost()	✖	置位或清 0 事件标志组中的事件标志
OSFlagQuery()	✖	查询事件标志组的状态
Message Mailboxes		
OSMboxAccept()	✖	等待邮箱中的消息
OSMboxCreate()	✖	建立一个邮箱
OSMboxDel()	✖	删除一个邮箱
OSMboxPend()	✖	等待邮箱中的消息
OSMboxPost()	✖	向邮箱发送一则消息
OSMboxPostOpt()	✖	向邮箱发送一则消息，新加的函数
OSMboxQuery()	✖	查询一个邮箱的状态
Memory Partition Management		
OSMemCreate()	✔	建立一个内存分区
OSMemGet()	✔	分配一个内存块
OSMemPut()	✔	释放一个内存块
OSMemQuery()	✔	查询一个内存分区的状态
Mutex Management		
OSMutexAccept()	✔	无等待地获取互斥型信号量(任务不挂起)
OSMutexCreate()	✔	建立一个互斥型信号量
OSMutexDel()	✔	互斥型删除
OSMutexPend()	✔	等待一个互斥型信号量（挂起）
OSMutexPost()	✔	释放一个互斥型信号量
OSMutexQuery()	✔	获取互斥型信号量的当前状态
Message Queues		
OSQAccept()	✔	无等待地从消息队列中获得消息
OSQCreate()	✔	建立一个消息队列
OSQDel()	✖	删除一个消息队列
OSQFlush()	✖	清空消息队列
OSQPend()	✔	等待消息队列中的消息
OSQPost()	✔	向消息队列发送一则消息（FIFO）
OSQQuery()	✔	获取消息队列的状态
Semaphore Management		
OSSemAccept()	✔	无等待地请求一个信号量
OSSemCreate()	✔	建立一个信号量
OSSemDel()	✖	删除一个信号量
OSSemPend()	✔	等待一个信号量
OSSemPost()	✔	发出一个信号量
OSSemQuery()	✔	查询一个信号量的当前状态
Task Management		

OSTaskChangePrio()	✖	改变任务运行时的优先级
OSTaskCreate()	✖	创建任务的基本函数
OSTaskCreateExt()	✔	创建任务扩展函数，提供一些高级功能
OSTaskDel()	✔	删除任务
OSTaskDelReq()	✔	删除包含信号量等系统资源的任务
OSTaskResume()	✔	恢复任务运行
OSTaskStkChk()	✔	检查任务栈使用情况
OSTaskSuspend()	✔	将任务挂起
OSTaskQuery()	✔	获取任务信息，该函数一般用于调试
Time Management		
OSTimeDly()	✔	将调用该 API 的任务延时多少时间片
OSTimeDlyHMSM()	✔	将任务延时具体的时间（时分秒毫秒）
OSTimeDlyResume()	✔	取消任务延时状态，任务状态变为就绪
OSTimeGet()	✔	获取系统时钟节拍
OSTimeSet()	✔	改变系统时钟节拍
OSTimeTick()	✔	节拍处理函数
User-Defined Functions		
OSTaskCreateHook()	✖	用户定义的钩子函数，扩展系统功能
OSTaskDelHook()	✖	本测试系统没有使用
OSTaskStatHook()	✖	
OSTaskSwHook()	✖	
OSTimeTickHook()	✖	

说明：

1）有些系统 API 没有特地去调用测试是因为这些 API 已经被其他 API 调用。比如事件标志组（Event Flags）API，消息队列（Message Queues）API 的实现就是以事件标志组 API 为基础。这里为了更加有效的测试，没有对此类 API 进行调用。

2）用户自定义的 API 钩子函数这里也没有调用。这些函数主要用于扩展系统功能。本测试系统不对其进行测试。

2.3 测试框架图

本系统使用如下 STM32F207 芯片外设模块来测试 μC/OS-II 的 API：

1） **USART** PC 调试接口，实现一个 Shell 接口，测试 OS 中断管理 API
2） **WATCHDOG** 系统监视，OS 宕机及时发现
3） **TIMER** 定时器，和 GPIO 配置测试 OS 定时器部分
4） **ADC** 可以搞个项目中的传感器测试，测试 OS 队列，OS 定时器
5） **GPIO** 流水灯指示系统状态，测试 OS 延时功能，信号量
6） **PWM** 呼吸灯，测试 OS 定时器
7） **USB** 传输大数据给 PC，测试 OS 负载能力
8） **DCMI** 摄像头模块配合 USB 模块，传输数据给 PC，测试 OS 高负载下的稳定性

9） **FSMC LCD**	调试信息实时显示，测试 OS 任务创建，OS 的优先级效果
10）**FSMC NAND**	移植 FAT FS 文件系统，相关 OS 应用，提升系统负载
11）**CAN 网络**	测试 OS 队列，内存管理，定时器
12）**RTC**	读取系统时间，供其他模块使用，测试 OS 消息队列
13）**IIC**	IIC FALSH 的使用，测试 OS 互斥量 Mutex
14）**SPI**	SPI FALSH 的使用，保存一些参数，测试 OS 消息队列
15）**ETH**	加入网卡功能，实现 WEB SERVER，接入公司局域网。测试系统稳定性，可以远程查看测试系统当前状态
16）**DS18B20**	一线温度传感器，在有 OS 的情况下怎样产生严格的时序

系统框图如图 2-1 所示：

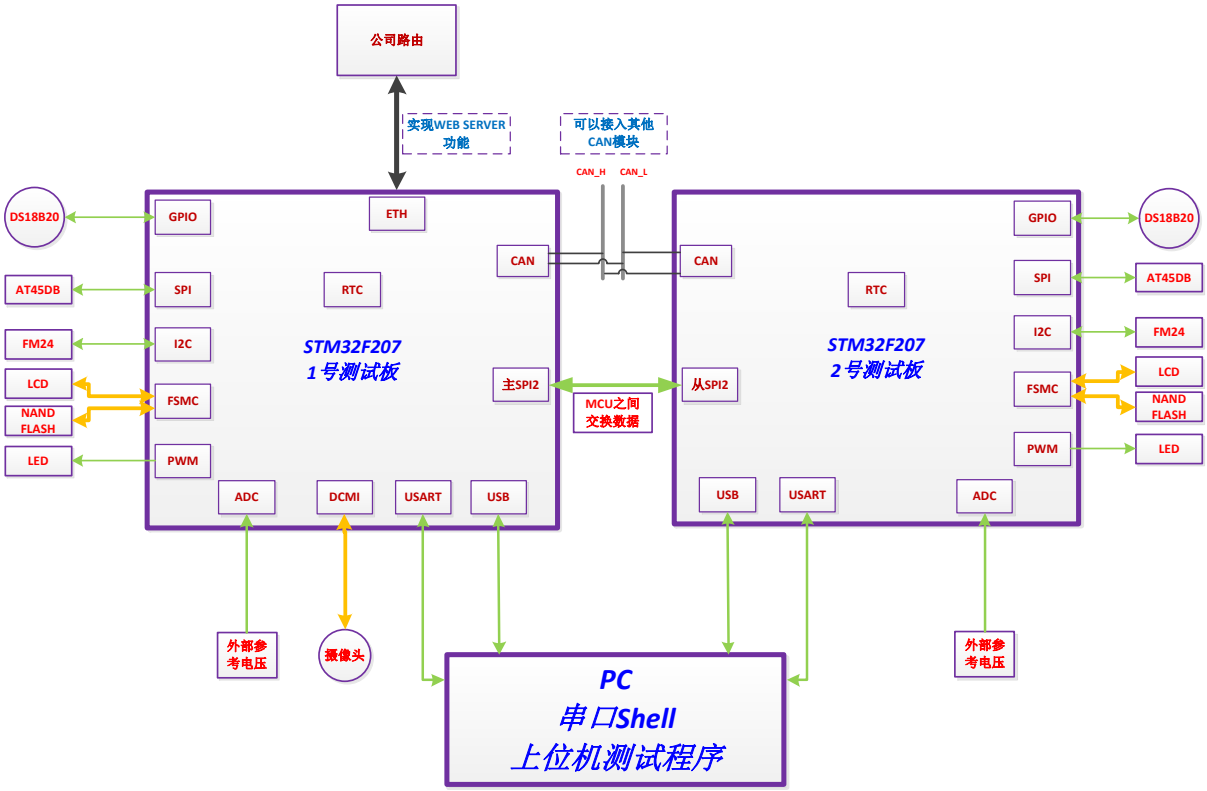

图 2-1 测试系统框架

下一章将对这些模块对系统进行测试的方法进行详细说明。

3 各模块具体测试步骤

本章详细讲解各个模块的测试方法，每个模块使用哪些 OS 资源。

3.1 USART 实现 Shell

使用过 Linux 的人可能都知道 Shell 解释程序，本测试系统中实现一个简单的 Shell 的解释程序。使用这个 Shell 解释程序和系统进行实时交互，比如查看系统状态，执行一些特定的程序。

实现时注意模块化，和具体硬件相关的独立出来，方便以后的移植和使用。

需要使用 OS API：任务创建，任务延时。

3.2 WATCHDOG 监控系统状态

创建一个最低优先级的任务，在这个任务中进行“喂狗”操作。如果系统因为某些原因无法调度到这个最低优先级的任务，那么 WATCHDOG 超时后系统即重启。系统每次启动时都需要检查 WATCHDOG 状态寄存器，如果是 WATCHDOG 引起的系统重启则记录下来。

调用的 OS API：任务创建，任务延时。

3.3 TIMER 定时器

OS TICK 就在定时器中断中进行，本测试系统使用 10ms 一次 tick。

调用 OS API：定时 TICK

3.4 ADC 读取

使用消息队列的方式，ADC 读取 task 一直在等待读取 ADC 的消息。一旦收到消息，即启动 ADC 读取。

调用 OS API：任务管理，消息队列，内存管理。

3.5 GPIO 控制 LED

使用 GPIO 来控制 LED，采用消息队列的方式实现任务间的通信，此任务主要是方便查看系统的状态。

调用 OS API：任务创建，消息队列。

3.6 PWM 呼吸灯

使用 PWM 实现一个呼吸灯的功能。此任务主要是为了测试 PWM。

调用 OS API：任务管理

3.7 USB 和 DCMI 摄像头模块

采用 USB 将摄像头拍摄的图像上传上位机，并且在本地系统保持下来（结合文件系统的使用）。

3.8 LCD 调试功能

将一些信息显示在 LCD 上，实现调试功能。

3.9 NAND 文件系统

在 NAND 上面实现一个文件系统，存储系统信息，测试其长时间读写的稳定性。采用模块化设计，方便以后项目中使用。

3.10 CAN 网络

使用之前测试过的方法，进行 CAN 网络的稳定性测试。具体参考文档《STM32 CAN 总线稳定性测试报告.doc》

调用 OS API：任务管理，消息队列，内存管理，中断管理，定时器管理。

3.11 RTC

设置系统时间，方便测试。

3.12 I2C 和 SPI 测试

I2C 总线读写 FM24 FLASH，SPI 读写 AT45DB FLASH。另外使用另一路 SPI 来和另外一块测试板进行 MCU 之间的通信。

调用 OS API：任务管理，消息队列，内存管理，中断处理。

3.13 ETH

在 STM32 上移植 LwIP TCP/IP 协议栈，将测试板接入公司路由，实现在该局域网中远程查看测试板当前状态。

3.14 DS18B20 温度传感器

DS18B20 对操作时序要求比较严格，在 RTOS 系统中由于任务调度的存在，使得延时函数并不准确，在这里我们使用一个单独的硬件定时器来实现精确定时的效果。

调用 OS API：任务管理

4 补充说明

此测试系统一些简单的功能已经实现，比如 USART、TIMER、PWM、GPIO 等驱动。比较复杂的子模块还没有实现，比如 NAND 文件系统、TCP/IP 协议栈等。但这些子模块有现成的开源实现，我们只需要进行移植，主要的工作还是需要对这些移植模块进行稳定性测试。也就是说，本测试系统虽然在做 μC/OS-II 内核的稳定性测试，但我们同时也间接地测试了各个模块的稳定性。以后的项目可以以此测试系统为基础，进行进一步的开发。

后面希望大家有时间加入本测试系统的开发和调试。

最后，由于个人能力有限，欢迎大家对本测试系统提出不足和建议，让我们一起把基础架构做好！