uCOS 学习随笔 StepbyStep-1

——构建模板（基于 STM32 控制的第四代圆梦小车）

一、 序

基于第四代圆梦小车 —— FIRA 设计了一个使用 STM32 的控制板（详细介绍见项目中的说明： Introduction B - Hardware of the Smart Car.pdf ）。

既然硬件从 51 升级到 ARM，软件也应该相应升级，似乎不能再编写那种简单的轮询调度程序，也应该相应升级到基于操作系统编程。

按 STM32 的规模和性能，以及小车的控制需求，实时多任务操作系统 uCOSII 应该是不二的选择，不论从其性能和功能考虑，还是从学习角度考虑，uCOSII 都很适合。

首先，它是开源的，有丰富的资源。

其次，它是可靠的，符合正式的工业控制、产品设计需求。

小车所面对的是那些学习相关专业的大学生，作为他们学习的辅助工具，趣味性只是为了降低学习的枯燥性，不是目的。他们借助这个平台是为了积攒应付未来工作的能力，所以，学习内容的实用性是必须考虑的。

本人从未基于操作系统编写嵌入式程序。

开始使用 MCU 的时候，MCU 的内存太小，256 字节 RAM ，2K 字节 ROM，能勉强把程序装入就不错了，连 C 语言都不敢选择。

而且，那时好像也没有 RTOS（Real Time Operation System），或者是由于信息交流渠道匮乏，不知道有 RTOS。

既然我提供了这个平台，也借此机会尝试一下，和大家一起学习使用 uCOSII。（从单片机应用升级为嵌入式应用 ^_^）

二、Step1 想要得到什么？（需求分析）

第一步我想得到的是：

1) 如何建立一个基于 uCOSII 的编程环境（目录、文件组织）；

2) 如何基于 IDE（IAR 或 RvMDK）建立一个工程，能够产生可以运行的程序；

3) 得到一个“干净的”、可以作为模板的 uCOSII 程序组（Project）；

4) 通过上述过程初步理解在 uCOSII 下如何编写应用程序。

之所以要把“如何建立……”作为需求，而不是找一个现成的模板或示例程序修改、添加自己的功能，是因为看了许多这种程序，感觉“极不可靠”！因为程序中有太多的东西不知道为何而存在？不知道为何而被注释掉？似乎这些东西都像“定时炸弹”，早晚会给你的程序带来麻烦。

同时，也给自己理解程序的构成和运行机制带来困扰，既然是学习，就应该知其然、知其所以然，否则也谈不上“掌握”，更不敢在日后的工作中应用（如果是打工，也许还敢试试，如果是用自己的钱做产品、项目，我想你一定不敢用），如此则和做此事的初衷相悖了。

三、如何入手？

uCOS 的书有很多，也看了许多，但多数都是解析操作系统本身的，或者是如何移植，鲜有书籍、资料教你如何在操作系统下编程。

实际上，对于学习者，特别是初学者，更多需要的是学会如何在一个移植好的系统下编程，等到能基于操作系统实现自己的功能后，才会有心思去探究操作系统是如何在自己的 MCU 上运行的（移植），以及那些神秘的系统功能是如何实现的（了解系统函数及运行机制）。

而且这种探究也是有选择性的，首先是自己用到的功能才有兴趣去研究，否则如坠云雾。其次，取决于自己所扮演的角色，如果只是学习一下，那只需泛泛了解，有个定性的认识即

可。如果要用于产品，那可能要深究，吃透其源代码，以保证产品的可靠和高效。

所以，要想学习有效，学习的方式首先要“正确”。

在编程理念上，人们已经接受了“面向对象”的思维方式，并且承认了其优越之处。

可在学习方式上似乎并未接受，至少大多数书籍还是基于“过程”的，目前所倡导的“任务驱动”（或者称之为“项目驱动”）模式似乎未被响应，而所谓“任务驱动”我觉得从实质上讲类似于编程中的“面向对象”概念。

“面向对象”核心是将编程的关注点放在要实现的功能上，而非实现功能的方式。

“任务驱动”核心是将学习的关注点放在要完成的任务上，而非完成任务的技能。

“面向对象”的优点源于这种思维方式转变带来了逻辑关系的清晰，从而使程序易于理解，带来的所有好处我认为都源于“易于理解”，如可移植性、可靠性、便于多人合作等。

而“任务驱动”同样也是得益于“易于理解”！（使用 OS 编程也是为了“易于理解”）

以往的“教”和“学”都是先传授知识、技能，后让学生使用之。可痛苦在于，这些知识、技能是前人为了解决某些问题而创建的，准确的说应该是解决问题后抽象出来的。学生们却要先把抽象的“记住”、“理解”，“暂存”后再去找机会用。想象一下，学的过程会有多“枯燥”，更可悲的是，到需要用时，“暂存”的东西找不到了 ☹。

人通常对于明确目标的事情有较强的兴趣，而且目标越近，动力越大，越亢奋。对于不知为何而做的事很难投入，通常是三心二意应付了之。目前的大学多数是这个状况。

所谓“任务驱动”就是先明确目标，再去学习实现目的所需的知识和技能。这样学生在整个学习过程中都会主动去思考，不断斟酌正在学的东西可以怎样帮助自己实现目标。此时，你想让他走神都难。

所以，本学习过程尝试采用“任务驱动”方式。首先要确定一个合适的目标作为学习的素材，目标要可行、能提起兴趣，否则无异于没有。

我所选择的目标是用 STM32 去控制一个小车。小车控制涵盖了数字输出输入、模拟输入、定时器应用、通讯应用等，应该说嵌入式控制常用的知识均已包含。

因为电机驱动，转动检测、电机电流检测、通讯等需求同时存在，而且这些任务都有响应时间要求，uCOS 的实时多任务特征正好可以得到应用。

小车可以扩展，控制板有 I2C 接口，很多传感器都带。但那是在小车控制自如后的事了。

本系列文章就准备在这个基础上逐渐深入。

四、初步规划

从 03 年开始关注 uCOSII ，买了邵老师的书。真正开始看是 08 年，我的第二代小车推出 STM32 扩展版之后。晕晕乎乎将书看完，结果还是无从下手，半途而废。

这次重新启动是因为第四代小车没有设计 51 的控制器，为了演示只能编写 STM32 的控制程序。

为避免再次夭折，强迫自己不再为了销售而编写不带系统的示例程序。那样虽很快捷，借助于 ST 的库，初始化好硬件，直接使用原来基于 51 的 C 程序即可。

可如果那样，或许就没有动力和压力去“折磨”自己，尝试在 uCOSII 下编程了。不过这次我决定更章易辙，从应用入手，不再纠缠于系统本身。

1）买一个移植成功的学习板作为参考（我买的是“奋斗 STM32”），期望少走些弯路。

2）买了本《基于嵌入式实时操作系统的程序设计技术》，期待得到系统的指引。

3）从 uCOS 的官网（http://micrium.com/page/home）下载最新的 uCOS 源程序及资料。

4）以驱动小车控制板上唯一的 LED 为任务，自己构建编程环境和工程，作为日后深入的基础。

因为 51 我是用 Keil 的，所以 ARM 决定还是用 Keil，减小难度。

五、实施

5.1 准备工作

uCOS 官网上有移植好的 Cortex-M3 上的 uCOSII，程序包为：

Micrium-ST-uCOS-II-LCD-STM32.exe

下载安装后，里面有 uCOS 源程序、ST 库、基于 ST 学习板的示例程序、相应的说明。

看了若干资料，觉得最有价值莫过于从官网上下载的手册：

AN-1018 ： µC/OS-II and the ARM Cortex-M3 Processors

特别是其中的这张图：

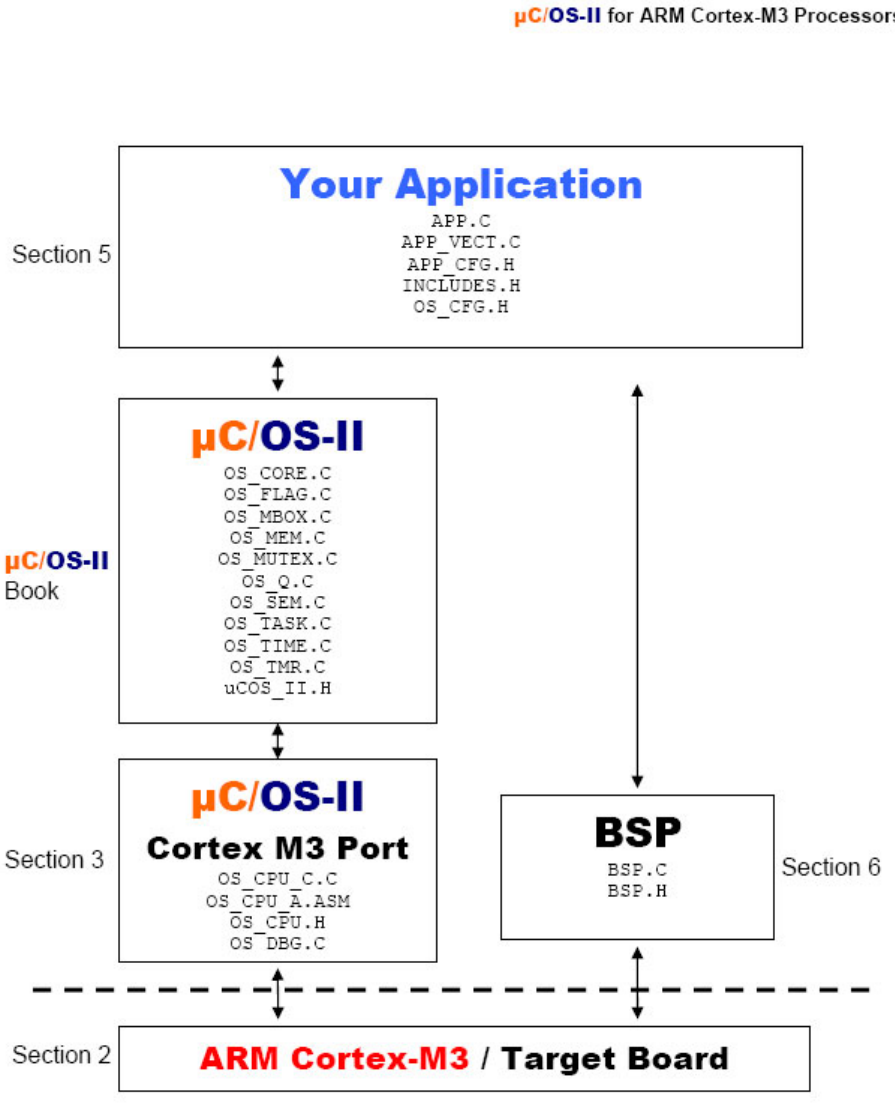
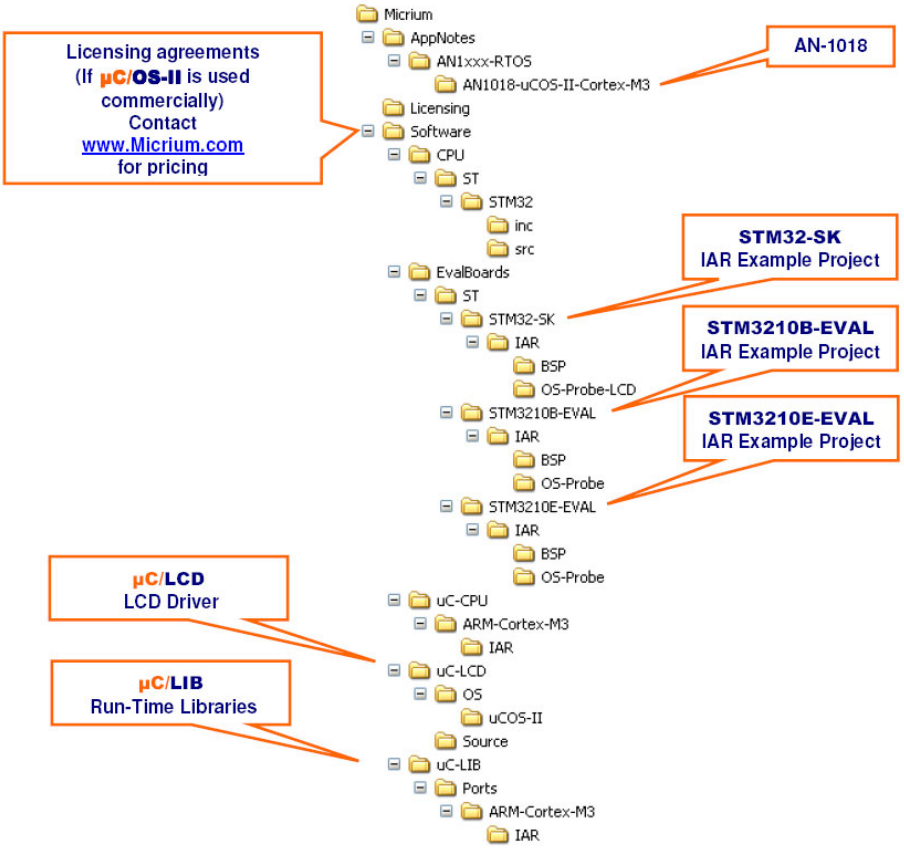
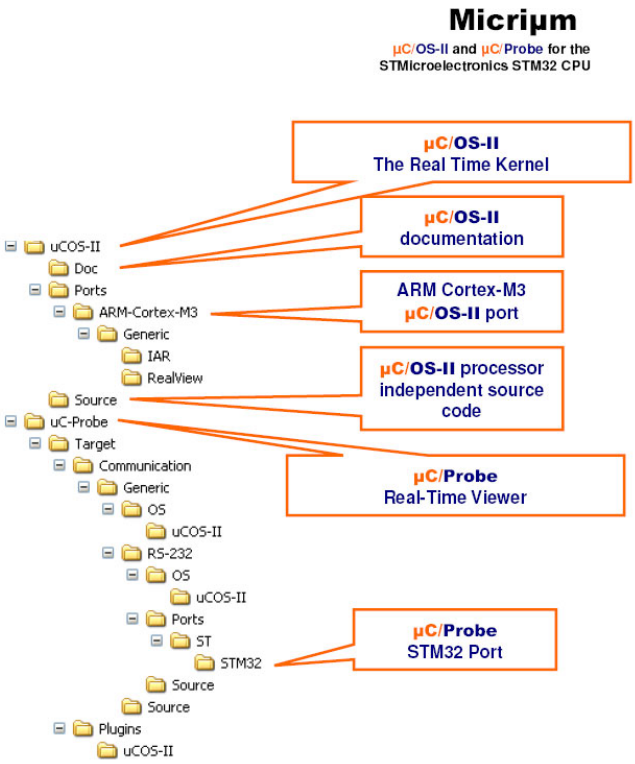


Figure 1-1, Relationship between modules.

对照 ReadMe 文件中的目录说明：





结合“程序关系图”、“目录文件说明”，浏览一下相关内容，那张“程序关系图”最好能映射在脑中，对理解、构建程序极为有益!

通过浏览，在脑中形成一个 uCOS 实现的框架，此时不必了解细节。

再把示例程序在 IDE 中打开编译、运行看看，因为我没有 ST 的学习板，就以奋斗 STM32 板的示例为参考。

这一步主要是为了验证 IDE 环境是否正确，因为示例的工程是正确的，如果此时有问题应出在 IDE 环境安装环节。如果开始就建立自己的工程，出现障碍则无法判断。

这次看这些文件似乎有些感觉，不知道是不是那些似懂非懂的阅读从量变到质变了。

不过回想学计算机的经历，似乎每次学习新的东西都要有个“从混沌的积累到顿悟”的过程，或许这就是计算机知识的特征：每个概念都建立在一大堆概念之上，而且都是多因素网状关联，需要同时“拥有”才能得到“答案”。

5.2 动手实施

示例程序还是比较“复杂”，因为它需要演示板上所有的功能。以 STM3210B-EVAL 为例，它上面有 LCD、按钮、JoyStick、M25 Flash 等外设，还支持了串口调试工具 uC/Probe。

第一步我要的是一个“干净”的模板，只需要驱动一个 LED，即一个 IO 口，因为这几乎是所有系统都会设计的。

通过这个模板，我希望理解 uCOS 下的程序是如何工作的。

至于调试用的 uC/Probe，相对于这一步所具备的水平，属于“奢侈品”，暂时还无法享用，所以暂不考虑，等日后程序功能多了，编程自如了，再“锦上添花”。

第一步：构建程序目录

首先根据自己的需要构建一个目录。

因为我希望分步实现目标，将每一步都保留，而不是做完后只剩最后一步的内容，这样别人参考就比较容易，不会像 uCOS 附带的示例程序，内容太多，难以消化。

但不希望每一步的目录中都包含 ST 和 uCOS 的库文件，这样一是文件太大，二是如果库文件要升级会很麻烦。

为此，构建了以下目录结构：

COMMON 目录是基本不用修改的，其中内容来自：

..\Micrium\Software\CPU\ST\STM32 → STM32LIB
..\Micrium\Software\uC-CPU\ → uC-CPU
..\Micrium\Software\uC-LIB\ → uC-LIB
.. \Micrium\Software\uCOS-II\ → uCOS-II
.. \Micrium\Software\uC-Probe\ → uC-Probe

虽然这一步不用 uC/Probe，但日后考虑会使用，故保留。

感到遗憾的是 Micrium 程序包中所附的 ST 库是 2.0 版的，本来打算自己更换为 3.0 版，但发现有些困难，初学乍练，不敢造次，就放弃了，留待日后升级吧。

学习目录目前只有 StepByStep-1 一个，分为三个子目录：

asp —— 用于存放应用程序，就是实现功能的程序。工程文件在这个目录中。

bsp —— 用于存放相应的硬件驱动程序，用到什么添加什么。

out —— 存放所有编译、链接产生的文件，交流时这个目录内容可以不拷贝。

这样每做一步都建立一个目录，逐步丰富内容。

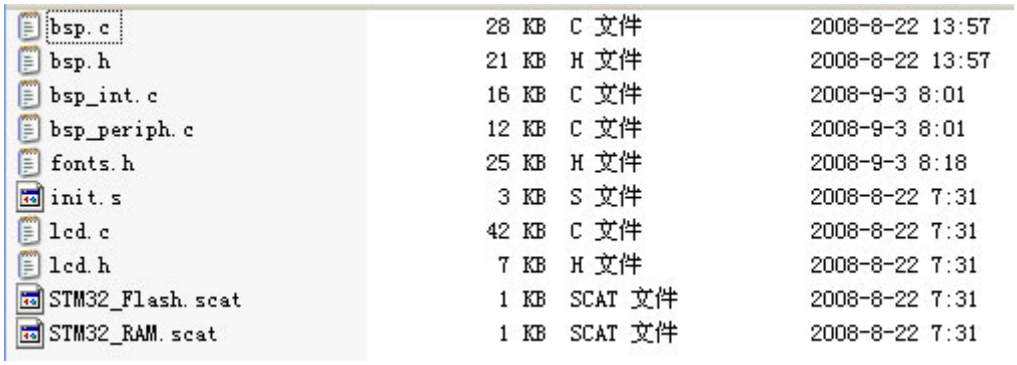
asp、bsp 目录中的文件均以上述“uCOS 程序关系图”中所描述的文件为基础。

第二步：构建自己的 bsp 文件

我参考的是这个目录：

.. \Micrium\Software\EvalBoards\ST\STM3210B-EVAL\RVMDK\BSP

它包含以下文件：



bsp.c	28 KB	C 文件	2008-8-22 13:57
bsp.h	21 KB	H 文件	2008-8-22 13:57
bsp_int.c	16 KB	C 文件	2008-9-3 8:01
bsp_periph.c	12 KB	C 文件	2008-9-3 8:01
fonts.h	25 KB	H 文件	2008-9-3 8:18
init.s	3 KB	S 文件	2008-8-22 7:31
lcd.c	42 KB	C 文件	2008-8-22 7:31
lcd.h	7 KB	H 文件	2008-8-22 7:31
STM32_Flash.scat	1 KB	SCAT 文件	2008-8-22 7:31
STM32_RAM.scat	1 KB	SCAT 文件	2008-8-22 7:31

我不用 LCD，所以去掉，bsp 文件如下：

bsp.c	11 KB	C 文件	2010-10-15 9:54
bsp.h	21 KB	H 文件	2010-10-15 0:04
bsp_int.c	16 KB	C 文件	2008-9-3 8:01
bsp_periph.c	12 KB	C 文件	2008-9-3 8:01
init.s	3 KB	S 文件	2008-8-22 7:31
STM32_Flash.scat	1 KB	SCAT 文件	2008-8-22 7:31
STM32_RAM.scat	1 KB	SCAT 文件	2008-8-22 7:31

注意：除删除了三个与 LCD 显示相关的文件外，bsp.h 和 bsp.c 也作了相应修改，主要是删除了不用的外设初始化和驱动程序，只保留了必须的驱动（详见程序清单）。

bsp_int.c 和中断初始化相关，因第一步不涉及中断，故暂不处理。

Bsp_periph.c 初始化各外设的时钟，因未吃透，也暂不处理。

Init.s 是启动代码，程序复位后的入口，因第一步无特殊要求，也没有能力变出花样，不去碰它。

两个 SCAT 文件是程序装载时的定位文件，STM32 在这上面有不少花样，可以将指定程序加载到指定位置等，未来如果要实现小车远程程序下载或许会用到，此时还一知半解。

按我的理解：

STM32_Flash.scat 文件是为了编译生成 FLASH 中运行的程序用的，STM32_RAM.scat 应该是编译生成 RAM 中运行的程序用的。刚从 51 上来，对在 RAM 中运行还不熟悉，故建立工程时用的是 FLASH 方式。

这个目录会随着外设的不断起用而丰富起来。

第三步：构建自己的应用程序

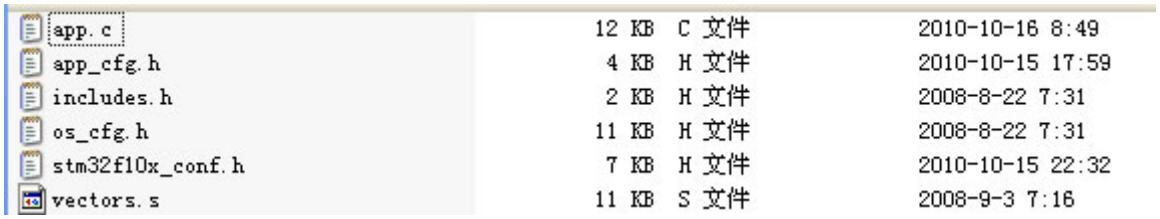
因为原来 STM3210B-Eval 板就有 LED 显示功能，而且是 4 个，我所做的就是删除所有不用的功能，将 4 个 LED 驱动改为一个即可。（注意：是删除，不是注释掉）

删除后的 app 相当“单纯”：

```
int  main (void)
{
    CPU_INT08U  os_err;


    BSP_IntDisAll();                                            /* Disable all ints until we are ready to accept them.  */

    OSInit();                                                   /* Initialize "uC/OS-II, The Real-Time Kernel".         */

    os_err = OSTaskCreateExt((void (*)(void *)) App_TaskStart,  /* Create the start task.                               */
                             (void           * ) 0,
                             (OS_STK         * )&App_TaskStartStk[APP_TASK_START_STK_SIZE - 1],
                             (INT8U            ) APP_TASK_START_PRIO,
                             (INT16U           ) APP_TASK_START_PRIO,
                             (OS_STK         * )&App_TaskStartStk[0],
                             (INT32U           ) APP_TASK_START_STK_SIZE,
                             (void           * )0,
                             (INT16U           )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));

#if (OS_TASK_NAME_SIZE >= 11)
    OSTaskNameSet(APP_TASK_START_PRIO, (CPU_INT08U *)"Start Task", &os_err);
#endif

    OSStart();                                                /* Start multitasking (i.e. give control to uC/OS-II).  */

    return (0);
}
```

```
static  void  App_TaskStart (void *p_arg)
{
  (void)p_arg;

  BSP_Init();                                                 /* Initialize BSP functions.                            */
  OS_CPU_SysTickInit();                                       /* Initialize the SysTick.                              */

#if (OS_TASK_STAT_EN > 0)
    OSStatInit();                                             /* Determine CPU capacity.                              */
#endif

  BSP_LED_Off();

  while (1)
  {                                                           /* Task body, always written as an infinite loop.       */
    BSP_LED_Toggle();
    OSTimeDlyHMSM(0, 0, 1, 0);                                /* 亮、暗各 1s.                                          */
  }
}
```

APP 目录下有如下文件：



app.c	12 KB	C 文件	2010-10-16 8:49
app_cfg.h	4 KB	H 文件	2010-10-15 17:59
includes.h	2 KB	H 文件	2008-8-22 7:31
os_cfg.h	11 KB	H 文件	2008-8-22 7:31
stm32f10x_conf.h	7 KB	H 文件	2010-10-15 22:32
vectors.s	11 KB	S 文件	2008-9-3 7:16

其中 app_cfg.h 也很“单纯”：

```
*********************************************************************************************************
*                                            TASK PRIORITIES
*********************************************************************************************************
*/

#define  APP_TASK_START_PRIO                              3    // 唯一的启动任务的优先级，设为系统之外的最高，后续的任务在这个任务中创建

/*
*********************************************************************************************************
*                                            TASK STACK SIZES
*                             Size of the task stacks (# of OS_STK entries)
*********************************************************************************************************
*/

#define  APP_TASK_START_STK_SIZE                        128    // 该任务的栈大小，如增加任务，则需继续定义
```

只有一个任务，确定其优先级和所用栈尺寸即可。

includes.h 里面有许多是系统需要的头文件，还没有吃透，暂不处理。

os_cfg.h 是 uCOSII 的配置文件，日后需要裁减系统时再琢磨，第一步不去惹它，以免系统罢工。

stm32f10x_conf.h 是 STM32 的配置文件，不启用的外设就注释掉，此处不用删除，因为恢复时太麻烦。

第一步应用部分只使用了一个 IO 口：PA3，所以从应用角度只需要打开 GPIO、GPIOA，但 AFIO 似乎系统用了，去掉后系统编译出错，只能保留。

按我的理解，一些基础的功能系统应该需要，如中断、时钟、定时器等，没想到可以将 SysTick 及所有 TIM 注释掉，不清楚系统是如何产生定时任务切换的，如果要深入系统，这就是一个值得探讨的点。（具体保留了那些外设详见程序）

vectors.s 中定义了 STM32 所有外设的中断向量，似乎没有必要去碰它。

至此，第一步所需要的源文件已经完成，下面要拿出来“遛遛”了。

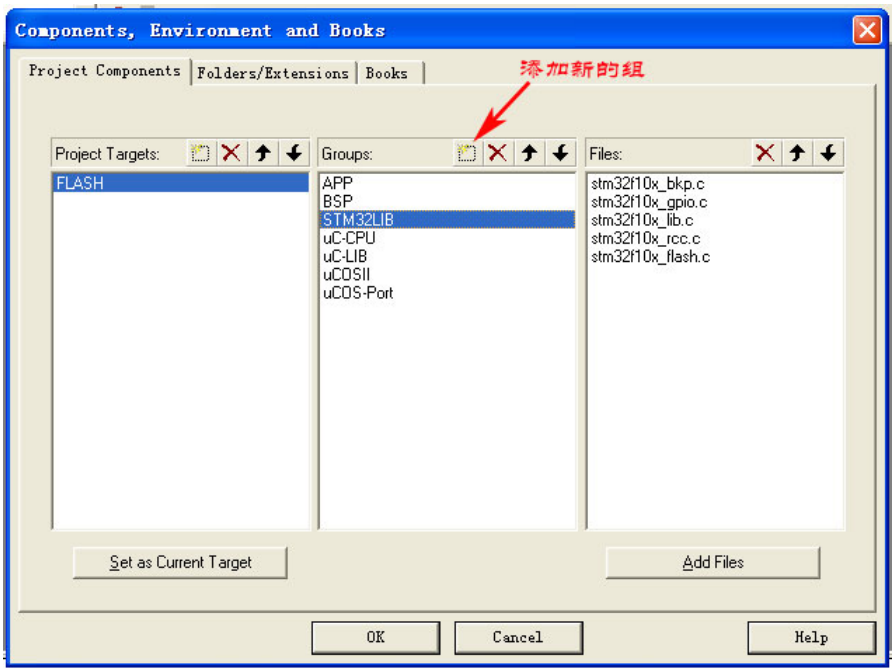
第四步：在 MDK 中构建工程

我所使用的是 MDK3.5 版本。构建过程如下:

1）在 Project 菜单中选择“New uVision Project”，创建一个新工程，放在 app 目录下；我所创建的工程文件名是:

YM4-STM32-Step1.Uv2

2）在 Project 菜单中选择“manage”，为工程设置工程目标名、文件组，并在每个文件组中添加对应的文件:



因为编译产生的文件是在 Flash 上运行的，所以目标名用“Flash”，我觉得这个纯粹是为了便于记忆、理解。

文件组是参考示例程序及自己的理解确定的。

确定组后，添加相应的文件，包括 C、S、asm（汇编）程序，asp、bsp 组的文件来自对应的目录。其它都是系统文件，具体如下:

STM32-LIB 文件添加自:（注：我是根据 STM32f10x-conf.h 确定要添加的文件的）

\uCOS-StepbyStep\COMMON\STM32LIB\src

uC-CPU 文件添加自：

\uCOS-StepbyStep\COMMON\uC-CPU\ARM-Cortex-M3\RealView

uC-LIB 文件添加自：（注：因第一步未用，故没有添加文件）

\uCOS-StepbyStep\COMMON\uC-LIB
\uCOS-StepbyStep\COMMON\uC-LIB\Ports\ARM-Cortex-M3\RealView

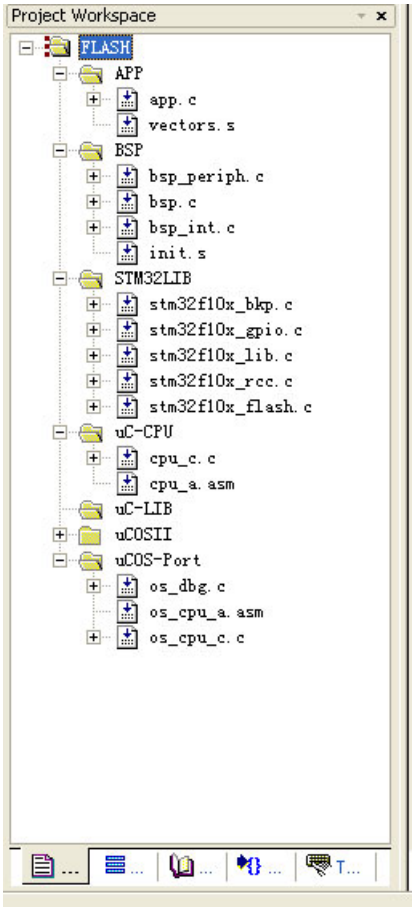
uCOSII 文件添加自：

\uCOS-StepbyStep\COMMON\uCOS-II\Source

uCOS-Port 文件添加自：

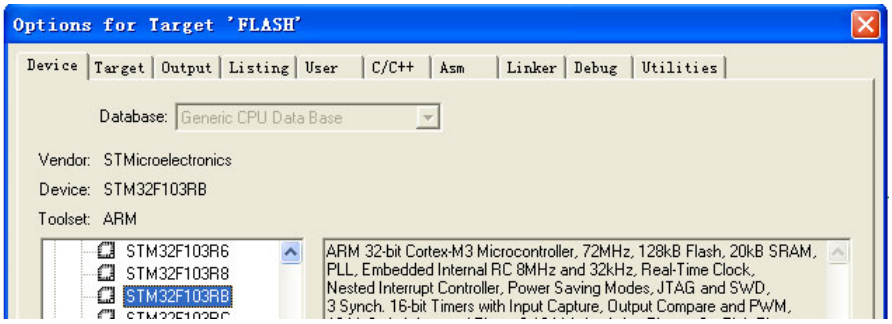
\uCOS-StepbyStep\COMMON\uCOS-II\Ports\ARM-Cortex-M3\Generic\RealView

完成上述步骤后，在 IDE 的 Project Workspace 窗口显示如下：



3）鼠标在工程目标名“Flash”上，点右键菜单，选择“Option for Taget Flash”，开始配置工程选项。

打开后的主界面如下：

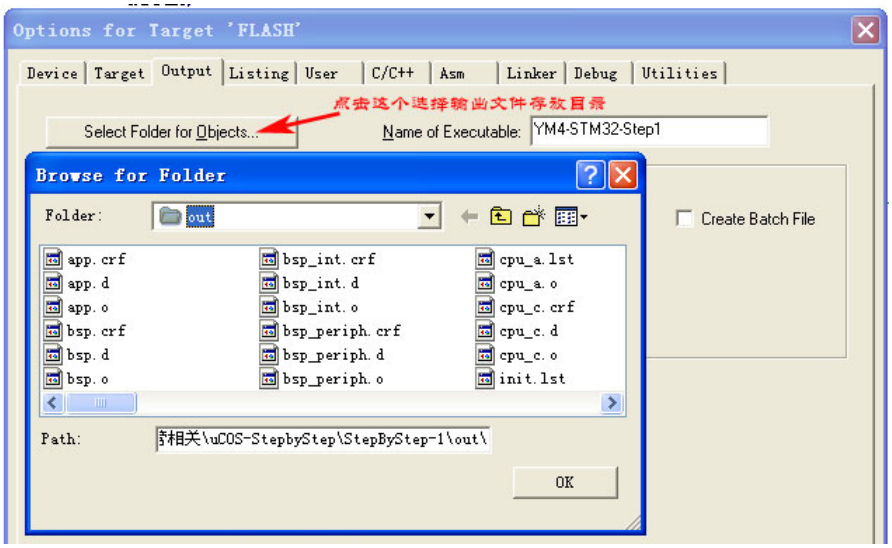


依次选择：

Device： ST 的 STM32F103RB

Target：默认值

Output：



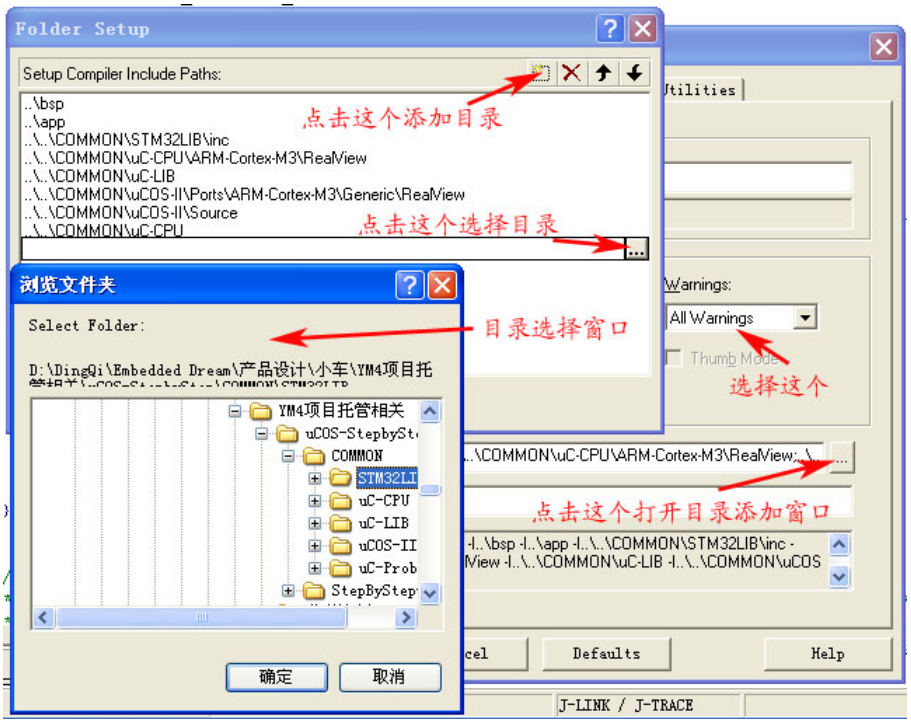
Listing：同 Output 一样选择输出文件存放目录，也放在 out 目录下。

User：默认（还不明白有什么用处 ☹）

C/C++：这个选项中有很多暂时没有吃透，第一步必须做的是：首先选择编译的的告警提示，建议用 “All Warning”，因为初学，多注意提示能帮助减少程序中的隐患。

其次就是配置 include 目录，将 COMMON 和 StepbyStep 目录下所有存在 *.h 的目录全部添加进去。

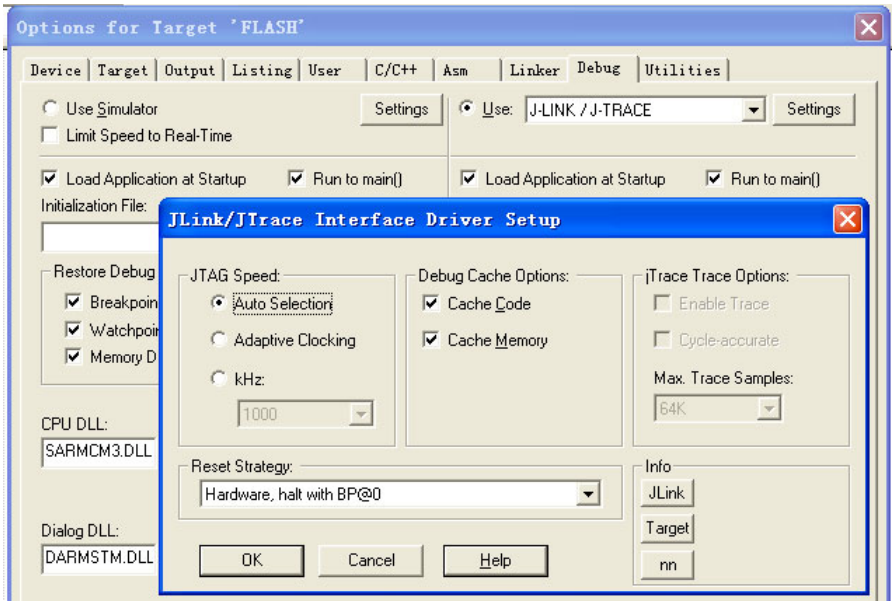
如下所示：



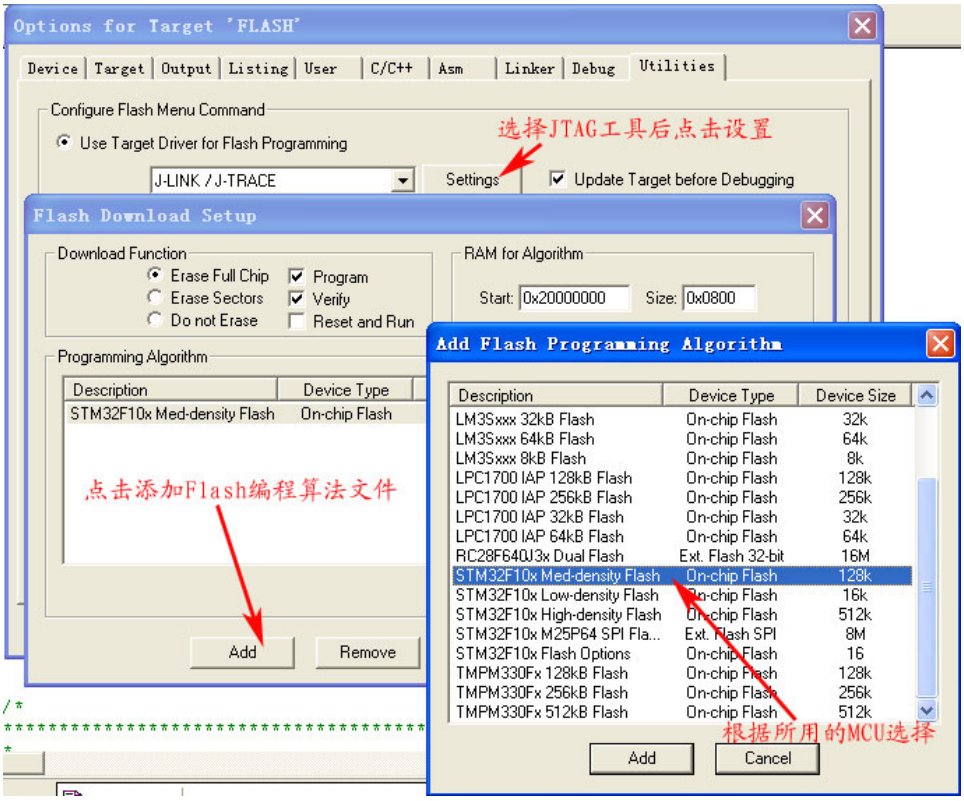
Asm：默认

Linker：

Debug：按如下内容配置



Utilities：这一步主要配置 STM32 的 FLASH 编程算法，如下：



至此，工程全部建立完毕，可以编译了。

第五步：编译、下载、调试

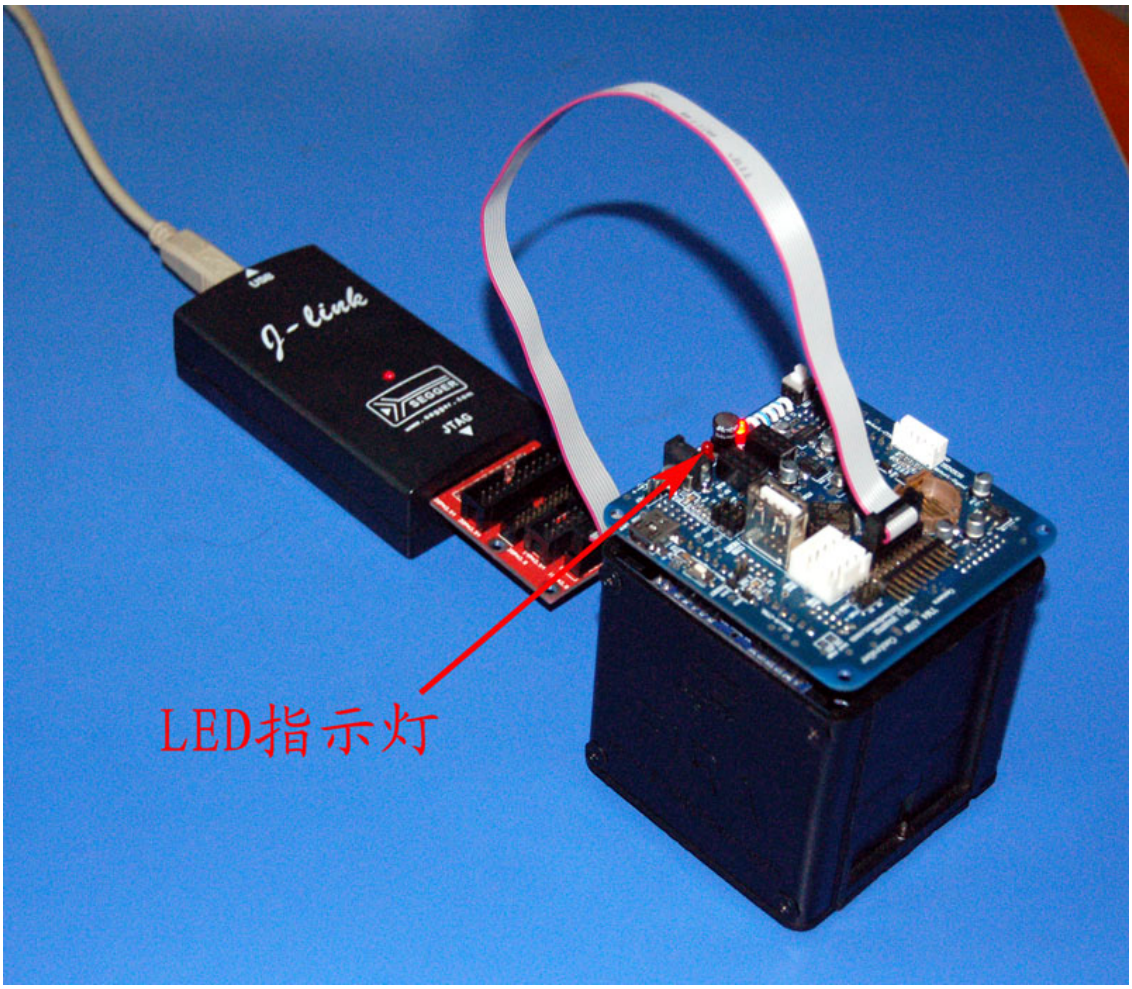
这部分操作没有什么特别之处，用过 Keil51 就会，即使不熟悉，参考 MDK 的手册即可。

JTAG 工具用的是市场上最常见的 JLink V8。小车控制板较小，选用的是 IDC10 接口，需要一个转接插头。

编译时建议先单独编译自己写的程序，此处是：asp.c　bsp.c 。这样便于出错，因为编译产生的提示信息相对少些。

排除错误后再进行“Build Taget”。

小车调试方式如下：



很幸运，uCOSII 下的第一个程序如愿运行了 ☺

六、总结

这是我的第一个 uCOSII 应用程序，虽然简单，但它帮助我初步理解了如何在 uCOS 下编写程序，uCOS 程序大概（也就达到了这个水平）是如何运行的。

有了这个基础，就可以逐步深入，以小车的控制为需求，依次启用 STM32 上的外设，编写相应的 BSP，并根据需要编写相应的应用程序。

通过这一步，自我感觉再看 uCOS 源程序时容易理解多了，以前看时是被动的接受书中的概念，而这次是先有疑问，再去找答案。

比如说，如果对照圆梦小车二代的 Step-1 程序，用延时方式控制 LED 闪烁，那就无法再处理其它任务，MCU 全给延时循环占用了。

为解此惑，我便看了 uCOS 的延时源程序，理解它是如何实现延时的。我主要看了两个与定时相关的函数：

OSTimeDly（INT16U ticks）

OSTimeDlyHMSM（INT8U hours, INT8U minutes, INT8U seconds, INT16U milli）

因为有明确的需求，所以容易理解多了，同时还发现了一个小“秘密”：用后一个函数不能提高延时的精度，虽然它提供了 ms 参数，看似可以精确到 ms，可如果你真想用它达到 ms 分辨率的延时一定会失望的，因为它是靠调用前一个函数实现的。

由此得出结论：

1）用系统延时函数最高分辨率就是系统的 ticks，通常 tick 为 10ms，太快了系统开销太大，所以用此方法实现 1ms 分辨率的延时不可行，虽说 Step1 的 tick 是 1ms，但后面会改为 10ms。

2）后一个函数最大的用途应该是长延时，用前一个函数最长只能 65535 个 ticks。

3）如果延时小于 65535 ticks，最好用前一个，可降低开销。

4）因为这个延时是利用任务控制块中的延时计数实现的，所以精度不高，如果要准确延时，比如形成周期，最好还是直接用定时器中断配合消息机制实现。

为理解 OSTimeDly，顺带看了时钟节拍函数：OSTimeTick（void），初步了解了任务切换的过程。

此文属于随笔，注重叙述过程和感悟，不想深入探讨系统实现，等小车的控制初具雏形后再考虑是否需要对应写一系列文章，专门探讨每一步所涉及的系统函数是如何工作的。

读者如果有兴趣可以自己先尝试一下，我觉得 Step1 应该了解的是：

1） uCOSII 的系统定时是如何产生的？使用的 STM32 什么硬件资源？

2） 看看用到的几个 OS 函数源代码，它们的功能是什么？如何实现？ 如：

BSP_IntDisAll()
OSInit()
OSTaskCreate()
OSTaskCreateExt()
OSTaskNameSet()
OSStart()

此外还有 BSP 中的一些函数

3） 消化任务控制块 TCB 中各个变量的作用，对理解 OS 函数很有帮助。

Step-1 到此结束，准备着手根据需求构建任务了。

但愿本文能对那些和我一样曾经蠢蠢欲动多次又无疾而终的 uCOS 学习者有帮助，大家共同交流、进步！

2010 年 10 月 20 日星期三

参考资料：

1、《嵌入式实时操作系统 uC/OS-II（第二版）》邵贝贝 等译 ISBN7-81077-290-2

2、《基于嵌入式实时操作系统的程序设计技术》周航慈 吴光文著 ISBN978-7-81077-941-8

3、《Cortex-M3 + uCOS-II 嵌入式系统开发入门与应用》陈瑶等著 ISBN978-7-115-23105-5

4、《uC/OS-II 标准教程》杨宗德 张兵 著 ISBN978-7-115-20442-4

5、uC/OS-II 官方网站资料

6、奋斗 STM32 MINI 学习板资料