# 移指RT-Thread到STM32

## RT-Thread的下载与安装

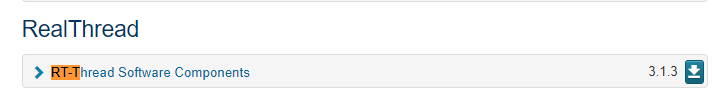
工程模板直接使用这个:

D:\fireClass\野火开源图书\[野火]《STM32库开发实战指南》\F103\_指南者\标准库\_配套代码\12-GPIO输出—使用固件库点亮LED灯

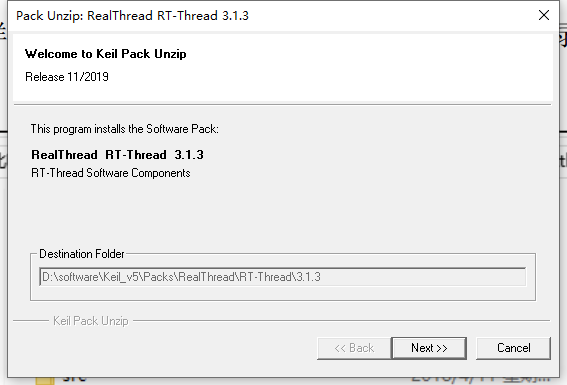
RT-Thread Master的源码的Github地址:

<https://github.com/RT-Thread/rt-thread>

RT-Thread Nano版本是Master版本的简化版, 在RT-Thread官网中并没有, 而实作为一个Package放在了KEIL官网, 可以在<https://www.keil.com/dd2/Pack/>中找到.



下载好后双击安装





## RT-Thread的代码文件夹组织

|  |  |  |
| --- | --- | --- |
| 文件夹 | 子文件夹 | 描述 |
| RT-Thread\3.1.3 | bsp | 板级支持包 |
| components/finsh | RT-Thread 组件 |
| include | 头文件 |
| include/libc |
| libcpu/arm/cortex-m0 | 与处理器相关的接口文件 |
| libcpu/arm/cortex-m3 |
| libcpu/arm/cortex-m4 |
| libcpu/arm/cortex-m7 |
| src | RT-Thread 内核源码 |

RT-Thread/3.1.3/bsp下的rtconfig.h拷贝到user目录.

这个配置头文件可以用于对内核的功能进行裁剪.

RT-Thread/3.1.3/bsp下的board.c 文件到 user 文件夹

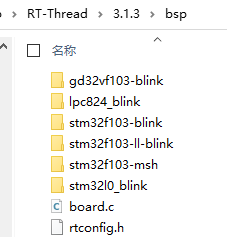
### (1) bsp 文件夹

存放板级支持包, nano版本只支持几款开发板的驱动.

用户在使用RT-Thread 的时候, 用户只需要修改 board.c 和 rtconfig.h 这两个文件的内容即可, 其它

文件我们不需要改动.

如果为了减小工程的大小, bsp 文件夹下面除了 board.c 和 rtconfig.h这两个文件要保留外, 其它的统统可以删除.



### (2) components文件夹

第三方的软件都是组件.

master版本有 gui, fatfs, lwip 和finsh等.

nano版本只有finsh, 它通过串口打印的方式来输出各种信息, 方便我们调试程序.

### (3) include文件夹

就是一些头文件了

### (4) libcpu 文件夹

与单片机CPU内核密切相关的文件, 提供了通往硬件的接口.

编写这些接口文件的过程就叫做移指.

RT-Thread 3.1.3的nano版本目前包含arm和risc-v架构的少数几款内核.

arm的内核有cortex-m0, m3, m4 和 m7等.

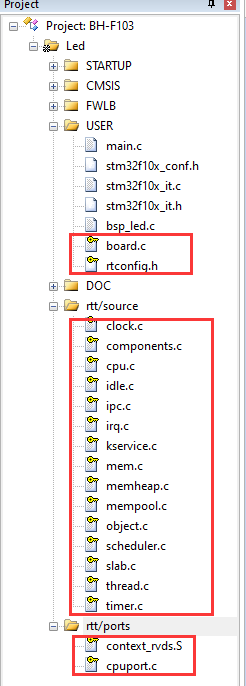
### (5) src文件夹

就是RT-Thread这个操作系统的源码了.

## 工添加RT-Thread源码

在开发环境里面新建 rtt/source 和 rtt/ports 两个组文件夹

添加以下文件



board.c和rtconfig.h的只读属性要去掉.

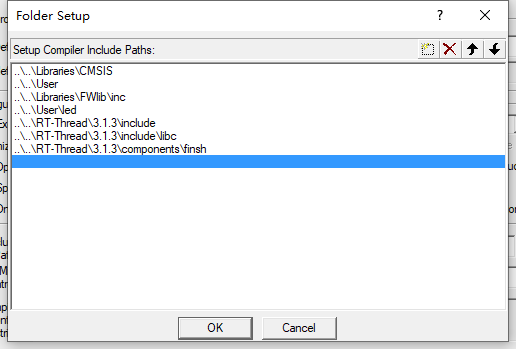
将这4个文件夹添加到头文件路径.

User

RT-Thread\3.1.3\components\finsh

RT-Thread\3.1.3\include

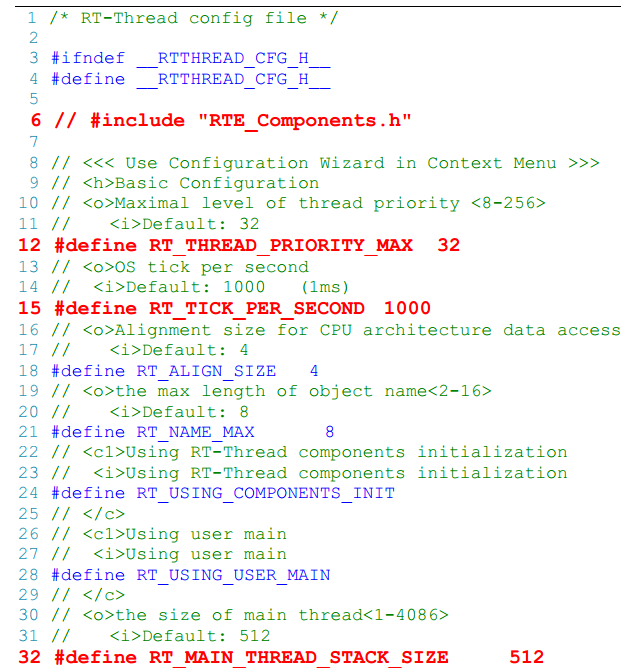
RT-Thread\3.1.3\include\libc



## 修改 rtconfig.h

头文件RTE\_Components.h 是在 MDK 中添加 RT-Thead Package时由MDK自动生成的, 目前我们没有使用 MDK中自带的RT-Thread的Package, 所以这个头文件不存在, 如果包含了该头文件, 编译的时候会报错, 等下修改rtconfig.h的时候需要注释掉该头文件.

只修改4个地方.

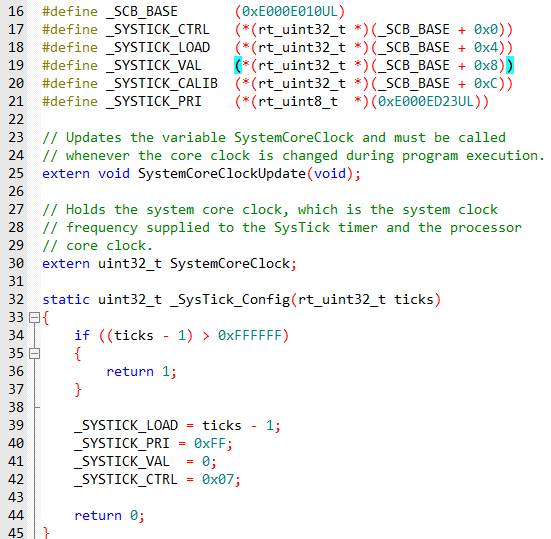


## 修改 board.c

rthw.h 是处理器相关, rtthread.h与内核相关

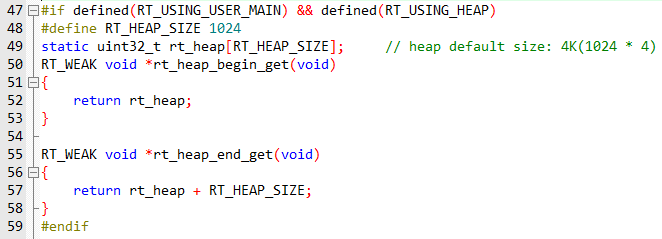


这部分代码可以直接注释掉, 直接使用固件库提供的函数.



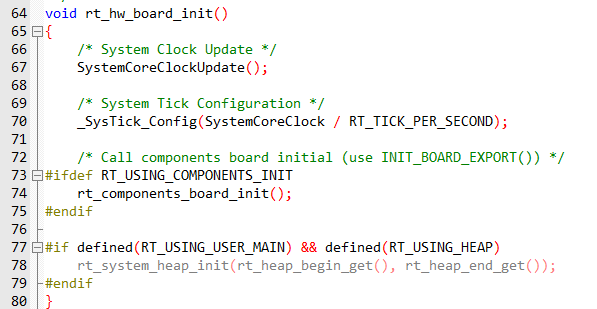
RT\_USING\_USER\_MAIN 和RT\_USING\_HEAP 这两个宏， 表示 RT-Thread 里面创建内核对象时使用动态内存分配方案。

堆大小默认为4K. RT\_WEAK表示弱定义.



rt\_hw\_board\_init()函数

它是用来初始化开发板硬件的，比如时钟，比如串口等.



SystemCoreClockUpdate()来自于

D:\STM32-rtthread\rtthread-app\Libraries\CMSIS\system\_stm32f10x.c

是由固件库提供的, 用于更行系统时钟.

\_SysTick\_Config(SystemCoreClock / RT\_TICK\_PER\_SECOND);

初始化系统定时器 SysTick

这里配置为 1000, 即1 秒钟内 SysTick 会中断 1000 次，即中断周期为 1ms.

等会用固件库SysTick\_Config()来代替.

rt\_components\_board\_init();

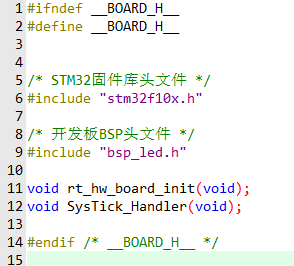
初始化LED，串口， LCD 等.

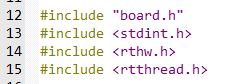
rt\_system\_heap\_init(rt\_heap\_begin\_get(), rt\_heap\_end\_get());

这部分是 RT-Thread 为开发板组件提供的一个初始化函数

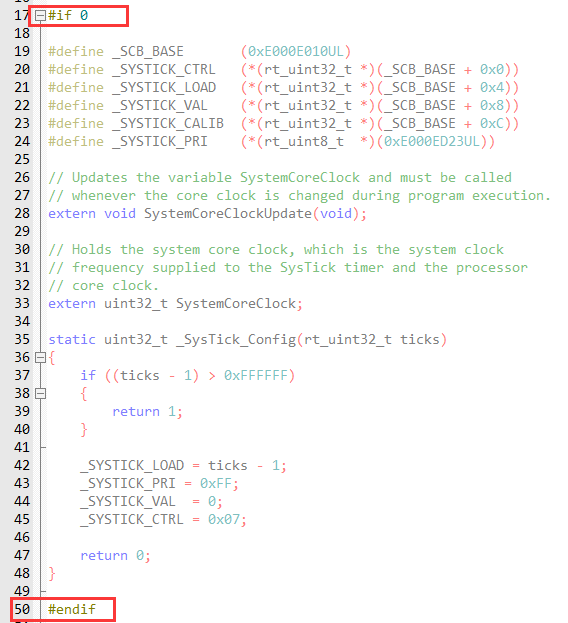
由于RT\_USING\_HEAP没有开启, 所以不使用.

需要添加board.h头文件





void rt\_hw\_board\_init()改为使用固件库的时钟初始化函数.

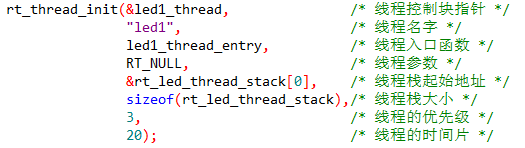




# 创建线程

|  |
| --- |
| #include <rtthread.h>  rt\_err\_t rt\_thread\_init(struct rt\_thread \*thread,  const char \*name,  void (\*entry)(void \*parameter),  void \*parameter,  void \*stack\_start,  rt\_uint32\_t stack\_size,  rt\_uint8\_t priority,  rt\_uint32\_t tick); |

实例



优先级数字越小优先级越大.

线程的时间片单位是系统节拍, 表示线程一次调度能够运行的最大事件长度.

|  |
| --- |
| #include <rtthread.h>  rt\_err\_t rt\_thread\_startup(rt\_thread\_t thread);  启动线程, 传入参数为线程控制块指针. |

总结创建线程的步骤:

(1) 定义线程控制块

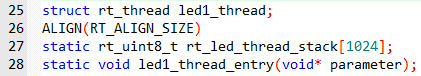
(2) 定义线程栈空间

(3) 定义线程函数

(4) 调用rt\_thread\_init()初始化线程.

(5) 调用rt\_thread\_startup()启动线程.

以上步骤对应如下代码:





注意, 初始化LED的代码在rt\_hw\_board\_init()里面实现了, 该函数在mian()函数之前已经调用.

调用链:

$Sub$$main(void);

rtthread\_startup();

rt\_hw\_board\_init();

$Sub$$是指对main程序进行打补丁.