文档编号：AN2048

上海东软载波微电子有限公司

应用笔记

CDK

修订历史

版本	修订日期	修改概要
V1.00	2023-12-12	初版
V1.01	2024-04-18	1. 增加优化等级说明； 2. 更新 CDK 下载网址。
V1.02	2025-11-17	1. 更新建议使用的 CDK 版本； 2. 增加将变量、常量、函数指定到具体地址的方法； 3. 增加 bin 文件合并方法。

地　址：中国上海市徐汇区古美路 1515 号凤凰园 12 号楼 3 楼
邮　编：200235
E-mail：support@essemi.com
电　话：+86-21-60910333
传　真：+86-21-60914991
网　址：http://www.essemi.com/

目 录

内容目录

第1章 开发环境

1. 1 CDK安装包下载

CDK 官方下载地址：https://www.xrvm.cn/community/download。搜索关键词“CDK”，找到最新版本（V2.24.14 及以上）的“剑池 CDK 集成开发环境”，点击“下载”即可。注意：登录后方可下载。

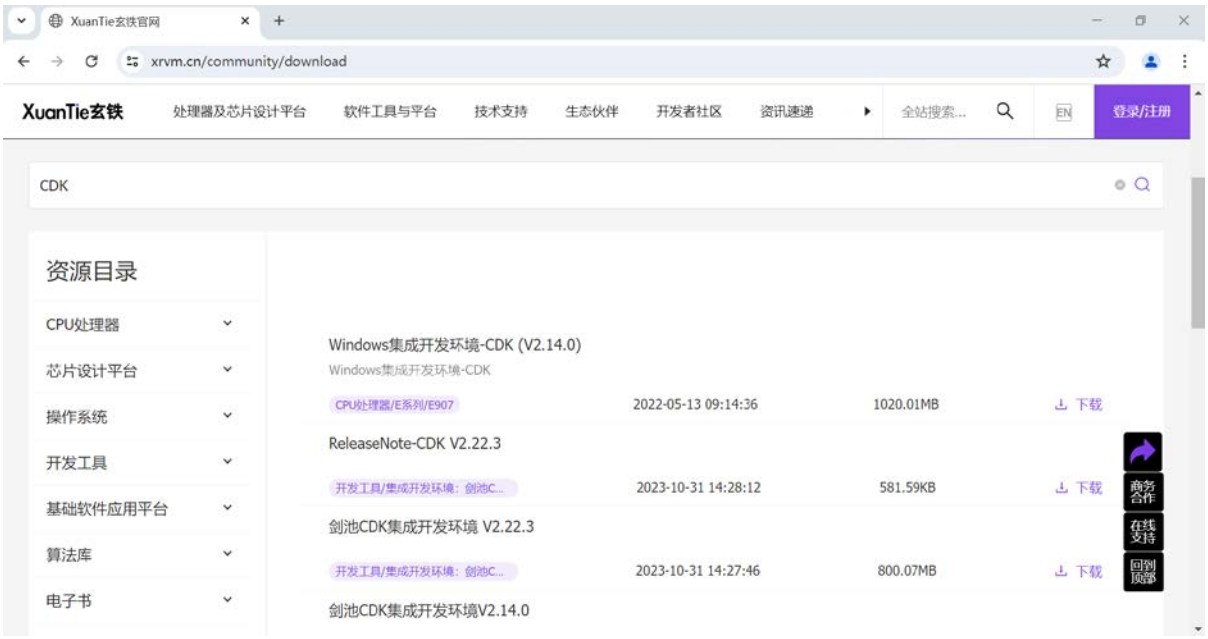


图 1-1 CDK 安装包官方下载网页

1. 2 调试环境搭建

CDK 集成了 T-Head DebugServer，安装好 CDK 后，便可使用 CKLink 调试器对芯片进行调试，也可使用 ESLinkIIOB 或 ESLinkIIPro 进行调试。ESLinkIIOB 或 ESLinkIIPro 也可配合 ESBurner 上位机对芯片进行擦除、编程等操作。

CDK 配合 ESLinkIIOB 或 ESLinkIIPro 调试前，需进行如下操作：

① 打开 ESBurner 上位机，点击“设备”选择对应的调试器，点击“芯片”选择 ES32VF2264 芯片，并确保待调试的芯片配置字里的 GBRDP 选择“读保护等级 Level0”，否则无法正常调试，量产烧录前可根据需求自行设置 GBRDP。

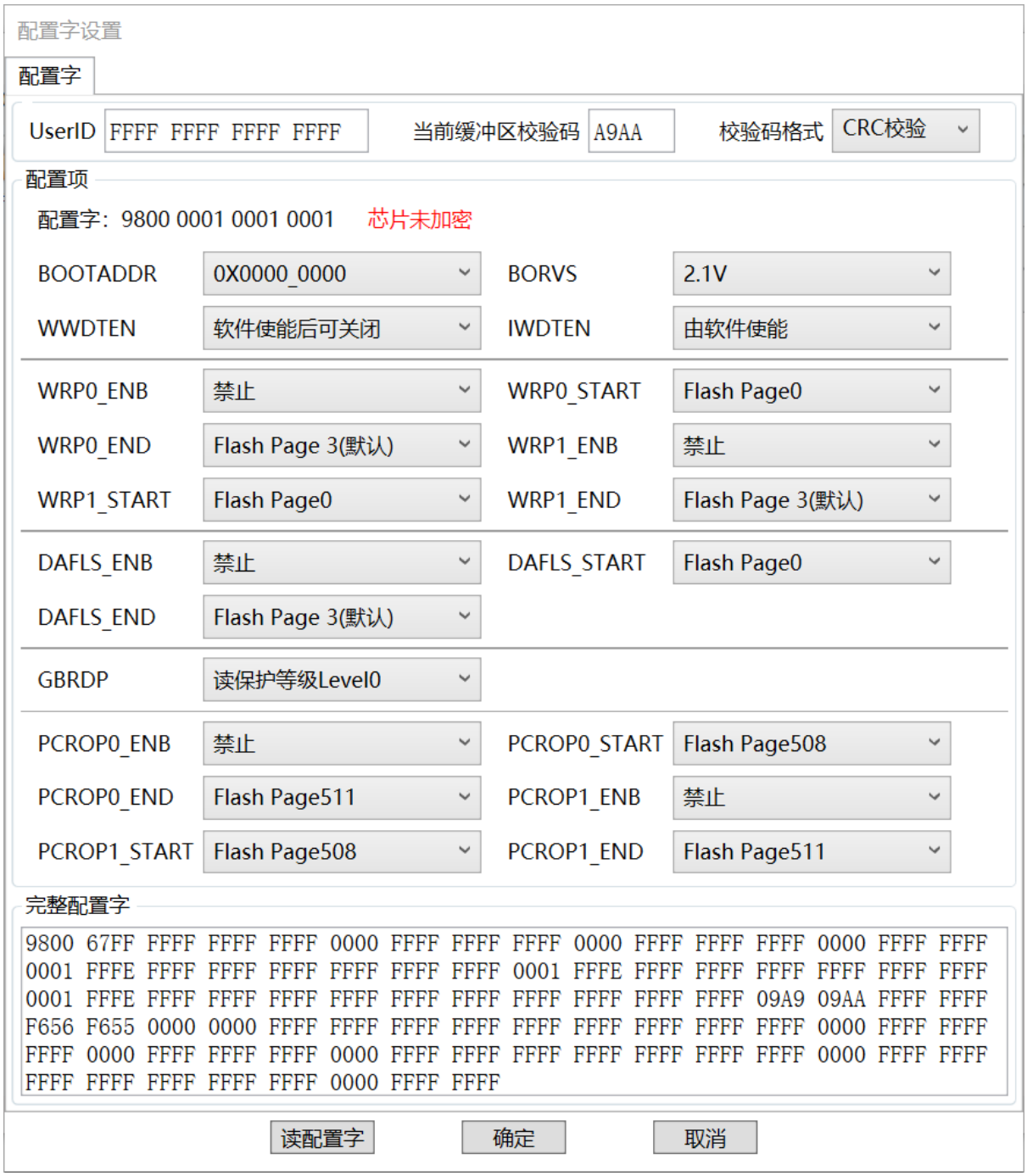


图 1-2　配置字界面

② CDK 集成了多个版本的 T-Head DebugServer，请选择 V5.16.8 及以上版本的 DebugServer。

③ debug 选项需添加 Other Flags: -vid 0x30cc -pid 0x9528，否则无法识别 ES 调试器。

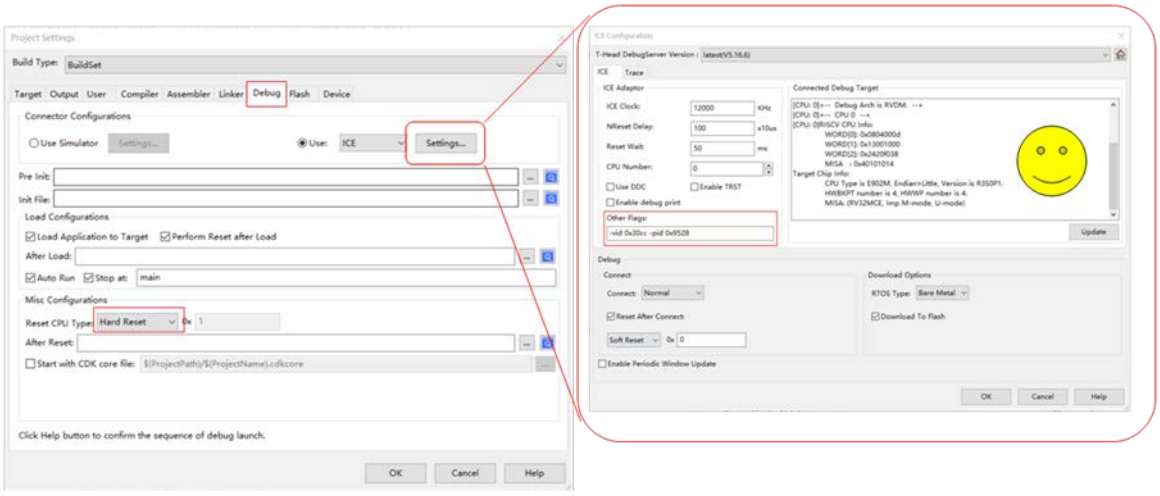


图 1-3　debug 选项添加 Other Flags: -vid 0x30cc -pid 0x9528

④ 将烧录算法文件*.elf 复制到 CDK 安装目录 C-Sky\CDK\CSKY\Flash 下，并在 CDK Option Flash 标签页 Add 烧录算法。烧录算法文件*.elf 位于 SDK 路径下的 Utilities 文件夹。

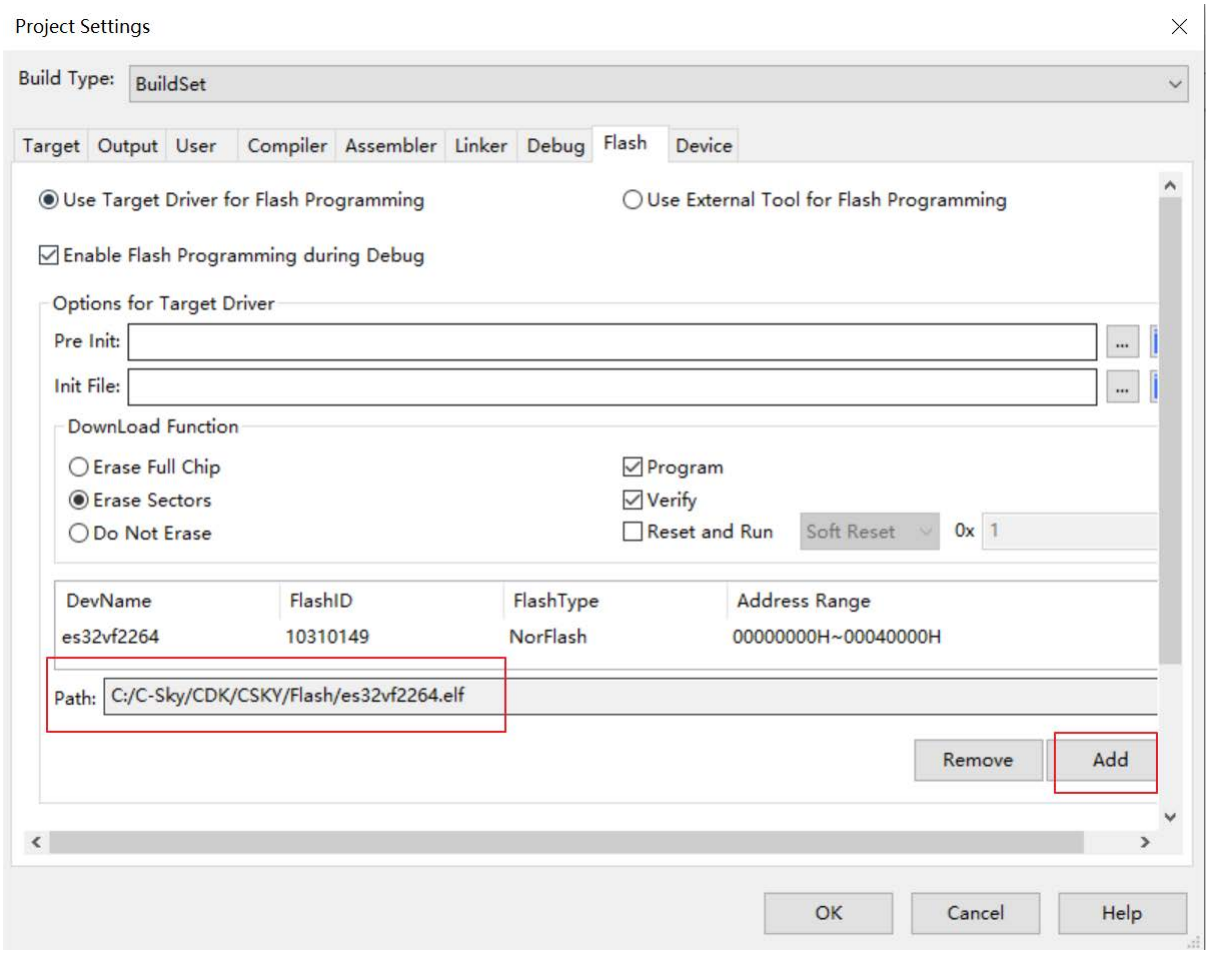


图 1-4　添加烧录算法

第2章 CDK使用注意事项

2. 1 优化等级

CDK 支持-O0，-Og，-O1，-O2，-O3，-Os 六种类型的代码优化模式：

-O0，不优化，每一条 C 语言都有与之相对应的汇编代码，故这时候调试信息也比较准确的，但缺点是代码比较冗余，性能较差；

-Og 是在开启部分优化的情况下保证调试信息的准确性，即使用该选项时不影响程序的调试；

-O1，-O2 开启优化，-O2 相对-O1 来说优化力度更大，得到最后的代码执行性能也会更好，所以在性能优先的条件下，建议使用-O2 优化；

-O3，在-O2 的基础上，进行基于性能优先的激进式优化，但是激进优化的结果并不稳定（可能导致代码过度膨胀，也可能导致性能下降），建议针对个别性能关键的函数进行优化，并测试结果，这样可能能提升性能。一般条件下，建议不要使用；

-Os，开启基于 Space 优先的优化。该优化会以代码密度为优先，尽可能优化生成的代码。在此选项下，生成的代码较小，比较适合空间敏感的 MCU 程序编译。

需要发挥内联函数特性时，优化等级不得低于-O2。当优化等级设置为-O0 时，如图 2-1 所示，内联函数 md_cmu_enable_perh_all()对应的反汇编指令为 jal，编译器将函数识别为普通函数，执行该函数仍需经过跳转过程；当优化等级设置为-O2 时，如图 2-2 所示，执行该函数无需跳转。

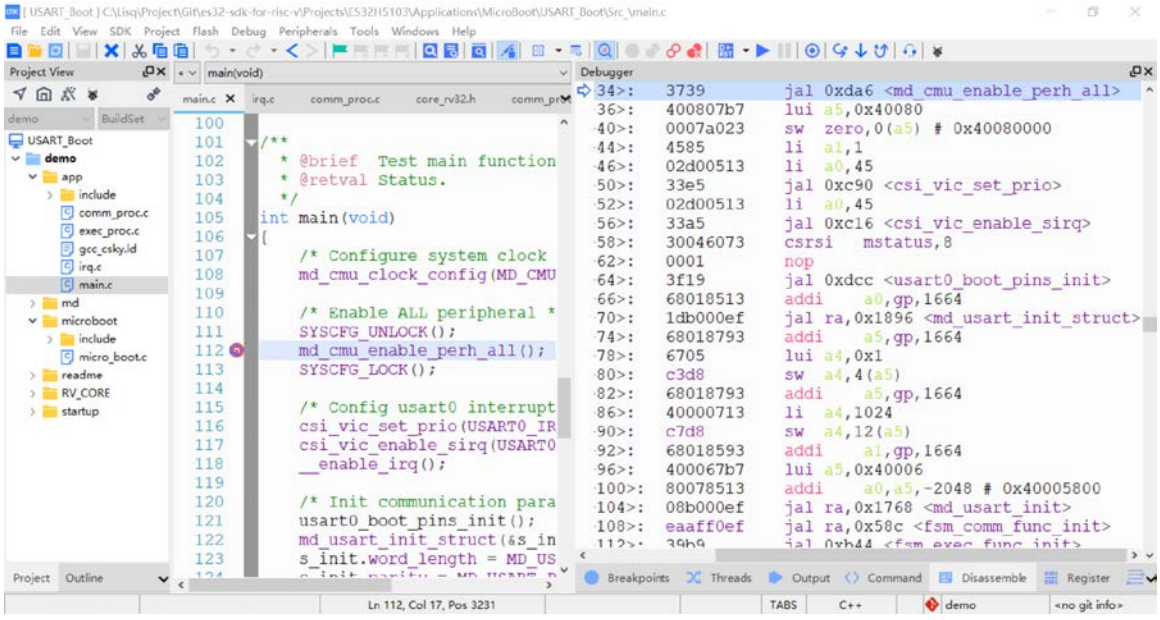


图 2-1 优化等级 O0 时内联函数对应的反汇编信息

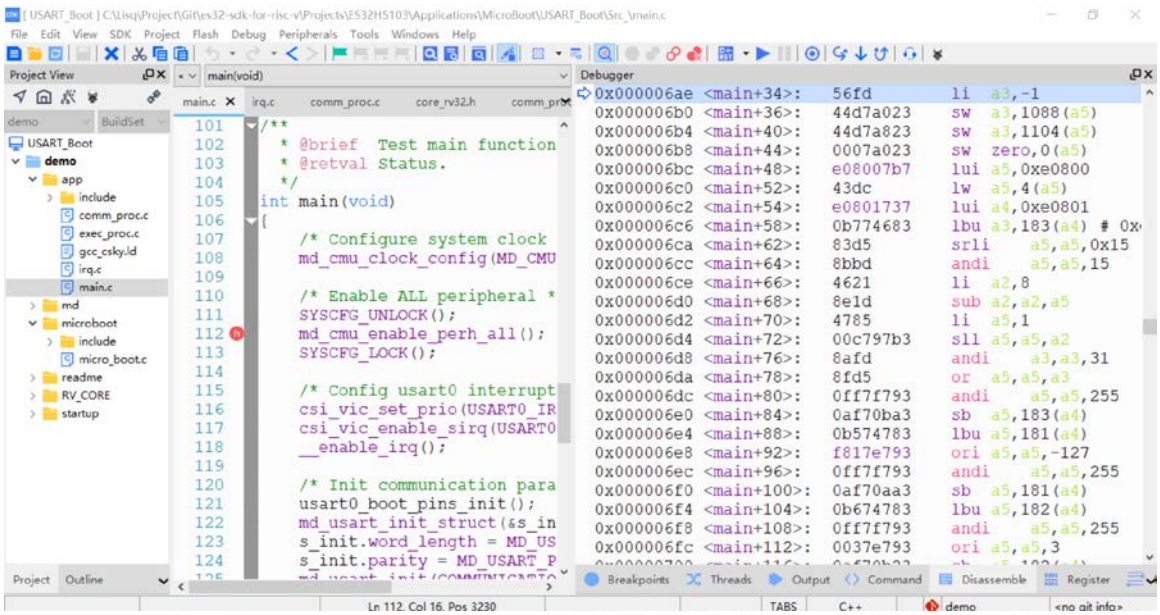


图 2-2 优化等级 O2 时内联函数对应的反汇编信息

2. 2 malloc等函数使用

C 标准库的 malloc、free 和 printf 函数在使用时，需在“Project Settings”的“Linker”标签页配置 flag: -specs=nosys.specs，如图 2-3、图 2-3-1 所示。针对不同 CDK 版本，用户操作如下表。

CDK 版本	操作
V2.22.0 <	参考图 2-3
V2.22.0 ~ V2.24.4	无
> V2.24.4	参考图 2-3-1

注意：V2.22.0 及以上版本的 CDK 集成了 nosys，如果使用 V2.22.0 及以上版本的 CDK，请勿手动添加 flag: -specs=nosys.specs，否则编译报错。

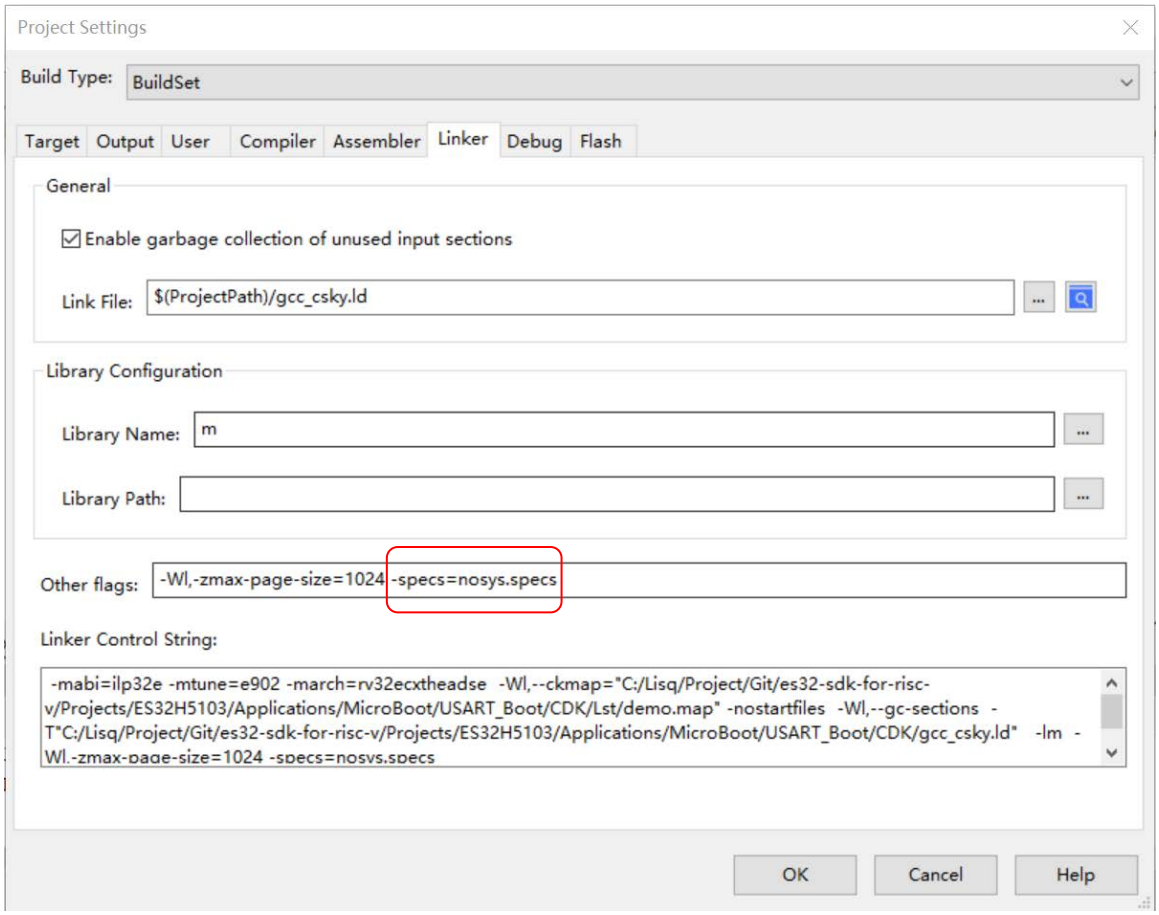


图 2-3　Linker 配置（CDK 2.22.0 以下）

V2.24.5 及以上版本的 CDK 需要选择 Link Specs，如下图所示。

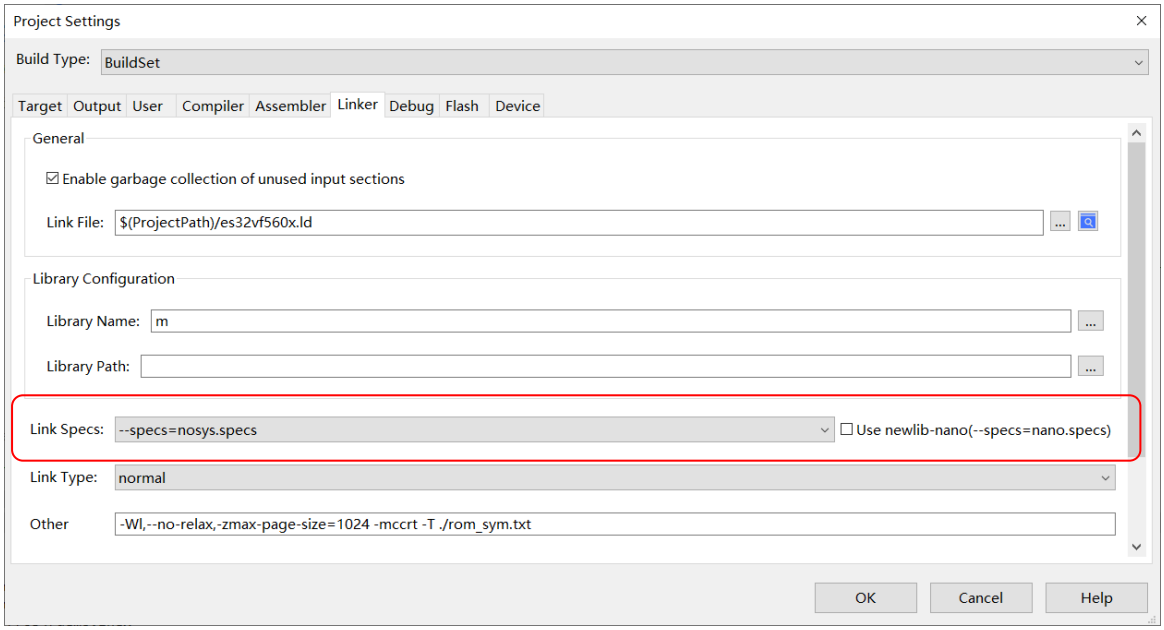


图 2-3-1　Linker 配置（CDK 2.24.5 以上）

2. 3 garbage collection

CDK 支持 garbage collection 功能，可以优化程序中的无用数据，使用该功能后大幅降低代码编译占用的空间。使用方法如图 2-4 所示。

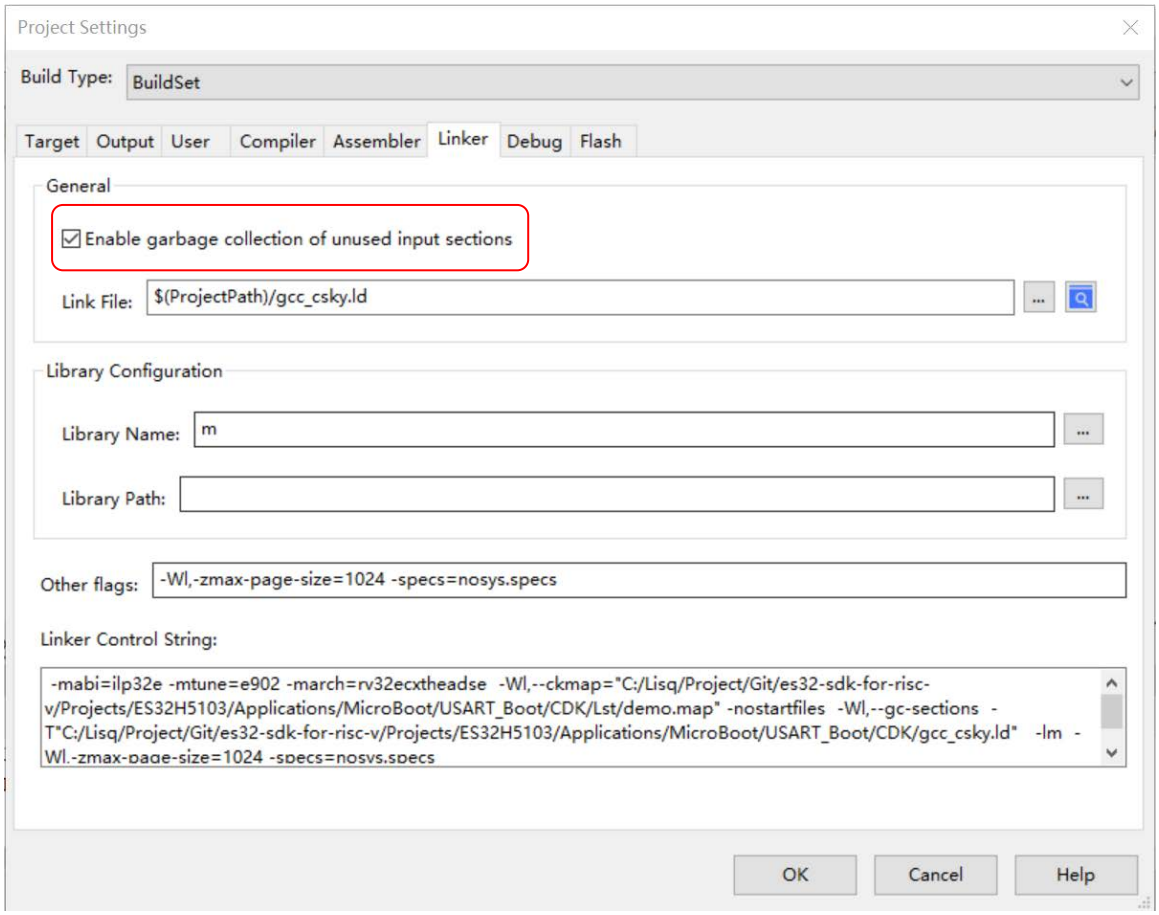


图 2-4 garbage collection 设置

需要注意的是，使用 garbage collection 后，编译器可能会误优化一些 section，此时可在链接文件*.ld 中用关键字 KEEP 强制保留一些特定的 section。

示例：

```
.rodata: {
        ……
        . = ALIGN(0x4);
        KEEP(*(.rti_fn.1))
        ……
}
```

如此，便可将.rti_fn.1 数据段强制保留下来，即使开启 garbage collection 模式，也不会将其优化掉。

2. 4 CDK调试选项

CDK 调试支持 ICE 和 Remote ICE 模式。Remote ICE 需配合独立的 T-HeadDebugServer 软件使用，ICE 模式可以配合 CDK 集成的 T-HeadDebugServer 插件使用。目前来看，CDK 集成的

T-HeadDebugServer 插件需升级到 V5.16 以上版本，才能稳定使用。

2. 5 SRAM运行程序

程序从 SRAM 地址启动，需对作如下配置。图 2-7 所示的配置容易被忽视，如果选择 Perform reset after load，CDK 的复位行为不会重新配置 PC，导致 PC 指向非 Reset_Handler 地址。

```
MEMORY
{
    I-SRAM : ORIGIN = 0x20004000 , LENGTH = 0x4000
    D-SRAM : ORIGIN = 0x20000000 , LENGTH = 0x4000
    O-SRAM : ORIGIN = 0x50000000 , LENGTH = 0x800000
    SRAM   : ORIGIN = 0x60000000 , LENGTH = 0x20000
}
```

图 2-5 *.ld 文件设置程序运行地址位于 SRAM 空间

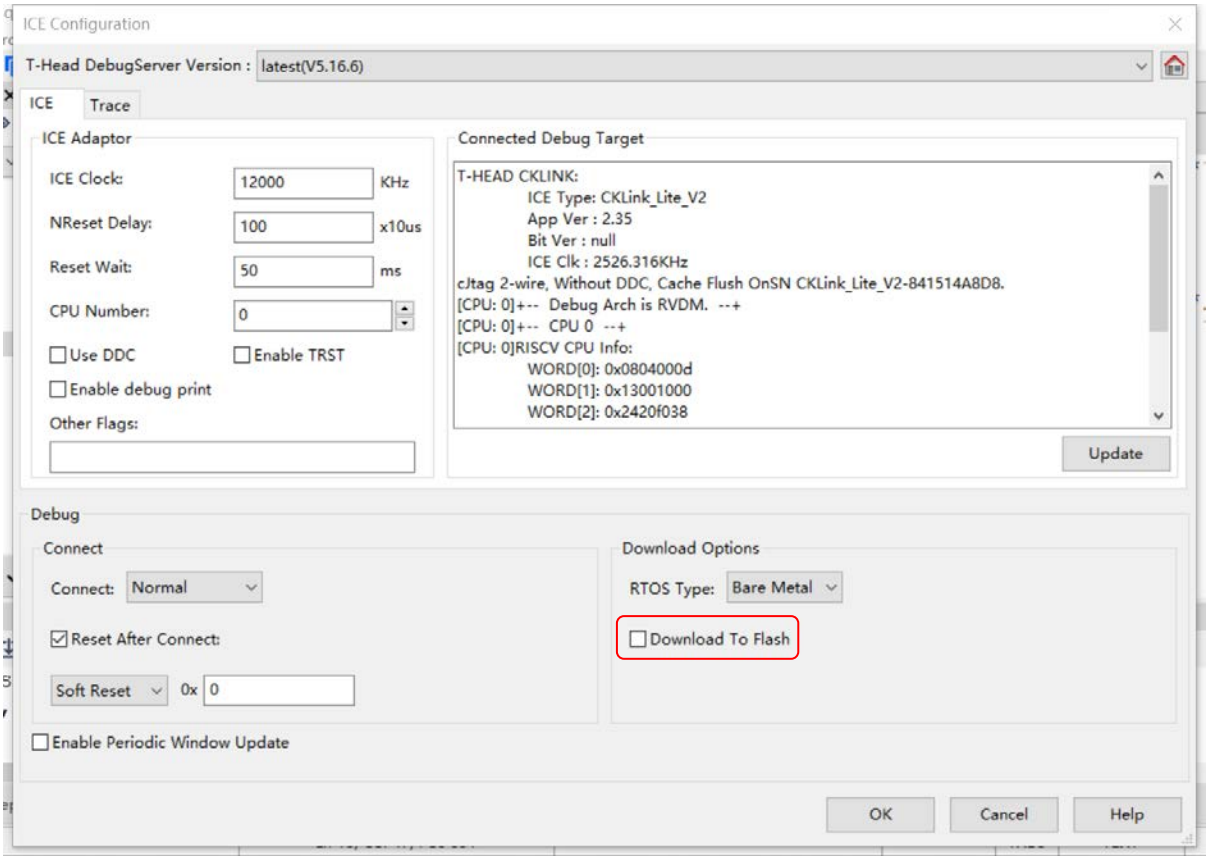


图 2-6 ICE Configuration 取消 Download To Flash

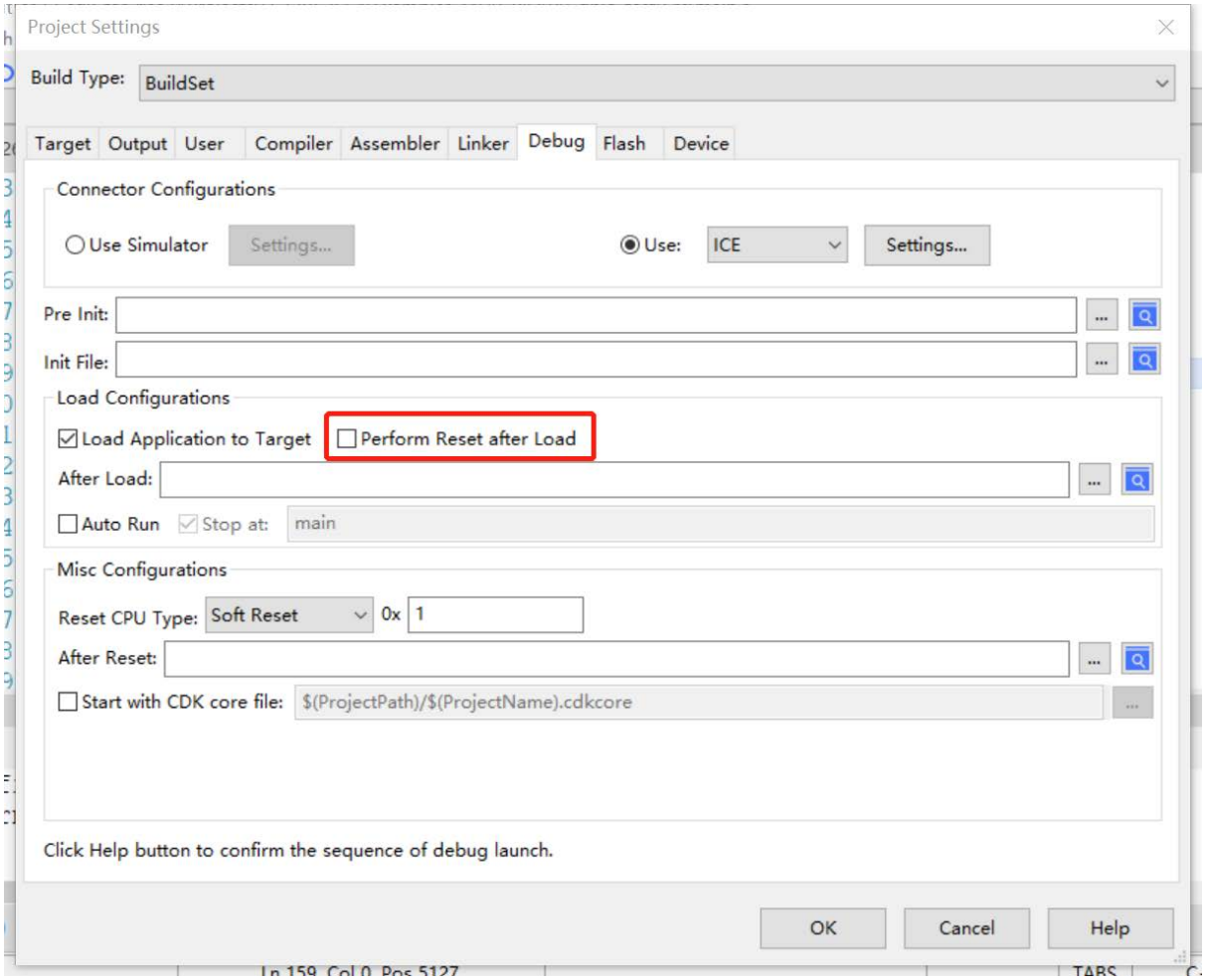


图 2-7　Debug 标签页取消 Perform Reset after Load

2. 6　中断延迟时间优化

CDK 优化等级对内核中断延迟时间是有影响的。优化等级 O0 时，中断服务程序入栈会多执行一些指令，中断延迟时间也会相应延长十几个时钟周期。优化等级 O2 时，中断延迟时间接近理论值，E902M 内核实测 12T 左右，平头哥给出的理论值 9T~13T。

2. 7　复位方式选择

CDK debug 可选择复位方式，如调试时只需复位 CPU 内核、中断模块，则选择 Soft Reset；如需系统级复位，则选择 Hard Reset，如图 2-8 所示。

同样的，CDK 烧录程序时的复位方式也可以选择 Soft 或 Hard 模式，在 ICE 设置里选择如图 2-9 所示。建议选择 Hard Reset 模式，若如此，烧录程序前，CDK 会对系统复位（包括时钟），否则，CDK 只会对内核等复位（时钟会保持复位前的配置）。

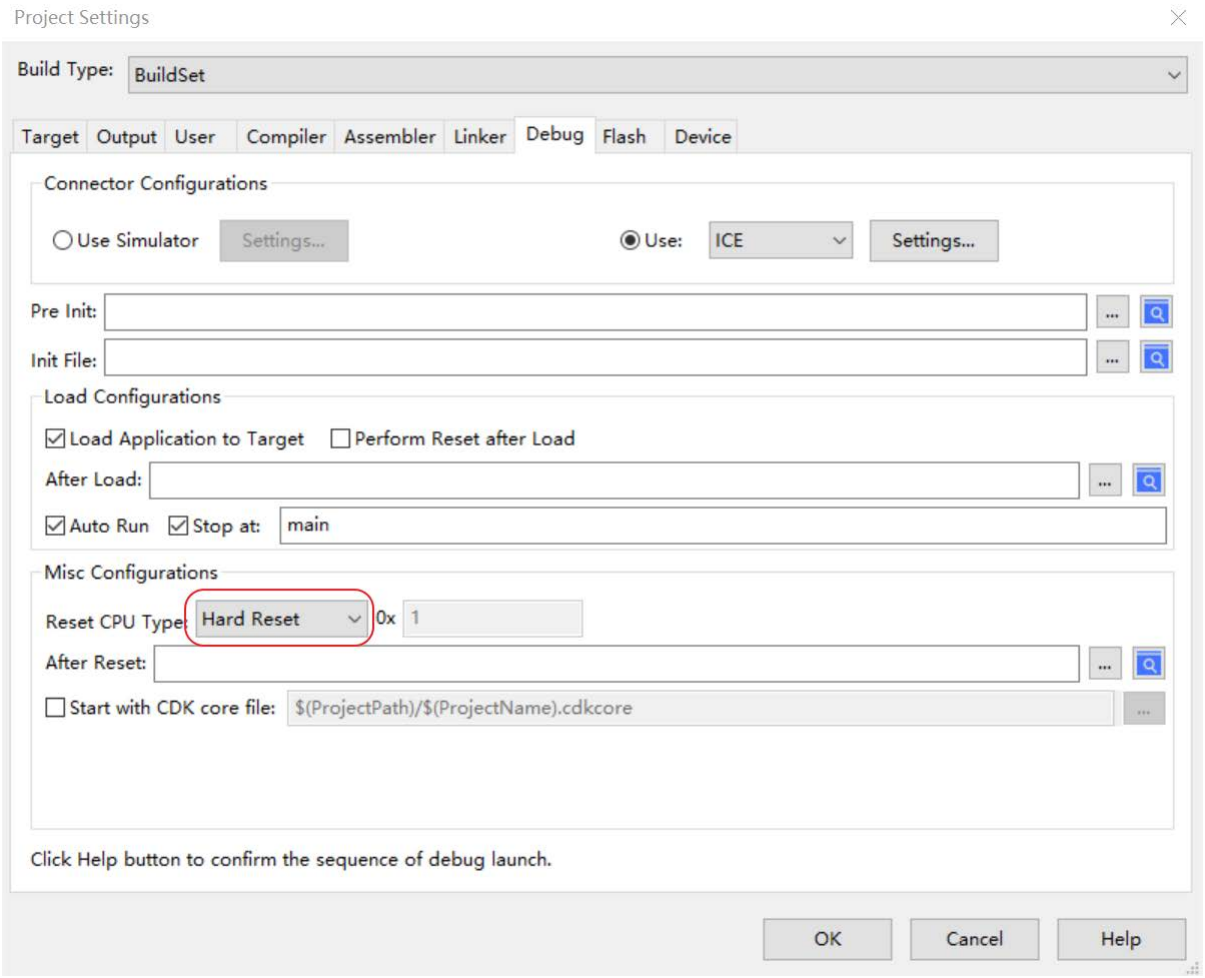


图 2-8　Debug 复位模式选择 Hard Reset

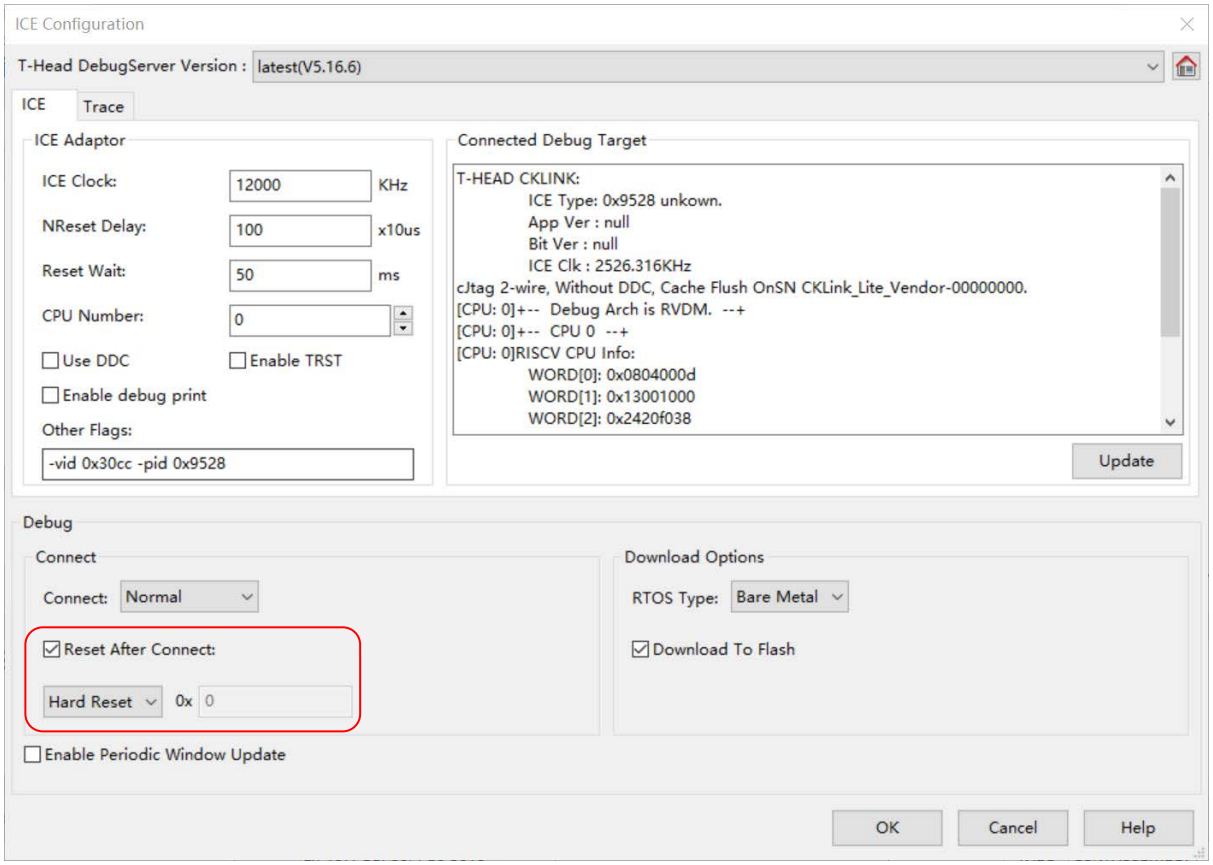


图 2-9　Reset After Connect 复位方式选择 Hard Reset

2. 8　codesize优化

CDK 编译 codesize 可优化，需在 Linker 标签页的 Other flags 栏添加“-mccrt”，注意与其他 flag 用空格隔开。不同工程优化程度不一样，与代码内容有关。

2. 9　硬件断点或软件断点的选择

CDK 调试 MCU 时，设置硬件断点会对调试运行效率产生一定影响。Dhrystone 程序实测，设置硬件断点后，ES32VF2264 性能下降 11%左右。

CDK 软件断点对调试运行效率无影响，但 CDK 默认前四个固定为硬件断点，后续为软件断点。如需全部断点为软件断点，请在 Options->Debug->ICE Settings->Other Flags 补充-no-hwbp 标签（需更新 CDK 至 V2.20 及以上版本，并选择 V5.16.8 及以上版本的 T-Head DebugServer）。

2. 10 使用乘除法指令

E902 系列内核支持扩展乘除法指令。如使用该指令，则需要选择 E902M 内核。具体操作如下：Options -> Device -> 点选 E902M -> OK。

2. 11 Download和Debug选择

程序烧录、调试方面，CDK 有 Download、Debug with download 和 Debug without download

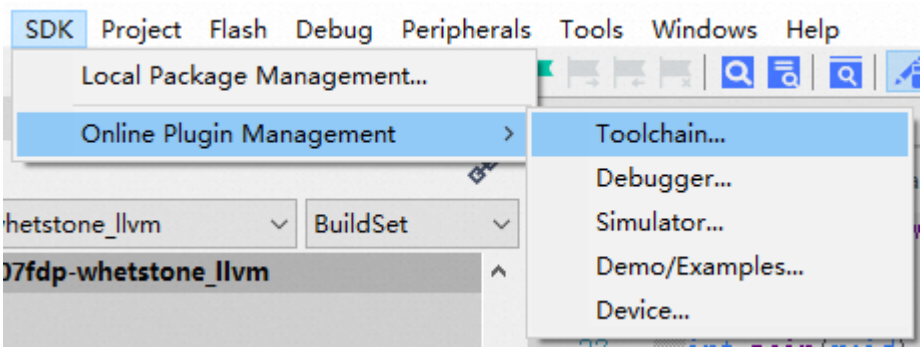
三种选择，分别对应程序仅烧录到 Flash、烧录到 Flash 后启动 Debugger 和不烧录直接启动 Debugger 三种功能。如果选择 Download，则在烧录完毕后程序会自动运行，Flash 标签页里的 Reset and run 选择与否只会影响运行前是否复位，而 run 的动作一定会发生。如需 download 后不自动运行程序，请选择 Debug with download 功能，并且不勾选 Option->Debug 标签页的 Auto Run 选项。

2. 12 玄铁LLVM工具链选择

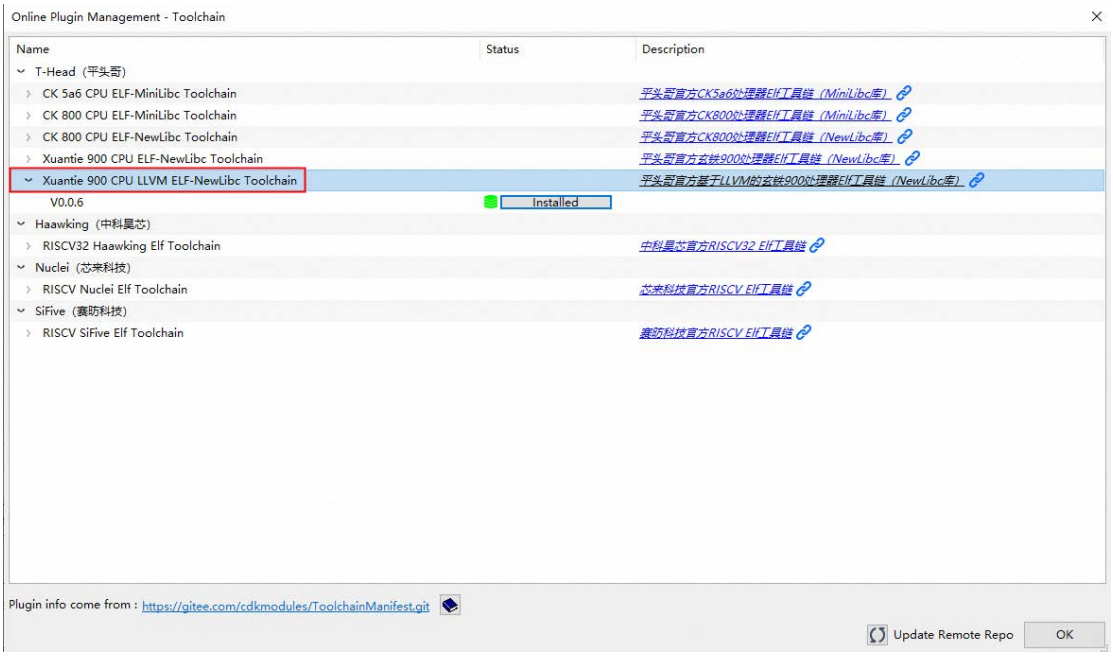
CDK 编译 E902 系列内核工程，默认的工具链是 GCC。从 V2.22.0 版本开始，CDK 增加支持玄铁 LLVM 工具链，有需要的用户只需对工程做简单配置，便可实现玄铁 LLVM 工具链的切换。

① 确认玄铁 LLVM 工具链插件是否发布。

点击 CDK 菜单栏 SDK->Online Plugin Management->Toolchain…



在弹出的对话框中确认 T-Head 节点下，是否存在 Xuantie900 CPU LLVM ELF-NewLibc Toolchain 的节点。

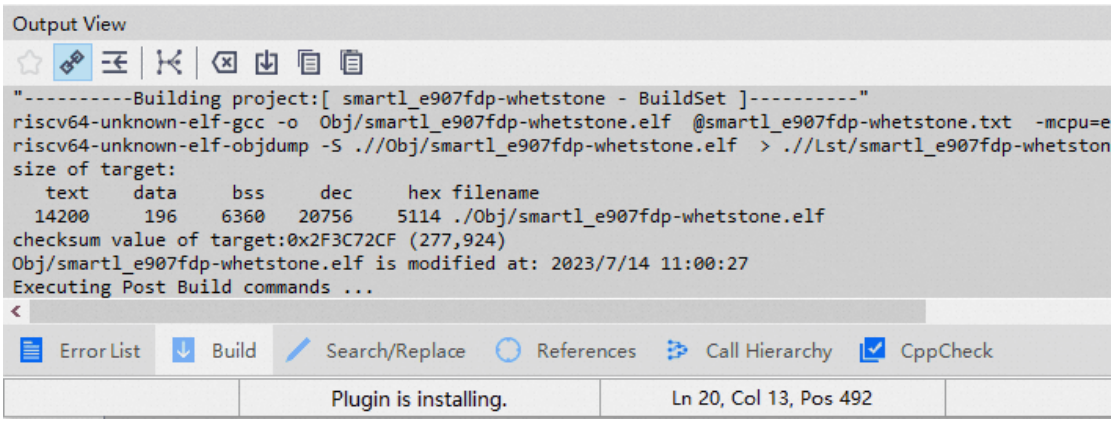


如果不存在此节点，需要点击底部的 Update Remote Repo，更新 CDK 插件库。

② LLVM 工具链插件的下载和安装。

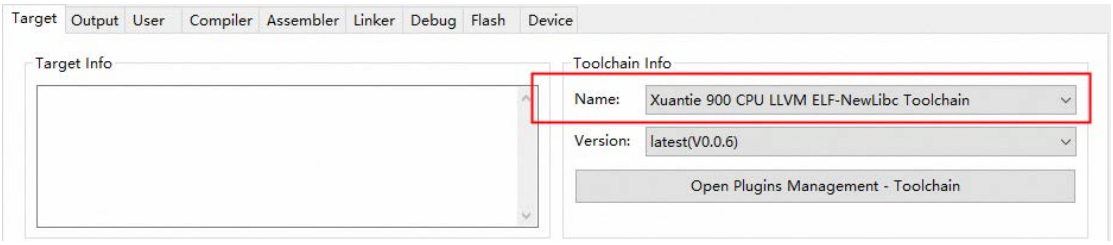
第①步中的 Online Plugin Management 窗口中，展开 Xuantie 900 CPU LLVM ELF-NewLibc Toolchain 节点，确认 V0.0.6 版本的状态，如果不是 Installed 状态，则点击 install，进行在线安装； 因为工具链镜像大小比较大（>500MB），所以下载和安装时间可

能会稍微长一些。等待 CDK 的下载和安装，直到 CDK 主界面底部状态栏中有绿色闪烁的 Plugin Installed 字样，表示插件已经安装完成了。



③ 工程选择玄铁 LLVM 工具链。

点击基于玄铁 900 系列处理的 CDK 工程配置界面，选择 Target Tab，并在该界面中的 Toolchain Info 处选择玄铁 LLVM 工具链。



点击 OK，即完成玄铁 LLVM 工具链的切换，后续的工程构建，都会使用玄铁 LLVM 工具链。

2. 13 变量、常量、函数指定到具体地址

CDK 编译 RISC-V MCU 使用的是 GCC 编译器，在该环境下为 MCU 的变量、常量或函数指定绝对地址，核心方法是结合使用编译器的段（Section）属性和自定义链接脚本（Linker Script）。具体方法如下。

①修改链接脚本（.ld 文件）

```
MEMORY
{
  RAM (xrw)     : ORIGIN = 0x20000000, LENGTH = 0x8000
  FLASH (rx)    : ORIGIN = 0x00000000, LENGTH = 0x3FFF8
  /* 例如，在 Flash 末尾划出 8 字节的自定义区域 */
  CUSTOM_FLASH (rx) : ORIGIN = 0x0003FFF8, LENGTH = 8
}
```

②定义自定义段及其地址

```
SECTIONS
{
  /* 其他标准段，如 .text, .data 等 ... */

  /* 方法 1：使用绝对地址直接定位 */
```

```
  .my_absolute_section 0x20001000 : /* 直接指定地址 0x20001000 */
  {
    . = ALIGN(4);   /* ALIGN(4)确保地址 4 字节对齐 */
    KEEP(*(SORT_NONE(.my_absolute_section))) /* KEEP 防止链接器优化掉 */
    . = ALIGN(4);
  } > RAM /* 指定此段在 RAM 区域 */

  /* 方法 2：在.ld 文件已定义的 MEMORY 区域内定位 */
  .version_section :
  {
    . = ALIGN(4);
    KEEP(*(.software_version)) /* 对应代码中的 section 名 */
    . = ALIGN(4);
  } > CUSTOM_FLASH /* 放置到之前定义的 CUSTOM_FLASH 区域 */
}
```

③在 C 代码中使用 __attribute__将变量、常量或函数放入自定义的段

```
/* 将常量定位到 Flash 的特定段 */
const char SoftwareVersion[] __attribute__((section(".software_version"))) = "V1.0.0.1";

/* 将变量定位到 RAM 的特定段 */
int __attribute__((section(".my_absolute_section"))) my_variable_at_fixed_addr;

/* 将函数定位到特定段，例如为了在 RAM 中全速运行 */
__attribute__((section(".ramfunc"))) void my_fast_function(void)
{
    /* 函数体 */
}
```

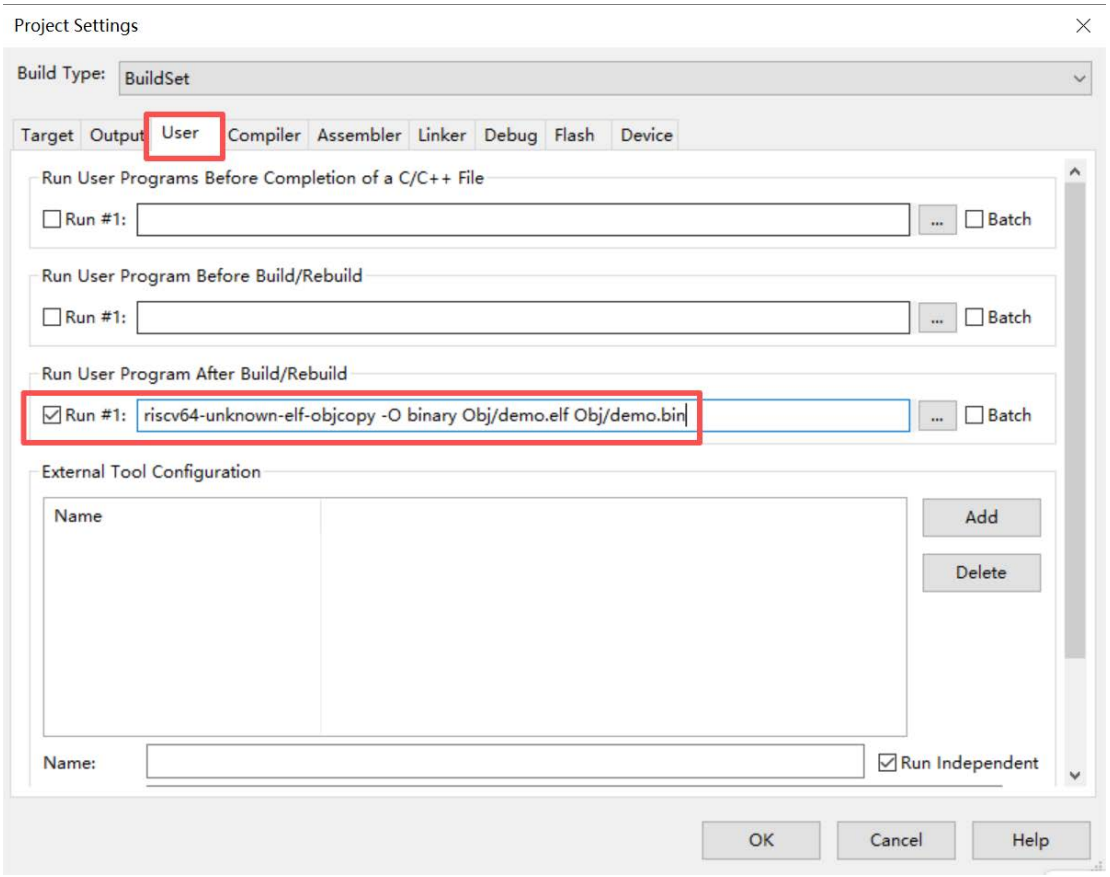
④另外，如果需要将整个 .c 文件的所有代码和数据都放置到特定地址，可以在链接脚本的 SECTIONS 中直接指定目标文件（.o）。

如：

```
SECTIONS
{
  .custom_code 0x00001000 : {
    *my_special_file.o(*) /* 将 my_special_file.o 中的所有段都放入此区域 */
} > FLASH
```

2. 14 bin文件合并

CDK 按照 section 生成 bin 文件，如果 section 不连续就会生成多个 bin 文件。这种情况下，如需这些 bin 文件合并为一个文件，则需要手动写 objcopy 命令。方法如下图所示。命令示范：riscv64-unknown-elf-objcopy -O binary Obj/demo.elf Obj/demo.bin。



注意：bin 文件合并后，数据为空的地址被 0x00 填充，如需填充 0xFF，则在上述命令后面加上--gap-fill 0xff。

第3章 CDK现有bug及规避方案

3. 1 断点无效

bug 现象：debug 时打断点，可能会出现全速运行后不在断点处停止，需鼠标点击暂停调试方能在断点处停下。该 bug 易出现在大量 for 循环后，系 Trace 采集数据过多导致卡住。

规避方法：Options->Debug->ICE Settings->Trace，取消 Enable PCSamples。

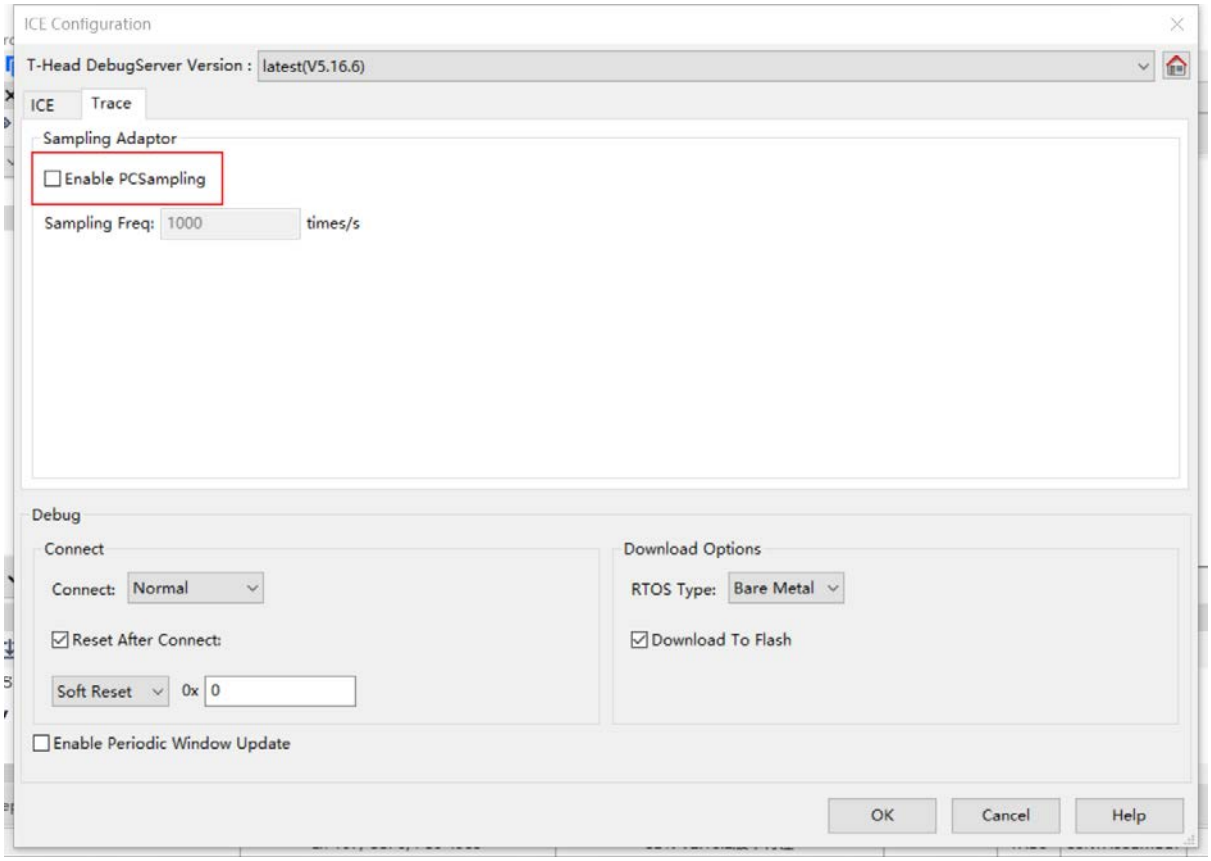


图 3-1 取消 Enable PCSamples

3. 2 for语句无法单步调试

bug 现象：debug 单步运行到 for 语句，若 for 循环内无内容，如

```
for(i=0;i<65535;i++);
```

或

```
for(i=0;i<65535;i++)
{
}
```

再往下单步运行，可能箭头不会回到 for 前面，也不会跳出 for 循环到下一句代码前面。这是由于 CDK 在 debug 时将无内容的 for 循环误认为只需执行一次的代码，循环次数较多的 for 实际需要的 Step Over 时间又比较长，所以既不回到 for 前面，也不跳出 for 循环。

规避方法：在 for 循环内加内容，如

```
for(i=0;i<65535;i++)
{
```

```
        i++;
        i--;
    }
```

便可在 for 循环内单步运行。或者在 for 循环后的一句打断点，全速运行，便可跳出 for 循环，停在断点处。