从 STM32F10x 系列移植到 GD32F30x 系列

目录

目录 *1*
1. 前言 *3*
2. 硬件差异 *3*
3. 内部资源对比 *3*
4. 软件环境设置 *3*
4.1. 使用 Keil 开发 GD32F30x **4**
4.1.1 在 Keil4 中添加 GD32F30x MCU Device 4
4.1.1. 在 Keil5 中添加 GD32F30x MCU Device 6
4.2. 使用 GD-Link 开发 GD32F30x **8**
4.3. 使用 J-Link 开发 GD32F30x **10**
4.4. MDK 使用常见问题解答 **12**
4.4.1. Keil4 打开 Keil5 工程 12
4.4.2. Keil5 打开 Keil4 工程 12
4.4.3. GigaDevice.GD32F30x_DFP.pack 特性 12
4.4.4. Pack 包对 keil 版本要求 13
4.4.5. Keil5 打开 Keil4 工程，编译报错 13
4.5. 使用 IAR 开发 GD32F30x **14**
4.5.1. 在 IAR 中添加 GD32F30x MCU Device 14
4.5.2. 在 IAR 中编译调试 GD32F30x 15
5.1. System **18**
5.1.1. HSE 注意事项 18
5.1.2. 如何通过软件区分 GD32 和 STM32 18
5.1.3. GD32F30x Flash 取指零等待，软件方面注意事项 18
5.1.4. GD32F30x 上电启动异常常见原因 18
5.1.5. MCU 无法正常使用 SWD 下载程序 18
5.1.6. 代码超过 256K 后执行速度慢 19
5.2. CAN **19**
5.2.1. CAN 离线后无法自动恢复 19
5.2.2. CAN 接收异常，接收两帧数据会丢一包数据 19
5.3. ADC **19**
5.3.1. ADC 采集数据异常问题分析 19
5.3.2. ADC1 和 ADC2 同步模式下，ADC2 注入组无数据 19
5.3.3. ADC_CR2 中的 ADCON 使用注意事项 20
5.3.4. ADC 查询法采集数据，出现通道错乱的情况 20
5.3.5. ADC 工作在 DMA 模式下通道数据错乱 20
5.4. SPI **20**
5.4.1. SPI 通信 BSY 标志位 20

5.4.2. SPI 从机模式管脚模式 20
5.5. Uart 20
5.5.1. Uart DMA 注意事项 20
5.6. Flash 21
5.6.1. Flash 函数修改要点 21
5.6.2. Flash 操作地址问题 22

1.前言

本文档就专门介绍从 STM32F10x 移植到 GD32F30x 系列的相关细节，如有纰漏还望见谅。

2.硬件差异

	GD32F30x	STM32F10x
LQFP48	管脚全兼容	
LQFP64	管脚全兼容	
LQFP100	管脚全兼容	
LQFP144	管脚全兼容	

3.内部资源对比

	GD32F303	GD32F305/307	STM32F103	STM32F105/107
Core	$M4_{ROP1}$	$M4_{ROP1}$	$M3_{R1P1}$	$M3_{R1P1}$
Flash	256K-3M	128K-1M	16K-1M	64K/256K
RAM	48K-96K	64K/96K	6K-96K	64K
主频	120M	120M	72M	72M
TIMER	7/8/13/14	7/8/14	4/5/8/14	8
U(S)ART	3/5	5	2/3/5	5
I2C	2	2	1/2	2
SPI (I2S)	3(2)	3(2)	1/2/3(2)	3(2)
CAN	1	2	1	2
USB	Device	OTG	Device	OTG
SDIO	0/1	--	0/1	--
Eth	--	0/1	--	0/1
EXMC	0/1	0/1	1	--
ADC	3(10)/2(16)/3(21)	2(16) /2(21)	2(10)/2(16) /3(16)/3(21)	2(16)
DAC	2	2	2	2

4.软件环境设置

GD32F30x系列为通用型MCU，所以开发环境也可以使用通用型的IDE，目前使用较多的是KEIL，IAR和Visual GDB，客户可以根据个人喜好来选择相应的开发环境，该文档主要介绍KEIL和IAR这两种开发环境。

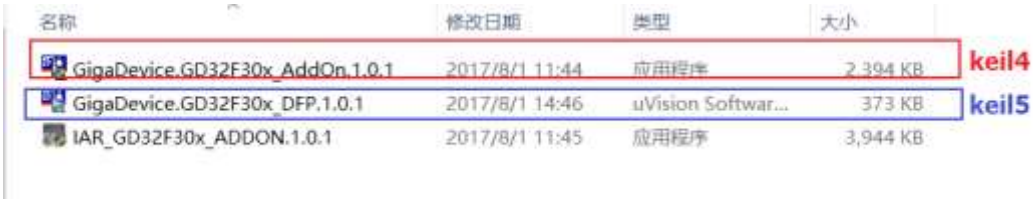
4.1. 使用 Keil 开发 GD32F30x

目前市面通用的MDK for ARM版本有Keil 4和Keil 5：使用Keil 4建议安装4.74及以上；使用Keil 5建议安装5.20以上版本。

4.1.1 在 Keil4 中添加 GD32F30x MCU Device

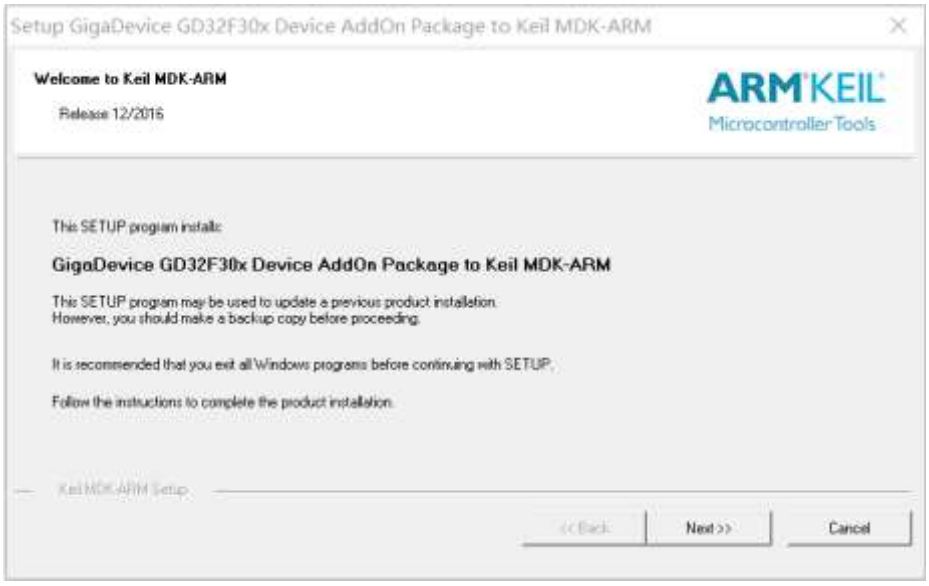
1. 从 MCU 官网或网盘下载相关的 GD32F30x 系列插件 MDK-ARM_AddOn_GD32F30x_V1.0.0.rar。

图 4.1 GD32F30x 系列 MCU 型号支持 pack 包名称（keil4）



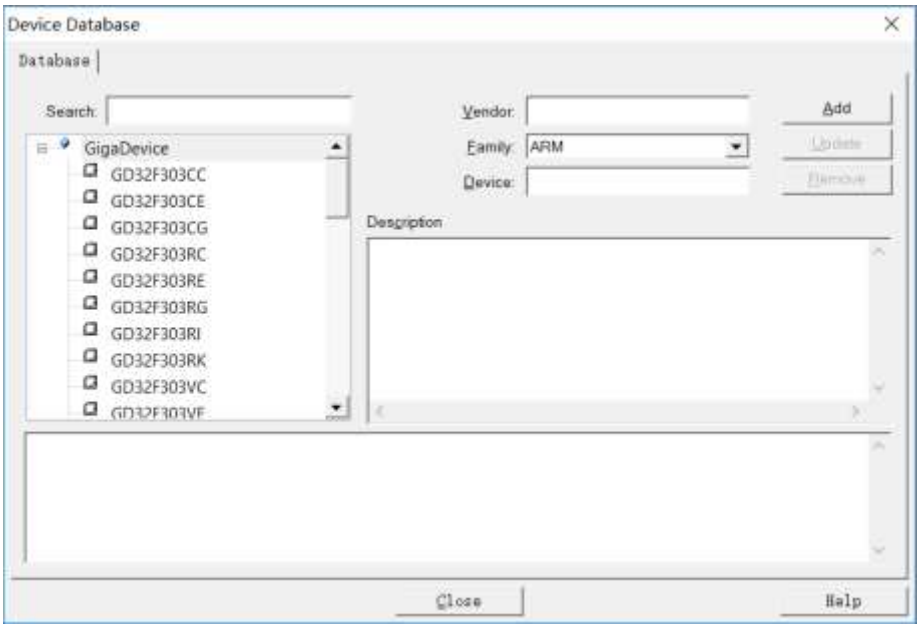
2. 双击解压安装至Keil 4的目录，一般都会默认选择，如若同时安装了Keil 4和Keil 5才需要手动选择。

图 4.2 Pack 包安装示意图（keil4）



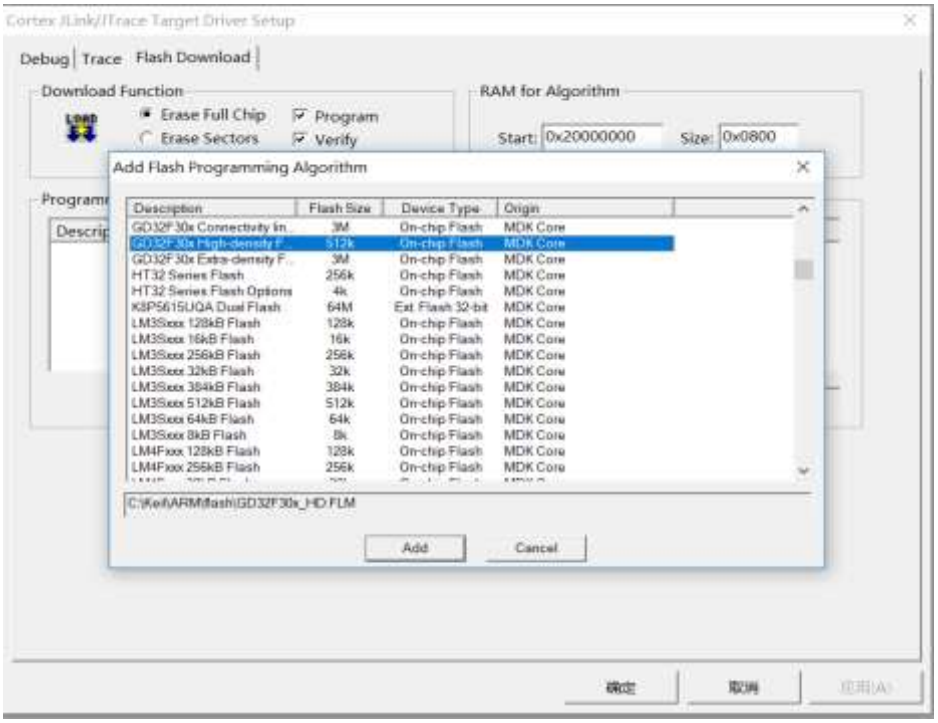
3. 安装成功后，重新打开Keil 4，则可以在File->Device Database中出现Gigadevice的下拉选项，点击可以查看到相应的型号。

图 4.3 Pack 包成功安装示意图（keil4）



4. 为了后续debug工作的顺利进行，建议检查一下安装路径下是否有下载算法，可以通过如下方式查看：打开一个工程，将型号选为GD32F30x的型号，然后Options for Target -> Debug ->Settings -> Flash Download-> Add，如果下拉选项中有GD32F30x的下载算法则完全安装成功。

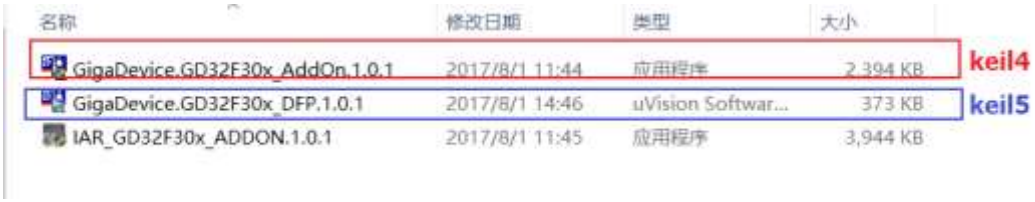
图 4.4 Flash 算法文件选择示意图（keil4）



4.1.1. 在 Keil5 中添加 GD32F30x MCU Device

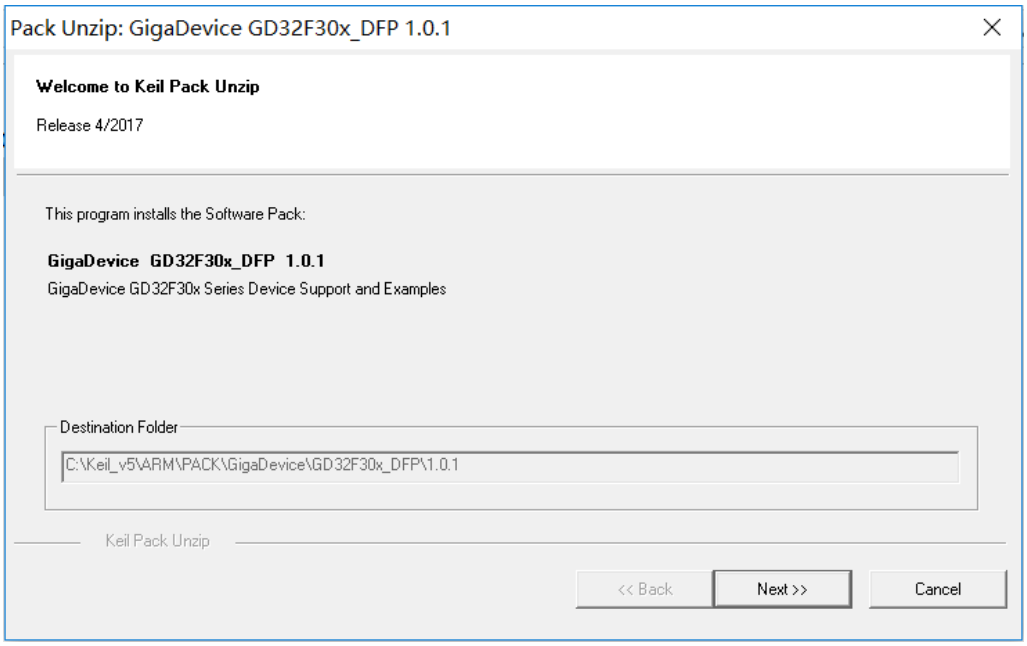
1. 从相关网站下载相关的GD32F30x系列插件Keil.GD32F30x_DFP.1.1.0.rar。

图 4.5 GD32 MCU 型号支持 pack 包名称（keil5）



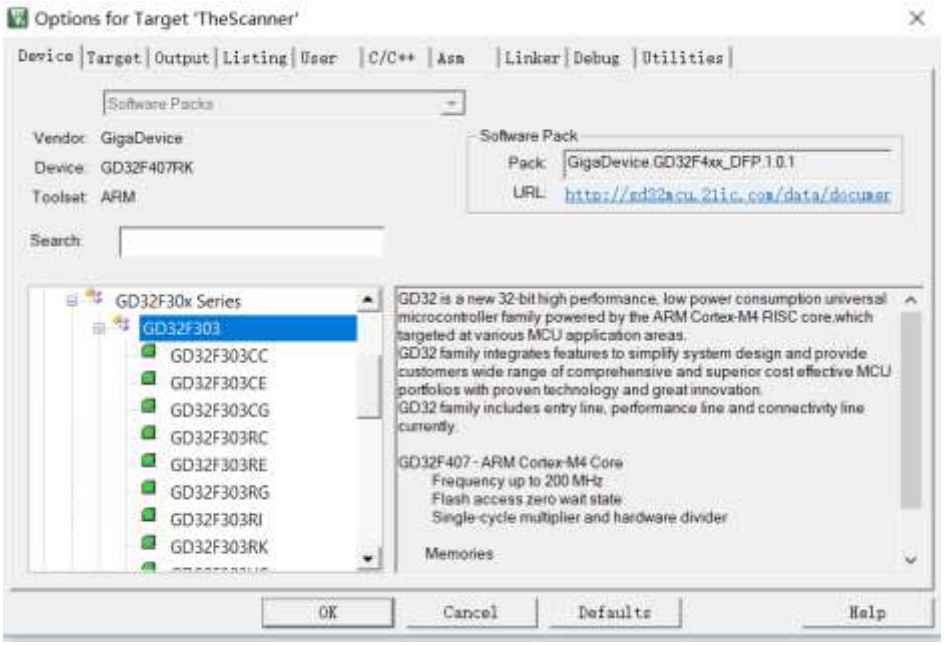
2. 解压并安装至Keil 5的目录，一般都会默认选择。

图 4.6 Pack 包安装示意图（keil5）



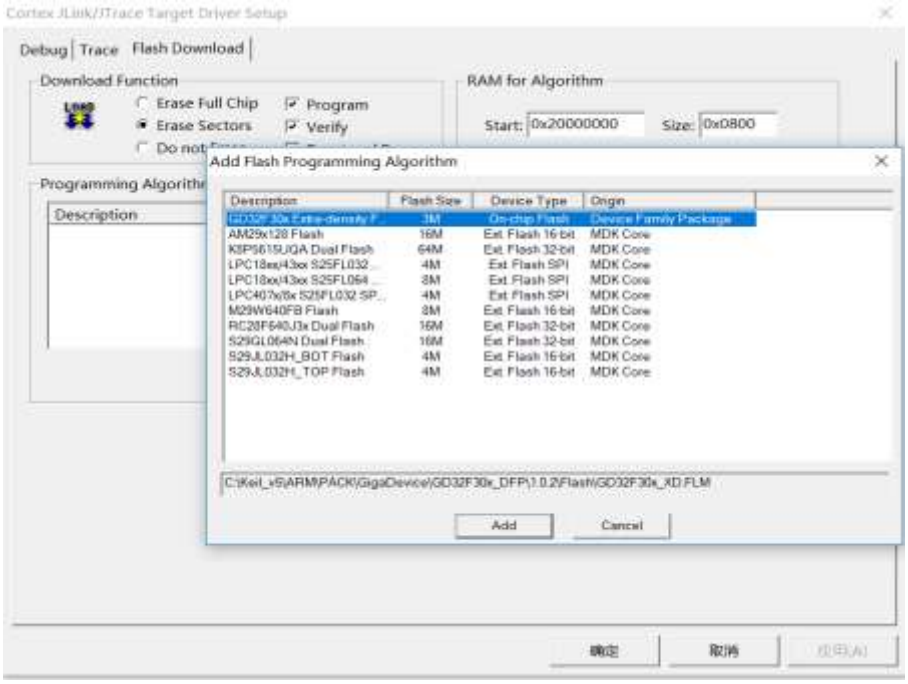
3. 安装完后重新打开keil5工程，即可在Device->Database中出现Gigadevice的型号

图 4.7 Pack 包安装成功示意图（keil5）



4. 在Options for Target -> Debug ->Settings ->Flash Download 中添加flash算法，会出现GD32F30X的算法，即说明安装成功。根据相应的芯片选择合适的算法，即可下载仿真。

图 4.8 Flash 算法文件添加示意图（keil5）



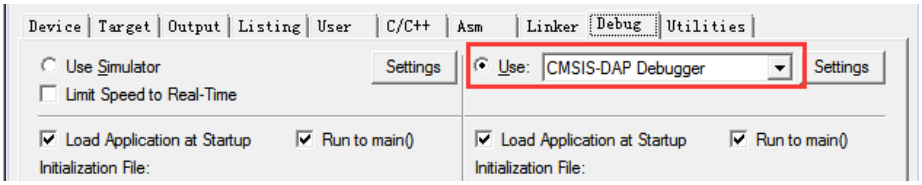
5. 用Keil 5打开Keil 4工程，如果报找不到器件信息等错误，将Keil 4的插件安装在Keil 5的目录下，具体操作方式参考Keil 4插件相关内容。

4.2. 使用 GD-Link 开发 GD32F30x

GD32F30x的开发板自带GD-link，可以用电路板上的GD-link调试仿真代码，操作方法如下。

1. 在Options for Target -> Debug 中选择“CMSIS-DAP Debugger”，部分客户反馈找不到这一驱动器选项，那是因为MDK版本过低，只有Keil4.74以上的版本和Keil5才支持CMSIS-DAP Debugger选项。

图 4.9 GD-Link 选择 Debugger 类型

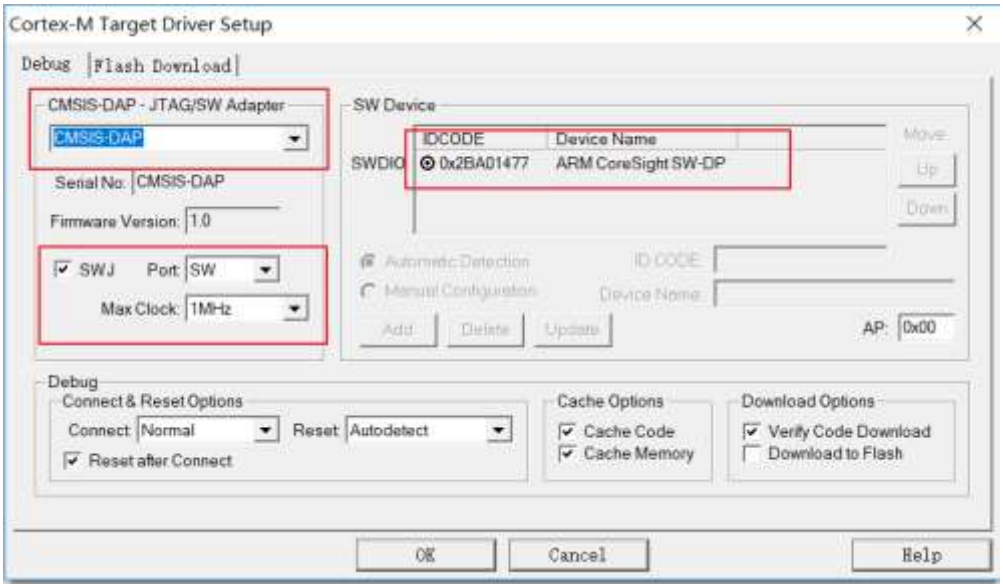


2. 在Options for Target -> Utilities，也要选择“CMSIS-DAP Debugger”。

图 4.10 GD-Link 在 Utilities 中选择 Debugger 类型

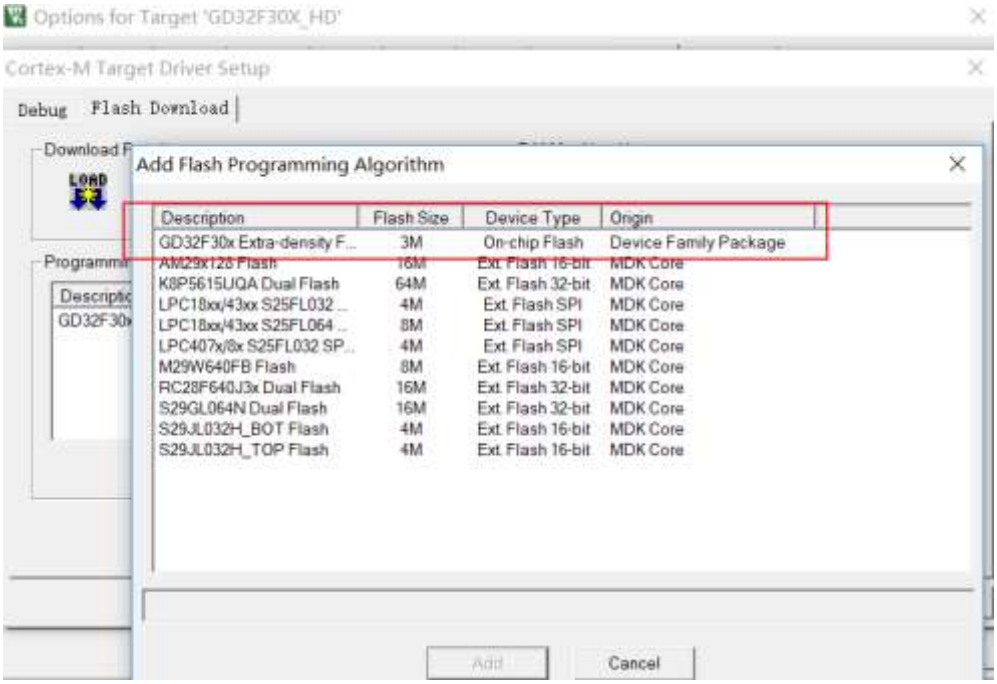
3. 在Options for Target -> Debug ->Settings勾选SWJ、Port选择 SW。右框IDcode会出现”0xXBAXXXXX”。

图 4.11 GD-Link 成连接目标板示意图



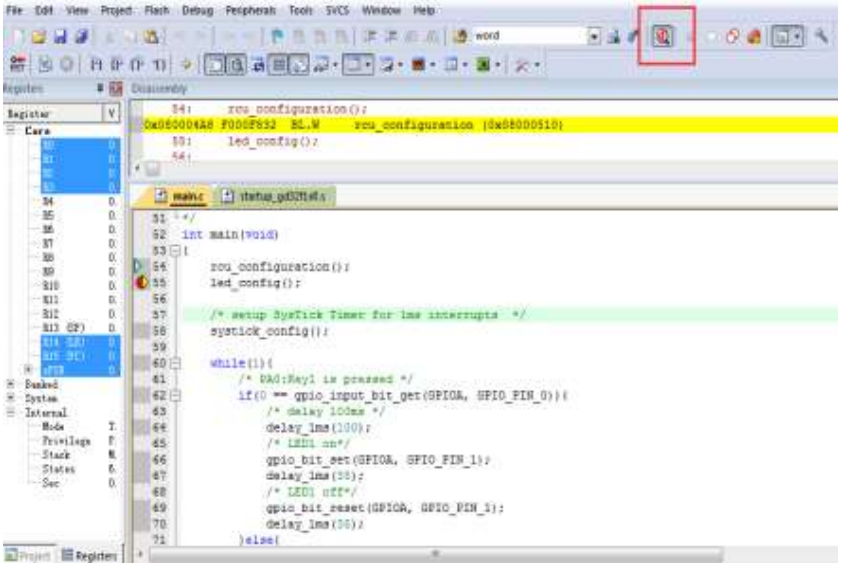
4. 在Options for Target -> Debug ->Settings -> Flash Download中添加GD32的flash算法。

图 4.12 GD-Link 添加 Flash 算法文件示意图



5. 单击下图的快捷方式“debug”，即可使用GD-Link进行仿真。

图 4.13 GD-Link 仿真示意图

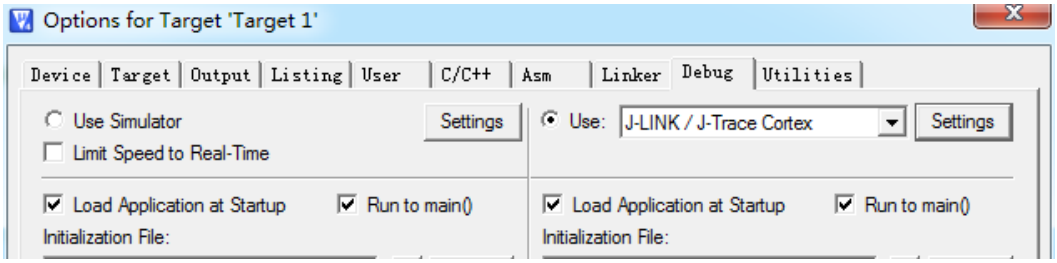


4.3. 使用 J-Link 开发 GD32F30x

使用J-Link来debug GD MCU，具体配置如下：

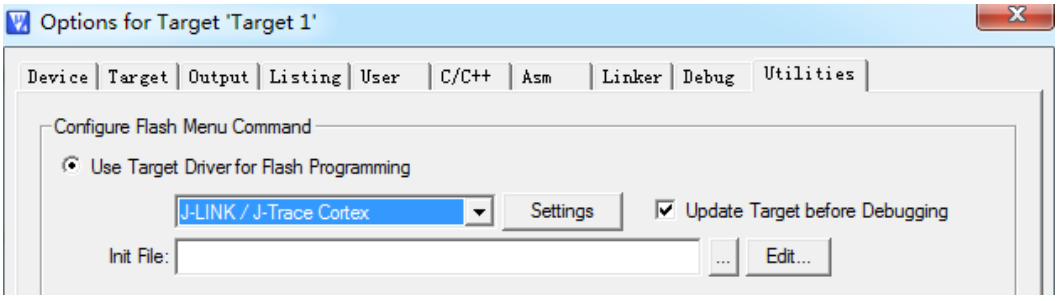
1. 在Options for Target -> Debug中选择“J-LINK/J-Trace Cortex”

图 4.14 J-Link 在 Keil 中选择 Debugger 示意图



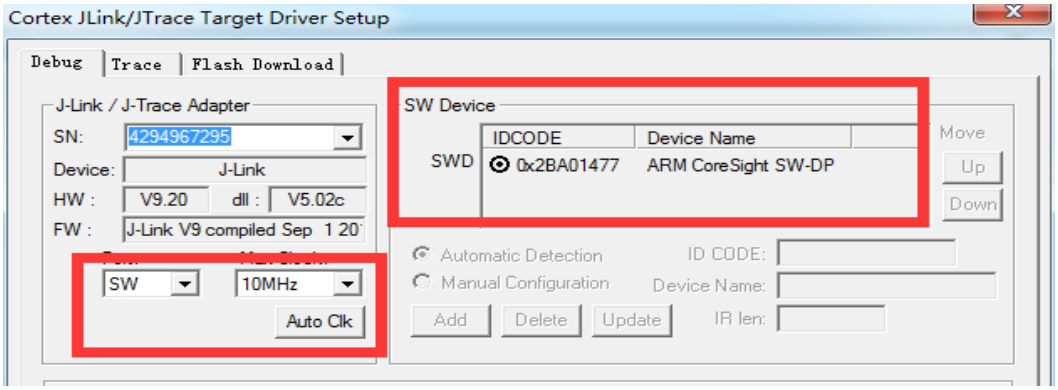
2. 在Options for Target -> Debug ->Utilities,也要选择“J-LINK/J-Trace Cortex”。

图 4.15 J-Link 在 Utilities 下选择 Debugger 示意图



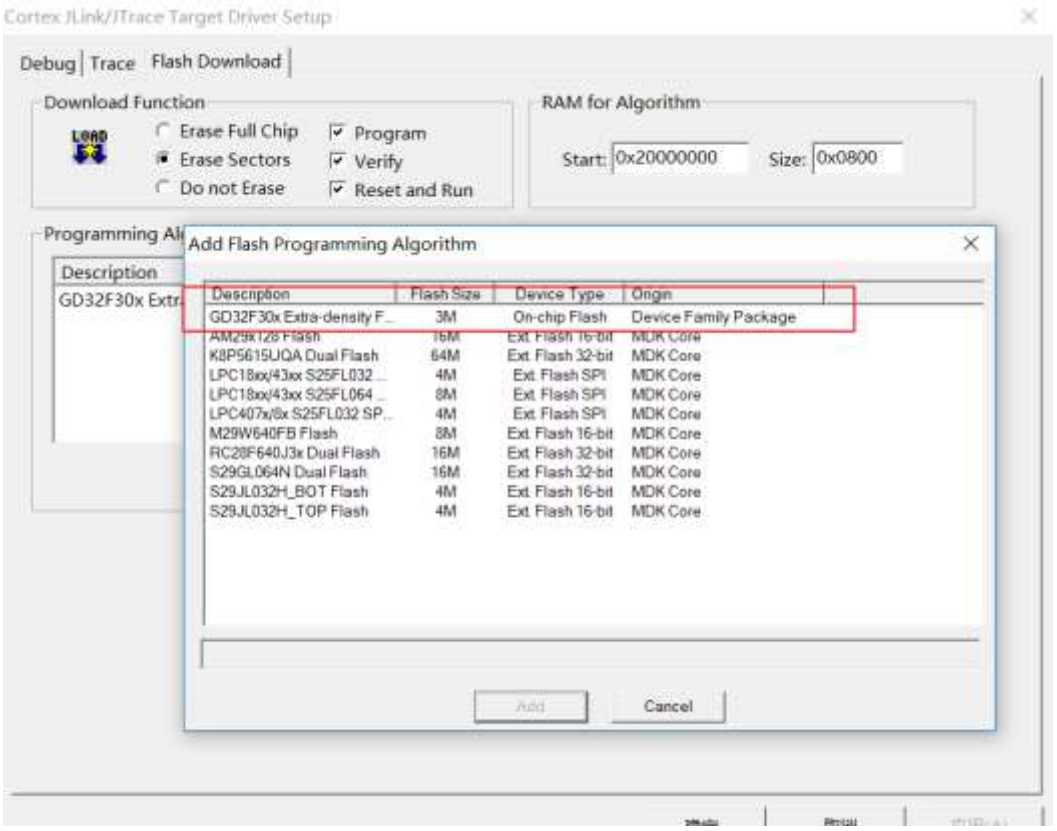
3. 在Options for Target -> Debug ->Settings勾选SWJ，Port选择 SW。右框IDcode会出现“0xXBAXXXXX”。

图 4.16 J-Link 成功连接目标板示意图



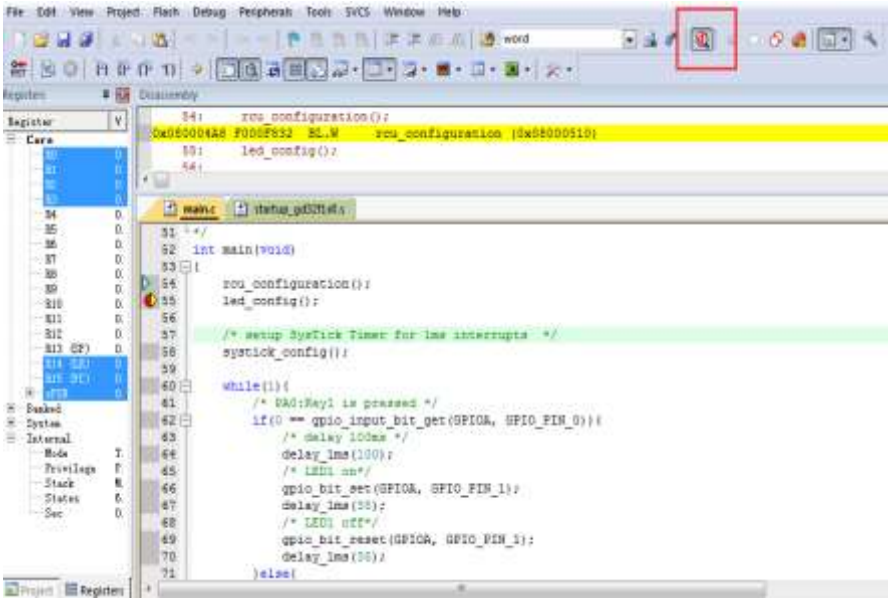
4. 在Options for Target -> Debug ->Settings -> Flash Download中添加GD32的flash算法。

图 4.17 J-Link 在 Keil 下添加 flash 算法文件示意图



5. 单击下图的快捷方式“debug”，即可使用J-Link进行仿真。

图 4.18 J-Link 成功仿真示意图



4.4. MDK 使用常见问题解答

4.4.1. Keil4 打开 Keil5 工程

如果没有安装Keil 5，也是能够使用Keil 4来编译Keil 5的工程，具体做法就是修改工程的后缀名，将Keil5工程的后缀名xxxx.uvprojx修改为xxxx.uvproj，即可使用Keil 4来查看编译了。

4.4.2. Keil5 打开 Keil4 工程

如果使用Keil 5打开Keil 4工程，打开时会遇到找不到MCU器件的情况，这种可以直接将Keil4工程的后缀名xxxx.uvproj修改为xxxx.uvprojx，即可正常使用Keil 5来查看编译了。

4.4.3. GigaDevice.GD32F30x_DFP.pack 特性

1. 支持在线安装方式；
2. 支持本地安装方式；
3. 自动生成GD32F30x系列MCU列表及对应的特征信息；
4. 自动匹配所选芯片对应的Flash算法；
5. 支持用户在Debug模式下查看寄存器状态；
6. 利用Books选项卡获取文档资料。

4.4.4. Pack 包对 keil 版本要求

Pack包适用于Keil 5.15及以上版本，对于Keil5.13和5.14版本，有如下两个问题：

1. Debug模式下无法调用SVD文件查看寄存器状态；
2. 对Pack进行Schema check会报错。

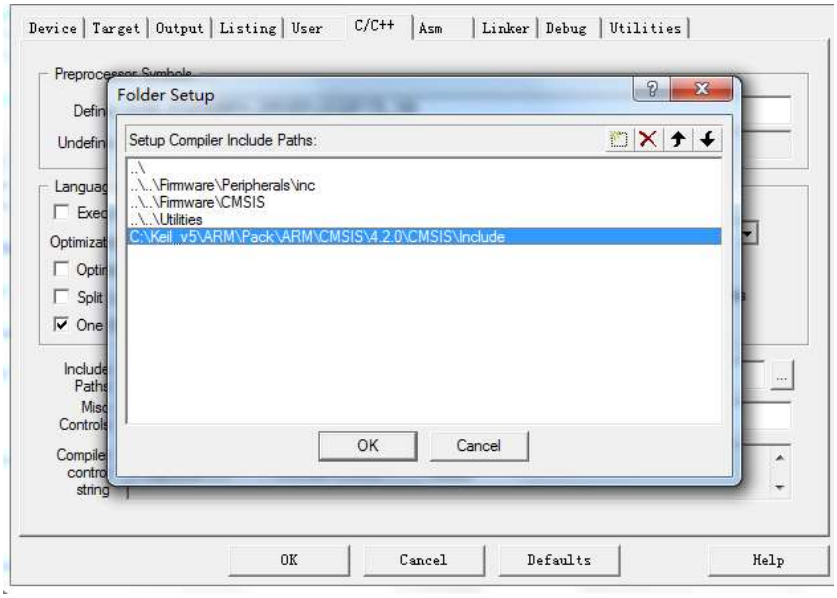
4.4.5. Keil5 打开 Keil4 工程，编译报错

图 4-4.19. 编译错误示意图

```
..\..\Firmware\CMSIS\core_cm3.h(147): error:  #5: cannot open source input file "core_cmInstr.h": No such file or directory
  #include <core_cmInstr.h>               /* Core Instruction Access                          */
..\..\Firmware\Peripherals\src\gd32flm0_dma.c: 0 warnings, 1 error
compiling gd32flm0_fmc.c...
..\..\Firmware\CMSIS\core_cm3.h(147): error:  #5: cannot open source input file "core_cmInstr.h": No such file or directory
  #include <core_cmInstr.h>               /* Core Instruction Access                          */
..\..\Firmware\Peripherals\src\gd32flm0_fmc.c: 0 warnings, 1 error
```

错误原因是core_cmInstr.h文件的路径在Keil5和Keil4中不同，可在Option for Target的C/C++中添加core_cmInstr.h的文件路径，如图**4-4.20**所示：

图 4-4.21. 文件路径添加示意图



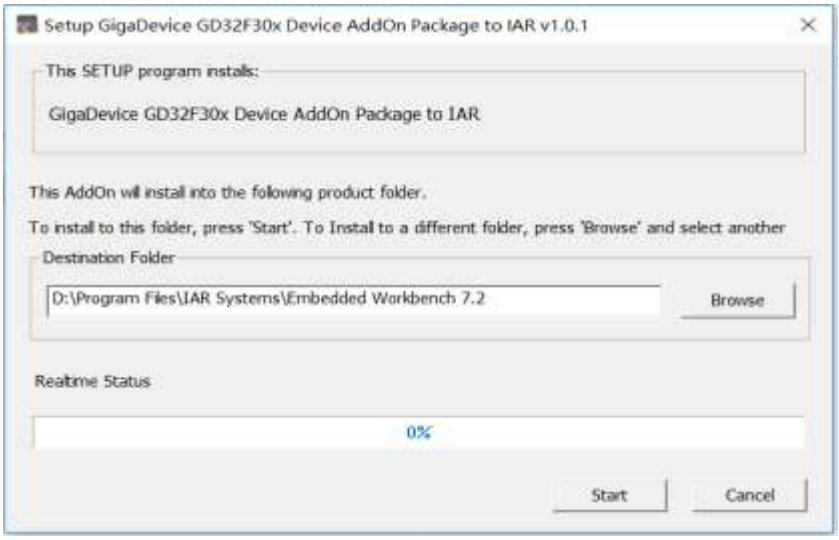
4.5. 使用 IAR 开发 GD32F30x

IAR版本众多，版本之间的兼容性并不好，如果初次使用建议安装7.3以上的版本,安装好IAR以后再根据该文档来添加GD的器件型号，进行相关的debug工作。

4.5.1. 在 IAR 中添加 GD32F30x MCU Device

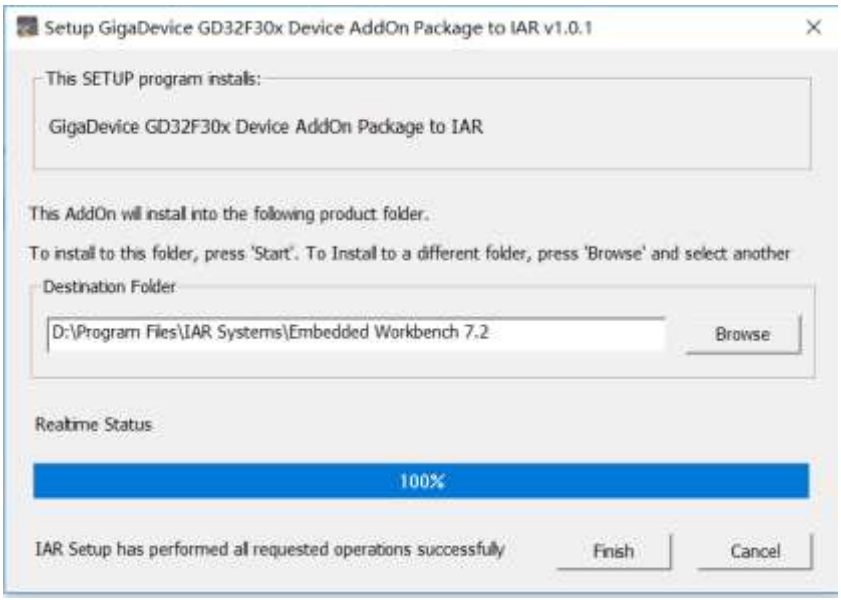
1. 从相关网站下载相应的GD32F30x系列插件IAR_GD32F30x_ADDON.1.0.0.exe：
2. 运行IAR_GD32F30x_ADDON.1.0.0.exe,单击start开始安装插件。

图 4-4.22. IAR 中安装支持 GD32 型号 pack 包示意图



3. 安装成功后单击Finish，结束插件安装。

图 4-4.23. IAR 下 pack 包安装示意图

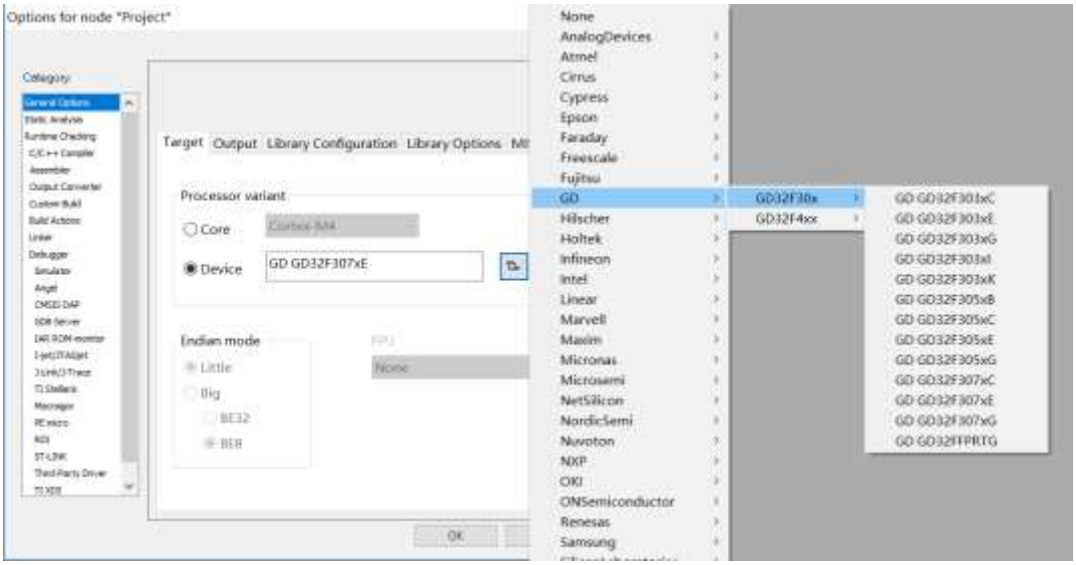


4.5.2. 在 IAR 中编译调试 GD32F30x

在上一小节中我们已经添加了GD32F1系列的插件，这一小节我们介绍应如何使用它。

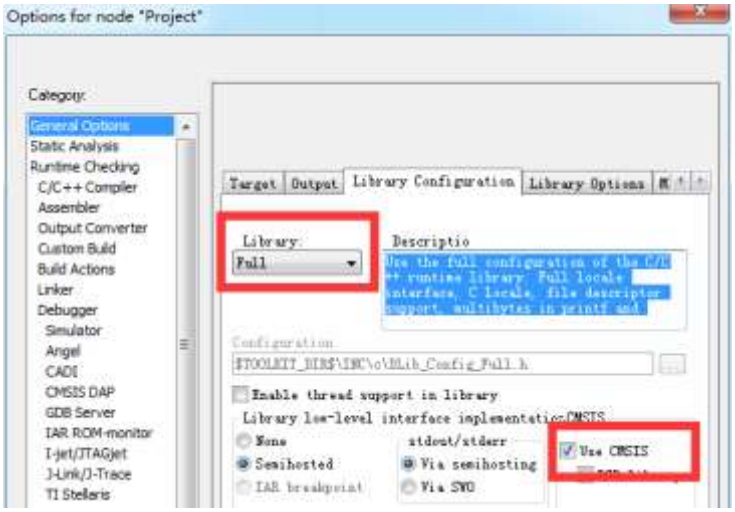
使用IAR编译GD的型号，有两个办法，一种是使用现有的工程进行修改，还有就是重新建立工程，这里就不细说具体工程应该如何建立，GD的工程建立和别的平台都一致，建立工程时选择GD的相应型号。如果没有安装GD的插件，可以选择别的M3厂家型号。

图 4-4.24. 在 IAR 下选择芯片型号示意图



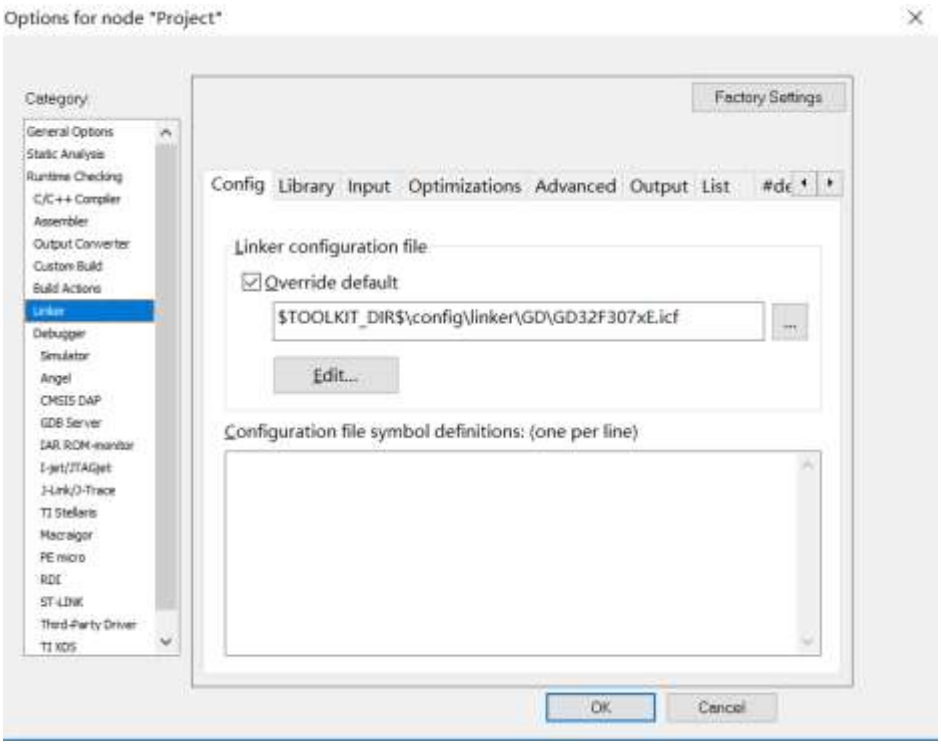
6.1以后的IAR不需要添加CMSIS文件（core_cm4.c和core_cm4.h），但是需要勾选General Options->Library Configuration的Use CMSIS，如果软件代码有使用到printf函数，还需要修改Library为FULL。

图 4-4.25. 在 IAR 下添加 CMSIS 文件示意图



芯片的Link文件建立工程时会默认根据型号选定，但是编译前还是要有检查的习惯，检查一下ICF文件是否有配置，是否正确。

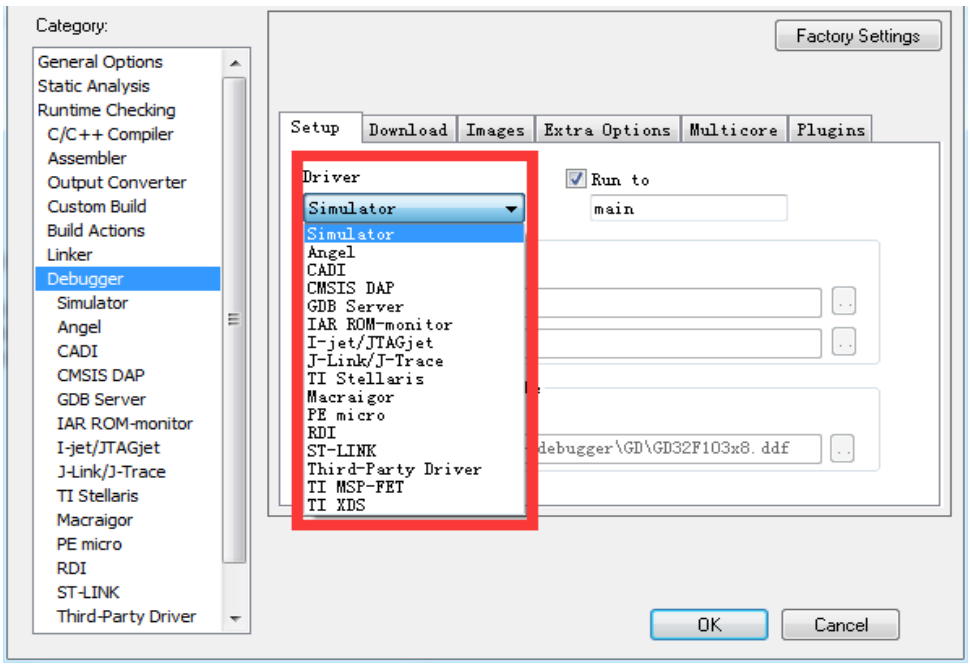
图 4-4.26. 在 IAR 下添加 ICF 文件示意图



配置Debugger->Setup选项，新建立的工程默认是Simulator模拟，如果需要调试那么需要根据实际情况来选择：

1. 使用GD-Link选择CMSIS DAP（兼容性不好，不建议在IAR下使用）；
2. 使用J-Link选择J-Link/J-Trace。

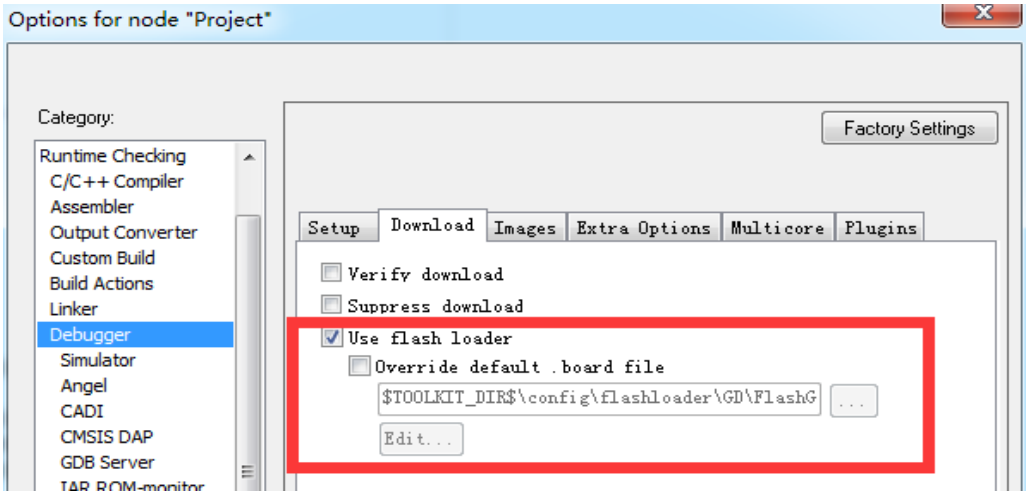
图 4-4.27. 在 IAR 下选择 Debugger 示意图



配置Debugger->Download选项，新建的工程有可能没有配置download选项，如果我们需要调试代码那么务必要勾选User flash loader选项，且保证board file准确，否则程序无法

正常下载至芯片内部。

图 4-4.28. 在 IAR 下配置 flash loader 示意图



如果选择了Debugger选项，那么还需要根据Debugger选项设定对应的调试选项；如果选择的是GD的型号，在IAR下面已经固定将所有的调试接口都配置为SWD接口，可以忽略该选项配置，直接进行相关的代码debug工作。

5.1. System

5.1.1. HSE 注意事项

GD32 外部晶体起振时间会比 STM32F10x 系列要长，所以原有的晶体超时时间需要加大：

调整前：

```
#define HSE_STARTUP_TIMEOUT      ((uint16_t)0x0500) /*!< Time out for HSE start up */
```

调整后：

```
#define HSE_STARTUP_TIMEOUT      ((uint16_t)0xFFFF) /*!< Time out for HSE start up */
```

5.1.2. 如何通过软件区分 GD32 和 STM32

GD32F30x 在设计阶段，已经预留了相关寄存器，用户只需要软件读取寄存器,即可获取到相关的型号信息，GD32F30x 每一型号此处值都为固定值。

```
Code_Num=*( uint32_t *)( 0x40022100 );
```

5.1.3. GD32F30x Flash 取指零等待，软件方面注意事项

GD32F30x 系列 Flash 都为零等待设计，在同主频下，带来了更高的性能体验。如果用户代码有用到 for 循环或者是 while 循环语句来做延时，延时时间在 GD32F30x 系列上会变短，需要适当的加大延时参数或改用 Timer 来做延时函数。

5.1.4. GD32F30x 上电启动异常常见原因

1. 检查板子上 Boot 0 引脚是否悬空，GD32F30x 运行用户程序必须要求 Boot0 经 10K 电阻接 GND；
2. 如果板子上有大功率器件（Wifi、GSM、GPS 等），检查大功率器件开启瞬间 V_{DD} 是否存在跌落情况，如存在跌落可以适当加大电源输出端的负载电容；
3. 观察芯片的复位管脚，复位管脚是否一直处于拉低状态，检查是否供电异常或者是芯片硬件看门狗使能了，芯片处于反复复位状态。

5.1.5. MCU 无法正常使用 SWD 下载程序

1. 接线异常，SWD 相关的调试口未正常接好；

2. 芯片是否被读保护或者处于反复复位状态；
3. SWD 的调试线过长或者是通信速率过高，适当减短 SWD 数据线，同时降低 SWD 速率；
4. 按照硬件指南给 SWD 添加相应的上下拉电阻，提高通信抗干扰能力。

5.1.6. 代码超过 256K 后执行速度慢

GD32F30x 系列的 Flash 分为 Code 区（前 256K）和 Data 区（256K 以后的区域），二者在擦写操作上没有区别，但是读操作时间上存在较大差别，code 区代码取值零等待，data 区执行代码会有较长延时。应用中如果涉及该架构影响到使用可以通过分散加载来改善，具体做法参考分散加载应用文档。

5.2. CAN

5.2.1. CAN 离线后无法自动恢复

GD32F10x CAN 模块的离线自动恢复功能与 CAN 协议定义的离线恢复序列存在一定的理解差异，因此有可能出现 CAN 离线后无法自动恢复的现象。该功能可以通过使能离线中断，在离线中断内重新初始化 CAN 模块来实现。

5.2.2. CAN 接收异常，接收两帧数据会丢一包数据

GD32F30x 接收缓存会自动释放，如果手动多调用一次清缓存的动作会导致 CAN 接收丢包，也就是软件中无需主动调用 CAN_FIFORelease 函数，CAN FIFO 会被自动释放。

5.3. ADC

5.3.1. ADC 采集数据异常问题分析

1. ADC 通道的采集引脚未配置为模拟输入，GD32 要求通道 IO 口必须配置为模拟输入；
2. ADC 时钟过高，ADC 采样时钟高于 40M 获取到的数据不具有参考意义；
3. ADC 不耐 5V 的 IO 口被接入超过 VDDA 的电平信号；
4. ADC 采样值偏小或不稳定，应该适当的降低 ADC 时钟，加大采样周期的值。

5.3.2. ADC1 和 ADC2 同步模式下，ADC2 注入组无数据

如果 ADC1 和 ADC2 同步采集，ADC2 是跟着 ADC1 同步触发，此时 ADC2 的注入组的触发方式需要手动配置成软件触发（默认是 TIMER1_TRGO）。

5.3.3. ADC_CR2 中的 ADCON 使用注意事项

ADC 使能以后需要在代码里面插入 20us 的延时；

```
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
Delay_us(20);
```

5.3.4. ADC 查询法采集数据，出现通道错乱的情况

ADC 使用查询法采集数据时，如果使能了 ADC 的 SCAN 模式，就有可能会出现 ADC 数据错乱的情况；ADC 采集通道 SCAN 功能只适用于多通道注入采样和 DMA 模式。

5.3.5. ADC 工作在 DMA 模式下通道数据错乱

关掉 ADC 的 DMA 开关无法复位 ADC 的 DMA 请求，导致重新配置 DMA 模块时，有可能会残留一个没有得到应答的 DMA 请求。当 DMA 模块使能后，会立刻响应这个请求，导致软件与硬件的数据错位。可以在 DMA 的 TC 中断中迅速关掉 ADC 的 DMA 开关，或者 ADC_ON，保证下一个 DMA 请求不会发出。也可以使用 RCC 中的 ADC 复位寄存器将 ADC 模块复位，使 DMA 请求清零。

5.4. SPI

5.4.1. SPI 通信 BSY 标志位

在 SPI 程序编写的过程中，轮询使用 BSY 作为通信标志位，导致传送数据丢失或者是错误。这主要是因为 GD 的 BSY 标志位不是在写入 DR 后就置位的，而是发送完第一个 bit 才被置位，传输过程中不要使用 BSY 作为每次传输的判断，使用 TXE 和 RXNE 来进行判断。

5.4.2. SPI 从机模式管脚模式

从机模式下 CLK、MISO、NSS 需要将 IO 配置成 Input_floating,才能正常工作。

5.5. Uart

5.5.1. Uart DMA 注意事项

使用 uart DMA 发送数据的时候，可能丢掉一帧中的第一个 byte 数据，注意尽量不要在发送的时候频繁的开关 uart 发送，
usart_transmit_config(USARTx,USART_TRANSMIT_ENABLE),
假如要关闭 uart 发送，需要按照以下流程：
先开启 usart_transmit_ enable 再去打开 dma_channel_enable,如下图。

```
void USART2_Send_Buf(uint8_t *buf , uint32_t len)
{
    usart_transmit_config(USART2,USART_TRANSMIT_DISABLE); //
    dma_channel_disable(DMA0, DMA_CH1); //
    dma_transfer_number_config(DMA0,DMA_CH1,len);
    dma_memory_address_config(DMA0,DMA_CH1,(uint32_t) buf);
    usart_transmit_config(USART2,USART_TRANSMIT_ENABLE); //
    /* enable DMA channel1 */
    dma_channel_enable(DMA0, DMA_CH1);
    }
```

5.6. Flash

5.6.1. Flash 函数修改要点

GD 的 Flash 执行速度快，但是写操作慢，所以在对 Flash 操作的时候需要修改下面几个函数：

修改 Flash 擦除和编程超时宏定义：

#define EraseTimeout ((uint32_t)0xFFFFFFFF)//0x000B0000

#define ProgramTimeout ((uint32_t)0xFFFFFFFF)//00002000

修改选项字节操作函数：

FLASH_Status FLASH_EraseOptionBytes(void);

FLASH_Status FLASH_ProgramOptionByteData(uint32_t Address, uint8_t Data);

FLASH_Status FLASH_EnableWriteProtection(uint32_t FLASH_Pages);

FLASH_Status FLASH_ReadOutProtection(FunctionalState NewState);

在这四个函数写完 key（ FLASH->OPTKEYR = FLASH_KEY1;FLASH->OPTKEYR = FLASH_KEY2;）后添加两个__nop()语句或者是增加 如下语句：

```
while((FLASH->CR&CR_OPTWRE_Set)!=CR_OPTWRE_Set )
    {}
```

```
#define    CR_OPTWRE_Set                          ((uint32_t)0x00000200)
FLASH_Status FLASH_EraseOptionBytes(void)
{
  uint16_t rdptmp = RDP_Key;

  FLASH_Status status = FLASH_COMPLETE;

  /* Get the actual read protection Option Byte value */
  if(FLASH_GetReadOutProtectionStatus() != RESET)
  {
    rdptmp = 0x00;
  }

  /* Wait for last operation to be completed */
  status = FLASH_WaitForLastOperation(EraseTimeout);
  if(status == FLASH_COMPLETE)
  {
    /* Authorize the small information block programming */
    FLASH->OPTKEYR = FLASH_KEY1;
    FLASH->OPTKEYR = FLASH_KEY2;
    while((FLASH->CR&CR_OPTWRE_Set)!=CR_OPTWRE_Set )
                          {}
    /* if the previous operation is completed, proceed to erase the option bytes */
    FLASH->CR |= CR_OPTER_Set;
    FLASH->CR |= CR_STRT_Set;
    /* Wait for last operation to be completed */
    status = FLASH_WaitForLastOperation(EraseTimeout);
```

5.6.2. Flash 操作地址问题

写 Flash，必须采用绝对地址，也就是 0x08000000 为首地址。而对于读操作，既可以使用绝对地址，也可以用相对地址 0x00000000。