

# 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 .....  | 1  |
| 2 | CMSIS DSP Library .....                                     | 1  |
| 3 | Building the DSP Library in Code Composer Studio v6.1 ..... | 3  |
| 4 | ARM Example Projects .....                                  | 14 |
| 5 | Conclusion .....  | 32 |
| 6 | References .....  | 33 |

## Trademarks

Code Composer Studio, TivaWare are trademarks of Texas Instruments.  
Stellaris is a registered trademark of Texas Instruments.  
ARM, Cortex are registered trademarks of ARM Limited.  
All other trademarks are the property of their respective owners.

## 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, glib, 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`.

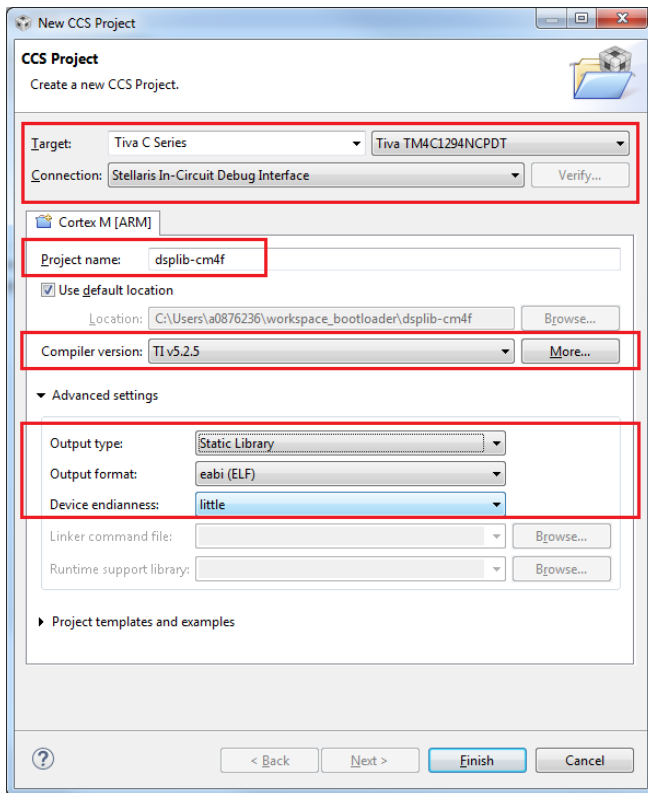


Figure 1. Creating the dsplib Project

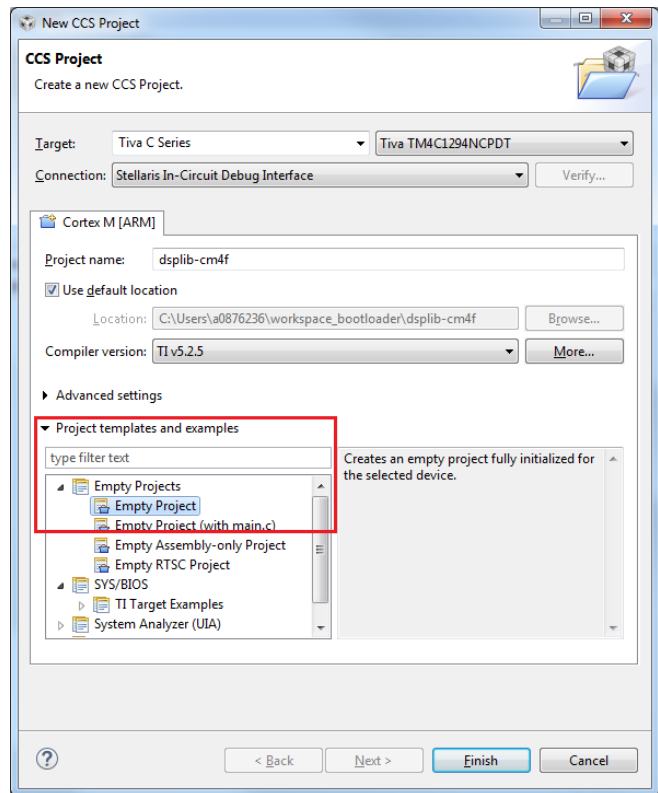


Figure 2. Creating the dsplib Project

### 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.

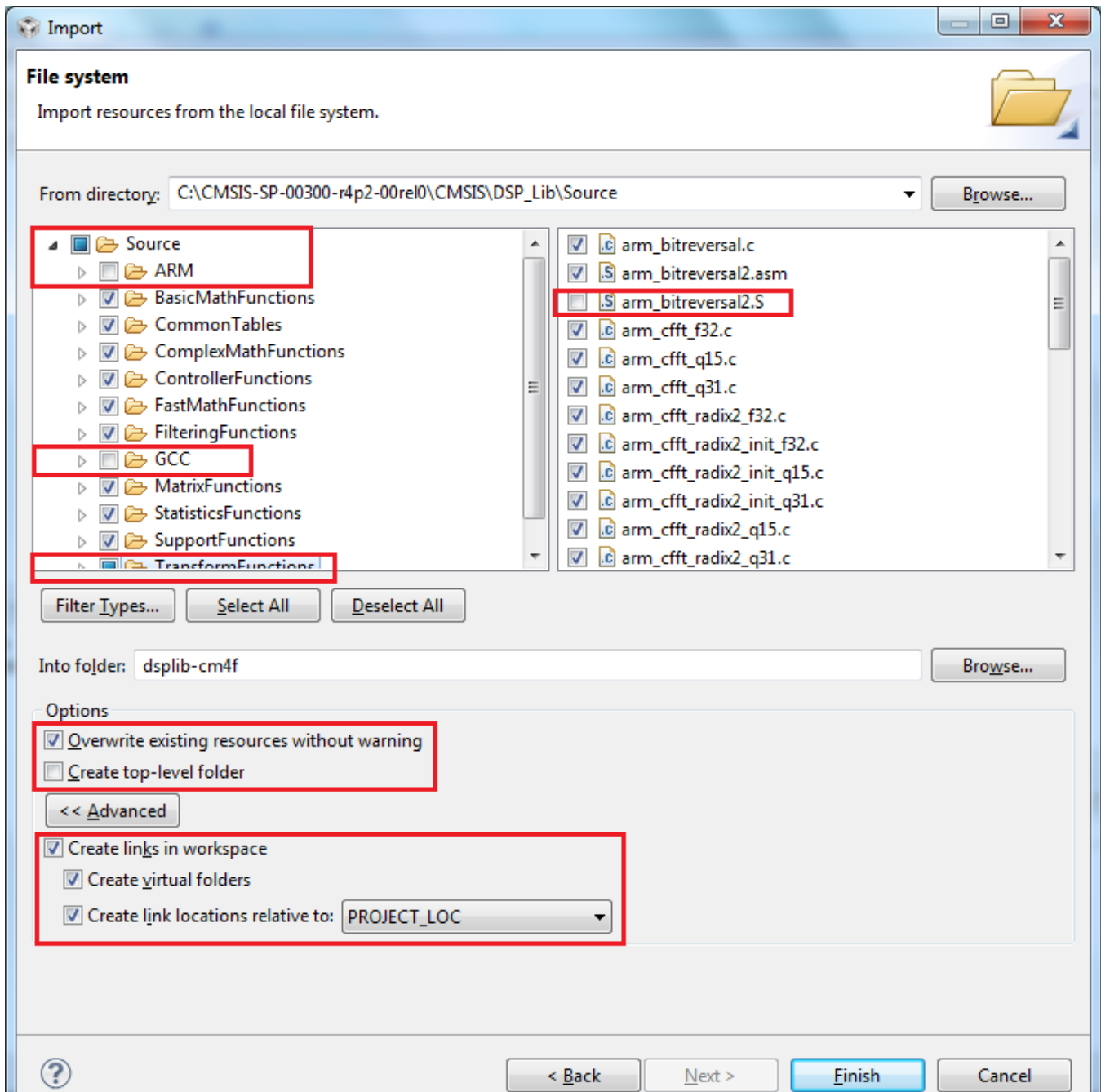
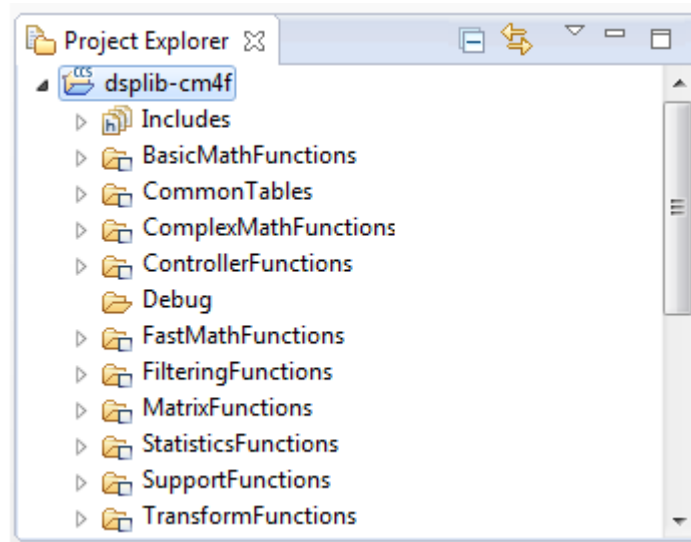


Figure 3. Importing the DSP\_Lib Source Code

- 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:

- Right-click the dsplib-cm4f project in the Project Explorer and select *Properties*.
- Expand the *Build* entry, and then expand the *ARM Compiler* entry.
- 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).

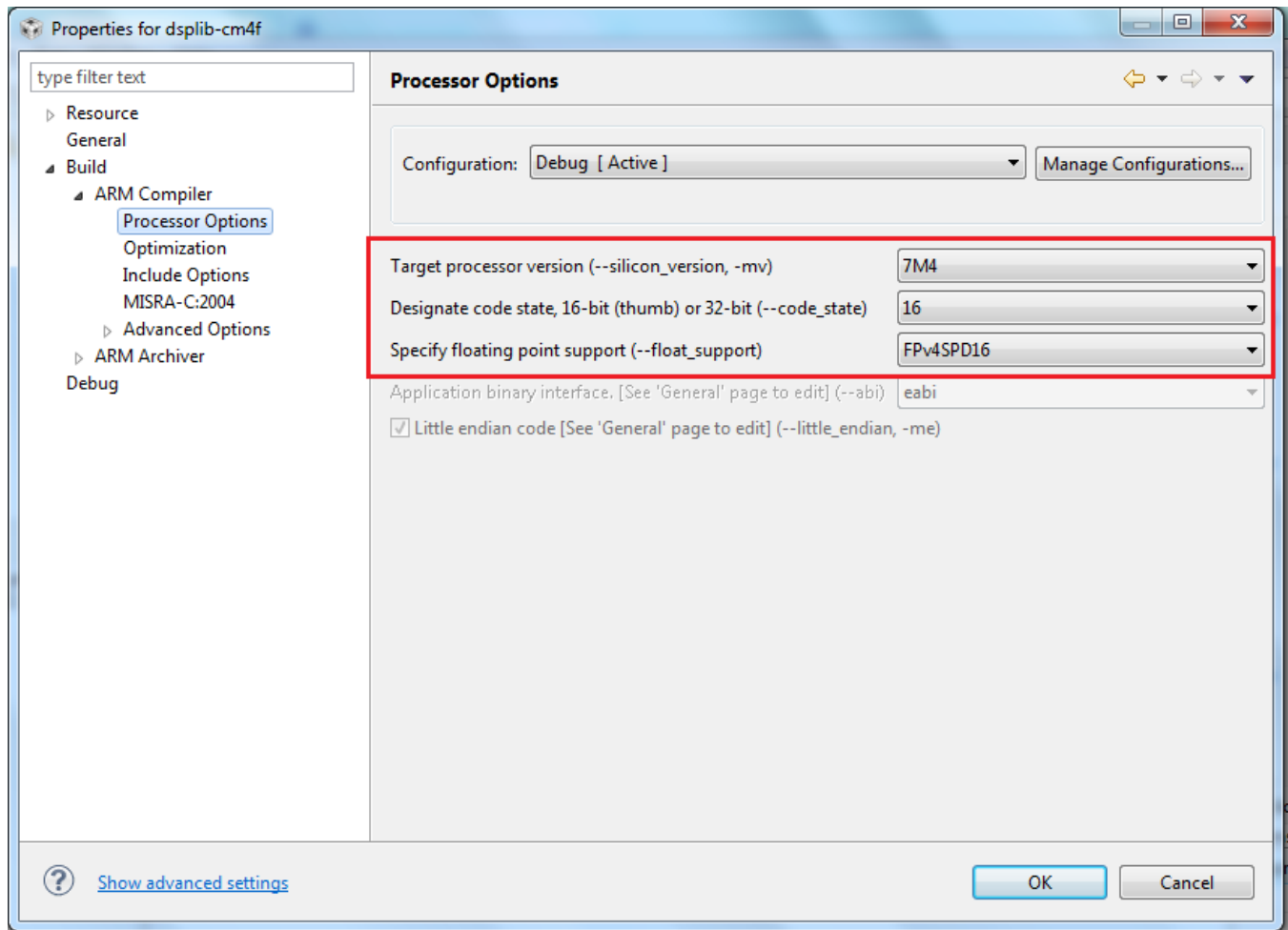
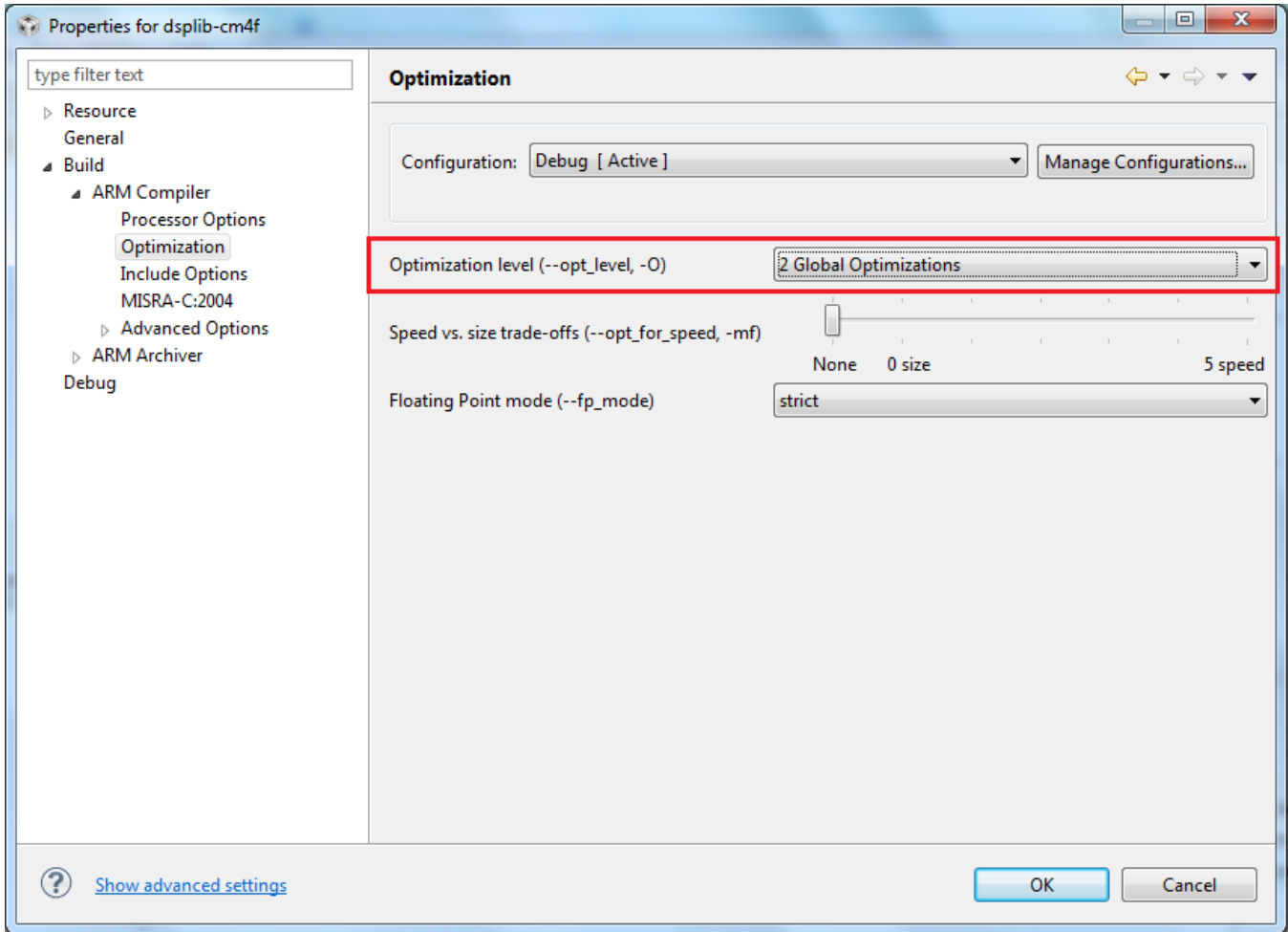


Figure 5. The Processor Settings for a Cortex-M4 Processor With Hardware FPU Support



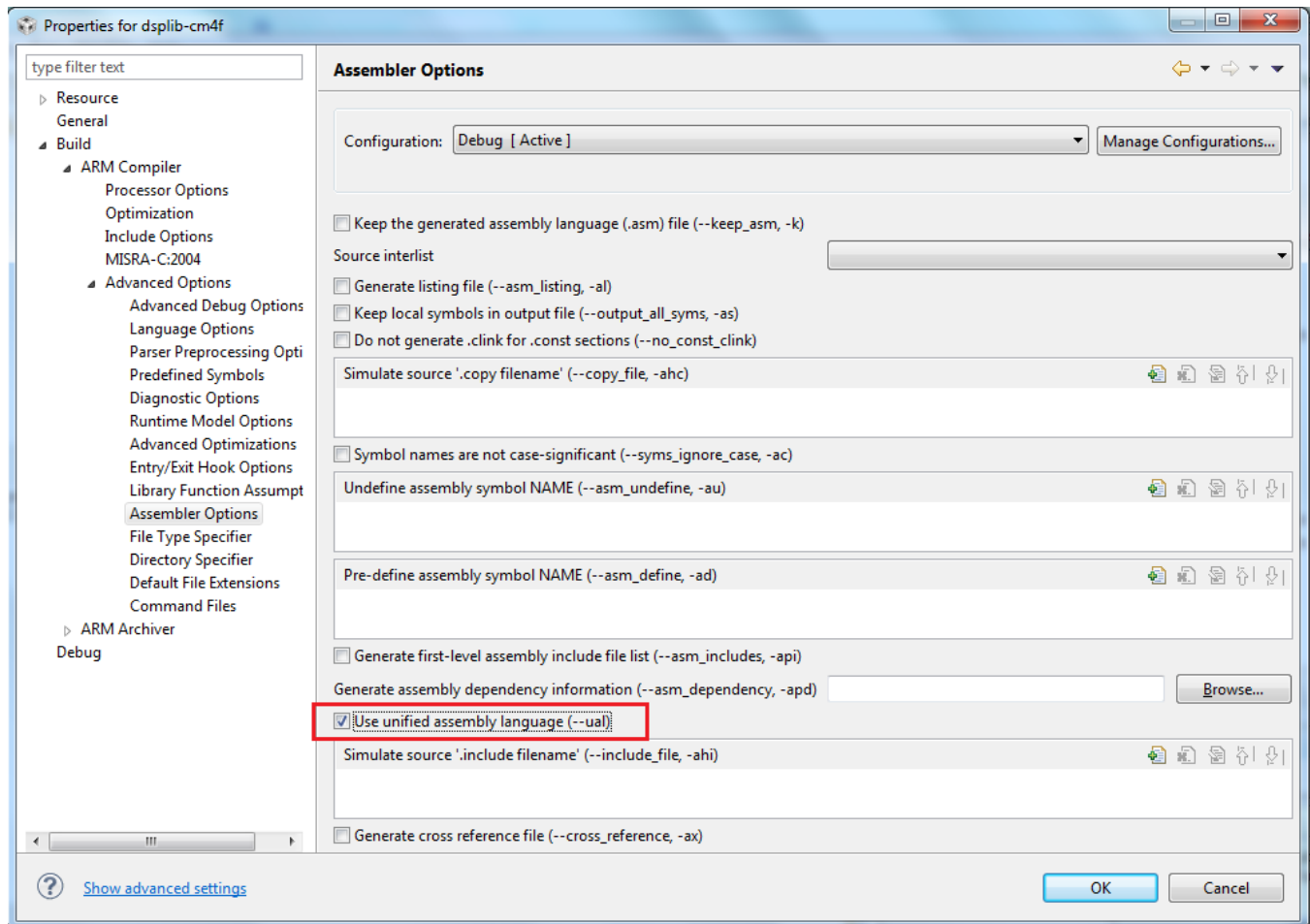
4. Click the *Optimization level* (`--opt_level`, `-O`) drop-down menu in the Optimization panel and select 2 (see Figure 6).



**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](#)).



**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.

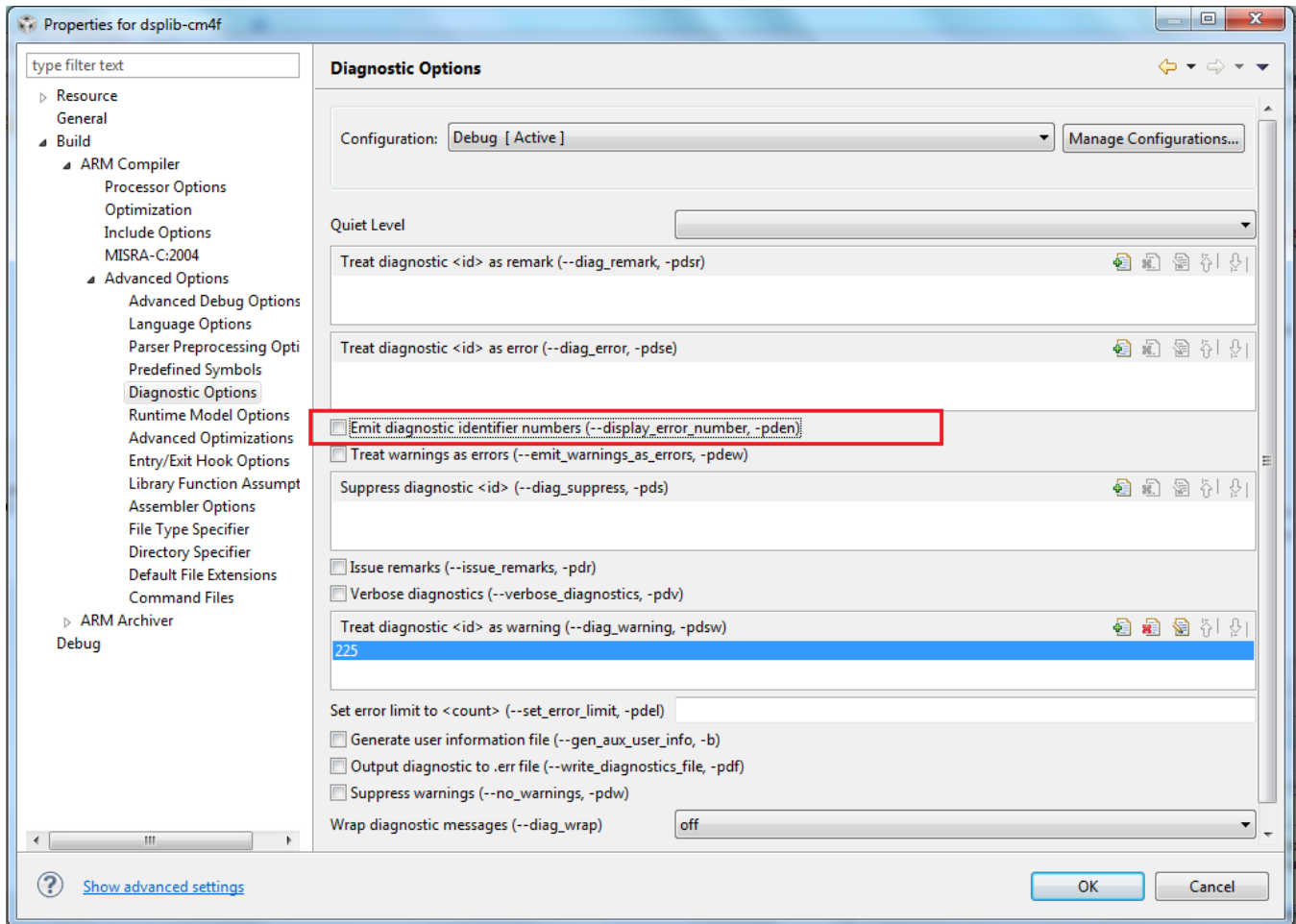
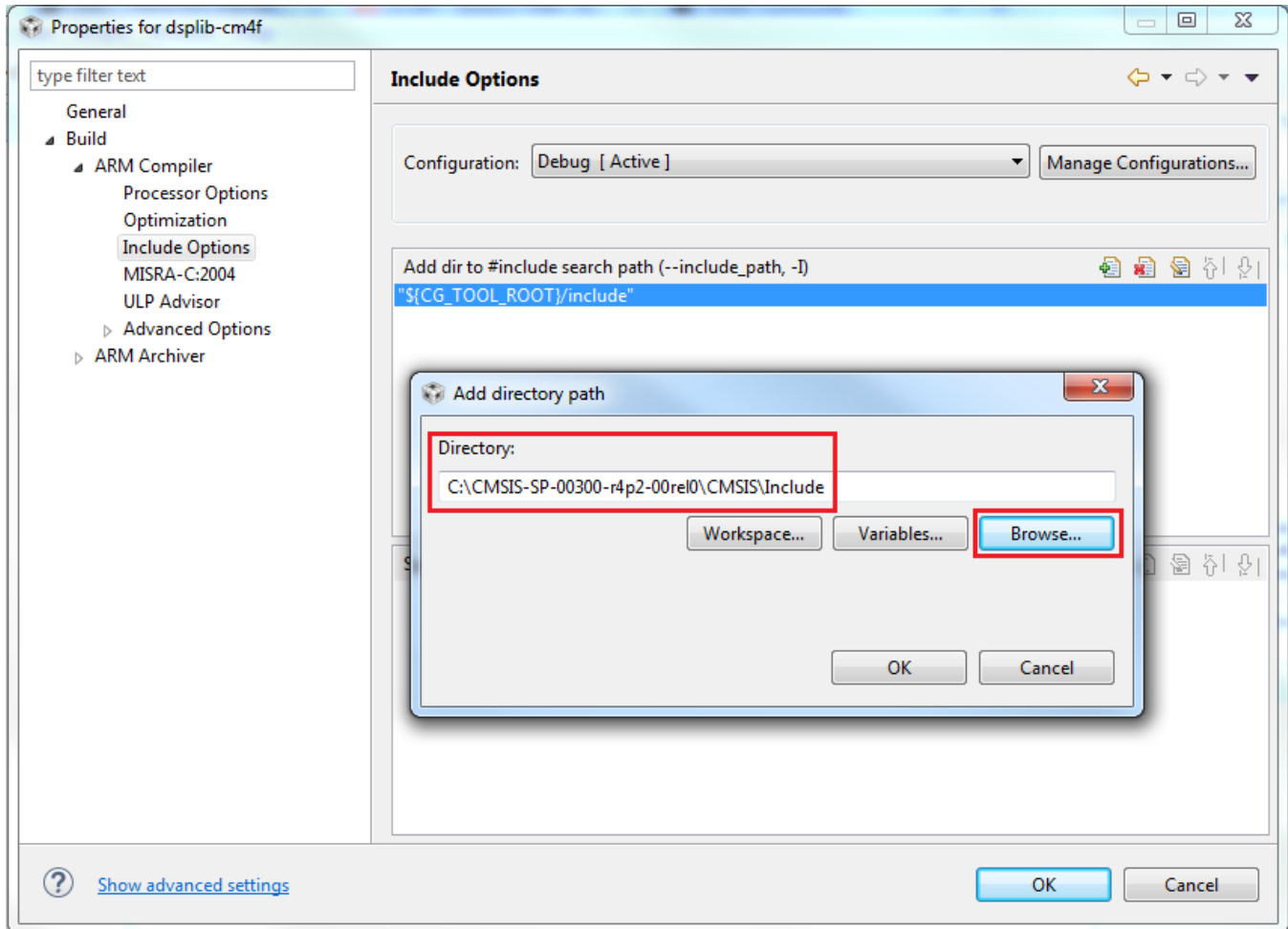


Figure 8. Verifying That Diagnostic Identifier Numbers will not be Emitted

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 (CMSIS-<version>/CMSIS/Include).



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

- 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**.

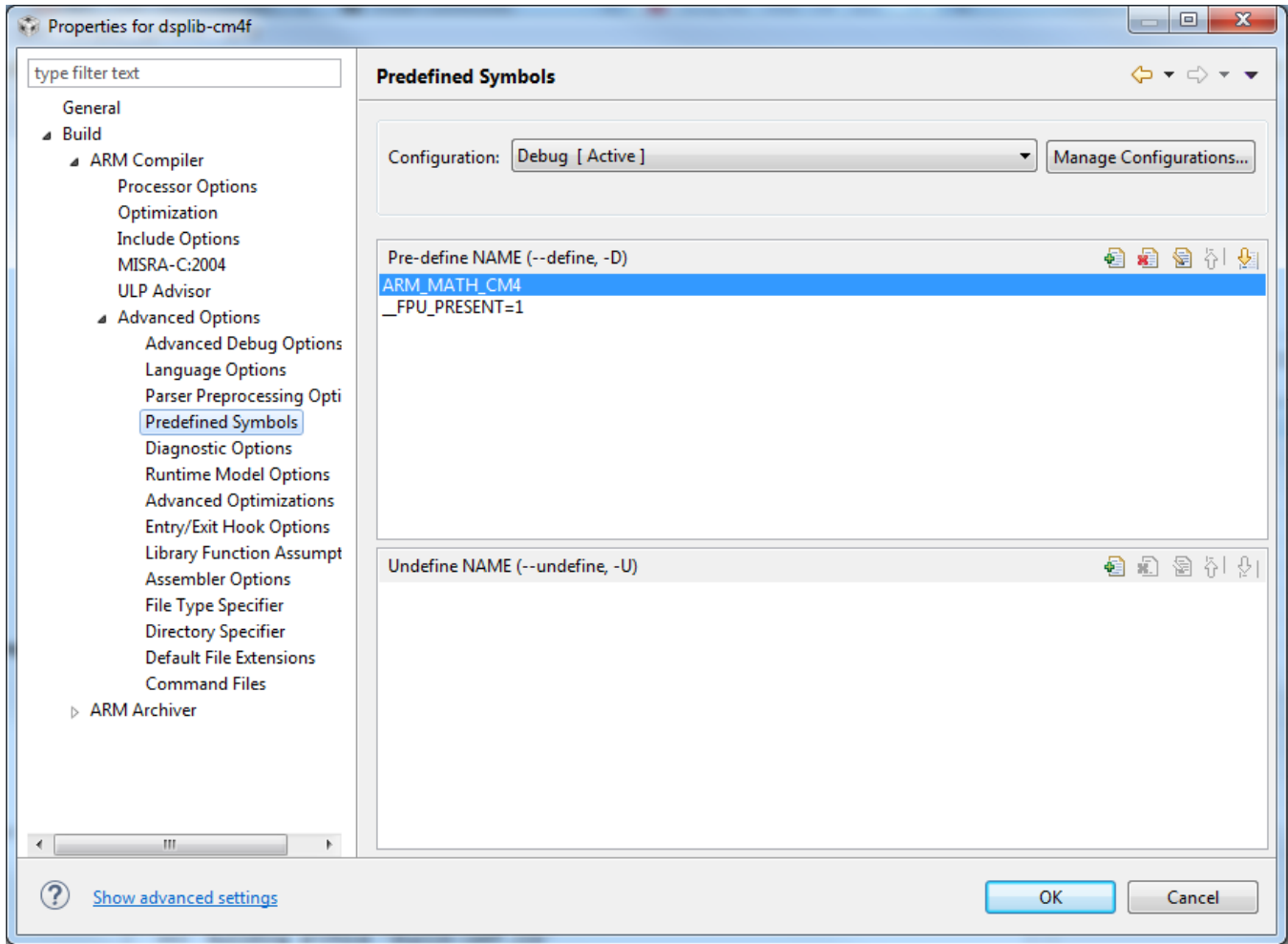


Figure 10. Adding Project Level #defines for the Processor Characteristics

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).

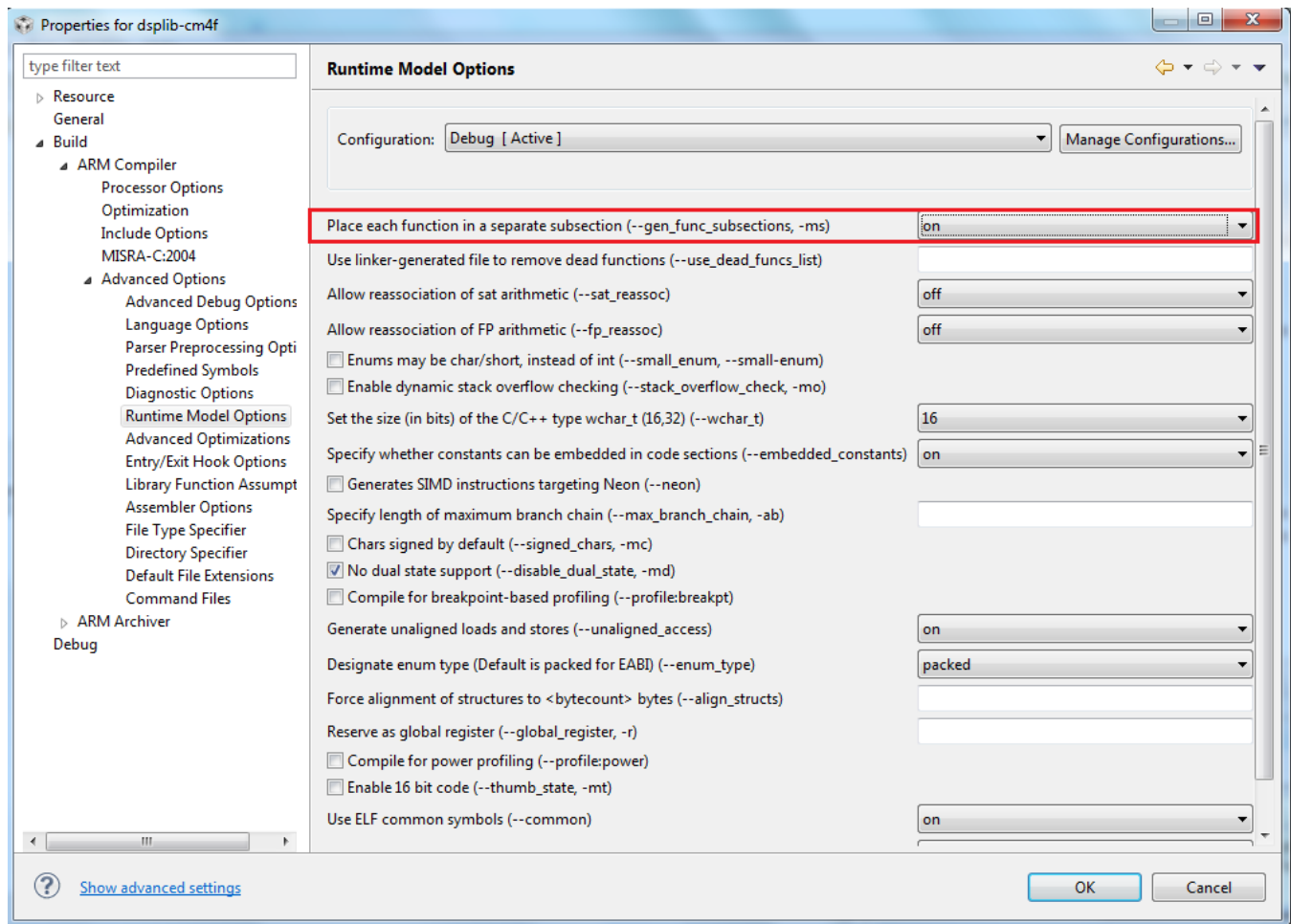


Figure 11. The Proper Runtime Model Options for Compiling the DSP\_Lib Source Code

### 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.

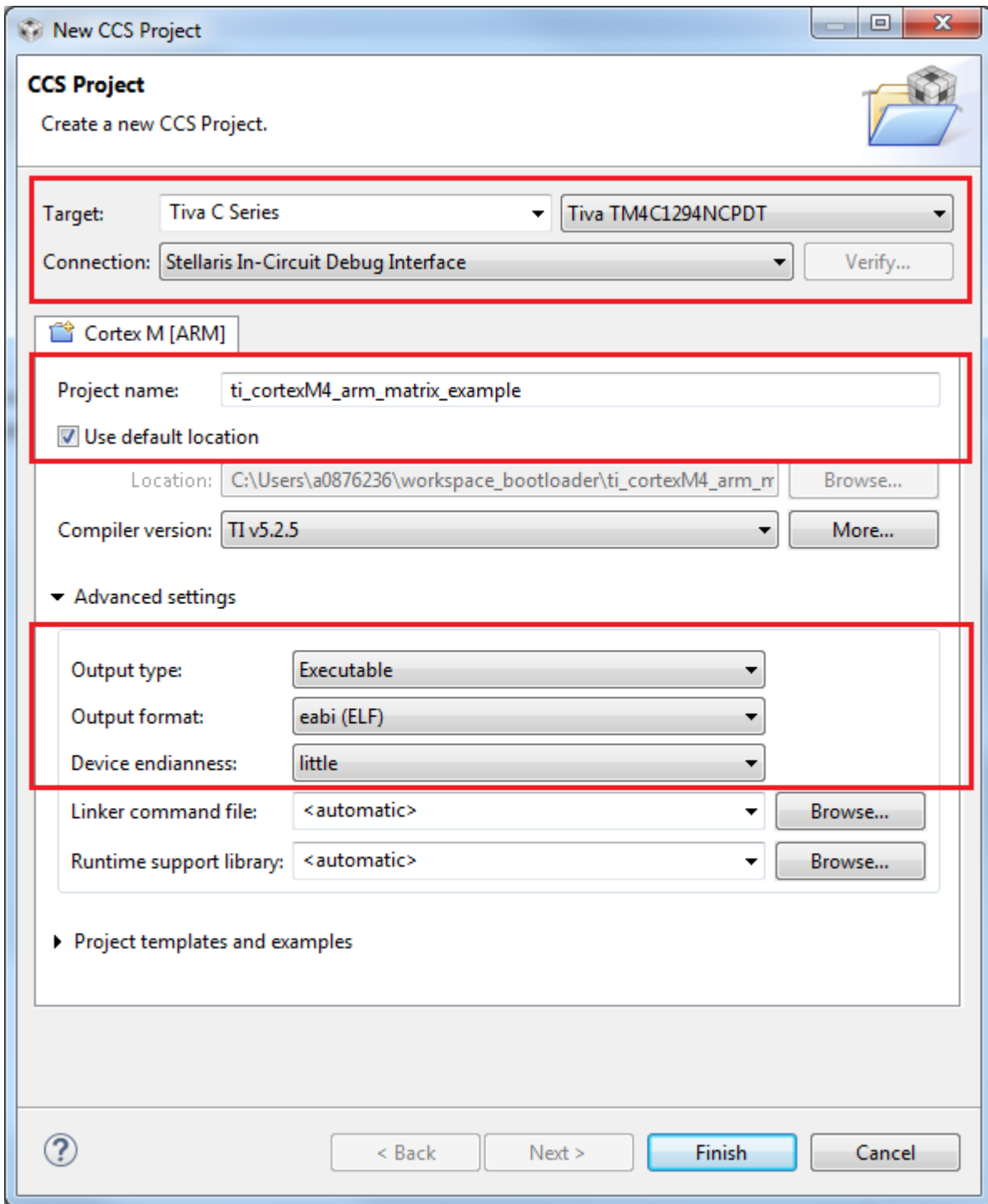


Figure 12. The New CCS Project Window With Options Set to Build the arm\_dotproduct\_example Project



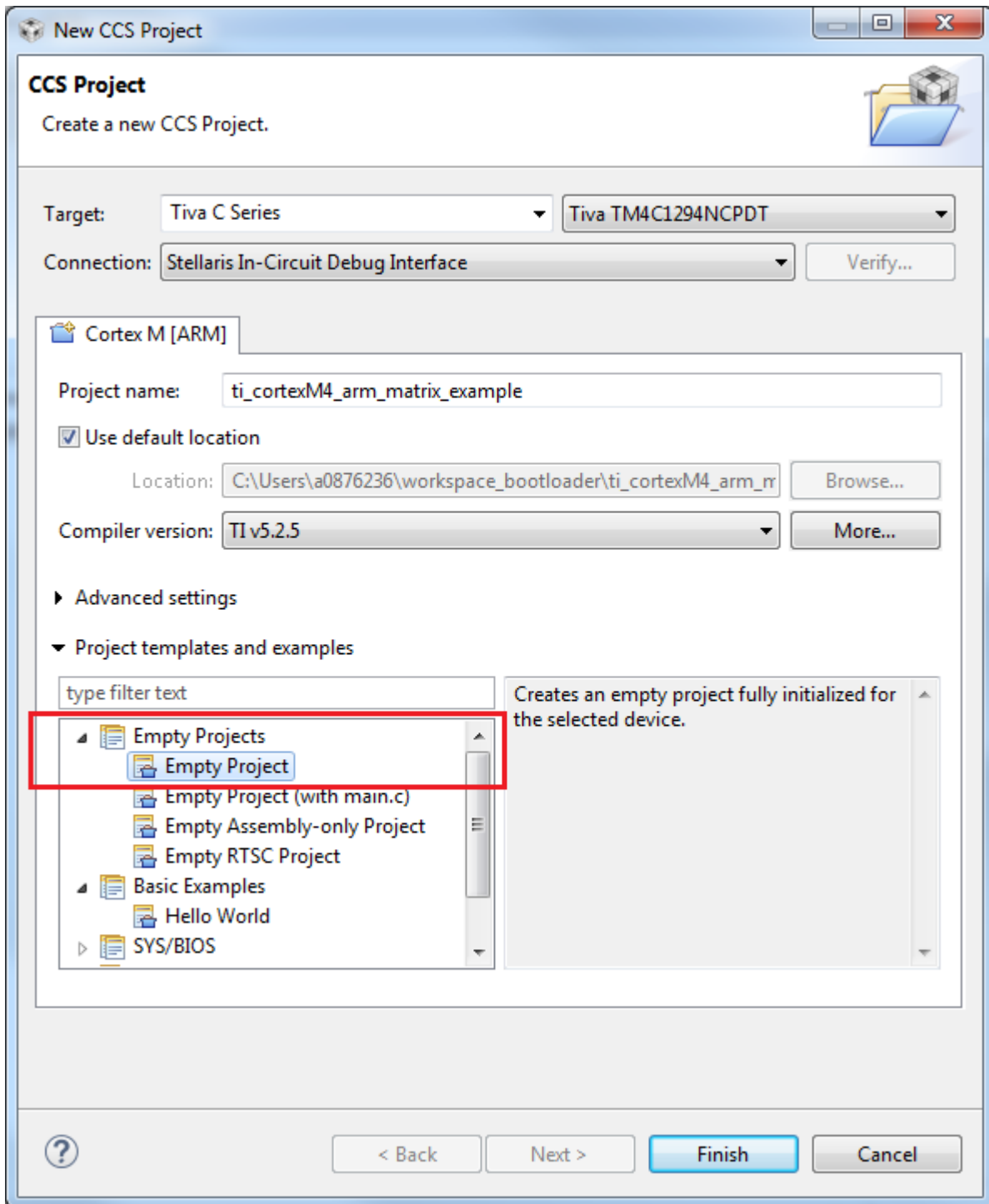


Figure 13. The New CCS Project Window With Options Set to Build the arm\_dotproduct\_example Project

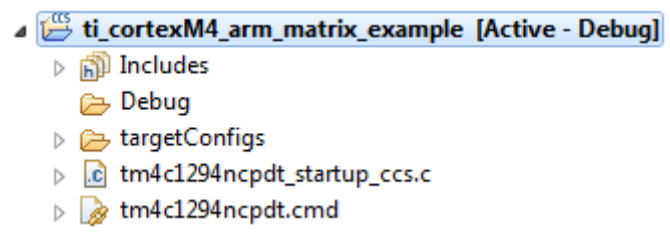


Figure 14. The Project Explorer After the arm\_matrix\_example Project has Been Created

## 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).

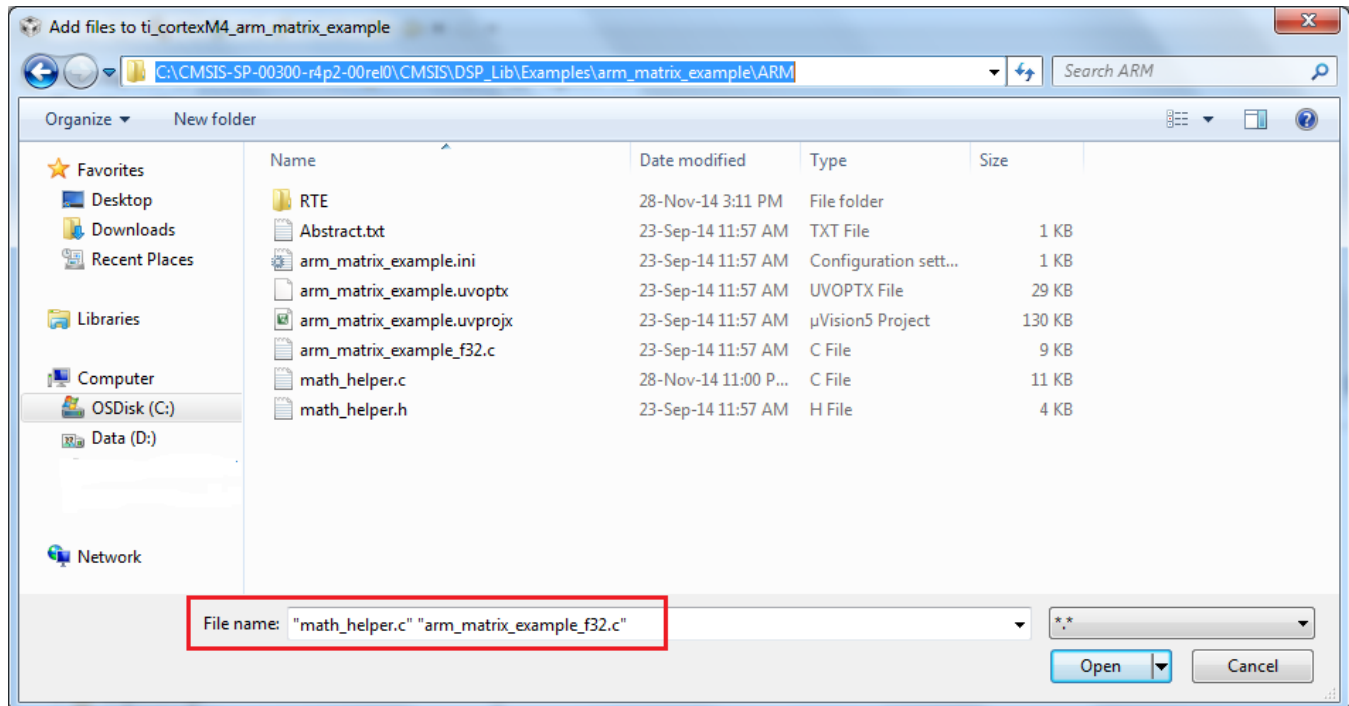
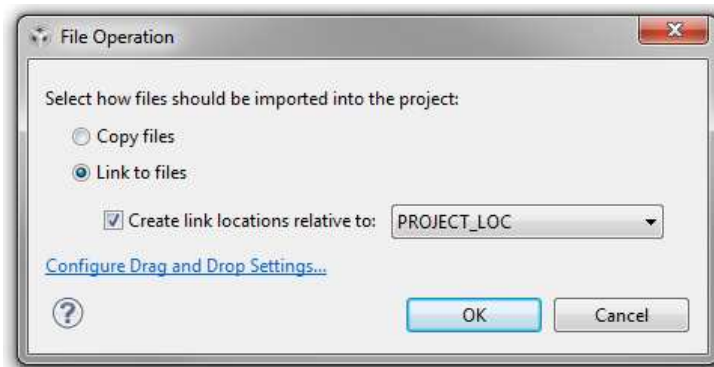


Figure 15. Adding the Source Files for arm\_matrix\_example to the Project

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.

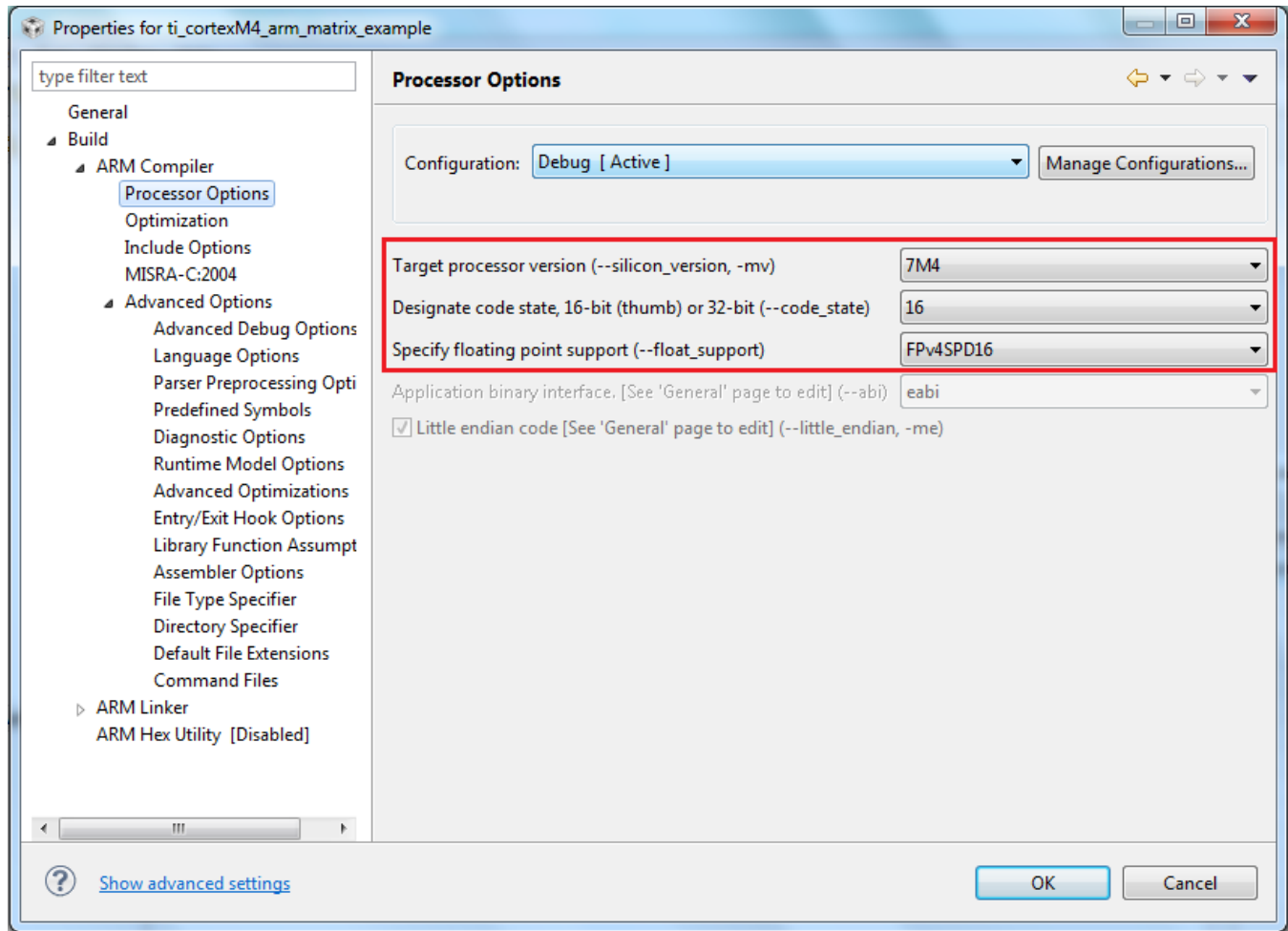


**Figure 16. Selecting the Proper Options to Link the Source Files Into the Project**

### 4.3 Editing the Example Project Settings

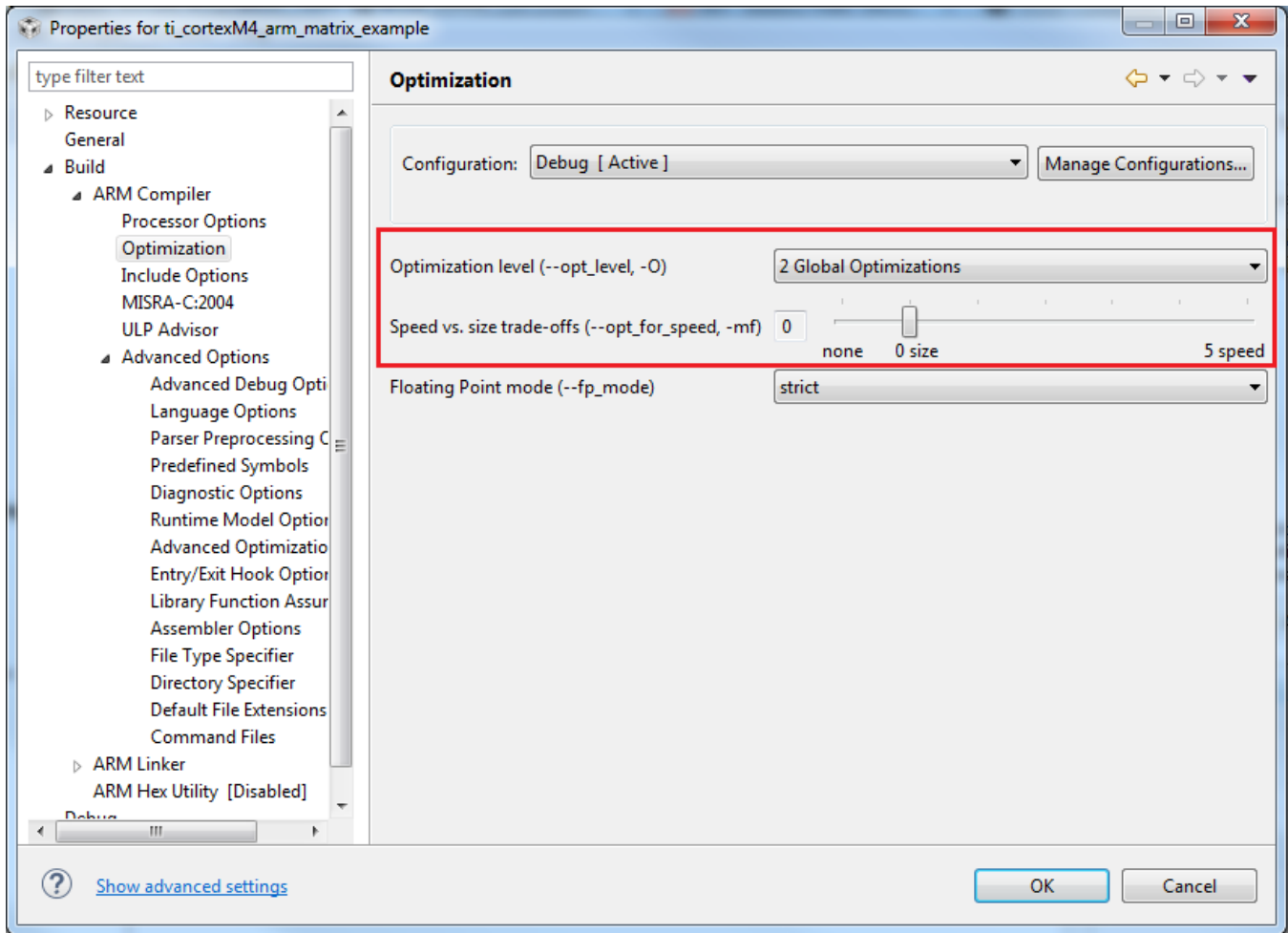
Before building the example projects, it is necessary to properly configure the project settings:

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).



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

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



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

- 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).

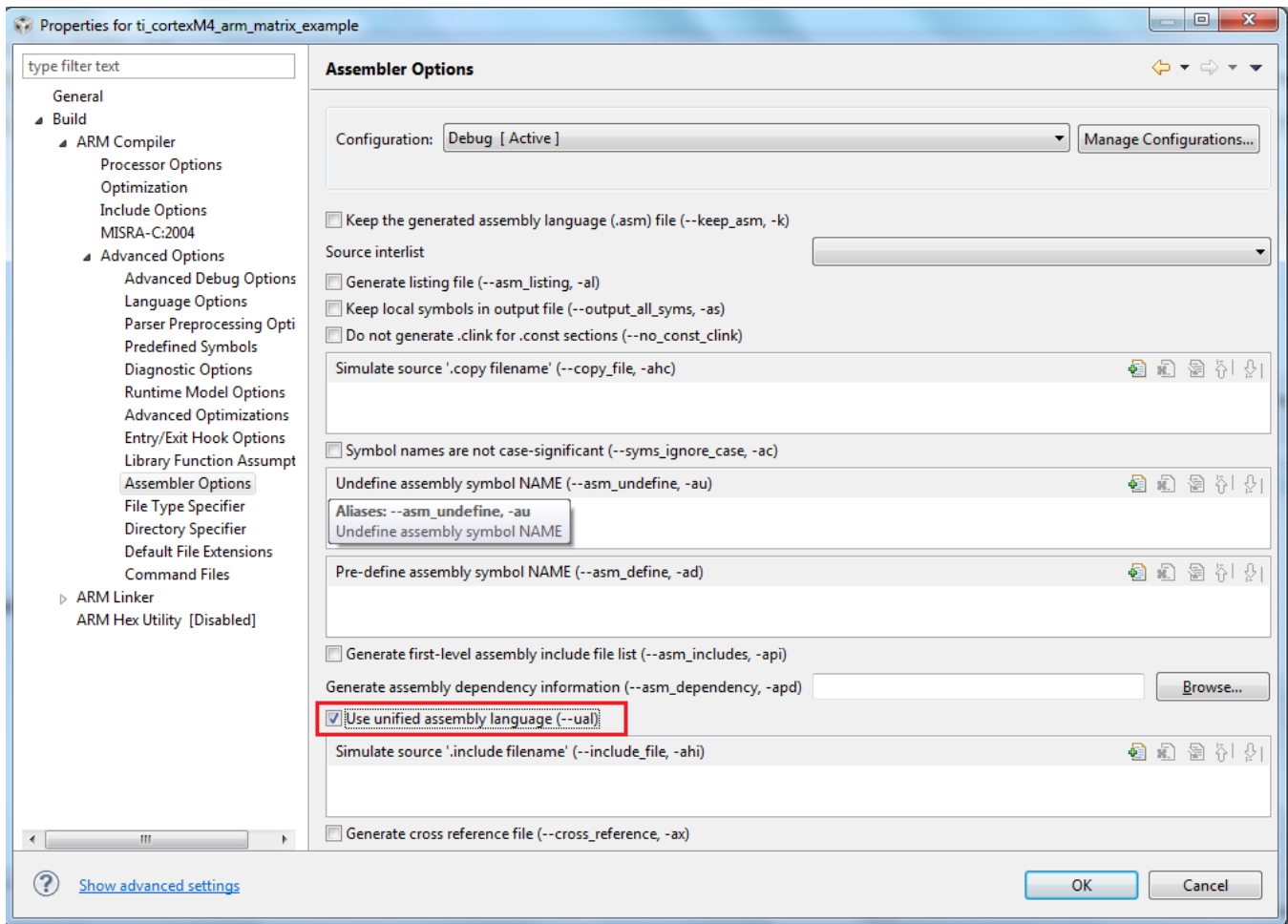
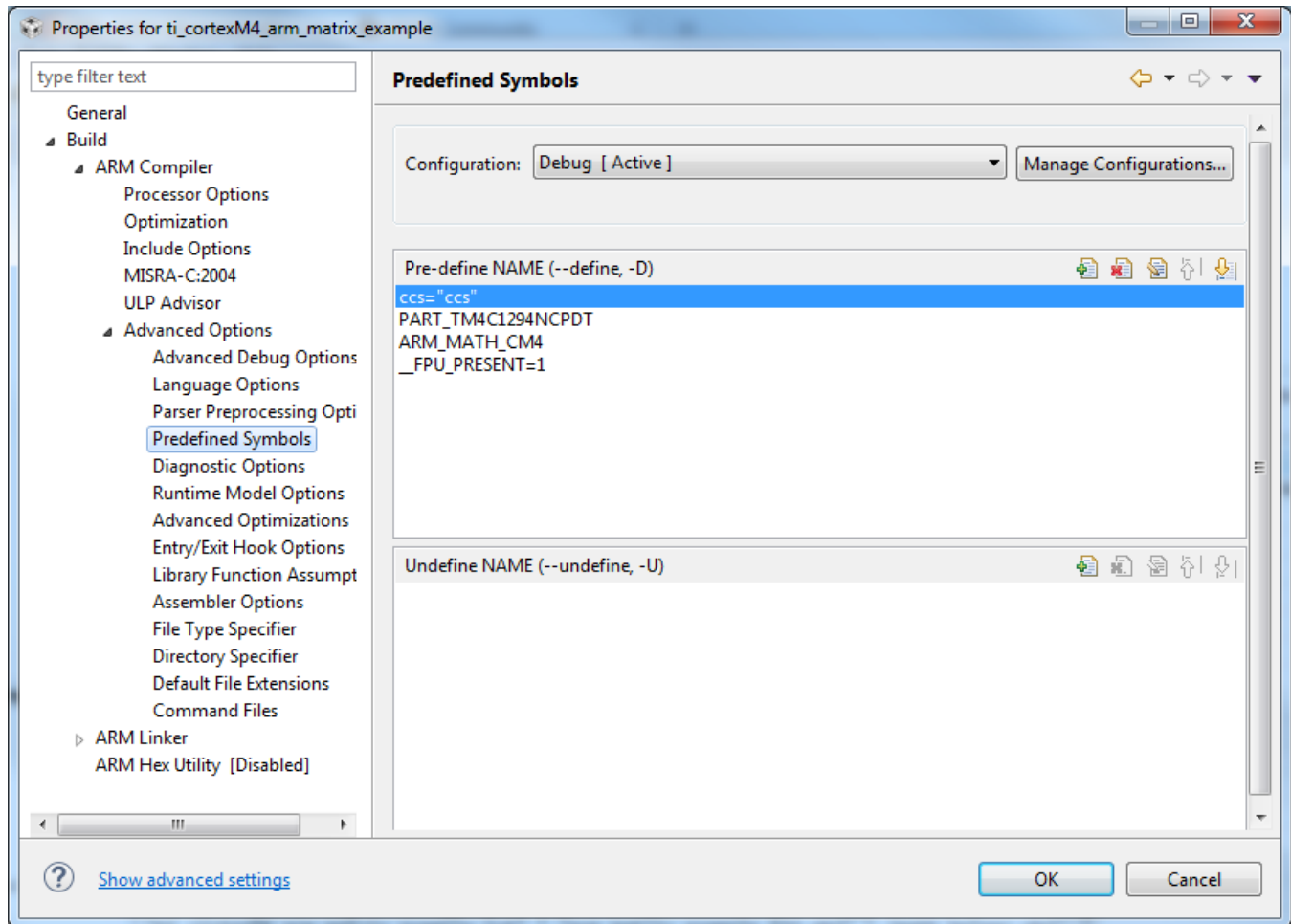


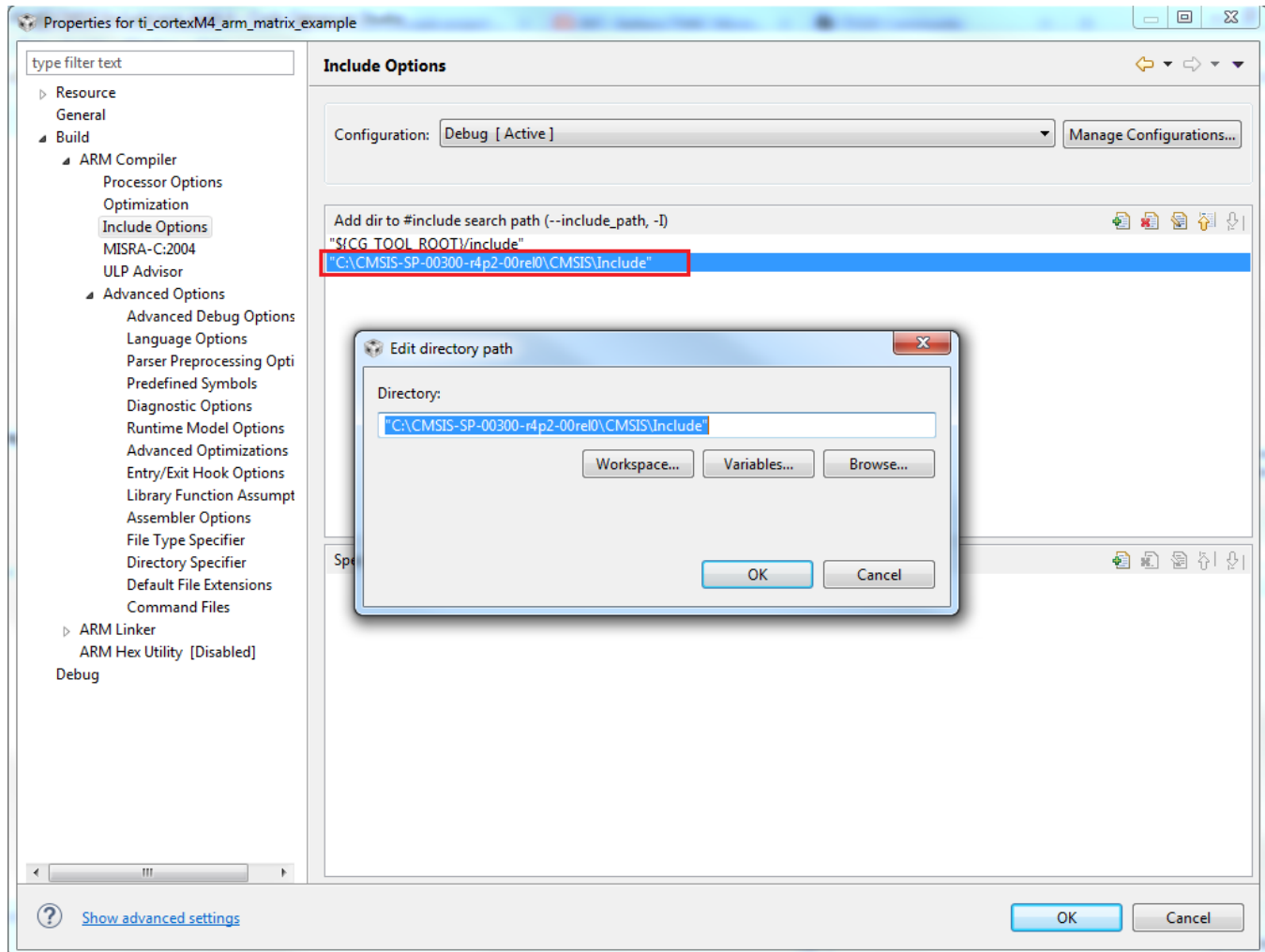
Figure 19. The Proper Assembler Options Needed for Compiling the Example Projects

- 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**.



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

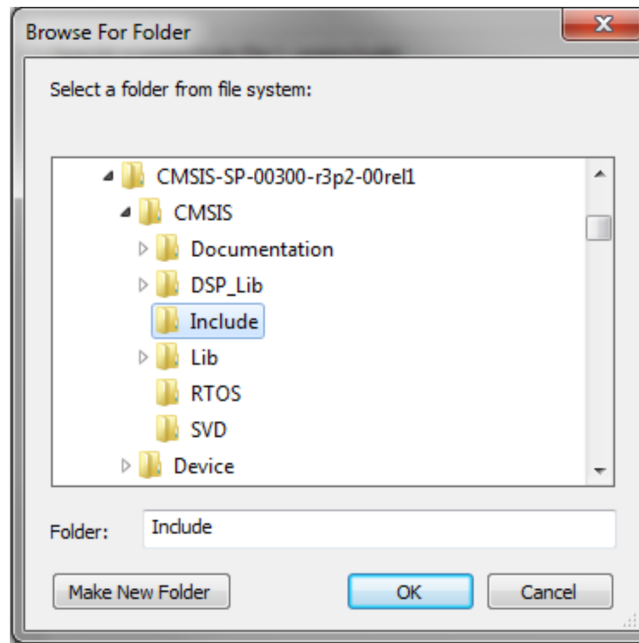
- Look for the **Add** button to **#include search path (--include\_path, -I)** field in the **Include Options** section (see [Figure 22](#)).



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

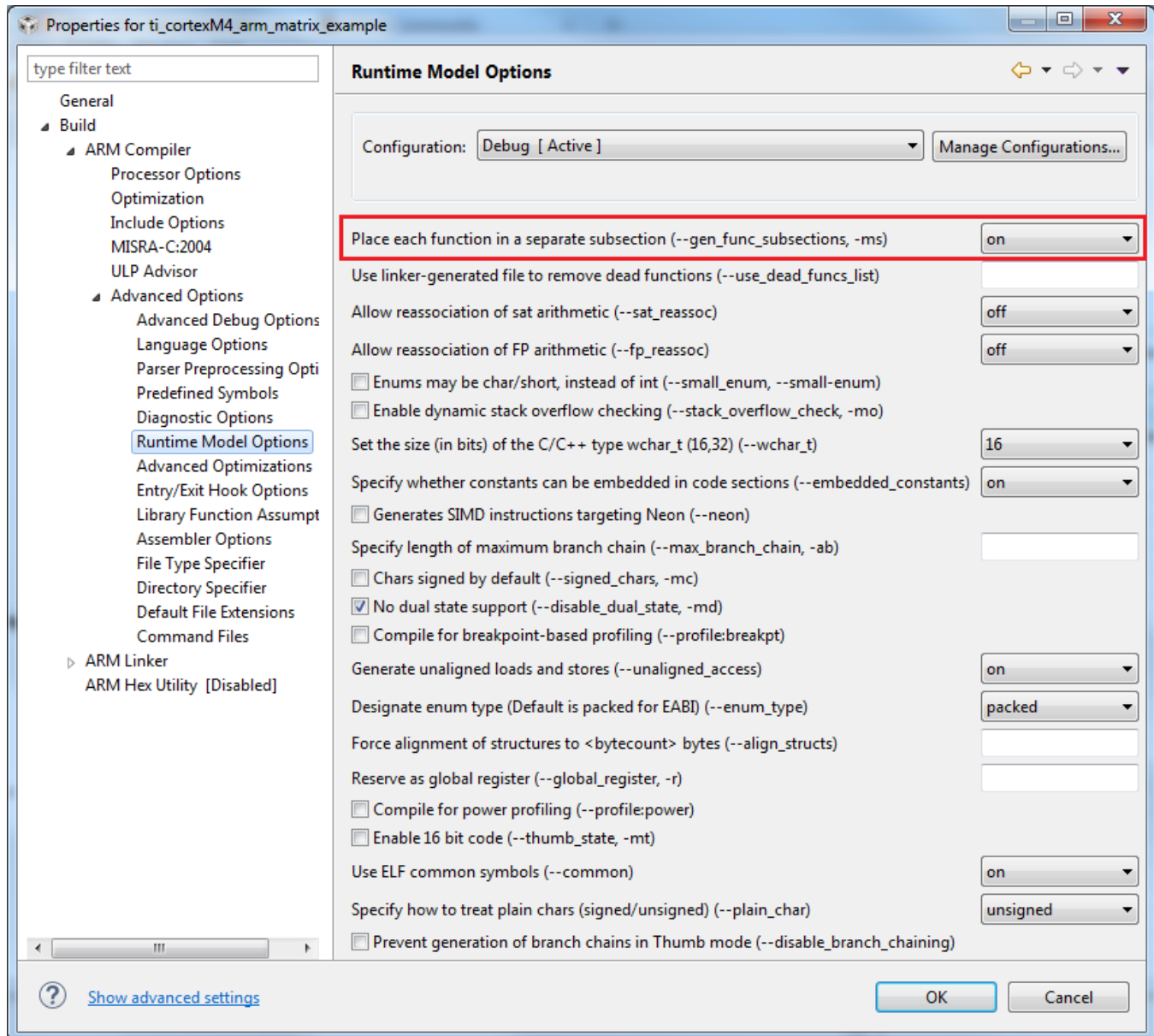


- Click the *Add...* button again, then the *Browse* button and browse to the Include directory located in the CMSIS directory, then click **OK** (see [Figure 21](#)).



**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.



**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**.

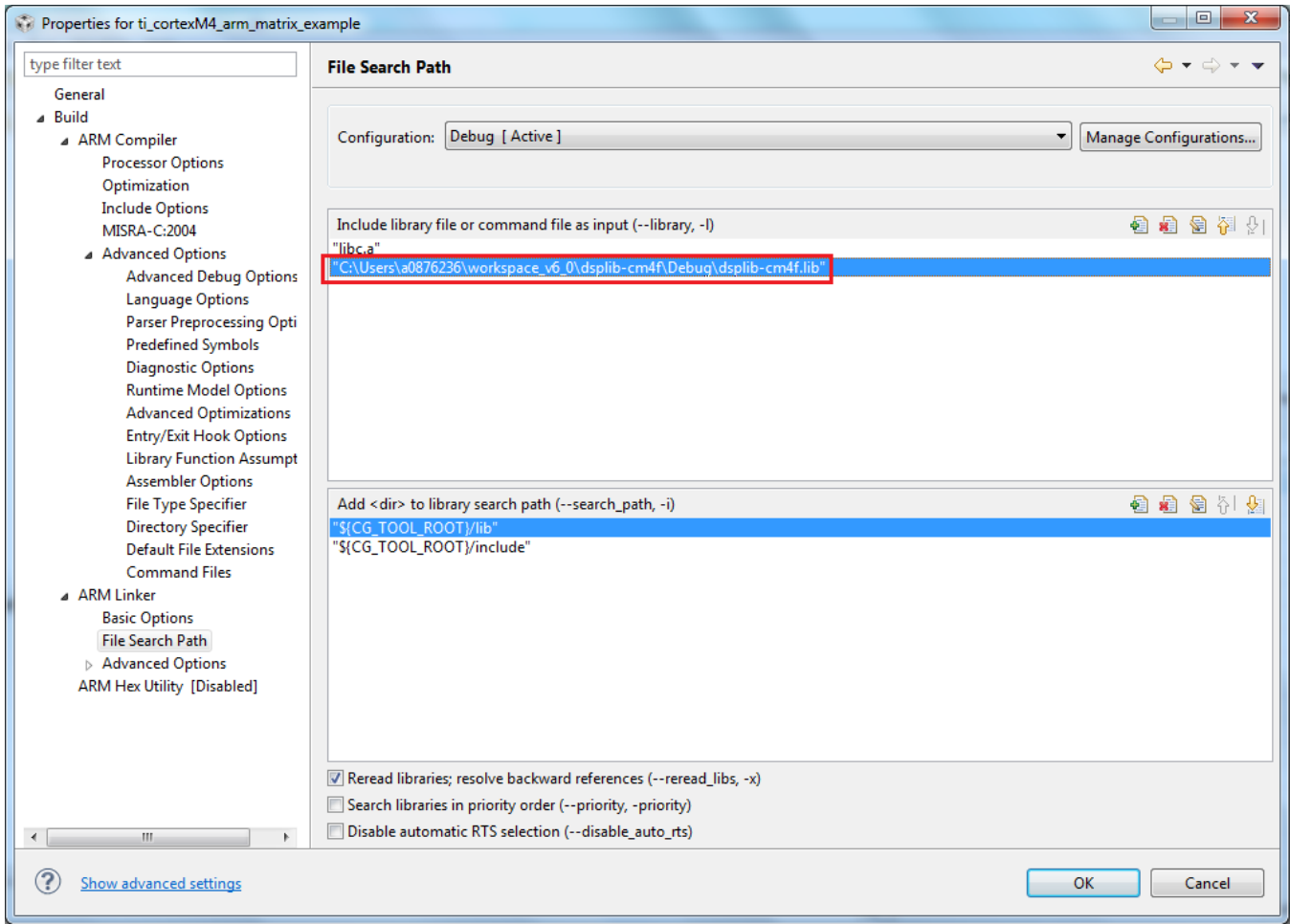


Figure 24. The Linker's File Search Path Modified to Include the dsplib Binary Compiled in Section 3

## 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 Studio 6.1 Quick Start Guide, available at [http://processors.wiki.ti.com/index.php/Category:Code\\_Composer\\_Studio\\_v6](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).

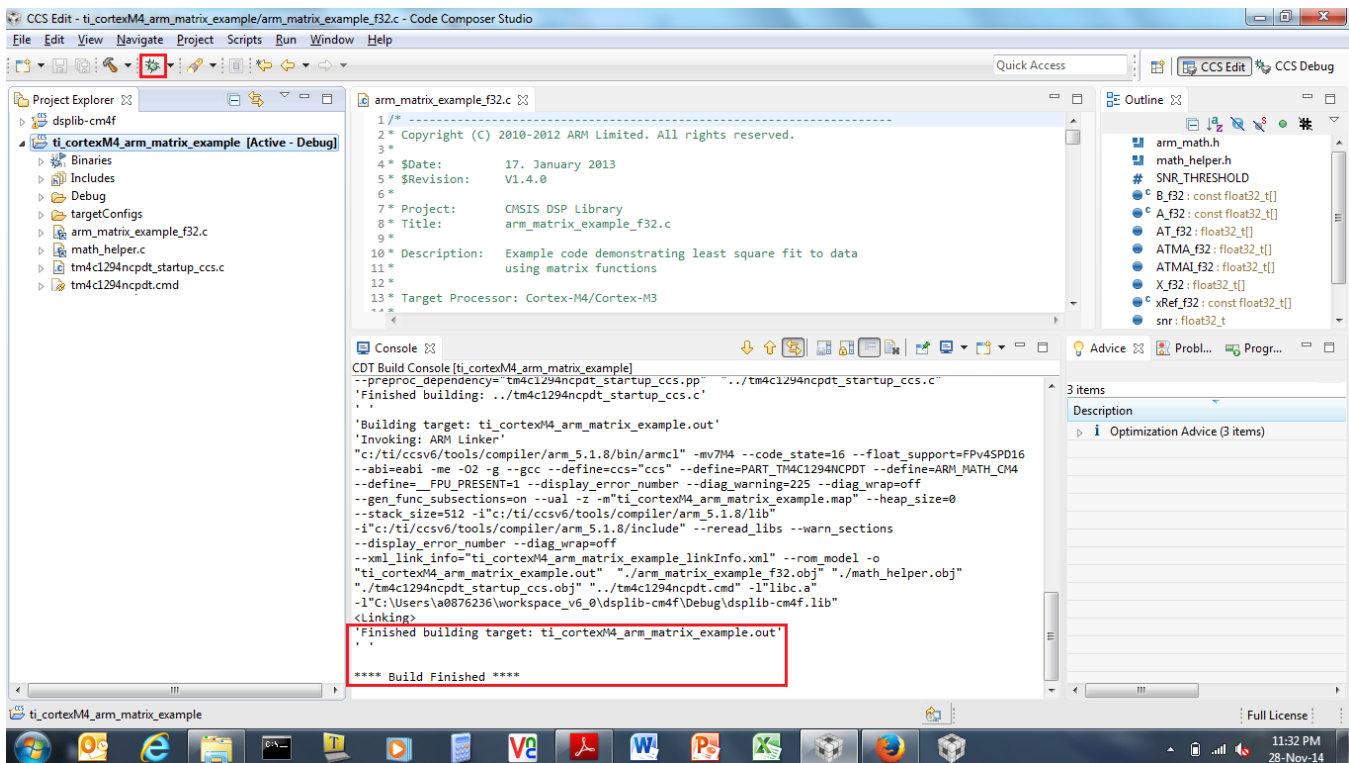
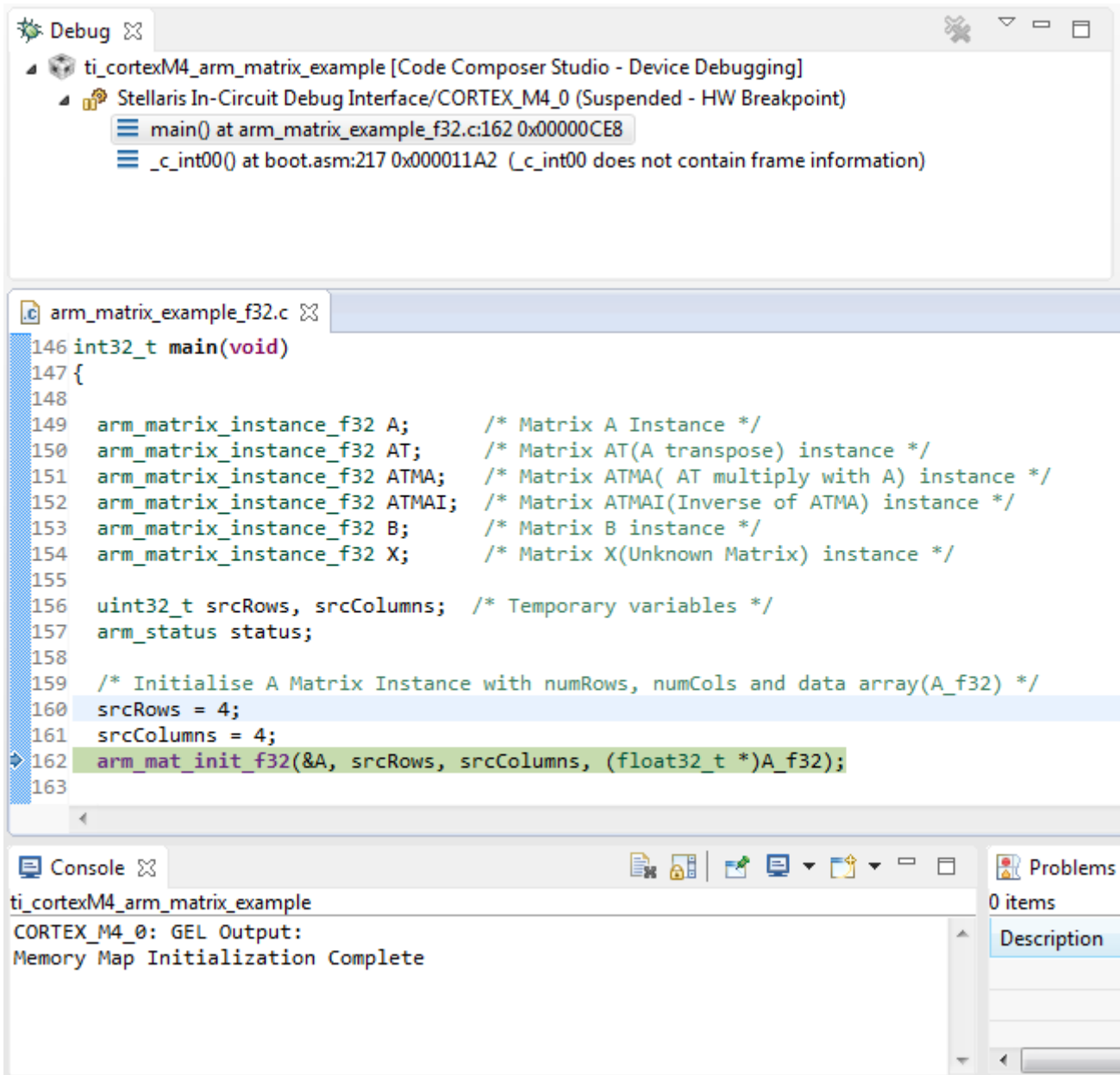


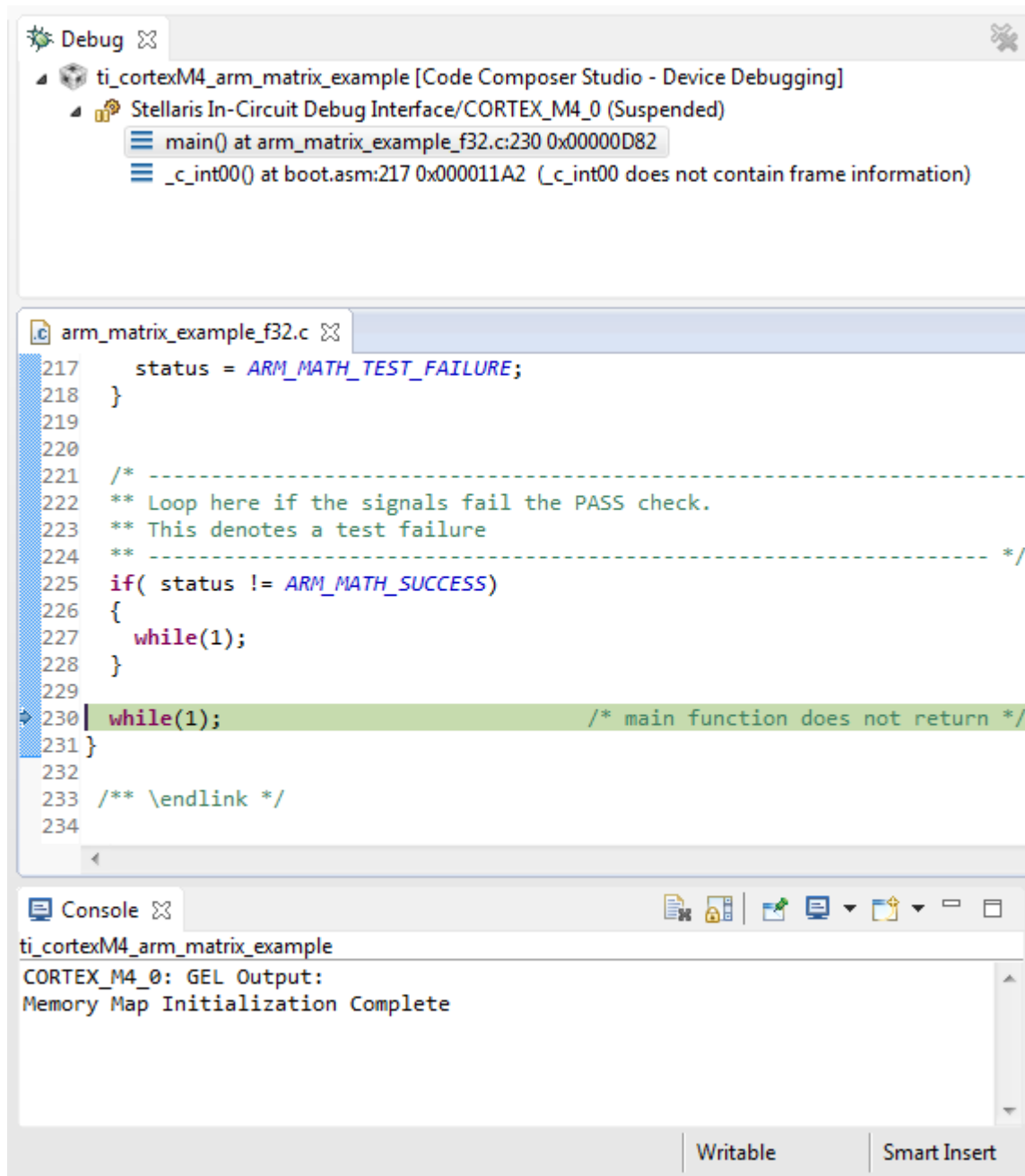
Figure 25. The Debug Context Being Displayed After the arm\_matrix\_example Project has Been Set Up for Debugging

- 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.



**Figure 26. The arm\_matrix\_example Project, After it has Been Loaded Into Flash and the Startup Code has run to the main() Function**

- 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.



**Figure 27. The arm\_matrix\_example Project Having Run to Successful Completion**

- 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.

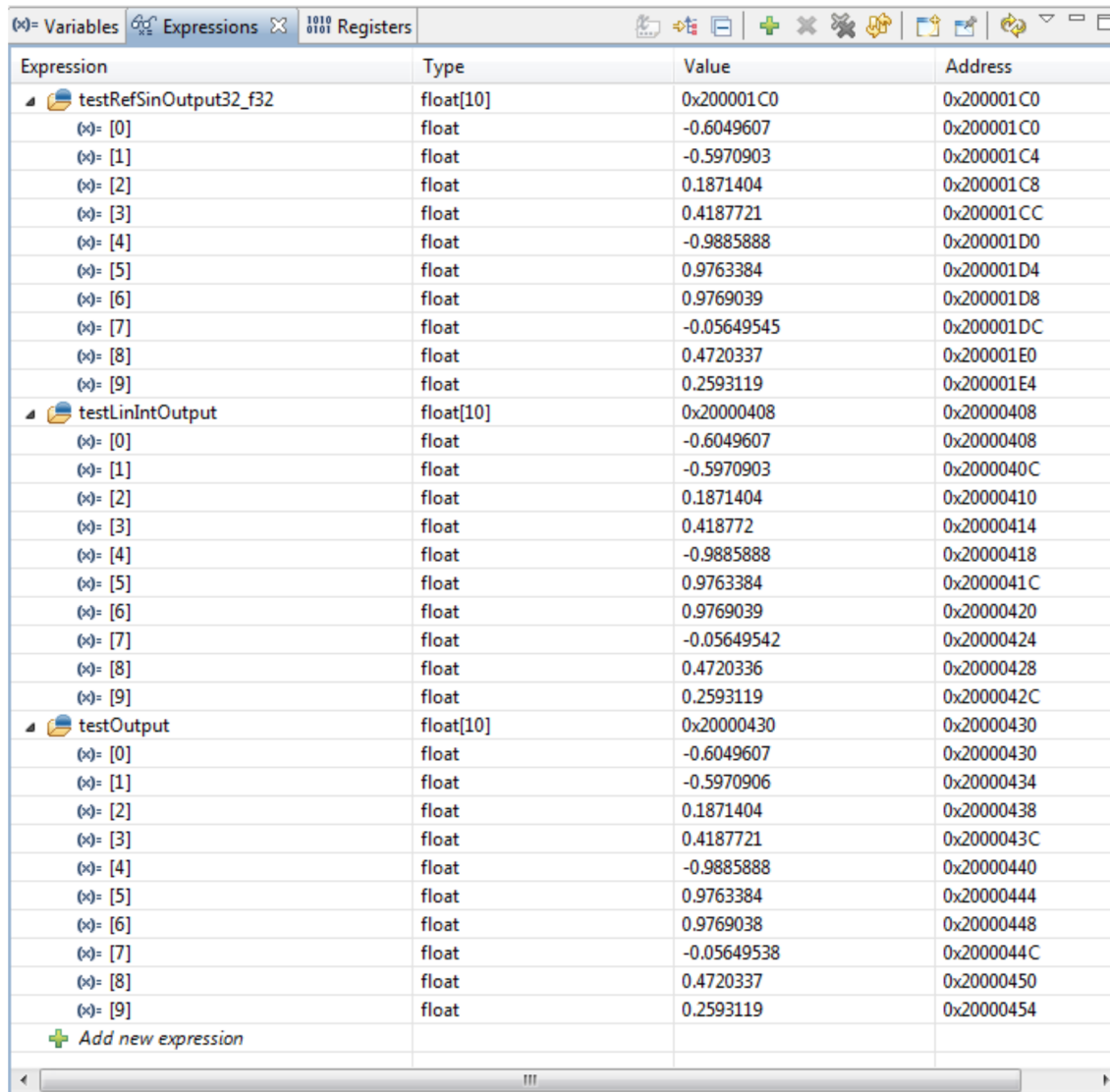
## 4.5 Source Code Modifications

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

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](http://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](http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v6).

## Revision History

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

| <b>Changes from F Revision (May 2015) to G Revision</b>        | <b>Page</b> |
|--|-------------|
| • Updates were made in the Abstract.....                       | 1           |
| • Information was updated in <a href="#">Section 3.1</a> ..... | 3           |
| • Information was updated in <a href="#">Section 3.2</a> ..... | 3           |
| • Information was updated in <a href="#">Section 3.4</a> ..... | 7           |
| • Information was updated in <a href="#">Section 4.1</a> ..... | 15          |
| • Information was updated in <a href="#">Section 4.3</a> ..... | 20          |

## IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ("TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products (<http://www.ti.com/sc/docs/stdterms.htm>), [evaluation modules](#), and [samples](http://www.ti.com/sc/docs/sampterm.htm) (<http://www.ti.com/sc/docs/sampterm.htm>).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2018, Texas Instruments Incorporated