

32 位微控制器

HC32F4A0 系列的 RT-Thread 系统移植

本产品支持芯片系列如下

F 系列	HC32F4A0
-------------	-----------------

目 录

1	摘要	3
2	RT-Thread 系统的移植.....	3
2.1	获取 RT-Thread Nano 源码	3
2.2	获取 HC32F4A0 裸机工程.....	3
2.3	往裸机工程添加 RT-Thread 源码	3
2.3.1	复制 RT-Thread 源码到裸机工程目录	3
2.3.2	复制 board.c 和 rtconfig.h 文件到用户文件夹	4
2.3.3	添加 RT-Thread 到工程文件	4
2.3.4	工程配置	6
2.4	添加 driver 文件到工程	6
2.5	修改 rtconfig.h	7
2.6	修改 board.c.....	8
2.6.1	添加 HC32F4A0 相关函数	8
2.6.2	rt_hw_board_init()函数	9
2.6.3	重映射串口到 rt_kprintf 函数	10
2.7	修改 main.c.....	12
2.8	下载验证	14
3	总结	14
4	版本信息 & 联系方式	15

1 摘要

RT-Thread，全称是 Real Time-Thread，是一个嵌入式实时多线程操作系统。目前 32 位 MCU 是 RT-Thread 的主要运行平台。

本篇应用笔记主要介绍了 HC32F4A0 系列基于 RT-Thread Master V4.0.2 的移植。

2 RT-Thread 系统的移植

2.1 获取 RT-Thread Nano 源码

RT-Thread 有 Nano 和 Master 两个版本，其中 Nano 是 Master 的精简版，去掉了一些组件和各种开发板的 BSP，保留了核心功能。

本篇应用笔记使用的是 RT-Thread Nano V3.1.3，源码可以从 RT-Thread 官网获取。

2.2 获取 HC32F4A0 裸机工程

HC32F4A0 的裸机工程可以从华大半导体网站获取，本篇应用笔记使用的是设备驱动库 hc32f4a0_ddl_Rev1.2.0 里 Template 样例工程，路径如下：

HC32F4A0_SDK\驱动库及样例\hc32f4a0_ddl_Rev1.2.0\example\template

2.3 往裸机工程添加 RT-Thread 源码

2.3.1 复制 RT-Thread 源码到裸机工程目录

为方便移植，将 RT-Thread 源码复制到 HC32F4A0 DDL 目录下。具体如图 1 所示：

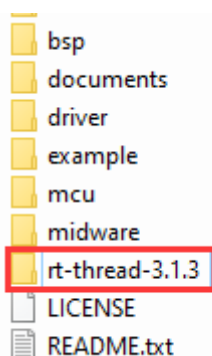


图 1 文件结构图

rt-thread-3.1.3 文件夹下是 RT-Thread Nano 的全部内容，该文件夹下的具体内容如表 1 所示。

表 1 RT-Thread 文件内容

文件夹	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	头文件
	lipcpu/arm/common	与 arm 处理器相关的公共接口文件
	lipcpu/arm/cortex-m4	与 arm cortex-m4 相关的接口文件
	src	RT-Thread 内核源码

2.3.2 复制 board.c 和 rtconfig.h 文件到用户文件夹

将 rt-thread-3.1.3/bsp 文件夹下的 board.c 和 rtconfig 文件复制到用户文件夹，示例中是 source 文件夹，路径如下：

hc32f4a0 ddl Rev1.2.0 \example\template\source

source 文件夹内容如图 2 所示，其中 board.c 和 rtconfig 文件需要修改。

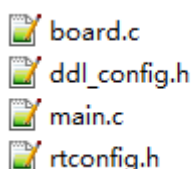


图 2 source 文件夹内容

2.3.3 添加 RT-Thread 到工程文件

在前两节内容里将 RT-Thread 的源码放到了本地工程目录下，这里我们以 IAR 工程为例将源码添加进工程组。源码添加完毕后，具体见图 3。

添加 kernel group 存放 rt-thread-3.1.3/src 文件夹内容，及 RT-Thread 内核文件。添加 mcu group 存放 rt-thread-3.1.3/lipcpu/arm 文件夹内容，及处理器相关接口文件。添加 board.c 到 source group。

注意:

- HC32F4A0 是 cortex-M4 的内核，添加 cortex-M4 内核相关文件，在路径 rt-thread-3.1.3/lipcpu/arm/cortex-m4 中，IAR 工程，则添加启动文件 cortex_iar.S。

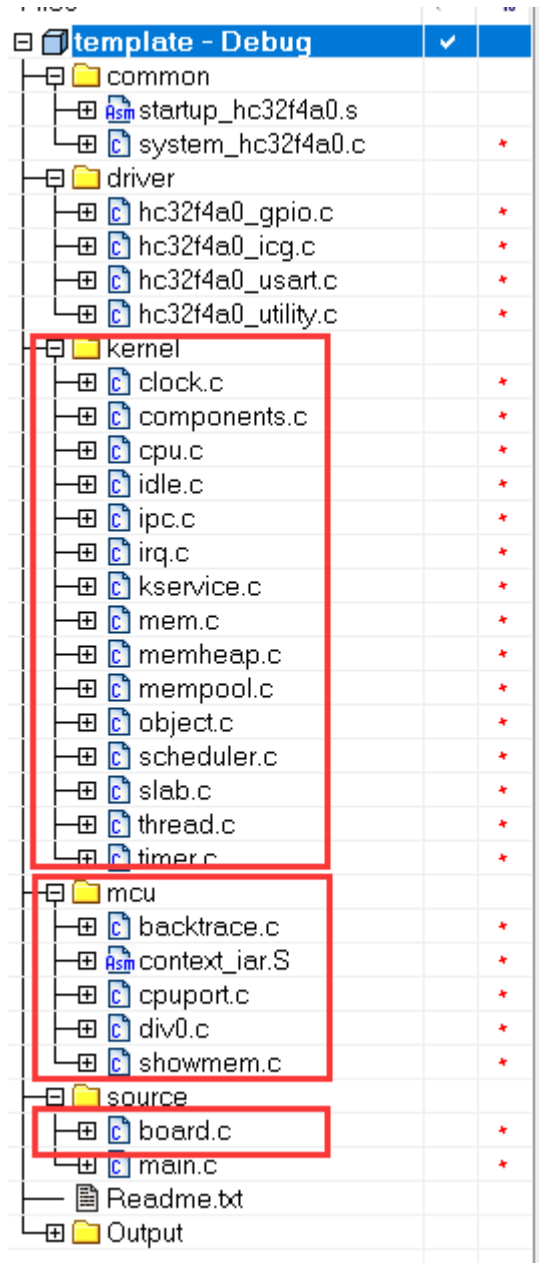


图 3 添加 rt-thread 源码到工程示意图

2.3.4 工程配置

RT-Thread 的源码已经添加到开发环境的组文件夹下，编译的时候需要为这些源文件指定头文件的路径，否则编译会报错。

具体添加过程如图 4 所示。

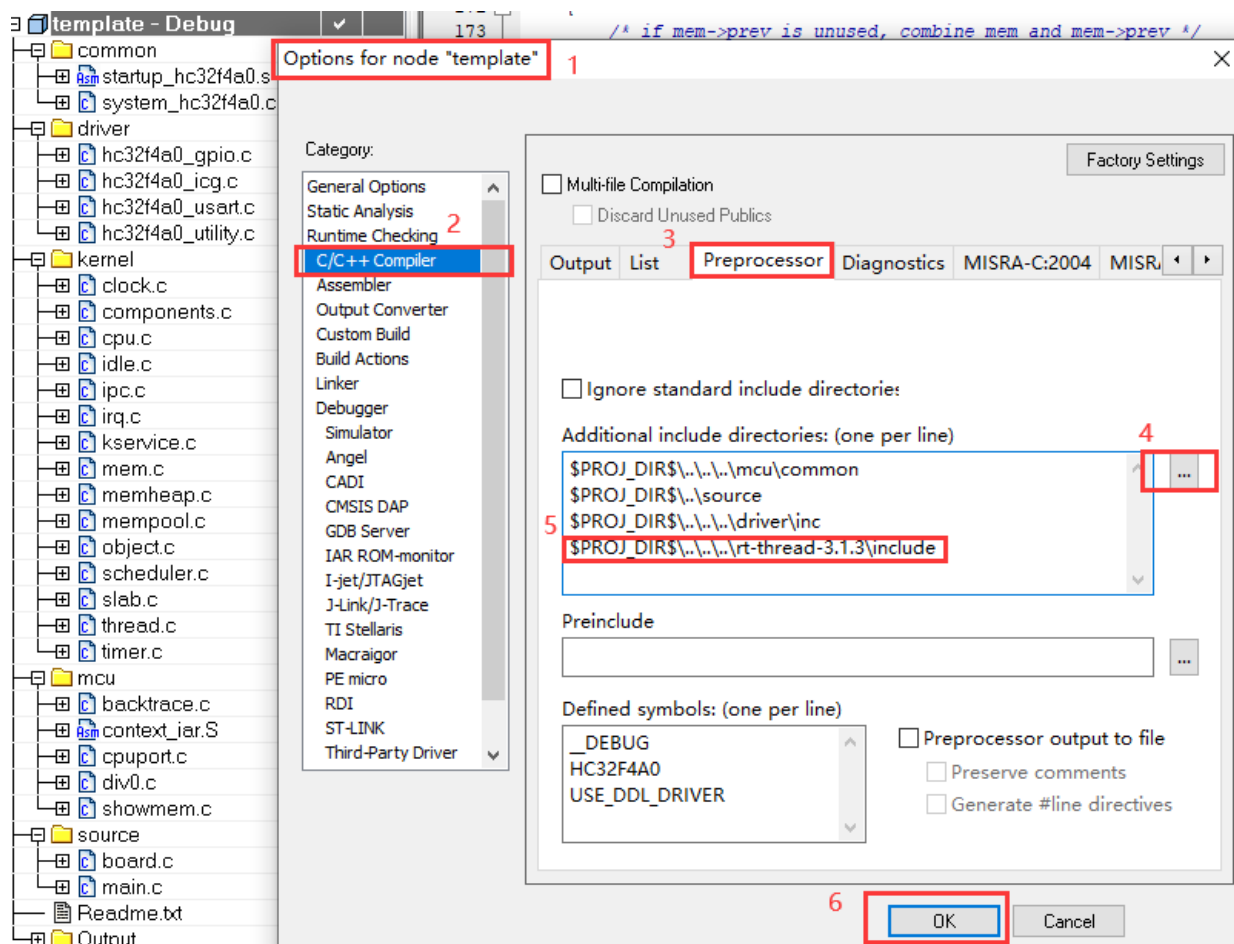


图 4 添加 RT-Thread 头文件示意图

2.4 添加 driver 文件到工程

本篇应用笔记中会用到 gpio 和 usart 相关内容，因此，添加 hc32f4a0_gpio.c 和 hc32f4a0_usart.c 到工程 driver group，如图 3 所示。

为使 gpio 和 usart 能正常使用，还需修改 source 文件夹下的 ddl_config.h 文件，打开 gpio 和 usart 对应的宏定义，修改为如下代码。

```
#define DDL_GPIO_ENABLE          (DDL_ON)
#define DDL_USART_ENABLE        (DDL_ON)
```

2.5 修改 rtconfig.h

rtconfig.h 文件对裁剪整个 RT-Thread 所需的功能的宏均做了定义，有些宏定义被使能，有些宏定义被失能，表 2 简单介绍一些宏定义功能。

表 2 rtconfig.h 宏定义

宏定义	描述
RT_THREAD_PRIORITY_MAX	RT-Thread 支持多少个优先级，取值范围 8~256
RT_TICK_PER_SECOND	操作系统每秒钟有多少 tick。
RT_USING_COMPONENTS_INIT	使用 RT-Thread 组件初始化，默认使能
RT_USING_USER_MAIN	使用用户 main 函数，默认打开
RT_MAIN_THREAD_STACK_SIZE	main 线程栈大小，取值范围为 1~4086
RT_USING_HEAP	是否使用堆

rtconfig.h 头文件的内容中，修改了 RT_THREAD_PRIORITY_MAX、RT_TICK_PER_SECOND 和 RT_MAIN_THREAD_STACK_SIZE 三个宏的大小，如下代码所示。

```
#define RT_THREAD_PRIORITY_MAX    8
#define RT_TICK_PER_SECOND        1000
#define RT_MAIN_THREAD_STACK_SIZE 512
```

2.6 修改 board.c

board.c 文件中存放的是与硬件相关的初始化函数。整个文件中包含 `rt_hw_board_init()` 函数和 `SysTick_Handler()` 函数。

2.6.1 添加 HC32F4A0 相关函数

在 board.c 文件中会调用 HC32F4A0 库函数，需添加 HC32F4A0 寄存器解除保护 `Peripheral_WE()` 和写保护 `Peripheral_WP()` 函数，以及相应的头文件。

```
#include "hc32_ddl.h"
```

`Peripheral_WE()` 函数代码如下所示：

```
static void Peripheral_WE(void)
{
    /* Unlock GPIO register: PSPCR, PCCR, PINAER, PCRxy, PFSRxy */
    GPIO_Unlock();
    /* Unlock PWC register: FCG0 */
    //PWC_FCG0_Unlock();
    /* Unlock PWC, CLK, PVD registers, @ref PWC_REG_Write_Unlock_Code for details */
    //PWC_Unlock(PWC_UNLOCK_CODE_0 | PWC_UNLOCK_CODE_1);
    /* Unlock SRAM register: WTCR */
    //SRAM_WTCR_Unlock();
    /* Unlock SRAM register: CKCR */
    // SRAM_CKCR_Unlock();
    /* Unlock all EFM registers */
    //EFM_Unlock();
    /* Unlock EFM register: FWMC */
    // EFM_FWMC_Unlock();
    /* Unlock EFM OTP write protect registers */
    // EFM_OTP_WP_Unlock();
}
```

`Peripheral_WP()` 函数代码如下所示：

```
static void Peripheral_WP(void)
{
    /* Lock GPIO register: PSPCR, PCCR, PINAER, PCRxy, PFSRxy */
    GPIO_Lock();
    /* Lock PWC register: FCG0 */
    // PWC_FCG0_Lock();
    /* Lock PWC, CLK, PVD registers, @ref PWC_REG_Write_Unlock_Code for details */
    //PWC_Lock(PWC_UNLOCK_CODE_0 | PWC_UNLOCK_CODE_1);
    /* Lock SRAM register: WTCR */
    // SRAM_WTCR_Lock();
    /* Lock SRAM register: CKCR */
    // SRAM_CKCR_Lock();
    /* Lock all EFM registers */
    // EFM_Lock();
    /* Lock EFM OTP write protect registers */
    // EFM_OTP_WP_Lock();
}
```



```
/* Lock EFM register: FWMC */  
// EFM_FWMC_Lock();  
}
```

这两个函数由用户根据实际使用情况决定是否注释相应代码。本篇应用笔记中的工程只涉及到 GPIO 模块。

2.6.2 rt_hw_board_init()函数

RT-Thread 启动时会调用该函数，该函数用来初始化开发板硬件，包括时钟、串口、GPIO 等，具体内容由用户决定。

本篇应用笔记中的工程会用到 LED 灯和串口打印，因此在该函数中添加 Peripheral_WE()函数、LED_Init()函数和 DDL_PrintfInit()函数，如图 5 所示。

系统时钟使用 MCU 上电默认时钟 MRC，8MHz。用户若有需求可参考 DDL 中时钟样例在 board.c 文件中添加 SystemClock_Config ()函数，并在该函数中 Peripheral_WE()函数后调用 SystemClock_Config ()函数。

```
void rt_hw_board_init()  
{  
    Peripheral_WE();  
  
    /* System Clock Update */  
    SystemCoreClockUpdate();  
  
    /* System Tick Configuration */  
    _SysTick_Config(SystemCoreClock / RT_TICK_PER_SECOND);  
  
    LED_Init();  
    DDL_PrintfInit();  
  
    /* Call components board initial (use INIT_BOARD_EXPORT()) */  
#ifndef RT_USING_COMPONENTS_INIT  
    rt_components_board_init();  
#endif  
  
#if defined(RT_USING_USER_MAIN) && defined(RT_USING_HEAP)  
    rt_system_heap_init(rt_heap_begin_get(), rt_heap_end_get());  
#endif  
}
```

图 5 rt_hw_board_init()函数

DDL_PrintfInit()函数已在 hc32f4a0_utility.c 中实现，LED_Init()函数代码如下：

```
static void LED_Init(void)
{
    stc_gpio_init_t stcGpioInit;
    /* LED initialize */
    (void)GPIO_StructInit(&stcGpioInit);
    (void)GPIO_Init(GPIO_PORT_C, GPIO_PIN_09, &stcGpioInit);

    /* "Turn off" LED before set to output */
    GPIO_ResetPins(GPIO_PORT_C, GPIO_PIN_09);

    /* Output enable */
    GPIO_OE(GPIO_PORT_C, GPIO_PIN_09, Enable);
}
```

2.6.3 重映射串口到 rt_kprintf 函数

在 RT-Thread 中，有一个打印函数 rt_kprintf()供用户使用，方便在调试的时候输出各种信息。

如果要使用 rt_kprintf()函数，必须将控制台（USB、串口、CAN 等设备）重映射到 rt_kprintf()。本节介绍如何将串口重映射到 rt_kprintf()函数。

rt_kprintf()函数在 kservice.c 中实现，是属于内核服务类的函数。代码内容如图 6 所示。目前我们不使用 RT-Thread 的设备驱动，通过 rt_kprintf()输出的内容则由 rt_hw_console_output()函数处理。这个函数在 board.c 中实现，具体代码如下。

```
#define DEBUG_USARTx (M4_USART1)
```

至此，board.c 修改完成。

```
void rt_hw_console_output(const char *str)
{
    rt_enter_critical();

    while (*str!='\0')
    {
        if (*str=='\n')
        {
            USART_SendData(DEBUG_USARTx, '\r');
            while (USART_GetStatus(DEBUG_USARTx, USART_FLAG_TXE) == Reset);
        }

        USART_SendData(DEBUG_USARTx, *str++);
        while (USART_GetStatus(DEBUG_USARTx, USART_FLAG_TXE) == Reset);
    }

    rt_exit_critical();
}
```

```

/**
 * This function will print a formatted string on system console
 *
 * @param fmt the format
 */
void rt_kprintf(const char *fmt, ...)
{
    va_list args;
    rt_size_t length;
    static char rt_log_buf[RT_CONSOLEBUF_SIZE];

    va_start(args, fmt);
    /* the return value of vsnprintf is the number of bytes that would be
     * written to buffer had if the size of the buffer been sufficiently
     * large excluding the terminating null byte. If the output string
     * would be larger than the rt_log_buf, we have to adjust the output
     * length. */
    length = rt_vsnprintf(rt_log_buf, sizeof(rt_log_buf) - 1, fmt, args);
    if (length > RT_CONSOLEBUF_SIZE - 1)
        length = RT_CONSOLEBUF_SIZE - 1;
#ifdef RT_USING_DEVICE
    if (_console_device == RT_NULL)
    {
        rt_hw_console_output(rt_log_buf);
    }
    else
    {
        rt_uint16_t old_flag = _console_device->open_flag;

        _console_device->open_flag |= RT_DEVICE_FLAG_STREAM;
        rt_device_write(_console_device, 0, rt_log_buf, length);
        _console_device->open_flag = old_flag;
    }
#else
    rt_hw_console_output(rt_log_buf);
#endif
    va_end(args);
}

```

图 6 rt_kprintf()函数代码

2.7 修改 main.c

原工程里 main 函数是一个 while 循环的空函数。这里创建两个线程，线程 1 是 LED 灯以 1S 的频率闪烁，并打印输出 LED_ON 和 LED_OFF 的信息，优先级为 3。线程 2 是打印线程，每隔 4S 打印输出 “print_thread is running!” 信息，优先级为 2。

详细代码如下：

```

/*****
* Include files
*****/
#include "hc32_ddl.h"
#include "rtthread.h"

/*****
* Local pre-processor symbols/macros ('#define')
*****/
#define LED_PORT      (GPIO_PORT_C)
#define LED_PIN       (GPIO_PIN_09)

#define LED_ON()      GPIO_SetPins(LED_PORT, LED_PIN)
#define LED_OFF()     GPIO_ResetPins(LED_PORT, LED_PIN)

#define DELAY_MS      (RT_TICK_PER_SECOND) /* 1s */

/*****
* Local variable definitions ('static')
*****/
static rt_thread_t led_thread = RT_NULL;
static rt_thread_t print_thread = RT_NULL;

/*****
* Function implementation - global ('extern') and local ('static')
*****/
static void led_thread_entry(void* parameter)
{
    rt_kprintf("led_thread is running! \n");
    while (1)
    {
        LED_ON();
        rt_thread_delay(DELAY_MS);
        rt_kprintf("LED ON! \n");

        LED_OFF();
        rt_thread_delay(DELAY_MS);
        rt_kprintf("LED OFF! \n");
    }
}

static void print_thread_entry(void* parameter)
{
    while(1)
    {
        rt_kprintf("print_thread is running! \n");
    }
}

```

```
        rt_thread_delay(DELAY_MS*4);
    }
}

/**
 * @brief Main function of template project
 * @param None
 * @retval int32_t return value, if needed
 */
int32_t main(void)
{
    rt_kprintf("Os start!\n");

    led_thread = rt_thread_create("led",
                                   led_thread_entry,
                                   RT_NULL,
                                   512,
                                   3,
                                   20);

    if (led_thread != RT_NULL)
        rt_thread_startup(led_thread);
    else
        return -1;

    print_thread = rt_thread_create("print",
                                     print_thread_entry,
                                     RT_NULL,
                                     512,
                                     2,
                                     20);

    if (print_thread != RT_NULL)
        rt_thread_startup(print_thread);
    else
        return -1;
}
```

至此，main.c 修改完成。用户可根据自己需求创建线程。

2.8 下载验证

编译工程并下载到目标板，打开串口调试助手，打印信息如图 7 所示，符合预期。

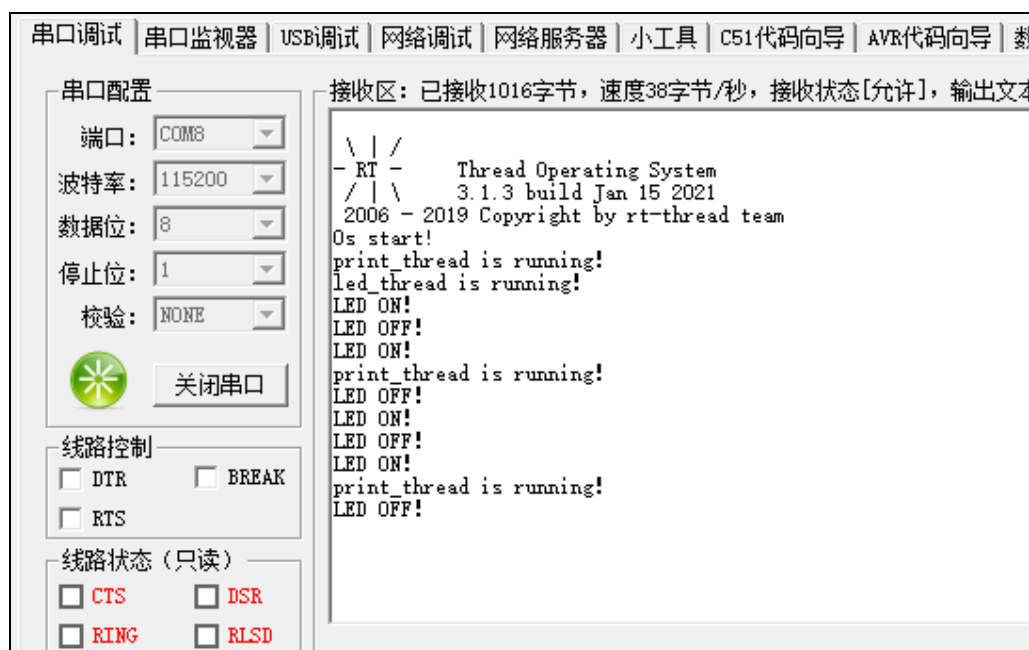


图 7 线程输出信息

3 总结

本篇应用笔记介绍了 HC32F4A0 的 RT-Thread 系统的移植，在实际开发中，用户可根据具体应用场景按需配置驱动。

4 版本信息 & 联系方式

日期	版本	修改记录
2021/7/27	Rev1.0	初版发布



如果您在购买与使用过程中有任何意见或建议，请随时与我们联系。

Email: mcu@hdsc.com.cn

网址: www.hdsc.com.cn

通信地址: 上海市浦东新区中科路 1867 号 A 座 10 层

邮编: 201203

