

ABSTRACT

Migrating embedded projects between IDEs requires careful planning due to differences in toolchains, project structures, and configurations. This document is a structured guide to transition projects from IAR Embedded Workbench[®] to Texas Instruments' Code Composer Studio[™] (CCS).

Table of Contents

| 1 Introduction | 2 |
|---|---|
| 2 Pre-Migration Preparation | 2 |
| 2.1 CCS Version Comparison | 2 |
| 2.2 Pre-Migration Preparation | 2 |
| 3 Porting Code to CCS | 4 |
| 3.1 Prepare for Porting | 4 |
| 3.2 Set Up CCS Environment | 4 |
| 3.3 Import Source Code and Files in CCS | 5 |
| 3.4 Handle Device-Specific Code | 6 |
| 3.5 Adapt Code for CCS | 7 |
| 3.6 Build and Debug | 7 |
| 4 Post-Migration Optimization | 8 |
| 5 Summary | 8 |
| 6 References | 8 |

List of Figures

| Figure 3-1. Create a New CCS Project | .4 |
|--------------------------------------|----|
| Figure 3-2. CCS Project Properties | 5 |
| Figure 3-3. Linker File Comparison | 6 |

List of Tables

| Fable 2-1. Comparison between CCS v20 and CCS v12.8 | . 2 |
|---|-----|
| Table 2-2. Toolchain and Compiler Differences | . 2 |
| Fable 2-3. Project Structure Differences | . 3 |
| Fable 2-4. Debugging and Hardware Support Differences | . 3 |
| Table 2-5. Ecosystem & Integration Differences | 3 |
| Table 2-6. Build and Optimization Differences | 3 |
| Fable 3-1. A Comparsion Table of Common Used IAR Flag vs CCS Equivalent | 6 |
| Table 3-2. Comparison of Assembly Code Example in IAR and CCS | .7 |

Trademarks

Code Composer Studio[™] are trademarks of Texas Instruments. IAR Embedded Workbench[®] is a registered trademark of IAR Systems AB. Arm[®] and Cortex[®] are registered trademarks of Arm Limited.

All trademarks are the property of their respective owners.

1



1 Introduction

This application note is designed to assist developers in migrating code and projects written with IAR Embedded Workbench to Code Composer Studio (CCS). IAR and CCS are two widely used embedded development environments provided by IAR Systems and Texas Instruments, respectively. There are differences in functionality, interface design, and usage habits between the two, so special attention needs to be paid to compatibility and configuration issues during the migration process. This document provides detailed steps and considerations for migrating from IAR to CCS, helping developers complete the code migration smoothly. Although there are differences between the two in some aspects, careful inspection and adjustments can achieve a seamless transition. For new users, TI recommends to be familiar with the operation methods and interface layout of CCS to adapt more quickly.

2 Pre-Migration Preparation

2.1 CCS Version Comparison

During the preparation of this document, Texas Instruments released Code Composer Studio[™] (CCS) v20, a significant architectural overhaul transitioning from the legacy Eclipse-based framework to the modern Theia IDE platform. While this update introduces enhanced toolchain integration and a streamlined user interface, the technical analysis and methodologies presented herein remain primarily grounded in CCS v12.8 and earlier iterations. The migration to CCS v20 has minimal bearing on the core content of this article; however, to make sure of clarity for readers utilizing the latest environment, TI provide a concise comparison of critical differences between v12.8 and v20 in the following section.

| | CCS v12.8 and Earlier | CCS v20 | |
|---|--|--|--|
| Architecture | Eclipse Rich Client Platform | Eclipse Theia | |
| Strengths Mature and stable, good for deeply customizable plugins and toolchains in embedded development. | | Modern architecture supporting cloud or desktop hybrid workflows, native compatibility with VS Code extensions, and seamless DevOps integration. | |
| Weaknesses | Relies on legacy technology, limited support for modern web standards, and higher memory and resource usage. | Smaller community ecosystem compared to Eclipse; some advanced plugins require third-party adaptation. | |
| User Experience | Classic multi-window layout with nested menus and a steep learning curve. | VS Code-like interface with drag-and-drop panel customization (for example, terminal, memory views). | |

|--|

2.2 Pre-Migration Preparation

Before starting the migration, be familiar with the differences between IAR Embedded Workbench (EW) and Code Composer Studio that are in the toolchains, project management, and ecosystem integration. A concise breakdown of the differences are shown below.

1. Toolchain and Compiler: IAR uses a proprietary compiler, while CCS typically uses TI's compiler (based on GCC or Clang) or other supported compilers.

| Table 2-2. Toolchain and Compiler Difference | s |
|--|---|
|--|---|

| IAR EW | CCS |
|--|--|
| Uses proprietary compiler of IAR (ICCARM for ARM). | Uses TI Arm Clang (based on LLVM/Clang) for TI devices. |
| Flags likedebug, -Oh, -D for defines. | Flags differ (for example, -g for debug,define=NAME for macros). |
| Strict adherence to IAR-specific syntax (for example, #pragma vector). | Requires TI-compatible syntax (for example, attribute ((interrupt))). |



2. Project Structure: IAR and CCS have different project file structures and configurations. **Table 2-3. Project Structure Differences**

| IAR EW | CCS |
|--|--|
| Proprietary project format (.ewp, .eww). | Eclipse-based project (.cproject, .project). |
| Manages settings via GUI or .icf linker files. | Uses linker command files (.cmd) and Eclipse-style configuration menus. |
| Limited plugin ecosystem. | Extensible via Eclipse plugins (for example, TI Resource Explorer, GIT integration). |

3. Debugging Tools and Hardware Support: CCS integrates TI-specific debugging tools, which can differ from the debugging environment of the IAR.

Table 2-4. Debugging and Hardware Support Differences

| IAR EW | CCS | |
|---------------------------------|---|--|
| Broad third-party debug probes. | Support TI debug probes (XDS110 and so forth.) and third party debug probes | |
| Requires manual HAL setup. | Pre-integrated TI libraries (for example, TI-RTOS, FreeRTOS). | |
| Limited RTOS integration. | Native support for TI-RTOS and real-time debugging tools. | |

4. Ecosystem and Integration: CCS is free to use and supports a variety of tools to help users design projects. **Table 2-5. Ecosystem & Integration Differences**

| IAR EW | CCS |
|--|---|
| Paid license with limited free features. | Free tier with optional paid upgrades. |
| Minimal vendor-specific tools. | Tight integration with TI tools (for example, UniFlash, SysConfig). |
| Community support via IAR forums. | Strong TI community (E2E forums, detailed app notes). |

5. Build and Optimization: CCS supports a variety of optimization level to meet different requirements.

| Table 2-6. Build and Optimization Differences | | | |
|---|---|--|--|
| IAR EW | CCS | | |
| Known for highly optimized code. | Balances optimization with TI-specific tuning. | | |
| Custom build steps via GUI. | Flexible build customization using Eclipse or Makefile. | | |
| Static memory allocation via .icf. | Dynamic linker configuration (.cmd files). | | |



3 Porting Code to CCS

3.1 Prepare for Porting

First, understand the IAR project structure. Document the project hierarchy, source files, and dependencies. Note the target microcontroller (MCU) and the variant (for example, TI MSP430, Arm[®] Cortex[®]-M). Identify compiler and linker flags, memory configurations (.icf/.xcl files), and preprocessor symbols.

Second, backup the project. Create a copy of the original IAR project for reference and rollback.

Third, review code for toolchain-specific features. Issues like, check for IAR-specific pragmas, intrinsics (for example, _____no__init), or inline assembly. Identify dependencies on IAR libraries or runtime files (for example, low_level__init.c). Note down important settings such as: compiler flags, linker configuration, memory layout (for example, linker script or ICF file), preprocessor definitions and include paths.

Finally, identify project dependencies. List all external libraries, drivers, and middleware used in the IAR project.

3.2 Set Up CCS Environment

- 1. Install CCS:
 - a. Download and install the latest version of Code Composer Studio from TI's website. Make sure that the required device support packages (Cores, Compilers, and Debuggers) are installed in CCS.
 - b. Install device-specific SDKs (for example, MSP432, C2000, or SimpleLink SDKs) by TI Resource Explorer or TI website.
 - c. Install required libraries. If the project uses TI-specific libraries (for example, DriverLib, TivaWare, or MSPWare), then download and install them.
- 2. Create a New CCS Project:
 - a. File \rightarrow New \rightarrow CCS Project.
 - b. Select the target MCU, compiler (TI Arm Clang), and project template (for example, Empty Project).
 - c. Make sure the output format (for example, ELF) matches the target.

| onnection: | Texas In | struments XDS110 | USB Debug | Probe [Default] Debu | aaer | Verify |
|-----------------------------|--------------|---------------------|--------------|--|--------------------------------|----------------------------|
| Cortex N | 1 [Arm] | | | | 55 | |
| Project na | me: | new_project | Proje | ct name | | |
| 🔽 Use de | fault loca | ition | | | | |
| Lo | ocation: | C:\Users'\workspa | ace_v12_8\ne | ew_project | | Browse |
| Compiler | version: | TI Clang v3.2.2.LTS | 5 | Compiler | ~ | More |
| type filter | text | to Ducket T | | Creates an empty proje selected device. The pro | ct initialize oject will co | d for the ontain an |
| type filter | text | | | Creates an empty proje | ct initialize | d for the |
| E E | mpty Project | oject | mplate | empty 'main.c' source-f | ile. | |
| Empty Project (with main.c) | | | | Note: For this device, consider starting with | | |
| • 🖃 SDK |) Den Res | ource Explorer | | one of the many examp Software Development | ole project: Kit (SDK). : | s from the SDK lets you |
| | | | | do more with your proi | iects, and i | ts examples |
| | | | | | | |
| | | | | | | |

Figure 3-1. Create a New CCS Project



3.3 Import Source Code and Files in CCS

- 1. Copy source files:
 - a. Copy the source files (`.c`, `.h`, `.asm`) from the IAR project to the CCS project directory.
 - b. Use Project Explorer \rightarrow Right-click *Import* \rightarrow *File System* to add files.
- 2. Include paths and preprocessor symbols:
 - a. Add the necessary include paths in the project properties (Right-click *Project* > Properties > Build > Include Options). Shown in figure below.
 - b. Under Predefined Symbols, define macros if necessary.

| 😚 Properties for new_project | | | — 🗆 | × |
|---|---|--------------------------|--------------------------|-------------------------------|
| type filter text | Include Options | | ¢ • | • 🗢 • 8 |
| > Resource General > Build > Arm Compiler Processor Option Optimization Include Options > Predetined Symbol > Advanced Option > Arm Linker Arm Hex Utility (Disa Arm Objcopy Utility) > Debug | Configuration: Debug [Active] Add dir to #include search path (-I) SPROJECT_ROOT)= \$(CG_TOOL_INCLUDE_PATH) = | ✓ Ma | anage Configu Add new | urations path ১ জ ডা ও। |
| ③ Show advanced setting | as | Apply and Close | Can | cel |

Figure 3-2. CCS Project Properties

- 3. Linker configuration:
 - a. Replace IAR .icf/.xcl with a TI linker command file (.cmd).
 - b. Configure memory regions (for example, FLASH, RAM) in the .cmd file. Users not familiar with .cmd file need to refer to the TI Linker Command File Primer for basic explanation of the code, which typically appears in most TI linker command files.



| A typical IAR .icf file | A typical CCS .cmd file |
|--|--|
| *************************************** | 34 - uinterruntVectors |
| /*###ICF### Section handled by ICF editor, don't touch! ****/ | 35-stack size-256 |
| /*-Editor annotation file-*/ | 32 |
| /* IcfEditorFile="\$TOOLKIT DIP\$\config\ide\IcfEditor\contex v1 4 vml" */ | 70 |
| /*-Shariale-*/ | 37 |
| define sumpal ICEEDIT intuce stant - 0x0000000 | SSMEMOKY |
| define symbolicrebil_intvec_start_ = 0x0000000; | 39 { |
| /* Manager Daniana */ | 40 FLASH (RX) : origin = 0x00000000, length = 0x00040000 |
| /*-remorp keglons-*/ | 41 SRAM (RWX) : origin = 0x20200000, length = 0x00008000 |
| define symbolICFEDI1_region_IROM1_start = 0x00000000; | 42 BCR_CONFIG (R) : origin = 0x41C00000, length = 0x000000FF |
| define symbolICFEDIT_region_IROM1_end_ = 0x0001FFFF; | 43 BSL_CONFIG (R) : origin = 0x41C00100, length = 0x00000080 |
| <pre>define symbolICFEDIT_region_IROM2_start_ = 0x0;</pre> | 44 } |
| <pre>define symbolICFEDIT_region_IROM2_end = 0x0;</pre> | 45 |
| <pre>define symbolICFEDIT_region_EROM1_start = 0x0;</pre> | 46 SECTIONS |
| define symbolICFEDIT_region_EROM1_end = 0x0; | 475 |
| <pre>define symbolICFEDIT_region_EROM2_start = 0x0;</pre> | 48 intyers: > 0x0000000 |
| <pre>define symbolICFEDIT_region_EROM2_end = 0x0;</pre> | 49 text : palign(8) (1 > ELACH |
| <pre>define symbolICFEDIT_region_EROM3_start = 0x0;</pre> | E cont : palign(0) () > ELACH |
| define symbolICFEDIT_region_EROM3_end_ = 0x0; | f_{1} dist palign(0) () that |
| define symbol ICFEDIT region IRAM1 start = 0x20200000; | Si .cinit : palign(8) {} FLASH |
| define symbol ICFEDIT region IRAM1 end = 0x20207FFF; | 52 .pinit : paign(a) {} > FLASH |
| define symbol ICFEDIT region IRAM2 start = 0x0: | 53 .rodata : palign(8) {} > FLASH |
| define symbol ICFEDIT region IRAM2 end = 0x0: | 54 .ARM.exidx : palign(8) {} > FLASH |
| define symbol ICFEDIT region ERAM1 start = 0x0: | 55 .init_array : palign(8) {} > FLASH |
| define symbol ICEEDIT region FRAM1 and - 0x0. | 56 .binit : palign(8) {} > FLASH |
| define symbol ICEEDIT region FRAM2 start - 0x0; | 57 .TI.ramfunc : load = FLASH, palign(8), run=SRAM, table(BINIT) |
| define symbol ICEEDIT region EDAN2 and _ DAG | 58 |
| define symbol ICEEDIT region EDMA2 start - 0x0; | 59 .vtable : > SRAM |
| define symbol | 60 .args : > SRAM |
| define symbolICFEDIT_TERION_ENANS_ENU_ = 0x0, | 61 .data : > SRAM |
| define symbolrepoin_region_NON_MAIN_DCR_start = 0x4100000; | 62 .bss :> SRAM |
| define symbolicrebit_region_NON_MAIN_BCR_end = 0x41c000/F; | 63 Sysmem : > SRAM |
| define symbolICFEDIT_region_NON_MAIN_BSL_start _ = 0X4IC00100; | 64 stack · SRAM (HTGH) |
| detine symbolicrEDII_region_NON_MAIN_BSL_end = 0x41C0017F; | 65 |
| /*-51Zes-*/ | 66 BCBConfig + () > BCB CONFIG |
| <pre>define symbolICFEDIT_size_proc_stack_ = 0x0000;</pre> | 67 BCI Config () > BCI CONFIG |
| | b/ .BSLCONTIG : () > BSL_CONFIG |
| | 66 J |

Figure 3-3. Linker File Comparison

- 4. Translate compiler and linker flags:
 - a. Set stack or heap size in the linker file or by Project Properties \rightarrow Build \rightarrow ARM Linker \rightarrow Basic Options.
 - b. Map IAR flags to TI Arm Clang equivalents:

Table 3-1. A Comparsion Table of Common Used IAR Flag vs CCS Equivalent

| IAR Flag | CCS Equivalent (TI Arm Clang) | Purpose | |
|-------------------|--------------------------------|---------------------------|--|
| debug | -g | Debug symbols | |
| -Oh | -03 | High optimization | |
| -DNAME | define=NAME | Define preprocessor macro | |
| cpu=cortex-m4 | -mcpu=cortex-m4 Target CPU | | |
| -I <path></path> | -I <path></path> | Include directory | |
| data_model medium | Not needed (configure in .cmd) | Memory model | |

3.4 Handle Device-Specific Code

- 1. Replace IAR Startup Code:
 - a. Use TI-provided startup files (for example, startup_<device>.c from the SDK) instead of IAR's startup_<device>.s.
 - b. Update interrupt vector tables to match TI's syntax (for example, #pragma DATA_SECTION for vectors).
- 2. Adapt Hardware Abstraction:
 - a. Replace IAR-specific HAL functions with TI DriverLib or register-based code.
 - b. Example: use MAP_GPIO_setAsOutputPin() instead of the GPIO library of the IAR.
- 3. Update Inline Assembly and Pragmas:
 - a. Rewrite IAR-specific pragmas (for example, __packed becomes __attribute__((packed))).
 - b. Convert inline assembly to TI Clang syntax.



Table 3-2. Comparison of Assembly Code Example in IAR and CCS

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

3.5 Adapt Code for CCS

- 1. Fix compiler-specific code:
 - a. IAR and CCS compilers can have different syntax or behavior for certain constructs (for example, inline assembly, pragmas).
 - b. Update any compiler-specific code to be compatible with CCS.
- 2. Replace IAR-specific functions:
 - a. Replace IAR-specific functions (for example, `__enable_interrupt()`) with equivalent CCS or TI-specific functions.
- 3. Update debugging code:
 - a. If the project uses IAR-specific debugging macros or functions, then replace those macros or functions with CCS-compatible alternatives.

3.6 Build and Debug

- 1. Build and validate the project:
 - a. Resolve build errors if there is any. Check the error report in the Problems console first. Then locate the error and fix. Common issues are missing includes, undefined macros, and syntax mismatches.
 - b. Adjust compiler optimization levels and other settings for performance or size. Please refer to TI ARM Clang Compiler User Manual for a more detailed description of the optimization options.
- 2. Create a debug configuration:
 - a. Run \rightarrow Debug Configurations \rightarrow New Launch Configuration.
 - b. Select the target connection (XDS110, JTAG, SWD).
 - c. Use a .ccxml file to define the debug probe and MCU.
- 3. Run on hardware and debug:
 - a. Connect the target microcontroller to CCS and load the program.
 - b. Load the program and verify breakpoints, register views, and memory inspection. Verify that the program runs correctly and performs as expected.
 - c. Compare behavior with the original IAR project.

7



4 Post-Migration Optimization

- 1. Document the migration:
 - a. Document all changes made during the migration process for future reference.
 - b. Update the project documentation to reflect the new CCS environment.
- 2. Common concerns:
 - a. Interrupt handlers: Make sure ISRs use #pragma WEAK or are correctly named (for example, void TIMER0_A0_ISR(void)).
 - b. Memory alignment: TI Clang can enforce stricter alignment than IAR. Use __attribute __((aligned(8))) if needed.
 - c. Linker errors: Verify .cmd file addresses match the MCU memory map.

5 Summary

By following this application note, users can systematically transition projects to CCS while addressing toolchainspecific nuances. Test incrementally and leverage TI's robust debugging tools to streamline validation.

6 References

- Texas Instruments, ARM-CGT, webpage
- IAR Systems AB, IAR Embedded Workbench, webpage
- Texas Instruments, CCSTUDIO, webpage

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2025, Texas Instruments Incorporated