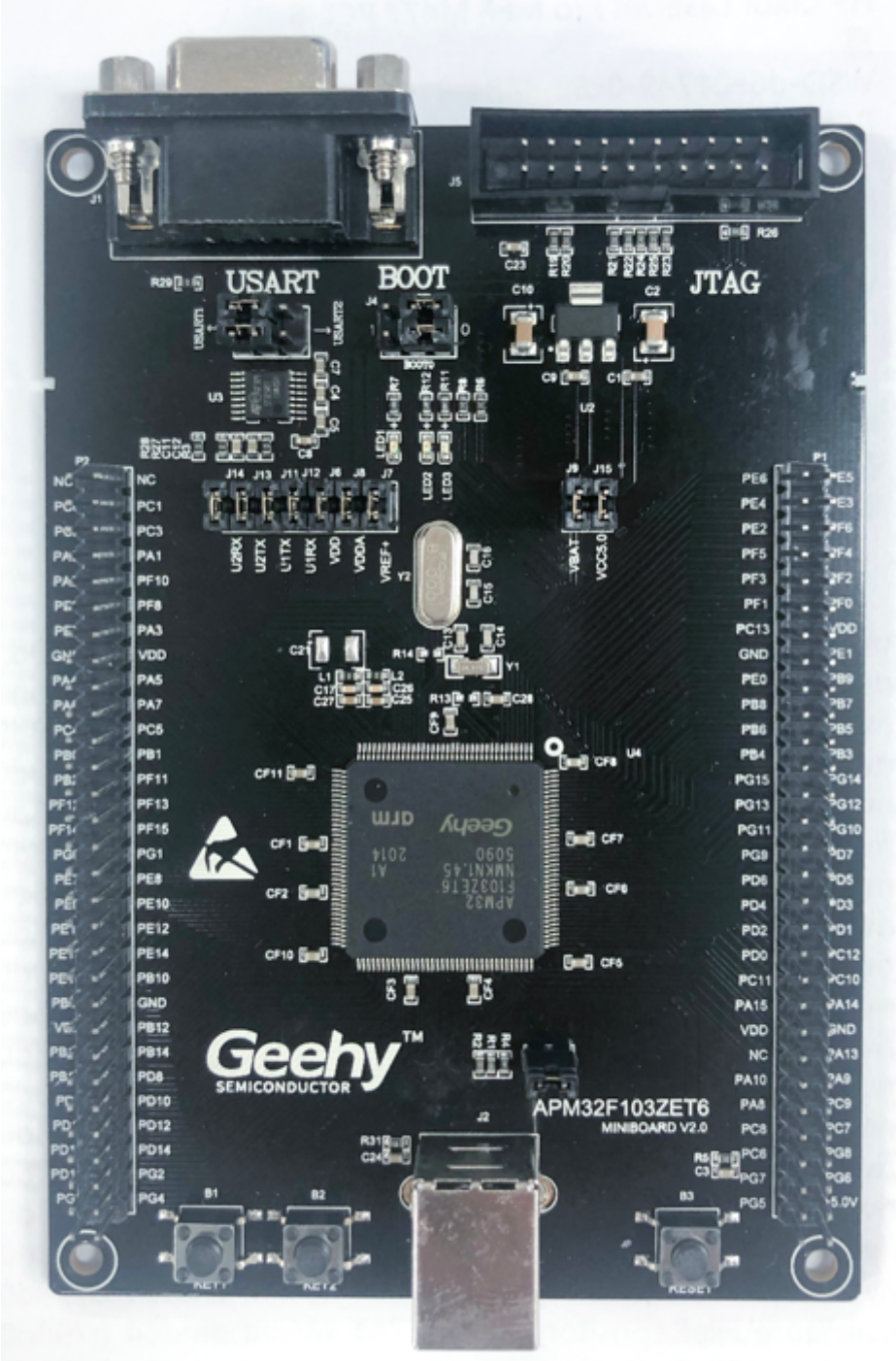
【国产MCU移植】 APM32F103ZE

1 环境搭建

1.1 硬件准备

APM32F103ZE MINI BROARD



1.2 软件准备

编程器，使用JLINK进行下载调试。
集成开发环境，安装KEIL5。
串口助手使用的是Tera Term。

1.2.1 APM32F103 SDK

下载APM32F103ZE SDK包

APM32F10x_SDK_V1.5.zip

并安装其中的keil支持文件： 

。

1.2.2 RT-thrad

下载RT-thread源码。

下载RT-Thread env 工具 RT-Thread物联网操作系统。

2 BSP标准工程生成

其实移植RT-THREAD到一些比较通用的内核还是比较方便的，因为可以投机取巧。那接下来告诉你怎么投机取巧移植RT-Thread到国产MCU。

本文只适配KEIL5的环境，GCC、KEIL4和IAR环境不做讲解。

基础模板：首先看看RT-Thread代码仓库中已有的BSP存在同是M3内核的芯片STM32F103。而我要移植的是APM32F103，参照STM32F103的工程，我们新建相似的工程目录。然后就开始增删改查，完成最终的BSP。

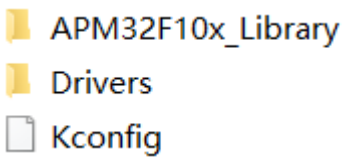
2.1 制作通用文件结构

bsp 文件夹目录下新建文件夹 apm32，再在apm32文件夹中新建libraries和apm32f103xe-minibroard两个文件夹。

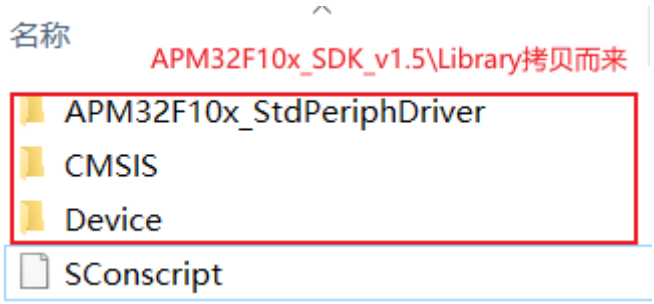
项目	需要修改的内容说明
libraries（文件夹）	apm32相关驱动库
apm32f103xe-minibroard （文件夹）	F1 系列 103XE MINI BORAD开发板 BSP

2.1.1 libraries 文件夹

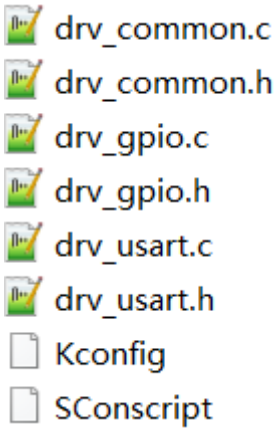
libraries 文件夹下新建 APM32F10x_Library 文件夹以存放APM32F10x系列的libraries，新建 Drivers 文件夹以存放APM32F10x系列的RT-thread 外设驱动。创建Kconfig文件（后续编写文件内容）。



复制我们下载好的 APM32F10x系列的SDK下的 文件夹Library 内容到这里，再在该文件夹下新建SConscript文件。这样子我们的APM32F10x_Library文件夹内容就有：



在Drivers文件夹中 新建若干文件如下（后续编写文件内容）：

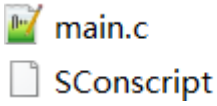


2.1.2 apm32f103xe-minibroard 文件夹

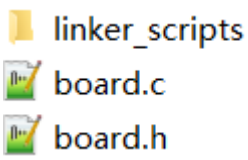
apm32f103xe-minibroard 文件夹下新建applications和board文件夹。

项目	需要修改的内容说明
applications（文件夹）	BSP 应用 APP
board（文件夹）	BSP 板载驱动文件

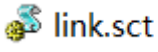
applications 文件夹中新建两个文件，main.c 和 SConscript。文件内容稍后会在后续章节进行编写。



board 文件夹新建文件夹 linker_scripts 及两个文件，board.c 和 board.h。文件内容稍后会在后续章节进行编写。

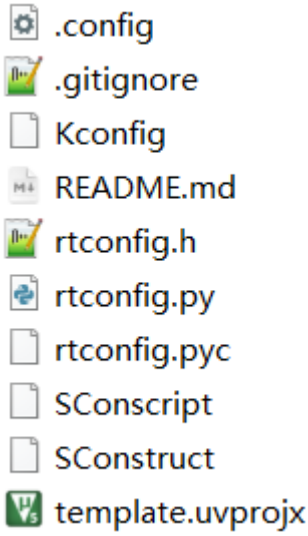


linker_scripts 文件夹下新建文件link.sct。文件内容稍后会在后续章节进行编写。



2.1.3 其他文件

我们还需要复制以下文件（如：\bsp\stm32\stm32f103-atk-nano\）下至我们的工作目录（bsp\apm32\apm32f103xe-minibroard）。后续我们会对部分文件进行编辑，请留意。

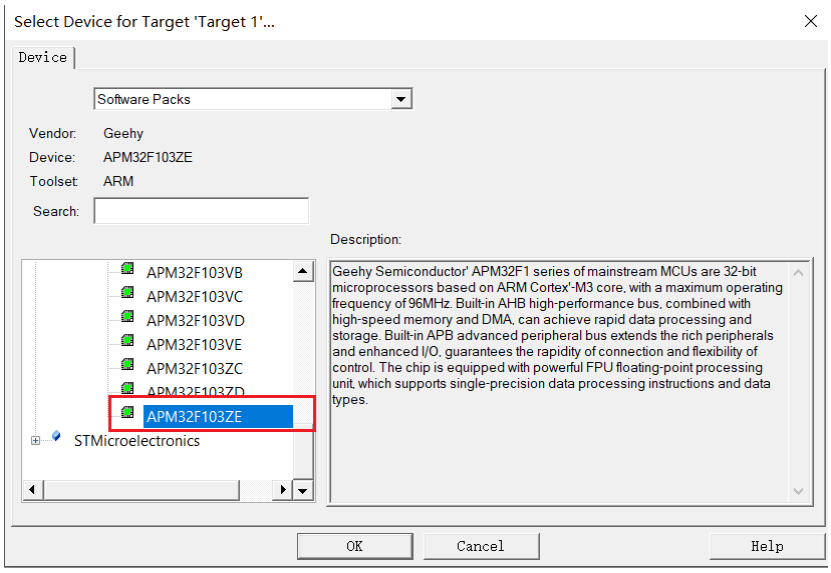


2.2 创建工程

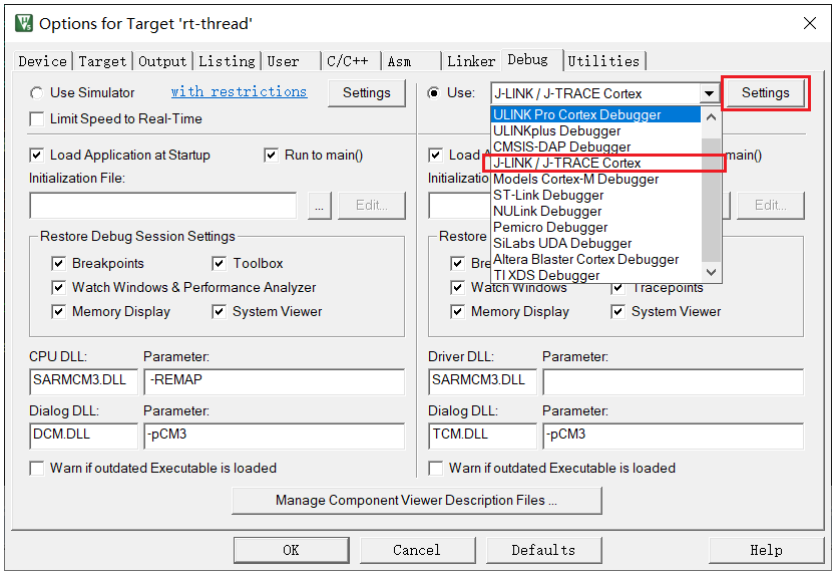
(创建工程前请安装好APM32F1系列keil支持pack包。)

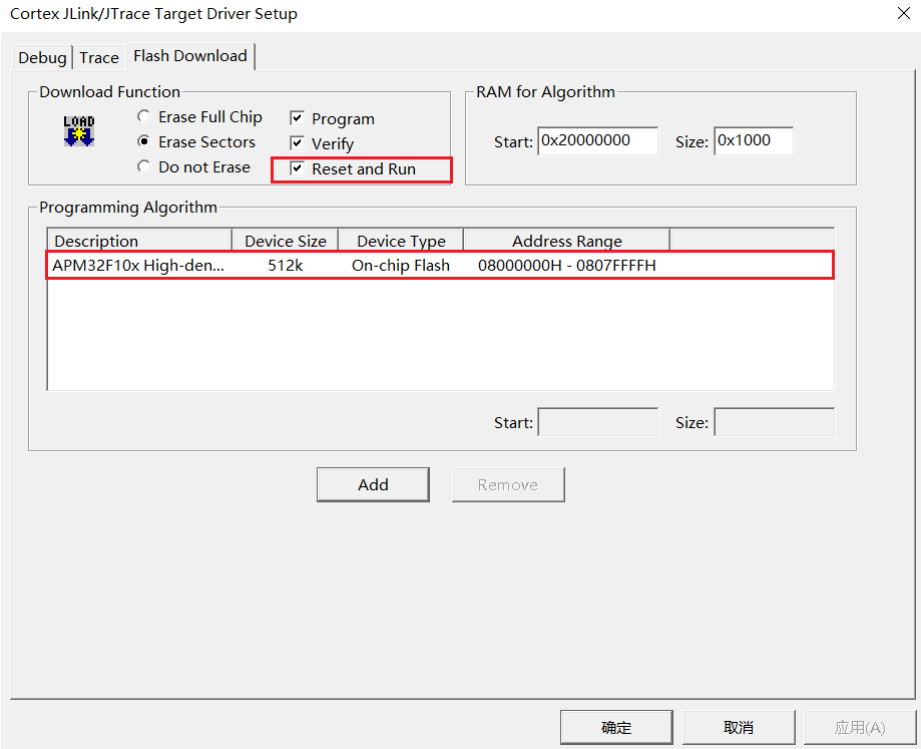
点击template文件打开工程，将芯片更换为APM32F103ZE。

Device选项卡，选择芯片 APM32F103ZE，点击OK，芯片更换完毕。



Debug选项卡，选择手上的仿真器型号，这里笔者是J-link,选择后点击“Seting”选择下载后的操作选项及下载算法。





至此工程创建完毕。下一步我们将基于该工程，编译我们的RT-thread。

2.3 Kconfig及SConstruct文件编写

2.3.1 apm32f103xe-minibroard 目录

rtconfig.h内容更改为：

```
#ifndef RT_CONFIG_H__
#define RT_CONFIG_H__

/* Automatically generated file; DO NOT EDIT. */
/* RT-Thread Configuration */

/* RT-Thread Kernel */

#define RT_NAME_MAX 8
#define RT_ALIGN_SIZE 4
#define RT_THREAD_PRIORITY_32
#define RT_THREAD_PRIORITY_MAX 32
```

```
#define RT_TICK_PER_SECOND 1000
#define RT_USING_OVERFLOW_CHECK
#define RT_USING_HOOK
#define RT_USING_IDLE_HOOK
#define RT_IDLE_HOOK_LIST_SIZE 4
#define IDLE_THREAD_STACK_SIZE 256
#define RT_DEBUG
#define RT_DEBUG_COLOR

/* Inter-Thread communication */

#define RT_USING_SEMAPHORE
#define RT_USING_MUTEX
#define RT_USING_EVENT
#define RT_USING_MAILBOX
#define RT_USING_MESSAGEQUEUE

/* Memory Management */

#define RT_USING_MEMPOOL
#define RT_USING_SMALL_MEM
#define RT_USING_HEAP

/* Kernel Device Object */

#define RT_USING_DEVICE
#define RT_USING_CONSOLE
#define RT_CONSOLEBUF_SIZE 128
#define RT_CONSOLE_DEVICE_NAME "uart1"
#define RT_VER_NUM 0x40001
#define ARCH_ARM
#define ARCH_ARM_CORTEX_M
#define ARCH_ARM_CORTEX_M3

/* RT-Thread Components */

#define RT_USING_COMPONENTS_INIT
#define RT_USING_USER_MAIN
#define RT_MAIN_THREAD_STACK_SIZE 2048
#define RT_MAIN_THREAD_PRIORITY 10

/* C++ features */


/* Command shell */

#define RT_USING_FINSH
#define FINSH_THREAD_NAME "tshell"
#define FINSH_USING_HISTORY
#define FINSH_HISTORY_LINES 5
#define FINSH_USING_SYMTAB
#define FINSH_USING_DESCRIPTION
#define FINSH_THREAD_PRIORITY 20
#define FINSH_THREAD_STACK_SIZE 4096
#define FINSH_CMD_SIZE 80
#define FINSH_USING_MSH
#define FINSH_USING_MSH_DEFAULT
#define FINSH_USING_MSH_ONLY
```

```
#define FINSH_ARG_MAX 10

/* Device virtual file system */


/* Device Drivers */

#define RT_USING_DEVICE_IPC
#define RT_PIPE_BUFSZ 512
#define RT_USING_SERIAL
#define RT_SERIAL_USING_DMA
#define RT_SERIAL_RB_BUFSZ 64
#define RT_USING_PIN

/* Using WiFi */


/* Using USB */


/* POSIX layer and C standard library */


/* Network */

/* Socket abstraction layer */


/* Network interface device */


/* light weight TCP/IP stack */


/* Modbus master and slave stack */


/* AT commands */


/* VBUS(Virtual Software BUS) */


/* Utilities */


/* RT-Thread online packages */

/* IoT - internet of things */


/* Wi-Fi */

/* Marvell WiFi */


/* Wiced WiFi */
```

```
/* IoT Cloud */


/* security packages */


/* language packages */


/* multimedia packages */


/* tools packages */


/* system packages */


/* peripheral libraries and drivers */

/* sensors drivers */


/* miscellaneous packages */


/* samples: kernel and components samples */

#define SOC_FAMILY_APM32
#define SOC_SERIES_APM32F1

/* Hardware Drivers Config */

#define SOC_APM32F103ZE

/* Onboard Peripheral Drivers */


/* On-chip Peripheral Drivers */

#define BSP_USING_GPIO
#define BSP_USING_UART
#define BSP_USING_UART1

/* Board extended module Drivers */


#endif
```

Kconfig内容更改为：

```
mainmenu "RT-Thread Configuration"

config BSP_DIR
    string
    option env="BSP_ROOT"
```

```
    default "."

config RTT_DIR
    string
    option env="RTT_ROOT"
    default "../../.."

config PKGS_DIR
    string
    option env="PKGS_ROOT"
    default "packages"

source "$RTT_DIR/Kconfig"
source "$PKGS_DIR/Kconfig"
source "../libraries/Kconfig"
source "board/Kconfig"
```

SConscript文件更改为：

```
# for module compiling
import os
Import('RTT_ROOT')
from building import *

cwd = GetCurrentDir()
objs = []
list = os.listdir(cwd)

for d in list:
    path = os.path.join(cwd, d)
    if os.path.isfile(os.path.join(path, 'SConscript')):
        objs = objs + SConscript(os.path.join(d, 'SConscript'))

Return('objs')
```

SConstruct文件内容更改为：

```
import os
import sys
import rtconfig

if os.getenv('RTT_ROOT'):
    RTT_ROOT = os.getenv('RTT_ROOT')
else:
    RTT_ROOT = os.path.normpath(os.getcwd() + '/../../..')

sys.path = sys.path + [os.path.join(RTT_ROOT, 'tools')]
try:
    from building import *
except:
    print('Cannot found RT-Thread root directory, please check RTT_ROOT')
    print(RTT_ROOT)
    exit(-1)

TARGET = 'rtthread.' + rtconfig.TARGET_EXT
```

```
DefaultEnvironment(tools=[])
env = Environment(tools = ['mingw'],
    AS = rtconfig.AS, ASFLAGS = rtconfig.AFLAGS,
    CC = rtconfig.CC, CCFLAGS = rtconfig.CFLAGS,
    AR = rtconfig.AR, ARFLAGS = '-rc',
    CXX = rtconfig.CXX, CXXFLAGS = rtconfig.CXXFLAGS,
    LINK = rtconfig.LINK, LINKFLAGS = rtconfig.LFLAGS)
env.PrependENVPath('PATH', rtconfig.EXEC_PATH)

if rtconfig.PLATFORM == 'iar':
    env.Replace(CCCOM = ['$CC $CCFLAGS $CPPFLAGS $_CPPDEFFLAGS $_CPPINCFLAGS -o
$TARGET $SOURCES'])
    env.Replace(ARFLAGS = [''])
    env.Replace(LINKCOM = env["LINKCOM"] + ' --map rtthread.map')

Export('RTT_ROOT')
Export('rtconfig')

SDK_ROOT = os.path.abspath('./')

if os.path.exists(SDK_ROOT + '/libraries'):
    libraries_path_prefix = SDK_ROOT + '/libraries'
else:
    libraries_path_prefix = os.path.dirname(SDK_ROOT) + '/libraries'

SDK_LIB = libraries_path_prefix
Export('SDK_LIB')

# prepare building environment
objs = PrepareBuilding(env, RTT_ROOT, has_libcpu=False)

apm32_library = 'APM32F10x_Library'
rtconfig.BSP_LIBRARY_TYPE = apm32_library

# include libraries
objs.extend(SConscript(os.path.join(libraries_path_prefix, apm32_library,
'SConscript')))

# include drivers
objs.extend(SConscript(os.path.join(libraries_path_prefix, 'Drivers',
'SConscript')))

# make a building
DoBuilding(TARGET, objs)
```

2.3.2 applications 目录

SConscript文件更改为：

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

cwd = GetCurrentDir()
src = Glob('*.c')
CPPPATH = [cwd]

group = DefineGroup('Applications', src, depend = [''], CPPPATH = CPPPATH)

Return('group')
```

2.3.3 boardd 目录

Kconfig文件更改为：

```
menu "Hardware Drivers Config"

config SOC_APM32F103ZE
    bool
    select SOC_SERIES_APM32F1
    select RT_USING_COMPONENTS_INIT
    select RT_USING_USER_MAIN
    default y

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

    source "../libraries/HAL_Drivers/Kconfig"

endmenu

endmenu
```

SConscript文件更改为：

```
import os
import rtconfig
from building import *
```

```
Import('SDK_LIB')

cwd = GetCurrentDir()

# add general drivers
src = Split('''
board.c
''')

path =  [cwd]

startup_path_prefix = SDK_LIB

if rtconfig.CROSS_TOOL == 'keil':
    src += [startup_path_prefix +
'/APM32F10x_Library/Device/Geehy/APM32F10x/Source/ARM/startup_apm32f10x_hd.s']

# You can select chips from the list above
CPPDEFINES = ['APM32F103xE']
group = DefineGroup('Drivers', src, depend = [''], CPPPATH = path, CPPDEFINES =
CPPDEFINES)
Return('group')
```

2.3.4 bsp\apm32\libraries目录

Kconfig文件更改为：

```
config SOC_FAMILY_APM32
    bool

config SOC_SERIES_APM32F1
    bool
    select ARCH_ARM_CORTEX_M3
    select SOC_FAMILY_APM32
```

2.3.5 bsp\apm32\APM32F10x_Library目录

SConscript文件更改为：

```
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("""
Device/Geehy/APM32F10x/Source/system_apm32f10x.c
APM32F10x_StdPeriphDriver/src/apm32f10x_gpio.c
APM32F10x_StdPeriphDriver/src/apm32f10x_misc.c
APM32F10x_StdPeriphDriver/src/apm32f10x_rcm.c
APM32F10x_StdPeriphDriver/src/apm32f10x_usart.c
APM32F10x_StdPeriphDriver/src/apm32f10x_eint.c
""")

path = [cwd + '/Device/Geehy/APM32F10x/Include',
```

```
    cwd + '/APM32F10x_StdPeriphDriver/inc',
    cwd + '/CMSIS/Include']

CPPDEFINES = ['USE_STDPERIPH_DRIVER']
group = DefineGroup('Libraries', src, depend = [''], CPPPATH = path, CPPDEFINES
= CPPDEFINES)

Return('group')
```

2.3.6 bsp\apm32\Drivers目录

Kconfig文件暂时不编辑，保留为空，后续BSP包支持需要再进行编辑。SConscript文件更改为：

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

cwd = GetCurrentDir()

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

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

if GetDepend(['RT_USING_SERIAL']):
    if GetDepend(['RT_USING_SERIAL_V2']):
        src += ['drv_usart_v2.c']
    else:
        src += ['drv_usart.c']

src += ['drv_common.c']

path =  [cwd]
path += [cwd + '/config']

group = DefineGroup('Drivers', src, depend = [''], CPPPATH = path)

Return('group')
```

2.3 链接文件编写

board\linker_scripts目录下文件link.sct内容更改为：

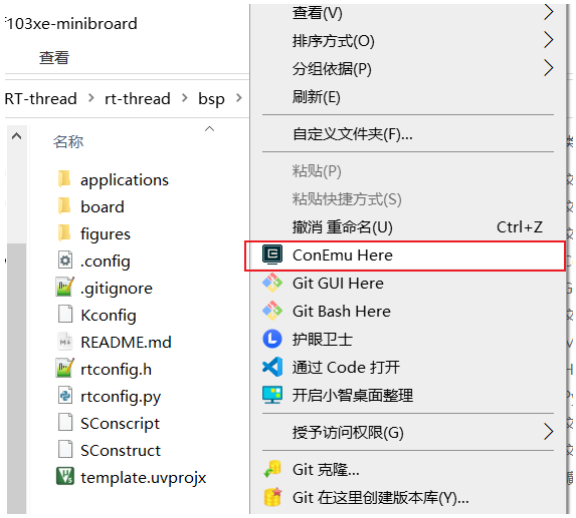
```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08000000 0x00080000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00080000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
   .ANY (+XO)
  }
```

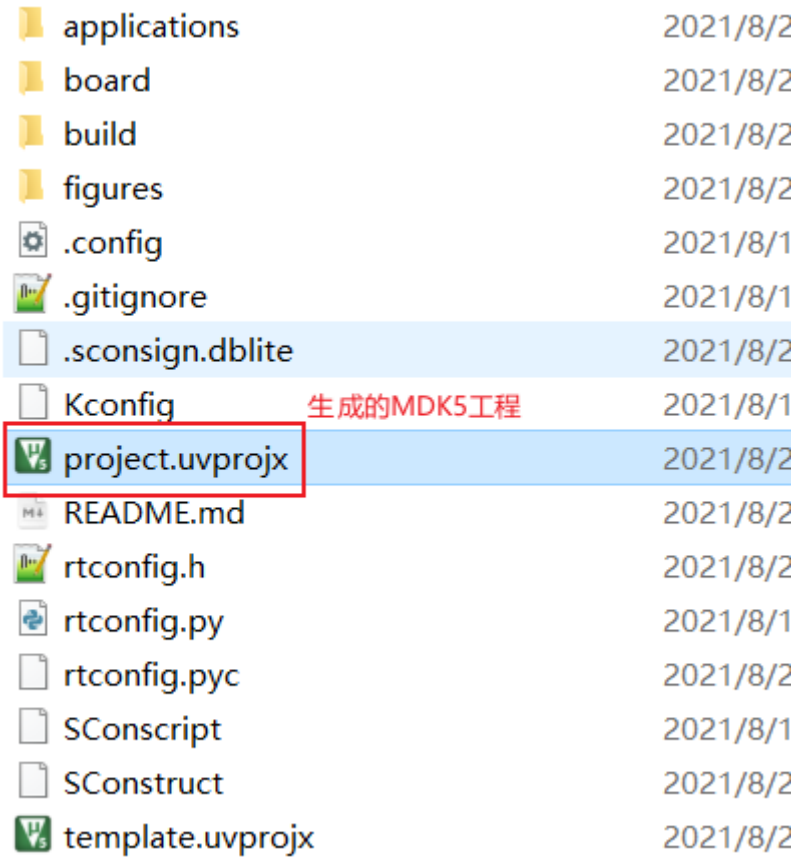
```
  RW_IRAM1 0x20000000 0x00020000  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

2.4 标准工程生成

（需提前注册evn工具至右键菜单）

在apm32f103xe-minibroard点击鼠标右键，选择“ConEmu Here”，调出命令行窗口输入“scons --target=mdk5”后按回车，生成MDK5工程。

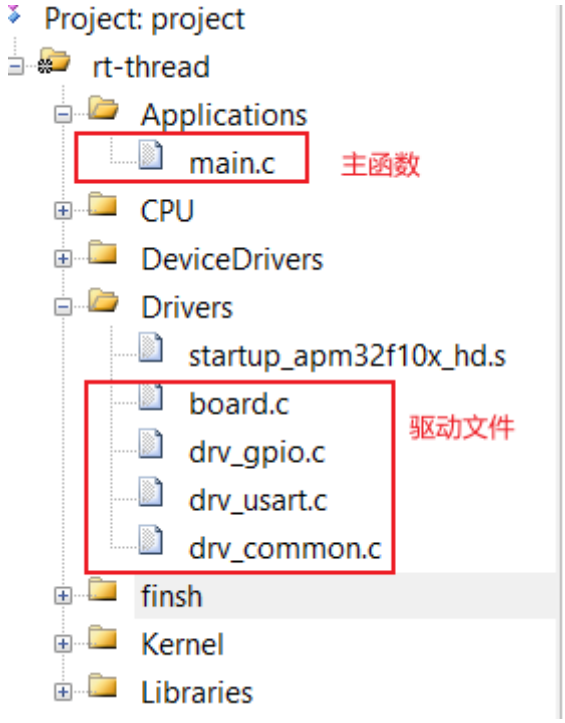




3 BSP驱动文件编写及下载

要完成RT-thread适配，各种驱动及系统运行前的初始化必不可少，下面我们开始编写外设驱动及系统时钟初始化。

我们点击上一章节生成的工程“project.uvprojx”，查看一下我们需要编写的文件。



3.1 Pin 驱动

为了跟其他rt--thread工程一样用统一的方法控制GPIO，移植PIN功能很有必要。关键是要实现好IO port和pin的映射关系，中断的映射关系。这里就不展开文件drv_gpio.c/h文件的编写工作了。大家可直接查看源码。

3.2 串口驱动

可参照APM32的官方例程，对照RT-thread底层接口完成，这里drv_usart.c/h也不展开。大家可直接查看源码。

3.3 板载初始化即主函数

板载文件主要是初始化串口驱动所需的时钟及IO，主函数是完成对板载一个LED灯的控制，程序较为简单，大家可直接看源码。

3.4 编译验证

通过前面的章节，我们已经完成了BSP的主要移植工作，现在我们编译下载一下看看吧。连接串口，下载程序后，我们可以看到Tera Term窗口显示：

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.4 build Aug 20 2021
 2006 - 2021 Copyright by rt-thread team
msh >
```

4 移植心得

关于RT-Thread的移植还是比较方便的，由于APM32F103是较为通用的M3内核，国内外都有较为详细的移植教程及移植文件。笔者本次在各位前辈的基础上较为轻松地完成了APM32F103ZE的移植工作。