

# MDK 工程结构说明

第一章，MDK 工程结构规范作用 .....	3
第二章、不使用 CubeMX 的工程结构 .....	4
2.1 寄存器版本工程结构 .....	4
2.2 HAL 库版本工程结构 .....	9
第三章、使用 CubeMX 的工程结构 .....	11
第四章、MDK 工程说明 .....	13
4.1 寄存器版本工程说明 .....	14
4.2 HAL 库版本工程说明 .....	17
4.3 CubeMX 版本工程说明 .....	18

# 第一章，MDK 工程结构规范作用

为了更好的适应公司发展需求，并提供给客户更加规范的范例代码，我们对正点原子所有嵌入式单片机教学的工程结构进行新的约束和规范，以满足公司未来发展的使用需求。

从 2020 年开始，所有新编写的嵌入式单片机代码，都必须遵循本规范，且需要对老产品/模块进行升级，使用新规范，满足新需求。

本规范共三章：

- 1， 不使用 CubeMX 的工程结构
- 2， 使用 CubeMX 的工程结构
- 3， MDK 工程说明

## 第二章、不使用 CubeMX 的工程结构

当我们不完全使用 CubeMX 生产工程的时候（即可以用 CubeMX 生成参考代码，但是例程还是由我们自己通过 MDK 新建工程实现），我们参考 ST 提供的 STM32Cube\_FW\_XX（XX=F1/F4/F7/H7 等）固件库包，通过 MDK 自行新建工程，有以下两种情况：

- 1，寄存器版本工程
- 2，HAL 库版本工程

接下来，我们分别介绍这两种情况。

### 2.1 寄存器版本工程结构

寄存器版本新建工程，工程目录结构如图 2.1.1 所示：



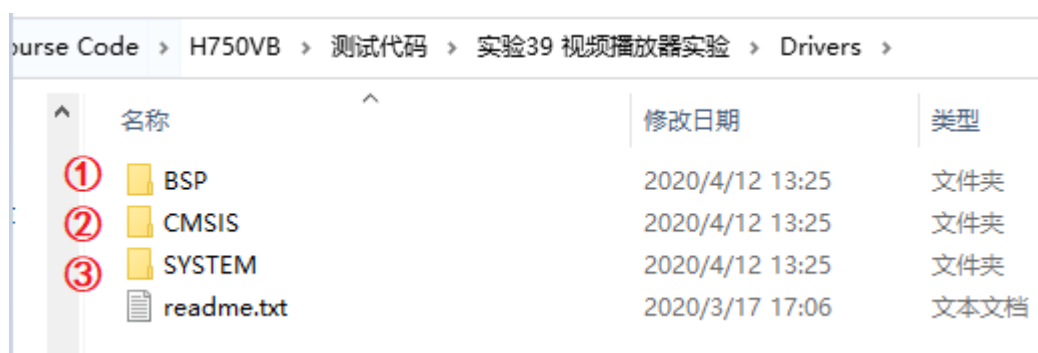
The screenshot shows the directory structure of an MDK project. The breadcrumb path is 'Source Code > H750VB > 测试代码 > 实验39 视频播放器实验'. The table lists the following items:

名称	修改日期	类型
Drivers	2020/4/12 13:25	文件夹
Middlewares	2020/4/12 13:25	文件夹
Output	2020/4/12 13:25	文件夹
Projects	2020/4/12 13:25	文件夹
User	2020/4/12 13:25	文件夹
keilkill.bat	2020/3/17 17:59	Windows 批处理...
readme.txt	2020/3/18 13:05	文本文档

图 2.1.1 MDK 工程结构（非 CubeMX 直接生成）

其中：

1，**Drivers** 文件夹，用于存放各种底层驱动代码，包括芯片厂家提供的驱动代码、ARM 提供的驱动代码、以及我们自己编写的底层驱动代码等三部分，如图 2.1.2 所示：



The screenshot shows the contents of the 'Drivers' folder. The breadcrumb path is 'Source Code > H750VB > 测试代码 > 实验39 视频播放器实验 > Drivers >'. The table lists the following items:

名称	修改日期	类型
① BSP	2020/4/12 13:25	文件夹
② CMSIS	2020/4/12 13:25	文件夹
③ SYSTEM	2020/4/12 13:25	文件夹
readme.txt	2020/3/17 17:06	文本文档

图 2.1.2 Drivers 文件夹内容

①，BSP 文件夹，相当于我们以前工程的 **HARDWARE** 文件夹，里面存放我们自己编写的各种底层驱动代码，如图 2.1.3 所示：

Code > H750VB > 测试代码 > 实验39 视频播放器实验 > Drivers > BSP		
名称	修改日期	类型
24CXX	2020/4/12 13:25	文件夹
ADC	2020/4/12 13:25	文件夹
BEEP	2020/4/12 13:25	文件夹
DAC	2020/4/12 13:25	文件夹
DCMI	2020/4/12 13:25	文件夹
DHT11	2020/4/12 13:25	文件夹
DMA	2020/4/12 13:25	文件夹
DS18B20	2020/4/12 13:25	文件夹
EXTI	2020/4/12 13:25	文件夹
IIC	2020/4/12 13:25	文件夹
JPEGCODEC	2020/4/12 13:25	文件夹
KEY	2020/4/12 13:25	文件夹

图 2.1.3 BSP 文件夹内容

- ②，CMSIS 文件夹，包含 ARM 提供基于 CMSIS 标准的底层代码，简单的说，包含我们新建工程需要的启动文件（.s 文件）和相关头文件（ARM 提供和 ST 独有两部分）等。该文件夹内容经过正点原子删减，只留下必要的文件和文件夹，其他非必需的全部删除了，从而避免工程过大的问题。

CMSIS 文件夹内容如图 2.1.4 所示：

B > 测试代码 > 实验39 视频播放器实验 > Drivers > CMSIS >	
名称	修改日期
Device	2020/4/12 13:25
Include	2020/4/12 13:25

图 2.1.4 CMSIS 文件夹内容

**Include** 文件夹，包含 ARM 提供 Cortex M 系列通用头文件（我们删减到只剩下当前版本芯片支持相关的头文件），以减少工程总大小。

**Device** 文件夹，包含 ST 公司具体某个 STM32 芯片（F1/F4/F7/H7 等）的头文件（路径：Device\ST\STM32H7xx\Include）、启动文件（路径：Device\ST\STM32H7xx\Source\Templates\arm）和 system\_stm32h7xx.c 文件（路径：Device\ST\STM32H7xx\Source\Templates）。同样，为了减少工程大小，我们也把不相关的.h、.s 文件等全部删除，只留下我们需要的文件。

注意：对于寄存器版本工程来说 system\_stm32h7xx.c 是可以不要的，HAL 库版本工程，则必须要这个文件。因此，为了兼容，我们将 system\_stm32h7xx.c 保留。

- ③，SYSTEM 文件夹，相当于我们以前的 SYSTEM 文件夹，里面存放我们自己编写的系统、延时、串口相关的代码，方便项目使用。如图 2.1.5 所示：

/B > 测试代码 > 实验39 视频播放器实验 > Drivers > SYSTEM		
名称	修改日期	类型
delay	2020/4/12 13:25	文件夹
sys	2020/4/12 13:25	文件夹
usart	2020/4/12 13:25	文件夹
readme.txt	2020/3/17 17:12	文本文档

图 2.1.5 SYSTEM 文件夹内容

这三个文件夹里面的代码全部按照《【正点原子】嵌入式单片机 C 代码规范与风格.pdf》重写。

**2, Middlewares 文件夹**, 用于存放各种中间层/组件/Lib 代码, 包括我们自己写的或者第三方提供的, 如 USMART、MALLOC、TEXT、PICTURE、FATFS、OS、GUI、USB 等, 如图 2.1.6 所示:

/B > 测试代码 > 实验39 视频播放器实验 > Middlewares		
名称	修改日期	类型
FATFS	2020/4/12 13:25	文件夹
MALLOC	2020/4/12 13:25	文件夹
MJPEG	2020/4/12 13:25	文件夹
PICTURE	2020/4/12 13:25	文件夹
TEXT	2020/4/12 13:25	文件夹
USMART	2020/4/12 13:25	文件夹
readme.txt	2020/3/17 16:59	文本文档

图 2.1.6 Middlewares 文件夹内容

注意: 为了统一风格, 我们在一些没有用到任何 Middlewares 内容的工程里面 (如 USMART 之前的例程), 也会预留 Middlewares 文件夹, 以统一风格, 并方便后续使用。

**3, Output 文件夹**, 相当于我们以前的 OBJ 文件夹, 用于存放各种编译中间文件及输出文件, 如图 2.1.7 所示:

Code > H750VB > 测试代码 > 实验39 视频播放器实验 > Output		
名称	修改日期	类型
atk_h750.axf	2020/4/12 20:05	AXF 文件
atk_h750.build_log.htm	2020/4/12 20:05	搜狗高速浏览器H...
atk_h750.hex	2020/4/12 20:05	HEX 文件
atk_h750.htm	2020/4/12 20:05	搜狗高速浏览器H...
atk_h750.lnp	2020/4/12 20:05	LNP 文件
atk_h750.map	2020/4/12 20:05	MAP 文件
atk_h750_MJPEG.dep	2020/4/12 20:05	DEP 文件

图 2.1.7 Output 文件夹内容

注意：在打包的时候可以通过运行 keilkill.bat 删除 Output 文件夹里面的绝大部分文件，以减少打包后的工程大小。

4, **Projects** 文件夹，用于存放工程文件，如图 2.1.8 所示：



名称	修改日期	类型
MDK-ARM	2020/4/12 20:05	文件夹
readme.txt	2020/3/17 17:37	文本文档

图 2.1.8 Projects 文件夹内容

该文件夹下面只包含一个 MDK-ARM 的文件夹，用于存放 MDK 版本工程文件，打开后如图 2.1.9 所示：



名称	修改日期	类型
DebugConfig	2020/4/12 20:05	文件夹
atk_h750.uvguix.Administrator	2020/4/12 20:05	ADMINISTRATO...
atk_h750.uvoptx	2020/4/5 15:11	UVOPTX 文件
atk_h750.uvprojx	2020/4/5 12:53	MDK5 Project
EventRecorderStub.scvd	2020/4/5 15:02	SCVD 文件

图 2.1.9 MDK-ARM 文件夹内容

其中：atk\_h750.uvprojx，就是 MDK5 的工程文件，DebugConfig 用于存放仿真调试配置信息，由 MDK 自动生成，我们无视他即可。

注意：.uvprojx 的命名方式我们统一使用 atk\_xxx 的命名方式，如 F103，我们就命名为：atk\_f103.uvprojx（注意，是在使用 MDK 新建工程的时候，确定命名的）。

5, **User** 文件夹，用于存放用户编写的代码（如果有分散加载，我们也把分散加载文件放这个文件夹下），如图 2.1.10 所示：



名称	修改日期	类型
APP	2020/4/12 13:25	文件夹
SCRIPT	2020/4/12 13:25	文件夹
main.c	2020/4/5 18:49	C 文件

图 2.1.10 User 文件夹内容

其中：

main.c，主要包含了 main 函数，相当于我们以前的 test.c 文件。

APP 文件夹，用于存放自己编写的应用代码（绝大部分基础例程代码并不多，只用 main.c 实现就够了，并不用 APP 文件夹），因为视频播放代码比较多，所以放在 APP 文件夹里面

(videoplayer.c)。该文件夹是可以删除的，并不一定要保留，只有需要的时候，才留下。

SCRIPT 文件夹，用于存放我们编写分散加载文件，仅 F750/H750 等开发板需要这个文件夹，其他开发板用不到分散加载的，都可以删了这个文件夹。



## 2.2 HAL 库版本工程结构

HAL 库版本新建工程，工程目录结构如图 2.2.1 所示：

工程 - HAL库版本 V4 > 新结构代码 > 实验13 TFTLCD (MCU屏) 实验 >		
名称	修改日期	类型
Drivers	2020/3/25 20:18	文件夹
Middlewares	2020/3/25 20:18	文件夹
Output	2020/3/26 20:20	文件夹
Projects	2020/3/25 20:18	文件夹
User	2020/3/25 20:25	文件夹
keilkill.bat	2020/3/18 20:10	Windows 批处理...
readme.txt	2020/3/18 20:07	文本文档

图 2.2.1 MDK 工程结构（非 CubeMX 直接生成）

可以看到，HAL 库版本的工程结构和寄存器版本完全一模一样，不过部分地方有差异，接下来仅对差异部分进行说明。

**1, Drivers 文件夹**，用于存放各种底层驱动代码，包括芯片厂家提供的驱动代码、ARM 提供的驱动代码、以及我们自己编写的底层驱动代码等三部分，如图 2.2.2 所示：

4 > 新结构代码 > 实验13 TFTLCD (MCU屏) 实验 > Drivers >		
名称	修改日期	类型
① BSP	2020/3/25 20:18	文件夹
② CMSIS	2020/3/25 20:18	文件夹
③ STM32H7xx_HAL_Driver	2020/3/25 20:18	文件夹
④ SYSTEM	2020/3/25 20:18	文件夹
readme.txt	2020/3/17 17:06	文本文档

图 2.2.2 Drivers 文件夹内容

- ①，BSP 文件夹，同寄存器版本类似，不过驱动是使用 HAL 库编写的。
- ②，CMSIS 文件夹，同寄存器版本完全一样。
- ③，STM32H7xx\_HAL\_Driver 文件夹，用于存放 ST 提供的 H7xx 系列芯片的 HAL 库驱动代码，如图 2.2.3 所示：

FTLCD (MCU屏) 实验 > Drivers > STM32H7xx_HAL_Driver		
名称	修改日期	类型
Inc	2020/3/25 20:18	文件夹
Src	2020/3/25 20:18	文件夹

图 2.2.3 STM32H7xx\_HAL\_Driver 文件夹内容

Inc 包含 H7xx HAL 库驱动的各种头文件。

Src 包含 H7xx HAL 库驱动的各种源码文件。

注意：寄存器版本代码，没有这个文件夹。

④，SYSTEM 文件夹，同寄存器版本类似，不过是 HAL 库版本。

2，Middlewares 文件夹，同寄存器版本一样。

3，Output 文件夹，同寄存器版本一样。

4，Projects 文件夹，同寄存器版本一样。

5，User 文件夹，用于存放用户编写的代码（如果有分散加载，我们也把分散加载文件放这个文件夹下），如图 2.2.4 所示：

新结构代码 > 实验13 TFTLCD (MCU屏) 实验 > User >		
名称	修改日期	类型
SCRIPT	2020/3/25 20:18	文件夹
main.c	2020/3/25 20:25	C 文件
stm32h7xx_hal_conf.h	2020/1/11 11:40	H 文件
stm32h7xx_it.c	2020/3/25 18:51	C 文件
stm32h7xx_it.h	2019/12/11 23:00	H 文件

图 2.2.4 User 文件夹内容

其中：

main.c，主要包含了 main 函数，相当于我们以前的 test.c 文件。

stm32h7xx\_hal\_conf.h，HAL 库配置头文件，可以开启/关闭相关功能参与编译。

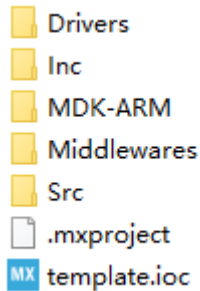
stm32h7xx\_it.c，存放中断服务函数。

stm32h7xx\_it.h，中断服务头文件。

SCRIPT 文件夹，用于存放我们编写分散加载文件，仅 F750/H750 等开发板需要这个文件夹，其他开发板用不到分散加载的，都可以删了这个文件夹。

### 第三章、使用 CubeMX 的工程结构

当我们全部使用 CubeMX 生产工程的时候（即自己在建工程了，直接用 CubeMX 生成的工程），接下来，我们看看使用 CubeMX 生成的工程结构。



```
zhang@DESKTOP-78HS3G4:~/studyboard-tutorial/template$ tree -L 2
.
├── Drivers
│   ├── CMSIS
│   └── STM32F1xx_HAL_Driver
├── Inc
│   ├── FreeRTOSConfig.h
│   ├── main.h
│   ├── stm32f1xx_hal_conf.h
│   └── stm32f1xx_it.h
├── MDK-ARM
│   ├── RTE
│   ├── startup_stm32f103xe.s
│   ├── template
│   ├── template.uvoptx
│   └── template.uvprojx
├── Middlewares
├── Src
│   ├── freertos.c
│   ├── main.c
│   ├── stm32f1xx_hal_msp.c
│   ├── stm32f1xx_it.c
│   └── system_stm32f1xx.c
└── template.ioc

9 directories, 13 files
```

我们以前的例程相比，修改点有如下几点：

- 1，启动文件改为存放到 MDK-ARM 下
- 2，用户源码都放在了 Src 文件夹下
- 3，用户头文件放在 Inc 文件夹下
- 4，驱动都放在了 Driver 下

- ├── Drivers 存放板级驱动，默认包含 HAL 库及 CMSIS（与用户配置相关）
  - | ├── CMSIS
  - | └── STM32F1xx\_HAL\_Driver
- ├── Inc 存放外设，gpio 初始化，中间件相关头文件
  - | ├── main.h
  - | ├── stm32f1xx\_hal\_conf.h
  - | └── stm32f1xx\_it.h
- ├── Src 存放外设，gpio 初始化，中间件相关源码及 main.c
  - | ├── freertos.c
  - | ├── main.c
  - | ├── stm32f1xx\_hal\_msp.c
  - | ├── stm32f1xx\_it.c
  - | └── system\_stm32f1xx.c
- ├── MDK-ARM 存放 MDK 相关文件及启动文件
  - | ├── RTE
  - | ├── startup\_stm32f103xe.s
  - | ├── template
  - | ├── template.uvoptx
  - | └── template.uvprojx
- ├── Middlewares 存放中间件源码
- └── template.ioc cubemx 工程文件

## 第四章、MDK 工程说明

本章分为以下三个部分：

- 1， 寄存器版本工程说明
- 2， HAL 库版本工程说明
- 3， CubeMX 版本工程说明

接下来，我们分别介绍这三种情况。

## 4.1 寄存器版本工程说明

寄存器版本 MDK 工程打开后如图 4.1.1 所示：

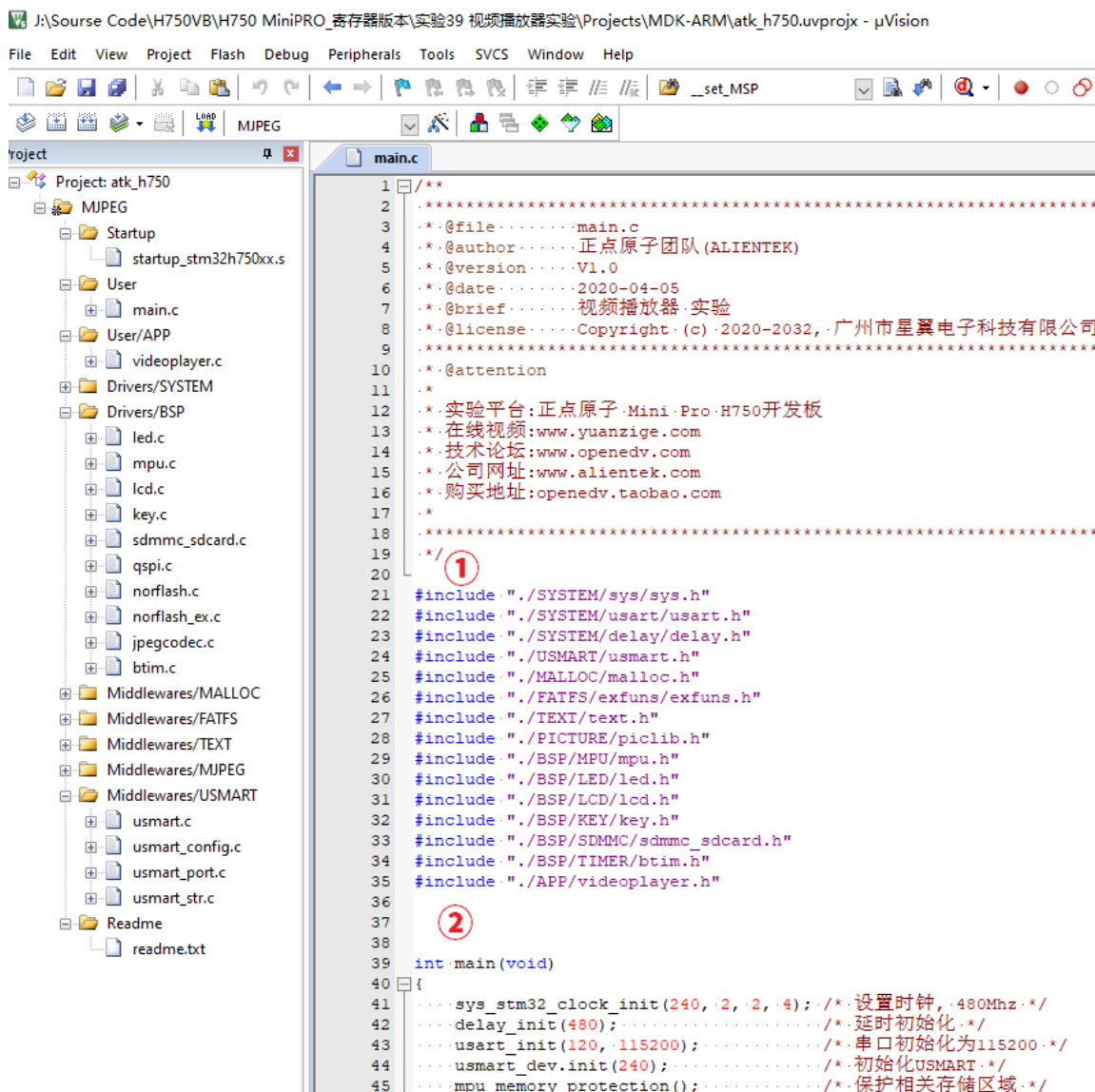


图 4.1.1 寄存器版本 MDK 工程

① 处，文件注释和头文件之间，有且只空一个空行。

② 处，头文件和代码之间，有且只空两个空行。

另外，器函数之间，有且只空一格空行。对于.h文件，宏定义之间，不的空行规定，但是建议大家根据自己的需要，按功能区分，可以适当加一些空行，不过，我们规定：任何两段程序段（宏定义/全局变量/函数申明等）之间的空行，不应该超过3个。

如果还不清楚的，可以参考最新版本的代码。

回到代码，同我们以前的例程相比，修改点有如下几点：

1，工程名字由原来的 test 修改为：atk\_h750，以便区分当前工程对应的芯片系列。

注意：针对不同的 STM32 型号，需要用不同的名字，比如：F103 用 atk\_f103；F407 用 atk\_f407；F767 用 atk\_f767。

2，项目目标 (Project Targets) 由原来默认的 Target 1 改为具体的例程名字简写，比如 LED、

KEY、USART、EXTI、BTIM、GTIM、ATIM、LCD、MJPEG 等，便于区分例程功能。

3，启动文件改为存放到 Startup 组下，替换原来的 Source Group1 组，提高易读性。

4，使用 main.c 替换原来的 test.c，更规范。

5，所有的源码都按前三章所说的方式分类放置，体现在 MDK 工程里面，我们使用 1/2 级目录分组形式来将源码添加到工程里面，如：

User	一级目录分组
User/APP	二级目录分组
Drivers/SYSTEM	二级目录分组
Middlewares/MALLOC	二级目录分组
Middlewares/USB_CORE	二级目录分组（下划线方式）

使用这种路径分组形式，跟容易知道源码是存放在哪里的，方便使用。

注意：路径一般按实际路径填写（大小写和实际的名字一样），一般不超过 2 级。

6，Include 一般使用相对路径（自己编写的代码），免去频繁的添加头文件路径操作。如：

```
#include "../SYSTEM/sys/sys.h"
#include "../SYSTEM/usart/usart.h"
#include "../SYSTEM/delay/delay.h"
#include "../USMART/usmart.h"
#include "../MALLOC/malloc.h"
#include "../FATFS/exfuns/exfuns.h"
```

其中：.表示当前目录，..表示上一级目录。

不过，即便使用相对路径，我们还得添加第一级文件夹的路径，否则 include 的路径就要很长，不利于编写代码。第一级路径添加如图 4.1.2 所示：

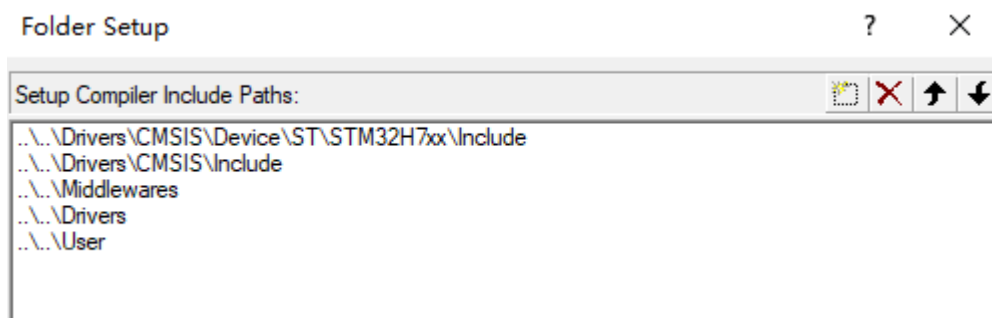


图 4.1.2 添加第一级头文件包含路径

注意，我们自己编写的代码，一般使用相对路径即可，但是对于一些第三方的代码，可以不使用相对路径，避免改动第三方代码/包含太长的相对路径，如上图中的：

```
..\..\Drivers\CMSIS\Device\ST\STM32H7xx\Include
..\..\Drivers\CMSIS\Include
```

我们把 CMSIS 里面的头文件全路径添加了，这样在包含 CMSIS 里面的头文件的时候，我们不需要使用相对路径，直接 include 即可，如：

```
#include "stm32h7xx.h"
```

而不要添加太长的相对路径，也不需要去改动库函数。

7，使用空格替代 TAB 键，并显示空格出来，方便检查，避免使用他软件打开源码不对齐的问题。代码对齐和注释对齐全部采用新的 TAB 方式（空格替代）对齐。

TAB 键使用空格替代，通过 MDK 的 Configuration 选项卡配置，如图 4.1.3 所示：

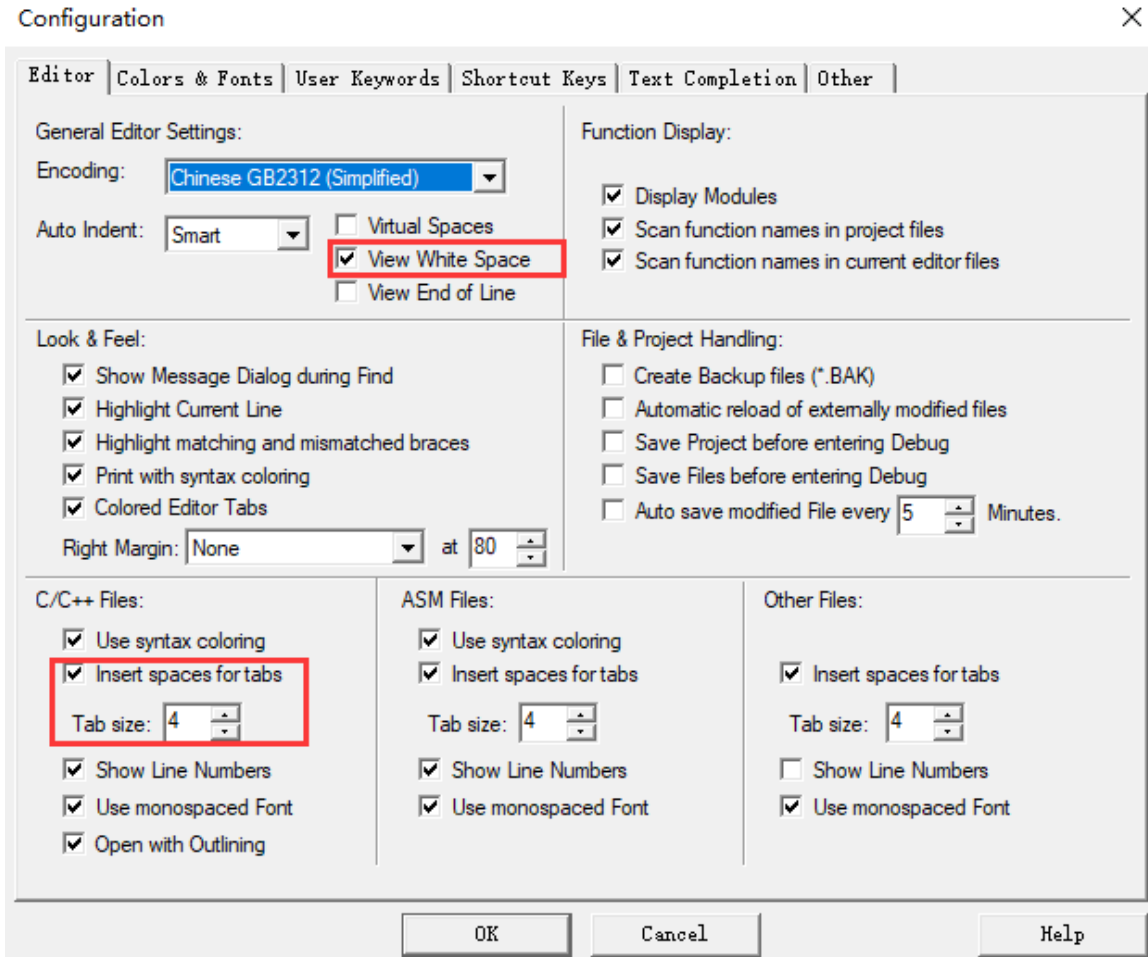


图 4.1.3 配置 TAB 键用空格（4 个）替代且显示空格

8，注释风格全部使用 /\* ... \*/，详见：【正点原子】嵌入式单片机 C 代码规范与风格.pdf。

9，编译输出全部放到 Output 文件夹，替代原来的 OBJ 文件夹。

10，MDK 版本使用最新的 MDK5.29（如果后续再出新的也可以用更新的版本）。



## 4.2 HAL 库本工程说明

同寄存器版本。

### 4.3 CubeMX 版本工程说明

CubeMx MDK 工程打开后如图 4.3.1 所示：

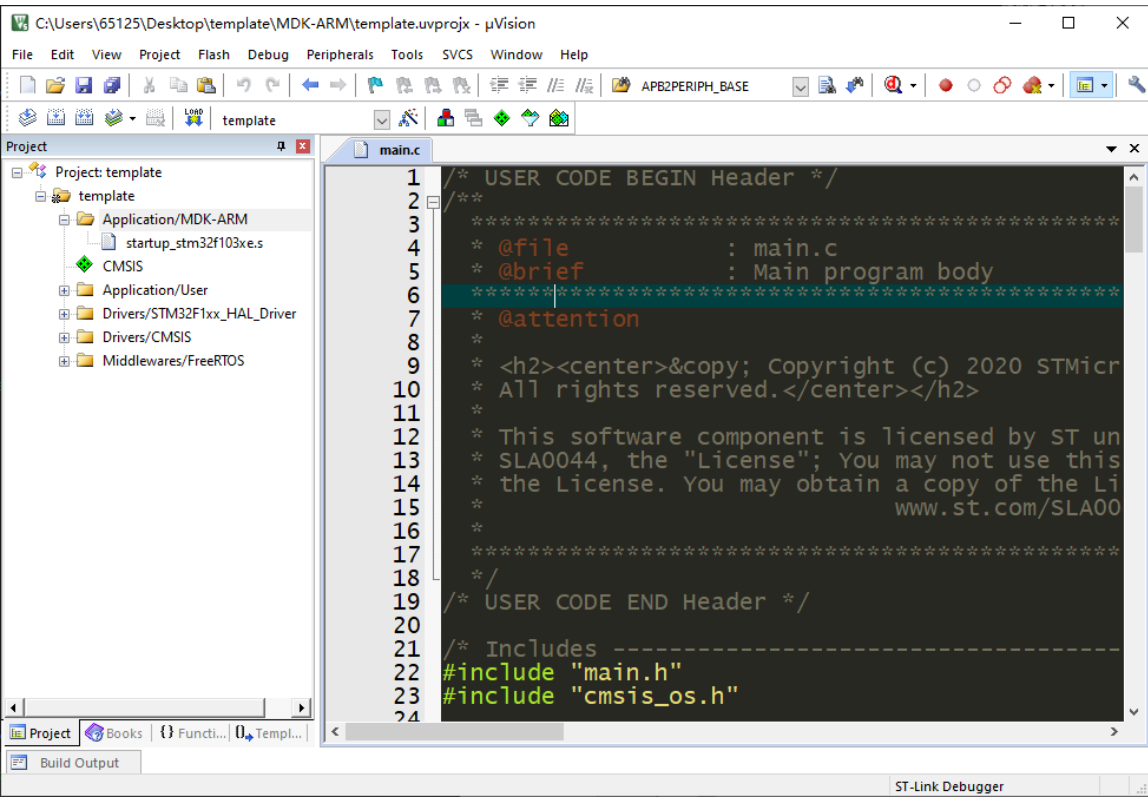
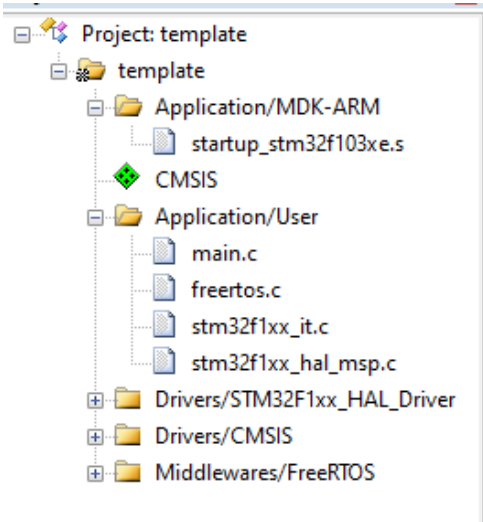


图 4.3.1 CubeMx 版本 MDK 工程

在 MDK 中目录结构如图



不同之处：

- 1, 工程名字由 cube mx 中配置，source group 与工程同名
- 2, 所有的源码都按前三章所说的方式分类放置，体现在 MDK 工程里面，我们使用 1/2 级目录分组形式来将源码添加到工程里面，如：

Drivers/SYSTEM                      二级目录分组  
Middlewares/MALLOC                二级目录分组

Middlewares/USB\_CORE            二级目录分组（仅部分例程有，如 USB、OS 等）  
使用这种路径分组形式，跟容易知道源码是存放在哪里的，方便使用。  
注意：路径一般按实际路径填写（大小写和实际的名字一样），一般不超过 2 级。

### 3， CubeMx 工程头文件引用并未使用相对路径

当 cubemx 工程建立后，不允许修改工程名字及存放路径。你把他移到其他地方照样在原路径生成。

编写代码必须遵从 CubeMx 的规则，若在规定区域之外编写，重新自动生成将会被清除。

User sections **shall neither be moved nor renamed**. Only the user sections defined by STM32CubeMX are preserved. **User created sections will be ignored and lost at next C code generation.**

只有cube mx生成的USER CODE 区域能添加，自行添加USER CODE区域无效！且不能移动！

CubeMX工程生成文件中包括：

```
/* USER CODE BEGIN 0 */  
(..)  
/* USER CODE END 0 */  
/* USER CODE BEGIN 1 */  
(..)  
/* USER CODE END 1 */  
.....
```

但是如果自行把这段复制然后想在自动生成的代码其他地方加东西，无效，照样清除！