Application Report
SPMA041G – January 2012 – Revised October 2015

Using the CMSIS DSP Library in Code Composer Studio™ for TM4C MCUs

Amit Ashara

ABSTRACT

This application report describes the process required to build the ARM® CMSIS DSP library in Code Composer Studio v6.1 with ARM Compiler version up to 5.2.5 . This document also describes how to use Code Composer Studio v6.1 to build, run, and verify the 11 ARM DSP example projects that are included in the CMSIS package.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/spma041.

NOTE: This document applies to both the TM4C Series and the Stellaris® Cortex®-M4 MCUs. All screen captures reflect the TM4C version of the device.

Contents

1 Introduction

Many microcontroller-based applications can benefit from the use of an efficient digital signal processing (DSP) library. To that end, ARM has developed a set of functions called the CMSIS DSP library that is compatible with all Cortex M3 and M4 processors and that is specifically designed to use ARM assembly instructions to quickly and easily handle various complex DSP functions. Currently, ARM supplies example projects for use in their Keil uVision IDE that are meant to show how to build their CMSIS DSP libraries and run them on an M3 or M4. This application report details the steps that are necessary to build these DSP libraries inside Code Composer Studio version 6 and run these example applications on a TM4C Series TM4C129 Connected LaunchPad.

2 CMSIS DSP Library

To build the CMSIS DSP library, download and extract the source code from the ARM CMSIS website: http://cmsis.arm.com. The source code for the DSP library and example projects are in this directory:

CMSIS-<*version*>/CMSIS/DSP_Lib

A full description of the DSP libraries, including a description of examples, the data structures used, and an API for each available function, is in the ARM-provided documentation at this location:

CMSIS-<*version*>/CMSIS/Documentation/DSP/html/index.html

If ARM releases a future update to CMSIS, you might need to download and install a patch to the DSP library in order to provide support for new functionality and to fix any bugs that ARM discovers in the CMSIS source code. After you download the patch files from the ARM web site, follow these instructions to install:

1. Unzip the patch file.
2. Navigate to the patch directory and copy any files found in that directory to the corresponding location of the CMSIS DSP library.
3. Overwrite existing files when prompted.

For example, if the patch directory contains a file named arm_common_tables.c in the CMSIS/DSP_Lib/Source/CommonTables directory, copy this file into the same directory (CMSIS/DSP_Lib/Source/CommonTables) of your original CMSIS installation, overwriting the arm_common_tables.c that already exists in the original installation directory.

After the CMSIS source code has been downloaded, you must download and unzip the CCS CMSIS Patch Files. This CCS CMSIS zip package is located on the Texas Instruments' website at http://www.ti.com/lit/zip/spma041. The zip package contains a set of support files that are needed for building and running the CMSIS DSP library in Code Composer Studio. After you download the zip package, run the unzip application and select a location in which to extract the files.

3 Building the DSP Library in Code Composer Studio v6.1

This section details the steps required to build the ARM CMSIS DSP library from source. It is possible to skip this section by using a precompiled .lib (such as one of those found in CMSIS-<*version*>/CMSIS/Lib/ARM or CMSIS-<*version*>/CMSIS/Lib/GCC), but doing so requires changing the Code Composer Studio compiler settings to call floating-point functions in a way that is different from the default Code Composer Studio settings. This requires rebuilding all .lib files that are used in a project with the DSP libraries, most notably the TivaWare™ for C Series Software driverlib, grlib, and usblib libraries. This method is not recommended and the process is not described in this application report. Also this application report has been updated for the support of CMSIS release r4p2 onwards.

3.1 Adding the CCS-Required Header Files to the DSP Libraries

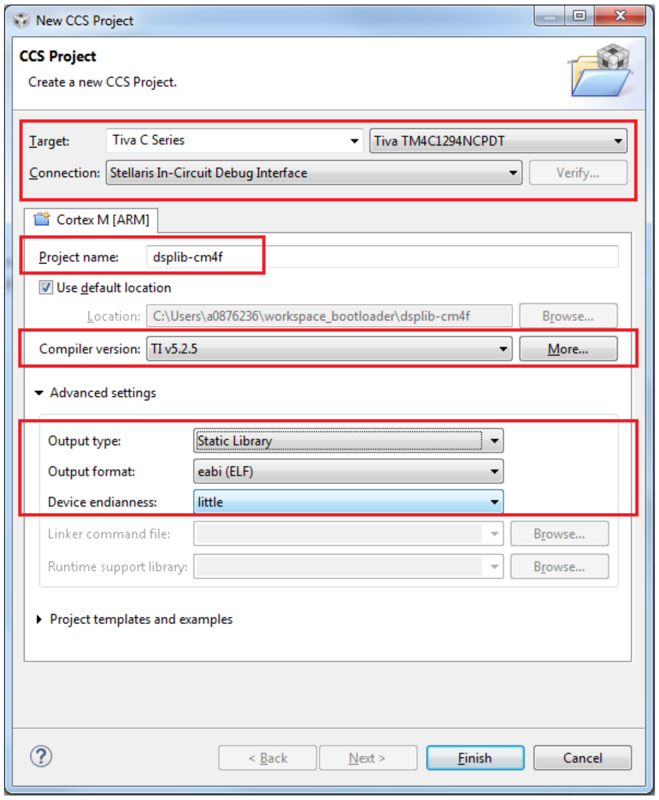
To compile the CMSIS DSP libraries using Code Composer Studio, you must modify the DSP library include files, add a Code Composer Studio specific include file, and add a new assembly file. The zip package contains pre-modified versions of these files, which can be used during the build process or you can elect to modify the files yourself by using the following steps:

1. Copy arm_math.h and cmsis_ccs.h from this application report into the CMSIS/Include directory
2. Copy arm_bitreversal2.asm from this application report into CMSIS/DSP_Lib/Source/TransformFunctions.

3.2 Creating the dsplib Project

Before building the DSP library in Code Composer Studio, you must create a project for the library. You can build a project by completing the following steps:

1. Launch CCSv6.1 and select an empty workspace.
2. Select File → New → CCS Project. The *New Code Composer Studio Project* window will be displayed.
3. Select Target as TM4C Series and then use the drop down menu to select TM4C1294NCPDT. Select the Connection as Stellaris In-Circuit Debug Interface (see Figure 1).
4. In the Project name, type dsplib-cm4f and keep the check box ticked for the Use default location.
5. In Advanced settings, select:
   - Output type: Static Library
   - Output format: eabi (ELF)
   - Device endianness: little
6. In Project templates and examples (see Figure 2), select Empty Project.
7. Click **Finish** to create the project. The dsplib-cm4f project appears in the Project Explorer.



New CCS Project

CCS Project

Create a new CCS Project.

Target: Tiva C Series | Tiva TM4C1294NCPDT

Connection: Stellaris In-Circuit Debug Interface | Verify...

Cortex M [ARM]

Project name: dsplib-cm4f

☑ Use default location

Location: C:\Users\a0876236\workspace_bootloader\dsplib-cm4f | Browse...

Compiler version: TI v5.2.5 | More...

Advanced settings

Output type: Static Library

Output format: eabi (ELF)

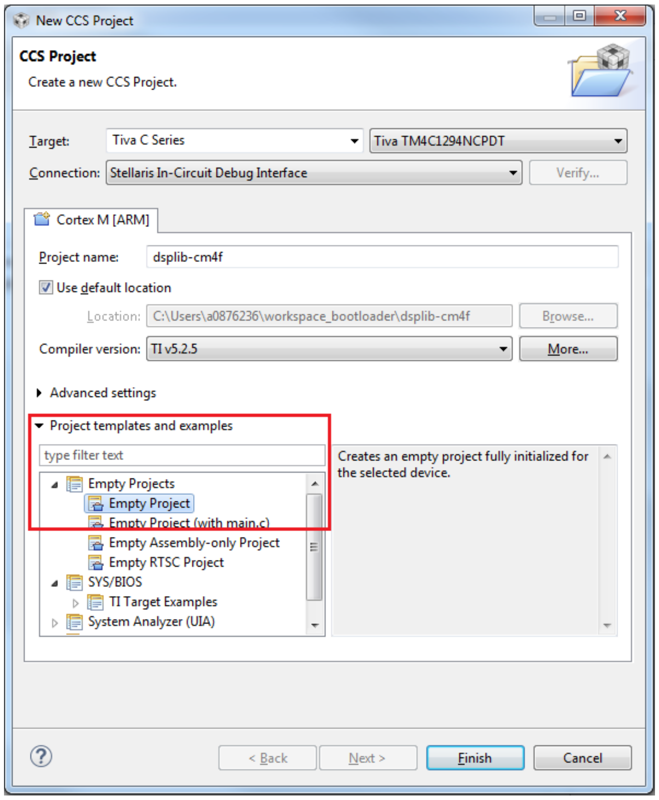
Device endianness: little

Linker command file: | Browse...

Runtime support library: | Browse...

Project templates and examples

Figure 1. Creating the dsplib Project



New CCS Project

CCS Project

Create a new CCS Project.

Target: Tiva C Series | Tiva TM4C1294NCPDT

Connection: Stellaris In-Circuit Debug Interface | Verify...

Cortex M [ARM]

Project name: dsplib-cm4f

☑ Use default location

Location: C:\Users\a0876236\workspace_bootloader\dsplib-cm4f | Browse...

Compiler version: TI v5.2.5 | More...

Advanced settings

Project templates and examples

type filter text

- Empty Projects
  - Empty Project
  - Empty Project (with main.c)
  - Empty Assembly-only Project
  - Empty RTSC Project
- SYS/BIOS
  - TI Target Examples
- System Analyzer (UIA)

Creates an empty project fully initialized for the selected device.

Figure 2. Creating the dsplib Project

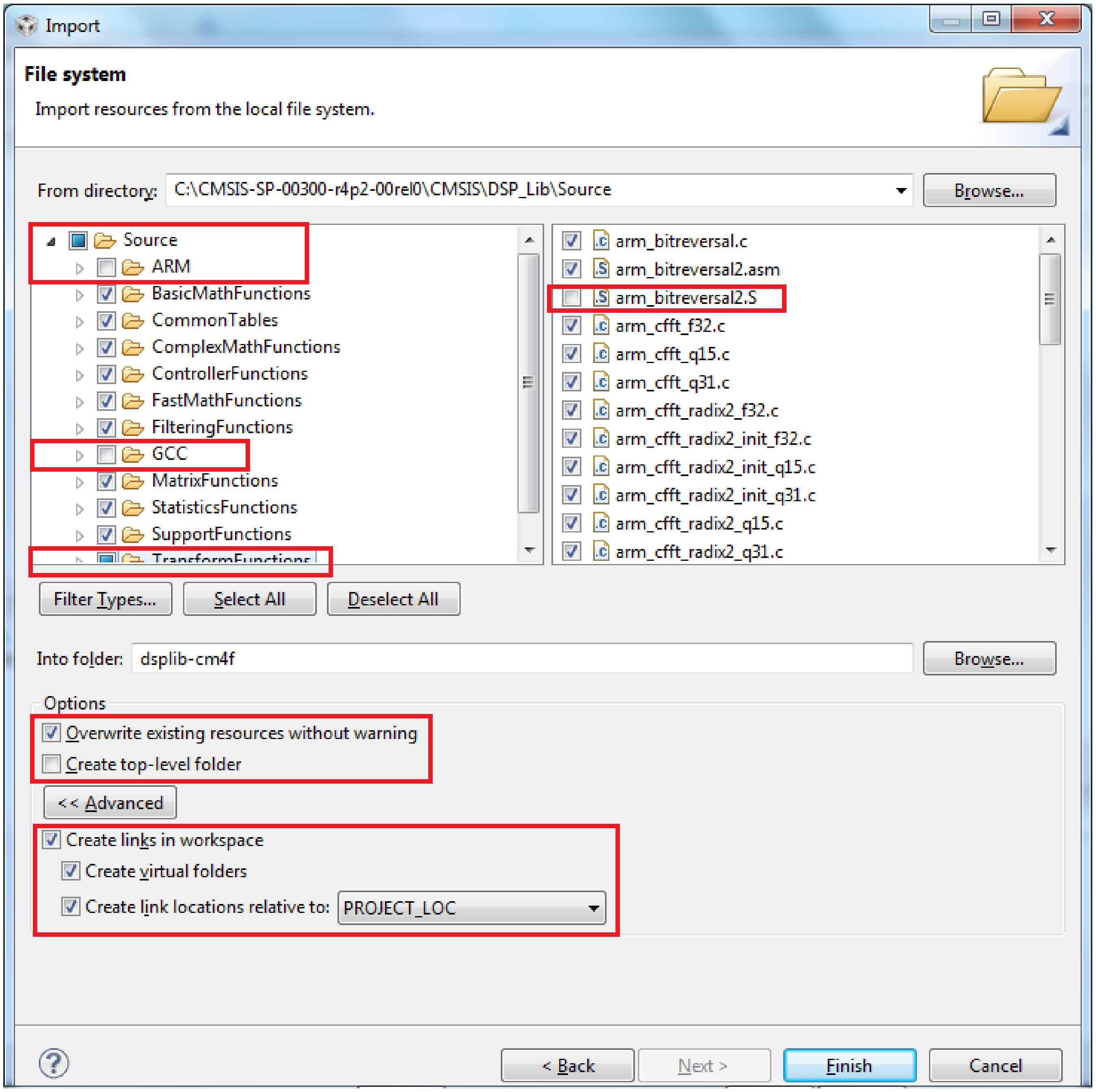
3.3 Adding the dsplib Source Code

Before adding the dsplib source code to the project, you should familiarize yourself with the CMSIS library structure. Open your preferred file navigation tool and navigate to the directory where the CMSIS .zip file downloaded from ARM was extracted. Then, descend to CMSIS-<*version*>/CMSIS/DSP_Lib/Source/. This is the directory ARM uses to group the DSP functions into various sub-categories. The ARM directory contains the project files necessary to build the DSP library in uVision with ARM's compiler, and the GCC directory contains the project files to build the DSP library in uVision using the open source GCC compiler. All other directories contain the source code necessary to build the category of functions indicated by the directory name.

To add the dsplib source code to the dsplib project in Code Composer Studio:

1. Right-click the dsplib-cm4f project in the Project Explorer and click *Import…*
2. Click *General* to expand and then click *File System*. Click **Next**.
3. Click the **Browse** button and navigate to the location of the CMSIS DSP library source code.
4. Select the top level Source directory and click **OK**.
5. When the Source directory appears in the Import window, click the checkbox beside the folder to select all of the contents of that folder to be imported.
6. Deselect the ARM and GCC folders by clicking to the left of the checkbox.
7. Click the TransformFunctions folder, which causes the contents of that folder to be displayed in the panel on the right.
8. Uncheck the box beside arm_bitreversal2.S.
9. Make sure that the *Into Folder:* text field contains the name of the DSP library project where you want to import the files (for this example, dsplib-cm4f).
10. Check the Overwrite existing resources without warning. Verify the *Create top-level folder* check box is deselected.
11. Click the **Advanced** button, then click to select the *Create links in workspace* checkbox.
12. Verify the *Create link locations relative to:* checkbox is selected. If it is not, click to select it.

13. Verify that the drop-down menu of environment variables is set to PROJECT_LOC (see Figure 3). If there are no variables listed in the drop-down menu, select *Edit Variables…* and add a variable to represent the location of the dsplib project file.



Import

File system

Import resources from the local file system.

From directory: C:\CMSIS-SP-00300-r4p2-00rel0\CMSIS\DSP_Lib\Source

- Source
  - [ ] ARM
  - [✓] BasicMathFunctions
  - [✓] CommonTables
  - [✓] ComplexMathFunctions
  - [✓] ControllerFunctions
  - [✓] FastMathFunctions
  - [✓] FilteringFunctions
  - [ ] GCC
  - [✓] MatrixFunctions
  - [✓] StatisticsFunctions
  - [✓] SupportFunctions
  - TransformFunctions

- [✓] arm_bitreversal.c
- [✓] arm_bitreversal2.asm
- [ ] arm_bitreversal2.S
- [✓] arm_cfft_f32.c
- [✓] arm_cfft_q15.c
- [✓] arm_cfft_q31.c
- [✓] arm_cfft_radix2_f32.c
- [✓] arm_cfft_radix2_init_f32.c
- [✓] arm_cfft_radix2_init_q15.c
- [✓] arm_cfft_radix2_init_q31.c
- [✓] arm_cfft_radix2_q15.c
- [✓] arm_cfft_radix2_q31.c

Into folder: dsplib-cm4f

Options
- [✓] Overwrite existing resources without warning
- [ ] Create top-level folder
- [✓] Create links in workspace
  - [✓] Create virtual folders
  - [✓] Create link locations relative to: PROJECT_LOC

Figure 3. Importing the DSP_Lib Source Code

14. Click **Finish** to link the DSP_Lib source code into the project.

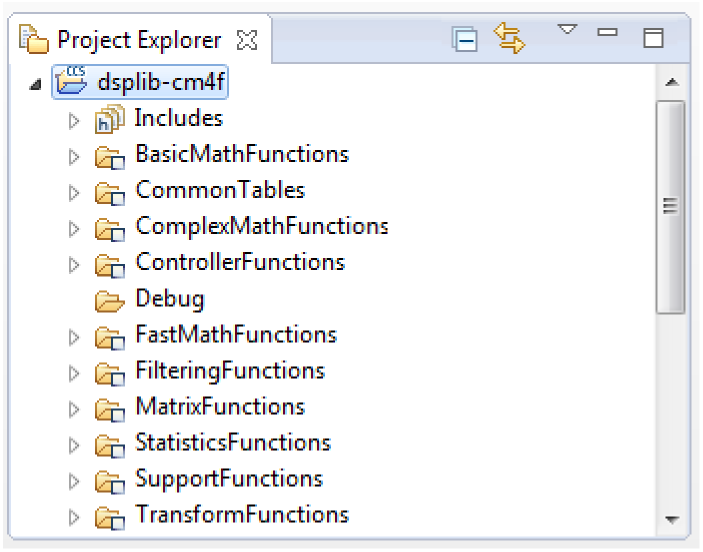
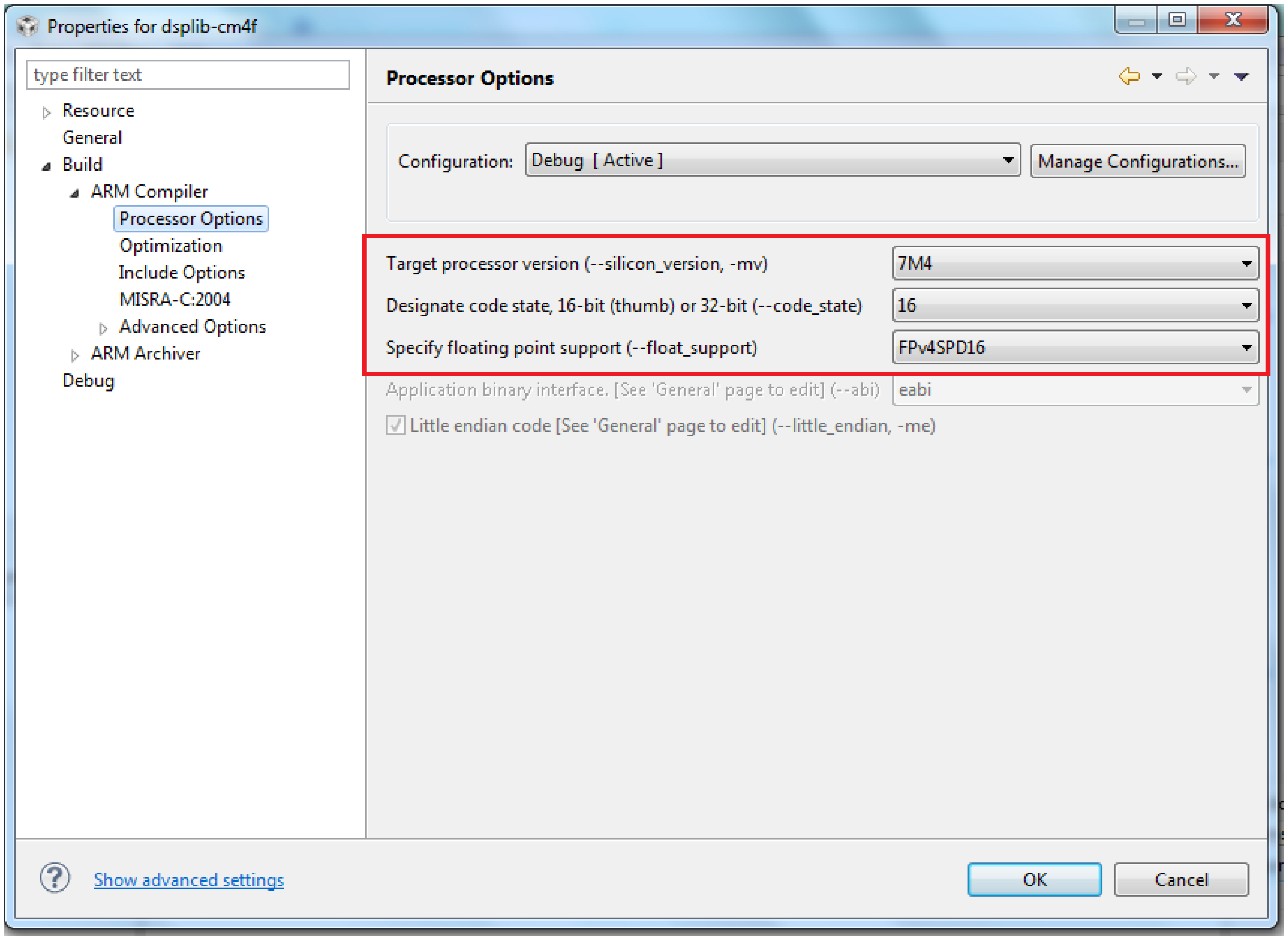


Figure 4. The Project Explorer Window After the DSP_Lib Code has Been Imported

3.4 Editing the dsplib Project Settings

After linking in all the source files, change the following default Code Composer Studio project settings:

1. Right-click the dsplib-cm4f project in the Project Explorer and select *Properties*.
2. Expand the *Build* entry, and then expand the *ARM Compiler* entry.
3. Confirm that the *Target_processor version (--silicon_version, -mv)* entry matches your processor in the Processor Options panel (see Figure 5). For this example, the Target processor should be 7M4 (as opposed to 7M3 for any of the Stellaris Cortex-M3 products).

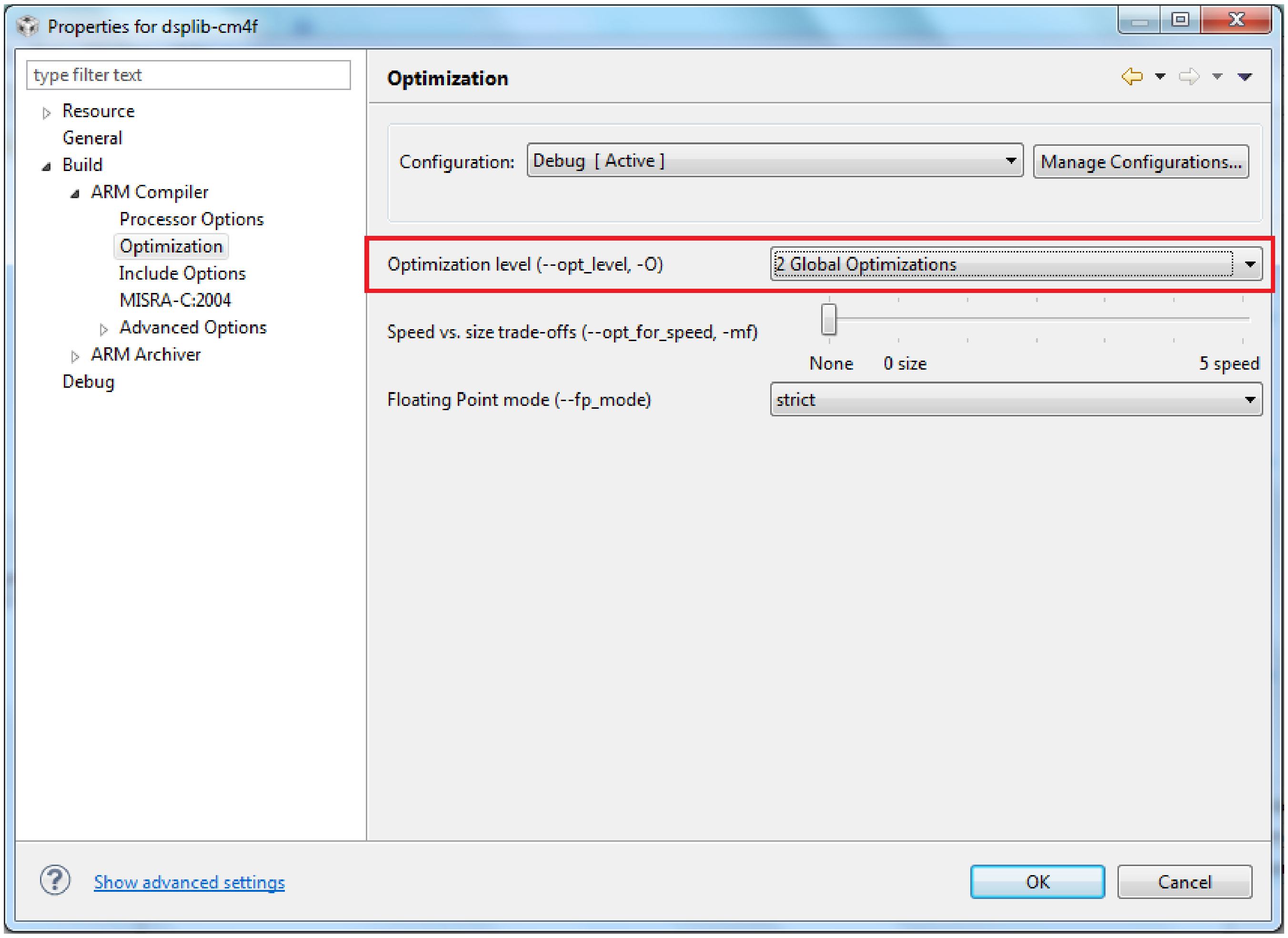


Configuration: Debug [Active]

Target processor version (--silicon_version, -mv): 7M4
Designate code state, 16-bit (thumb) or 32-bit (--code_state): 16
Specify floating point support (--float_support): FPv4SPD16
Application binary interface. [See 'General' page to edit] (--abi): eabi
[✓] Little endian code [See 'General' page to edit] (--little_endian, -me)

Figure 5. The Processor Settings for a Cortex-M4 Processor With Hardware FPU Support

4. Click the *Optimization level (--opt_level, -O)* drop-down menu in the Optimization panel and select 2 (see Figure 6).



Properties for dsplib-cm4f

type filter text
- Resource
- General
- Build
  - ARM Compiler
    - Processor Options
    - Optimization
    - Include Options
    - MISRA-C:2004
    - Advanced Options
  - ARM Archiver
- Debug

Optimization

Configuration: Debug [Active]

Optimization level (--opt_level, -O): 2 Global Optimizations

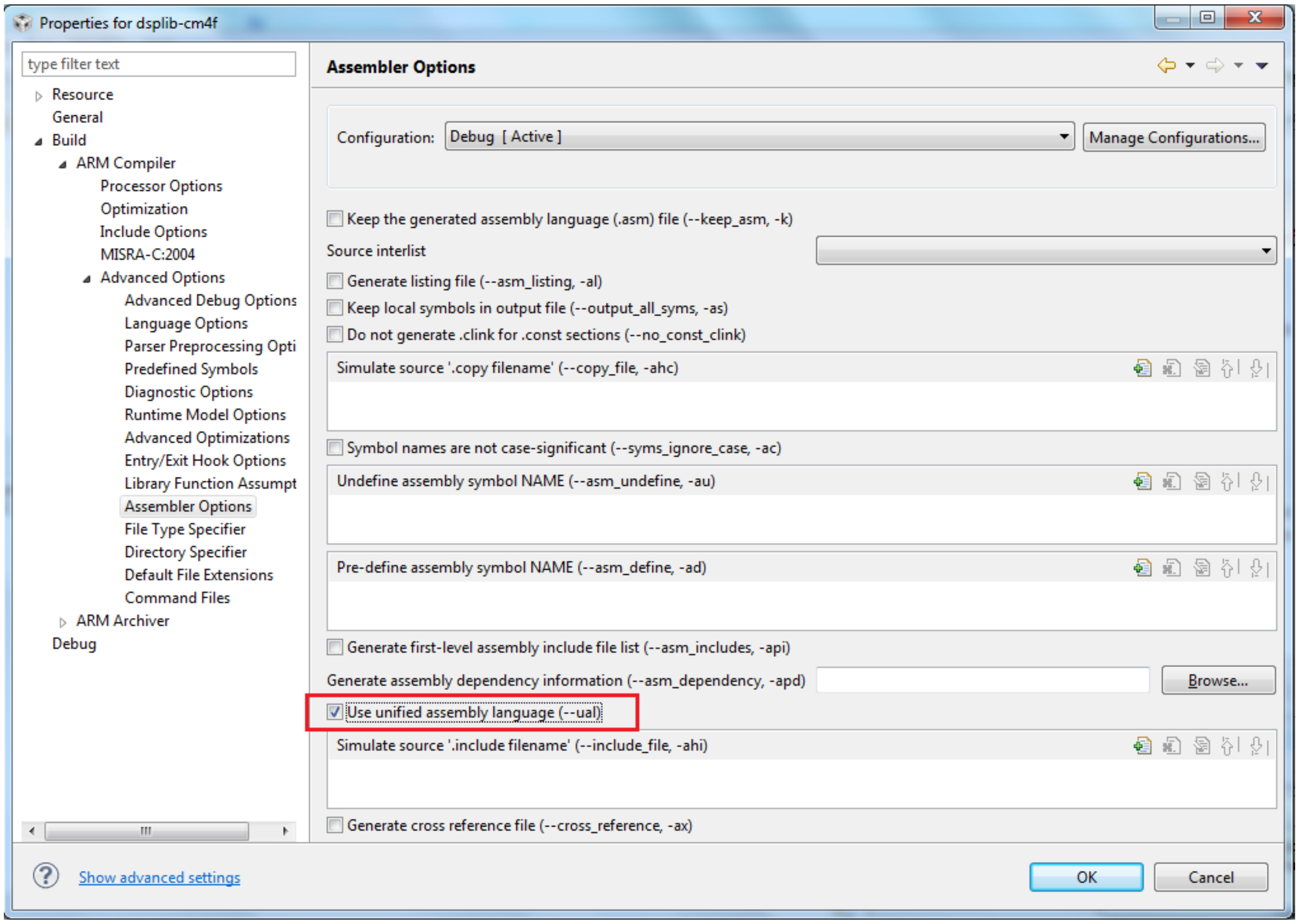
Speed vs. size trade-offs (--opt_for_speed, -mf): None

Floating Point mode (--fp_mode): strict

Figure 6. The Proper Optimization Settings for Compiling the DSP_Lib Source Code

5. Expand the *Advanced Options* section of the *ARM Compiler* pane, and select *Assembler Options*.

6. Click the *Use unified assembly language (--ual)* checkbox to select that option (see Figure 7).



Properties for dsplib-cm4f

type filter text

- Resource
- General
- Build
  - ARM Compiler
    - Processor Options
    - Optimization
    - Include Options
    - MISRA-C:2004
    - Advanced Options
      - Advanced Debug Options
      - Language Options
      - Parser Preprocessing Opti
      - Predefined Symbols
      - Diagnostic Options
      - Runtime Model Options
      - Advanced Optimizations
      - Entry/Exit Hook Options
      - Library Function Assumpt
      - Assembler Options
      - File Type Specifier
      - Directory Specifier
      - Default File Extensions
      - Command Files
  - ARM Archiver
- Debug

Assembler Options

Configuration: Debug [Active] Manage Configurations...

- [ ] Keep the generated assembly language (.asm) file (--keep_asm, -k)

Source interlist

- [ ] Generate listing file (--asm_listing, -al)
- [ ] Keep local symbols in output file (--output_all_syms, -as)
- [ ] Do not generate .clink for .const sections (--no_const_clink)

Simulate source '.copy filename' (--copy_file, -ahc)

- [ ] Symbol names are not case-significant (--syms_ignore_case, -ac)

Undefine assembly symbol NAME (--asm_undefine, -au)

Pre-define assembly symbol NAME (--asm_define, -ad)

- [ ] Generate first-level assembly include file list (--asm_includes, -api)

Generate assembly dependency information (--asm_dependency, -apd) Browse...

- [✓] Use unified assembly language (--ual)

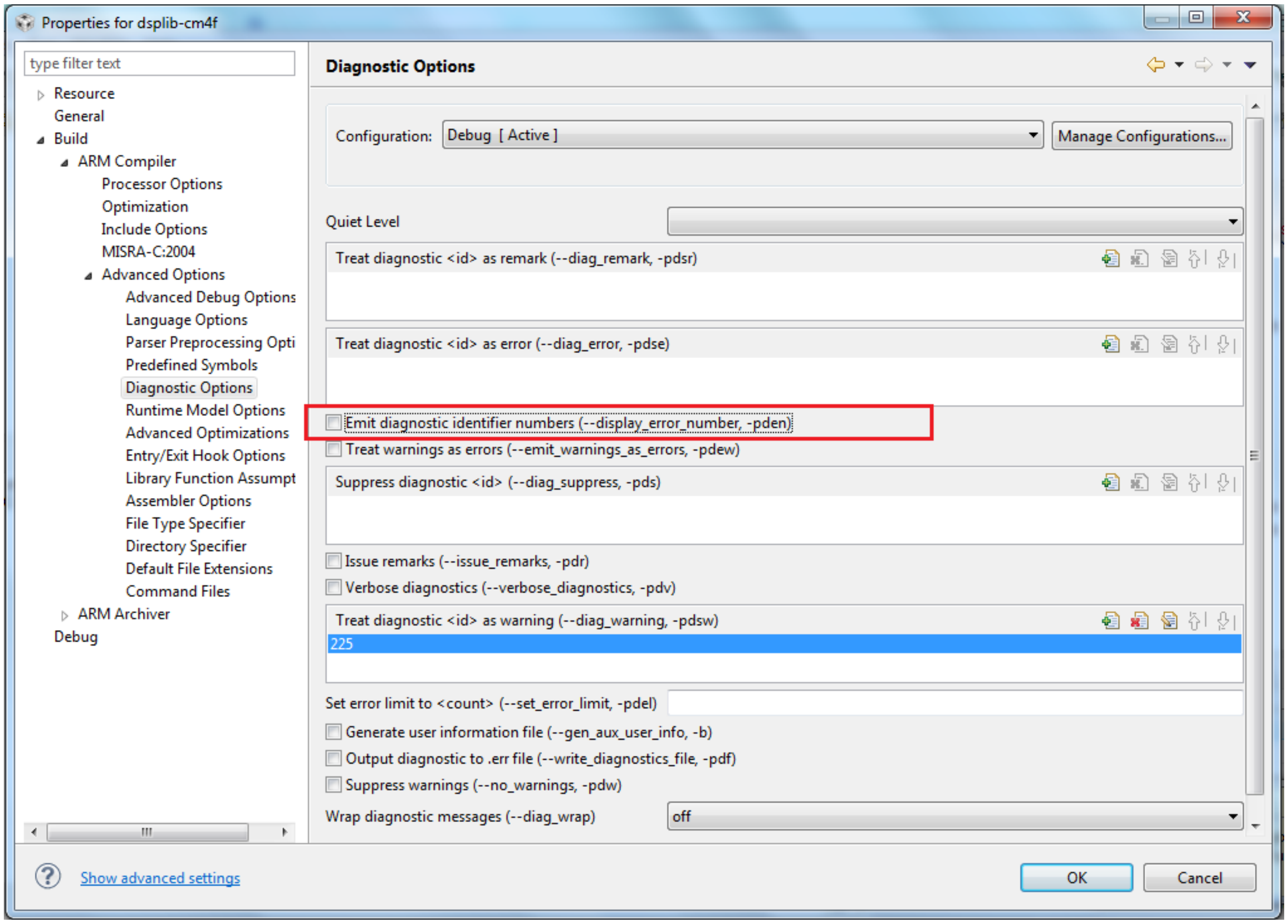
Simulate source '.include filename' (--include_file, -ahi)

- [ ] Generate cross reference file (--cross_reference, -ax)

Show advanced settings OK Cancel

Figure 7. Setting the Assembler to Use Unified Assembly Language

7. Click the *Emit diagnostic identifier numbers (--display_error_number, -pden)* checkbox in the Diagnostic Options panel to deselect.



Configuration: Debug [Active]

Quiet Level

Treat diagnostic <id> as remark (--diag_remark, -pdsr)

Treat diagnostic <id> as error (--diag_error, -pdse)

☐ Emit diagnostic identifier numbers (--display_error_number, -pden)

☐ Treat warnings as errors (--emit_warnings_as_errors, -pdew)

Suppress diagnostic <id> (--diag_suppress, -pds)

☐ Issue remarks (--issue_remarks, -pdr)

☐ Verbose diagnostics (--verbose_diagnostics, -pdv)

Treat diagnostic <id> as warning (--diag_warning, -pdsw)
225

Set error limit to <count> (--set_error_limit, -pdel)

☐ Generate user information file (--gen_aux_user_info, -b)

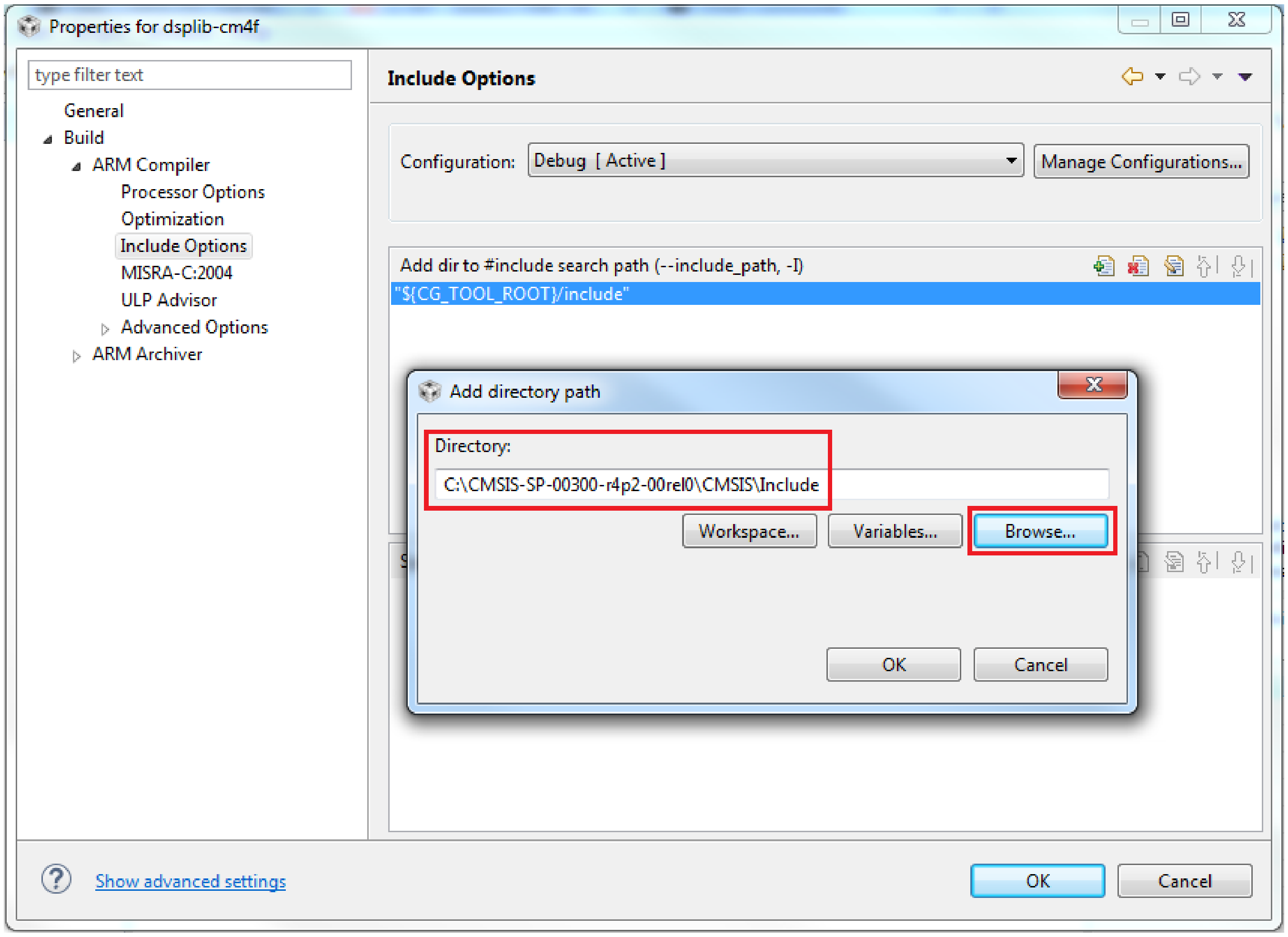
☐ Output diagnostic to .err file (--write_diagnostics_file, -pdf)

☐ Suppress warnings (--no_warnings, -pdw)

Wrap diagnostic messages (--diag_wrap) off

Figure 8. Verifying That Diagnostic Identifier Numbers will not be Emitted

8. Add the DSP library CMSIS-<*version*>/CMSIS/Include directory to the compiler's include path in the Include Options panel. This is done by pressing the *Add* button by the *Add dir to #include search path (--include_path, -I)* (see Figure 9), then either typing in the path to the CMSIS Include directory or clicking *Browse* and navigating to the Include directory and navigating to the Include directory (CMSIS-<*version*>/CMSIS/Include).



Configuration: Debug [Active]

Add dir to #include search path (--include_path, -I)

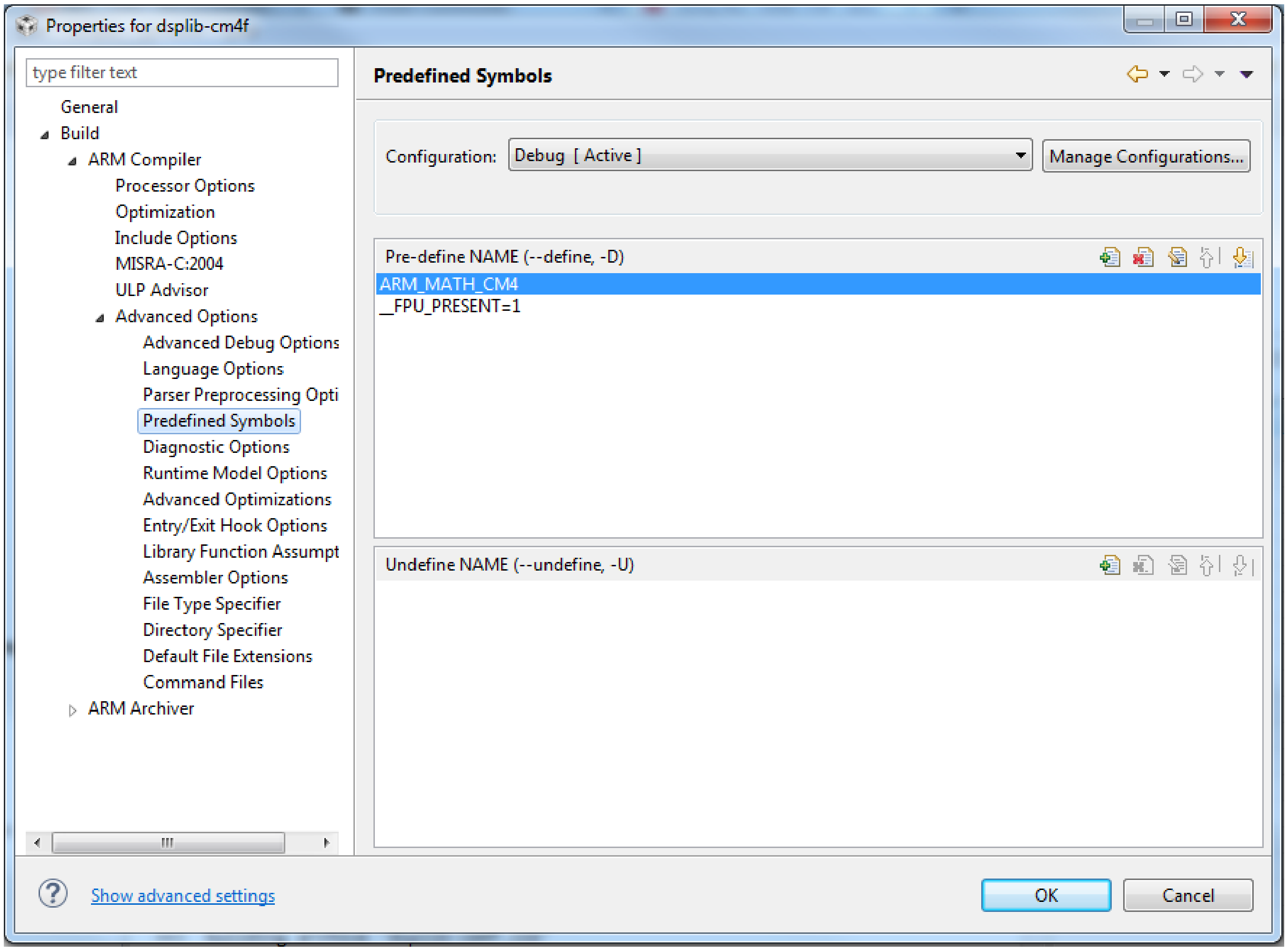
"${CG_TOOL_ROOT}/include"

Directory:

C:\CMSIS-SP-00300-r4p2-00rel0\CMSIS\Include

Figure 9. Adding the CMSIS Top Level Include Directory to the Compiler's #include Search Path

9. Expand the Advanced Options menu again and select the Predefined Symbols panel. Create a symbol to tell the DSP library to use Cortex-M4 based math functions. Click the *Add...* button in the *Pre-define NAME (--define, -D)* area. In the *Enter Value* dialog box, type ARM_MATH_CM4 into the *Pre-define NAME (--define, -D)* field and click **OK** (see Figure 10). Click the *Add…* button again type __FPU_PRESENT=1 into the *Pre-define NAME (--define, -D)* field and click **OK**.



Configuration: Debug [Active]

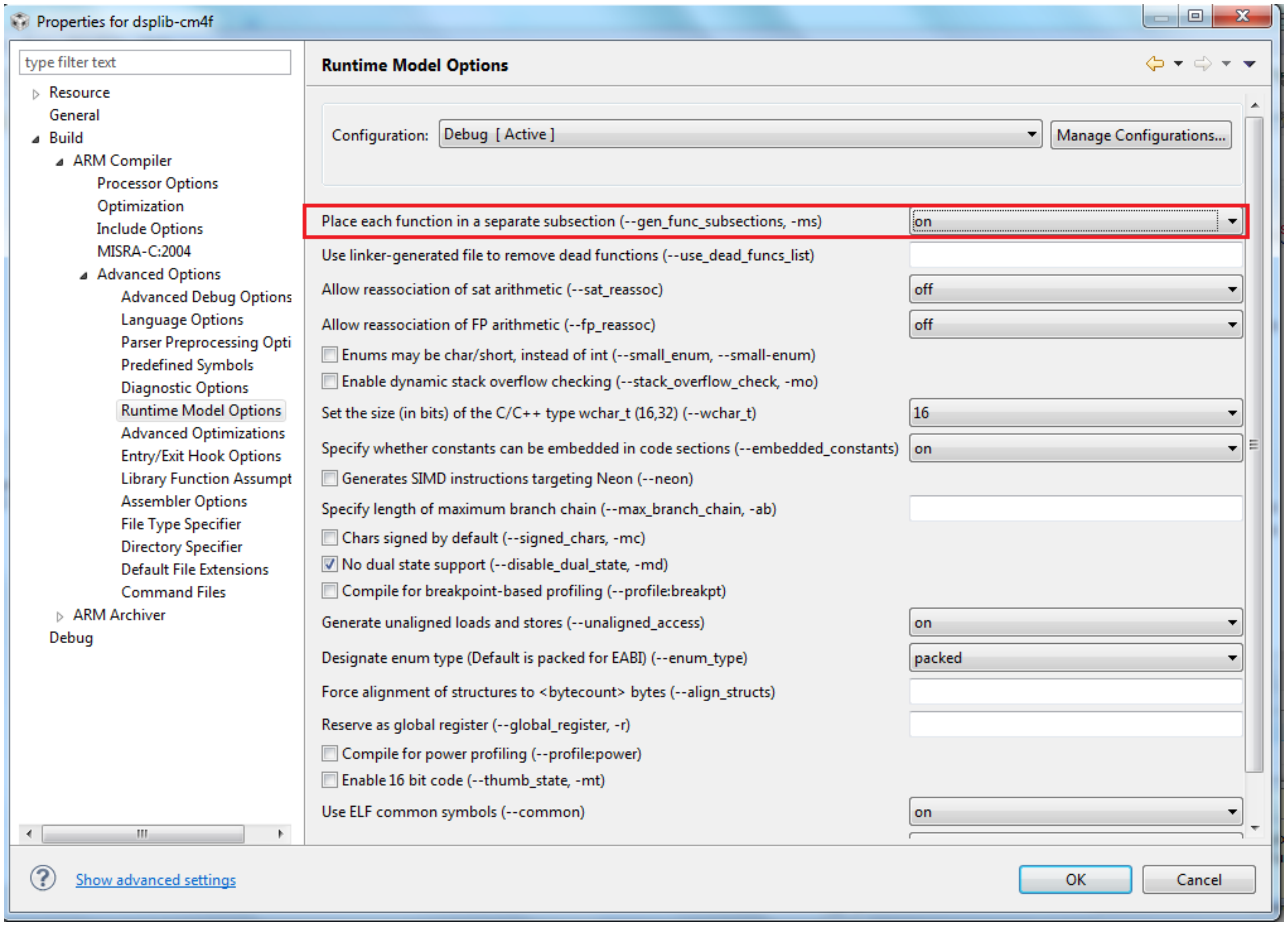
Pre-define NAME (--define, -D)

```
ARM_MATH_CM4
__FPU_PRESENT=1
```

Undefine NAME (--undefine, -U)

Figure 10. Adding Project Level #defines for the Processor Characteristics

10. Select the *on* option from the *Place each function in a separate subsection (--gen_func_subsections, -ms)* drop-down menu in the Runtime Model Options panel (see Figure 11).



Properties for dsplib-cm4f

Runtime Model Options

Configuration: Debug [Active]

Setting	Value
Place each function in a separate subsection (--gen_func_subsections, -ms)	on
Use linker-generated file to remove dead functions (--use_dead_funcs_list)	
Allow reassociation of sat arithmetic (--sat_reassoc)	off
Allow reassociation of FP arithmetic (--fp_reassoc)	off
Enums may be char/short, instead of int (--small_enum, --small-enum)	☐
Enable dynamic stack overflow checking (--stack_overflow_check, -mo)	☐
Set the size (in bits) of the C/C++ type wchar_t (16,32) (--wchar_t)	16
Specify whether constants can be embedded in code sections (--embedded_constants)	on
Generates SIMD instructions targeting Neon (--neon)	☐
Specify length of maximum branch chain (--max_branch_chain, -ab)	
Chars signed by default (--signed_chars, -mc)	☐
No dual state support (--disable_dual_state, -md)	✓
Compile for breakpoint-based profiling (--profile:breakpt)	☐
Generate unaligned loads and stores (--unaligned_access)	on
Designate enum type (Default is packed for EABI) (--enum_type)	packed
Force alignment of structures to <bytecount> bytes (--align_structs)	
Reserve as global register (--global_register, -r)	
Compile for power profiling (--profile:power)	☐
Enable 16 bit code (--thumb_state, -mt)	☐
Use ELF common symbols (--common)	on

Figure 11. The Proper Runtime Model Options for Compiling the DSP_Lib Source Code

3.5 *Building the dsplib Source Code*

Build the CMSIS DSP libraries by right-clicking dsplib-cm4f in the Project Explorer and selecting *Build Project.* Depending on hardware, this build might take up to ten minutes to complete. After the build is finished, the resulting dsplib-cm4f.lib file is created in the Debug folder of the project workspace.

NOTE: The CMSIS file structure currently contains a directory located at CMSIS-<*version*>/CMSIS/Lib that is intended for storing compiled library files. It is recommended for organization's sake that this directory be used for storing the Code Composer Studio compiled CMSIS DSP libraries. To do so, create a CCS/M4 sub-directory inside CMSIS/Lib, then copy the .lib that was generated by the above steps into the Code Composer Studio sub-directory.

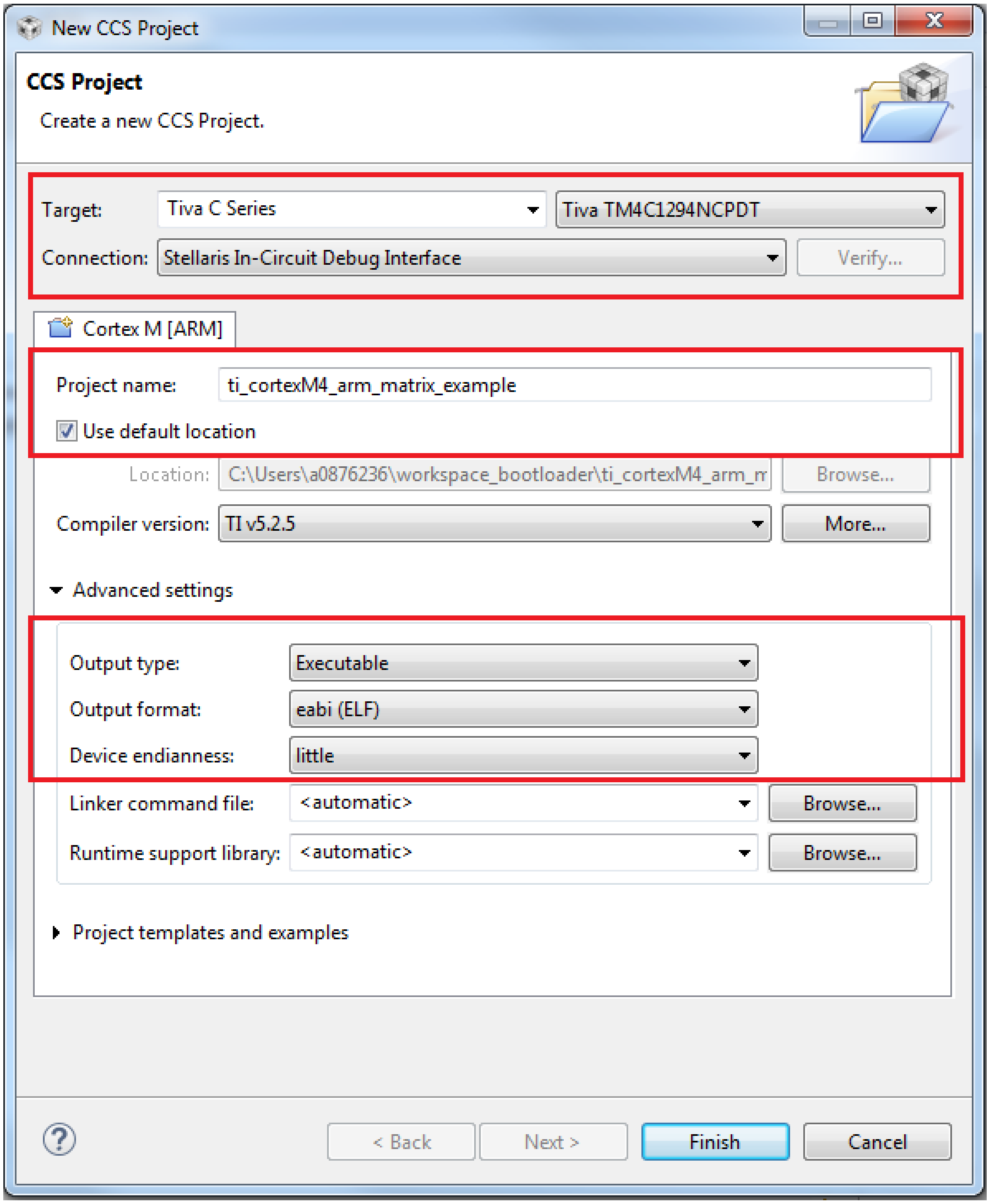
4 ARM Example Projects

The ARM CMSIS download contains eleven example projects that demonstrate how to use the various DSP library functions. This section details the steps required to create the same projects in Code Composer Studio v6.1, compile the projects, and run them on a TM4C microcontroller. These steps are focused on running the code on an EK-TM4C129 Connected LaunchPad, but can be easily modified to work with any other TM4C or Stellaris MCU (see http://www.ti.com/tool/ek-tm4c1294xl).

4.1 Creating the ARM Example Projects

The source code for all of the example projects can be found at CMSIS-<*version*>/CMSIS/DSP_Lib/Examples. Projects for each of the ARM examples can be created in Code Composer Studio via the following steps:

1. Launch CCSv6.1 and select either an empty workspace or the workspace used in the previous section to build the DSP library.
2. Select File > New > CCS Project. The *New CCS Project* window will be displayed.
3. Type ti_cortexM4_<example name> in the *Project name* field.
4. Select Executable from the Output Advanced settings.
5. Make the following selections from the Target Device:
   - Target: TM4C Series
   - TM4C1294NCPDT
   - Connections: Stellaris In-Circuit Debug Interface
6. Select the Empty Project in the *Project templates and examples* field.
7. Click **Finish** to create the project (see Figure 12 and Figure 13). The project now appears in the Project Explorer.


New CCS Project

CCS Project

Create a new CCS Project.

Target: Tiva C Series | Tiva TM4C1294NCPDT

Connection: Stellaris In-Circuit Debug Interface

Cortex M [ARM]

Project name: ti_cortexM4_arm_matrix_example

☑ Use default location

Location: C:\Users\a0876236\workspace_bootloader\ti_cortexM4_arm_m

Compiler version: TI v5.2.5

Advanced settings

Output type: Executable

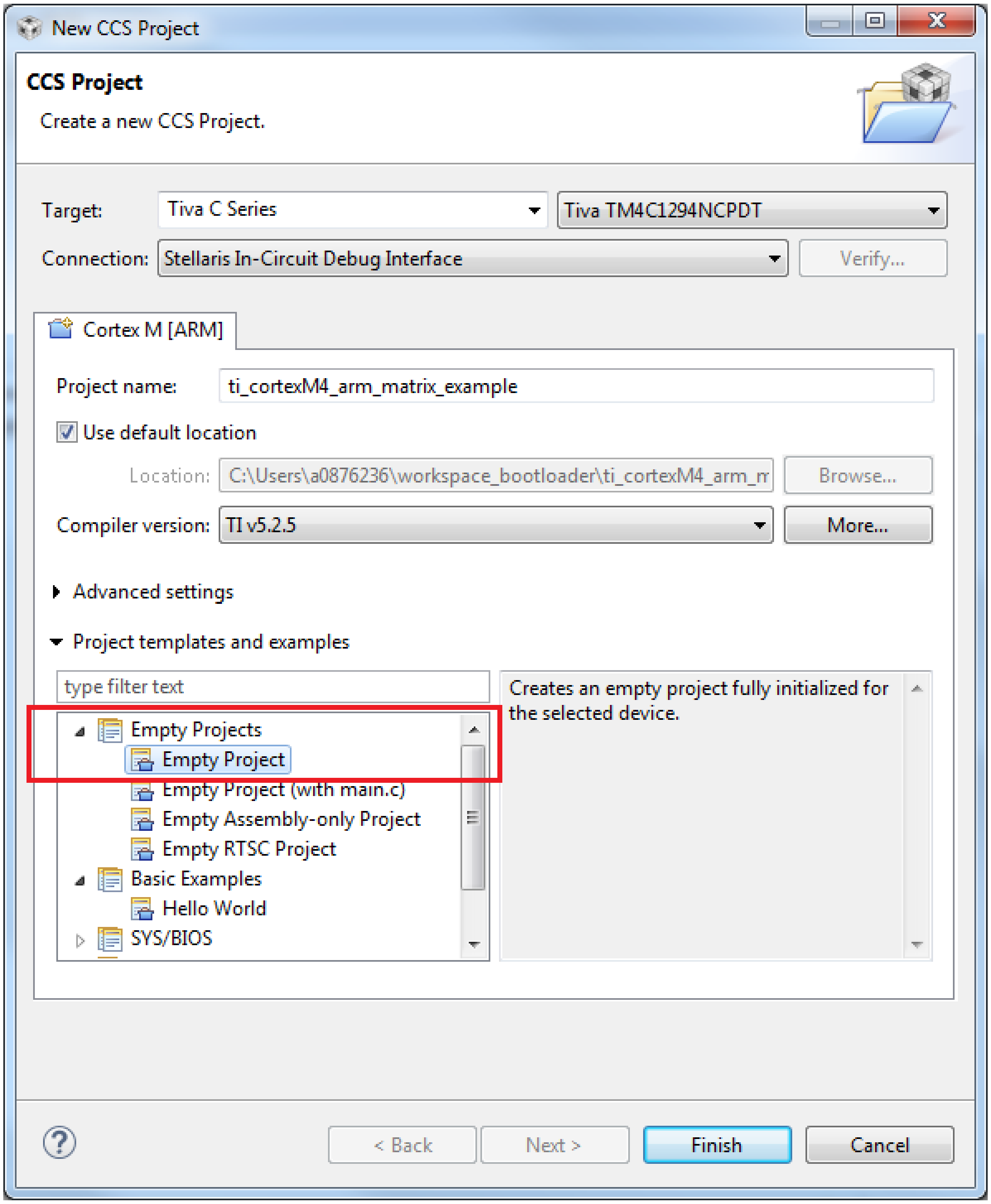
Output format: eabi (ELF)

Device endianness: little

Linker command file: <automatic>

Runtime support library: <automatic>

Figure 12. The New CCS Project Window With Options Set to Build the arm_dotproduct_example Project



New CCS Project

CCS Project

Create a new CCS Project.

Target: Tiva C Series — Tiva TM4C1294NCPDT

Connection: Stellaris In-Circuit Debug Interface

Cortex M [ARM]

Project name: ti_cortexM4_arm_matrix_example

☑ Use default location

Location: C:\Users\a0876236\workspace_bootloader\ti_cortexM4_arm_m

Compiler version: TI v5.2.5

Advanced settings

Project templates and examples

type filter text

- Empty Projects
  - Empty Project
  - Empty Project (with main.c)
  - Empty Assembly-only Project
  - Empty RTSC Project
- Basic Examples
  - Hello World
- SYS/BIOS

Creates an empty project fully initialized for the selected device.

Figure 13. The New CCS Project Window With Options Set to Build the arm_dotproduct_example Project

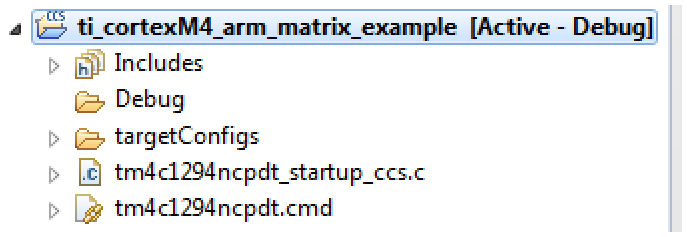
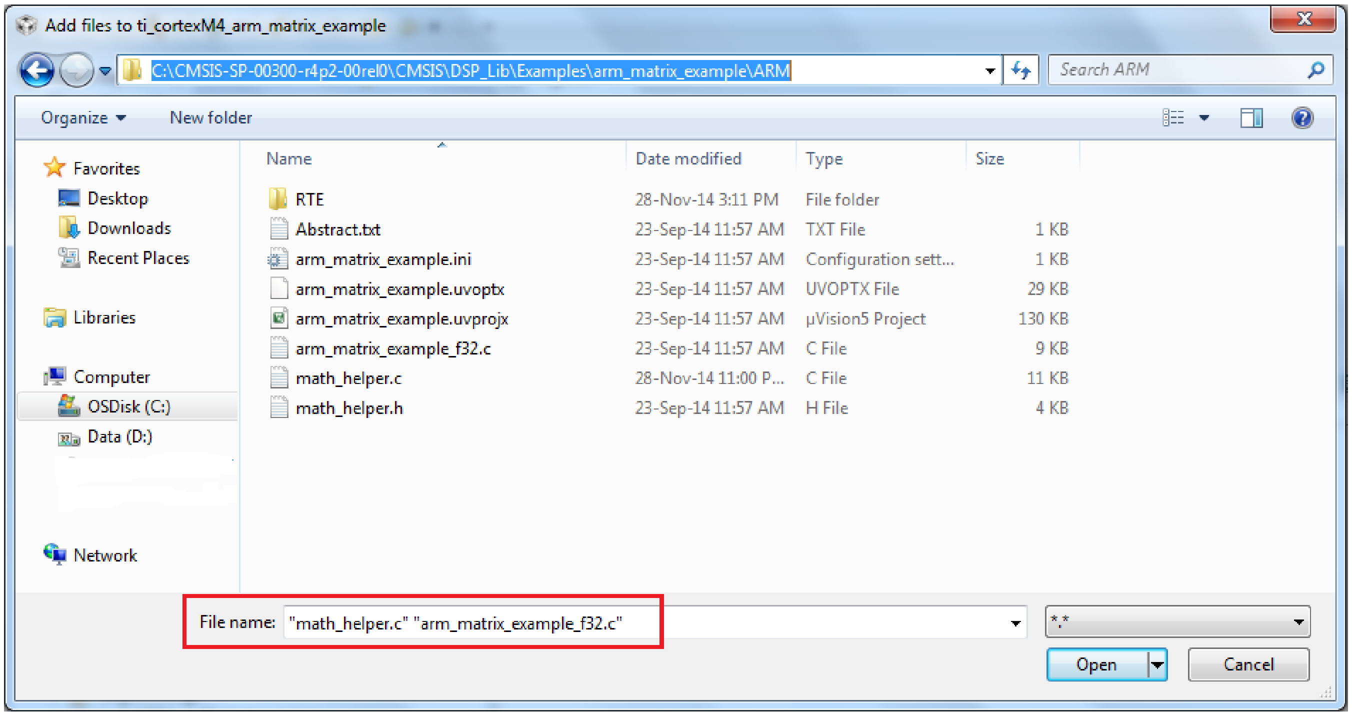

ti_cortexM4_arm_matrix_example [Active - Debug]
Includes
Debug
targetConfigs
tm4c1294ncpdt_startup_ccs.c
tm4c1294ncpdt.cmd

Figure 14. The Project Explorer After the arm_matrix_example Project has Been Created

4.2 Adding the Example Source Code

Once the project is created, it is necessary to point the project to the source files necessary for compilation:

1. Right-click the project in the Project Explorer and select *Add Files…*
2. Navigate to the CMSIS-<*version*>/CMSIS/DSP_Lib/Examples/<*example*> directory. Select all of the C source file and click **Open** (see Figure 15).

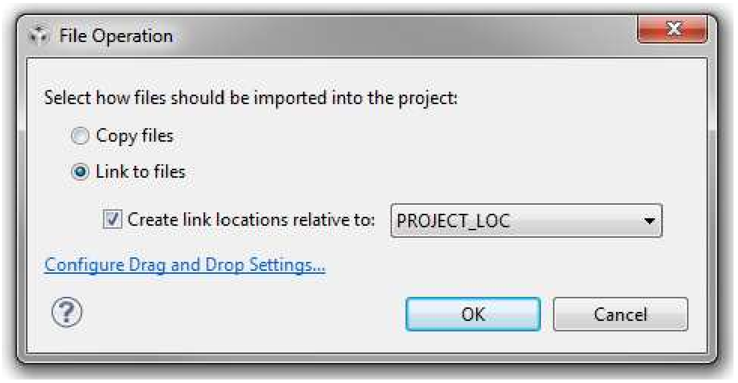


Name	Date modified	Type	Size
RTE	28-Nov-14 3:11 PM	File folder	
Abstract.txt	23-Sep-14 11:57 AM	TXT File	1 KB
arm_matrix_example.ini	23-Sep-14 11:57 AM	Configuration sett...	1 KB
arm_matrix_example.uvoptx	23-Sep-14 11:57 AM	UVOPTX File	29 KB
arm_matrix_example.uvprojx	23-Sep-14 11:57 AM	µVision5 Project	130 KB
arm_matrix_example_f32.c	23-Sep-14 11:57 AM	C File	9 KB
math_helper.c	28-Nov-14 11:00 P...	C File	11 KB
math_helper.h	23-Sep-14 11:57 AM	H File	4 KB

File name: "math_helper.c" "arm_matrix_example_f32.c"

Figure 15. Adding the Source Files for arm_matrix_example to the Project

3. Select the *Link to files* radio button, check the *Create link locations relative to:* checkbox, and select PROJECT_LOC from the drop-down menu when the File Operation dialog box appears (see Figure 16). Press **OK** to add the source file(s) to the project.


File Operation

Select how files should be imported into the project:

( ) Copy files
(●) Link to files

[✓] Create link locations relative to: PROJECT_LOC

Configure Drag and Drop Settings...

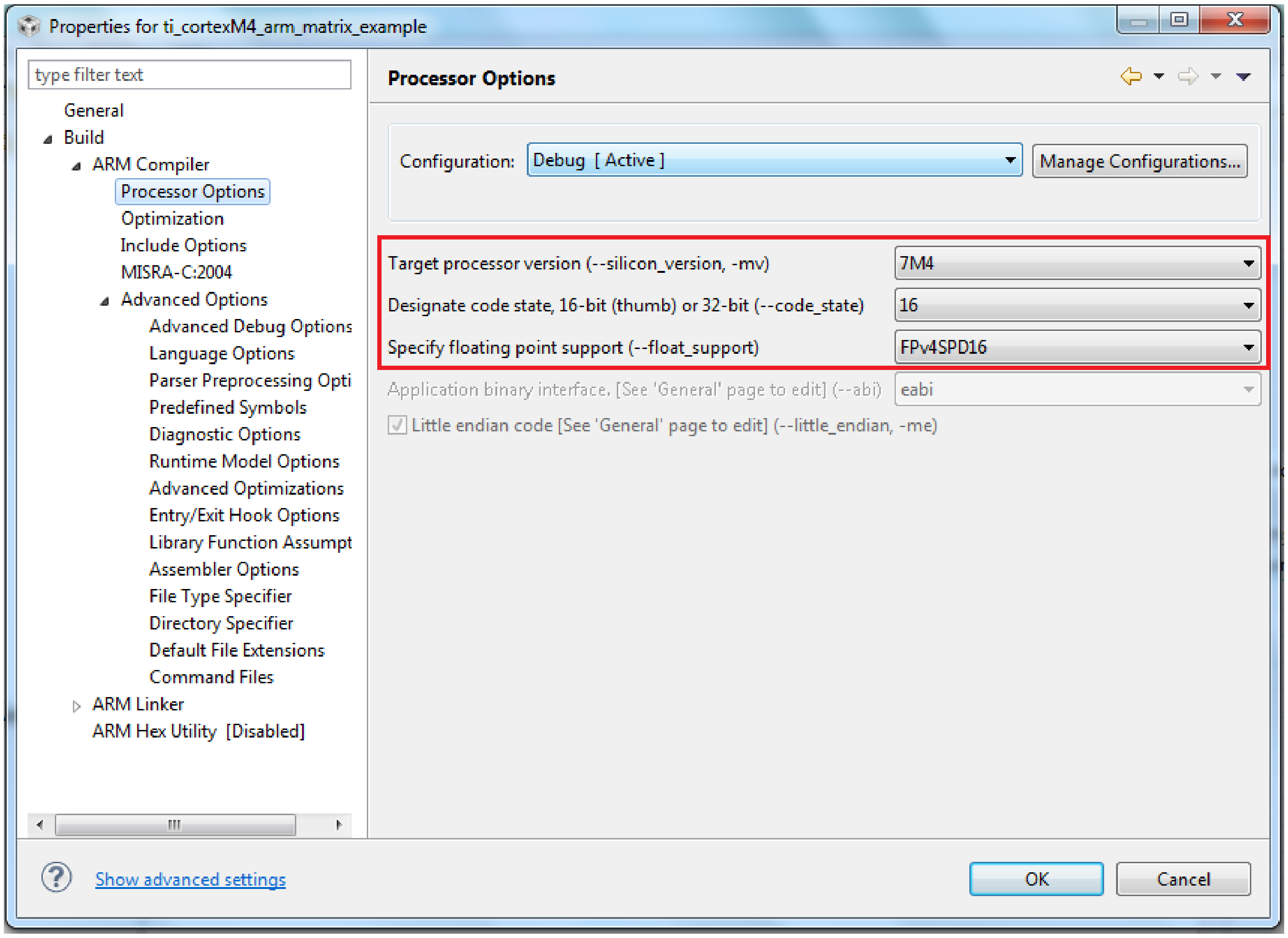
OK Cancel

Figure 16. Selecting the Proper Options to Link the Source Files Into the Project

4.3 Editing the Example Project Settings

Before building the example projects, it is necessary to properly configure the project settings:

1. Right-click the project in the Project Explorer and select *Properties*.
2. Expand the *Build* entry, and then expand the *ARM Compiler* entry.
3. Confirm that the *Target_processor version (--silicon_version, -mv)* entry matches your processor in the Processor Options panel (see Figure 17). For this example, the Target processor should be 7M4 (as opposed to 7M3 for any of the Stellaris Cortex-M3 products).



Configuration: Debug [Active]

Target processor version (--silicon_version, -mv): 7M4

Designate code state, 16-bit (thumb) or 32-bit (--code_state): 16

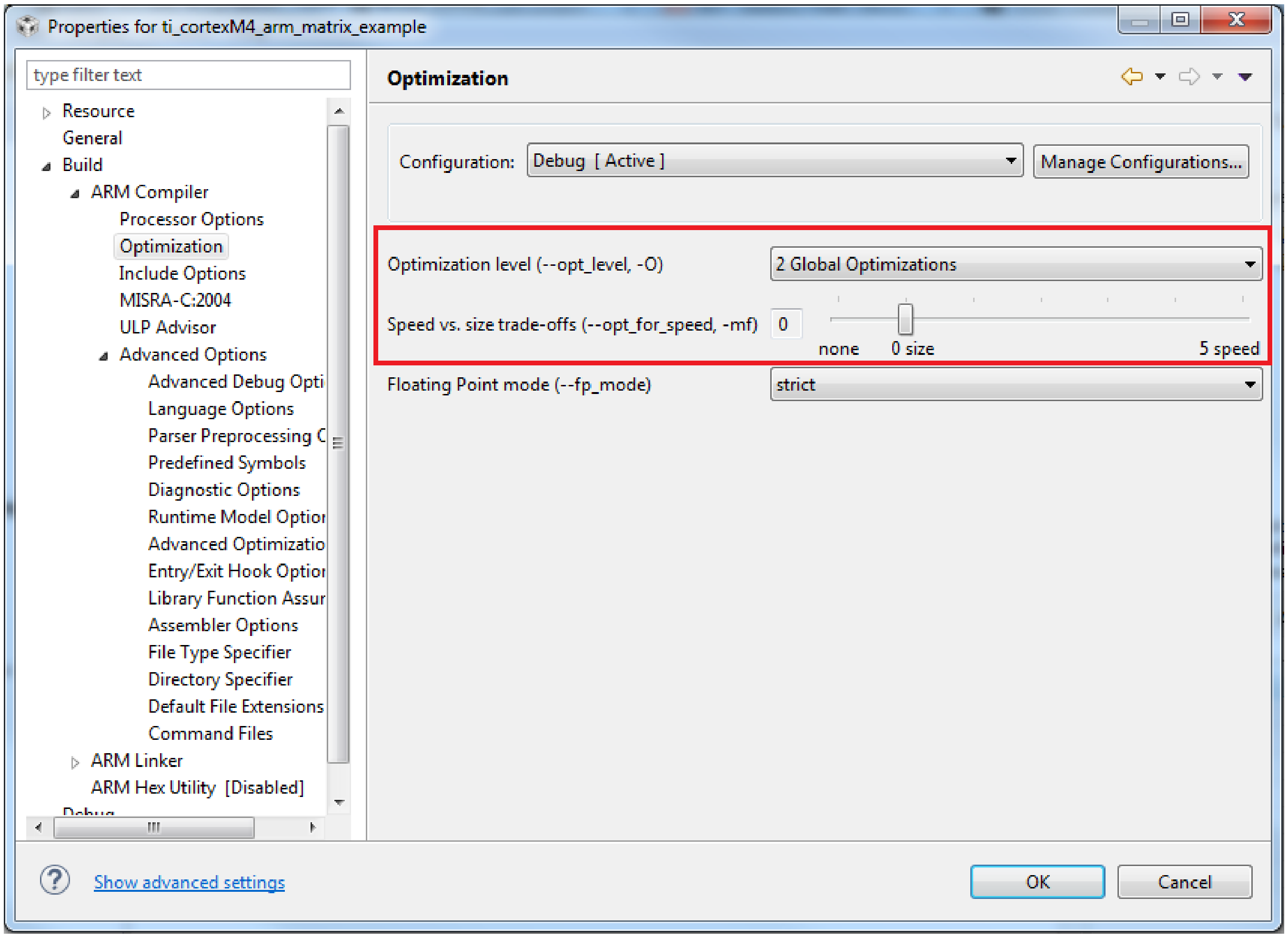
Specify floating point support (--float_support): FPv4SPD16

Application binary interface. [See 'General' page to edit] (--abi): eabi

☑ Little endian code [See 'General' page to edit] (--little_endian, -me)

Figure 17. The Processor Options Used for Building the Example on a Cortex-M4 Process With hardware FPU Support

4. Click the *Optimization level (--opt_level, -O)* drop-down menu and select 2 in the Optimization panel (see Figure 18).


Properties for ti_cortexM4_arm_matrix_example

Optimization

Configuration: Debug [Active]

Optimization level (--opt_level, -O): 2 Global Optimizations

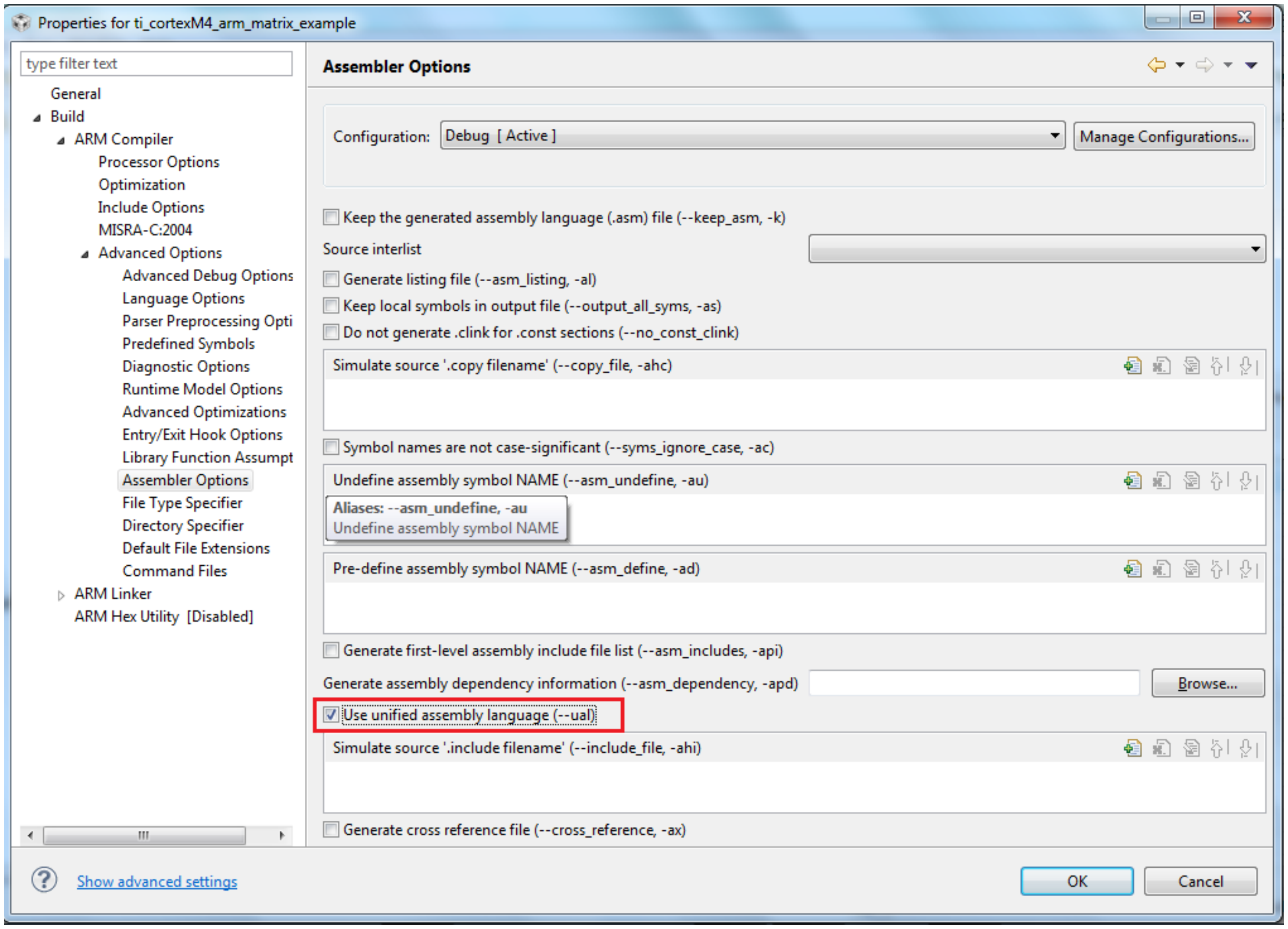
Speed vs. size trade-offs (--opt_for_speed, -mf): 0

Floating Point mode (--fp_mode): strict

Figure 18. The Proper Optimization Settings for Compiling the Example Projects

5. Expand the *Advanced Options* section of the *ARM Compiler* pane, and select *Assembler Options*.

6. Click the *Use unified assembly language (--ual)* checkbox to select that option (see Figure 19).


Properties for ti_cortexM4_arm_matrix_example

type filter text
- General
- Build
  - ARM Compiler
    - Processor Options
    - Optimization
    - Include Options
    - MISRA-C:2004
    - Advanced Options
      - Advanced Debug Options
      - Language Options
      - Parser Preprocessing Opti
      - Predefined Symbols
      - Diagnostic Options
      - Runtime Model Options
      - Advanced Optimizations
      - Entry/Exit Hook Options
      - Library Function Assumpt
      - Assembler Options
      - File Type Specifier
      - Directory Specifier
      - Default File Extensions
      - Command Files
  - ARM Linker
  - ARM Hex Utility [Disabled]

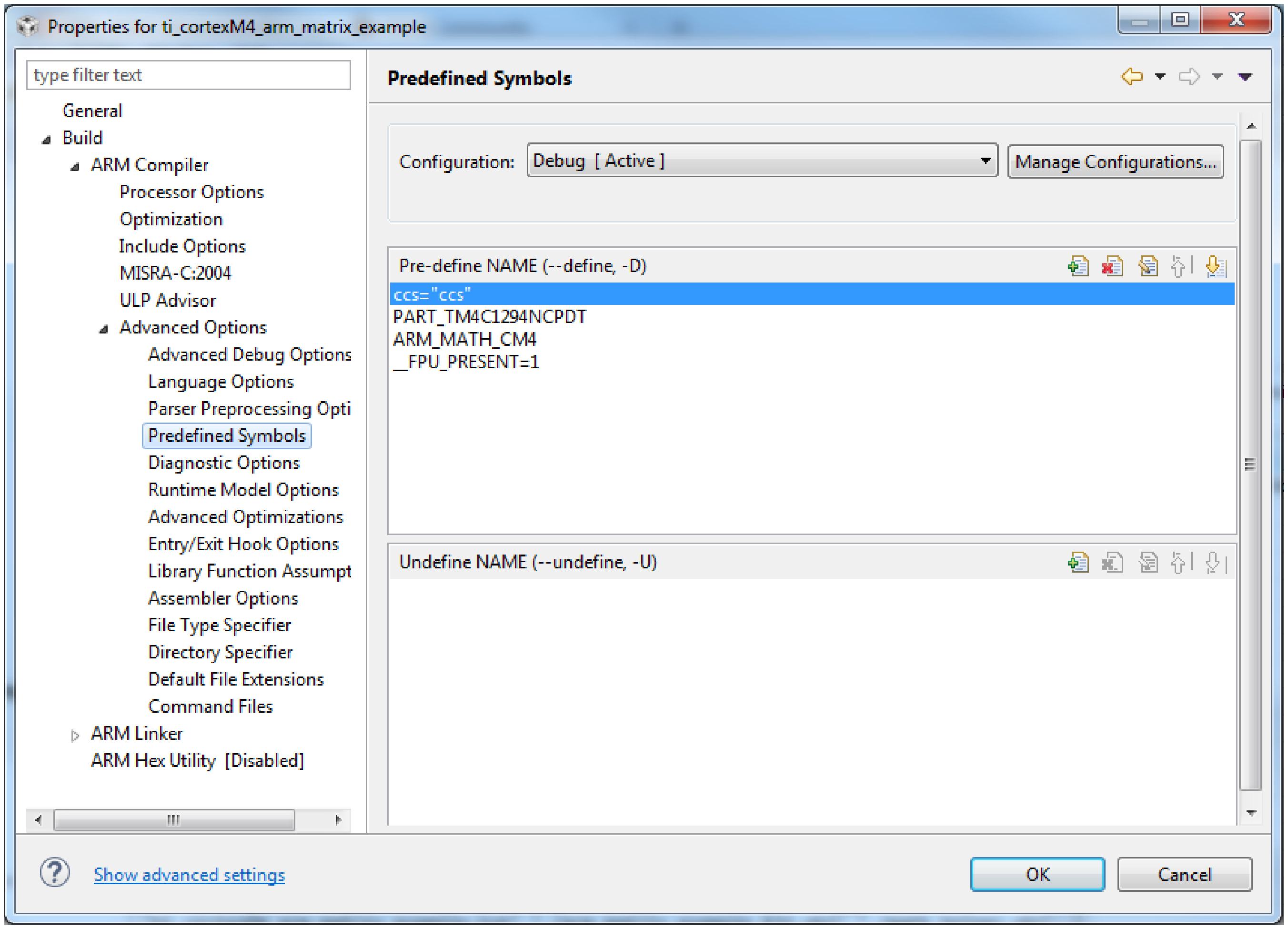
Assembler Options

Configuration: Debug [Active] Manage Configurations...

- [ ] Keep the generated assembly language (.asm) file (--keep_asm, -k)
- Source interlist
- [ ] Generate listing file (--asm_listing, -al)
- [ ] Keep local symbols in output file (--output_all_syms, -as)
- [ ] Do not generate .clink for .const sections (--no_const_clink)
- Simulate source '.copy filename' (--copy_file, -ahc)
- [ ] Symbol names are not case-significant (--syms_ignore_case, -ac)
- Undefine assembly symbol NAME (--asm_undefine, -au)
  - Aliases: --asm_undefine, -au
  - Undefine assembly symbol NAME
- Pre-define assembly symbol NAME (--asm_define, -ad)
- [ ] Generate first-level assembly include file list (--asm_includes, -api)
- Generate assembly dependency information (--asm_dependency, -apd) Browse...
- [✓] Use unified assembly language (--ual)
- Simulate source '.include filename' (--include_file, -ahi)
- [ ] Generate cross reference file (--cross_reference, -ax)

Figure 19. The Proper Assembler Options Needed for Compiling the Example Projects

7. Expand the Advanced Options menu again and select the Predefined Symbols panel. Create a symbol to tell the DSP library to use Cortex-M4 based math functions. Click the *Add...* button in the *Pre-define NAME (--define, -D)* area. In the *Enter Value* dialog box, type ARM_MATH_CM4 into the *Pre-define NAME (--define, -D)* field and click **OK** (see Figure 20). Click the *Add...* button in the *Pre-define NAME (--define, -D)* area. In the *Enter Value* dialog box, type __FPU_PRESENT=1 into the *Pre-define NAME (--define -D)* field and click OK.



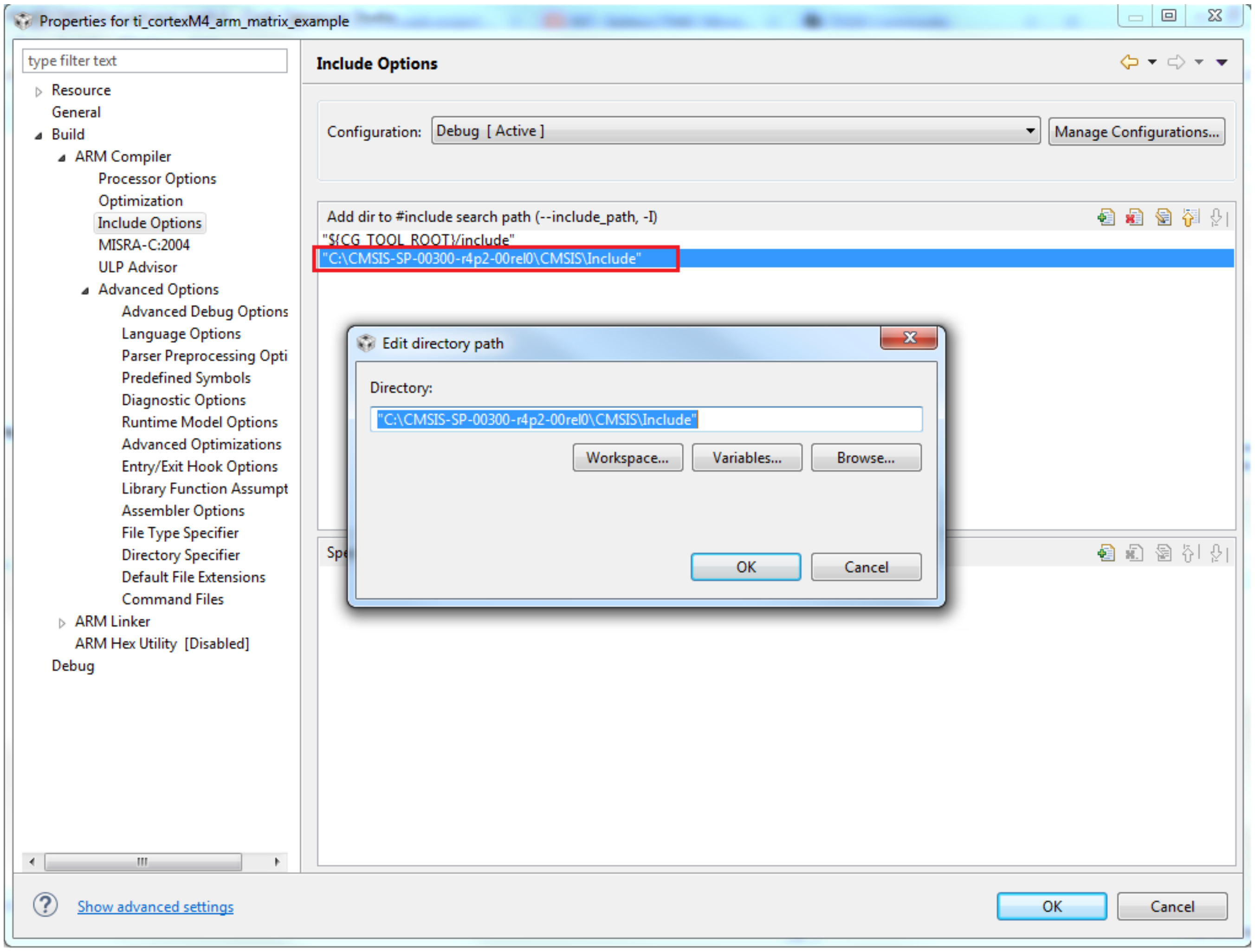
Configuration: Debug [Active]

Pre-define NAME (--define, -D)

```
ccs="ccs"
PART_TM4C1294NCPDT
ARM_MATH_CM4
__FPU_PRESENT=1
```

Figure 20. Adding the Pre-Processor Statements Necessary for Building an Example Project on a Cortex-M4 Part With Hardware FPU Support

8. Look for the *Add button to #include search path (--include_path, -I)* field in the *Include Options* section (see Figure 22).



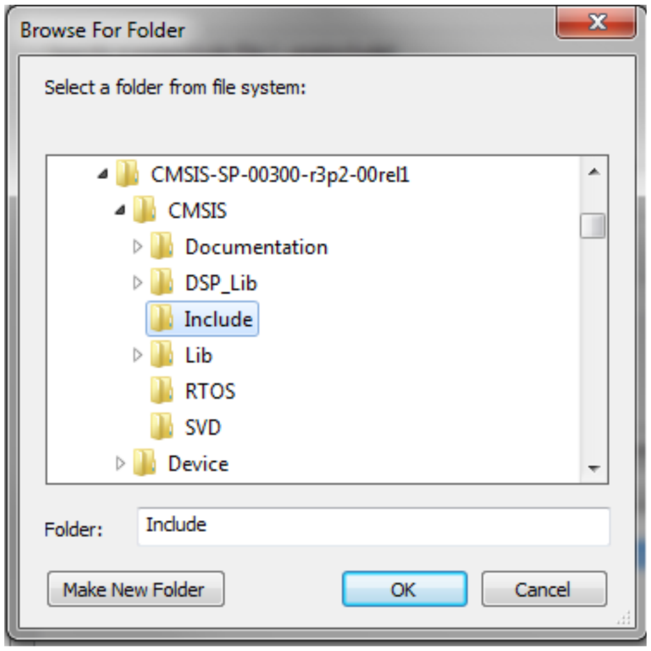
Configuration: Debug [Active]

Add dir to #include search path (--include_path, -I)
"${CG_TOOL_ROOT}/include"
"C:\CMSIS-SP-00300-r4p2-00rel0\CMSIS\Include"

Directory: "C:\CMSIS-SP-00300-r4p2-00rel0\CMSIS\Include"

Figure 21. Compiler's #include Search Path Modified to Contain Both the Base CMSIS Include Directory and the Example Projects' Common Include Directory

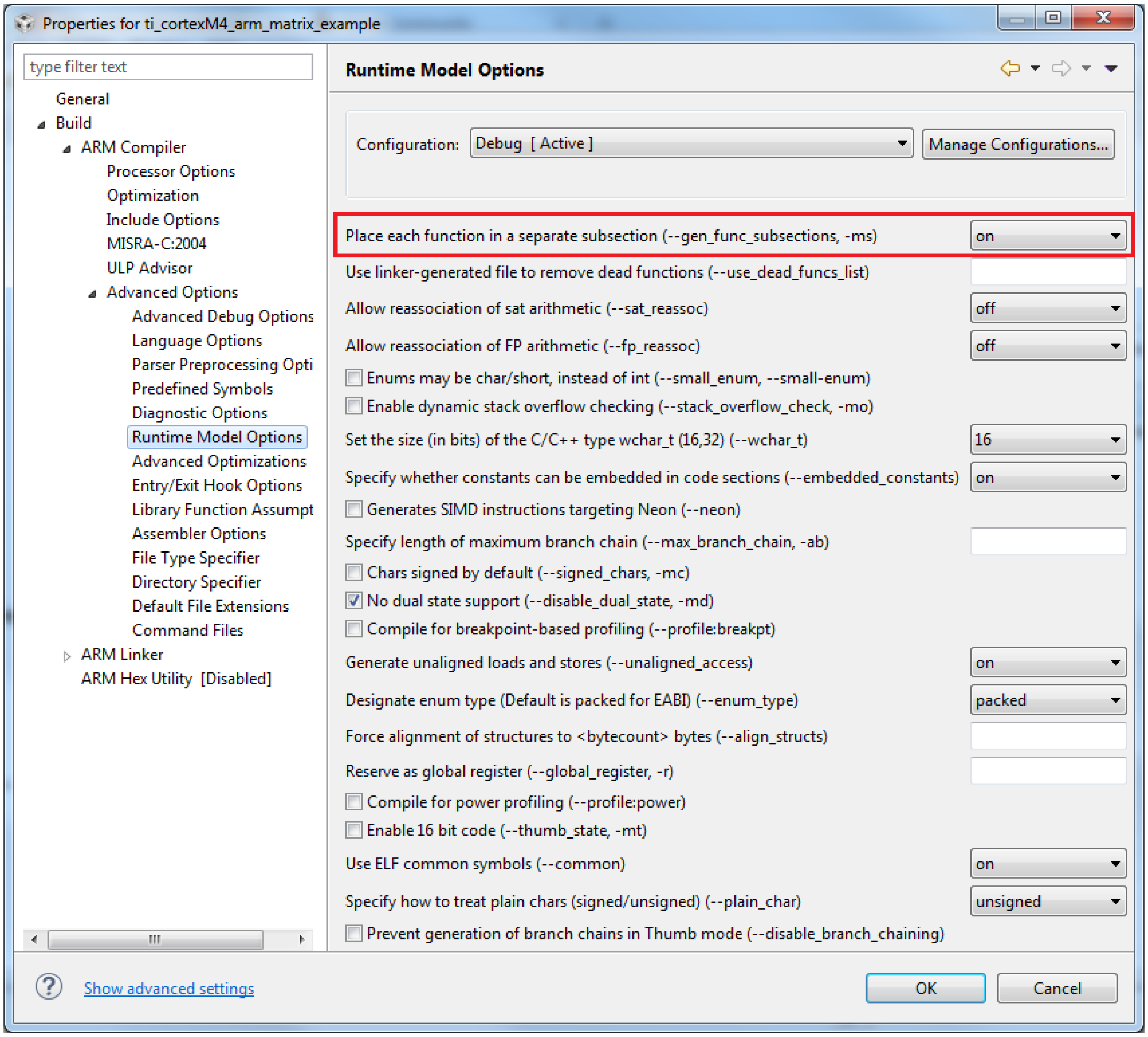
9. Click the *Add…* button again, then the *Browse* button and browse to the Include directory located in the CMSIS directory, then click **OK** (see Figure 21).



Folder: Include

Figure 22. Using the File System Option to Add the Base CMSIS Include Directory to the Compiler's #include Search Path

10. Select the *on* option from the *Place each function in a separate subsection (--gen_func_subsections, -ms)* drop-down menu (see Figure 23), in the Runtime Model Options panel.


Properties for ti_cortexM4_arm_matrix_example

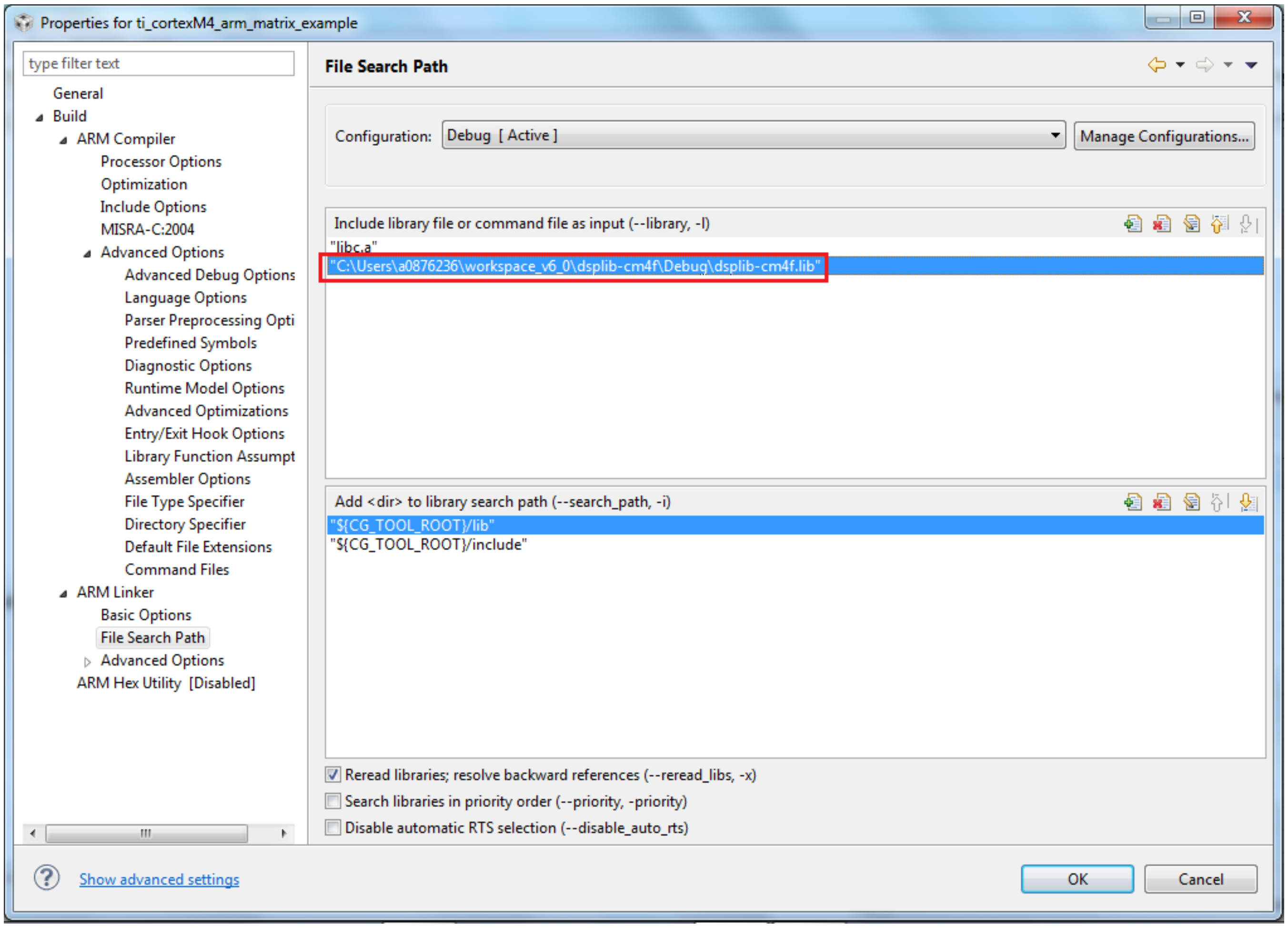
Runtime Model Options

Configuration: Debug [Active]

Place each function in a separate subsection (--gen_func_subsections, -ms)	on
Use linker-generated file to remove dead functions (--use_dead_funcs_list)	
Allow reassociation of sat arithmetic (--sat_reassoc)	off
Allow reassociation of FP arithmetic (--fp_reassoc)	off
☐ Enums may be char/short, instead of int (--small_enum, --small-enum)	
☐ Enable dynamic stack overflow checking (--stack_overflow_check, -mo)	
Set the size (in bits) of the C/C++ type wchar_t (16,32) (--wchar_t)	16
Specify whether constants can be embedded in code sections (--embedded_constants)	on
☐ Generates SIMD instructions targeting Neon (--neon)	
Specify length of maximum branch chain (--max_branch_chain, -ab)	
☐ Chars signed by default (--signed_chars, -mc)	
☑ No dual state support (--disable_dual_state, -md)	
☐ Compile for breakpoint-based profiling (--profile:breakpt)	
Generate unaligned loads and stores (--unaligned_access)	on
Designate enum type (Default is packed for EABI) (--enum_type)	packed
Force alignment of structures to <bytecount> bytes (--align_structs)	
Reserve as global register (--global_register, -r)	
☐ Compile for power profiling (--profile:power)	
☐ Enable 16 bit code (--thumb_state, -mt)	
Use ELF common symbols (--common)	on
Specify how to treat plain chars (signed/unsigned) (--plain_char)	unsigned
☐ Prevent generation of branch chains in Thumb mode (--disable_branch_chaining)	

Figure 23. The Runtime Model Options Set Up for Compiling the Example Projects

11. Open the *File Search Path* panel in the *ARM Linker* section.
12. Create an entry for the precompiled CMSIS DSP binary (.lib) that will be used in the *Include library file or command file as input (--library, -l)* area (see Figure 24). For this example, the library file created in section three will be used, so click on the *Add…* button, then the *File system…* button and navigate to the location of the .lib you want to use. If you built the precompiled binary from scratch as detailed in section 3 without changing the default project location, the .lib will be found at C:\Users\<user_name>\CCS workspaces\<your workspace>\dsplib-cm4f\Debug\dsplib-cm4f.lib. When you have found the binary, click **Open**, then **OK**.



Configuration: Debug [Active]

Include library file or command file as input (--library, -l)

"libc.a"
"C:\Users\a0876236\workspace_v6_0\dsplib-cm4f\Debug\dsplib-cm4f.lib"

Add <dir> to library search path (--search_path, -i)

"${CG_TOOL_ROOT}/lib"
"${CG_TOOL_ROOT}/include"

[✓] Reread libraries; resolve backward references (--reread_libs, -x)
[ ] Search libraries in priority order (--priority, -priority)
[ ] Disable automatic RTS selection (--disable_auto_rts)

Figure 24. The Linker's File Search Path Modified to Include the dsplib Binary Compiled in Section 3

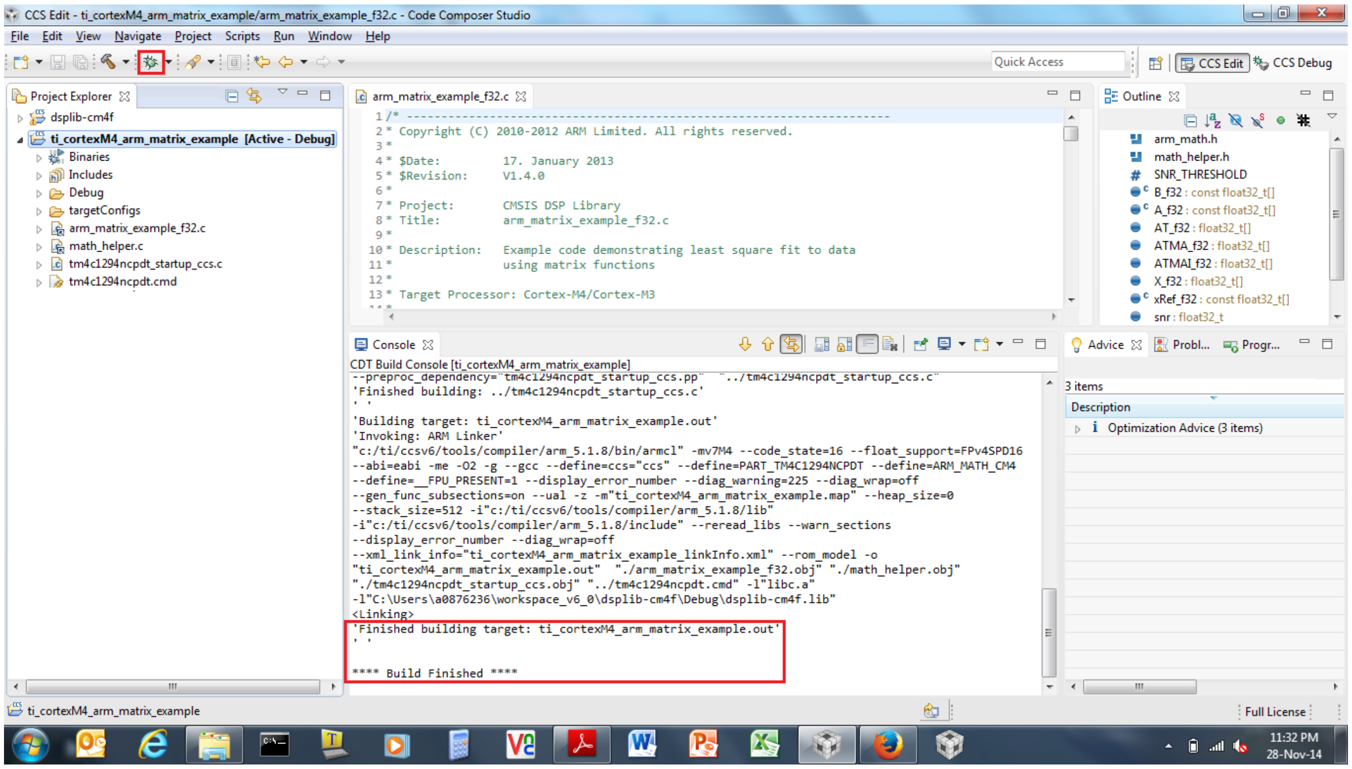
4.4 Building, Running, and Verifying the Project

Once the project has been created, the source code has been added to the work space, and the project properties have been properly configured, the project can be built by right clicking on it in the Project Explorer and selecting Build Project.

If this is the first time that Code Composer Studio is being used to connect to a target via the Stellaris In-Circuit Debug Interface, it might be necessary to install the proper drivers before it is possible to connect to the target to run code. Instructions for doing this can be found in the Code Composer Studiov6.1 Quick Start Guide, available at http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v6.

Once the code has been built and the proper drivers have been installed, you can run your code by using the following steps:

1. Press the *Debug* icon in the Code Composer Studio toolbar (see Figure 25).



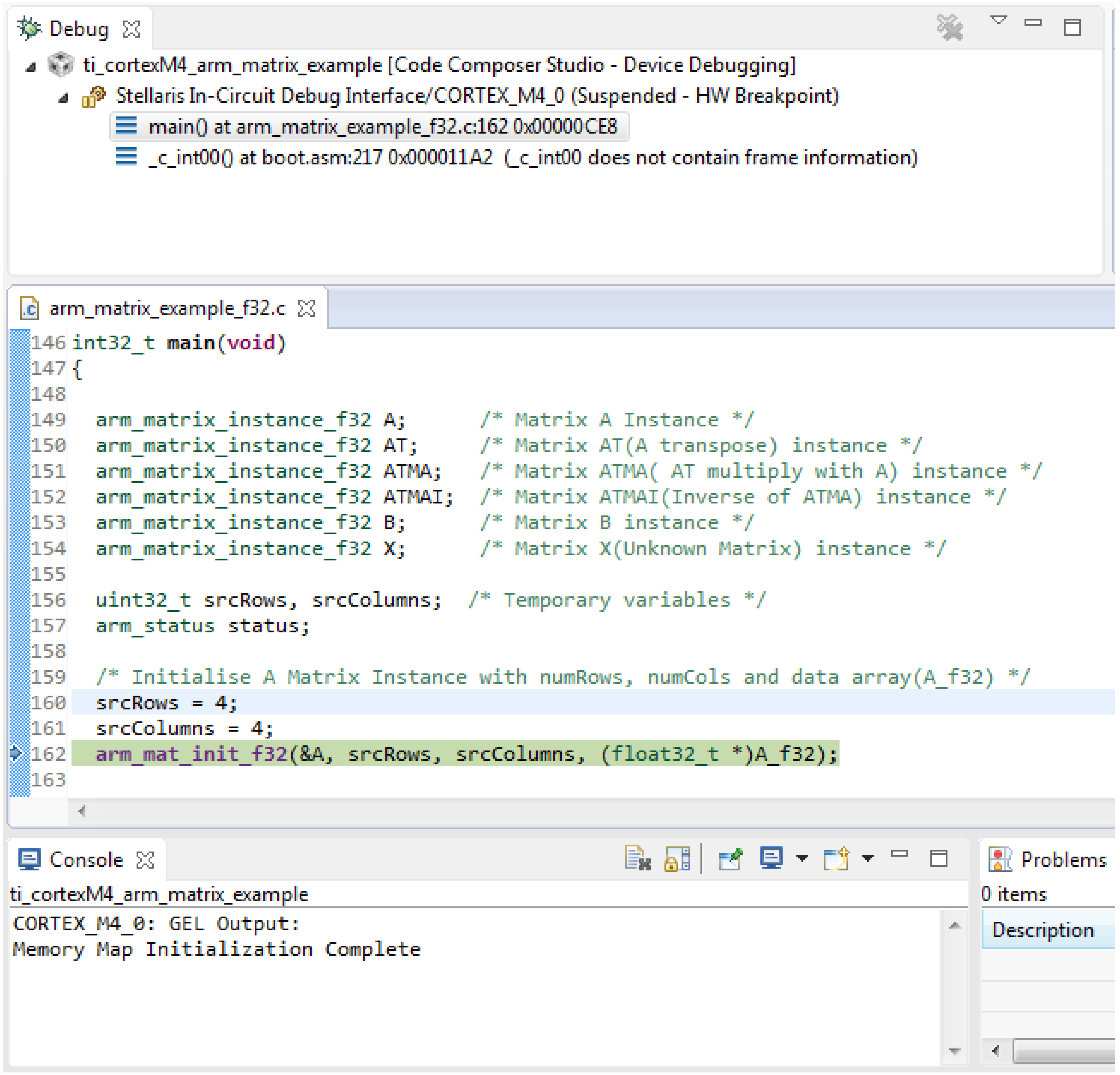
```
/* ----------------------------------------------------------------------
* Copyright (C) 2010-2012 ARM Limited. All rights reserved.
*
* $Date:        17. January 2013
* $Revision:    V1.4.0
*
* Project:      CMSIS DSP Library
* Title:        arm_matrix_example_f32.c
*
* Description:  Example code demonstrating least square fit to data
*               using matrix functions
*
* Target Processor: Cortex-M4/Cortex-M3
```

CDT Build Console [ti_cortexM4_arm_matrix_example]

```
--preproc_dependency="tm4c1294ncpdt_startup_ccs.pp"  "../tm4c1294ncpdt_startup_ccs.c"
'Finished building: ../tm4c1294ncpdt_startup_ccs.c'
' '
'Building target: ti_cortexM4_arm_matrix_example.out'
'Invoking: ARM Linker'
"c:/ti/ccsv6/tools/compiler/arm_5.1.8/bin/armcl" -mv7M4 --code_state=16 --float_support=FPv4SPD16
--abi=eabi -me -O2 -g --gcc --define=ccs="ccs" --define=PART_TM4C1294NCPDT --define=ARM_MATH_CM4
--define=__FPU_PRESENT=1 --display_error_number --diag_warning=225 --diag_wrap=off
--gen_func_subsections=on --ual -z -m"ti_cortexM4_arm_matrix_example.map" --heap_size=0
--stack_size=512 -i"c:/ti/ccsv6/tools/compiler/arm_5.1.8/lib"
-i"c:/ti/ccsv6/tools/compiler/arm_5.1.8/include" --reread_libs --warn_sections
--display_error_number --diag_wrap=off
--xml_link_info="ti_cortexM4_arm_matrix_example_linkInfo.xml" --rom_model -o
"ti_cortexM4_arm_matrix_example.out"  "./arm_matrix_example_f32.obj" "./math_helper.obj"
"./tm4c1294ncpdt_startup_ccs.obj" "../tm4c1294ncpdt.cmd" -l"libc.a"
-l"C:\Users\a0876236\workspace_v6_0\dsplib-cm4f\Debug\dsplib-cm4f.lib"
<Linking>
'Finished building target: ti_cortexM4_arm_matrix_example.out'
' '

**** Build Finished ****
```

Figure 25. The Debug Context Being Displayed After the arm_matrix_example Project has Been Set Up for Debugging

2. It takes a moment for Code Composer Studio to connect to the MCU and download the code. Once the connection has been established and the flash programmed with the compiled project code, the MCU will run until it reaches the project's *main()* function (see Figure 26). Press the *Resume* button (or F8) to cause the program to start executing.



```
ti_cortexM4_arm_matrix_example [Code Composer Studio - Device Debugging]
  Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Suspended - HW Breakpoint)
    main() at arm_matrix_example_f32.c:162 0x00000CE8
    _c_int00() at boot.asm:217 0x000011A2  (_c_int00 does not contain frame information)
```

arm_matrix_example_f32.c

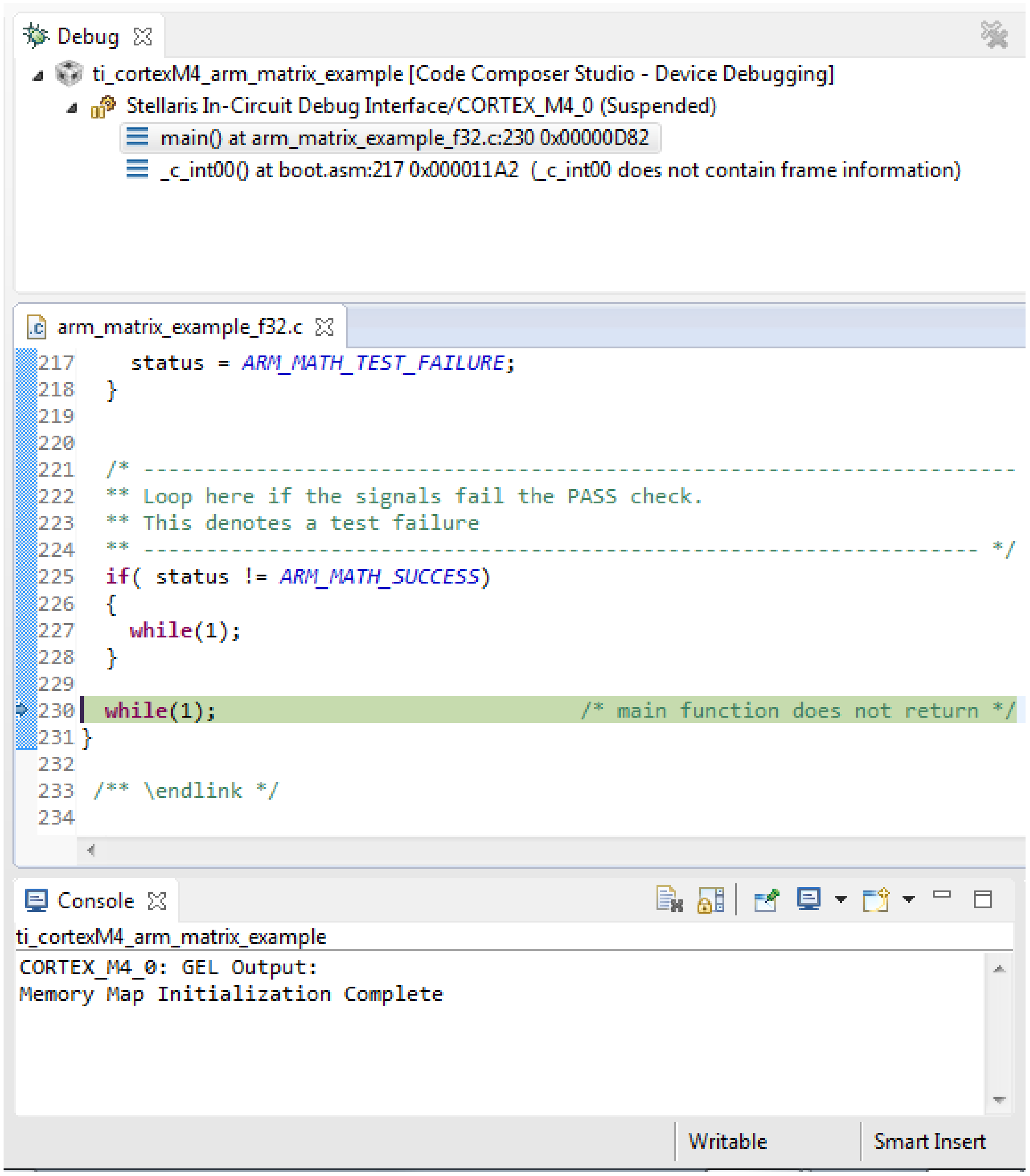
```
int32_t main(void)
{

  arm_matrix_instance_f32 A;      /* Matrix A Instance */
  arm_matrix_instance_f32 AT;     /* Matrix AT(A transpose) instance */
  arm_matrix_instance_f32 ATMA;   /* Matrix ATMA( AT multiply with A) instance */
  arm_matrix_instance_f32 ATMAI;  /* Matrix ATMAI(Inverse of ATMA) instance */
  arm_matrix_instance_f32 B;      /* Matrix B instance */
  arm_matrix_instance_f32 X;      /* Matrix X(Unknown Matrix) instance */

  uint32_t srcRows, srcColumns;  /* Temporary variables */
  arm_status status;

  /* Initialise A Matrix Instance with numRows, numCols and data array(A_f32) */
  srcRows = 4;
  srcColumns = 4;
  arm_mat_init_f32(&A, srcRows, srcColumns, (float32_t *)A_f32);

```

ti_cortexM4_arm_matrix_example

```
CORTEX_M4_0: GEL Output:
Memory Map Initialization Complete
```

Figure 26. The arm_matrix_example Project, After it has Been Loaded Into Flash and the Startup Code has run to the main() Function

3. After a few seconds have passed, the program will run to completion (see Figure 27). Press the suspend button, which will halt the processor and show you what line of code is being executed.



```
ti_cortexM4_arm_matrix_example [Code Composer Studio - Device Debugging]
  Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Suspended)
    main() at arm_matrix_example_f32.c:230 0x00000D82
    _c_int00() at boot.asm:217 0x000011A2  (_c_int00 does not contain frame information)
```

arm_matrix_example_f32.c

```
    status = ARM_MATH_TEST_FAILURE;
  }


  /* ----------------------------------------------------------------------
  ** Loop here if the signals fail the PASS check.
  ** This denotes a test failure
  ** ------------------------------------------------------------------- */
  if( status != ARM_MATH_SUCCESS)
  {
    while(1);
  }

  while(1);                                 /* main function does not return */
}

/** \endlink */

```

```
ti_cortexM4_arm_matrix_example
CORTEX_M4_0: GEL Output:
Memory Map Initialization Complete
```

Figure 27. The arm_matrix_example Project Having Run to Successful Completion

4. For every function other than the class marks example, the program will have halted in one of two while loops. If the program did not successfully execute, it will be caught in a while loop surrounded by an if statement with a test condition of (status != ARM_MATH_SUCCESS). If the program did successfully execute, it is caught in a while loop found immediately after the previously mentioned if statement. For the class marks example, there is no built in method by which the microcontroller's execution state can be verified.

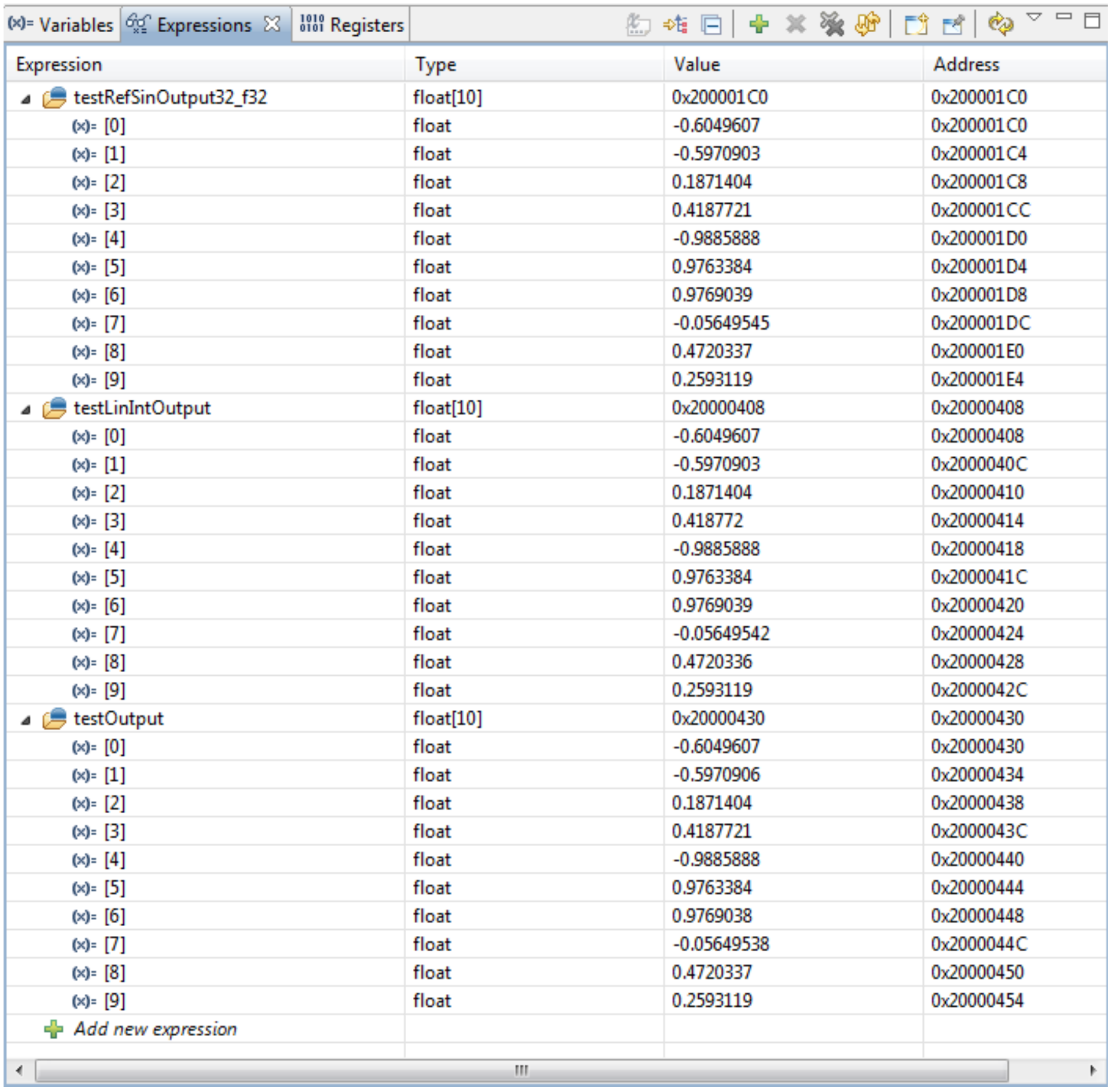
4.5 Source Code Modifications

For almost all of the ARM example projects, the above steps can be followed in a similar manner to build and run the ARM-provided source code. There is one project, though, that require modifications to the source code to properly build and run on the TM4C123G Launchpad.

The linear interpolation example contains a table of values meant to represent a waveform of sin(x) as x goes from negative pi to 2*pi by increments of 0.00005. This granularity causes the resulting compiled binary to be too large in size for the TM4C series launchpad. An alternate data file, ti_linear_interp_data_37968.c, has been provided along with this application report that represents the same array given increments of 0.00025 instead. This causes the compiled binary to be small enough to fit into a part with a flash size of 256 kB and an SRAM size of 32 kB for the TM4C123 Platform devices. This necessitates a change in the linear interpolation example code as well (as the size and name of the statically allocated array has been changed), so when adding the source code for this example, it is necessary to use the ti_linear_interp_example_f32.c file included with this application report

The linear interpolation example also contains a bug that might cause it to give the appearance of failing when executing. The purpose of the example is to show the difference in accuracy that can be achieved by using the CMSIS DSP library's linear interpolation sin function, which uses both cubic interpolation and linear interpolation to derive its return values, and the library's standard sin function, which uses only cubic interpolation. The method that is used to compare the accuracy of these two functions is to calculate the signal-to-noise ratio of both signals with respect to a pre-calculated signal that is known to be correct. Unfortunately, the method of using linear interpolation gives a result that almost exactly matches the pre-calculated signal, which causes the SNR function to attempt to take the log of a value divided by 0. As such, the function's self-test method cannot be assumed trustworthy. The user should instead use the debugger to verify that the 10-element-long arrays representing the sin values are indeed more accurate when using the linear interpolation functions than when using the standard functions. This can be done using the following steps:

1. Select the *Expression* view in the Code Composer Studio debugger context
2. Click *Add new expression*, and type in testRefSinOutput32_f32. This will add the array containing the pre-calculated reference sin output to the expressions list.
3. Click the arrow to the left of testRefSinOutput32_f32 to display all elements of the array.
4. Click *Add new expressions*, and type testOutput. This will add the array containing the sin values as calculated by the CMSIS DSP_Lib sin function that uses cubic interpolation to the expression list.
5. Click the arrow to the left of testOutput to display all elements of the array.
6. Click *Add new expressions*, and type testLinIntOutput. This will add the array containing the sin values as calculated by the CMSIS DSPlib that uses both cubic and linear interpolation sin function to the expression list.
7. Click the arrow to the left of testLinIntOutput to display all elements of the array (see Figure 28).



Expression	Type	Value	Address
testRefSinOutput32_f32	float[10]	0x200001C0	0x200001C0
[0]	float	-0.6049607	0x200001C0
[1]	float	-0.5970903	0x200001C4
[2]	float	0.1871404	0x200001C8
[3]	float	0.4187721	0x200001CC
[4]	float	-0.9885888	0x200001D0
[5]	float	0.9763384	0x200001D4
[6]	float	0.9769039	0x200001D8
[7]	float	-0.05649545	0x200001DC
[8]	float	0.4720337	0x200001E0
[9]	float	0.2593119	0x200001E4
testLinIntOutput	float[10]	0x20000408	0x20000408
[0]	float	-0.6049607	0x20000408
[1]	float	-0.5970903	0x2000040C
[2]	float	0.1871404	0x20000410
[3]	float	0.418772	0x20000414
[4]	float	-0.9885888	0x20000418
[5]	float	0.9763384	0x2000041C
[6]	float	0.9769039	0x20000420
[7]	float	-0.05649542	0x20000424
[8]	float	0.4720336	0x20000428
[9]	float	0.2593119	0x2000042C
testOutput	float[10]	0x20000430	0x20000430
[0]	float	-0.6049607	0x20000430
[1]	float	-0.5970906	0x20000434
[2]	float	0.1871404	0x20000438
[3]	float	0.4187721	0x2000043C
[4]	float	-0.9885888	0x20000440
[5]	float	0.9763384	0x20000444
[6]	float	0.9769038	0x20000448
[7]	float	-0.05649538	0x2000044C
[8]	float	0.4720337	0x20000450
[9]	float	0.2593119	0x20000454
Add new expression			

Figure 28. Using the Debugger to Examine the Results of the linear_interp_example Project

8. If you manually examine the values stored at each element, you will see that for the most part, the sin values calculated using both cubic and linear interpolation are closer to the reference values than those calculated using only cubic interpolation. In the example above, this is especially noticeable on element 7 of the output arrays.

5 Conclusion

Using the information provided in this document, combined with the resources available from ARM’s CMSIS website, it is possible to easily and quickly implement various complex DSP algorithms. While it is possible to code a number of these functions independently, the result would likely lead to a much greater development time and produce less efficient code. It is highly recommended that anytime a Texas Instruments’ TM4C or Stellaris microcontroller is being used for an application that requires complex DSP functionality, the procedure listed here should be followed to ensure accurate, reliable, efficient code.

6 References

The following related documents and software are available on the TM4C Series web site at: http://www.ti.com/product/tm4c1294ncpdt

- *Tiva TM4C1294NCPDT Microcontroller Data Sheet* (SPMS433)
- *Tiva C Series TM4C129x Microcontrollers Silicon Revisions 1, 2, and 3 Silicon Errata* (SPMZ850)
- The source code for the CMSIS DSP Library and example code can be downloaded from ARM’s CMSIS website: cmsis.arm.com.
- A quick start guide for using Texas Instruments’ Code Composer Studio v6.1 can be found on the TI processor wiki at: http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v6.

Revision History

Changes from F Revision (May 2015) to G Revision **Page**

- Updates were made in the Abstract.. 1
- Information was updated in Section 3.1. 2
- Information was updated in Section 3.2. 2
- Information was updated in Section 3.4. 6
- Information was updated in Section 4.1....................................... 14
- Information was updated in Section 4.3....................................... 19

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		