Application Note

IAR 至 CCS 项目移植指南

摘要

由于工具链、项目结构和配置存在差异，因此在不同 IDE 之间迁移嵌入式项目时需要仔细规划。本文档是一份条理清晰的指南，介绍了如何将项目从 IAR Embedded Workbench® 迁移到德州仪器 (TI) Code Composer Studio™ (CCS)。

内容

1 简介....2
2 迁移前准备工作....2
2.1 CCS 版本对比....2
2.2 迁移前准备工作....2
3 将代码移植至 CCS....4
3.1 移植准备....4
3.2 设置 CCS 环境....4
3.3 将源代码和文件导入 CCS....5
3.4 处理器件专属代码....6
3.5 针对 CCS 调整代码....7
3.6 构建和调试....7
4 迁移后优化....8
5 总结....8
6 参考资料....8

插图清单

图 3-1. 新建 CCS 项目....4
图 3-2. CCS 项目属性....5
图 3-3. 链接器文件比较....6

表格清单

表 2-1. CCS v20 与 CCS v12.8 的对比情况....2
表 2-2. 工具链和编译器的差异....2
表 2-3. 项目结构差异....3
表 2-4. 调试和硬件支持差异....3
表 2-5. 生态系统与集成差异....3
表 2-6. 构建和优化差异....3
表 3-1. 常用 IAR 标志与 CCS 等效标志对照表....6
表 3-2. IAR 与 CCS 中汇编代码比较示例....7

商标

1 简介

本应用手册旨在帮助开发人员将使用 IAR Embedded Workbench 编写的代码和项目迁移到 Code Composer Studio (CCS)。IAR 和 CCS 是两个应用广泛的嵌入式开发环境，分别由 IAR Systems 和德州仪器 (TI) 提供。两者在功能、界面设计和使用习惯方面存在差异，因此在迁移过程中需要特别注意兼容性和配置问题。本文档提供了从 IAR 迁移到 CCS 的具体步骤和注意事项，可帮助开发人员顺利完成代码迁移。虽然两者在某些方面存在差异，但通过仔细检查和调整，即可实现无缝过渡。对于新用户，TI 建议熟悉 CCS 的操作方法和界面布局，以便更快地适应这一环境。

2 迁移前准备工作

2.1 CCS 版本对比

在编写本文档期间，德州仪器 (TI) 发布了 Code Composer Studio™ (CCS) v20，这是一项重大架构改造，代表着从基于 Eclipse 的传统框架过渡到现代 Theia IDE 平台。尽管此次更新引入了增强型工具链集成和简化的用户界面，但此处展示的技术分析与方法主要基于 CCS v12.8 和更早的迭代版本。迁移到 CCS v20 对本文核心内容仅有极低限度的影响；但为了确保利用最新版环境的读者清晰了解情况，TI 在下一节中简要对比了 v12.8 和 v20 之间的关键差异。

表 2-1. CCS v20 与 CCS v12.8 的对比情况

	CCS v12.8 及更早版本	CCS v20
架构	Eclipse 富客户端平台	Eclipse Theia
优势	成熟且稳定，适合嵌入式开发中的深度可定制插件和工具链。	现代架构支持云或桌面混合工作流，原生兼容 VS Code 扩展并可与 DevOps 无缝集成。
劣势	依赖传统技术，对现代 Web 标准的支持有限，内存和资源使用量更高。	相较于 Eclipse，社区生态系统规模较小；部分高级插件需要第三方调整。
用户体验	具有嵌套式菜单的经典多窗口布局，学习曲线较为陡峭。	界面类似于 VS Code，支持拖放式面板自定义（例如、终端、内存视图）。

2.2 迁移前准备工作

在开始迁移之前，请熟悉 IAR Embedded Workbench (EW) 和 Code Composer Studio 在工具链、项目管理和生态系统集成方面的差异。下面简要列举了这些差异。

1. 工具链和编译器：IAR 使用专有编译器，而 CCS 通常使用 TI 的编译器（基于 GCC 或 Clang）或其他受支持的编译器。

表 2-2. 工具链和编译器的差异

IAR EW	CCS
使用 IAR 专有编译器（适用于 ARM 的 ICCARM）。	为 TI 器件使用 TI Arm Clang（基于 LLVM/Clang）。
为 defines 使用 --debug、-OH、-D 等标志。	使用不同的标志（例如、-g 表示 debug、--define=name 表示宏）。
严格遵守 IAR 特有语法（例如 #pragma vector）。	需要与 TI 兼容的语法（例如 __attribute__((interrupt))）。

2. 工程结构：IAR 和 CCS 具有不同的项目文件结构与配置。

表 2-3. 项目结构差异

IAR EW	CCS
专有项目格式（.ewp、.eww）。	基于 Eclipse 的项目（.cproject、.project）。
通过 GUI 或 .icf 链接器文件管理设置。	使用链接器命令文件 (.cmd) 和 Eclipse 风格的配置菜单。
插件生态系统有限。	可通过 Eclipse 插件进行扩展（例如，TI Resource Explorer、GIT 集成）。

3. 调试工具及硬件支持：CCS 集成了 TI 专属调试工具，这些工具可能不同于 IAR 的调试环境。

表 2-4. 调试和硬件支持差异

IAR EW	CCS
丰富的第三方调试探针。	支持 TI 调试探针（XDS110 等）和第三方调试探针
需要手动设置 HAL。	预先集成的 TI 库（例如 TI-RTOS、FreeRTOS）。
有限的 RTOS 集成。	原生支持 TI-RTOS 和实时调试工具。

4. 生态系统与集成：CCS 可免费使用，并且支持可帮助用户设计项目的多种工具。

表 2-5. 生态系统与集成差异

IAR EW	CCS
免费功能有限，许可证须付费。	提供免费套餐，可选择付费升级。
极少的供应商专属工具。	与 TI 工具（例如 UniFlash、SysConfig）紧密集成。
通过 IAR 论坛提供社区支持。	强大的 TI 社区（E2E 论坛、详细的应用手册）。

5. 编译和优化：CCS 支持多种优化级别，以满足不同要求。

表 2-6. 构建和优化差异

IAR EW	CCS
因高度优化的代码而著称。	通过 TI 特有调优来平衡优化。
通过 GUI 自定义构建步骤。	使用 Eclipse 或 Makefile 灵活定制构建。
通过 .icf 分配静态内存。	动态链接器配置（.cmd 文件）。

3 将代码移植至 CCS

3.1 移植准备

首先了解 IAR 项目结构。将项目层次结构、源文件和依赖关系记录下来。请注意目标微控制器 (MCU) 和型号（例如 TI MSP430、Arm® Cortex®-M）。识别编译器和链接器标志、内存配置（.icf/.xcl 文件）以及预处理器符号。

第二，备份项目。创建原始 IAR 项目的副本，以供参考和回滚。

第三，检查工具链特有功能的代码。检查 IAR 专属 pragma、内在函数（例如 __no_init）或内联汇编等问题。确定对 IAR 库或运行时文件（例如 low_level_init.c）的依赖关系。记下重要设置，例如：编译器标志、链接器配置、内存布局（例如链接器脚本或 ICF 文件）、预处理器定义和包含路径。

最后，确定项目依赖关系。列出 IAR 项目中使用的所有外部库、驱动程序和中间件。

3.2 设置 CCS 环境

1. 安装 CCS：
   a. 从 TI 网站下载并安装最新版本的 Code Composer Studio。确保 CCS 中安装了所需的器件支持包（内核、编译器和调试器）。
   b. 通过 TI Resource Explorer 或 TI 网站安装器件专属 SDK（例如 MSP432、C2000 或 SimpleLink SDK）。
   c. 安装所需库。如果项目使用 TI 专属库（例如 DriverLib、TivaWare 或 MSPWare），请下载并安装这些库。
2. 新建 CCS 项目：
   a. File → New → CCS Project。
   b. 选择目标 MCU、编译器 (TI Arm Clang) 和项目模板（例如 *Empty Project*）。
   c. 确保输出格式（例如 ELF）与目标匹配。

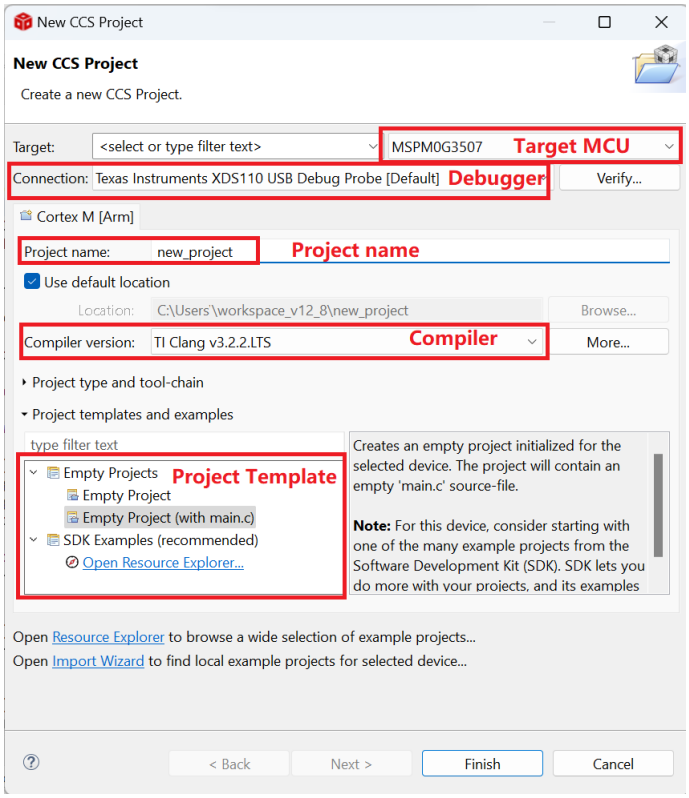

图 3-1. 新建 CCS 项目

3.3 将源代码和文件导入 CCS

1. 复制源文件：
   a. 将源文件（“.c”、“.h”、“.asm”）从 IAR 项目复制到 CCS 项目目录。
   b. 使用 Project Explorer → 右键单击 *Import* → *File System* 来添加文件。
2. 包含路径和预处理器符号：
   a. 在项目属性中添加必要的包含路径（右键单击 *Project* > Properties > Build > Include Options）。如下图所示。
   b. 在 *Predefined Symbols* 下，按需定义宏。

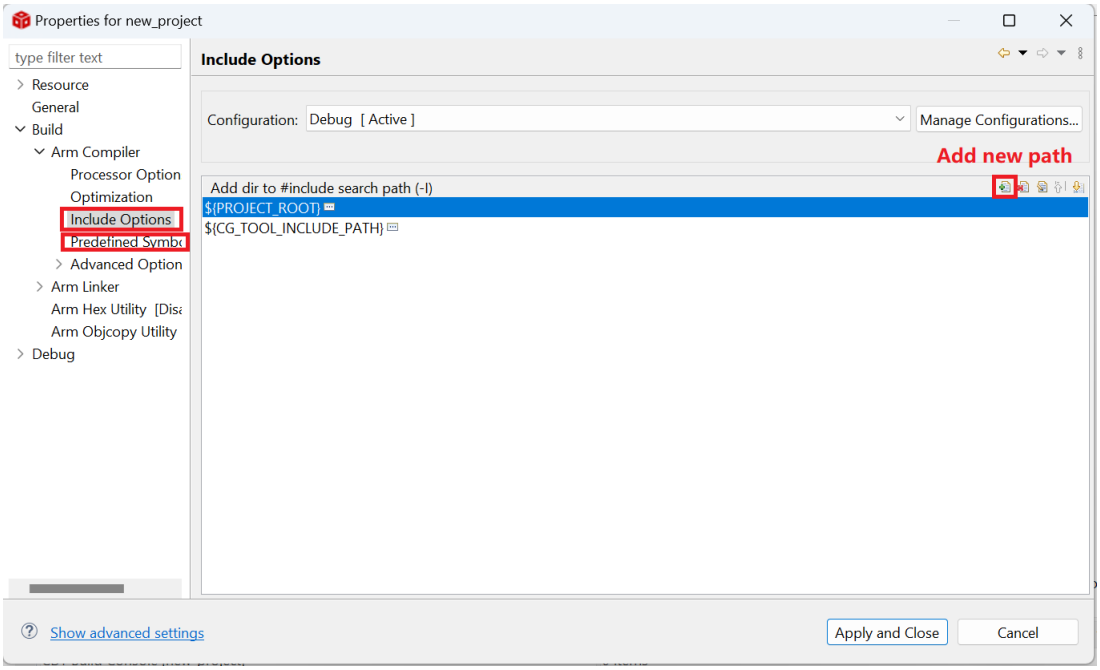

图 3-2. CCS 项目属性

3. 链接器配置：
   a. 将 IAR .icf/.xcl 替换为一个 TI 链接器命令文件 (.cmd)。
   b. 在 .cmd 文件中配置内存区域（例如 FLASH、RAM）。如果用户不熟悉 .cmd 文件，则需要参阅 TI 链接器命令文件入门，了解代码的基本说明，大多数 TI 链接器命令文件中通常都包含此入门内容。

A typical IAR .icf file

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_4.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x0000000;

/*-Memory Regions-*/
define symbol __ICFEDIT_region_IROM1_start__ = 0x00000000;
define symbol __ICFEDIT_region_IROM1_end__   = 0x0001FFFF;
define symbol __ICFEDIT_region_IROM2_start__ = 0x0;
define symbol __ICFEDIT_region_IROM2_end__   = 0x0;
define symbol __ICFEDIT_region_EROM1_start__ = 0x0;
define symbol __ICFEDIT_region_EROM1_end__   = 0x0;
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__ = 0x20200000;
define symbol __ICFEDIT_region_IRAM1_end__   = 0x20207FFF;
define symbol __ICFEDIT_region_IRAM2_start__ = 0x0;
define symbol __ICFEDIT_region_IRAM2_end__   = 0x0;
define symbol __ICFEDIT_region_ERAM1_start__ = 0x0;
define symbol __ICFEDIT_region_ERAM1_end__   = 0x0;
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;
define symbol __ICFEDIT_region_NON_MAIN_BCR_start__ = 0x41C00000;
define symbol __ICFEDIT_region_NON_MAIN_BCR_end__   = 0x41C0007F;
define symbol __ICFEDIT_region_NON_MAIN_BSL_start__ = 0x41C00100;
define symbol __ICFEDIT_region_NON_MAIN_BSL_end__   = 0x41C0017F;
/*-Sizes-*/
define symbol __ICFEDIT_size_proc_stack__ = 0x0000;
```

A typical CCS .cmd file

```
-uinterruptVectors
--stack_size=256


MEMORY
{
    FLASH           (RX)  : origin = 0x00000000, length = 0x00040000
    SRAM            (RWX) : origin = 0x20200000, length = 0x00008000
    BCR_CONFIG      (R)   : origin = 0x41C00000, length = 0x000000FF
    BSL_CONFIG      (R)   : origin = 0x41C00100, length = 0x00000080
}

SECTIONS
{
    .intvecs:   > 0x00000000
    .text   : palign(8) {} > FLASH
    .const  : palign(8) {} > FLASH
    .cinit  : palign(8) {} > FLASH
    .pinit  : palign(8) {} > FLASH
    .rodata : palign(8) {} > FLASH
    .ARM.exidx    : palign(8) {} > FLASH
    .init_array   : palign(8) {} > FLASH
    .binit        : palign(8) {} > FLASH
    .TI.ramfunc   : load = FLASH, palign(8), run=SRAM, table(BINIT)

    .vtable :   > SRAM
    .args   :   > SRAM
    .data   :   > SRAM
    .bss    :   > SRAM
    .sysmem :   > SRAM
    .stack  :   > SRAM (HIGH)

    .BCRConfig  : {} > BCR_CONFIG
    .BSLConfig  : {} > BSL_CONFIG
}
```

图 3-3. 链接器文件比较

4. 转换编译器和链接器标志：
   a. 在链接器文件中或通过 Project Properties → Build → ARM Linker → Basic Options 设置栈或堆大小。
   b. 将 IAR 标志映射到 TI Arm Clang 的等效项：

表 3-1. 常用 IAR 标志与 CCS 等效标志对照表

IAR 标志	CCS 等效标志 (TI Arm Clang)	用途
--debug	-g	调试符号
-Oh	-O3	高度优化
-DNAME	--define=NAME	定义预处理器宏
--cpu=cortex-m4	-mcpu=cortex-m4	目标 CPU
-I<path>	-I<path>	包含目录
--data_model medium	不需要（在 .cmd 中配置）	内存模型

3.4 处理器件专属代码

1. 替换 IAR 启动代码：
   a. 使用 TI 提供的启动文件（例如 SDK 中的 startup_<device>.c），而非 IAR 的 startup_<device>.s。
   b. 更新中断向量表以匹配 TI 的语法（例如，向量的 #pragma DATA_SECTION）。
2. 调整硬件抽象：
   a. 将 IAR 专属 HAL 函数替换为 TI DriverLib 或基于寄存器的代码。
   b. 示例：使用 MAP_GPIO_setAsOutputPin()，而非 IAR 的 GPIO 库。
3. 更新内联汇编和 pragma：
   a. 重写 IAR 专属 pragma（例如将 __packed 重写为 __attribute__((packed))）。
   b. 将内联汇编转换为 TI Clang 语法。

表 3-2. IAR 与 CCS 中汇编代码比较示例

IAR EW	CCS
#pragma asm MOV R0，#0x10 #pragma endasm	__asm(" MOV R0, #0x10 ");

3.5 针对 CCS 调整代码

1. 修复编译器专属代码：
   a. 对于某些构造（例如内联汇编、pragma），IAR 和 CCS 编译器可能具有不同的语法或行为。
   b. 更新任何编译器专属代码，使其兼容 CCS。
2. 替换 IAR 专属函数：
   a. 将 IAR 专属函数（例如“__enable_interrupt()”）替换为等效的 CCS 或 TI 专属函数。
3. 更新调试代码：
   a. 如果项目使用 IAR 专属调试宏或函数，请将这些宏或函数替换为兼容 CCS 的替代项。

3.6 构建和调试

1. 构建和验证项目：
   a. 如有构建错误，请先修复错误。先在 Problems 控制台中查看错误报告，随后找到并修复错误。常见问题包括 include 缺失、宏未定义以及语法不匹配。
   b. 调整编译器优化级别及其他性能或尺寸设置。有关优化选项的更详细说明，请参阅 TI ARM Clang 编译器用户手册。
2. 创建调试配置：
   a. Run → Debug Configurations → New Launch Configuration。
   b. 选择目标连接（XDS110、JTAG、SWD）。
   c. 使用 .ccxml 文件定义调试探针和 MCU。
3. 在硬件上运行并调试：
   a. 将目标微控制器连接到 CCS 并加载程序。
   b. 加载程序，并验证断点、寄存器视图和内存检查。确认程序是否正常运行且性能符合预期。
   c. 将行为与原始 IAR 项目进行比较。

4 迁移后优化

1. 将迁移记录到文档中：
   a. 记录迁移过程中做出的所有更改，以备将来参考。
   b. 更新项目文档，以体现新的 CCS 环境。
2. 常见问题：
   a. 中断处理程序：确保 ISR 使用 #pragma WEAK 或者使用正确的名称（例如 void TIMER0_A0_ISR(void)）。
   b. 内存对齐：TI Clang 可强制实施比 IAR 更严格的对齐。如有需要，请使用 __attribute__((aligned(8)))。
   c. 链接器错误：验证 .cmd 文件地址是否与 MCU 内存映射匹配。

5 总结

阅读本应用手册后，用户可以系统地将项目转换到 CCS，同时处理工具链方面的细微差别。采用增量的方式进行测试，并利用 TI 强大的调试工具简化验证。

6 参考资料

- 德州仪器 (TI)，ARM-CGT，网页
- IAR Systems AB，IAR Embedded Workbench，网页
- 德州仪器 (TI)，CCSTUDIO，网页

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265