HZ BOOKS
华章科技

从0到1教你写μC/OS-III内核，详解各个内核组件如何使用。
由浅入深，结合野火STM32全系列开发板，提供完整源代码，
极具可操作性。

野火
野火嵌入式系列

μC/OS-III™

The Real-Time Kernel

RTOS

μC/OS-III内核实现与应用开发实战指南

基于STM32

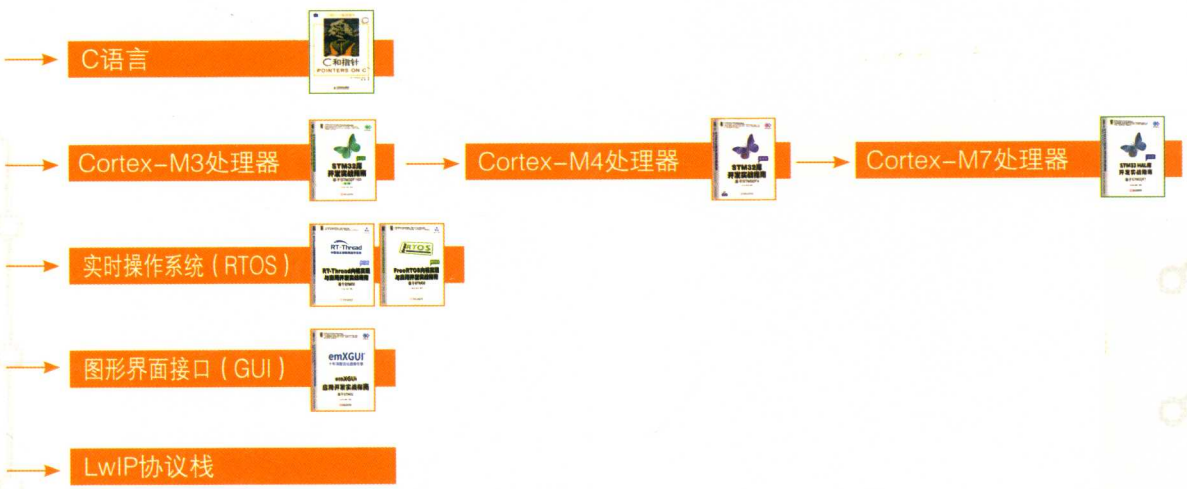
刘火良 杨森 编著

机械工业出版社
China Machine Press

本书基于野火STM32全系列开发板介绍μC/OS-III内核实现与应用开发，全书分为两部分：第一部分先教你如何从0到1把μC/OS-III内核写出来，从底层的汇编开始讲解任务如何定义、如何切换，还讲解了阻塞延时如何实现、如何支持多优先级、如何实现时基列表以及时间片等μC/OS的核心知识点；第二部分讲解μC/OS-III内核组件的应用以及使用μC/OS-III进行多任务编程。

本书内容翔实，案例丰富，配有大量示例代码，适合作为嵌入式领域科技工作者的参考书，也适合相关专业的学生学习参考。

嵌入式开发学习曲线



上架指导：计算机/嵌入式
ISBN 978-7-111-62824-8
9 787111 628248 >
定价：129.00元

投稿热线：(010) 88379604
购书热线：(010) 68326294
客服热线：(010) 88379426 88361066
华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

野火

野火嵌入式系列

µC/OS-III™

The Real-Time Kernel

RTOS

µC/OS-III内核实现与应用开发实战指南

基于STM32

刘火良 杨森 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

μC/OS-III 内核实现与应用开发实战指南：基于 STM32 / 刘火良，杨森编著．—北京：机械工业出版社，2019.6
（电子与嵌入式系统设计丛书）

ISBN 978-7-111-62824-8

I. μ…　II. ① 刘…　② 杨…　III. 实时操作系统－指南　IV. TP316.2-62

中国版本图书馆 CIP 数据核字（2019）第 097540 号

μC/OS-III 内核实现与应用开发实战指南：基于 STM32

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：赵亮宇　　责任校对：殷　虹
印　　刷：北京文昌阁彩色印刷有限责任公司　　版　　次：2019 年 7 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：32.5
书　　号：ISBN 978-7-111-62824-8　　定　　价：129.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　　读者信箱：hzit@hzbook.com

前　言

如何学习本书

本书从0开始教你如何把μC/OS-III写出来，既讲解源码实现，也讲解API如何使用。当你拿到本书开始学习时一定会惊讶，原来RTOS（Real Time Operation System，实时操作系统）的学习并没有那么复杂，原来自己也可以写操作系统，成就感立马爆棚。

全书内容循序渐进，不断迭代，前一章都是后一章的基础，因此最好从头开始阅读，不要跳跃。在学习时务必做到两点：一是不能一味地看书，要把代码和书本结合起来学习，一边看书，一边调试代码。如何调试代码呢？即单步执行每一条程序，看程序的执行流程和执行效果与自己所想的是否一致；二是在每学完一章之后，必须将配套的例程重写一遍（切记不要复制，哪怕是一个分号，但可以照书录入），以做到举一反三，确保真正理解。在自己写的时候肯定会错漏百出，这个时候要认真纠错，好好调试，这是你提高编程能力的最好机会。记住，编写程序不是一气呵成的，而是要一步一步地调试。

本书的编写风格

本书以μC/OS-III官方源码为蓝本，抽丝剥茧，不断迭代，教你逐步写出μC/OS-III。书中涉及的数据类型、变量名称、函数名称、文件名称、文件存放的位置都完全按照μC/OS-III官方的方式来实现。学完本书之后，可以无缝地切换到原版的μC/OS-III中使用。要注意的是，在实现的过程中某些函数中会去掉一些形参和冗余的代码，只保留核心的功能，但这并不会影响学习。注意，本书的目的并不是让你自己写一个操作系统，而是让你了解μC/OS-III是如何写出来的，着重讲解原理实现，当你学完这本书之后，再学习其他RTOS将会事半功倍。

本书的技术论坛

如果在学习过程中遇到问题，可以到野火电子论坛www.firebbs.cn发帖交流，开源共享，共同进步。

鉴于水平有限，本书难免有错漏之处，热心的读者也可把勘误发送到论坛上以便改进。祝你学习愉快，μC/OS-III的世界，野火与你同行。

引　言

为什么学习 RTOS

当我们进入嵌入式这个领域时，首先接触的往往是单片机编程，单片机编程又首选 51 单片机来入门。这里说的单片机编程通常都是指裸机编程，即不加入任何 RTOS 的编程。常用的 RTOS 有国外的 FreeRTOS、μC/OS、RTX 以及国内的 Huawei LiteOS、RT-Thread 等，其中开源且免费的 FreeRTOS 的市场占有率最高，历史悠久的 μC/OS 位居第二。

在裸机系统中，所有的程序基本都是用户自己写的，所有的操作都是在一个无限的大循环中实现。现实生活中的很多中小型电子产品中用的都是裸机系统，而且能够满足需求。那为什么还要学习 RTOS 编程，要涉及一个操作系统呢？一是基于项目需求，随着产品要实现的功能越来越多，单纯的裸机系统已经不能完美地解决问题，反而会使编程变得更加复杂，如果想降低编程的难度，可以考虑引入 RTOS 实现多任务管理，这是使用 RTOS 的最大优势。二是出于学习的需要，必须学习更高级的技术，实现更好的职业规划，为将来能有更好的职业发展做准备，而不是拘泥于裸机编程。作为一个合格的嵌入式软件工程师，学习是永远不能停止的，时刻都要为将来做准备。“书到用时方恨少”，希望当机会来临时，你不要有这种感觉。

为了帮大家厘清 RTOS 编程的思路，本书将简单分析这两种编程方式的区别，我们称之为“学习 RTOS 的命门”，只要掌握这一关键内容，以后的 RTOS 学习可以说是易如反掌。在讲解这两种编程方式的区别时，我们主要讲解方法论，不会涉及具体的代码，即主要还是通过伪代码来讲解。

如何学习 RTOS

RTOS 编程和裸机编程的风格不一样，而且很多人说学习 RTOS 很难，这就导致想要学习的人一听到 RTOS 编程就会忌惮三分，结果就是“出师未捷身先死”。

那么到底如何学习 RTOS 呢？最简单的方法就是在别人移植好的系统上先看一看 RTOS 中的 API 使用说明，然后调用这些 API 实现自己想要的功能，完全不用关心底层的移植，这是最简单、快速的入门方法。这种方法有利有弊：如果是做产品，好处是可以快速实现功能，

将产品推向市场，赢得先机；弊端是当程序出现问题时，因对 RTOS 不够了解，会导致调试困难。如果想系统地学习 RTOS，那么只会简单地调用 API 是不可取的，我们应该深入学习一款 RTOS。

目前市场上的 RTOS，内核实现方式差异不大，只需要深入学习其中一款即可。万变不离其宗，只要掌握了一款 RTOS，以后换到其他型号的 RTOS，使用起来自然也得心应手。那么如何深入地学习一款 RTOS 呢？这里有一个非常有效但也十分难的方法，就是阅读 RTOS 的源码，深入研究内核和每个组件的实现方式。这个过程枯燥且痛苦，但为了能够学到 RTOS 的精华，还是很值得一试的。

市面上虽然有一些讲解 RTOS 源码的书，但如果基础知识掌握得不够，且先前没有使用过该款 RTOS，那么只看源码不仅非常枯燥，而且难以从全局掌握整个 RTOS 的构成和实现。

现在，我们采用一种全新的方法来教大家学习一款 RTOS，既不是单纯地介绍其中的 API 如何使用，也不是单纯地拿里面的源码一句句地讲解，而是从 0 开始，层层叠加，不断完善，教大家如何把一个 RTOS 从 0 到 1 写出来，让你在每一个阶段都能享受到成功的喜悦。在这个 RTOS 实现的过程中，只需要具备 C 语言基础即可，然后就是跟着本书笃定前行，最后定有所成。

如何选择 RTOS

如何选择 RTOS 取决于是学习还是做产品，如果是学习，则可以选择一个历史较久、商业化成功、安全验证较多的来学习，而且应深入学习。符合前面这几个标准的只有 μC/OS，所以学习 RTOS，首选 μC/OS，而且 μC/OS 的相关资料是很丰富的。当然，选择其他的 RTOS 来学习也是可以的。学完之后就要使用，如果是做产品，即在产品中使用 μC/OS，则要面临授权问题，需要支付一定的费用，所以开源免费的 FreeRTOS 受到了各个半导体厂商和开发者的青睐。目前，FreeRTOS 是市场占有率最高的 RTOS，非常适合用来做产品。另外，国内的 RT-Thread 也在迅速崛起，它同样是开源免费的，也是不错的选择。

目　录

前言

引言

第一部分　从 0 到 1 教你写 μC/OS 内核

第 1 章　新建工程——软件仿真 …… 2

1.1　新建本地工程文件夹 …… 2

1.2　使用 KEIL 新建工程 …… 2

1.2.1　New Project …… 2

1.2.2　Select Device For Target …… 3

1.2.3　Manage Run-Time Environment …… 3

1.3　在 KEIL 工程中新建文件组 …… 4

1.4　在 KEIL 工程中添加文件 …… 4

1.5　调试配置 …… 6

1.5.1　设置软件仿真 …… 6

1.5.2　修改时钟大小 …… 6

1.5.3　添加头文件路径 …… 7

第 2 章　裸机系统与多任务系统 …… 8

2.1　裸机系统 …… 8

2.1.1　轮询系统 …… 8

2.1.2　前后台系统 …… 9

2.2　多任务系统 …… 10

第 3 章　任务的定义与任务切换 …… 13

3.1　多任务系统中任务的概念 …… 14

3.2　创建任务 …… 15

3.2.1　定义任务栈 …… 15

3.2.2　定义任务函数 …… 16

3.2.3　定义任务控制块 …… 17

3.2.4　实现任务创建函数 …… 18

3.3　操作系统初始化 …… 22

3.4　启动系统 …… 24

3.5　任务切换 …… 27

3.6　main() 函数 …… 33

3.7　实验现象 …… 37

第 4 章　任务时间片运行 …… 39

4.1　SysTick 简介 …… 39

4.2　初始化 SysTick …… 40

4.3　编写 SysTick 中断服务函数 …… 41

4.4　main() 函数 …… 42

4.5　实验现象 …… 44

第 5 章　空闲任务与阻塞延时 …… 45

5.1　实现空闲任务 …… 45

5.1.1　定义空闲任务栈 …… 45

5.1.2　定义空闲任务的任务控制块 …… 46

5.1.3 定义空闲任务函数……47
5.1.4 空闲任务初始化……47
5.2 实现阻塞延时……48
5.3 main() 函数……50
5.4 实验现象……52

第 6 章 时间戳……53

6.1 时间戳简介……53
6.2 时间戳的实现……53
6.3 时间戳代码……54
6.3.1 CPU_Init() 函数……54
6.3.2 CPU_TS_Init() 函数……55
6.3.3 CPU_TS_TmrInit() 函数……56
6.3.4 BSP_CPU_ClkFreq() 函数……57
6.3.5 CPU_TS_TmrFreqSet() 函数…58
6.3.6 CPU_TS_TmrRd() 函数……58
6.3.7 OS_TS_GET() 函数……59
6.4 main() 函数……59
6.5 实验现象……61

第 7 章 临界段……62

7.1 临界段简介……62
7.2 Cortex-M 内核快速关中断指令……62
7.3 关中断……63
7.4 开中断……63
7.5 临界段代码的应用……64
7.6 测量关中断时间……68
7.6.1 测量关中断时间初始化……68
7.6.2 测量最大关中断时间……69
7.6.3 获取最大关中断时间……70
7.7 main() 函数……71
7.8 实验现象……71

第 8 章 就绪列表……72

8.1 优先级表的定义及函数……72
8.2 就绪列表的定义及函数……80
8.3 main() 函数……88
8.4 实验现象……88

第 9 章 多优先级……89

9.1 定义优先级相关全局变量……89
9.2 修改 OSInit() 函数……89
9.3 修改任务控制块……90
9.4 修改 OSTaskCreate() 函数……90
9.5 修改 OS_IdleTaskInit() 函数……92
9.6 修改 OSStart() 函数……93
9.7 修改 PendSV_Handler() 函数……93
9.8 修改 OSTimeDly() 函数……94
9.9 修改 OSSched() 函数……95
9.10 修改 OSTimeTick() 函数……97
9.11 main() 函数……98
9.12 实验现象……101

第 10 章 时基列表……103

10.1 实现时基列表……103
10.1.1 定义时基列表变量……103
10.1.2 修改任务控制块……104
10.1.3 实现时基列表相关函数……106
10.2 修改 OSTimeDly() 函数……113
10.3 修改 OSTimeTick() 函数……114
10.4 main() 函数……114
10.5 实验现象……114

第 11 章 时间片……115

11.1 实现时间片……115

11.1.1 修改任务控制块 …… 115
11.1.2 实现时间片调度函数 …… 116
11.2 修改 OSTimeTick() 函数 …… 118
11.3 修改 OSTaskCreate() 函数 …… 118
11.4 修改 OS_IdleTaskInit() 函数 …… 120
11.5 main() 函数 …… 120
11.6 实验现象 …… 122

第 12 章 任务的挂起和恢复 …… 124

12.1 实现任务的挂起和恢复 …… 124
12.1.1 定义任务的状态 …… 124
12.1.2 修改任务控制块 …… 125
12.1.3 编写任务挂起和恢复函数 … 126
12.2 main() 函数 …… 131
12.3 实验现象 …… 133

第 13 章 任务的删除 …… 134

13.1 实现任务删除 …… 134
13.2 main() 函数 …… 136
13.3 实验现象 …… 136

第二部分 μC/OS-III 内核应用开发

第 14 章 移植 μC/OS-III 到 STM32 …… 138

14.1 获取 STM32 的裸机工程模板 … 138
14.2 下载 μC/OS-III 源码 …… 138
14.3 μC/OS-III 源码文件介绍 …… 141
14.3.1 EvalBoards …… 141
14.3.2 μC-CPU …… 141
14.3.3 μC-LIB …… 143
14.3.4 μC/OS-III …… 143
14.4 移植到 STM32 工程 …… 145
14.4.1 在工程中添加文件分组 …… 145
14.4.2 添加文件到对应分组 …… 145
14.4.3 添加头文件路径到工程中 … 146
14.4.4 具体的工程文件修改 …… 147
14.4.5 修改源码中的 bsp.c 与 bsp.h 文件 …… 149
14.5 按需配置最适合的工程 …… 151
14.5.1 os_cfg.h …… 151
14.5.2 cpu_cfg.h …… 153
14.5.3 os_cfg_app.h …… 154
14.6 修改 app.c …… 155
14.7 下载验证 …… 157

第 15 章 创建任务 …… 158

15.1 硬件初始化 …… 158
15.2 创建单任务 …… 160
15.2.1 定义任务栈 …… 160
15.2.2 定义任务控制块 …… 160
15.2.3 定义任务主体函数 …… 160
15.2.4 创建任务 …… 161
15.2.5 启动任务 …… 163
15.2.6 app.c …… 163
15.3 下载验证单任务 …… 166
15.4 创建多任务 …… 166
15.5 下载验证多任务 …… 171

第 16 章 μC/OS-III 的启动流程 … 172

16.1 “万事俱备，只欠东风”法 …… 172
16.2 “小心翼翼，十分谨慎”法 …… 173
16.3 两种方法的适用情况 …… 175
16.4 系统的启动 …… 175
16.4.1 系统初始化 …… 175

16.4.2 CPU 初始化……179
16.4.3 SysTick 初始化……182
16.4.4 内存初始化……183
16.4.5 OSStart() 函数……183
16.4.6 app.c……184

第 17 章 任务管理……187

17.1 任务的基本概念……187
17.2 任务调度器的基本概念……188
17.3 任务状态迁移……188
17.4 μC/OS 的任务状态……190
17.5 常用的任务函数……190
17.5.1 任务挂起函数 OS_TaskSuspend()……191
17.5.2 任务恢复函数 OSTaskResume()……194
17.5.3 任务删除函数 OSTaskDel()…198
17.5.4 任务延时函数……201
17.6 任务的设计要点……212
17.7 任务管理实验……213
17.8 实验现象……222

第 18 章 消息队列……223

18.1 消息队列的基本概念……223
18.2 消息队列的工作过程……224
18.2.1 消息池初始化……224
18.2.2 消息队列的运作机制……226
18.3 消息队列的阻塞机制……227
18.4 消息队列的应用场景……227
18.5 消息队列的结构……228
18.6 消息队列常用函数……230
18.6.1 消息队列创建函数 OSQCreate()……230
18.6.2 消息队列删除函数 OSQDel()……233
18.6.3 消息队列发送函数 OSQPost()……239
18.6.4 消息队列获取函数 OSQPend()……249
18.7 使用消息队列的注意事项……256
18.8 消息队列实验……257
18.9 实验现象……263

第 19 章 信号量……264

19.1 信号量的基本概念……264
19.1.1 二值信号量……264
19.1.2 计数信号量……265
19.2 信号量的应用场景……265
19.3 二值信号量的运作机制……266
19.4 计数信号量的运作机制……267
19.5 信号量控制块……267
19.6 信号量函数……268
19.6.1 信号量创建函数 OSSemCreate()……268
19.6.2 信号量删除函数 OSSemDel()……271
19.6.3 信号量释放函数 OSSemPost()……275
19.6.4 信号量获取函数 OSSemPend()……280
19.7 使用信号量的注意事项……285
19.8 信号量实验……286
19.8.1 二值信号量同步实验……286
19.8.2 计数信号量实验……292
19.9 实验现象……298
19.9.1 二值信号量同步实验现象…298
19.9.2 计数信号量实验现象……299

第 20 章　互斥量 …… 300

20.1　互斥量的基本概念 …… 300
20.2　互斥量的优先级继承机制 …… 300
20.3　互斥量的应用场景 …… 303
20.4　互斥量的运作机制 …… 303
20.5　互斥量控制块 …… 304
20.6　互斥量函数 …… 306
20.6.1　创建互斥量函数 OSMutexCreate() …… 306
20.6.2　删除互斥量函数 OSMutexDel() …… 308
20.6.3　获取互斥量函数 OSMutexPend() …… 314
20.6.4　释放互斥量函数 OSMutexPost() …… 320
20.7　互斥量相关实验 …… 324
20.7.1　模拟优先级翻转实验 …… 324
20.7.2　互斥量实验 …… 330
20.8　实验现象 …… 336
20.8.1　模拟优先级翻转实验现象 …… 336
20.8.2　互斥量实验现象 …… 336

第 21 章　事件 …… 338

21.1　事件的基本概念 …… 338
21.2　事件的应用场景 …… 339
21.3　事件的运作机制 …… 339
21.4　事件控制块 …… 341
21.5　事件函数 …… 342
21.5.1　事件创建函数 OSFlagCreate() …… 342
21.5.2　事件删除函数 OSFlagDel() …… 344
21.5.3　事件设置函数 OSFlagPost() …… 348
21.5.4　事件等待函数 OSFlagPend() …… 355
21.6　事件实验 …… 366
21.7　实验现象 …… 371

第 22 章　软件定时器 …… 372

22.1　软件定时器的基本概念 …… 372
22.2　软件定时器的应用场景 …… 374
22.3　软件定时器的精度 …… 374
22.4　软件定时器控制块 …… 375
22.5　软件定时器函数 …… 376
22.5.1　软件定时器创建函数 OSTmrCreate() …… 376
22.5.2　软件定时器启动函数 OSTmrStart() …… 379
22.5.3　软件定时器列表管理 …… 384
22.5.4　软件定时器停止函数 OSTmrStop() …… 387
22.5.5　软件定时器删除函数 OSTmrDel() …… 391
22.6　软件定时器任务 …… 393
22.7　软件定时器实验 …… 398
22.8　实验现象 …… 403

第 23 章　任务信号量 …… 405

23.1　任务信号量的基本概念 …… 405
23.2　任务信号量函数 …… 406
23.2.1　任务信号量释放函数 OSTaskSemPost() …… 406
23.2.2　任务信号量获取函数 OSTaskSemPend() …… 410
23.3　任务信号量实验 …… 414
23.3.1　任务信号量代替二值

信号量实验……414
23.3.2 任务信号量代替计数信号量实验……419
23.4 实验现象……424
23.4.1 任务信号量代替二值信号量实验现象……424
23.4.2 任务信号量代替计数信号量实验现象……425

第 24 章 任务消息队列……426
24.1 任务消息队列的基本概念……426
24.2 任务消息队列函数……427
24.2.1 任务消息队列发送函数 OSTaskQPost()……427
24.2.2 任务消息队列获取函数 OSTaskQPend()……431
24.3 任务消息队列实验……435
24.4 实验现象……439

第 25 章 内存管理……440
25.1 内存管理的基本概念……440
25.2 内存管理的运作机制……441
25.3 内存管理的应用场景……443
25.4 内存管理函数……443
25.4.1 内存池创建函数 OSMemCreate()……443
25.4.2 内存申请函数 OSMemGet()……447
25.4.3 内存释放函数 OSMemPut()……449
25.5 内存管理实验……451
25.6 实验现象……456

第 26 章 中断管理……457
26.1 异常与中断的基本概念……457
26.1.1 与中断相关的硬件……458
26.1.2 与中断相关的术语……458
26.2 中断的运作机制……459
26.3 中断延迟的概念……460
26.4 中断的应用场景……461
26.5 ARM Cortex-M 的中断管理……461
26.6 中断延迟发布……463
26.6.1 中断延迟发布的概念……463
26.6.2 中断队列控制块……465
26.6.3 中断延迟发布任务初始化函数 OS_IntQTaskInit()……466
26.6.4 中断延迟发布过程函数 OS_IntQPost()……468
26.6.5 中断延迟发布任务函数 OS_IntQTask()……471
26.7 中断管理实验……475
26.8 实验现象……481

第 27 章 CPU 利用率及栈检测统计……482
27.1 CPU 利用率的基本概念及作用…482
27.2 CPU 利用率统计初始化……483
27.3 栈溢出检测概念及作用……485
27.4 栈溢出检测过程……487
27.5 统计任务函数 OS_StatTask()……487
27.6 栈检测函数 OSTaskStkChk()……493
27.7 任务栈大小的确定……496
27.8 CPU 利用率及栈检测统计实验…496
27.9 实验现象……502

附录……503

第一部分

从 0 到 1 教你写 μC/OS 内核

本部分以 μC/OS-III 为蓝本，抽丝剥茧，不断迭代，教你如何从 0 开始把 μC/OS-III 写出来。这一部分着重讲解 μC/OS-III 实现的过程，当你学完这部分之后，再来重新使用 μC/OS-III 或者其他 RTOS，将会得心应手，不仅知其然，而且知其所以然。在源码实现的过程中，涉及的数据类型、变量名称、函数名称、文件名称以及文件的存放目录都会完全按照 μC/OS-III 的来实现，一些不必要的代码将会被剔除，但这并不影响我们理解整个操作系统的功能。

本部分几乎每一章都以前一章为基础，环环相扣，逐渐揭开 μC/OS-III 的神秘面纱，让人读起来会有一种豁然开朗的感觉。如果把代码都敲一遍，仿真时得出的效果与书中给出的一样，那从心底油然而生的成就感简直就要爆棚，让人恨不得一下子把本书读完，真是看了还想看，读了还想读。

第 1 章
新建工程——软件仿真

在开始写 RTOS 之前，先新建一个工程，Device 选择 Cortex-M3 内核的处理器，调试方式选择软件仿真，到最后写完整个 RTOS 之后，再把 RTOS 移植到野火 STM32 开发板上。最后的移植其实已经非常简单，只需要换一下启动文件并添加 bsp 驱动即可。

1.1 新建本地工程文件夹

在开始新建工程之前，我们先在本地计算机端新建一个文件夹用于存放工程。文件夹名设置为 RTOS，然后在该文件夹下新建各个文件夹和文件，有关这些文件夹的包含关系和作用如表 1-1 所示。

表 1-1 工程文件夹根目录下的文件夹的作用

文件夹名称			文件夹作用
Doc	—	—	用于存放整个工程的说明文件，如 readme.txt。通常情况下，我们要对整个工程实现的功能、如何编译、如何使用等做一个简要的说明
Project	—	—	用于存放新建的工程文件
User	μC/OS-III	Source	用于存放 μC/OS-III 源码，其中的代码是纯软件相关的，与硬件无关
		Ports	用于存放接口文件，即 μC/OS-III 与 CPU 连接的文件，也就是我们通常所说的移植文件。要想在单片机上运行 μC/OS-III，这些移植文件必不可少
	μC-CPU	—	用于存放 μC/OS-III 根据 CPU 总结的通用代码，只与 CPU 相关
	μC-LIB	—	用于存放一些 C 语言函数库
	—	—	用于存放用户程序，如 app.c，main() 函数就放在 app.c 文件中

1.2 使用 KEIL 新建工程

开发环境我们使用 KEIL5，版本为 5.15，高于版本 5 即可。

1.2.1 New Project

首先打开 KEIL5 软件，新建一个工程，工程文件放在目录 Project\RVMDK（uv5）下面，

名称为 YH-μC/OS-III，其中 YH 是野火拼音首字母的缩写，当然你也可以换成其他名称，但是必须是英文，不能是中文。

1.2.2　Select Device For Target

当设置好工程名称之后会弹出 Select Device for Target 对话框，在该对话框中可以选择处理器，这里选择 ARMCM3，具体如图 1-1 所示。

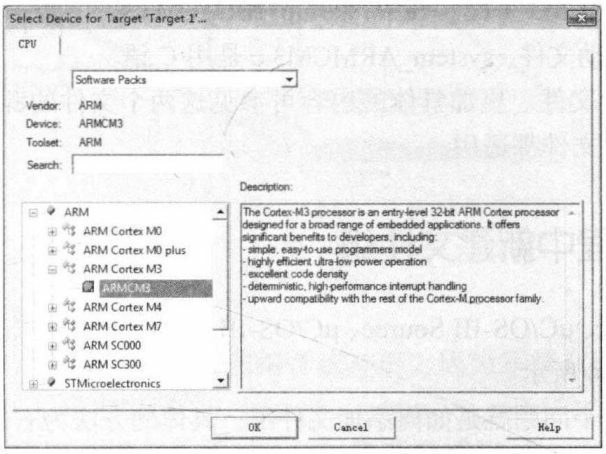


图 1-1　Select Device For Target 对话框

1.2.3　Manage Run-Time Environment

选择好处理器，单击 OK 按钮后会弹出 Manage Run-Time Environment 对话框。在 CMSIS 栏中选中 CORE，在 Device 栏中选中 Startup 文件，如图 1-2 所示。

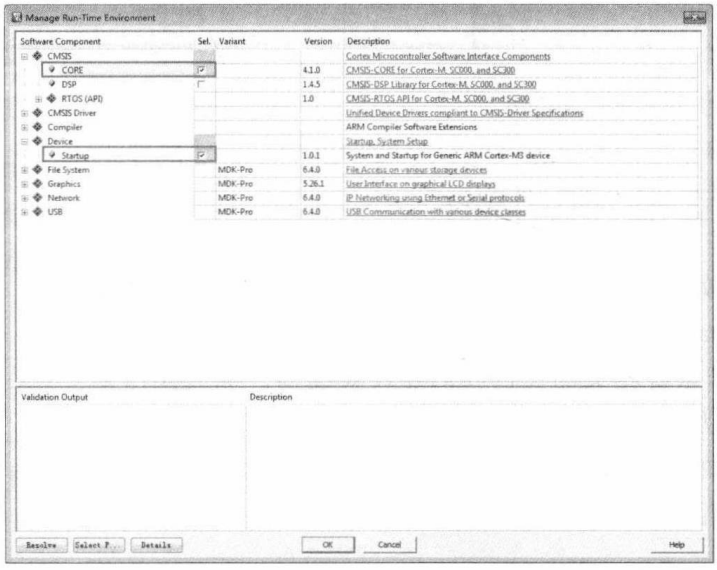


图 1-2　Manage Run-Time Environment 对话框

单击 OK 按钮，关闭 Manage Run-Time Environment 对话框之后，刚刚选择的 CORE（包含于 CMSIS）和 Startup（包含于 Device）这两个文件就会添加到工程组中，如图 1-3 所示⊖。

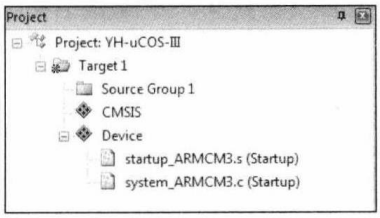


图 1-3　CORE 和 Startup 文件

其实这两个文件刚开始都是存放在 KEIL 的安装目录下，当配置 Manage Run-Time Environment 对话框之后，软件就会把选中的文件从 KEIL 的安装目录复制到工程目录 Project\RTE\Device\ARMCM3 下面。其中，startup_ARMCM3.s 是用汇编语言编写的启动文件，system_ARMCM3.c 是用 C 语言编写的与时钟相关的文件。更加具体的内容可参见这两个文件的源码。只要是 Cortex-M3 内核的单片机，这两个文件都适用。

1.3　在 KEIL 工程中新建文件组

在工程中添加 User、μC/OS-III Source、μC/OS-III Ports、μC/CPU、μC/LIB 和 Doc 文件组，用于管理文件，如图 1-4 所示。

对于新手，这里有个问题就是如何添加文件组，具体的方法为右击 Target1，在弹出的快捷菜单中选择 Add Group... 命令，如图 1-5 所示。需要多少个组，就按此方法操作多少次。

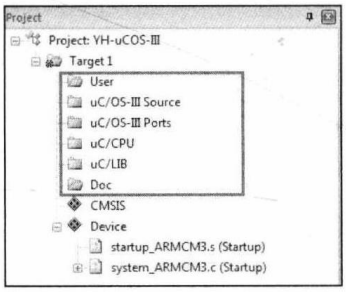


图 1-4　新添加的文件组

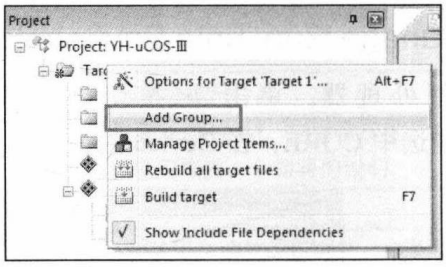


图 1-5　添加组

1.4　在 KEIL 工程中添加文件

在工程中添加好组之后，需要把本地工程中新建的文件添加到工程，具体为把 readme.txt 文件添加到 Doc 组，将 app.c 文件添加到 User 组，与操作系统相关的文件我们还没有编写，那么操作系统相关的组就暂时为空，如图 1-6 所示。

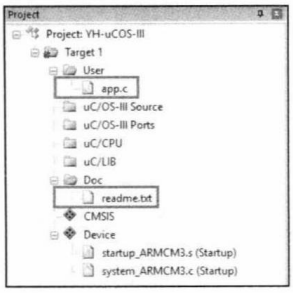


图 1-6　往组里面添加好的文件

将本地工程中的文件添加到工程组的具体的方法为双击相应的组，在弹出的对话框中找到要添加的文件，默认的文件类型是 C 文

⊖ 部分图片所示的文件或文件夹名称中，μ 或 / 无法显示，故将 μ 显示为 u，省略 /。——编辑注