Application Note

C2000 SysConfig Linker Command Tool

Nima Eskandari and Veena Kamath

ABSTRACT

Linker command files play an important role in embedded programs as they specify where code and data sections get allocated into target memory. Without this file, the linker does not know the target memory configuration and how to properly allocate the sections. For C2000™ real-time controllers, you have to understand the device memory by reviewing the device-specific data sheet and technical reference manual. Example C2000 linker command files are available in C2000Ware SDK but for any given application, you might have to modify the template linker command files available in the C2000Ware SDK to fit your application needs. This requires that you to learn the syntax and options available when writing a linker command file. The C2000 Linker CMD Tool simplifies the task of creating application-specific linker command files by providing an intuitive GUI and automatic code generation.

Table of Contents

List of Figures

Trademarks

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.

1 Introduction

The linker command file is used during the link stage of the application build where the linker combines object files and allocates sections into the target system's configured memory. The linker command file is an ASCII file that uses two linker directives, MEMORY and SECTIONS, to allocate sections into specific areas of memory. The MEMORY directive defines target memory configuration. The SECTIONS directive controls how sections are built and allocated. In addition, the linker command file can also include input filenames and linker options.

The Linker Command File Primer page covers the basics of linker command files, focusing on the MEMORY and SECTIONS directives.

Creating a new linker command file from scratch or even editing an existing linker command file template can be difficult for new users. Users must understand the structure of linker command files along with their specific device's memory structure.

The C2000 Linker CMD Tool significantly simplifies the task of creating new or editing existing linker command files by providing the following features:

- Intuitive GUI-based that showcases all available customization options
- Error checking to help you to avoid making mistakes
- Auto-generate CMD files
- Automatic Code Composer Studio™ project property modification
- Auto-generate additional C source and header files for initializing memory sections
- Showcasing the available memories for the selected device family

Utilizing the C2000 Linker CMD tool can speed up the software development for new and advanced users.

2 C2000 Linker Command Tool – GUI Configurations

The C2000 Linker Command Tool is a SysConfig-based product that is seamlessly integrated in C2000 System Configuration Tool.

For more information on the C2000 System Configuration Tool visit:

Video Series:

- 7.1 C2000™ SysConfig: Overview
- 7.2 C2000™ SysConfig: Getting Started
- 7.3 C2000™ SysConfig: PinMux
- 7.4 C2000™ SysConfig: Board Support
- 7.5 C2000™ SysConfig: Example Walkthrough
- 7.6 C2000™ SysConfig: Migrate C2000 Devices in under 10 minutes

Benefits of C2000 SysConfig:

- *Speed Up Development With C2000™ Real-Time MCUs Using SysConfig*

Application report - step by step guide for using C2000 SysConfig:

- *C2000 SysConfig*

SW getting started Guide:

- https://software-dl.ti.com/C2000/docs/software_guide/c2000_sysconfig.html

In order for developers to use the C2000 Linker CMD Tool, they must launch the C2000 SysConfig tool for their given device and package. *C2000 SysConfig* walks you through the steps needed to launch the C2000 SysConfig tool both in the context of a Code Composer Studio project as well as the SysConfig standalone tool.

The C2000 Linker Command Tool is shown in Figure 2-1.

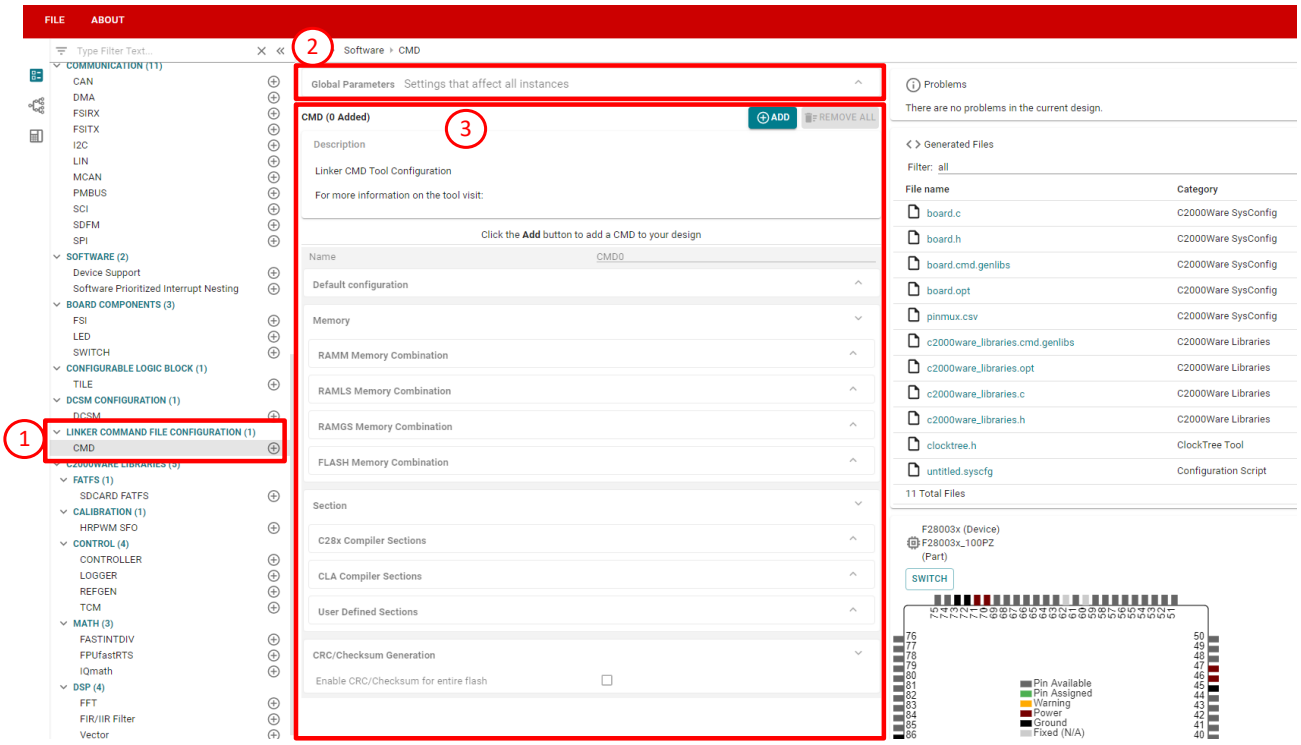

Figure 2-1. Linker CMD Tool - GUI Overview

1. The linker CMD module
2. The global settings effecting all CMD module instances added to the design
3. The instances of the CMD module in the design

Once a CMD module is added, additional files are generated by the tool.

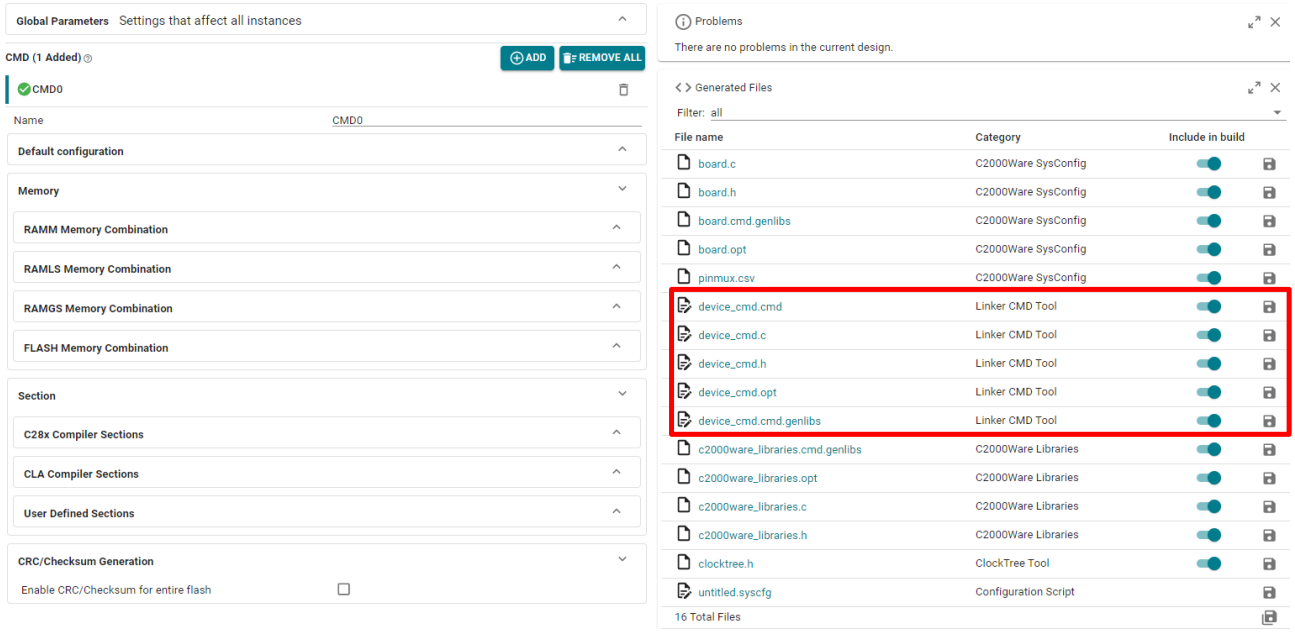

Figure 2-2. Linker CMD Tool - Generated Files

Note that you can add more than one CMD module. All the different instances of the CMD module can be saved within the **syscfg** file. You can decide which one of the CMD modules is active by selecting it in the **Global Parameters**.

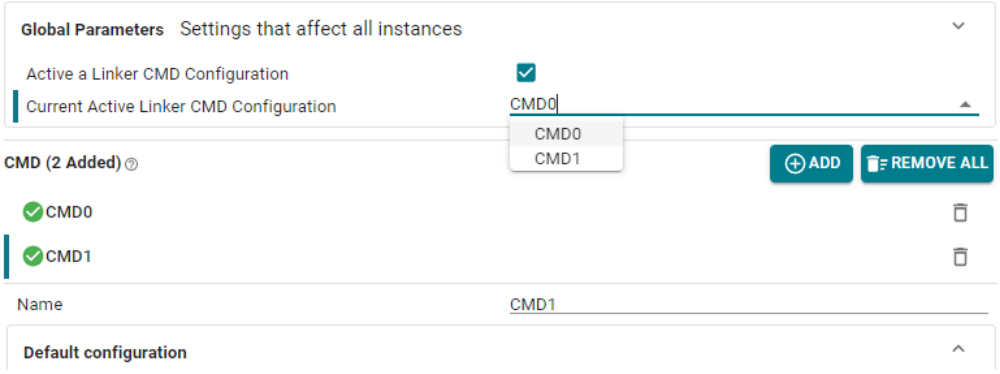

Figure 2-3. Linker CMD Tool - Global Parameters

Each instance of the CMD module has the following entries:

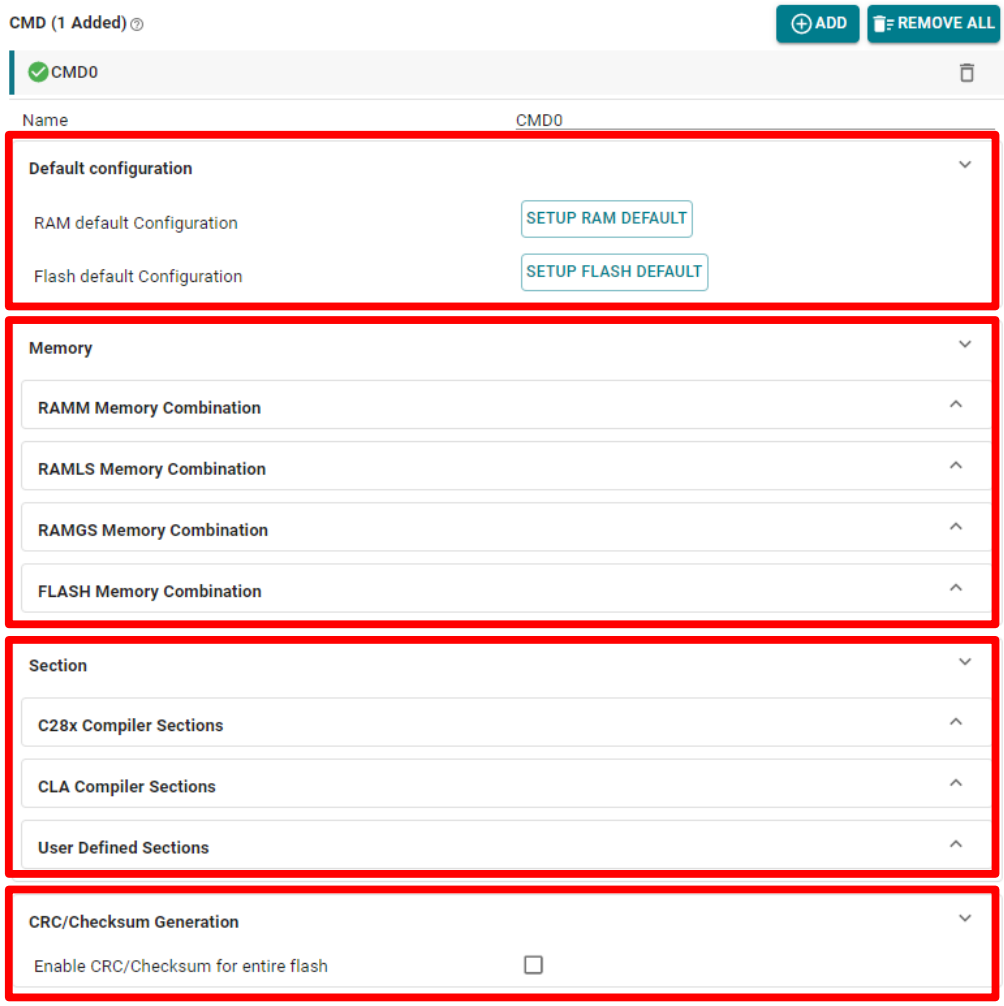

Figure 2-4. Linker CMD Tool - CMD Instance Configurations

- Default Configuration: Configure the instance to the default settings provided for this device
- Memory: Grouped by the memory type, combine the memory blocks to create larger memory groups
- Section: Assign device memory to their purpose
- CRC/Checksum Generation: Generate CRC/checksum for the entire flash

2.1 Memory Combination

You can combine different memory blocks of the same type and name the new memory block, as shown in Figure 2-5.

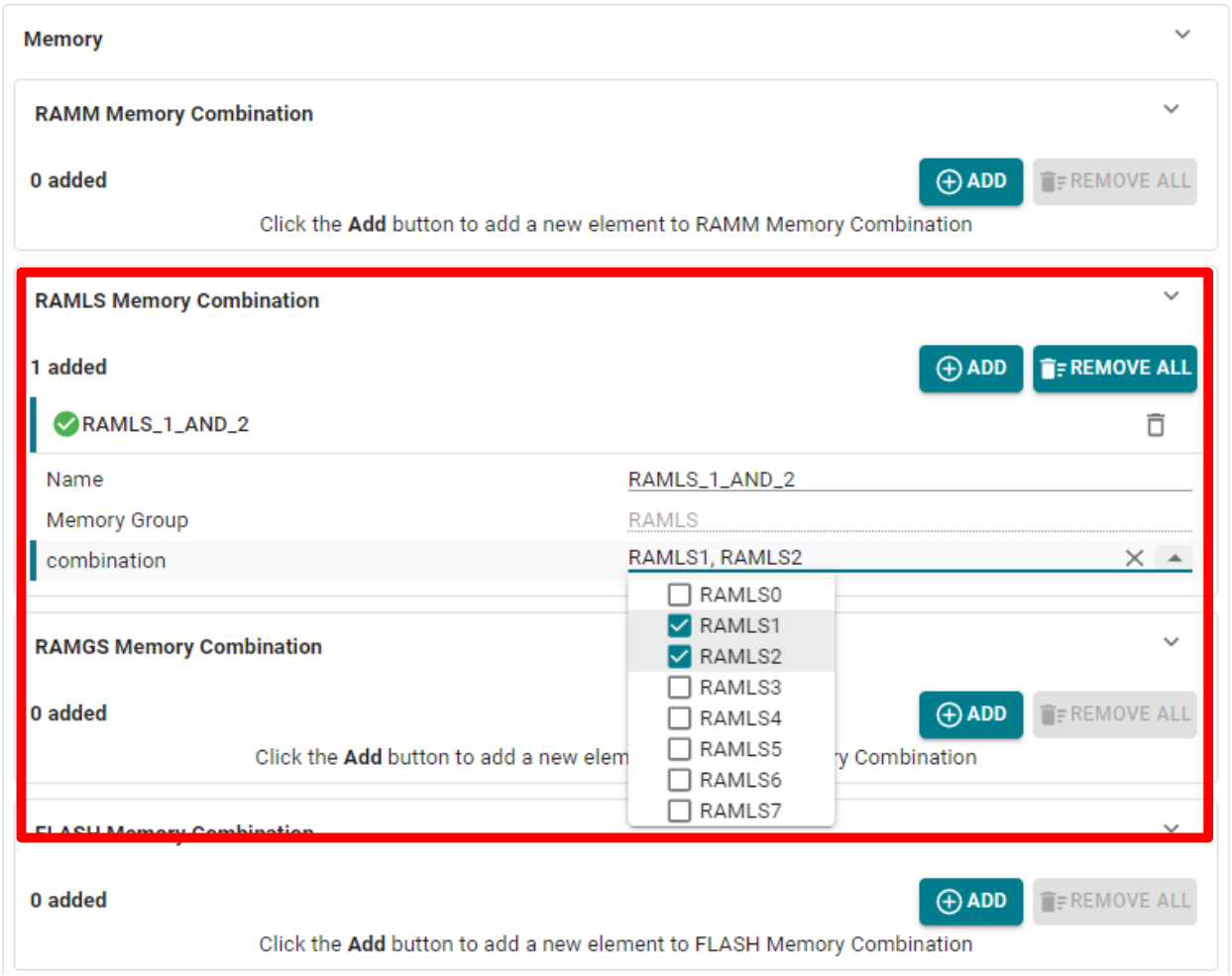

Figure 2-5. Linker CMD Tool - Memory Combination

The tool does not allow the combination of device memories that are not continuous.

2.2 Memory Sections

The sections are divided into groups depending on whether they belong to C28x, CLA or are a custom “User Defined” sections.

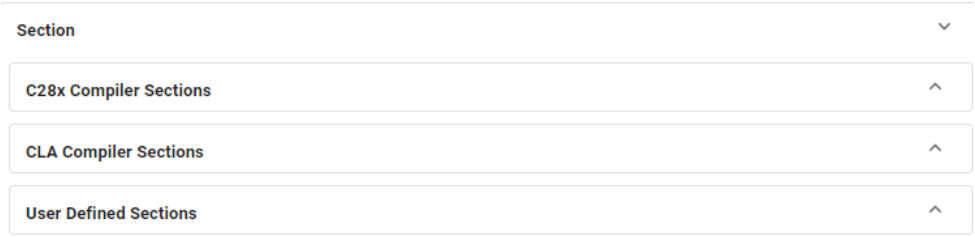

Figure 2-6. Linker CMD Tool - Memory Sections

The sections are each assigned a memory block. The options available are both the device memory blocks and the combined memory blocks.

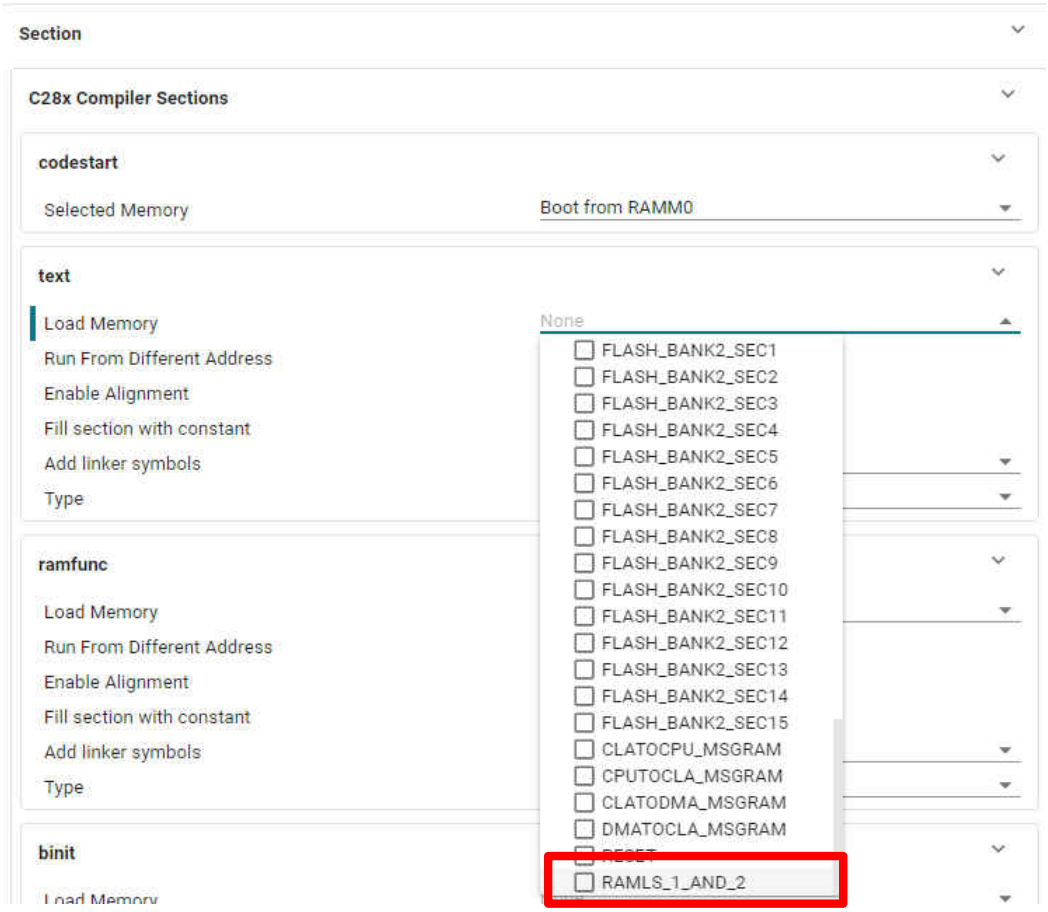

Figure 2-7. Linker CMD Tool - Load Memory

If a device has an additional CLA core, the CLA sections are present for you to configure.

User defined sections can be added and named as needed to meet the application needs.

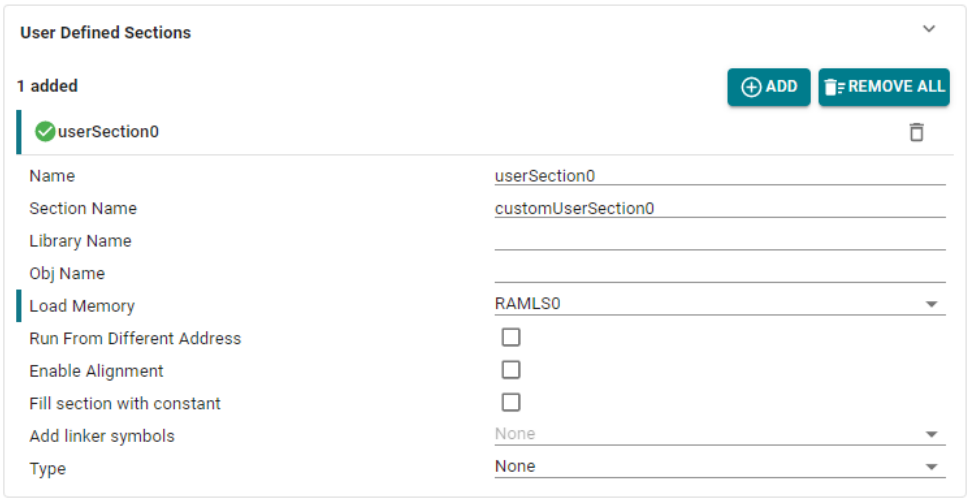

Figure 2-8. Linker CMD Tool - User Defined Section

2.3 CLA Sections

The Linker CMD tool also has support for the CLA sections on a device with CLA support.

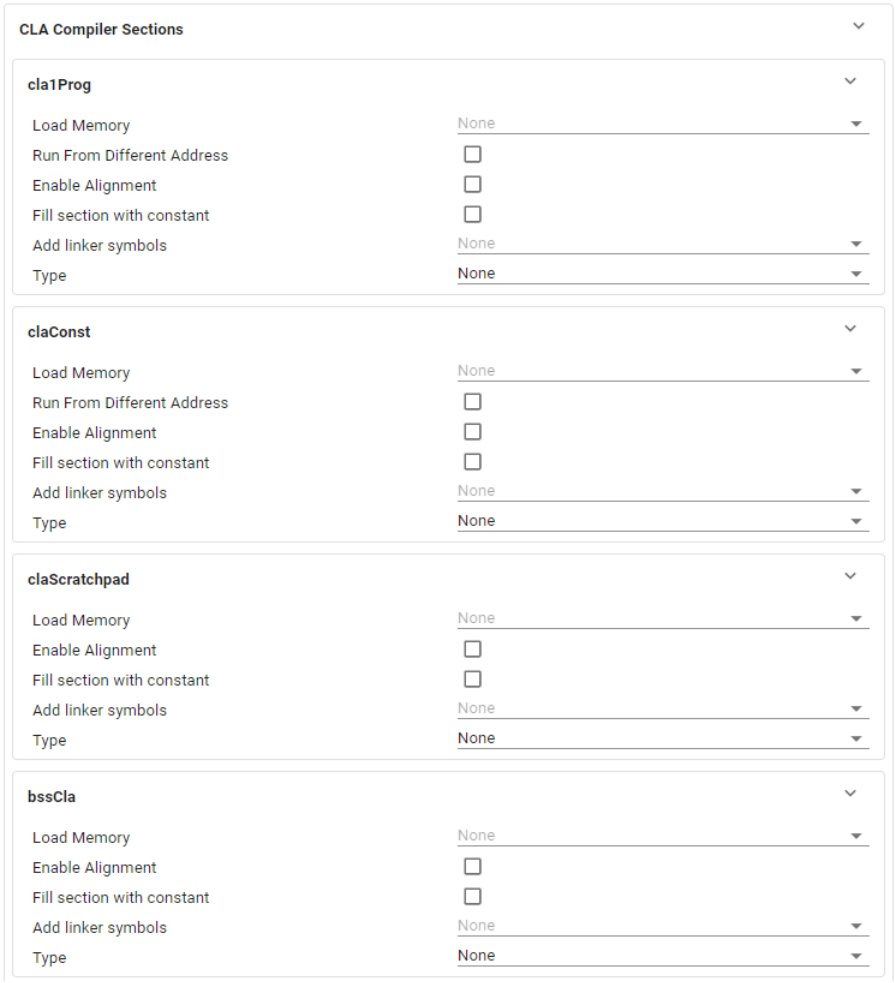

Figure 2-9. Linker CMD Tool - CLA Sections

3 C2000 Linker Command Tool – Code Generation

The linker CMD tool generates a series of files:

- **device_cmd.cmd:** This is the main file that contains the linker command file entries based on the selected options in the GUI.
- **device_cmd.c:** This file contains the initialization code for specific memory sections that are required to run from a different address than the one they were loaded from.
- **device_cmd.h:** The header source file that goes hand-in-hand with the device_cmd.c file.
- **device_cmd.opt:** In the context of a Code Composer Studio project, the entries in the OPT file are the compiler options needed by the CMD tool. These options are automatically appended to the project.
- **device_cmd.cmd.genlibs:** In the context of a Code Composer Studio project, the entries in the GENLIBs file are the linker options needed by the CMD tool. These options are automatically appended to the project, given that the linker references this file in the project properties.

3.1 device_cmd.cmd File

The device_cmd.cmd file contain the linker cmd entries.

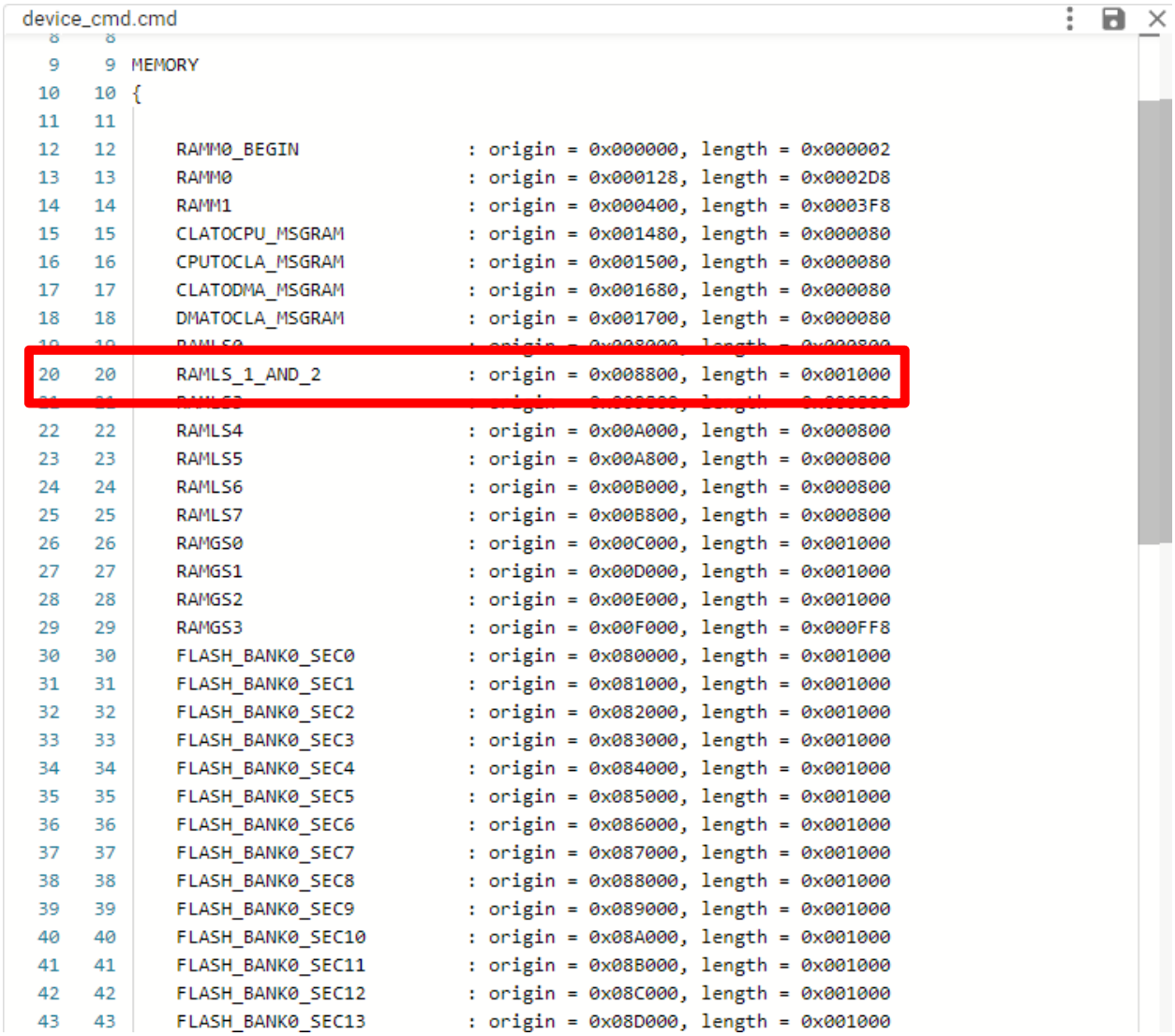

Figure 3-1. Generated Files - CMD File

The entries in the Memory part of the .cmd file includes the memory combination blocks. The size and origin of the memory combinations are present in the file.

The entries in the Sections part of the .cmd file only show up when that specific section has a valid “Load Memory” entry selected.

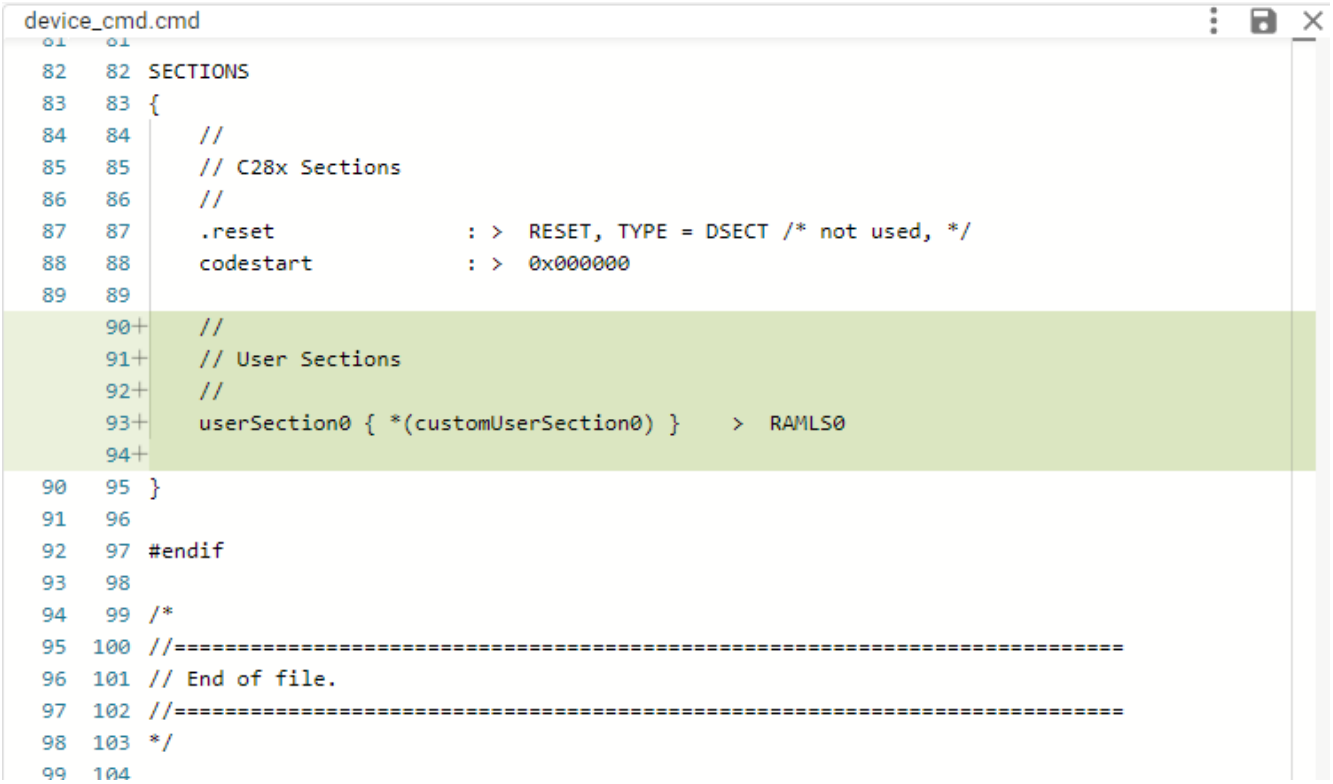

Figure 3-2. Generated Files - CMD File Sections

3.2 Supporting Files

The additional **device_cmd.c**, **device_cmd.h**, **device_cmd.opt** and **device_cmd.cmd.genlibs** are supporting files generated by the tool.

The code generation includes a LIVE DIFF tool which showcases how the changes in the GUI cause changed in the generated code.

If you decide that a section must be loaded and run from a different address, then the generated format of the entries in the linker cmd file automatically change.

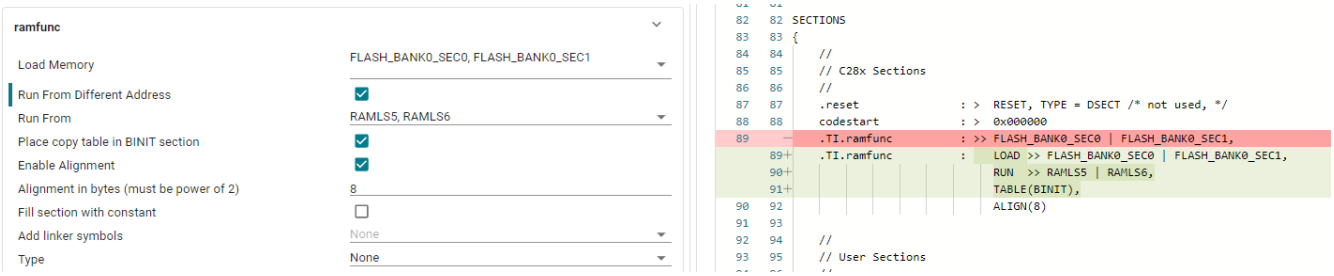

Figure 3-3. Generated Files - CMD File Diff

Once the **Place copy table in BINIT section** is deactivated, the .c and .h files are also updated and the initialization code needed is automatically generated.

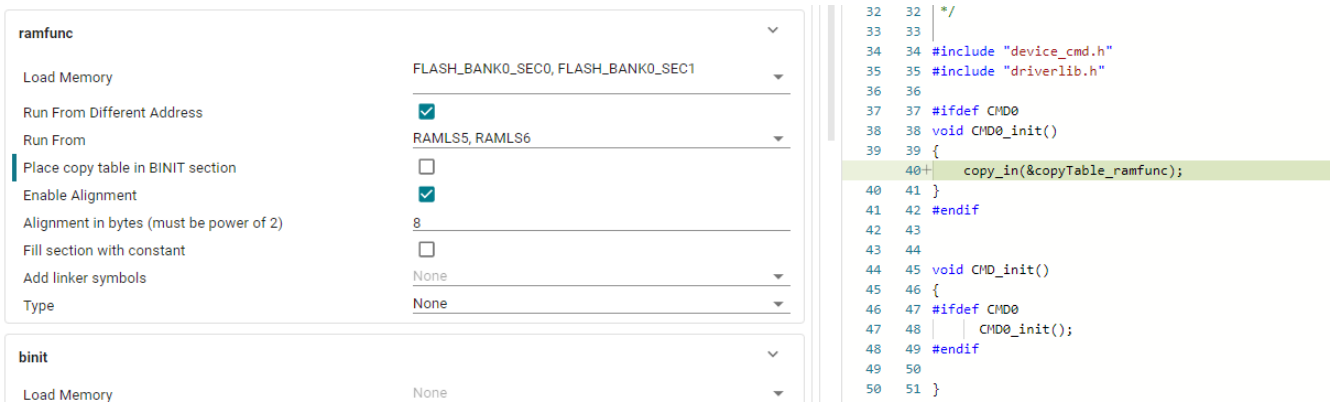

Figure 3-4. Generated Files - Copy Table

You need to call the **CMD_init** function in your application code to initialize such sections. The **device.c** for C2000 devices includes a **Device_init** function that can call this function, if needed.

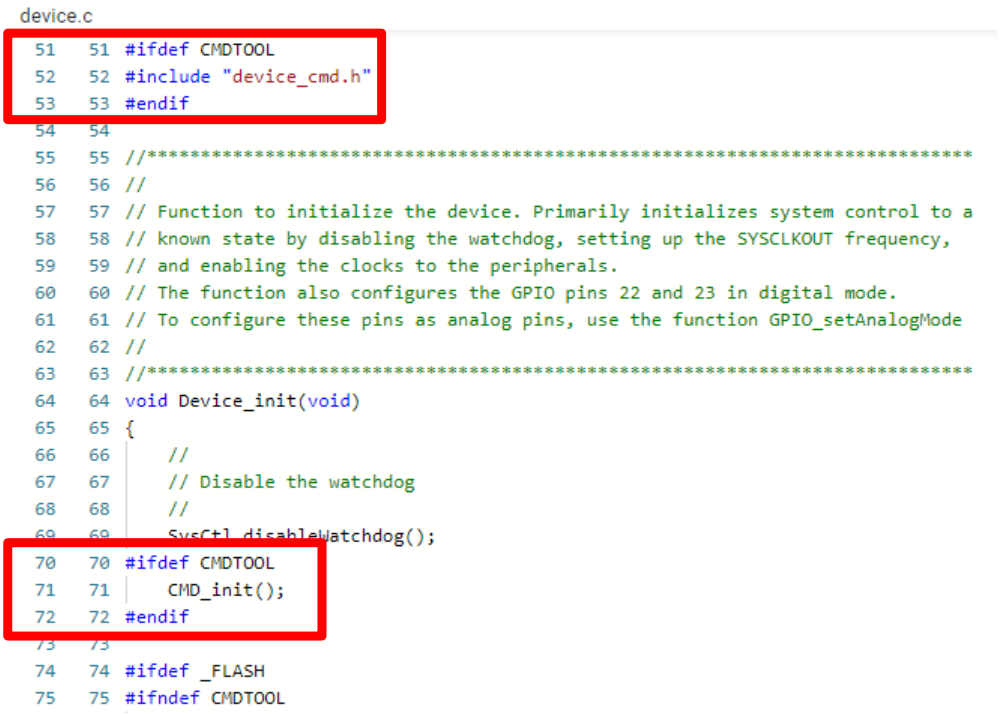

Figure 3-5. Generated Files - C File

device_cmd.opt and **device_cmd.cmd.genlibs** automatically setup the Code Composer Studio project properties.

The OPT file creates a predefined symbol for **CMDTOOL** and the active CMD module instance name.

device_cmd.opt

```
/*
 *  ======== device_cmd.opt ========
 *  Project options needed for this application's configuration
 *
 *  NOTE, this feature requires software components configured in your
 *  system to correctly indicate their project properties
 *  needed for your specific configuration.  If you find
 *  errors, please report them on TI's E2E forums
 *  (https://e2e.ti.com/) so they can be addressed in a future
 *  release.
 *
 *  This file allows one to portably link applications that use SysConfig
 *  _without_ having to make changes to build rules when moving to a new
 *  device OR when upgrading to a new version of a SysConfig enabled
 *  product.
 *
 *  DO NOT EDIT - This file is generated by the SysConfig tool for the
 *                TI C/C++ toolchain
 */
-DCMDTOOL
-DCMD0
```

Figure 3-6. Generated Files - OPT File

These predefined symbols are used in application C code: **device.c** and **device.h** files.

The **device_cmd.cmd.genlibs** file follows a similar path.

device_cmd.cmd.genlibs

```
/*
 *  ======== device_cmd.cmd.genlibs ========
 *  Project options needed for this application's configuration
 *
 *  NOTE, this feature requires software components configured in your
 *  system to correctly indicate their project properties
 *  needed for your specific configuration.  If you find
 *  errors, please report them on TI's E2E forums
 *  (https://e2e.ti.com/) so they can be addressed in a future
 *  release.
 *
 *  This file allows one to portably link applications that use SysConfig
 *  _without_ having to make changes to build rules when moving to a new
 *  device OR when upgrading to a new version of a SysConfig enabled
 *  product.
 *
 *  DO NOT EDIT - This file is generated by the SysConfig tool for the
 *                TI C/C++ toolchain
 */
--define=CMDTOOL
--define=CMD0

```

Figure 3-7. Generated Files - Genlibs File

4 Migration Across Device Families

Those who take advantage of the C2000 SysConfig tool can utilize the **SWITCH** button to migrate their design from one device family to another.

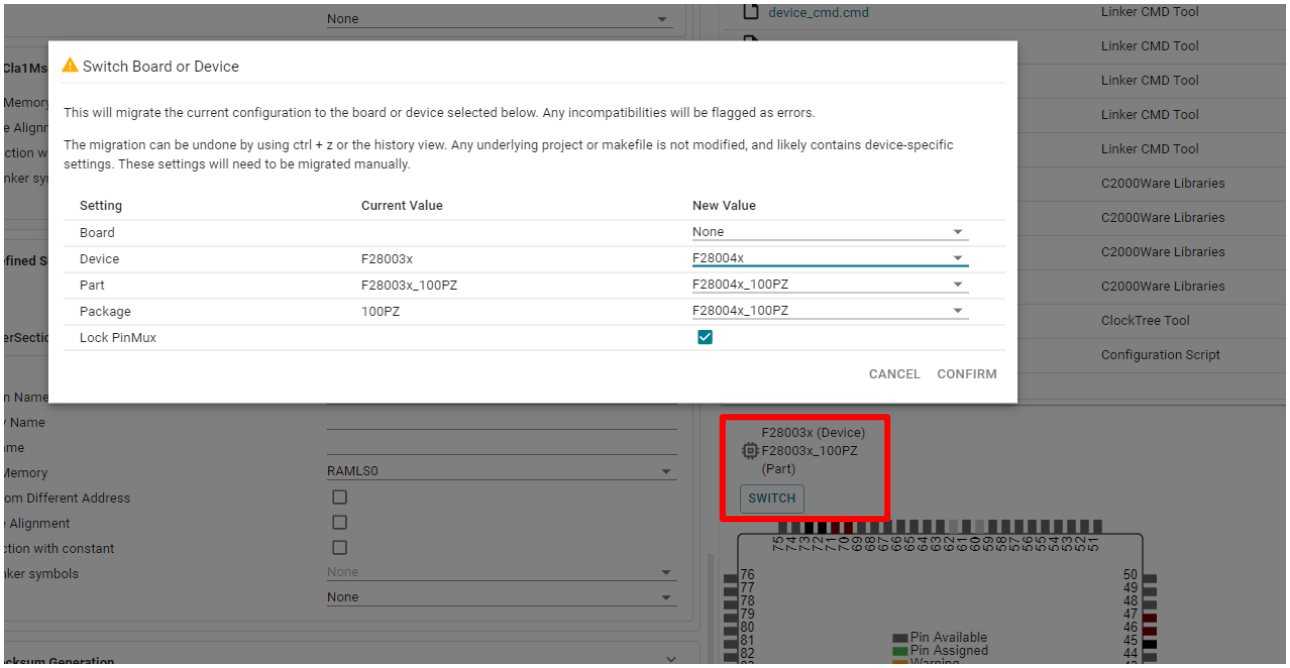

Figure 4-1. Device Migration - SWITCH

Once the migration is completed, all of the modified files generated by the SysConfig tool are identified.

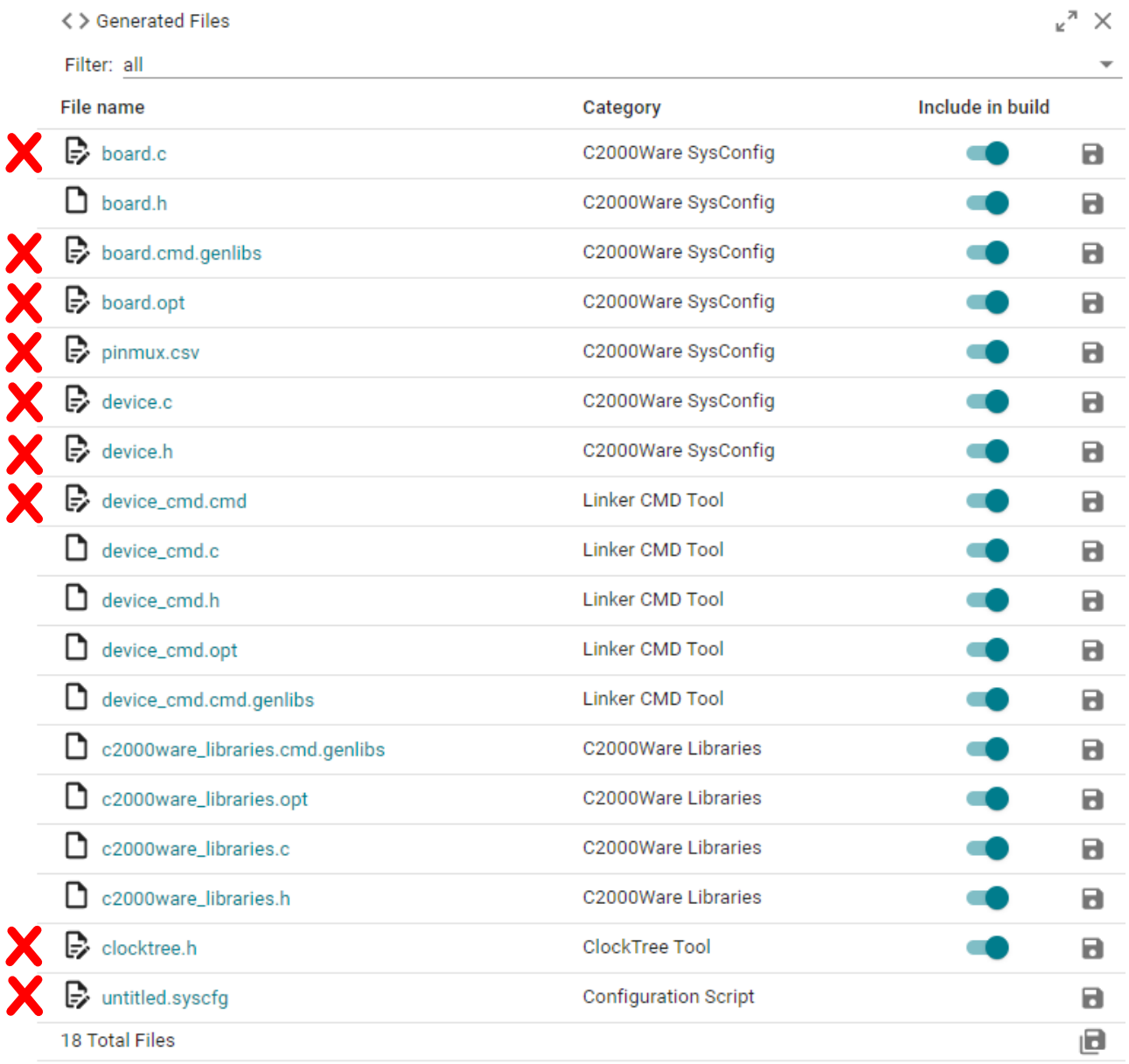

Figure 4-2. Device Migration - Files Changed

Each file also identifies the changes in the generated code.

device_cmd.cmd

```
 6   6 #define CMD0
 7   7 #ifdef CMD0
 8   8
 9   9 MEMORY
10  10 {
11  11
12  12     RAMM0_BEGIN             : origin = 0x000000, length = 0x000002
13      -  RAMM0                   : origin = 0x000128, length = 0x0002D8
    13 +   RAMM0                   : origin = 0x0000F6, length = 0x00030A
14  14     RAMM1                   : origin = 0x000400, length = 0x0003F8
15  15     CLATOCPU_MSGRAM         : origin = 0x001480, length = 0x000080
16  16     CPUTOCLA_MSGRAM         : origin = 0x001500, length = 0x000080
17      -  CLATODMA_MSGRAM         : origin = 0x001680, length = 0x000080
18      -  DMATOCLA_MSGRAM         : origin = 0x001700, length = 0x000080
19  17     RAMLS0                  : origin = 0x008000, length = 0x000800
20  18     RAMLS_1_AND_2           : origin = 0x008800, length = 0x001000
21  19     RAMLS3                  : origin = 0x009800, length = 0x000800
22  20     RAMLS4                  : origin = 0x00A000, length = 0x000800
23  21     RAMLS5                  : origin = 0x00A800, length = 0x000800
24  22     RAMLS6                  : origin = 0x00B000, length = 0x000800
25  23     RAMLS7                  : origin = 0x00B800, length = 0x000800
26      -  RAMGS0                  : origin = 0x00C000, length = 0x001000
27      -  RAMGS1                  : origin = 0x00D000, length = 0x001000
28      -  RAMGS2                  : origin = 0x00E000, length = 0x001000
29      -  RAMGS3                  : origin = 0x00F000, length = 0x000FF8
    24 +   RAMGS0                  : origin = 0x00C000, length = 0x002000
    25 +   RAMGS1                  : origin = 0x00E000, length = 0x002000
    26 +   RAMGS2                  : origin = 0x010000, length = 0x002000
    27 +   RAMGS3                  : origin = 0x012000, length = 0x001FF8
```

Figure 4-3. Device Migration - File Changes

The file diffs indicate all changes that has occurred as a result of the migration.

5 Summary

The C2000 Linker CMD Tool is an intuitive graphical user interface tool which configures the device memory for a given application. This tool can significantly simply the software development users by providing error checking, automatic project setup, automatic code generation, and device family migration support.

6 References

Video Series:

- 7.1 C2000™ SysConfig: Overview
- 7.2 C2000™ SysConfig: Getting Started
- 7.3 C2000™ SysConfig: PinMux
- 7.4 C2000™ SysConfig: Board Support
- 7.5 C2000™ SysConfig: Example Walkthrough
- 7.6 C2000™ SysConfig: Migrate C2000 Devices in under 10 minutes

Benefits of C2000 SysConfig:

- Texas Instruments: *Speed Up Development With C2000™ Real-Time MCUs Using SysConfig*

Application note - step by step guide for using C2000 SysConfig:

- Texas Instruments: *C2000 SysConfig*

Software getting started guides:

- https://software-dl.ti.com/C2000/docs/software_guide/c2000_sysconfig.html
- https://software-dl.ti.com/ccs/esd/documents/sdto_cgt_Linker-Command-File-Primer.html

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