

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	BSP 框架简介	3
3	HC32F4A0 系列的 BSP 制作	4
3.1	复制 BSP 通用模板	5
3.2	BOARD 配置	7
3.2.1	外设引脚配置	7
3.2.2	系统时钟配置	7
3.2.3	RAM 配置	8
3.3	KConfig 选项修改	9
3.4	工程构建相关文件修改	10
3.4.1	链接脚本 Link 修改	10
3.4.2	构建脚本 SConscript 修改	12
3.4.3	rtconfig.py 文件修改	14
3.4.4	工程模板修改	14
3.5	重新生成工程	16
3.5.1	重新生成 rtconfig.h 文件	16
3.5.2	重新生成 MDK/IAR 工程	17
4	样例代码	18
4.1	代码介绍	18
5	总结	18
6	版本信息 & 联系方式	19

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 v4.0.2 的移植。

2 BSP 框架简介

BSP 框架结构如图 1 所示：

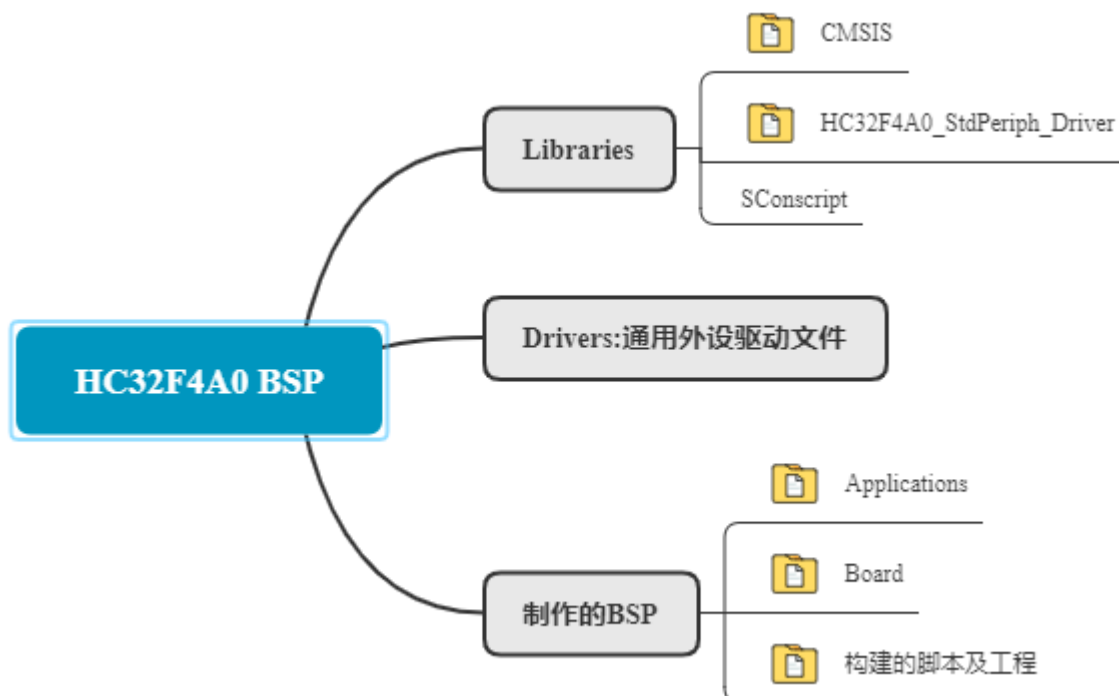


图 1 BSP 框架结构

HC32F4A0 系列的 BSP 分为三部分：开发板库文件、外设驱动文件以及制作的 BSP。本篇文档中制作的 BSP 是基于 HC32F4A0SITB EV 板，可作为 HC32F4A0 系列的 BSP 模板。

3 HC32F4A0 系列的 BSP 制作

本章节主要介绍 HC32F4A0 系列的 BSP 制作过程，主要分为以下五个步骤：

- 复制 BSP 通用模板
- BOARD 配置
- 修改 KConfig 文件
- 修改工程构建相关文件
- 重新生成工程

3.1 复制 BSP 通用模板

如果 RT-Thread BSP 中已经实现了芯片型号类似的 BSP，可以直接复制这个 BSP 做相应修改，如果没有则复制其他 BSP。

具体文件结构如图 2 所示。

rt_thread > bsp > hc32f4a0			
名称	类型	大小	
applications	文件夹		
board	文件夹		
build	文件夹		
drivers	文件夹		
Libraries	文件夹		
.config	XML Configuration File	16 KB	
Kconfig	文件	1 KB	
project.ewd	EWD 文件	88 KB	用户IAR工程文件
project.ewp	EWP 文件	63 KB	
project.eww	IAR IDE Workspace	1 KB	
project.uvoptx	UVOPTX 文件	42 KB	用户MDK工程文件
project.uvprojx	磳ision5 Project	35 KB	
README.md	MD 文件	4 KB	
rtconfig.h	Notepad++ Document	4 KB	
rtconfig.py	Notepad++ Document	4 KB	
SConscript	文件	1 KB	Scons构建文件
SConstruct	文件	2 KB	
template.ewd	EWD 文件	88 KB	IAR工程模板文件
template.ewp	EWP 文件	52 KB	
template.eww	IAR IDE Workspace	1 KB	
template.uvoptx	UVOPTX 文件	5 KB	MDK工程模板文件
template.uvprojx	磳ision5 Project	16 KB	

图 2 BSP 文件结构

在 BSP 制作过程中，需修改 board 文件夹内的配置文件，表 1 总结了 board 文件夹中需要修改的内容：

表 1 board 文件夹需修改内容

项目	修要修改的内容说明
linker_scripts （文件夹）	BSP 特定的链接脚本
board.c/h	系统时钟、GPIO 初始化函数芯片存储器大小
Board_config.c/h	外设管脚配置及初始化
KConfig	芯片型号、系列、外设资源
SConscript	芯片启动文件、目标芯片型号

3.2 BOARD 配置

3.2.1 外设引脚配置

HC32F4A0 系列的外设引脚配置在 board_config.c 文件中具体实现。图 3 列举了串口 USART1 的引脚配置，其他外设引脚的配置可参照该函数。

```
#if defined RT_USING_SERIAL
rt_err_t rt_hw_board_uart_init(M4_USART_TypeDef *USARTx)
{
    rt_err_t result = RT_EOK;

    switch ((rt_uint32_t)USARTx)
    {
        #if defined(BSP_USING_UART1)
        case (rt_uint32_t)M4_USART1:
            /* Configure USART RX/TX pin. */
            GPIO_SetFunc(USART1_RX_PORT, USART1_RX_PIN, \
                          GPIO_FUNC_33_USART1_RX, PIN_SUBFUNC_DISABLE);
            GPIO_SetFunc(USART1_TX_PORT, USART1_TX_PIN, \
                          GPIO_FUNC_32_USART1_TX, PIN_SUBFUNC_DISABLE);

            break;
        #endif
        default:
            result = -RT_ERROR;
            break;
    }

    return result;
}
#endif
```

图 3 串口引脚配置

3.2.2 系统时钟配置

在 board.c 文件中存放了负责初始化系统时钟的函数，当制作 BSP，对系统时钟重新配置时，需更新该函数。在 HC32F4A0 系列中，该函数内容如图 5 所示，系统时钟初始化为 240MHz。更新完系统时钟，还需更新 SysTick_Configuration 函数，配置系统时钟节拍，如图 4 所示。

```
void SysTick_Configuration(void)
{
    stc_clk_freq_t stcClkFreq;
    rt_uint32_t cnts;

    CLK_GetClockFreq(&stcClkFreq);

    cnts = (rt_uint32_t)stcClkFreq.hclkFreq / RT_TICK_PER_SECOND;

    SysTick_Config(cnts);
}
```

图 4 SysTick 配置函数

```

/**
 * @brief BSP clock initialize.
 *        Set board system clock to PLLH@240MHz
 * @param None
 * @retval None
 */
void rt_hw_board_clock_init(void)
{
    stc_clk_pllh_init_t stcPLLHInit;

    CLK_ClkDiv(CLK_CATE_ALL,
               (CLK_PCLK0_DIV1 | CLK_PCLK1_DIV2 | CLK_PCLK2_DIV4 | \
                CLK_PCLK3_DIV4 | CLK_PCLK4_DIV2 | CLK_EXCLK_DIV2 | \
                CLK_HCLK_DIV1));

    (void)CLK_PLLHStrucInit(&stcPLLHInit);
    /* VCO = (8/1)*120 = 960MHz */
    stcPLLHInit.u8PLLState = CLK_PLLH_ON;
    stcPLLHInit.PLLCFGR = 0UL;
    stcPLLHInit.PLLCFGR_f.PLLM = 1UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLN = 120UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLP = 4UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLQ = 4UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLR = 4UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLSRC = CLK_PLLSRC_XTAL;
    (void)CLK_PLLHInit(&stcPLLHInit);

    /* Highspeed SRAM set to 1 Read/Write wait cycle */
    SRAM_SetWaitCycle(SRAM_SRAMH, SRAM_WAIT_CYCLE_1, SRAM_WAIT_CYCLE_1);
    /* SRAM1_2_3_4_backup set to 2 Read/Write wait cycle */
    SRAM_SetWaitCycle((SRAM_SRAM123 | SRAM_SRAM4 | SRAM_SRAMB), \
                      SRAM_WAIT_CYCLE_2, SRAM_WAIT_CYCLE_2);

    /* 0-wait @ 40MHz */
    EFM_SetWaitCycle(EFM_WAIT_CYCLE_5);
    /* 4 cycles for 200 ~ 250MHz */
    GPIO_SetReadWaitCycle(GPIO_READ_WAIT_4);
    CLK_SetSysClkSrc(CLK_SYSCLKSOURCE_PLLH);
}

```

图 5 系统时钟函数

3.2.3 RAM 配置

在 board.h 文件中配置了 RAM 相关参数，如图 6 所示

```

19  /* board configuration */
20  #define SRAM_BASE 0x1FFE0000
21  #define SRAM_SIZE 0x80000
22  #define SRAM_END (SRAM_BASE + SRAM_SIZE)

```

图 6 RAM 配置

3.3 KConfig 选项修改

KConfig 文件位于 board 文件夹中，该文件包含芯片系列和 BSP 上的外设支持。具体内容如图 7 所示。KConfig 中的芯片信息须与 BSP 芯片信息一致，外设信息根据 BSP 实际支持情况添加，其中 GPIO 和串口驱动是必备的。本次移植中还包含了 SPI、I2C、RTC、TIMER 等非必须外设。

```
menu "Hardware Drivers Config"

config MCU_HC32F4A0
    bool
    select ARCH_ARM_CORTEX_M4
    select RT_USING_COMPONENTS_INIT
    select RT_USING_USER_MAIN
    default y

menu "Onboard Peripheral Drivers"

endmenu

menu "On-chip Peripheral Drivers"
    config BSP_USING_GPIO
        bool "Enable GPIO"
        select RT_USING_PIN
        default y

        menuconfig BSP_USING_UART
            bool "Enable UART"
            default y
            select RT_USING_SERIAL
            if BSP_USING_UART
                config BSP_USING_UART1
                    bool "Enable UART1"
                    default y
            endif
        endif
    endmenu
endmenu
```

和BSP芯片信息一致

根据芯片外设实际情况填写，GPIO和串口驱动必备

图 7 KConfig 文件

3.4 工程构建相关文件修改

3.4.1 链接脚本 Link 修改

GCC、MDK、IAR 的相关链接脚本统一放在 linker_scripts 目录下，该文件夹位于 board 文件夹内，具体文件如图 8 所示。

rt_thread > bsp > hc32f4a0 > board > linker_scripts		
名称	类型	修改日期
link.icf	ICF 文件	2020/9/8 13:34
link.lds	LDS 文件	2020/10/20 13:59
link.sct	Windows Script Com...	2020/9/1 16:23

图 8 linker_scripts 链接文件

MDK 使用的链接脚本 link.sct 按图 9 修改。

```

; *****
; *** Scatter-Loading Description File generated by uVision ***
; *****

LR_IROM1 0x00000000 0x00200000 { ; load region size_region
  ER_IROM1 0x00000000 0x00200000 { ; load address = execution address
    *.o (RESET, +First)
    *(InRoot$$Sections)
    .ANY (+RO)
  }
  RW_IRAM1 0x1FFE0000 0x80000 ; RW data
  .ANY (+RW +ZI)
}

```

修改为芯片实际FLASH大小，以十六进制表示

修改为芯片RAM实际起始地址和大小，以十六进制表示

图 9 MDK 链接脚本

本次使用的芯片是 HC32F4A0SITB，FLASH 为 2M，因此修改 LR_IROM1 和 ER_IROM1 的参数为 0x00200000。RAM 的大小为 512K，因此修改 RW_IRAM1 的参数为 0x80000。

GCC 使用链接脚本 link.lds 按图 10 修改。

```

/* Use contiguous memory regions for simple. */
MEMORY
{
  FLASH      (rx): ORIGIN = 0x00000000, LENGTH = 2M
  OTP        (rx): ORIGIN = 0x03000000, LENGTH = 6876
  RAM        (rwx): ORIGIN = 0x1FFE0000, LENGTH = 512K
  RAMB       (rwx): ORIGIN = 0x200F0000, LENGTH = 4K
}

```

直接修改为FLASH和RAM实际大小

图 10 GCC 链接脚本

IAR 使用的链接脚本 link.icf 按图 11 修改。

```

/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_vl_4.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x00000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_IROM1_start__ = 0x00000000;
define symbol __ICFEDIT_region_IROM1_end__ = 0x001FFFFFFF;
define symbol __ICFEDIT_region_IROM2_start__ = 0x03000000;
define symbol __ICFEDIT_region_IROM2_end__ = 0x030017FF;
define symbol __ICFEDIT_region_EROM1_start__ = 0x0;
define symbol __ICFEDIT_region_EROM1_end__ = 0x0;
define symbol __ICFEDIT_region_EROM2_start__ = 0x0;
define symbol __ICFEDIT_region_EROM2_end__ = 0x0;
define symbol __ICFEDIT_region_EROM3_start__ = 0x0;
define symbol __ICFEDIT_region_EROM3_end__ = 0x0;
define symbol __ICFEDIT_region_IRAM1_start__ = 0x1FFE0000;
define symbol __ICFEDIT_region_IRAM1_end__ = 0x1FFFFFFF;
define symbol __ICFEDIT_region_IRAM2_start__ = 0x20000000;
define symbol __ICFEDIT_region_IRAM2_end__ = 0x2001FFFF;
define symbol __ICFEDIT_region_IRAM3_start__ = 0x20020000;
define symbol __ICFEDIT_region_IRAM3_end__ = 0x2003FFFF;
define symbol __ICFEDIT_region_IRAM4_start__ = 0x20040000;
define symbol __ICFEDIT_region_IRAM4_end__ = 0x20057FFF;
define symbol __ICFEDIT_region_IRAM5_start__ = 0x20058000;
define symbol __ICFEDIT_region_IRAM5_end__ = 0x2005FFFF;
define symbol __ICFEDIT_region_IRAM6_start__ = 0x200F0000;
define symbol __ICFEDIT_region_IRAM6_end__ = 0x200F0FFF;
define symbol __ICFEDIT_region_ERAM1_start__ = 0x0;
define symbol __ICFEDIT_region_ERAM1_end__ = 0x0;
define symbol __ICFEDIT_region_ERAM2_start__ = 0x0;
define symbol __ICFEDIT_region_ERAM2_end__ = 0x0;
define symbol __ICFEDIT_region_ERAM3_start__ = 0x0;
define symbol __ICFEDIT_region_ERAM3_end__ = 0x0;

define symbol __ICFEDIT_region_RAM_end__ = __ICFEDIT_region_IRAM6_end__;
export symbol __ICFEDIT_region_RAM_end__;

/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x2000;
define symbol __ICFEDIT_size_proc_stack__ = 0x0;
define symbol __ICFEDIT_size_heap__ = 0x2000;
/**** End of ICF editor section. ###ICF###*/

```

按照芯片实际的地址域填写

图 11 IAR 链接脚本

3.4.2 构建脚本 SConscript 修改

构建脚本 SConscript 决定 MDK/IAR 工程的生成以及编译过程中要添加的文件。不同路径下的 SConscript 脚本包含的内容略有差异，drivers 文件夹中的构建脚本决定了编译过程中要添加的驱动文件，若 BSP 中有新支持的外设，可修改此文件，具体内容如图 12 所示。

```
from building import *

cwd = GetCurrentDir()

# add the general drivers.
src = Split("""
drv_irq.c
""")

if GetDepend(['RT_USING_PIN']):
    src += ['drv_gpio.c']

if GetDepend(['RT_USING_SERIAL']):
    src += ['drv_usart.c']

if GetDepend(['RT_USING_I2C', 'RT_USING_I2C_BITOPS']):
    src += ['drv_soft_i2c.c']

if GetDepend(['RT_USING_SPI']):
    src += ['drv_spi.c']
```

基本必须的

可选的 (BSP支持的外设)

图 12 驱动文件构建脚本

Libraries 文件夹下的构建脚本决定编译过程中要添加的库文件以及启动文件，具体内容如图 13 所示。这里做了一下区分，默认添加的库文件只包含内核外设，诸如时钟、FLASH、电源、中断等模块，其他外设库文件为可配置。

也可修改成图 13 中注释部分代码，可实现库文件全部添加编译进工程中。

代码如下：

```
src += Glob('HC32F4A0_StdPeriph_Driver/src/*.c')
```

```

import rtconfig
Import('RTT_ROOT')
from building import *

# get current directory
cwd = GetCurrentDir()

# The set of source files associated with this SConscript file.
src = Split("""
CMSIS/Device/HDSC/HC32F4A0/Source/system_hc32f4a0.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_clk.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_dma.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_efm.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_gpio.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_icg.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_interrupts.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_pwc.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_sram.c
HC32F4A0_StdPeriph_Driver/src/hc32f4a0_utility.c
""")

#src += Glob('HC32F4A0_StdPeriph_Driver/src/*.c')

if GetDepend(['RT_USING_SERIAL']):
    src += ['HC32F4A0_StdPeriph_Driver/src/hc32f4a0_usart.c']
    src += ['HC32F4A0_StdPeriph_Driver/src/hc32f4a0_tmr0.c']

if GetDepend(['RT_USING_I2C']):
    src += ['HC32F4A0_StdPeriph_Driver/src/hc32f4a0_i2c.c']

if GetDepend(['RT_USING_SPI']):

if GetDepend(['RT_USING_CAN']):

if GetDepend(['RT_USING_ADC']):

if GetDepend(['RT_USING_RTC']):

if GetDepend(['RT_USING_WDT']):

if GetDepend(['RT_USING_HWTIMER']) or GetDepend(['RT_USING_PWM']) or GetDepend(['

#add for startup script
if rtconfig.CROSS_TOOL == 'gcc':
    src = src + ['CMSIS/Device/HDSC/HC32F4A0/Source/GCC/startup_hc32f4a0.S']
elif rtconfig.CROSS_TOOL == 'keil':
    src = src + ['CMSIS/Device/HDSC/HC32F4A0/Source/ARM/startup_hc32f4a0.s']
elif rtconfig.CROSS_TOOL == 'iar':
    src = src + ['CMSIS/Device/HDSC/HC32F4A0/Source/IAR/startup_hc32f4a0.s']

```

默认添加的库文件

可选的库文件

启动文件

图 13 启动文件构建脚本

3.4.3 rtconfig.py 文件修改

该文件包含芯片内核信息，以及编译工具的位置、预编译信息。图 14 为必须修改内容，文件中其他地方也需要根据实际情况做相应的修改。

```
3 # toolchains options
4 ARCH='arm'
5 CPU='cortex-m4'
```

图 14 rtconfig.py 文件

3.4.4 工程模板修改

template 文件是生成 MDK/IAR 工程的模板文件，通过修改该文件可以设置工程中使用的芯片型号以及下载方式。

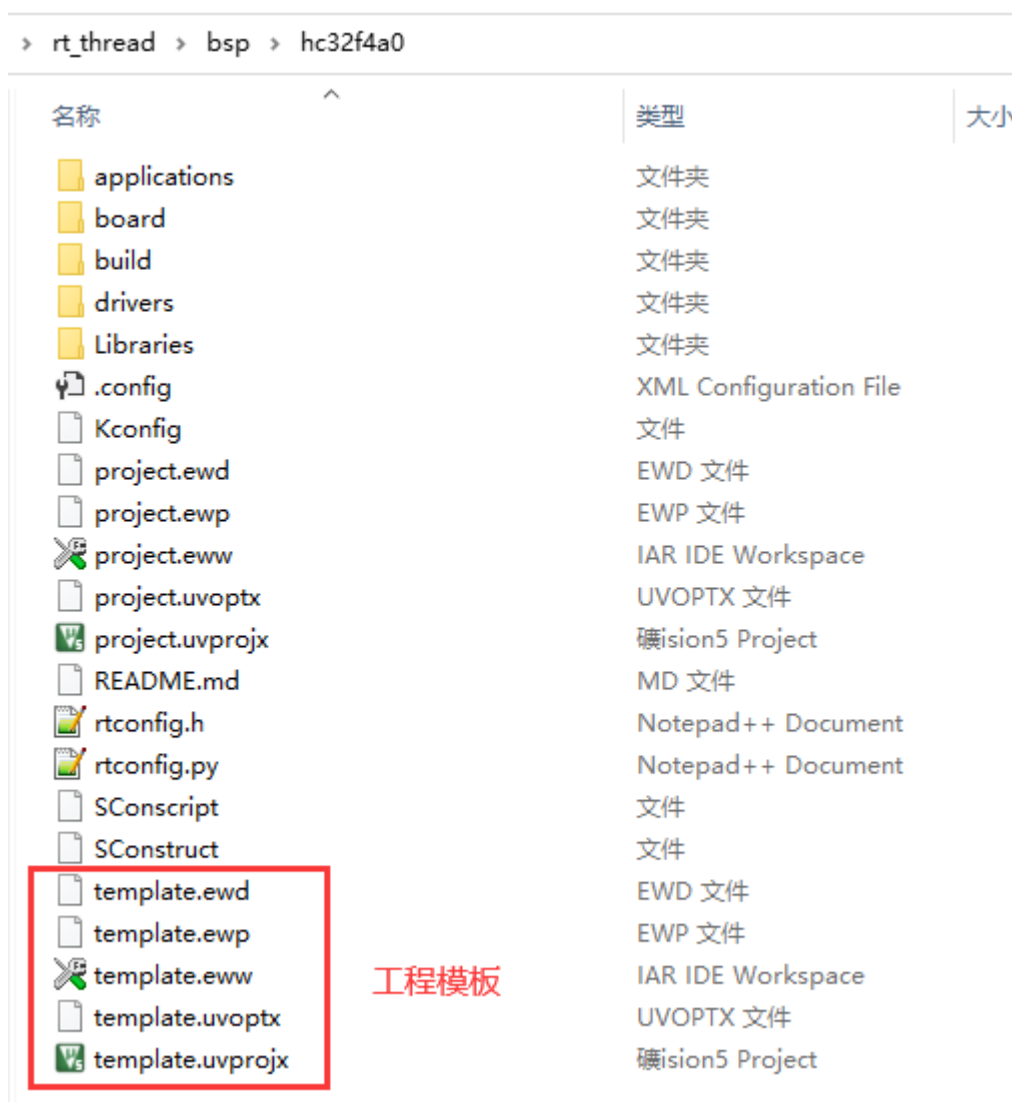


图 15 工程模板

下面以 MDK5 模板的修改为例，介绍如何修改模板配置：

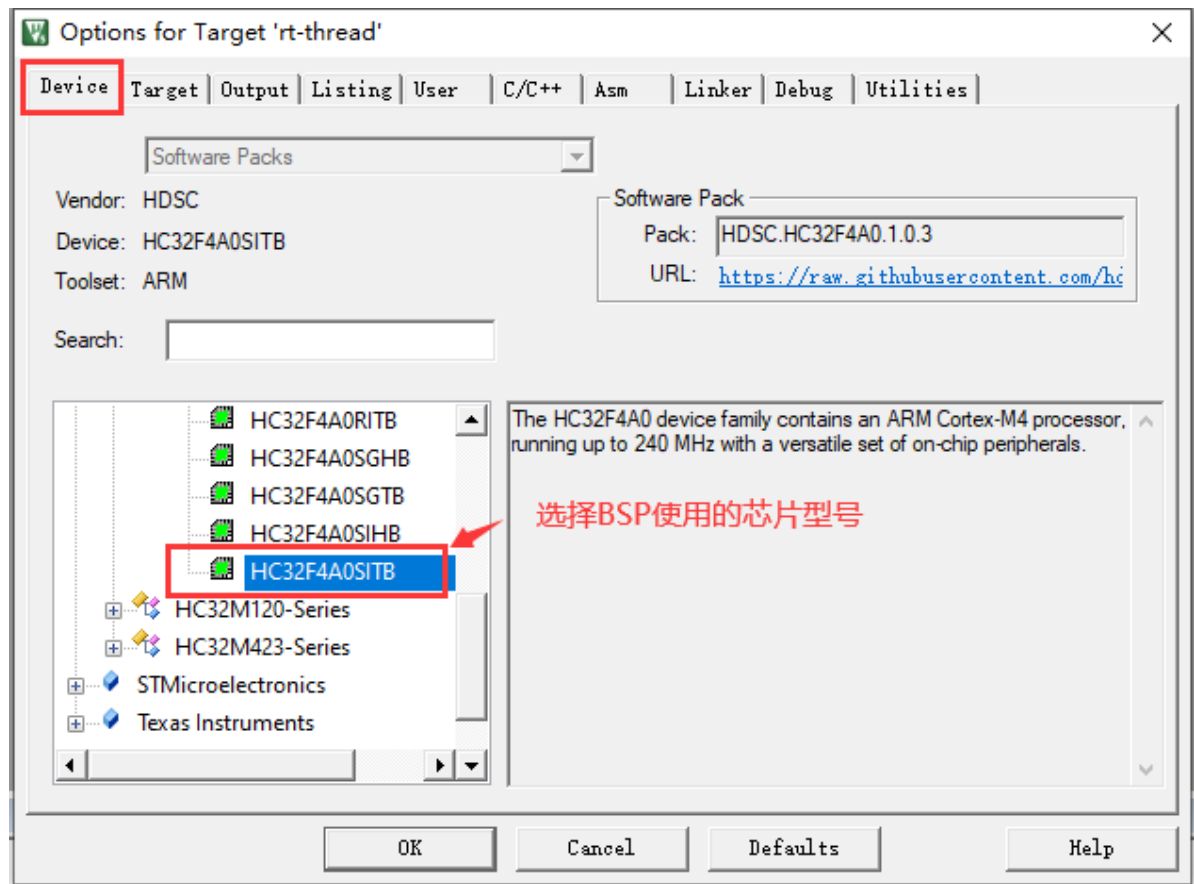


图 16 芯片型号配置

修改程序下载方式：

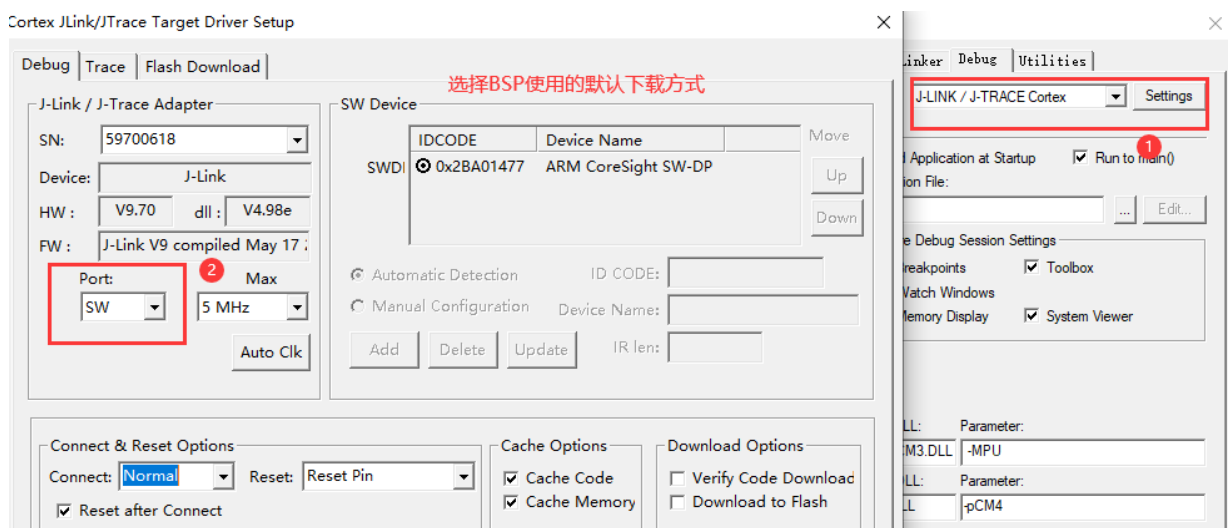


图 17 下载方式配置

3.5 重新生成工程

重新生成工程需要使用 ENV 工具。

3.5.1 重新生成 rtconfig.h 文件

在 ENV 界面输入命令 `menuconfig` 对工程进行配置，并生成新的 `rtconfig.h` 文件，如图 18 所示输入命令。

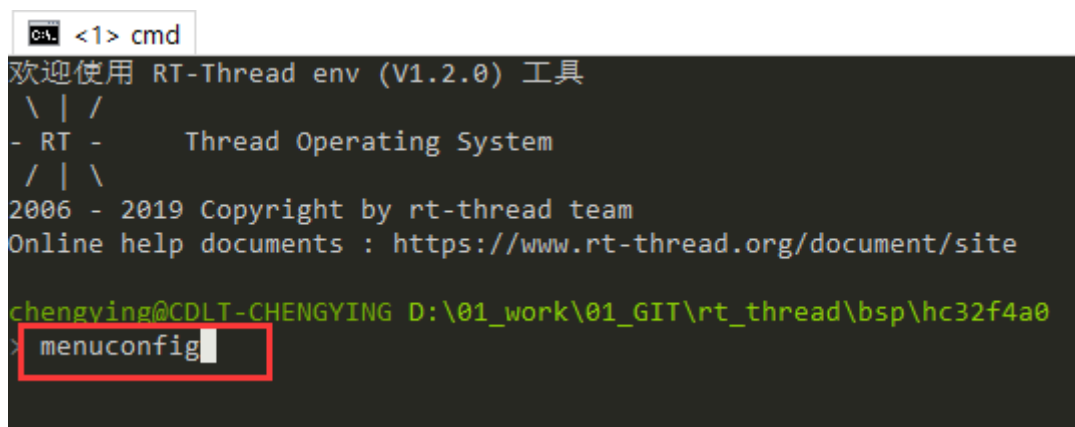


图 18 menuconfig

回车会出现弹框，按图 19 所示进入外设选择界面，选择需要打开的外设。

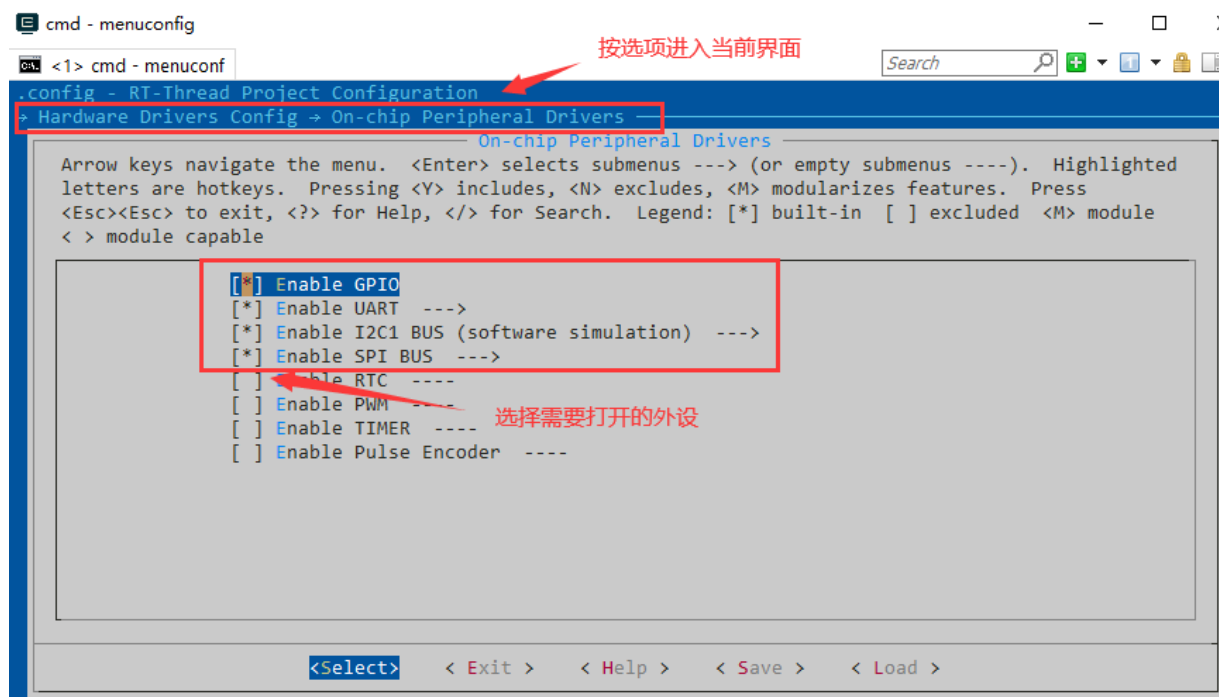
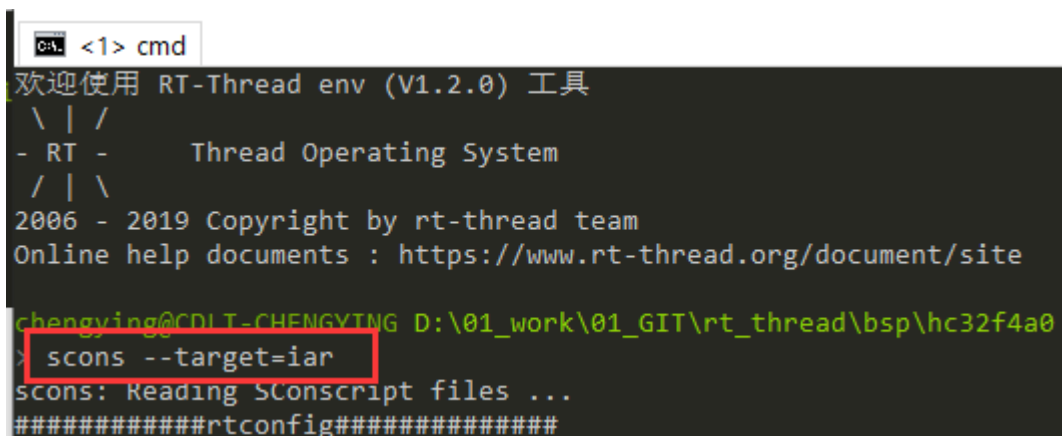


图 19 选择需要打开的外设

3.5.2 重新生成 MDK/IAR 工程

使用 ENV 工具输入命令 `scons --target=iar` 可重新生成 IAR 工程，输入命令 `scons --target=mdk5` 可以重新生成 MDK 工程，如图所示：

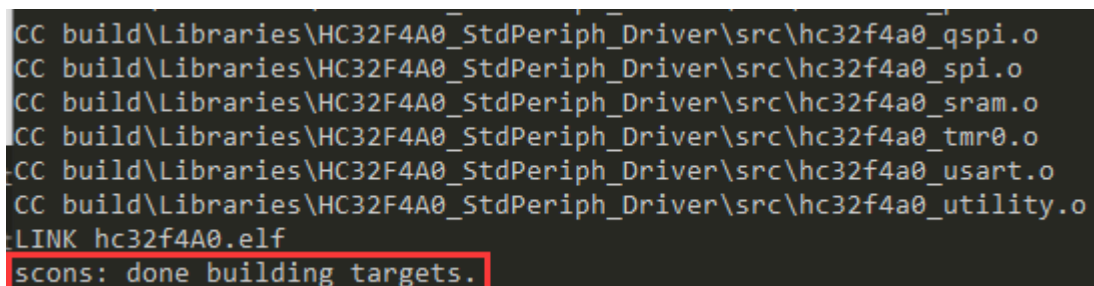


```
C:\> cmd
欢迎使用 RT-Thread env (V1.2.0) 工具
\ | /
- RT -      Thread Operating System
/ | \
2006 - 2019 Copyright by rt-thread team
Online help documents : https://www.rt-thread.org/document/site

chengying@CDLT-CHENGYING D:\01_work\01_GIT\rt_thread\bsp\hc32f4a0
> scons --target=iar
scons: Reading SConscript files ...
#####rtconfig#####
```

图 20 生成工程

重新生成工程成功



```
CC build\Libraries\HC32F4A0_StdPeriph_Driver\src\hc32f4a0_qspi.o
CC build\Libraries\HC32F4A0_StdPeriph_Driver\src\hc32f4a0_spi.o
CC build\Libraries\HC32F4A0_StdPeriph_Driver\src\hc32f4a0_sram.o
CC build\Libraries\HC32F4A0_StdPeriph_Driver\src\hc32f4a0_tmr0.o
CC build\Libraries\HC32F4A0_StdPeriph_Driver\src\hc32f4a0_usart.o
CC build\Libraries\HC32F4A0_StdPeriph_Driver\src\hc32f4a0_utility.o
LINK hc32f4A0.elf
scons: done building targets.
```

图 21 生成工程成功

到此新的 BSP 就可以使用了。

4 样例代码

按照 RT-Thread 要求，示例 main 函数里展示 LED 灯闪烁程序。

4.1 代码介绍

在第三章介绍 BSP 制作过程中，已经贴出了部分示例代码，这里只展示 main 函数。

```
int32_t main(void)
{
    rt_kprintf("Os is Start!!! \n");

    while(1)
    {
        rt_pin_mode(LED_PIN, PIN_MODE_OUTPUT);
        rt_pin_write(LED_PIN, PIN_HIGH);
        rt_thread_delay(DELAY_MS);
        rt_pin_write(LED_PIN, PIN_LOW);
        rt_thread_delay(DELAY_MS);
    };
}
```

图 22 示例代码

5 总结

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

6 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

