RT-THREAD NANO 配置

RT-THREAD 文档中心

WWW.RT-THREAD.ORG

Thursday 26th December, 2019

目录

目录 **i**

1 头文件 1

2 基础配置 1

3 内核调试功能配置 2

4 钩子函数配置 2

5 软件定时器配置 2

6 IPC 配置 2

7 内存配置 3

8 FinSH 控制台配置 3

9 常见问题 4

RT-Thread Nano 的配置在 rtconfig.h 中进行，通过开关宏定义来使能或关闭某些功能，接下来对该配置文件中的宏定义进行说明。

1 头文件

头文件 RTE_Components.h 仅由 Keil MDK 工程生成，其中仅定义了一个打开 FinSH 组件的宏 RTE_USING_FINSH。

```
#if defined (__CC_ARM) || (__CLANG_ARM)
#include "RTE_Components.h"               /* 用来开关 FinSH 组件，仅 MDK 会产生该文件 */

#if defined(RTE_USING_FINSH)
#define RT_USING_FINSH
#endif //RTE_USING_FINSH

#endif //(__CC_ARM) || (__CLANG_ARM)
```

非 Keil MDK 则不需要该头文件，若需打开 FinSH 组件，可直接在 rtconfig.h 中手动定义 RT_USING_FINSH 打开 FinSH 组件。

2 基础配置

1、设置系统最大优先级，可设置范围 8 到 256，默认值 8，可修改。

```
#define RT_THREAD_PRIORITY_MAX  8
```

2、设置 RT-Thread 操作系统节拍，表示多少 tick 每秒，如默认值为 100 ，表示一个时钟节拍（os tick）长度为 10ms。常用值为 100 或 1000。时钟节拍率越快，系统的额外开销就越大。

```
#define RT_TICK_PER_SECOND  100
```

3、字节对齐时设定对齐的字节个数，默认 4，常使用 ALIGN(RT_ALIGN_SIZE) 进行字节对齐。

```
#define RT_ALIGN_SIZE   4
```

4、设置对象名称的最大长度，默认 8 个字符，一般无需修改。

```
#define RT_NAME_MAX    8
```

5、设置使用组件自动初始化功能，默认需要使用，开启该宏则可以使用自动初始化功能。

```
#define RT_USING_COMPONENTS_INIT
```

6、开启 RT_USING_USER_MAIN 宏，则打开 user_main 功能，默认需要开启，这样才能调用 RT-Thread 的启动代码；main 线程的栈大小默认为 256，可修改。

```
#define RT_USING_USER_MAIN
#define RT_MAIN_THREAD_STACK_SIZE     256
```

3 内核调试功能配置

定义 RT_DEBUG 宏则开启 debug 模式，默认关闭。若开启系统调试，则可以打印系统 LOG 日志。

```
//#define RT_DEBUG                  // 关闭 debug

#define RT_DEBUG_INIT 0             // 启用组件初始化调试配置，设置为 1 则会打印自动初始化的函数名称

//#define RT_USING_OVERFLOW_CHECK   // 关闭栈溢出检查
```

4 钩子函数配置

设置是否使用钩子函数，默认关闭。

```
//#define RT_USING_HOOK                         // 是否 开启系统钩子功能

//#define RT_USING_IDLE_HOOK                    // 是否 开启空闲线程钩子功能
```

5 软件定时器配置

设置是否启用软件定时器，以及相关参数的配置，默认关闭。

```
#define RT_USING_TIMER_SOFT       0             // 关闭软件定时器功能，为 1 则打开
#if RT_USING_TIMER_SOFT == 0
#undef RT_USING_TIMER_SOFT
#endif

#define RT_TIMER_THREAD_PRIO        4           // 设置软件定时器线程的优先级，默认为 4

#define RT_TIMER_THREAD_STACK_SIZE  512         // 设置软件定时器线程的栈大小，默认为 512 字节
```

6 IPC 配置

系统支持的 IPC 有：信号量、互斥量、事件集、邮箱、消息队列。通过定义相应的宏打开或关闭该 IPC 的使用。

```
#define RT_USING_SEMAPHORE          // 设置是否使用 信号量

//#define RT_USING_MUTEX            // 设置是否使用 互斥量

//#define RT_USING_EVENT            // 设置是否使用 事件集

#define RT_USING_MAILBOX            // 设置是否使用  邮箱

//#define RT_USING_MESSAGEQUEUE     // 设置是否使用 消息队列
```

7 内存配置

RT-Thread 内存管理包含：内存池、内存堆、小内存算法。通过开启相应的宏定义使用相应的功能。

```
//#define RT_USING_MEMPOOL      // 是否使用 内存池

//#define RT_USING_HEAP         // 是否使用 内存堆

#define RT_USING_SMALL_MEM      // 是否使用 小内存管理

//#define RT_USING_TINY_SIZE    // 是否使用 小体积的算法，牵扯到 rt_memset、
    rt_memcpy 所产生的体积
```

8 FinSH 控制台配置

定义 RT_USING_CONSOLE 则开启控制台功能，失能该宏则关闭控制台，不能实现打印；修改 RT_CONSOLEBUF_SIZE 可配置控制台缓冲大小。

```
#define RT_USING_CONSOLE                        // 控制台宏开关
#define RT_CONSOLEBUF_SIZE          128         // 设置控制台数据 buf 大小，默认 128
     byte
```

FinSH 组件的使用通过定义 RT_USING_FINSH 开启，开启后可对 FinSH 组件相关的参数进行配置修改，FINSH_THREAD_STACK_SIZE 的值默认较小，请根据实际情况改大。

```
#if defined (RT_USING_FINSH)                    // 开关 FinSH 组件

    #define FINSH_USING_MSH                     // 使用 FinSH 组件 MSH 模式
    #define FINSH_USING_MSH_ONLY                // 仅使用 MSH 模式

    #define __FINSH_THREAD_PRIORITY     5       // 设置 FinSH 组件优先级，配置该值后
        通过下面的公式进行计算
    #define FINSH_THREAD_PRIORITY       (RT_THREAD_PRIORITY_MAX / 8 *
        __FINSH_THREAD_PRIORITY + 1)
```

```
    #define FINSH_THREAD_STACK_SIZE     512     // 设置 FinSH 线程栈大小，范围
        1-4096

    #define FINSH_HISTORY_LINES         1       // 设置 FinSH 组件记录历史命令个数，
        值范围 1-32

    #define FINSH_USING_SYMTAB                  // 使用符号表，需要打开，默认打开

#endif
```

9 常见问题

Q：移植完成之后出现 hard fault。

A：在默认情况下，系统配置的各种线程栈大小均较小，若不能正常运行，很有可能是栈不够用，可将栈值调大。例如 main 线程栈大小默认为 256，在实际使用时，main 中可能加入其它代码导致栈不够用的情况；FinSH 组件的线程 tshell，默认栈 512 也比较小，在使用时可以调大。