芯海科技
CHIPSEA
股票代码:688595

# 芯海通用 MCU 基于 IAR 芯片包 IAR9 开发指南

**V1.0 版本**

涉密等级：公开

芯海科技（深圳）股份有限公司
www.chipsea.com
sales@chipsea.com
+86-0755-8616 9257
518000

## 摘要

芯海科技与 IAR Systems 达成合作，IAR Embedded Workbench for Arm 已全面支持芯海科技 CS32F103X、CS32F03X、 CS32L010X 等系列 MCU 芯片。芯海科技基于 IAR 环境开发芯片包，给用户更多的编译选择，提高用户开发灵活性。本应用笔记从芯片包结构、DEMO 例程下载使用、工程建立等角度出发，详细介绍 IAR 芯片包的使用方法，旨在帮助指导用户针对芯海通用 MCU 基于 IAR 环境进行快速建立应用工程，快速开发，并针对常见错误问题，给出解决办法。

## 版本

| 历史版本 | 修改内容 | 日期 |
|---|---|---|
| V1.0 | 初版生成 | 2022-12-01 |

# 目 录

IAR Embedded Workbench 是嵌入式开发中比较常用的一种 IDE 工具，该工具链包括高度优化的编译器以及高级调试功能，IAR 功能丰富，易上手。本应用笔记将结合芯海科技 AIR 芯片包开发包(以 CS32L010F6 为例)对 IAR 的操作进行简单讲解。

芯海科技 AIR 芯片包开发包是基于 IAR 9.3 版本开发，用户应使用不低于此版本的 IAR。

## 1. IAR 芯片包开发包的目录结构

IAR 已将芯海科技的芯片包集成到最新的 IAR Embedded Workbench 中，用户将 IAR Embedded Workbench 更新到最新的版本即可，或者用户将“ChipSea.CS32L010_DFP1.1.0.5.zip”解压，并按如下目录分别拷贝复制文件夹。

在 IAR Embedded Workbench 的安装目录下(“IAR Systems\Embedded Workbench 9.x\arm\config”)有如下文件目录：

```
|——ARM
    |——Debugger
    |   |——ChipSea                    ; CS32L010F6.ddf/CS32L01x.dmac/CS32L010x.svd
    |——Devices
    |   |——ChipSea
    |       |——CS32L01                ; CS32L010F6.i79/CS32L010F6.menu
    |——Flashloader
        |——ChipSea
            |——FlashCS32L010x         ; FlashCS32L010x6x6.board/.flash/.out
    |——Linker
        |——ChipSea                    ; CS32L010F6.icf
|——INC
   |——ChipSea                         ; iocs32l01x.h
```

注意：IAR Embedded Workbench 下载的 Examples 工程存放在“ProgramData\IARSystems\EmbeddedWorkbench\DownloadedExamples\arm\9.30.1\ChipSea”目录下，因此用户自行解压的“ChipSea.CS32L010_DFP1.1.0.5.zip”中的 Examples 也需要存放在其目录下。

# 2. DEMO 下载及使用

## 2.1. 下载 DEMO

DEMO 不跟随 IAR 软件安装，用户需要在 IAR 自行下载，操作步骤如下：打开 IAR 软件，点击 Help,下拉菜单中选择 Information Center 选项，打开信息中心。

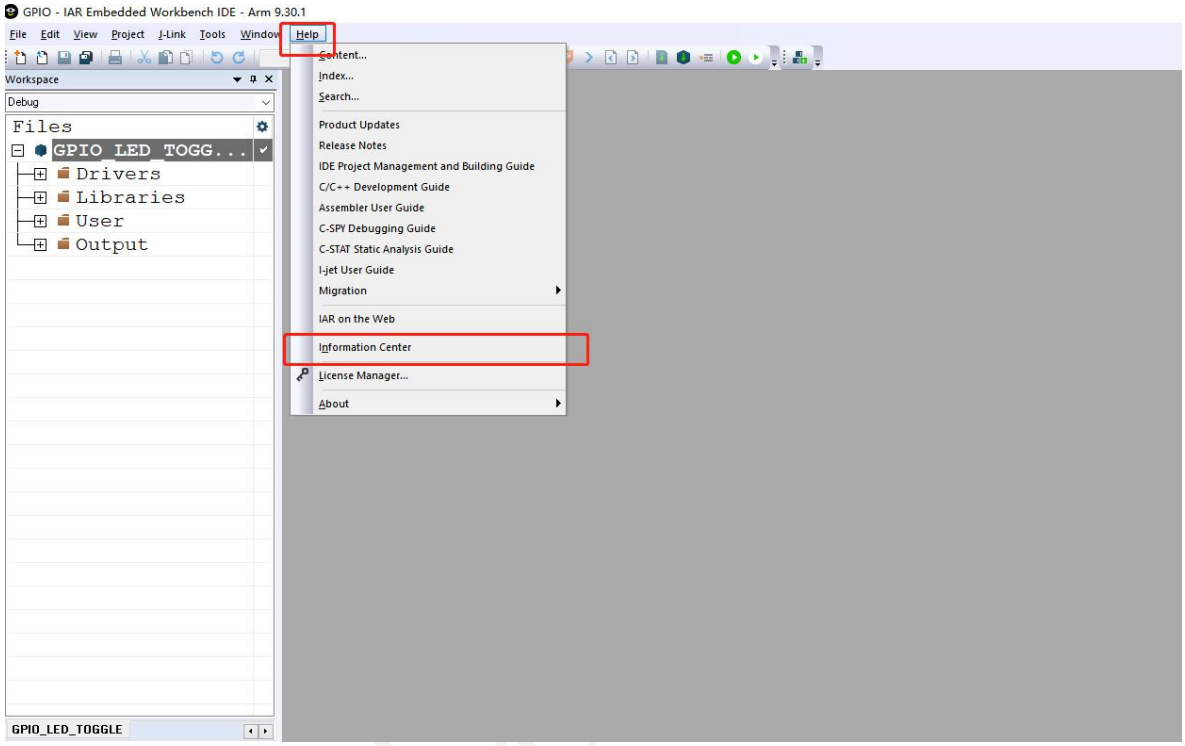



**图 2.1 打开信息中心**

## 2.2. 点击【Example projects】 菜单项

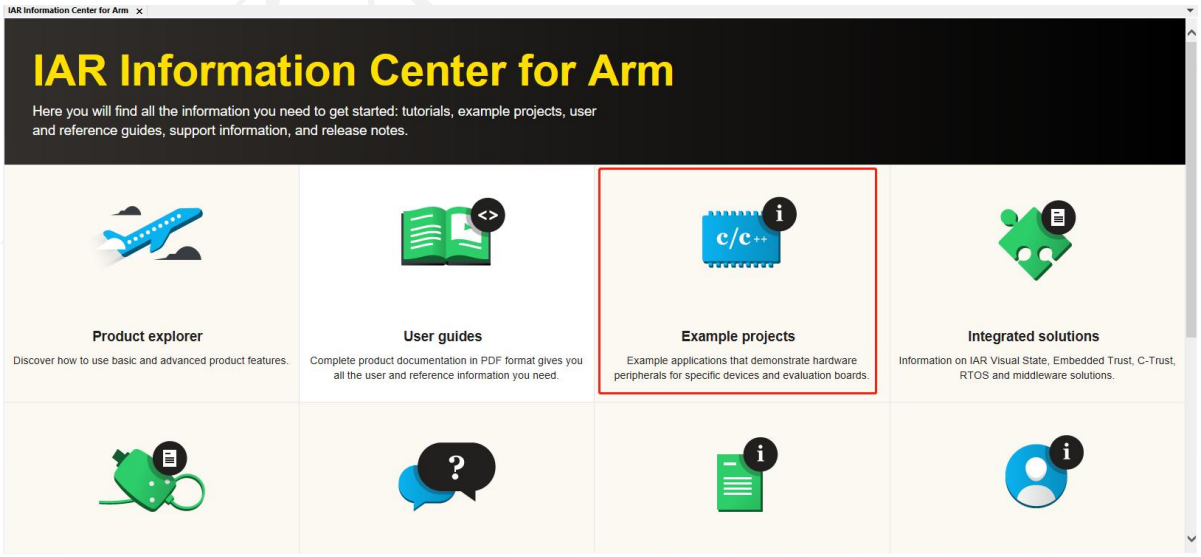



**图 2.2 菜单项**

## 2.3. 选择 ChipSea Example projects 下载

打开【Example projects】后，在【Example projects that can be downloaded】查找选择 ChipSea，点击下载按钮，开始下载，已下载的 DEMO 会显示在此页面。


**图 2.3.1 下载 DEMO**

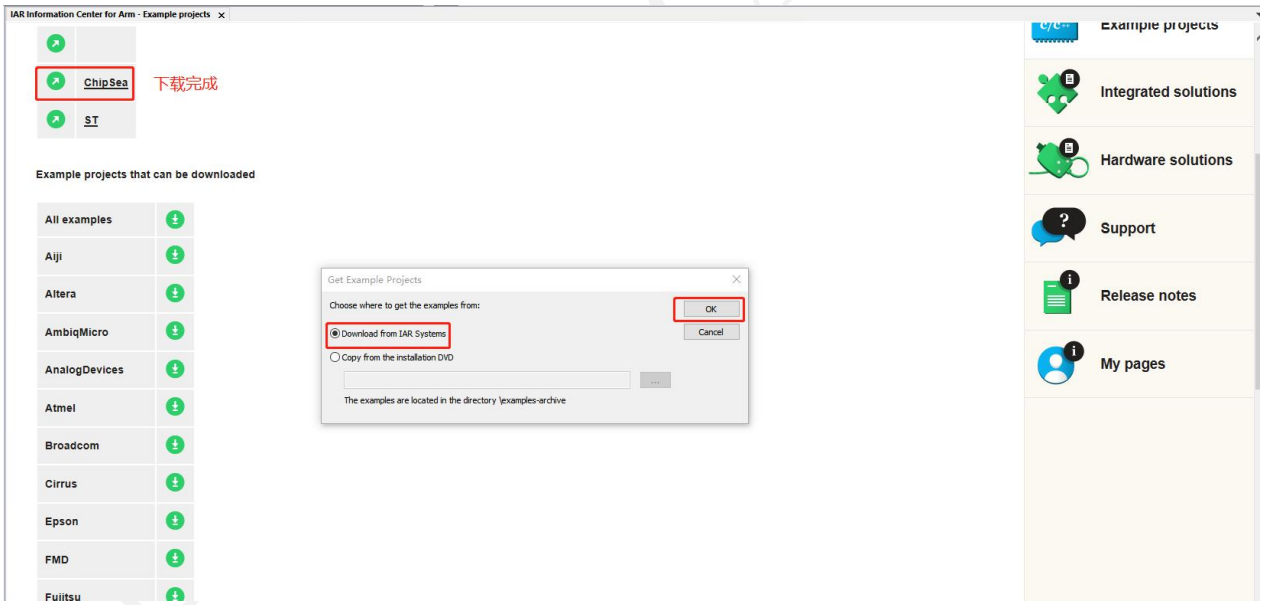



**图 2.3.2 下载完成**

下载成功的 DEMO 可以在当前页面打开工程，操作步骤如下图所示，首先在【Installed example projects】中选 ChipSea，然后如下图一直点击进入，找到合适的 DEMO 打开即可。也可以在下载的根目录打开，首先打开 IAR 目录“ProgramData\IARSystems\EmbeddedWorkbench\DownloadedExamples\arm\9.30.1\ChipSea”，选择 CS32L010F6 找到需要的 DEMO，例如 GPIO，打开 GPIO 文件夹再打开 GPIO.eww 便可。

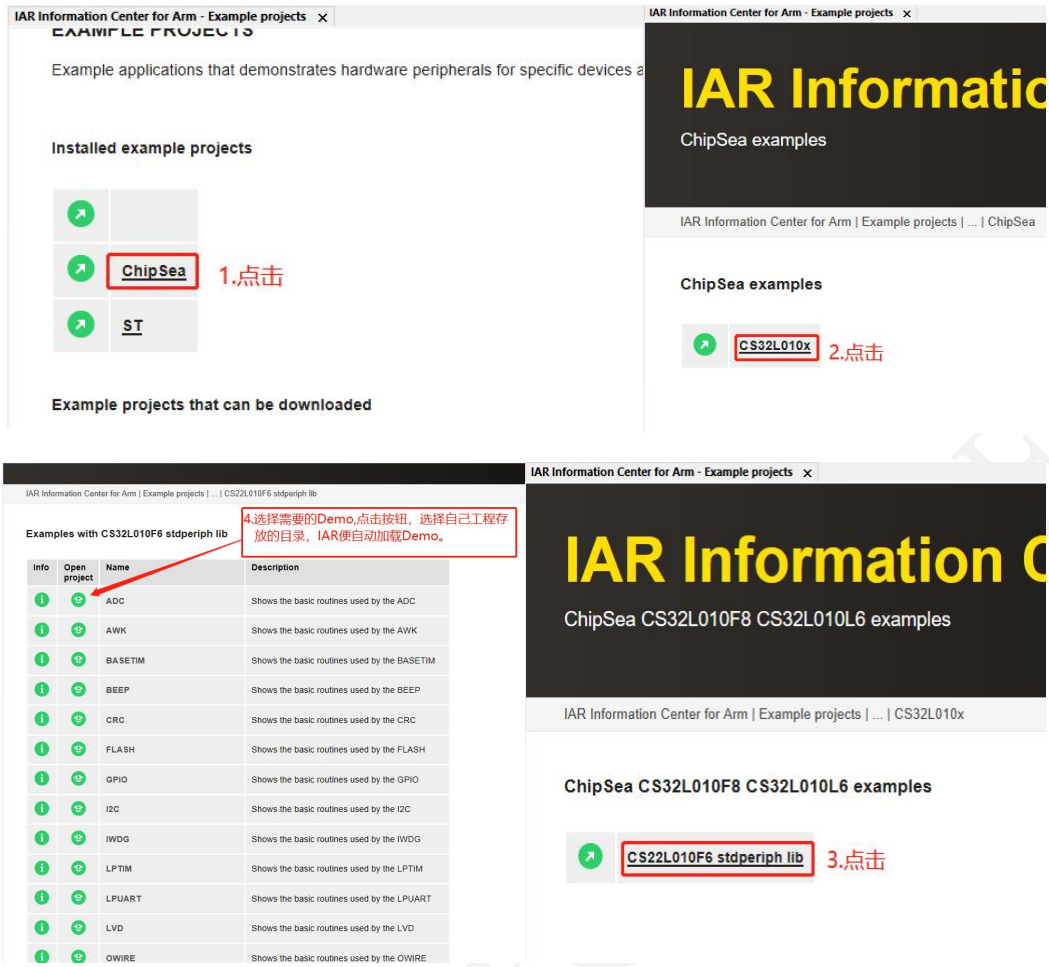



**图 2.3.3 DEMO 打开步骤**

## 2.4. DEMO 目录结构及编译烧录

以 CS32L010F6 为例介绍 DEMO 目录结构：

```
|——CS32L010F6
    |——Common                    ;公共文件如 LED/USART 初始化配置。
    |——Libraries
    |    |——CMSIS
    |         |——CM0             ; 内核文件
    |              |——Startup      ; 启动文件
    |    |——CS32L01x_Driver
    |              |——Inc          ; 驱动头文件
    |              |——Src          ; 驱动文件
    |——Project
```

|——DEMO　　;DEMO 例程

DEMO 打开后，可直接编译，烧录到开发板中，无需用户配置。DEMO 中例程用户可根据自身需要，自行修改。打开 DEMO 后如图所示：

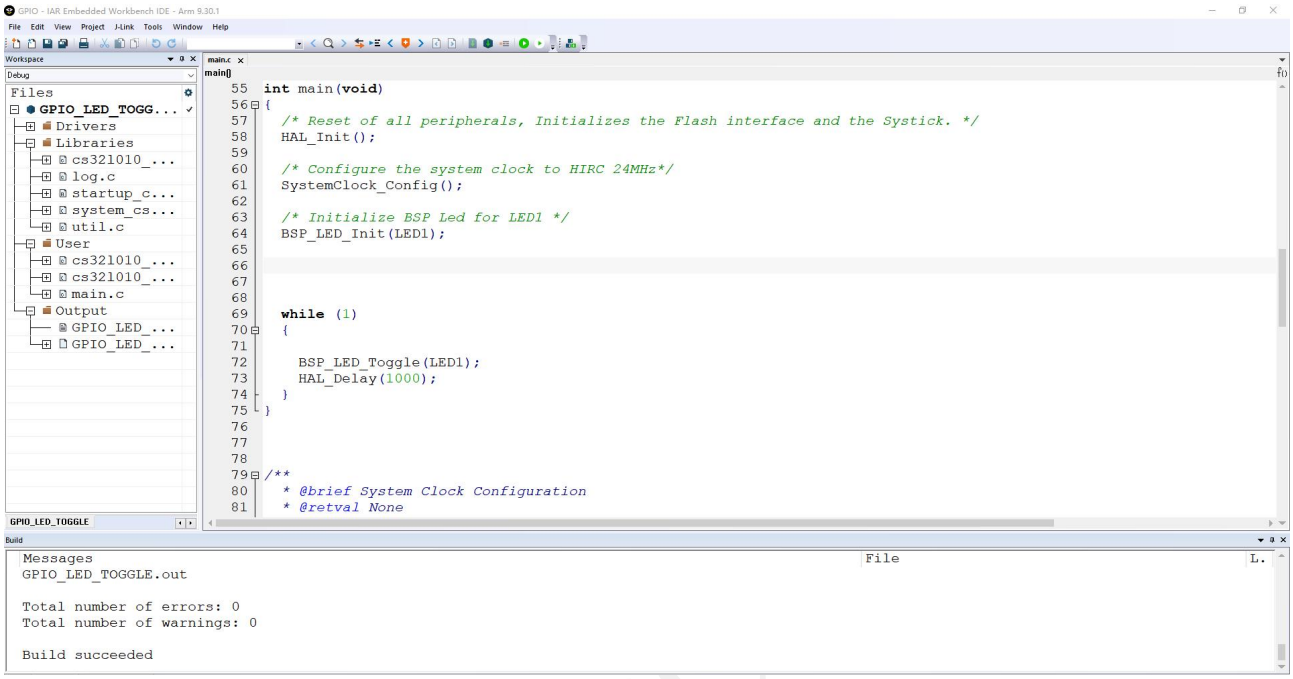



**图 2.4 DEMO 样例**

# 3. 创建工程及配置

## 3.1. 创建新工程

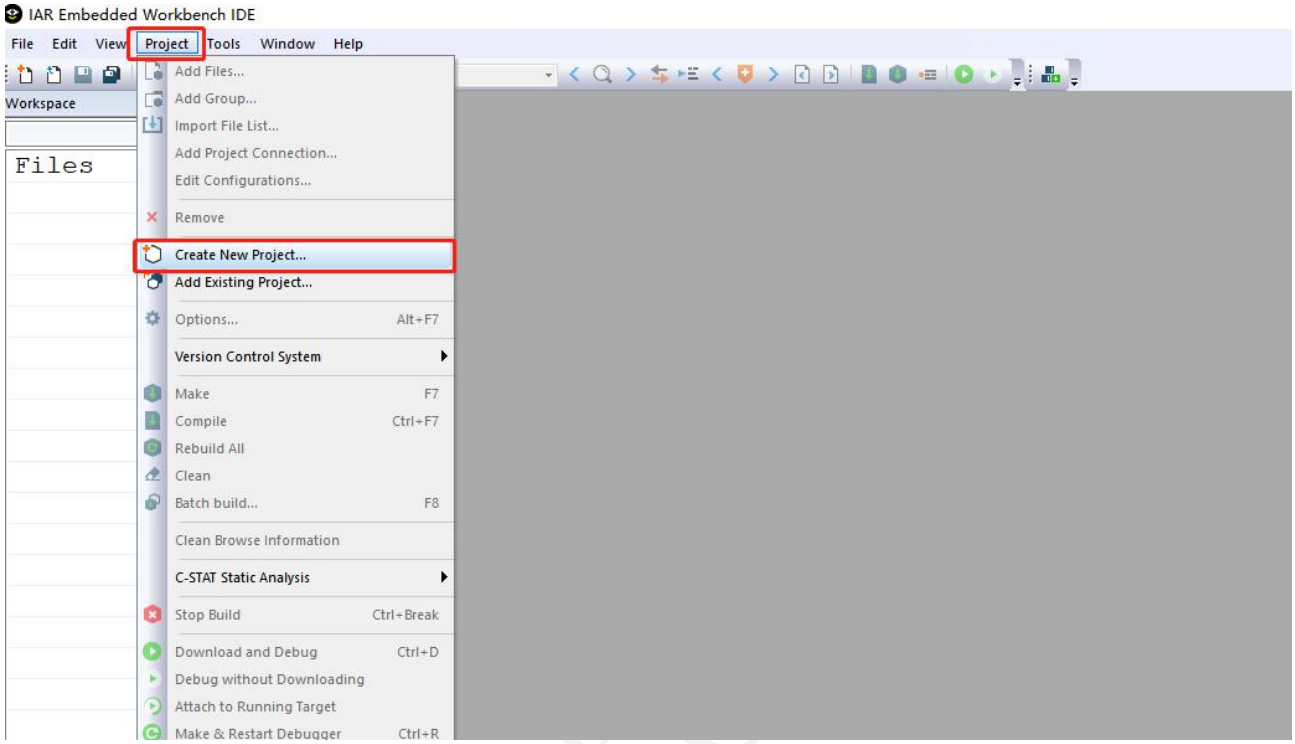



图 3.1 创建工程

## 3.2. 选择【Empty Project】

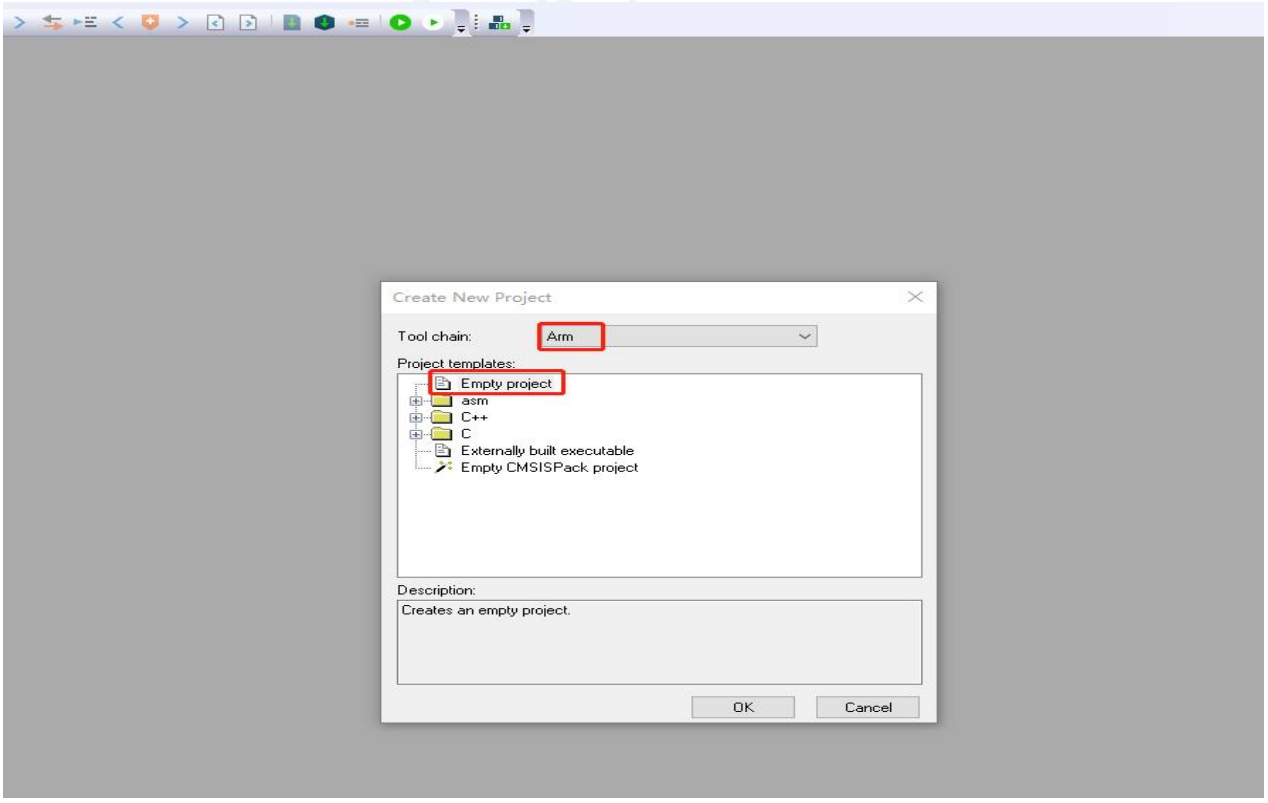



图 3.2 选择工程类型

### 3.3. 选择保存路径

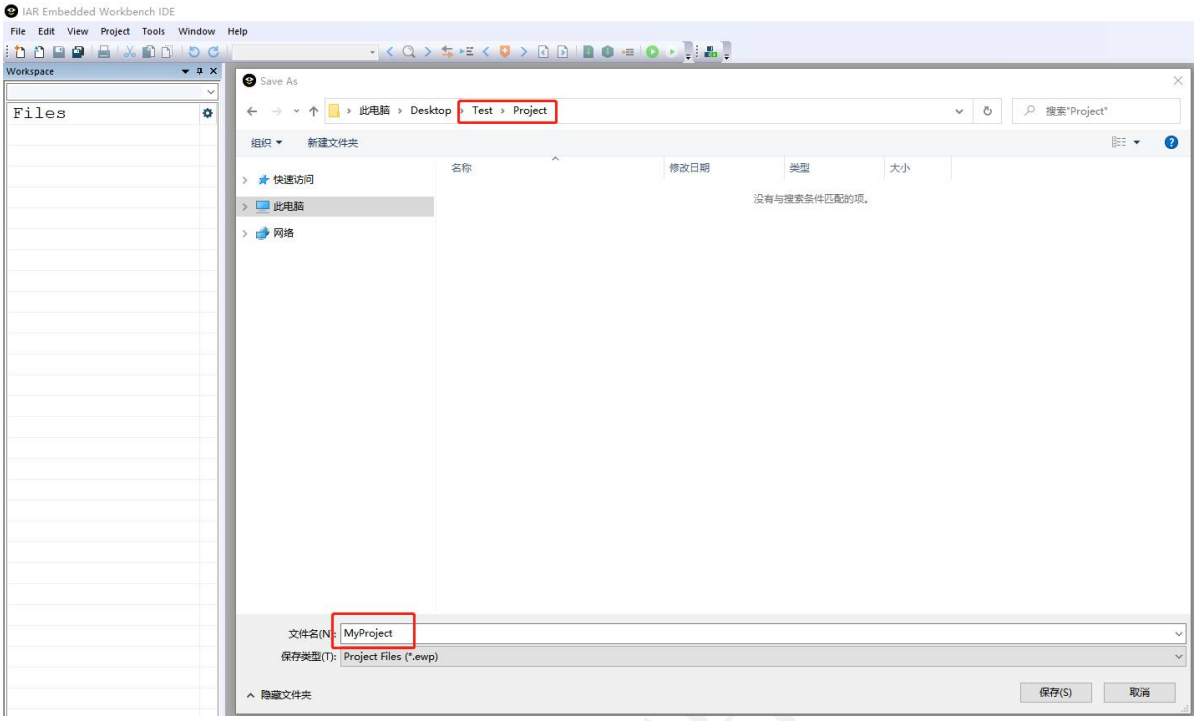



图 3.3 选择路径

### 3.4. 复制启动文件、SDK 库文件到 Test 工程

从“ChipSea.CS32L010_DFP1.1.0.5.zip”文件或者下载的 DEMO 文件中复制 Common 或者 Libraries 文件夹到 Test 文件夹中，DEMO 下载方式已在 2.1 节中介绍，目录结构已在本应用笔记 2.4 节中介绍。

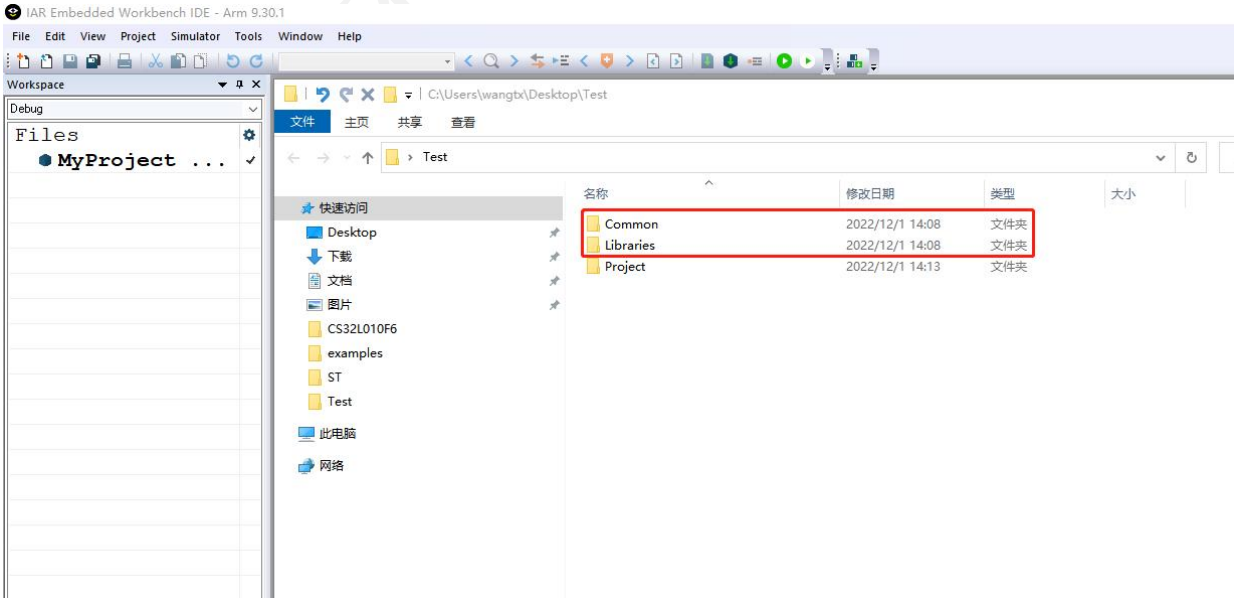



图 3.4 复制文件

## 3.5. 创建分组，加入代码文件

用户可自行定义分组数量和组名，本应用笔记以 DEMO 目录为例，介绍如何添加组、添加文件等操作步骤。

### 3.5.1 添加组与定义组名

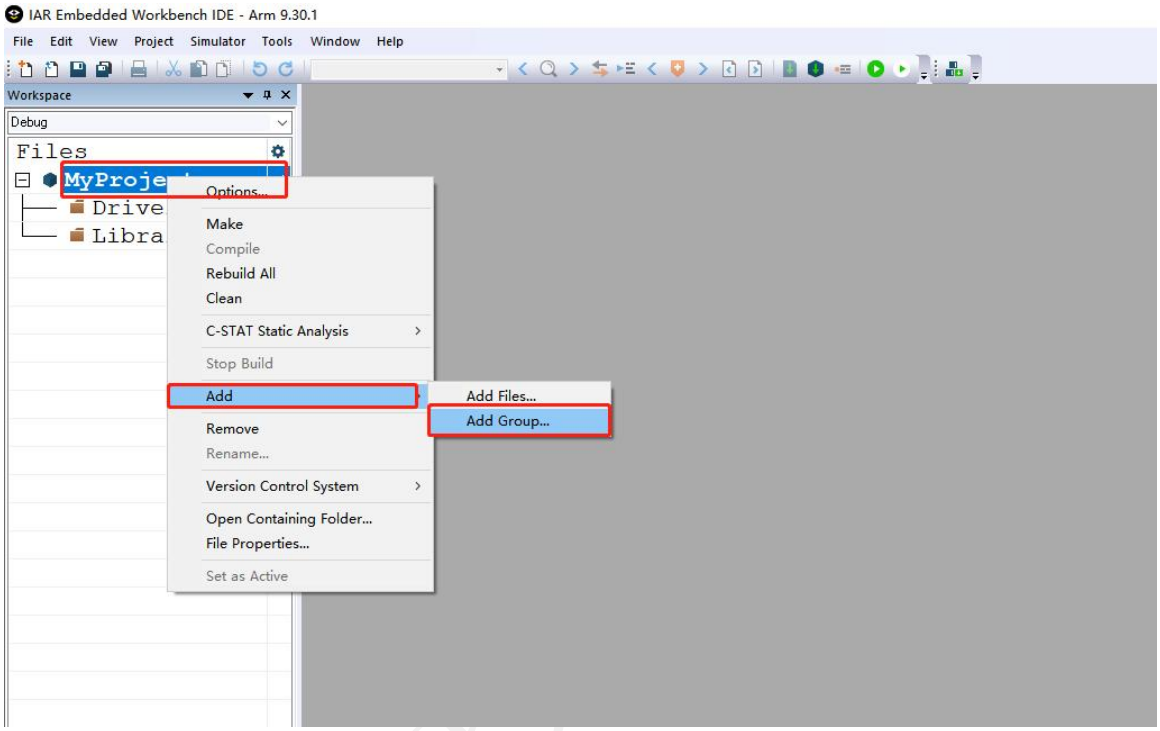



图 3.5.1 添加组名

### 3.5.2 添加 User 组，以用来添加 main.c 等用户自己的文件

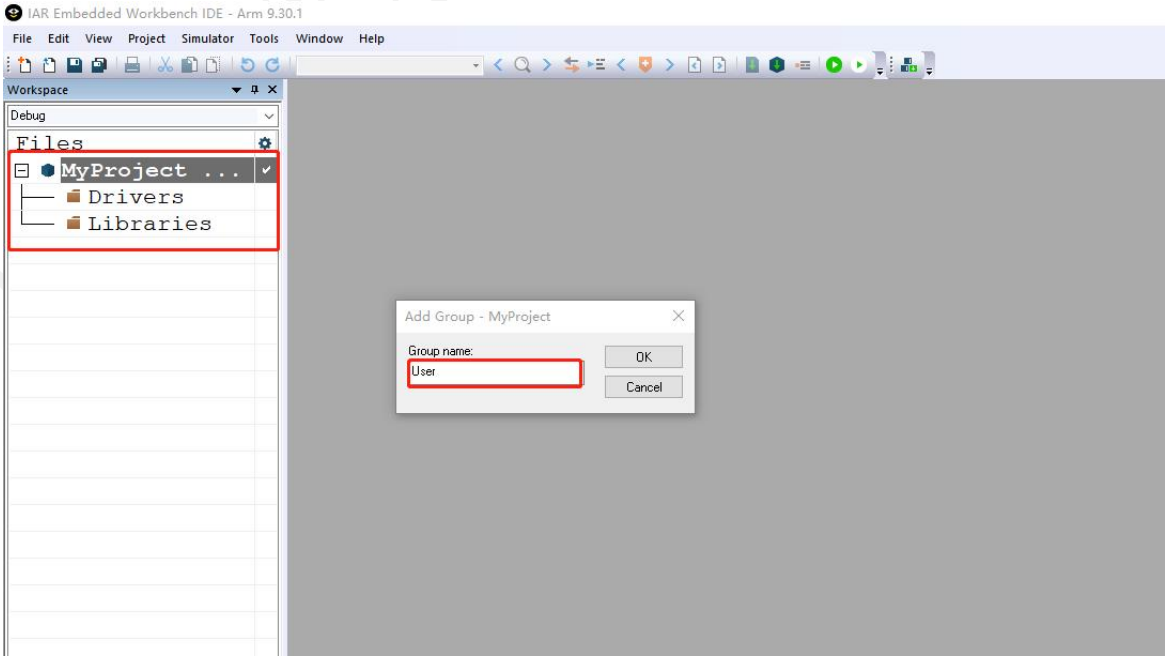



图 3.5.2 定义组名

### 3.5.3 添加启动文件、头文件、驱动文件等

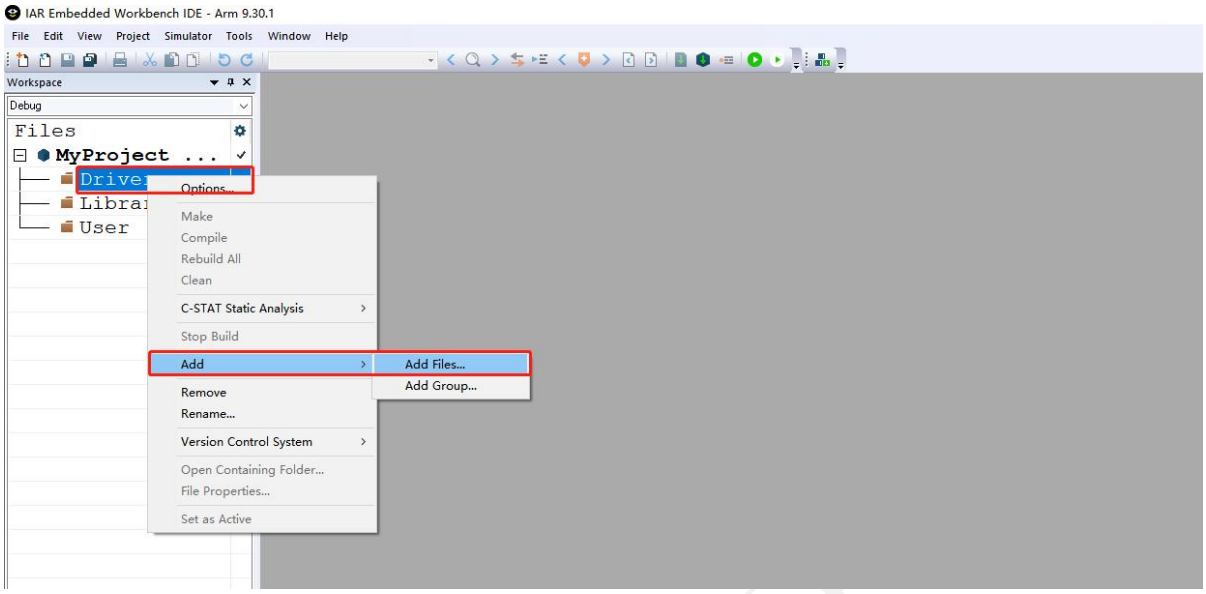



图 3.5.3 添加文件

### 3.5.4 创建 mian.c、中断处理函数等添加到 User

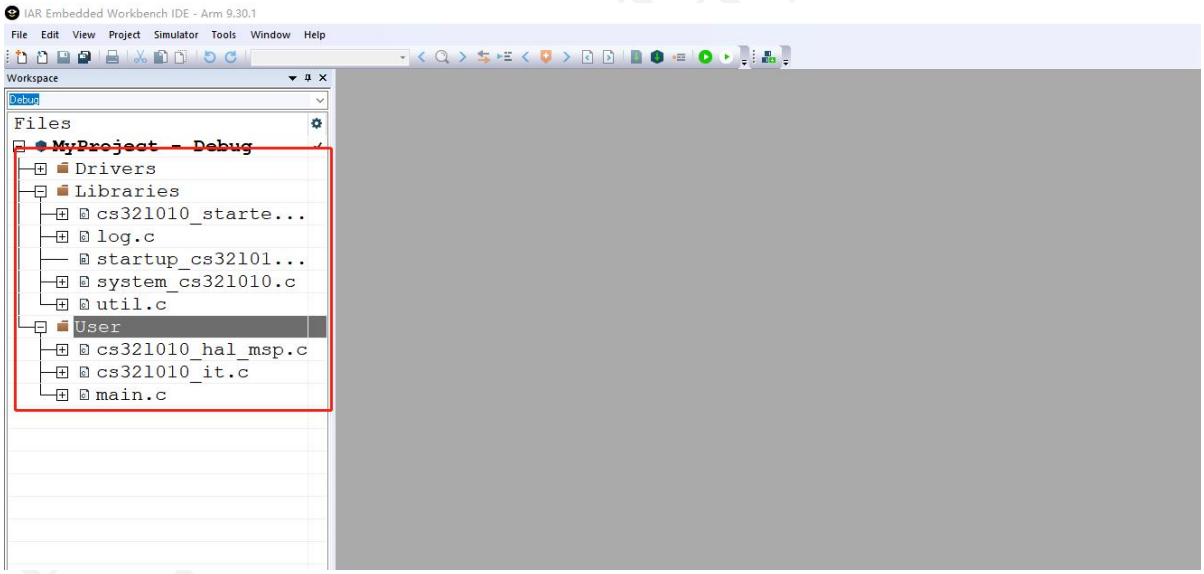



图 3.5.4 添加创建的文件

## 3.6. 选择芯片 CS32L010F6

点击 MyProject，点击鼠标右键选择 option。

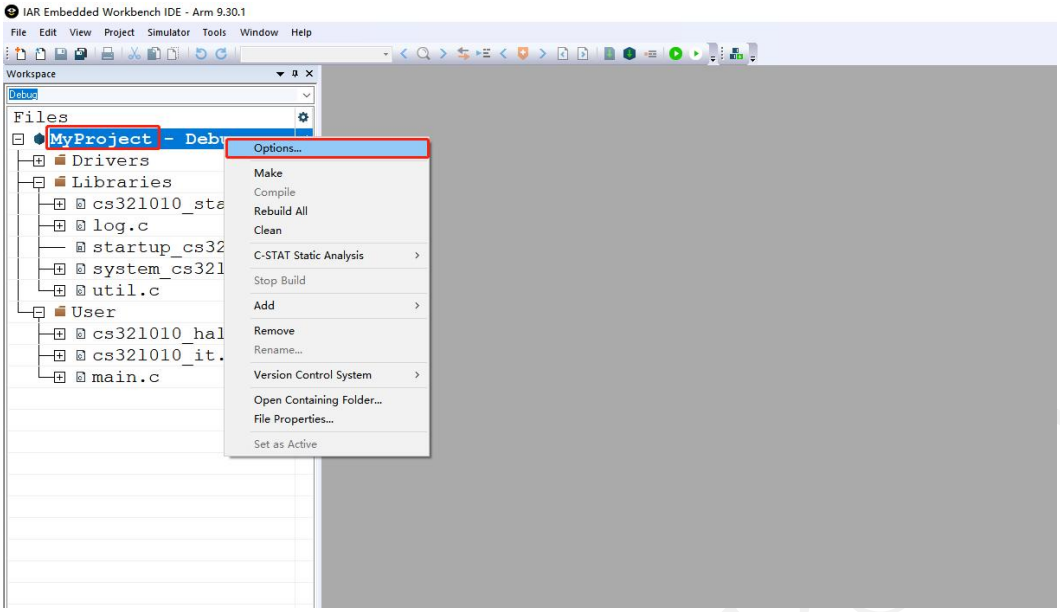



**图 3.6.1 选择 Option**

在打开的菜单中，选择 Device，找到 ChipSea，选择 CS32L010F6，IAR 会自动加载 CS32L010F6 的配置文件。

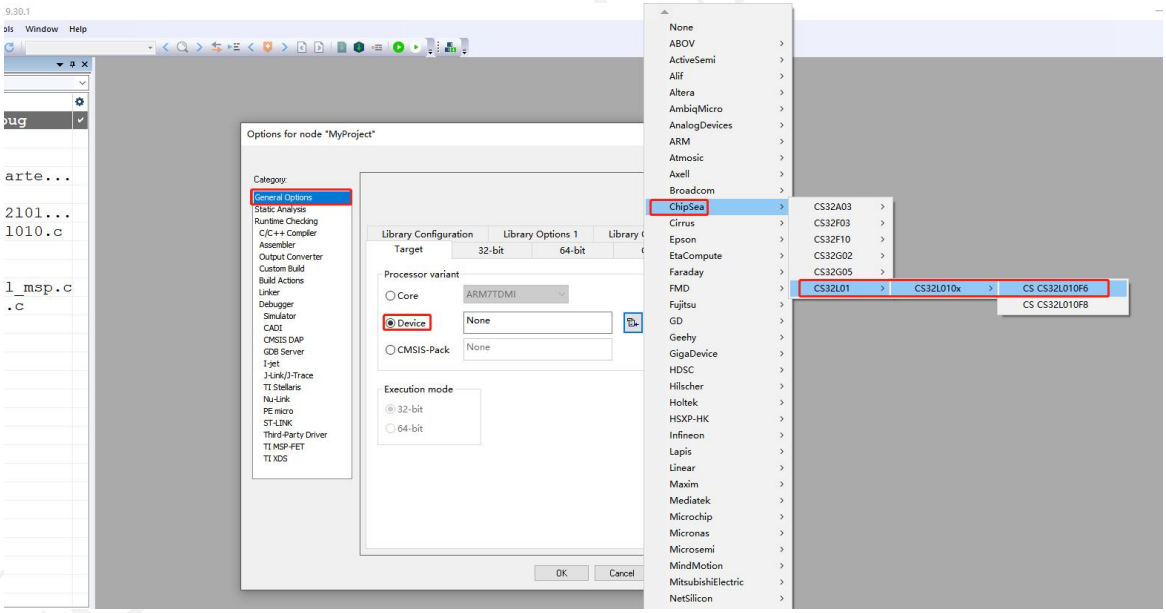



**图 3.6.2 选择 Device**

## 3.7. Option 选项配置

这一步是配置文件路径、工程设置和查看所选芯片是否自动加载相应配置文件。

### 3.7.1 通用设置，设置选项用红框表示

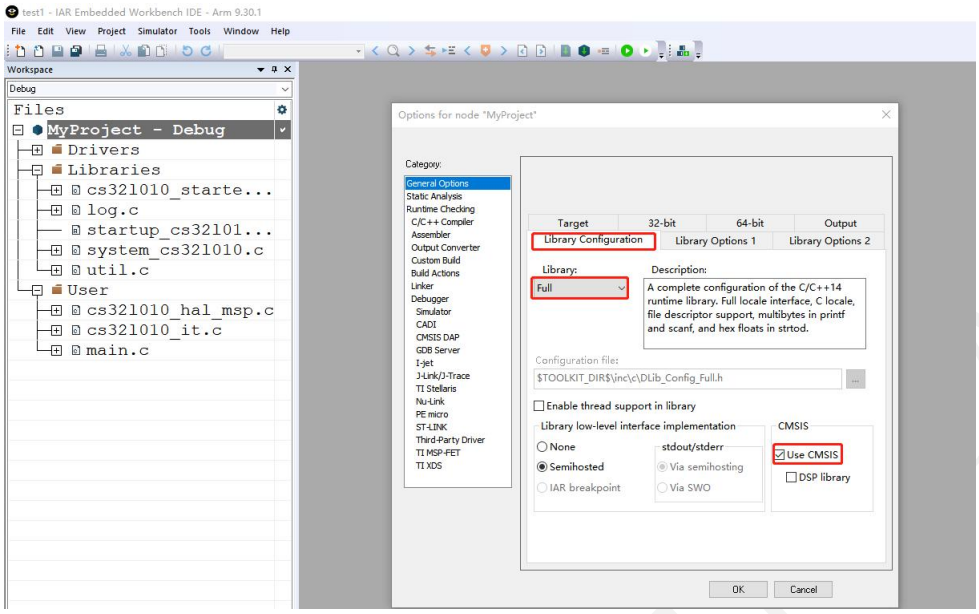



图 3.7.1 通用设置

### 3.7.2 修改 Debug 文件生成路径

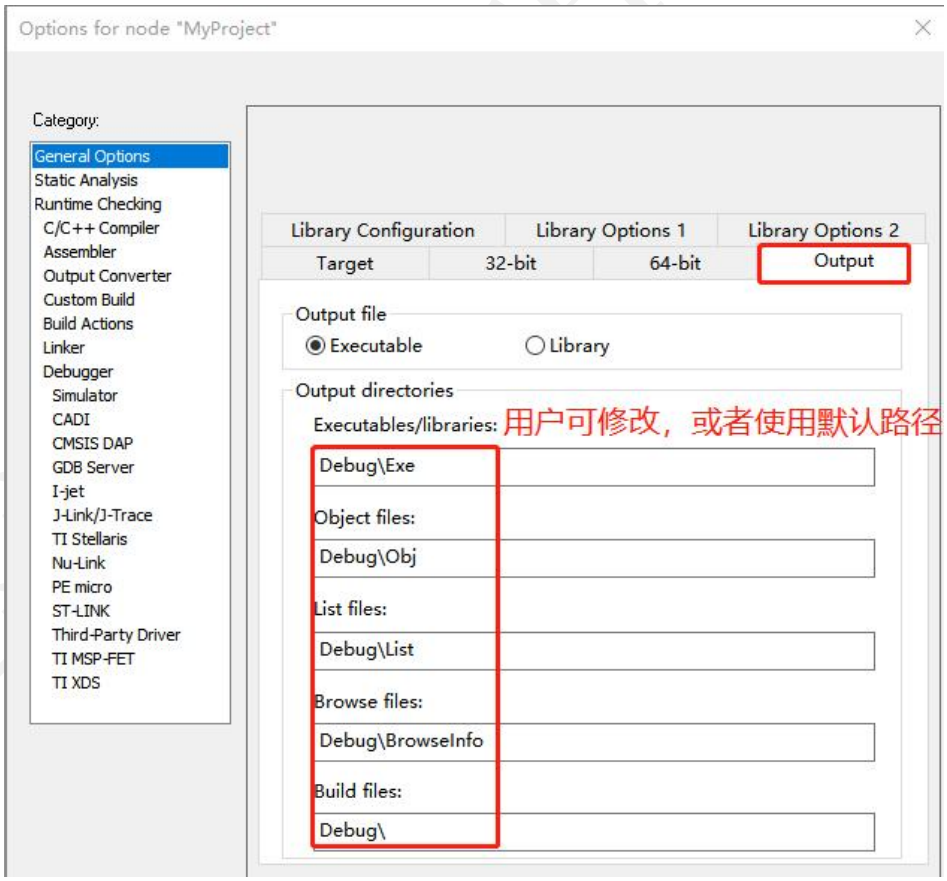



图 3.7.2 修改 Debug 路径

### 3.7.3 头文件路径选择，全局宏定义添加，优化等级

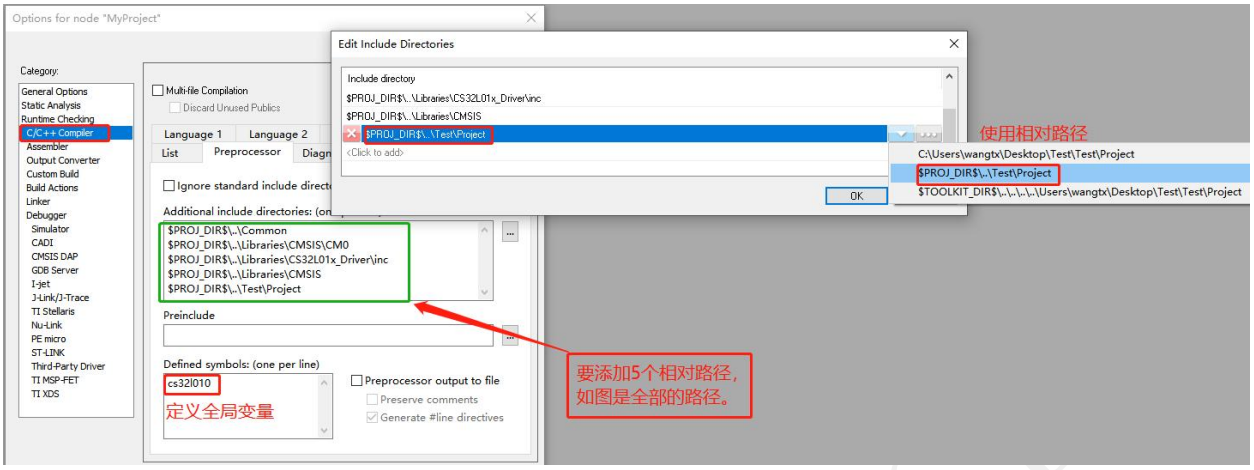



图 3.7.3.1 添加路径

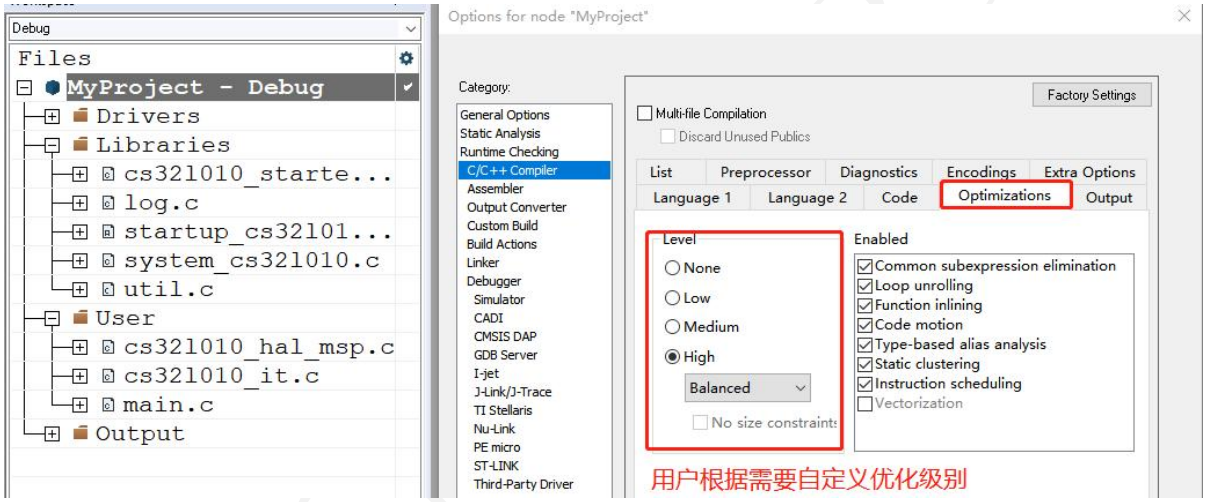



图 3.7.3.2 选择优化等级

### 3.7.4 添加 Lib 库文件

非必须，如 CS32G020 有的驱动封装为 Lib 库，则需要添加。

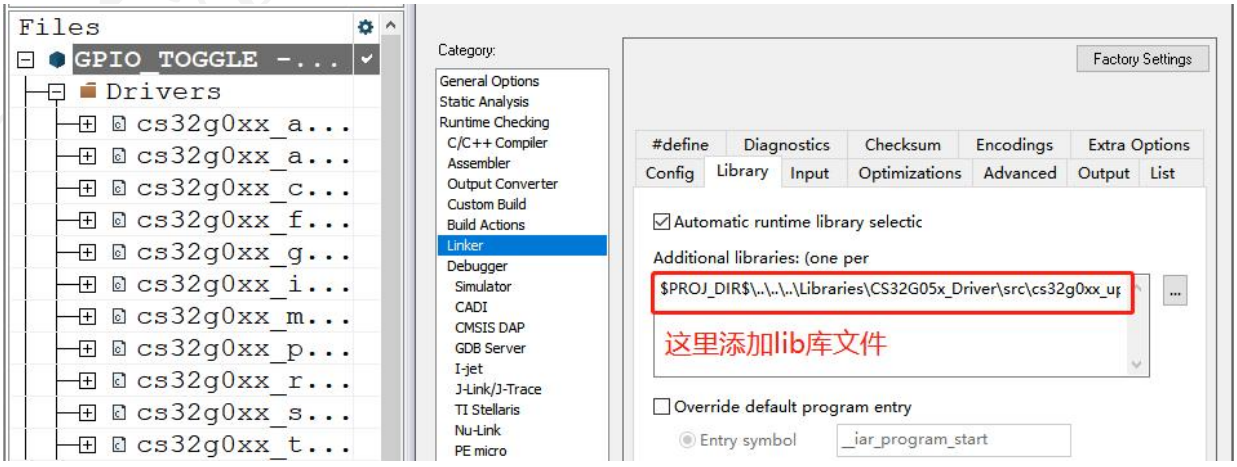



图 3.7.4 添加 Lib 库

### 3.7.5 程序编译

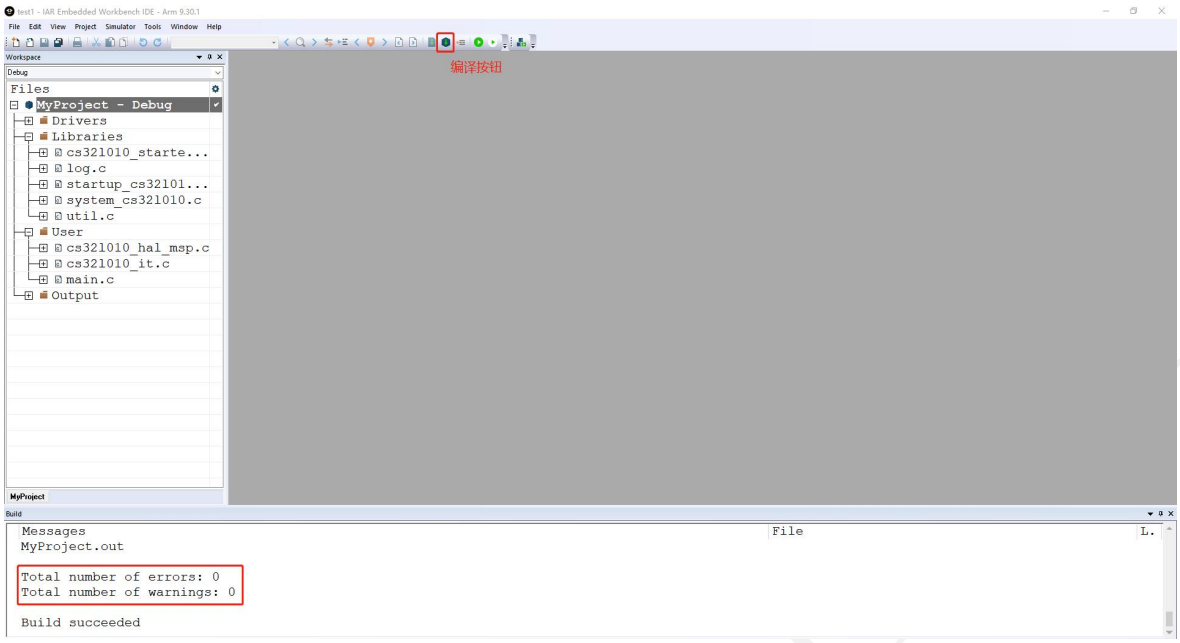
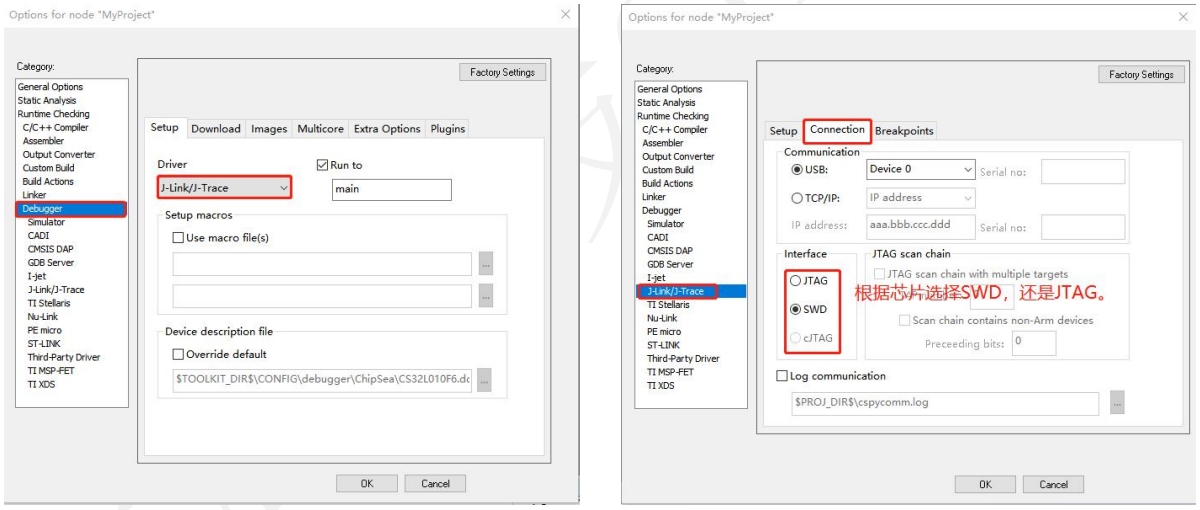



图 3.7.5 编译完成

## 3.8 烧录设置




(a)　　(b)

图 3.8 烧录设置

## 3.9 JLINK 设置

点击 Debug，弹出 Jlink 设置界面，选择 M0+内核。

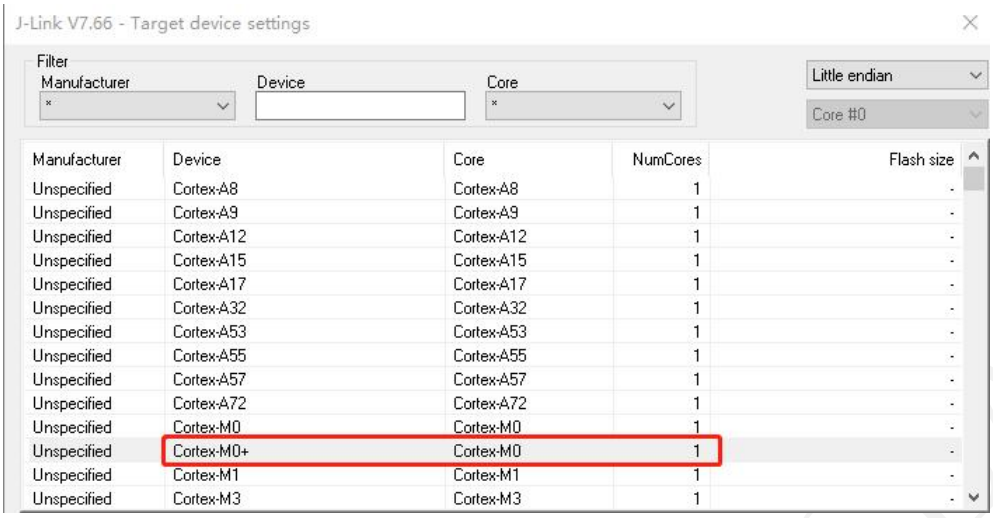



**图 3.9.1 选择内核**

开始 Debug，程序可以正常打断点。

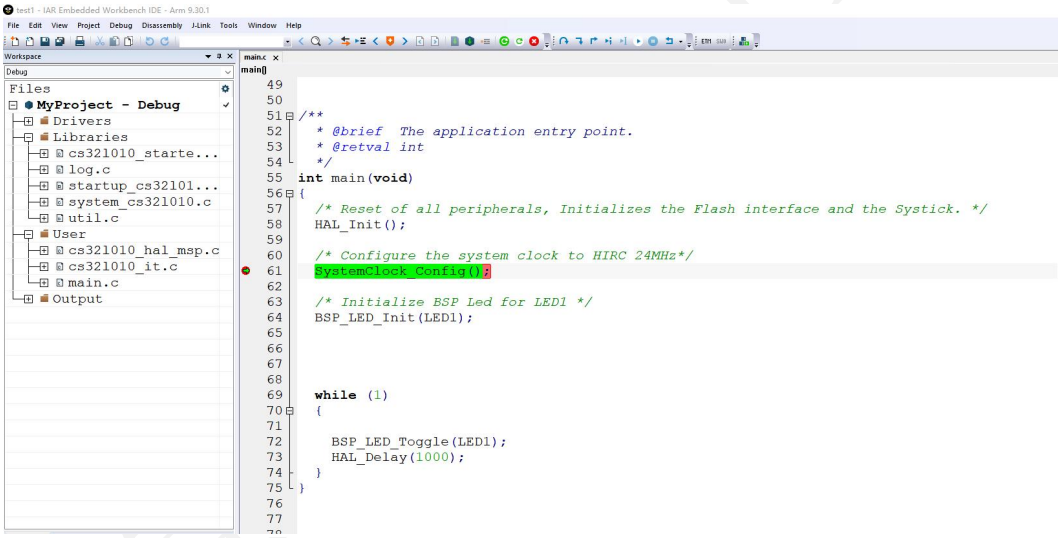



**图 3.9.2 打断点**

# 4. 配置文件说明及常见错误解决

## 4.1 ICF 文件

ICF 文件为 IAR 编译配置文件, 其中参数用户均可自行修改，用户即可在 IAR 中修改，也可直接在根目录直接修改.icf 文件，根目录为 IAR 安装目录 linker/ChipSea。

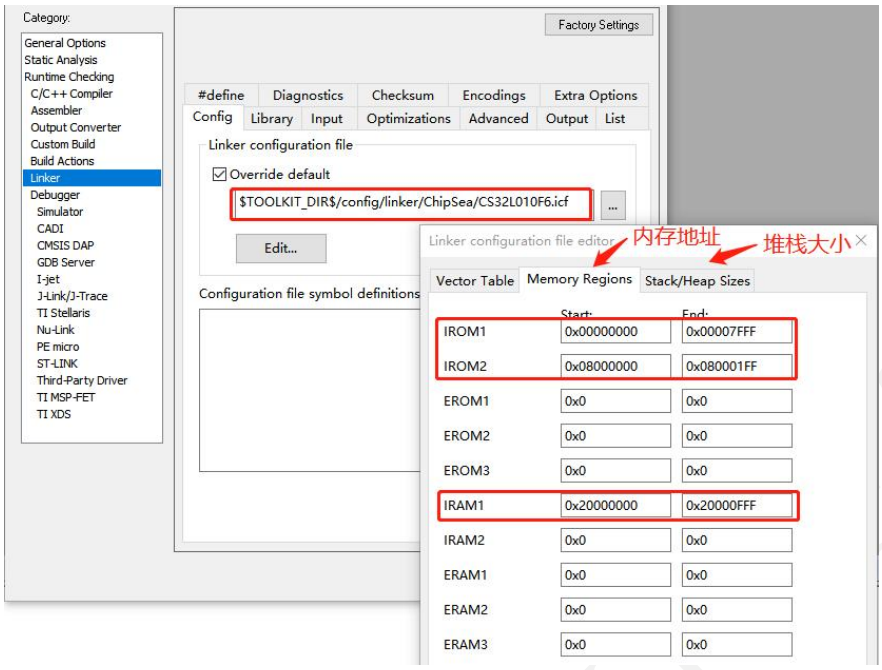



**图 4.1.1 查看内存值**

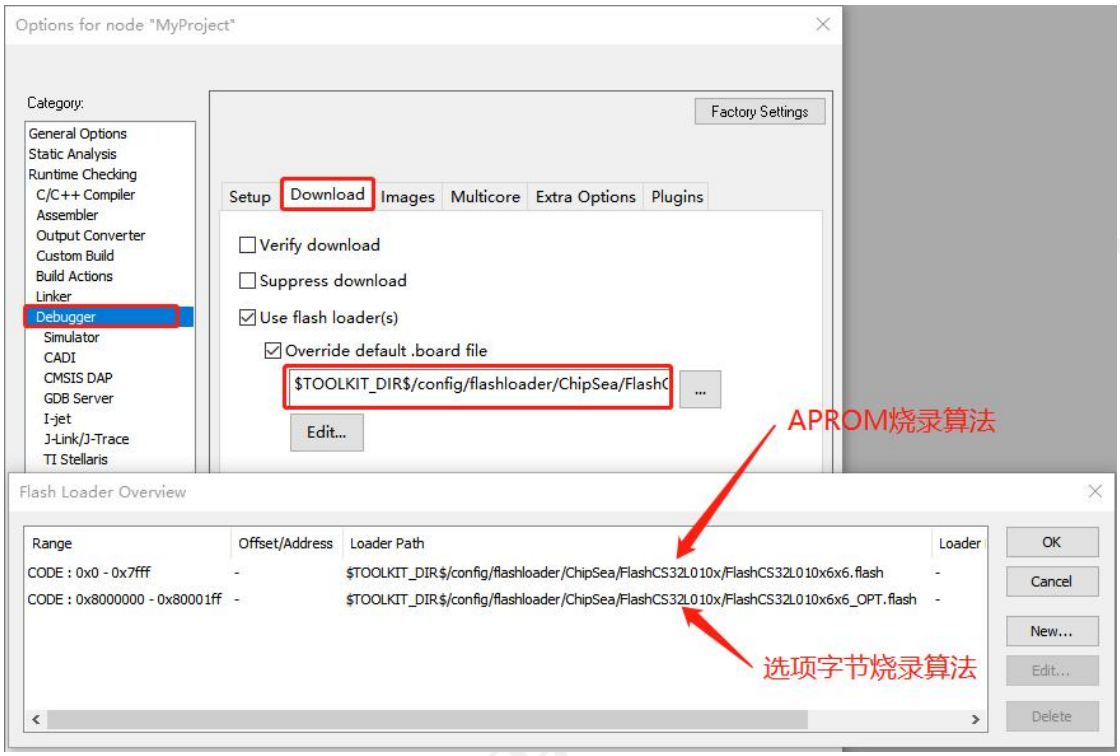
其中.icf 文件各参数如下：

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_1.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x00000000;        //向量表的起始地址
/*-Memory Regions-*/
define symbol __ICFEDIT_region_IROM1_start__ = 0x00000000;  //内部ROM的起始地址
define symbol __ICFEDIT_region_IROM1_end__   = 0x00007FFF;  //内部ROM的结束地址
define symbol __ICFEDIT_region_IROM2_start__ = 0x08000000;
define symbol __ICFEDIT_region_IROM2_end__   = 0x080001FF;
define symbol __ICFEDIT_region_EROM1_start__ = 0x0;         //外部ROM的起始地址
define symbol __ICFEDIT_region_EROM1_end__   = 0x0;         //外部ROM的结束地址
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__ = 0x20000000;  //内部RAM的起始地址
define symbol __ICFEDIT_region_IRAM1_end__   = 0x20000FFF;  //内部RAM的结束地址
define symbol __ICFEDIT_region_IRAM2_start__ = 0x0;
define symbol __ICFEDIT_region_IRAM2_end__   = 0x0;
define symbol __ICFEDIT_region_ERAM1_start__ = 0x0;         //外部RAM的起始地址
define symbol __ICFEDIT_region_ERAM1_end__   = 0x0;         //外部RAM的结束地址
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;

/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__   = 0x400;            //栈的大小
define symbol __ICFEDIT_size_heap__     = 0x400;            //堆的大小
/**** End of ICF editor section. ###ICF###*/


define memory mem with size = 4G;                           //芯片的存储空间

define region IROM_region   =   mem:[from __ICFEDIT_region_IROM1_start__ to __ICFEDIT_region_IROM1_end__]
                              | mem:[from __ICFEDIT_region_IROM2_start__ to __ICFEDIT_region_IROM2_end__];  //内部ROM大小
define region EROM_region   =   mem:[from __ICFEDIT_region_EROM1_start__ to __ICFEDIT_region_EROM1_end__]
                              | mem:[from __ICFEDIT_region_EROM2_start__ to __ICFEDIT_region_EROM2_end__]
                              | mem:[from __ICFEDIT_region_EROM3_start__ to __ICFEDIT_region_EROM3_end__];  //外部ROM大小
define region IRAM_region   =   mem:[from __ICFEDIT_region_IRAM1_start__ to __ICFEDIT_region_IRAM1_end__]
                              | mem:[from __ICFEDIT_region_IRAM2_start__ to __ICFEDIT_region_IRAM2_end__];  //内部RAM大小
define region ERAM_region   =   mem:[from __ICFEDIT_region_ERAM1_start__ to __ICFEDIT_region_ERAM1_end__]
                              | mem:[from __ICFEDIT_region_ERAM2_start__ to __ICFEDIT_region_ERAM2_end__]
                              | mem:[from __ICFEDIT_region_ERAM3_start__ to __ICFEDIT_region_ERAM3_end__];  //外部RAM大小

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };  //栈的大小，8字节对齐
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };  //堆的大小，8字节对齐

initialize by copy { readwrite };                           //启动时将RW数据搬入RAM
do not initialize  { section .noinit };                     //不初始化有.noinit性质的块
                                                            //在0x00000000处放置向量表
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in IROM_region  { readonly };                         //ROM放置的只读数据和代码
place in EROM_region  { readonly section application_specific_ro };
place in IRAM_region  { readwrite, block CSTACK, block HEAP };          //在RAM中放置读写数据与堆栈
place in ERAM_region  { readwrite section application_specific_rw };
```

**图 4.1.2 ICF 文件参数说明**

## 4.2 FLASH 文件

Flash 文件主要的作用是配置烧录程序，用户根据需求可修改其值，源文件在 IAR 安装目录 flashloader/ChipSea/FlashCS32L010x 下。




**4.2 Flash 文件说明**

## 4.3 常见错误及解决措施

### 4.3.1 串口打印无数据输出或提示“Linker Error: "no definition for __write""

IAR 9 版本不支持 fputc 打印输出，需要重定向 printf 串口底层，Common 中提供了 write.c 文件，用户需要修改其中的两个函数，切换不同路的 UART。Common 中无 write.c 文件的则相关函数集成到了 log.c 中，两种方式的函数实现一样。

```c
#pragma module_name = "?__write"
#include "cs32g0xx_uart.h"

int MyLowLevelPutchar(int x)
{
  uart_data_send((uint8_t) x);
  while(SET == uart_flag_status_get(UART_FLAG_TX_BUSY));
  return x;

}

size_t __write(int handle, const unsigned char * buffer, size_t size)
{
#if 1

  size_t nChars = 0;

  if (buffer == 0)
  {
    return 0;
  }

  if (handle != _LLIO_STDOUT && handle != _LLIO_STDERR)
  {
    return _LLIO_ERROR;
  }

  for (/* Empty */; size != 0; --size)
  {
    if (MyLowLevelPutchar(*buffer++) < 0)
    {
      return _LLIO_ERROR;
    }

    ++nChars;
  }

  return nChars;

#else

  return _LLIO_ERROR;

#endif
}
```

根据芯片修改

图 4.3.1 重写__write 函数

### 4.3.2 Identifier "FILE" is undefined，没有选择 FULL

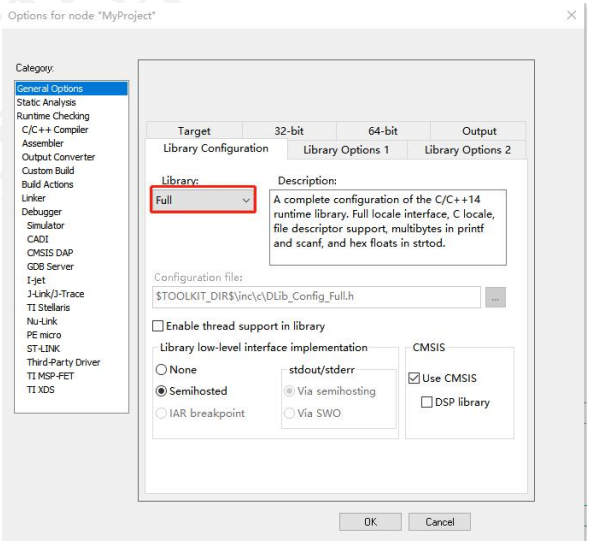



图 4.3.2 选择 FULL

### 4.3.3 Cannot open source file "cmsis_version.h"

错误原因是 Use CMSIS 没选。

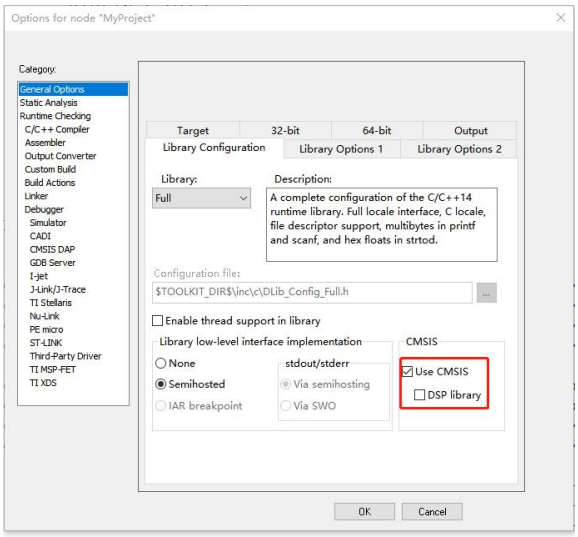



**图 4.3.3 勾选 Use CMSIS**

### 4.3.4 Fatal error :Selected core is not same as as the target core Session aborted

芯片选错：在 Option 中重新选择 Device 芯片类型。

Debug 时 M 核选错:在工程根目录下删除 setting 文件夹里的文件，重新编译，Debug 时重新选择 M 核，在 Option 中选择 Device 芯片类型时可以看到芯片核类型。

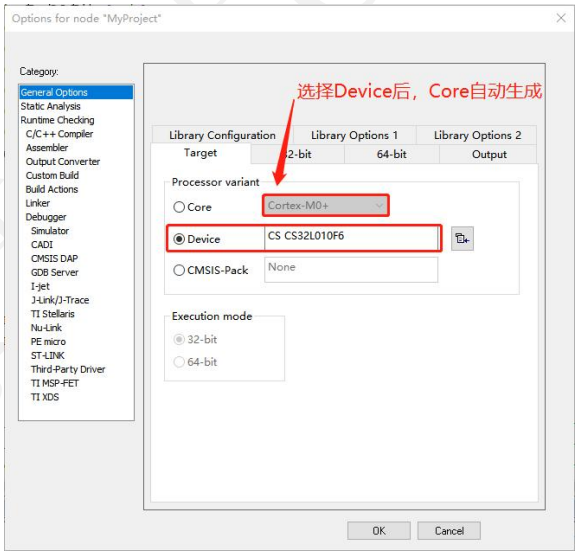



**图 4.3.4 选择 Device**

### 4.3.5 Failed to get cpu status after 4 retries Retry

跳线松了，重新连接跳线；烧录方式错了，烧录方式可以从开发板的管脚丝印或者开发手册中获得；芯片坏了，排除以上两种方式后，可能芯片发生锁死、看门狗开启、芯片上电就进入 PowerDown 模式等。

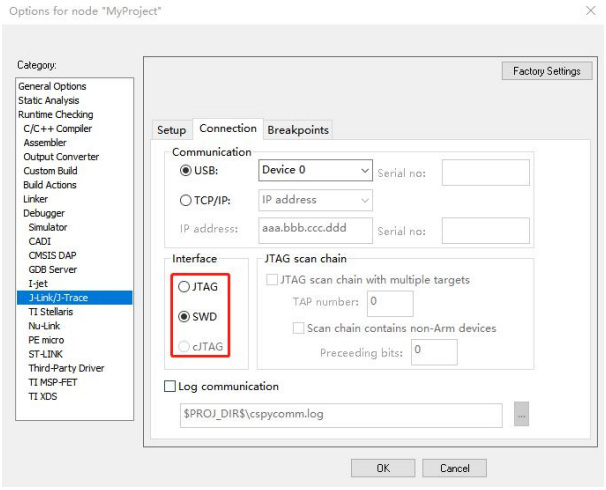



**图 4.3.5 烧录方式**

### 4.3.6 可以正常进入 DeBug，程序卡在 SetSysClock()函数

原因是在 Option 选项中 DeBugger 菜单，选择 Driver 时没有选择 J-link，错选了 Simulator。

```
SetSysClock()
  */
static void SetSysClock(void)
{
    /* Disable reg write protection */
    GCR->REGWRPROT = 0x59;
    GCR->REGWRPROT = 0x16;
    GCR->REGWRPROT = 0x88;

    /* Set Flash Latency to 1 for 32MHz*/
    RCC->CLKCON |= RCC_CLKCON_LATENCY;

    /* Enable reg write protection */
    GCR->REGWRPROT = 0x00;

    /* Wait till HSI is used as system clock source */
    while ((RCC->CLKSTATUS & RCC_CLKSTATUS_HSI_STB) == (uint32_t)RESET)
    {
    }

    /* PCLK = HCLK = SYSCLK = HRC_VALUE */
    RCC->CLKDIV = (uint32_t)0x00000200;
}

/**
  * @}
  */

/**
  * @}
  */
```

**图 4.3.6 错选 Simulator**

### 4.3.7 编译时报错 '.../xxx.o', needed by '.../MyProject.out', missing and no known rule to make it

原因是删除了某个 xxx.c 文件后直接编译导致的，解决办法是对工程进行一次 Clean 再进行编译。

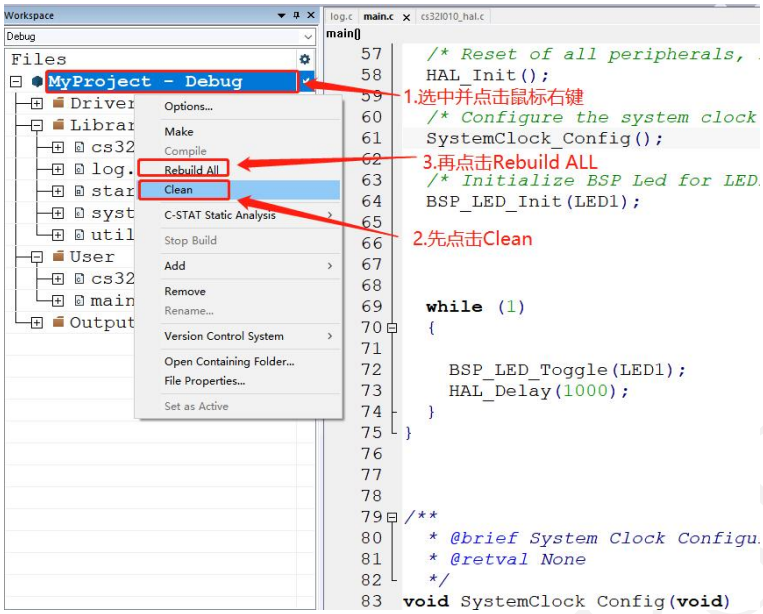



**图 4.3.7 重新编译**

## 免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，芯海科技不对信息的准确性、真实性做任何保证。

芯海科技不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他芯海科技提案、规格书或样品在他处提到的任何保证。

芯海科技不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。



芯海科技 CHIPSEA
股票代码:688595