![Texas Instruments logo]

Using Execute, Write, and Erase-Only Flash Protection on Stellaris Microcontrollers Using Code Composer Studio

Ashish Ahuja

ABSTRACT

Protection of code and IP in a microcontroller's Flash memory has always been an important consideration for the system designers. Stellaris® microcontrollers feature a code protection mechanism that enables developers to protect their code and IP in the end application, while providing the flexibility to upgrade the firmware using a boot loader. This application report describes using Flash protection to prevent code from being read while still allowing it to be executed (for example, the memory block may be written, erased, or executed but not read) using TI's Code Composer Studio™ v4.2.3.

Contents

List of Figures

1   Introduction

Stellaris microcontrollers offer different Flash memory protection modes as illustrated in Table 1. This document specifically focuses on the execute, write, and erase-only protection modes, and explains how these modes can be implemented using one of the examples provided in the StellarisWare®.

> **NOTE:** For more details on read-only Flash protection, see the *Flash Memory Protection* section in the *Stellaris LM3S9B96 Microcontroller Data Sheet* (SPMS182).

When the corresponding bits in the Flash Memory Protection Program Enable (FMPPEn) and the Flash Memory Protection Read Enable (FMPREn) registers are set to 1 and 0, respectively, you can erase and write to the Flash, and the Cortex core can execute code from Flash. Generally, most compilers put literal dump (constants and data) in the executable code. When such a code is located in executable-only (and read protected) blocks of the system memory, a literal dump can't be read. Therefore, the application will not execute properly. While using execute, write, and erase-only Flash protection, it is important that the code be compiled such that the literal dump does not reside in executable sections of the code, which may require using special compilers during the build process. The compiler that comes bundled with Texas Instruments Code Composer Studio's Code Generation Tools (at least v4.9) has this capability.

Table 1. Flash Protection Modes

Protection Mode	FMPPEn	FMPREn	Execute	Read	Write and Erase
Execute Only	0	0	√	x	x
Execute, Write and Erase Only	1	0	√	x	√

Table 1. Flash Protection Modes (continued)

Protection Mode	FMPPEn	FMPREn	Execute	Read	Write and Erase
Execute and Read Only	0	1	√	√	x
No Protection	1	1	x	x	x

This application report uses a "hello" example for the DK-LM3S9B96 kit to demonstrate execute, write, and erase-only Flash protection.

2   Requirements

You will need:

- A computer with the following software installed and running:
  - Windows XP or 7 operating system
  - Code Composer Studio v 4.2.3 with Code Generation Tools v4.9 that can be downloaded from the following URL: http://www.ti.com/ccs
  - StellarisWare software that can be downloaded from the following URL: http://www.ti.com/stellarisware
  - StellarisWare USB Drivers for Stellaris virtual COM port, Stellaris Evaluation board A and B that can be downloaded from the following URL: http://www.ti.com/tool/lm_ftdi_driver
- DK-LM3S9B96 Stellaris Development Kit
- USB cable with a standard type-A plug on one side and a mini-B plug (5 pin) on the other side

3 Procedure

Figure 1 illustrates the key steps that are covered in this application report. Each step is subsequently explained in the following sections.
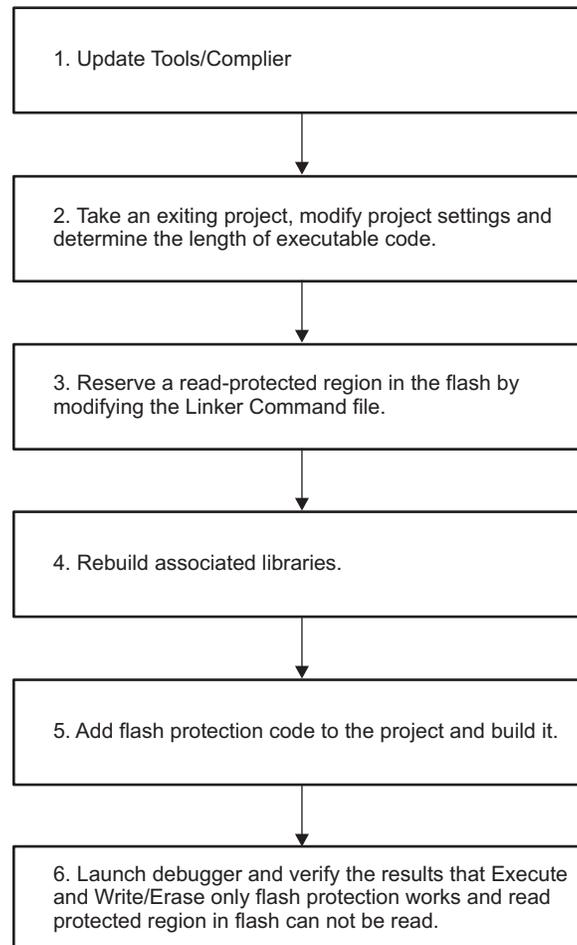
```
┌─────────────────────────────────────────┐
│  1. Update Tools/Complier                │
└─────────────────────────────────────────┘
                    │
                    ▼
┌─────────────────────────────────────────┐
│  2. Take an exiting project, modify      │
│  project settings and determine the      │
│  length of executable code.              │
└─────────────────────────────────────────┘
                    │
                    ▼
┌─────────────────────────────────────────┐
│  3. Reserve a read-protected region in   │
│  the flash by modifying the Linker       │
│  Command file.                           │
└─────────────────────────────────────────┘
                    │
                    ▼
┌─────────────────────────────────────────┐
│  4. Rebuild associated libraries.        │
└─────────────────────────────────────────┘
                    │
                    ▼
┌─────────────────────────────────────────┐
│  5. Add flash protection code to the     │
│  project and build it.                   │
└─────────────────────────────────────────┘
                    │
                    ▼
┌─────────────────────────────────────────┐
│  6. Launch debugger and verify the       │
│  results that Execute and Write/Erase    │
│  only flash protection works and read    │
│  protected region in flash can not be    │
│  read.                                   │
└─────────────────────────────────────────┘
```

Figure 1. Flowchart Showing Sequence of Key Steps

3.1 Update Tools and Compiler

3.1.1 Checking Currently Installed Code Generation Tools Version

Code Generation Tools v4.9 is not a standard component of the currently available Code Composer Studio v4.2.3 installation package.

To determine the current version of the Code Generation Tools installed on your machine:

1. Launch Code Composer Studio.
2. Go to the Help menu.
3. Click on Software Updates.
4. Go to the Manage Configuration option. The following window will appear.

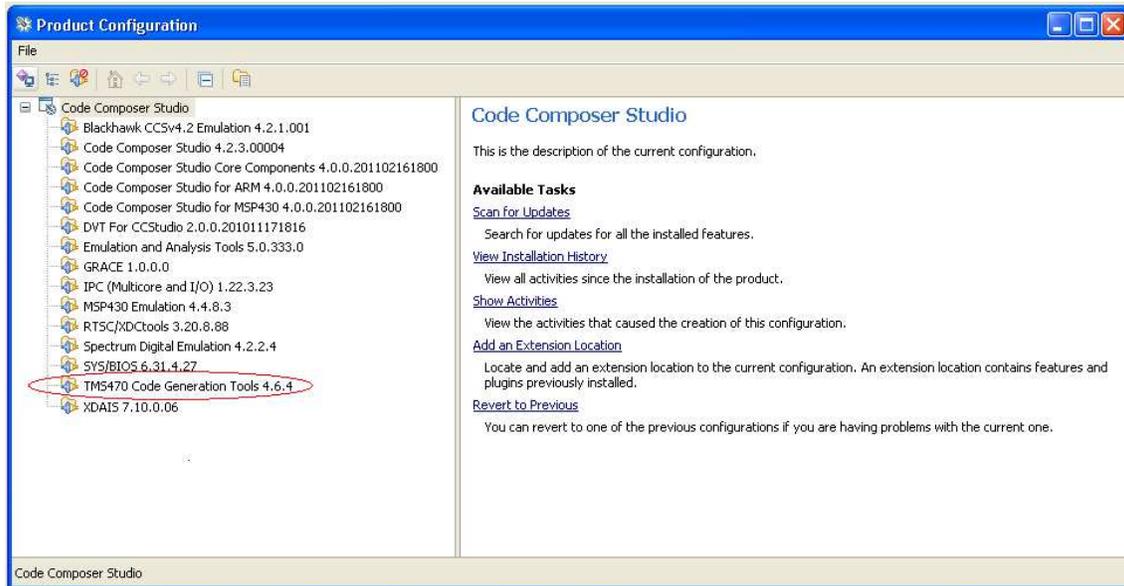The compiler version has been highlighted in Figure 2.

Figure 2. Determining Versions of Installed Components

If Code Generation Tools v4.9 is not installed on your machine, follow the instructions starting with Section 3.2 to update your current compiler to v4.9 or else proceed to Section 3.3.

3.2 Installing and Updating Code Generation Tools to Version 4.9

In Code Composer Studio:

1. Go to the Help menu.
2. Click on Software Updates.
3. Click on Find and Install. An Install and Update wizard will appear.
4. Select Search for new features to install.
5. Click on the Next button.
6. Select Code Generation Tools Updates on the screen that appears next (as shown in Figure 3).
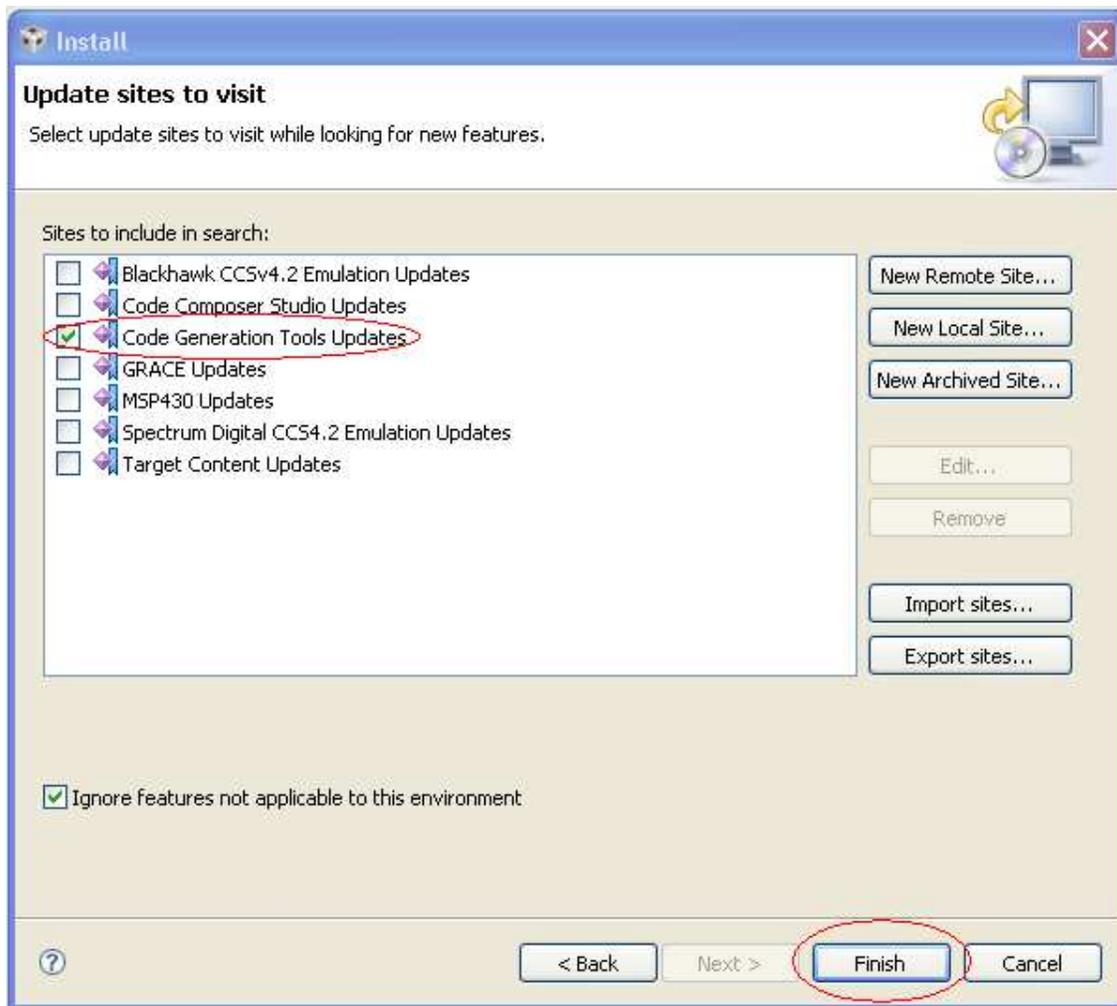7. Click on the Finish button.

Figure 3. Selecting Update Options

Code Composer Studio searches for updates and opens a list of available updates as shown in Figure 4.

1. Select TMS470 Code Generation Tools 4.9.0 → Other → Code Generation Tools Update.
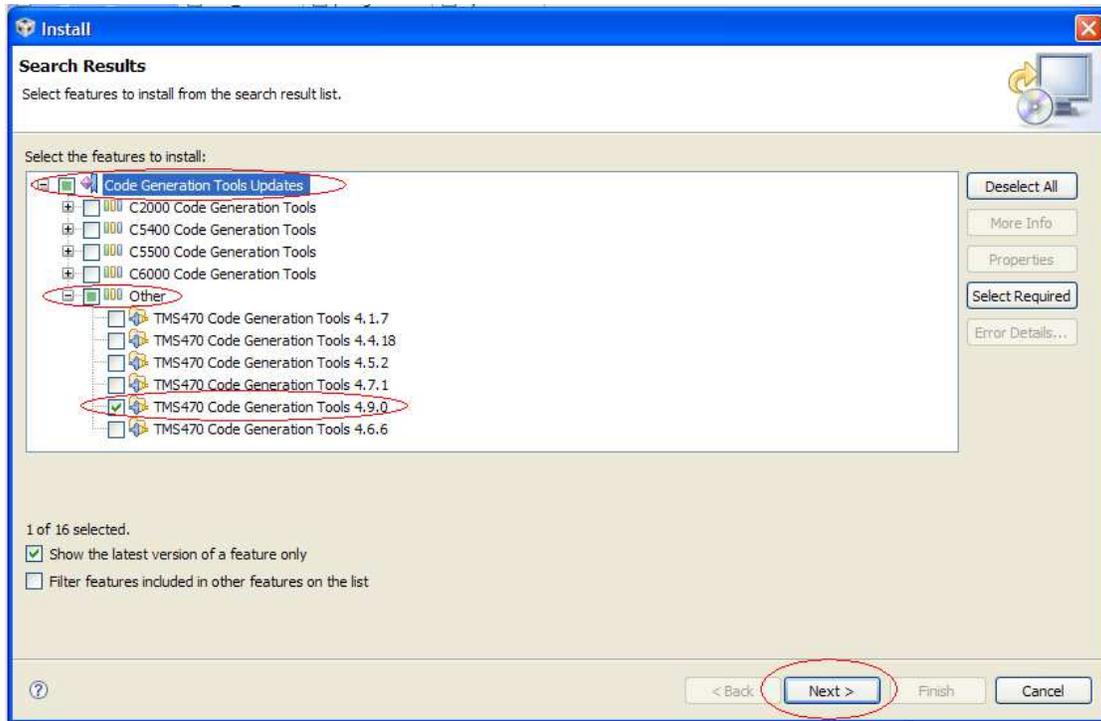2. Click on Next and proceed to the Feature License screen.

Figure 4. Selecting Update Components

3. Read and accept the license agreement and click on the Next button as shown in Figure 5.

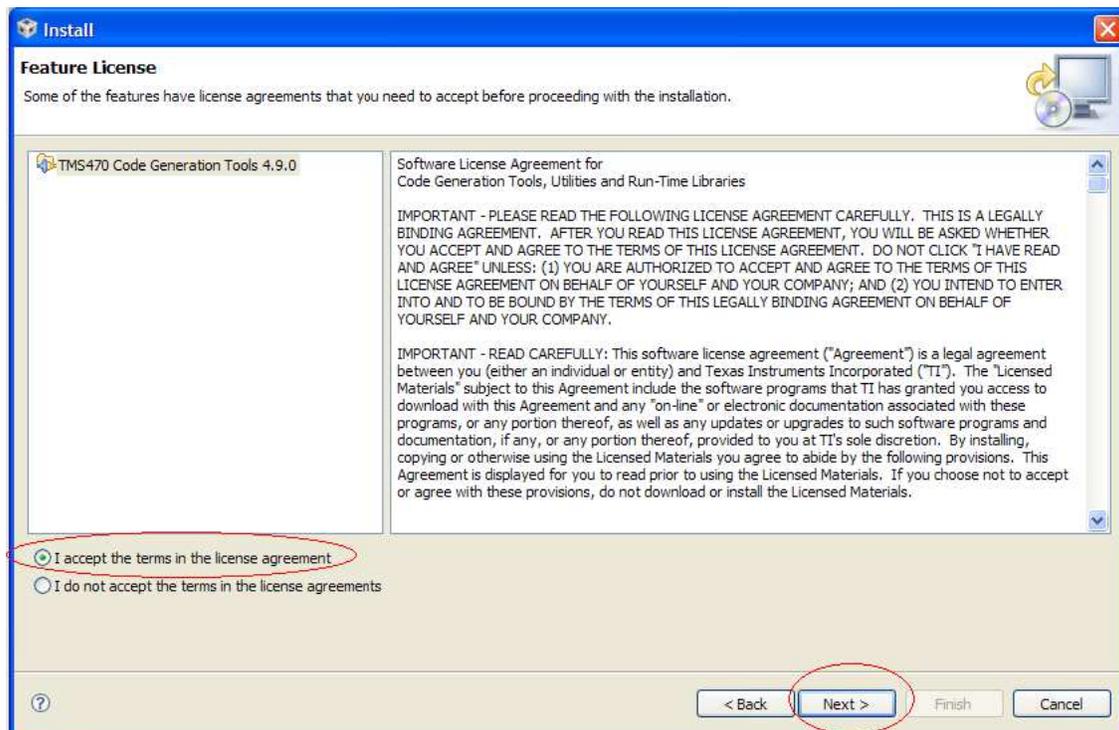**Figure 5. Install Window_Accept Terms**

Using Execute, Write, and Erase-Only Flash Protection on Stellaris
Microcontrollers Using Code Composer Studio

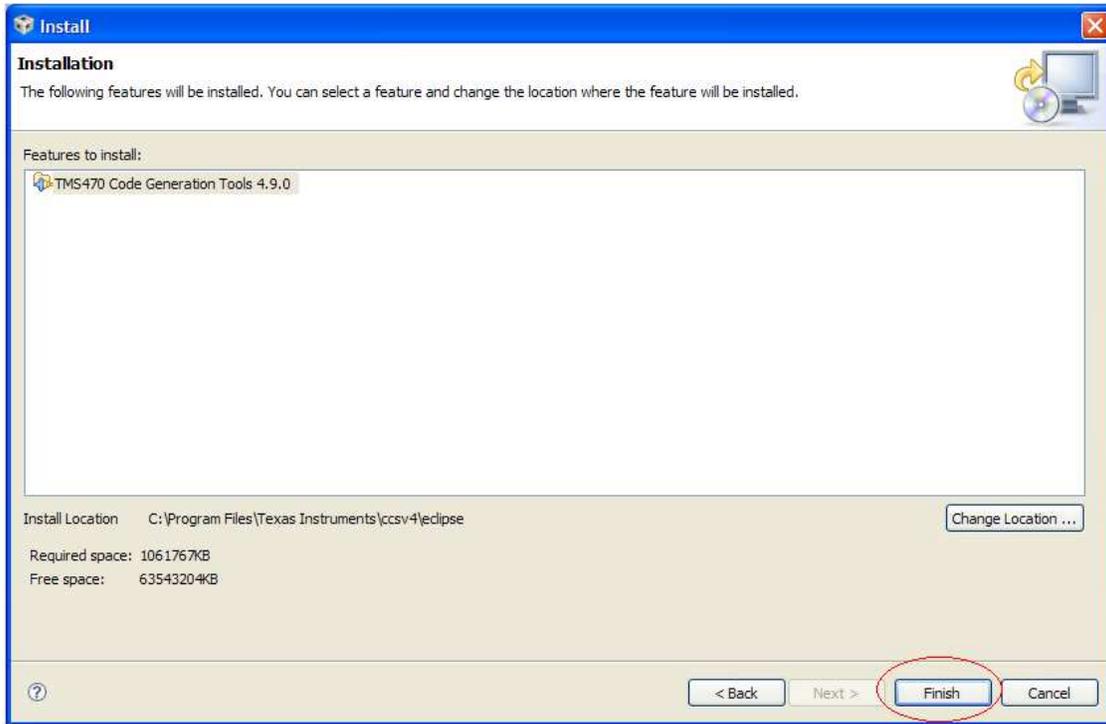4. Click on the Finish button as shown in Figure 6.

Figure 6. Install Window_Finish

5. Click on the Install button as shown in Figure 7. The verification screen appears next.
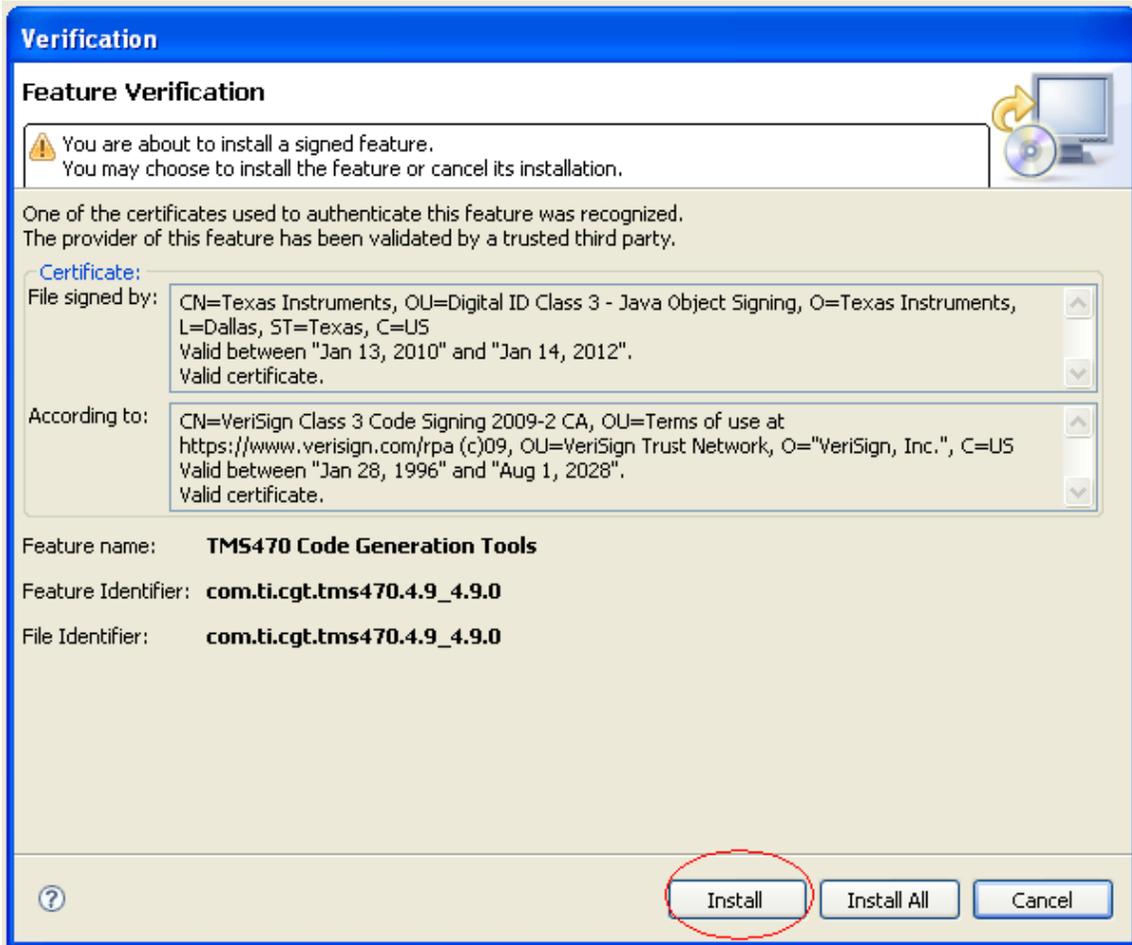
Figure 7. Verification Window

6. The following pop-up appears (see Figure 8) after the update is successfully installed.

Figure 8. Install and Update Dialog Box

7. Click on the Yes button and restart Code Composer Studio.

3.3   *Confirm That the Currently Installed Version of Code Generation Tools is 4.9*

Upon launching Code Composer Studio:

1. Go to Help menu.
2. Click on Software Updates.
3. Manage Configuration option. The following window will appear.

4. Confirm that Code Generation Tools v4.9 has been successfully installed as shown in the red circle in Figure 9.
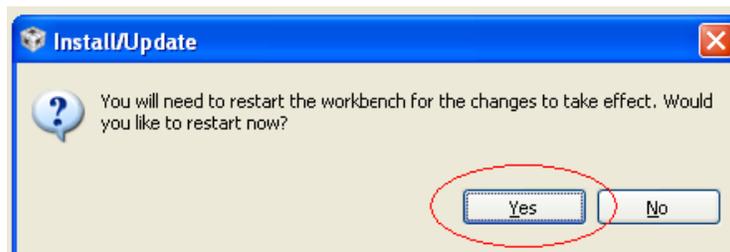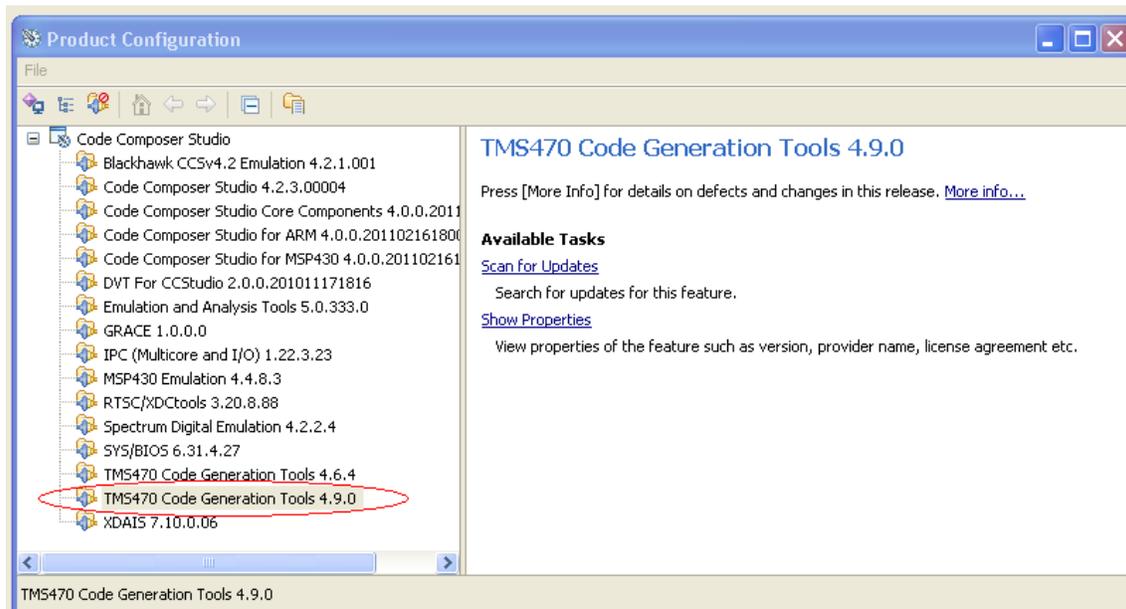

Figure 9. Checking Update Options

4 Modify Existing Project Settings and Determine the Length of Executable Code

In the following section, an existing example from StellarisWare is used. Project settings are modified and compiled in such a way that a literal dump does not reside in executable sections of code. This protocol enables you to use execute, write, and erase-only Flash protection, which provide the capability to write, erase, and execute from Flash but do not give you the ability to read from it.

4.1 Build an Existing Project (hello.c) in Code Composer Studio From StellarisWare

1. Connect the DK-LM3S9B96 board to the computer.
2. Launch Code Composer Studio.
3. Go to the Project menu.
4. Select the Import Existing CCS/Eclipse Project.
5. Click on the Browse button → browse to the *C:\StellarisWare\boards\dk-lm3s9b96\hello* location, or wherever the "hello" project is located in your computer.
6. Import the project files into a workspace.
7. Build, debug and download the project with default settings. The project should successfully build without any errors, and the application should execute normally as expected.

4.2 Modify Project Settings

In Code Composer Studio:
1. Go to the Project menu
2. Select Properties to open the Project properties window as shown in Figure 10.
3. Select Code Composer Studio Build
4. In the General tab → select the TI v4.9.0 compiler in the Code Generation Tools field.
5. Select <automatic> for the Runtime Support Library.
6. After selecting these options, click on the Apply button.
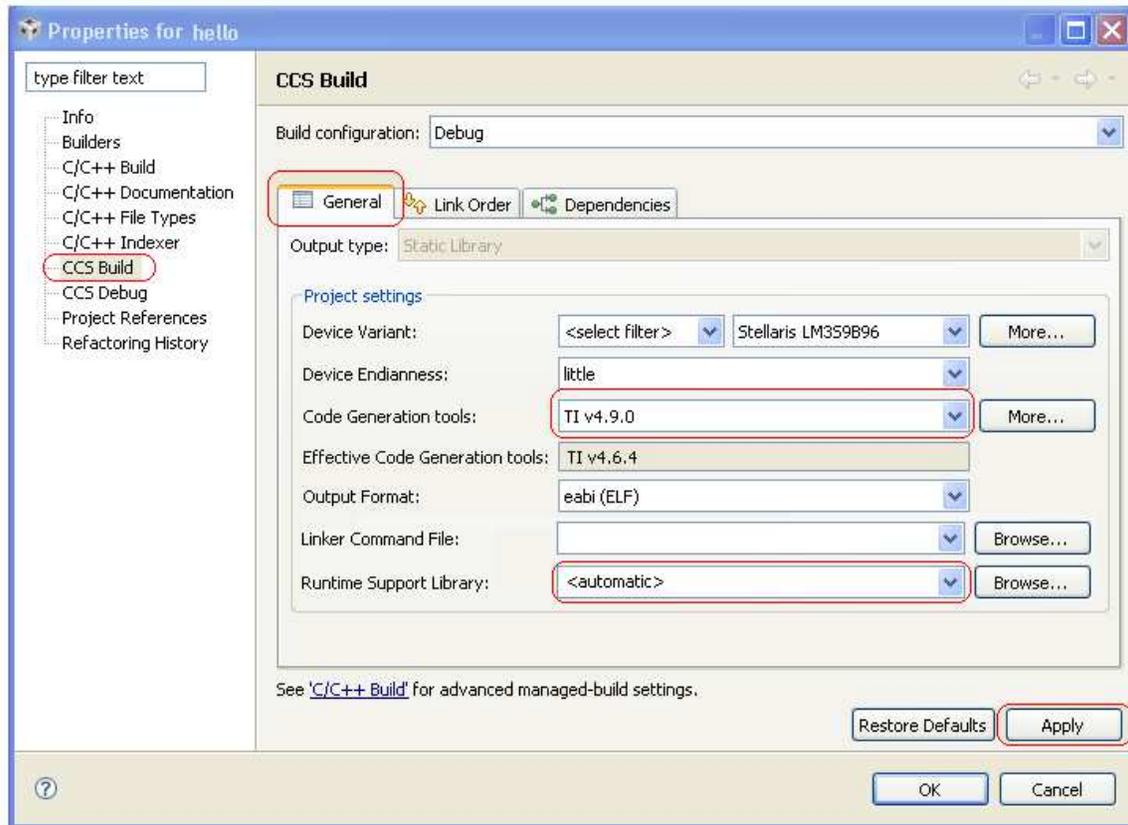
Figure 10. Properties for "hello"

Copyright © 2012, Texas Instruments Incorporated

7. The Build Configuration Settings window will appear. Select the options as shown in Figure 11 and click on the OK button.
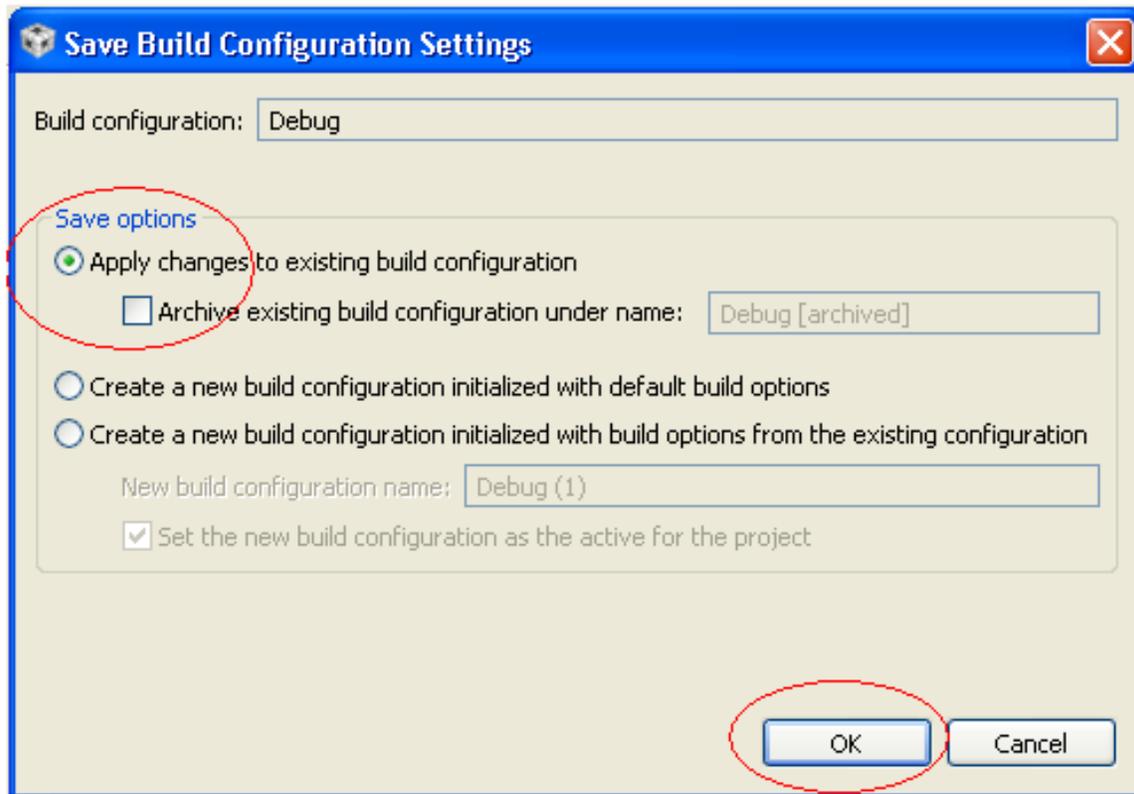
Figure 11. Save Build Configuration Settings

8. Click on the OK button when the dialog box appears as shown in Figure 12.
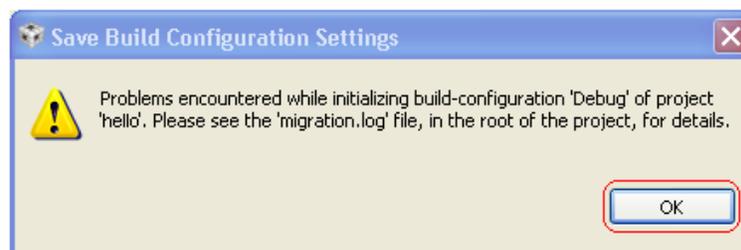
Figure 12. Save Build Configuration Settings

9. Click on the OK button on the window that appears next (as shown in Figure 13) and return to the Code Composer Studio project properties for this project.
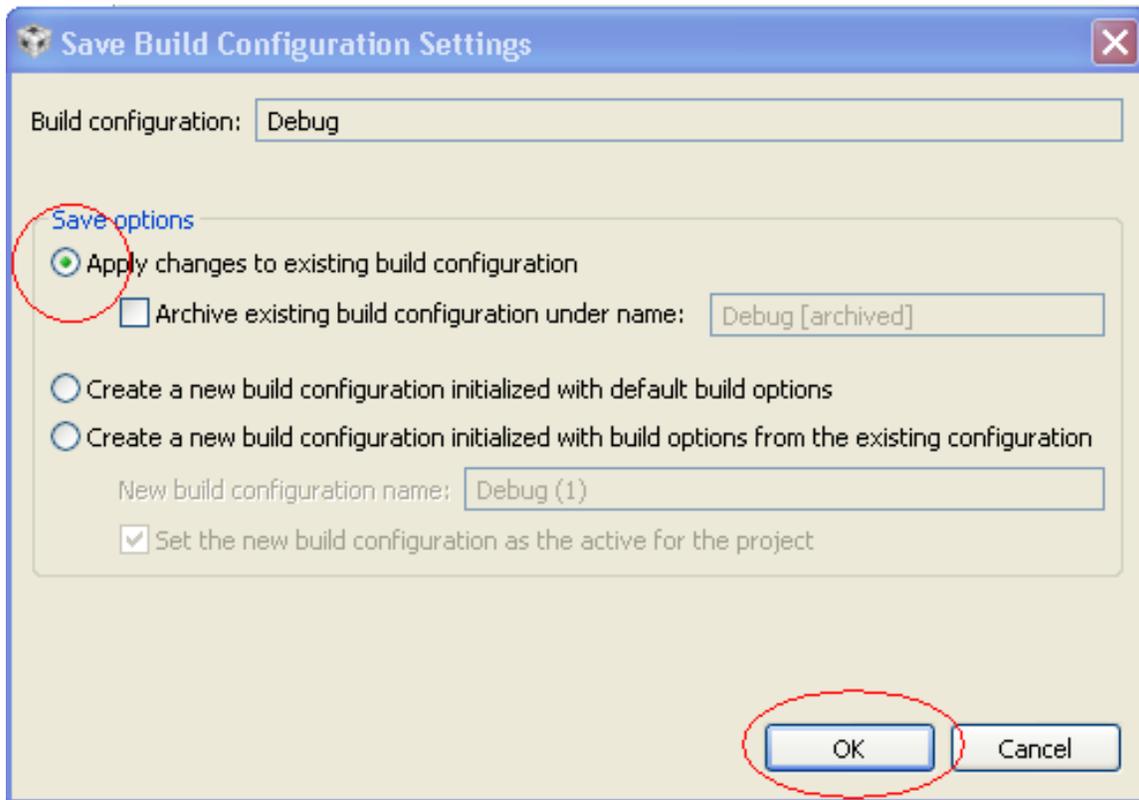
Figure 13. Properties for "hello" - Code Composer Studio Build

10. Select C/C++ Build under the Tool Settings tab.
11. Select Runtime Model Options in the TMS470 Compiler field.

12. Select options (on and off), as shown in red circles in Figure 14, → click on the Apply button.
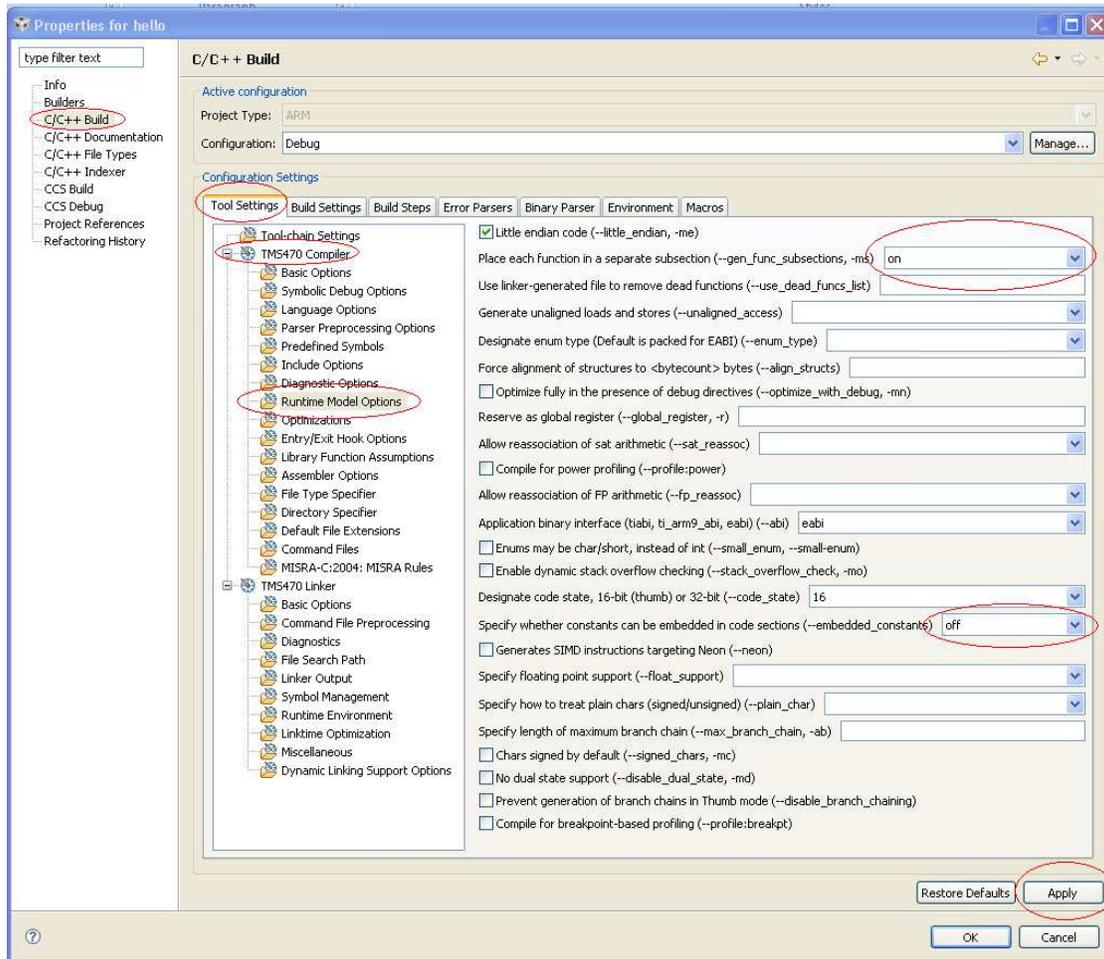
Figure 14. Properties for "hello" - C/C++ Build

As shown in Figure 15, select Predefined Symbols and add additional defines to the Predefined NAME section by clicking on the 🔲 button. Add Predefined NAMEs as shown in the red circles in Figure 15. Pay attention to the letter case; and include the following three symbols:

- TARGET_IS_TEMPEST_RB1
- ccs
- PART_LM3S9B96
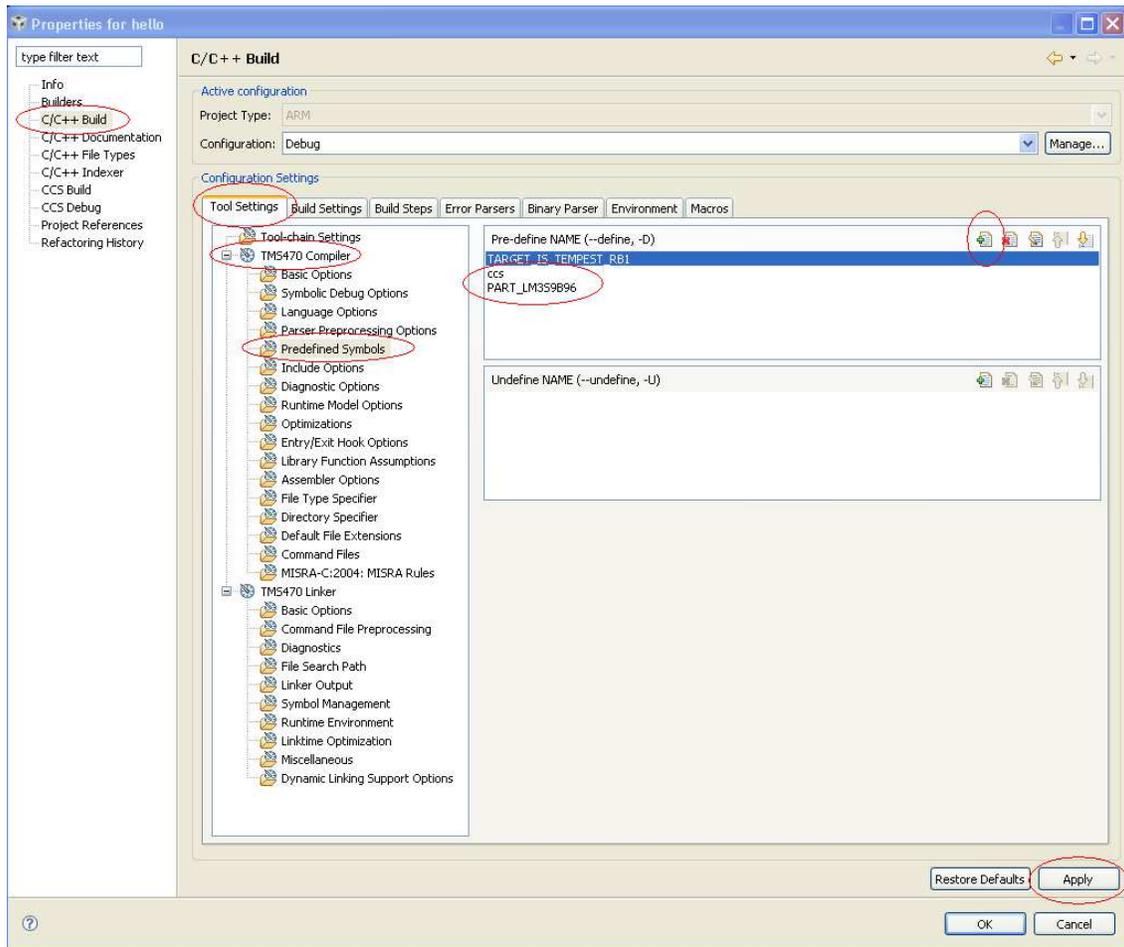
Figure 15. Properties for "hello" - Predefined Name

13. Click Apply → return to the Code Composer Studio main window/ file editor window.

4.3 Rebuilding the Project

Rebuild the code by selecting the Rebuild All option in the Project menu. The project should successfully build without any errors.

4.4 Determining How Much Flash to Protect by Examining Contents of the Map File

In this example, the executable section of code is protected in the Flash by making it execute, write, and erase only. The remaining sections of the code like the vector table, constants, and initialization values need not be protected.

In order to protect the executable code, the map file is first examined to determine the length of executable code. The map file in Code Composer Studio ends with the .map extension and (hello.map) can be found in the Debug directory in the Code Composer Studio project folder, which is located at *C:\StellarisWare\boards\dk-lm3s9b96\hello\ccs\Debug* assuming that StellarisWare has been installed in the *C:\ directory*. Map files can be modified using a text editor.

If the project was previously complied using an older version of Code Generation Tools (older compiler), the project folder will have a map file (hello_ccs.map) associated with that version of the compiler as well. Make sure that the map file (hello.map) being referred to is generated by Code Generation Tools v4.9. This information is available in the beginning of the map file as shown in Figure 16. Or, you can use the Windows time stamp to identify the map file that has just been generated.

```
********************************************************************************
                    TMS470 Linker PC v4.9.0
********************************************************************************
>> Linked Thu May 12 18:24:36 2011

OUTPUT FILE NAME:   <hello.out>
ENTRY POINT SYMBOL: "_c_int00"  address: 0000346d


MEMORY CONFIGURATION

          name          origin    length      used     unused   attr    fill
-----------------------  --------  --------  --------  --------  ----  --------
    FLASH               00000000  00040000  000036c0  0003c940  R   X
    SRAM                20000000  00018000  0000021e  00017de2  RW  X


SEGMENT ALLOCATION MAP

run origin  load origin   length    init length attrs members
----------  -----------  ----------  ----------- ----- -------
  00000000   00000000    000036c8    000036c8    r-x
  00000000   00000000    0000011c    0000011c    r-- .intvecs
  0000011c   0000011c    00001e9e    00001e9e    r-- .const
  00001fbc   00001fbc    000016c6    000016c6    r-x .text
  00003688   00003688    00000040    00000040    r-- .cinit
  20000000   20000000    00000200    00000000    rw-
  20000000   20000000    00000200    00000000    rw- .stack
  20000200   20000200    0000001e    0000001c    rw-
  20000200   20000200    0000001c    0000001c    rw- .data
  2000021c   2000021c    00000002    00000000    rw- .bss
```

Figure 16. Screen Shot of hello.map

As shown in Figure 16, the executable section of the code is 0x16C6 bytes long (5830 bytes long). As explained in the *Stellaris LM3S9B96 Microcontroller Data Sheet* (SPMS182), the Flash can only be protected in the multiples of 2KB. For the purpose of this project, at least 6KB (3 sectors) of Flash must to be protected to accommodate 5830 bytes of executable code.

Next, the length of readable section of code must be determined, which is the sum of the following sections:

- Vector table (.intvecs) : 0x11C
- Constants (.const) : 0x1e9c
- Initialization data (.cinit) : 0x40

The sum of these three sections is 8186 bytes (for example, 7KB), which translates to 8KB (4 sectors) in Flash.

NOTE: If the "hello" example is modified or rebuilt, the length of sections mentioned above may get modified. Options like compiler optimization also affect the length of different code sections. Therefore, keeping some additional margin is always recommended, but is not necessary.

---

5   Reserve a Read Protected Region in the Flash by Modifying the Linker Command File

The linker command file (also known as linker file, or scatter file in some integrated development environments (IDEs)) is used to define the addresses of sections in the memory. The linker command file in Code Composer Studio ends with the .cmd extension and it (hello_ccs.cmd) can be found in the *C:\StellarisWare\boards\dk-lm3s9b96\hello* directory assuming that StellarisWare has been installed in the *C:* directory. The linker command file can be modified using a text editor. Figure 17 shows the contents of the original default linked command file created for this example. As highlighted using red circles, two sections in memory are defined namely FLASH and SRAM.

```
/* System memory map */

MEMORY
{
    /* Application stored in and executes from internal flash */
    FLASH (RX) : origin = APP_BASE, length = 0x00040000
    /* Application uses internal RAM for data */
    SRAM (RWX) : origin = 0x20000000, length = 0x00018000
}

/* Section allocation in memory */

SECTIONS
{
    .intvecs:    > APP_BASE
    .text    :   > FLASH
    .const   :   > FLASH
    .cinit   :   > FLASH
    .pinit   :   > FLASH

    .vtable :    > RAM_BASE
    .data    :   > SRAM
    .bss     :   > SRAM
    .sysmem :    > SRAM
    .stack   :   > SRAM
}

__STACK_TOP = __stack + 512;
```
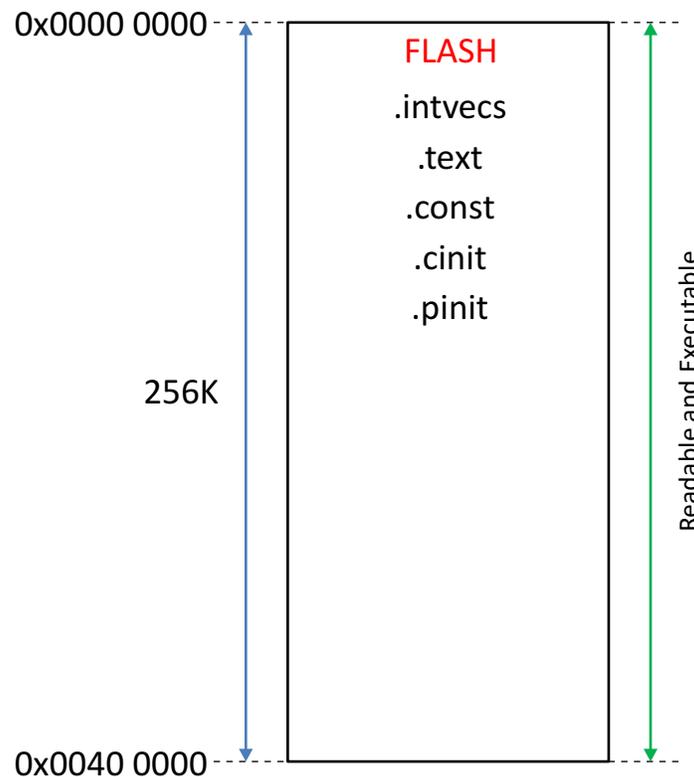
Figure 17. System Memory Map

The Flash memory map can be graphically represented in Figure 18 and includes the addresses from 0x0000 0000 to 0x0040 000. In other words, 256KB of Flash memory is executable and readable.

Figure 18. Memory Map_Flash

In order to protect executable code from being read, a separate read-protected section must be reserved in the Flash. Before such a section can be reserved, it must be first defined in the memory map. For the purpose of this project, 2KB in Flash is reserved for the vector table; named as FLASH_1 and configured as readable and executable (default configuration).

Per Section 4.4, 6KB in Flash is reserved for executable code; named as FLASH_EX and configured as not-readable but executable. The remaining Flash can be named as FLASH_2. It is configured as readable and executable (default configuration) for the rest of the code including constants and initialization data. This is represented in Figure 19.

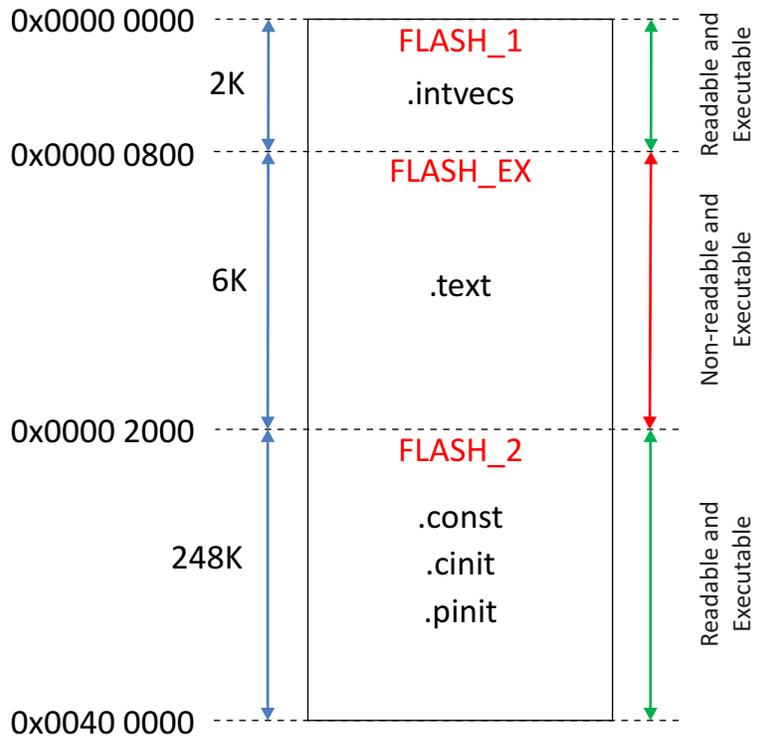| Address Space | 0x0000 0000 to 0x0000 0800 | Readable and Executable |
| Address Space | 0x0000 0800 to 0x0000 2000 | Non-Readable and Executable |
| Address Space | 0x0000 2000 to 0x0040 0000 | Readable and Executable |

Figure 19. Memory Map (FLASH_1, FLASH_EX and FLASH_2)

Copyright © 2012, Texas Instruments Incorporated

In order to implement the changes just described, modify the linker command file as shown in Figure 20.

```
#define APP_BASE 0x00000000
#define RAM_BASE 0x20000000

/* System memory map */

MEMORY
{
    /* Accessible region in internal flash where vector table resides */
    FLASH_1 (RX) : origin = APP_BASE, length = 0x00000800

    /* Read protected region in internal flash where executable code resides */
    FLASH_EX (X) : origin = 0x00000800 length = 0x00001800

    /* Read protected region in internal flash where constants & initialization data reside */
    FLASH_2 (RX) : origin = 0x00002000, length = 0x0003E000

    /* Application uses internal RAM for data */
    SRAM (RWX) : origin = 0x20000000, length = 0x00018000
}

/* Section allocation in memory */

SECTIONS
{
    .intvecs:    > APP_BASE
    .text    :   > FLASH_EX
    .const   :   > FLASH_2
    .cinit   :   > FLASH_2
    .pinit   :   > FLASH_2

    .vtable :    > RAM_BASE
    .data   :    > SRAM
    .bss    :    > SRAM
    .sysmem :    > SRAM
    .stack  :    > SRAM
}

__STACK_TOP = __stack + 512;
```

Figure 20. Linker Command File

Save and close the linker command file, and rebuild the project.

Now open the map file and make sure that it reflects the new sections that have been created in the memory as shown in Figure 21. Compare this with the map file referenced previously. Notice that an additional read protected region in Flash has been created, "FLASH_EX", along with another region, "FLASH_2", which is both readable and writeable.

Copyright © 2012, Texas Instruments Incorporated

```
*******************************************************************************
                     TMS470 Linker PC v4.9.0
*******************************************************************************
>> Linked Fri May 13 16:06:26 2011

OUTPUT FILE NAME:    <hello.out>
ENTRY POINT SYMBOL: "_c_int00"  address: 00001cb1


MEMORY CONFIGURATION

         name              origin    length      used     unused    attr   fill
-----------------------  ---------  ---------  ---------  ---------  ----  --------
   FLASH_1               00000000   00000800   0000011c   000006e4  R  X
   FLASH_EX              00000800   00001800   000016c6   0000013a     X
   FLASH_2               00002000   0003e000   00001ede   0003c122  R  X
   SRAM                  20000000   00018000   0000021e   00017de2  RW X


SEGMENT ALLOCATION MAP

run origin   load origin    length    init length  attrs  members
----------   -----------   ----------  -----------  -----  -------
00000000     00000000      0000011c    0000011c     r--
00000000     00000000      0000011c    0000011c     r--  .intvecs
00000800     00000800      000016c6    000016c6     r-x
00000800     00000800      000016c6    000016c6     r-x  .text
00002000     00002000      00001ee0    00001ee0     r--
00002000     00002000      00001e9e    00001e9e     r--  .const
00003ea0     00003ea0      00000040    00000040     r--  .cinit
20000000     20000000      00000200    00000000     rw-
20000000     20000000      00000200    00000000     rw-  .stack
20000200     20000200      0000001e    0000001c     rw-
20000200     20000200      0000001c    0000001c     rw-  .data
2000021c     2000021c      00000002    00000000     rw-  .bss
```

Figure 21. TMS470 Linker PC v4.9.0 Map File

6   Rebuild Associated Libraries (driver lib and graphics lib with Code Generation Tools v4.9)

This example ("hello") uses API function calls from the StellarisWare Driver Library and the Graphics Library. Therefore, these libraries must also be compiled using Code Generation Tools v4.9 with the same compiler settings that project ("hello") was built with. Instructions to compile DriverLib and GraphicsLib are illustrated in the following sections.

6.1   Compiling Driver Lib

Add the Driver Library (driverlib) project to an existing workspace where the "hello" project already exists.

1. Go to the Project menu in Code Composer Studio → select Import Existing CCS/Eclipse Project → click on the Browse button and browse to the *C:\StellarisWare\driverlib* location. Import the project file for the DriverLib in the current workspace.

2. The driverlib project will be set as the Active Project. If not, then it can be manually set as an active project in the project explorer window pane by right clicking on driverlib project and selecting Set as Active Project option as shown in Figure 22.
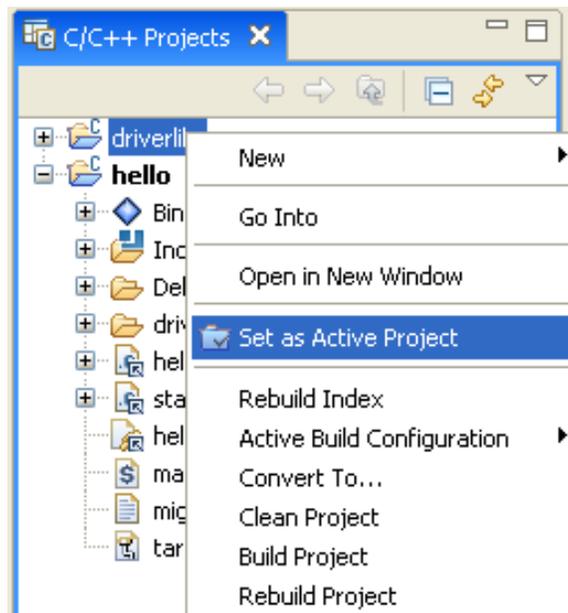
Figure 22. Setting a Project as an Active Project

3. Go to Project menu → select Properties to open Project properties window as shown in Figure 23.
4. Select Code Composer Studio Build in the General tab → select the TI v4.9.0 compiler in the Code Generation Tools field. After selecting these options → click on the Apply button.
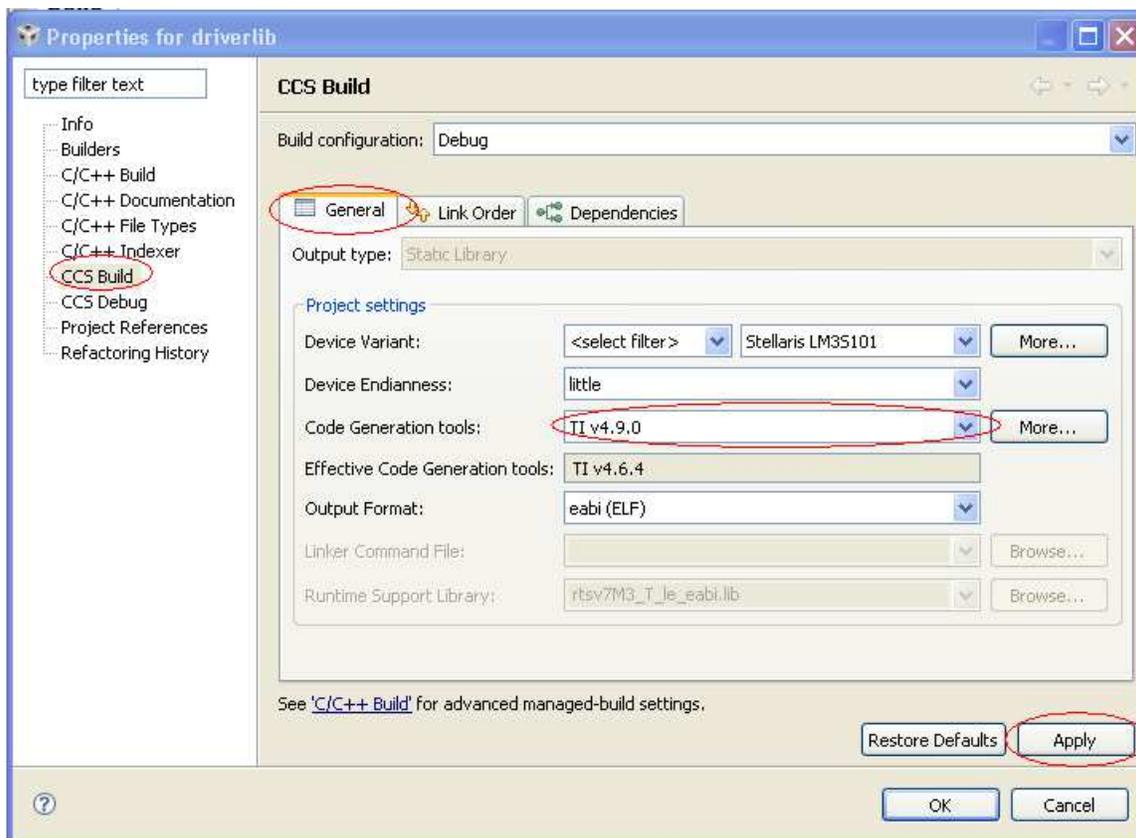

Figure 23. Properties for Driver Lib

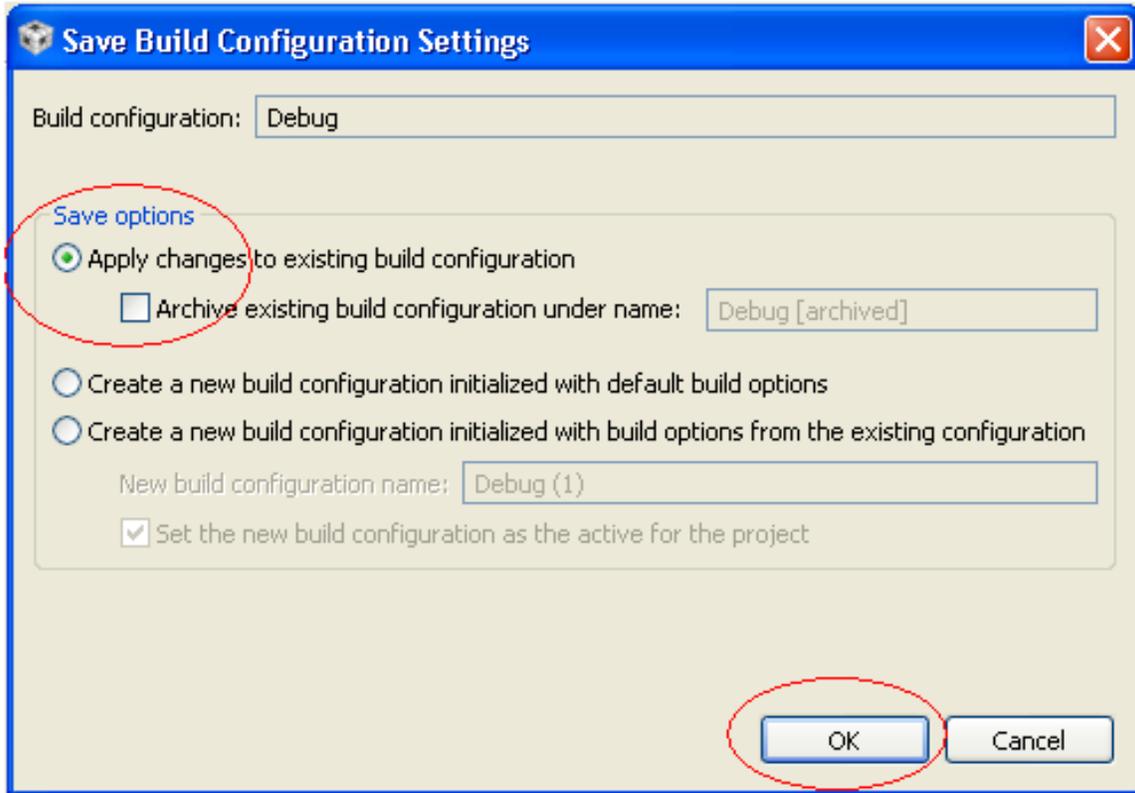5. The Build Configuration Settings window will appear → select options as in Figure 24 → click on the OK button.

Figure 24. Save Build Configuration Settings

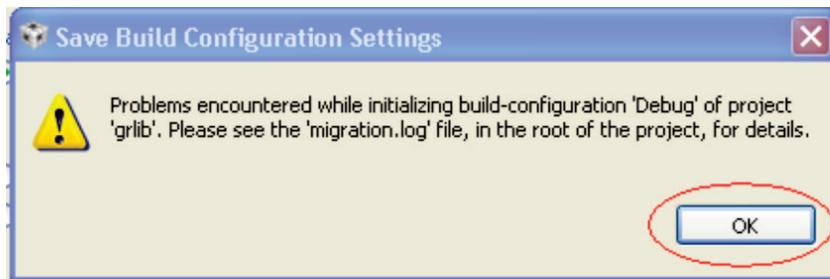6. Click on the OK button when the following dialog box appears.

Figure 25. Save Build Configuration Settings (Problems)

7. Click OK on the window that appears next (as shown in Figure 26) → return to the Code Composer Studio project properties for this project.
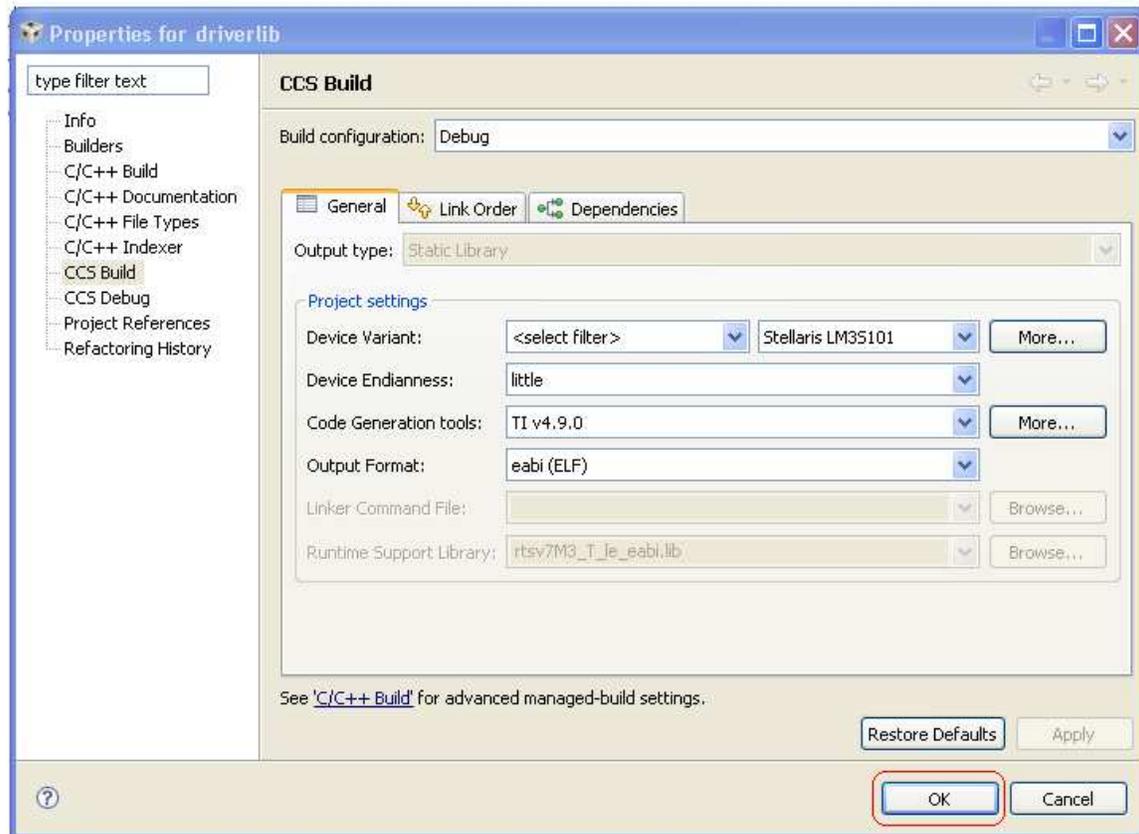

Figure 26. Properties for driverlib (Code Composer Studio Build)

8. Select C/C++ Build under the Tool Settings tab → select Runtime Model Options in the TMS470 Compiler field. Select options (on and off) as shown in red circles in Figure 27 and click on the Apply button.
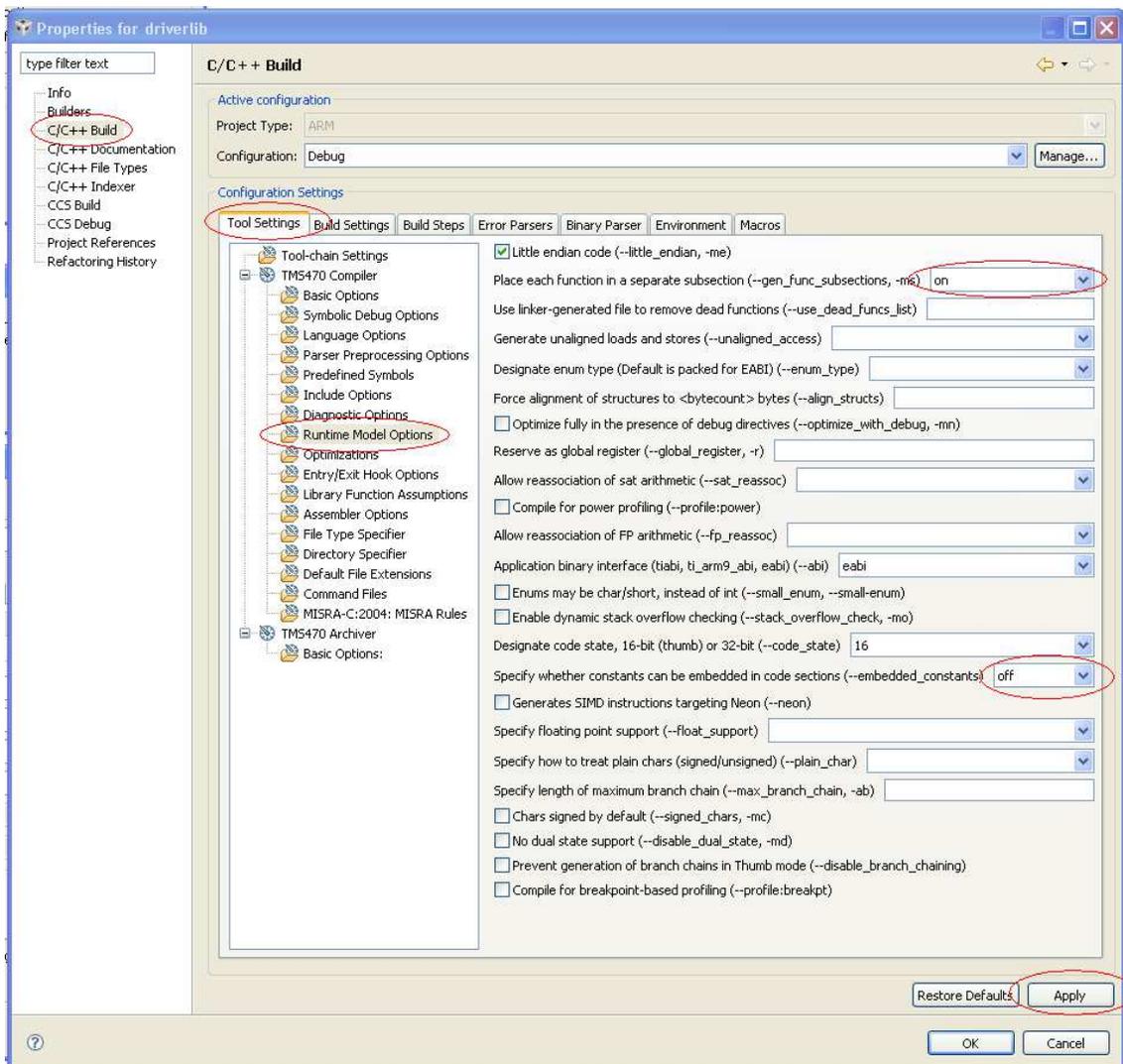

Figure 27. Properties for driverlib (C, C++ Build)

9. Next, as shown in Figure 28, select Predefined Symbols and add additional defines to the Pre-define

    NAME section by clicking on the ![button] button. Add Pre-define NAMEs as shown in the red circle in
    Figure 28. Pay attention to the letter cases and include the following two symbols:
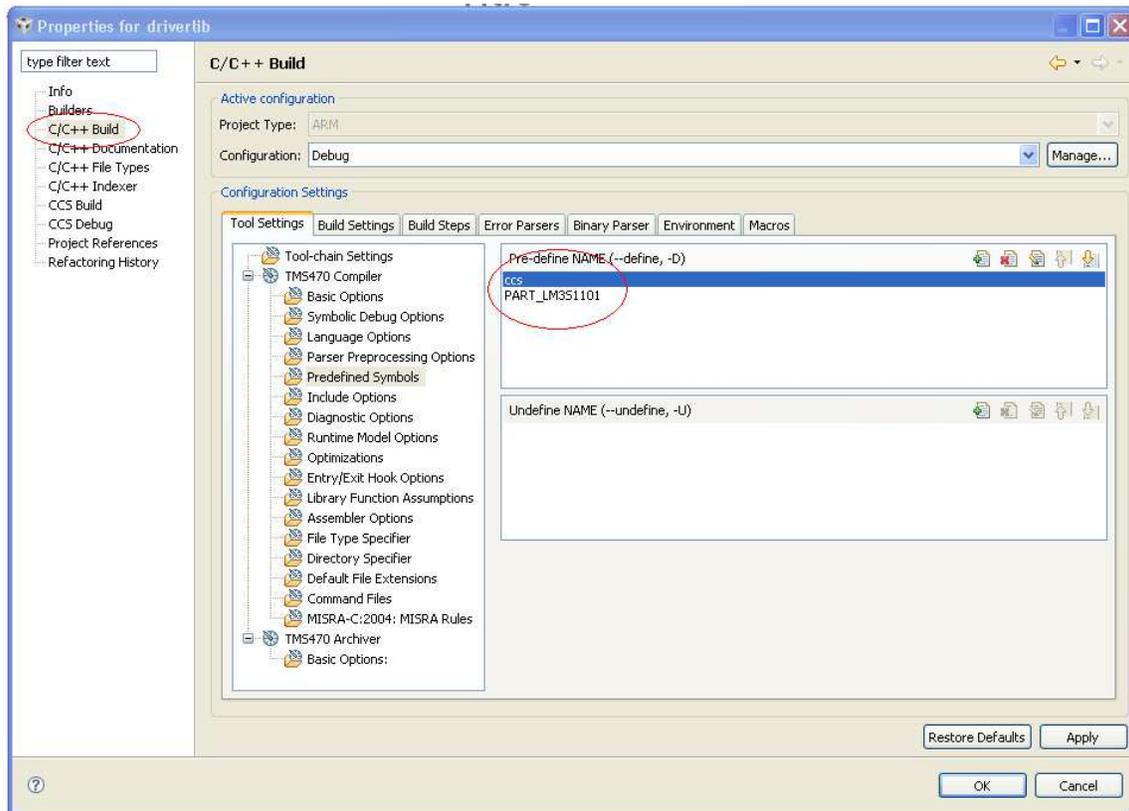
    *   ccs
    *   PART_LM3S1101

Figure 28. Properties for driverlib (Pre-Defined Name)

10. Click on the Apply button → return to the Code Composer Studio main window/ file editor window and
    rebuild the library.

6.2 *Compiling Graphics-Lib*

Add the GraphicsLib project to an existing workspace where the "hello" project already exists.

1. Go to the Project menu in Code Composer Studio → select Import Existing CCS/Eclipse Project →
    click on the Browse button → browse to the *C:\StellarisWare\grlib* location. Import the project file for
    the Graphics Library (grlib) to the current workspace.

    The grlib project will be set as an Active Project. If not, then it can be manually set as an active project
    in the project explorer window pane, by right clicking on grlib project and selecting Set as Active
    Project.

2. Go to the Project menu → select Properties to open the Project properties window as shown in
    Figure 29.

3. Select CCS Build In the General tab. Select the TI v4.9.0 compiler in the Code Generation Tools field.
    After selecting these options, click on the Apply button.
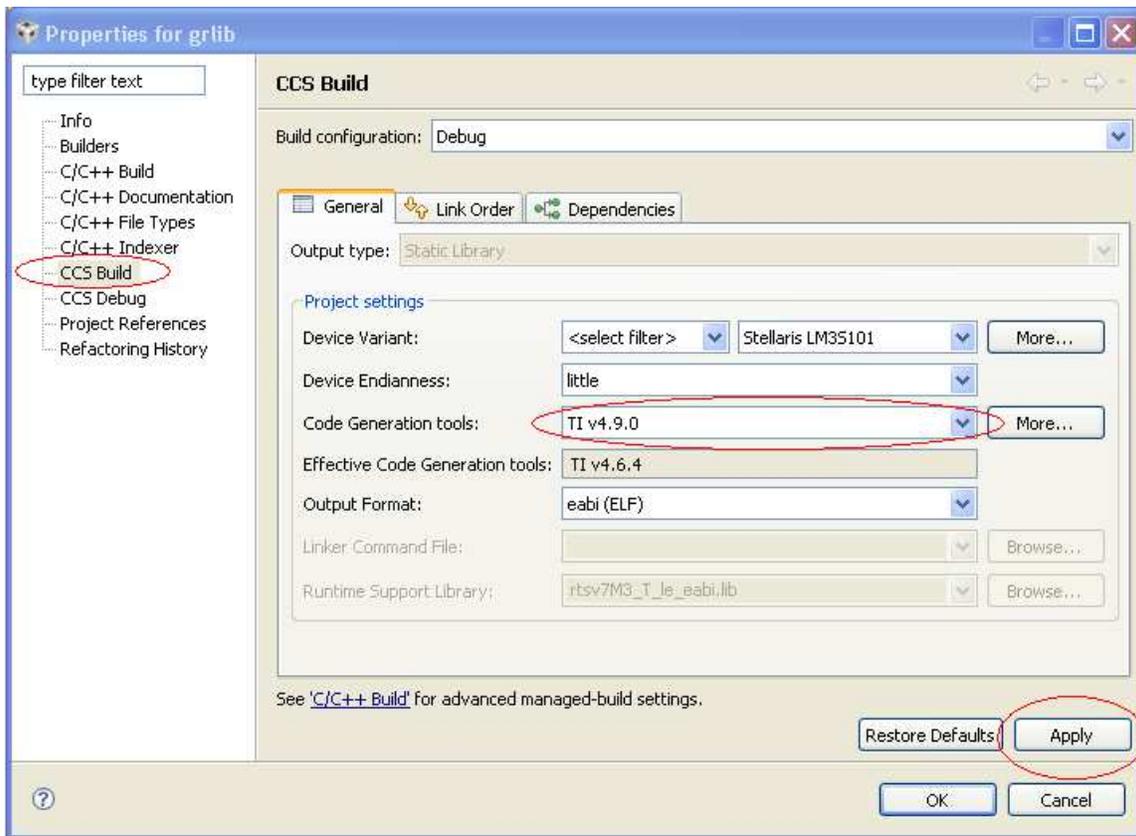
Figure 29. Properties for grlib

4. The Build Configuration Settings window will appear → select options as shown in Figure 30 → click OK button.
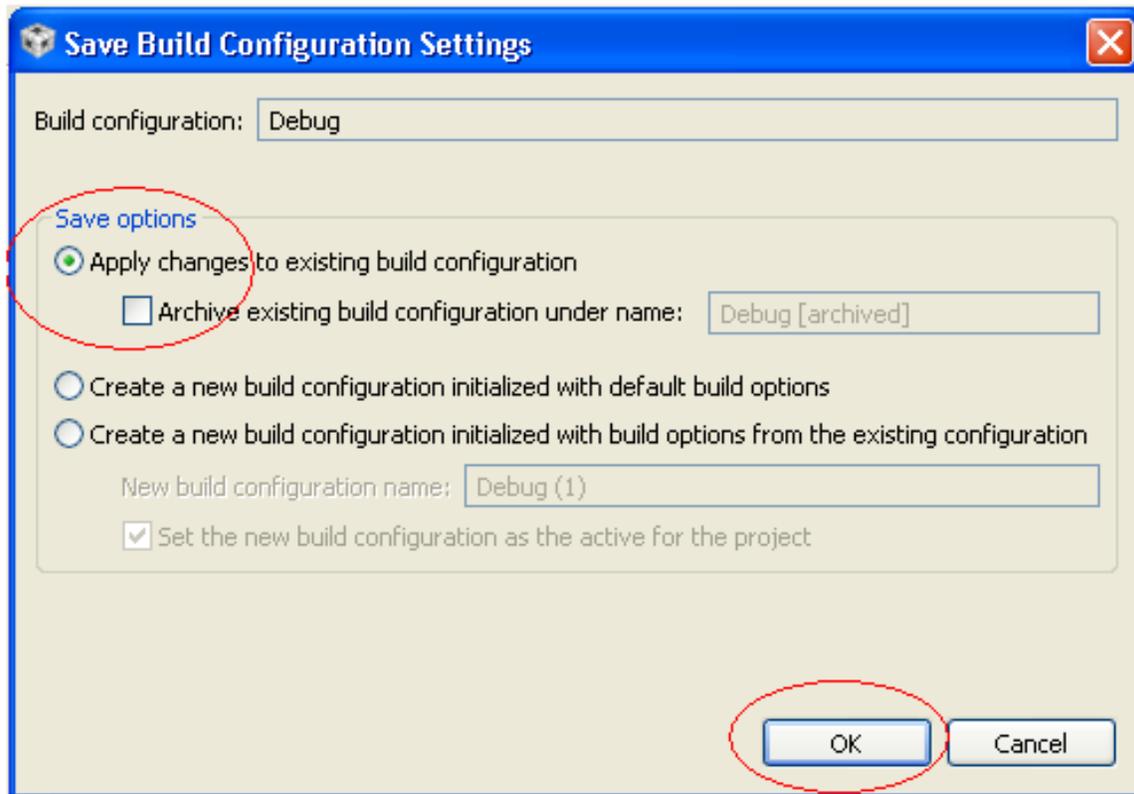
Figure 30. Save Build Configuration Settings (Debug)

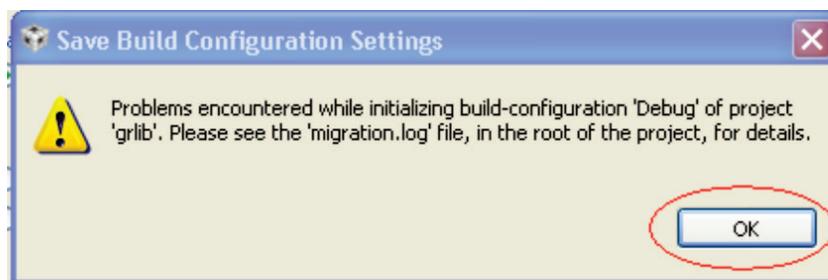5. Click the OK button when the following dialog box appears.

Figure 31. Save Build Configuration Settings (Problems)

6. Click the OK button on the window that appears next (as shown in Figure 32) and return to the Code Composer Studio properties for this project.
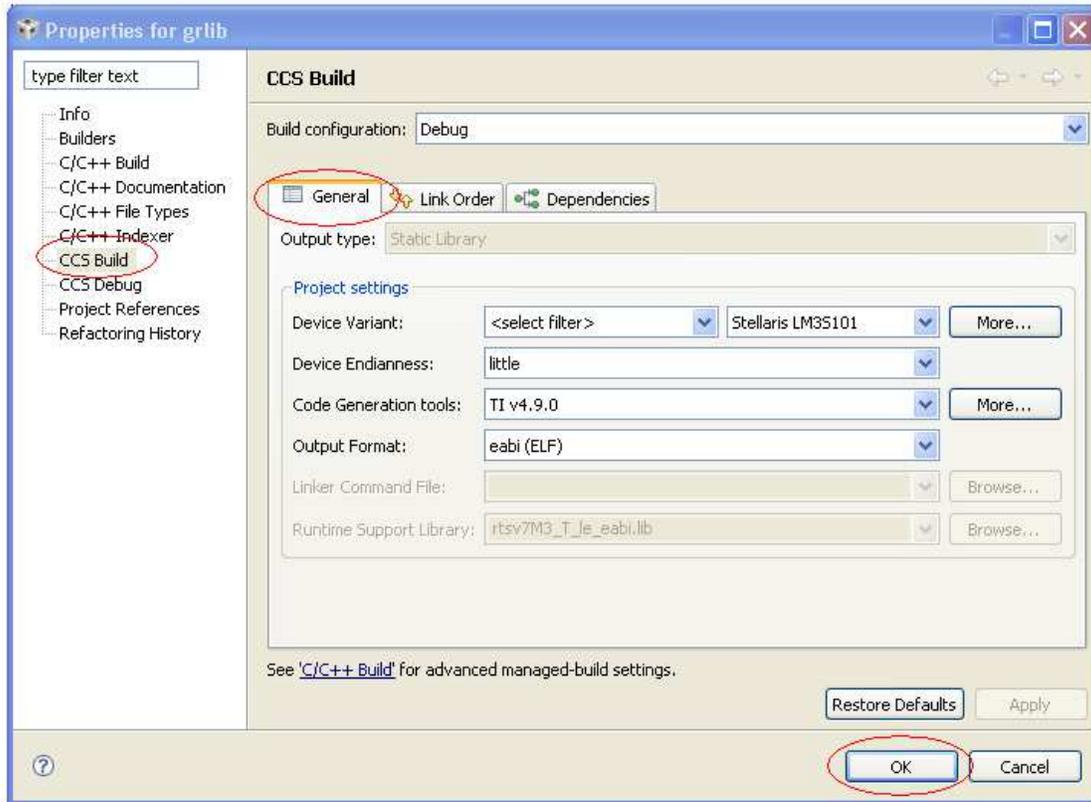
Figure 32. Properties for grlib (Code Composer Studio Build)

Copyright © 2012, Texas Instruments Incorporated

7. Select C/C++ Build under the Tool Settings tab → select the Runtime Model Options in the TMS470 Compiler field. Select options (on/off) as shown in red circles in Figure 33 → click on the Apply button.
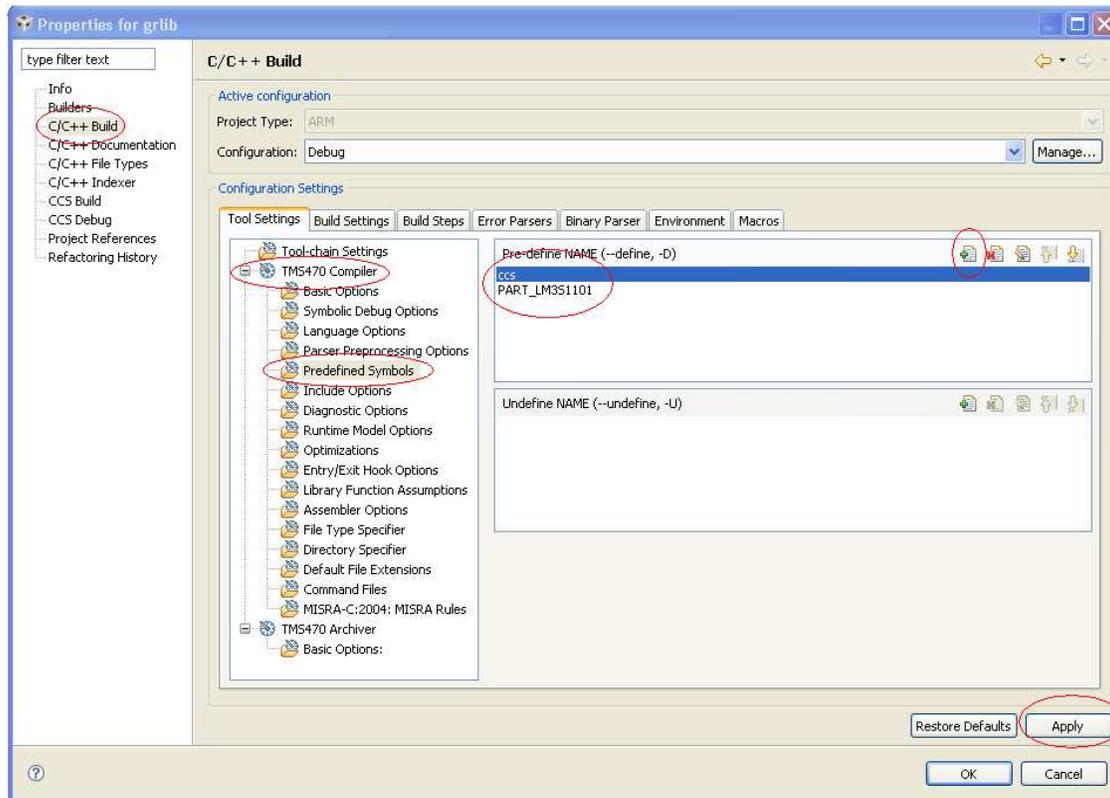

Figure 33. Properties for grlib (C, C++ Build)

Copyright © 2012, Texas Instruments Incorporated

8. Next, as shown in Figure 34 → select Predefined Symbols and add additional defines to the Pre-

    define NAME section by clicking on the [icon] button. Add Pre-define NAMEs as shown in the red circle
    in Figure 34. Pay attention to the letter cases and include the following two symbols:

    *   ccs
    *   PART_LM3S1101
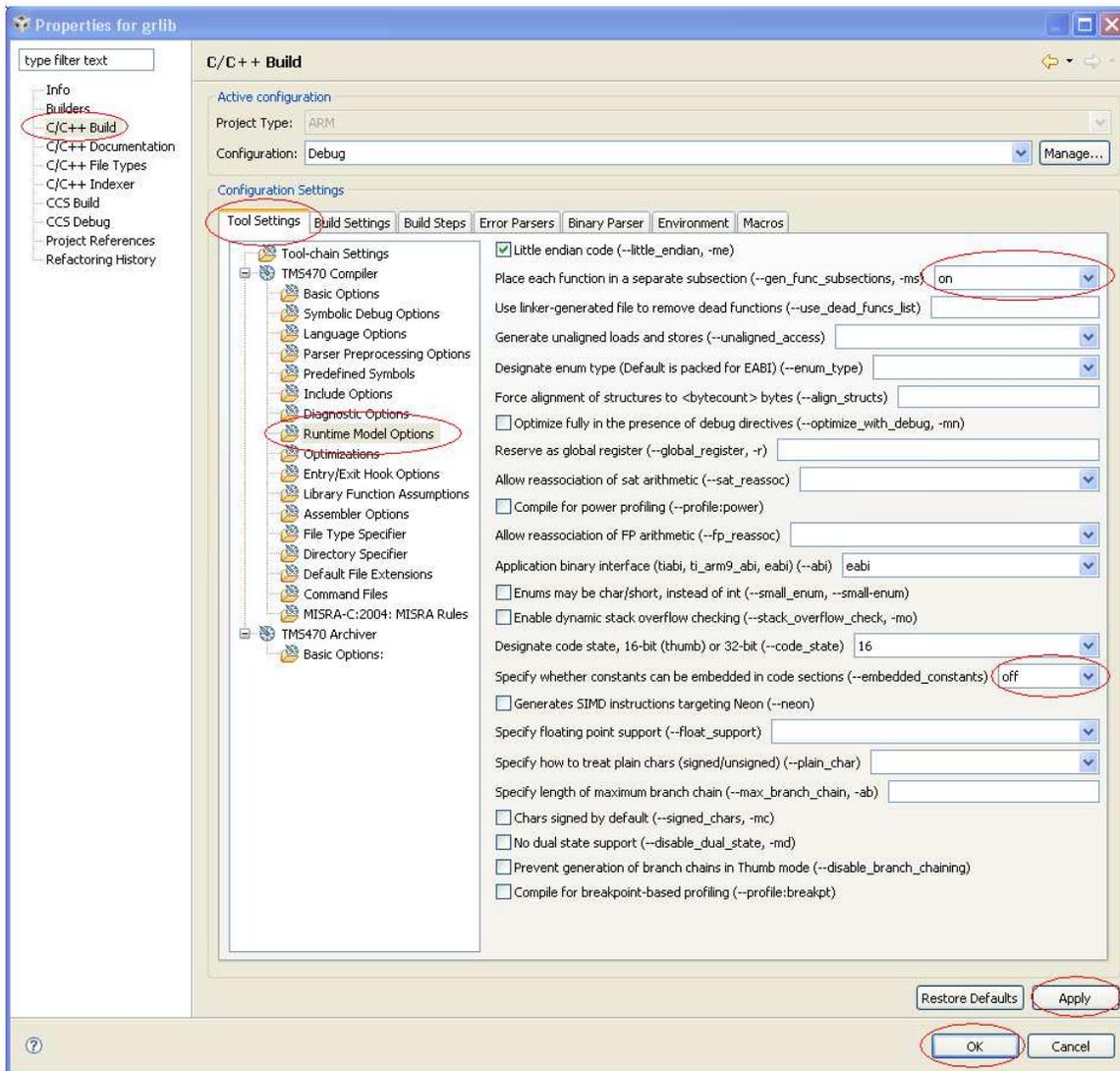
Figure 34. Properties for grlib (Debug)

9. Click Apply → return to the Code Composer Studio main file editor window and rebuild the library.

7   Add Flash Protection Code to the Project and Build It

7.1   Make "hello" Project Active Again

Set "hello" as the active project in the project explorer window pane, by right clicking on "hello" project and selecting Set as Active Project.

Copyright © 2012, Texas Instruments Incorporated

7.2 Include Appropriate Header Files in II

Flash APIs are declared in the *flash.h* header file. In order to use those APIs, include the *flash.h* header file as shown in Figure 35.

```
25 #include "inc/hw_types.h"
26 #include "driverlib/sysctl.h"
27 #include "driverlib/rom.h"
28 #include "grlib/grlib.h"
29 #include "drivers/kitronix320x240x16_ssd2119_8bit.h"
30 #include "drivers/set_pinout.h"
31 #include "driverlib/flash.h"
32
```

Figure 35. Including Header Files

7.3 Add Flash Protection Code to "hello" Project

In order to configure Flash memory as execute only (read protected), appropriate arguments should be passed in *FlashProtectSet()* API. The changes can be committed using *FlashProtectSave()* API. Once committed, the protection settings are permanent and cannot be undone by performing chip reset or power cycle. It is recommended to use *FlashProtectSave()* only after you are sure that your code works the way it is designed to work. The function calls are shown in Figure 36.

Note that the Flash protection settings can be reset to their factory default configuration by performing the debug port unlock sequence using LM Flash Programmer.

```
63  int
64  main(void)
65  {
66      tContext sContext;
67      tRectangle sRect;
68
69      //
70      // Set the clocking to run directly from the crystal.
71      //
72      ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_XTAL_16MHZ |
73                         SYSCTL_OSC_MAIN);
74
75      //
76      // Set Flash Protection as "Flash Execute Only"
77      //
78      FlashProtectSet(0x800, FlashExecuteOnly);
79      FlashProtectSet(0x1000, FlashExecuteOnly);
80      FlashProtectSet(0x1800, FlashExecuteOnly);
81
82
83      //
84      // Make currently programmed flash protection settings permanent.
85      // This is a non-reversible operation; a chip reset or power cycle
86      // will not change flash protection.
87      //
88      FlashProtectSave();
89
90
91      //
92      // Initialize the device pinout appropriately for this board.
93      //
94      PinoutSet();
95
96      //
97      // Initialize the display driver.
98      //
99      Kitronix320x240x16_SSD2119Init();
```

Figure 36. Adding Flash Protection Code

Rebuild the project now.

8 Launch Debugger

Next, you can verify the results that execute, write, and erase-only Flash protection works as expected and the read-protected region in Flash cannot be read.

After the code has been built, download the application to the Flash memory. Begin code execution by clicking on the Run button ![Run button]. Pause code execution by clicking on Pause/ Suspend button ![Pause button].

Launch the debugger to examine the contents of the memory location. The Memory window can be launched by selecting the Memory option in the View menu from the Code Composer Studio menu bar as shown in Figure 37.
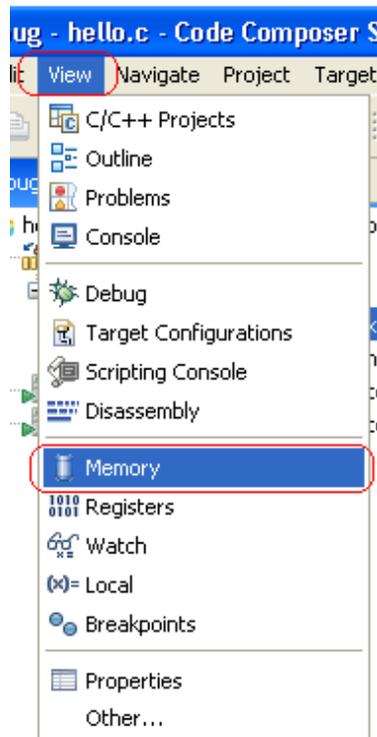
Figure 37. Launching Memory Window

Now, examine the contents of the Flash memory located at addresses in various regions - namely FLASH_1, FLASH_EX, FLASH_2.

First, read the contents in the FLASH_EX region. You will notice that the contents of Flash in the FLASH_EX region, say at address 0x0000 0800, are not readable. The debugger shows "????????" for the memory cells it cannot read. Additionally, some symbols may be shown like GrStringDraw. This is shown in Figure 38.

Figure 38. Memory Window_0x00000800

Upon attempting to access other addresses in the FLASH_EX region, the debugger will prompt an error in the console window as shown in Figure 39. This is expected and indicates that the Flash is read protected for those addresses.

Figure 39. Debugger Console

Second, read the contents in the FLASH_2 (or FLASH_1) region.Notice that the contents of the Flash in FLASH_2 region, say at address 0x0000 2000, are readable as shown in Figure 40.
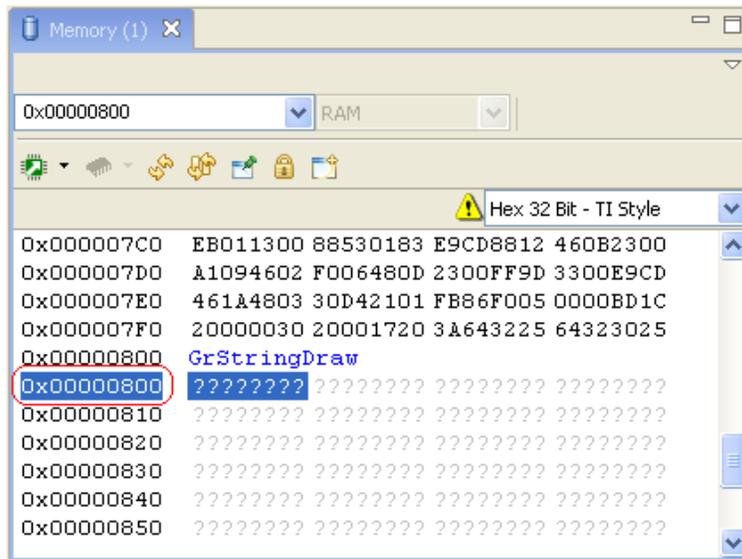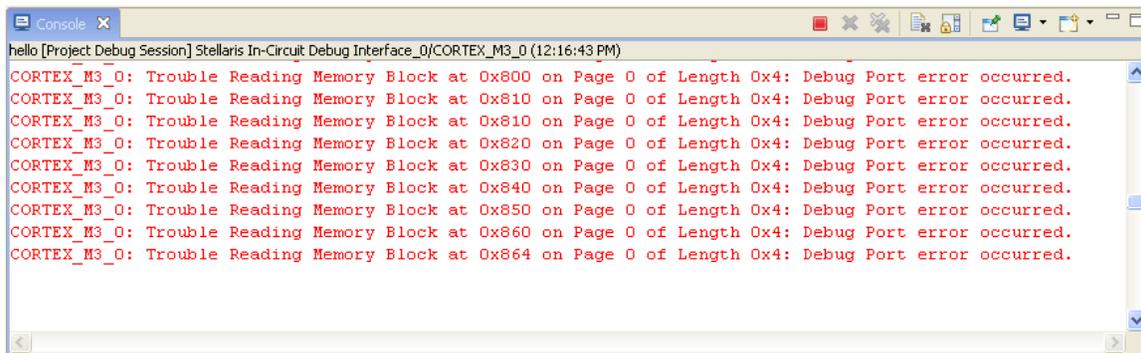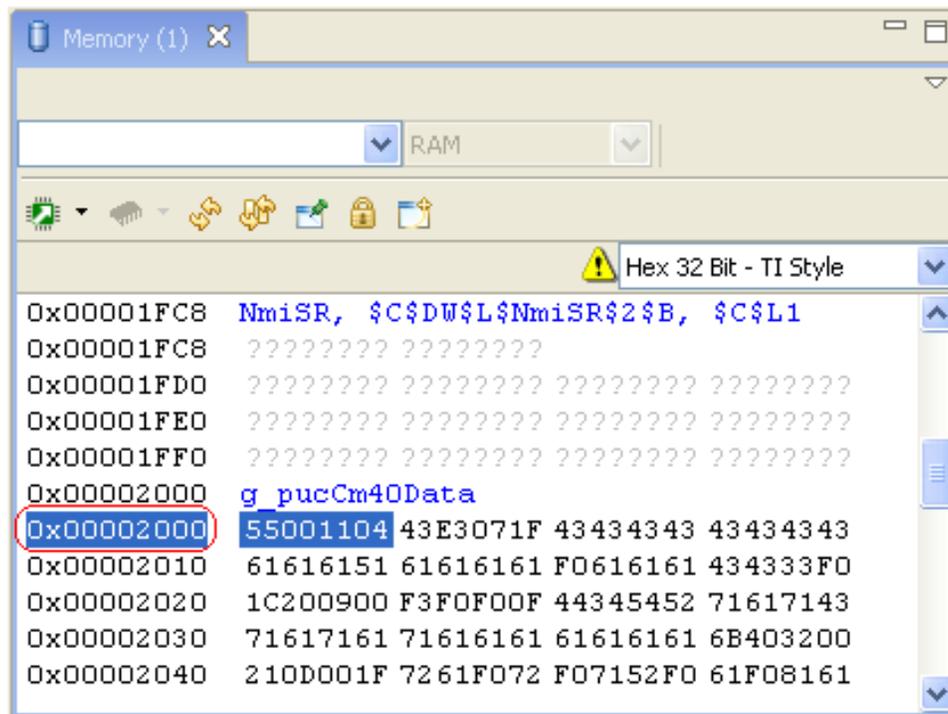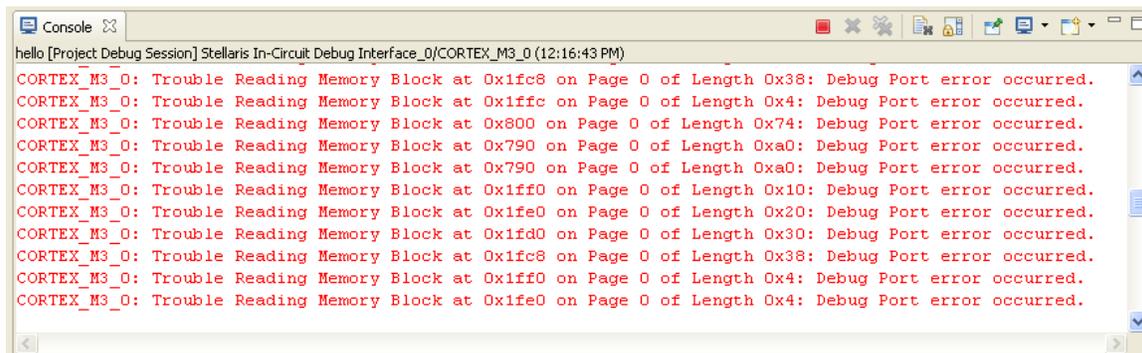
Figure 40. Memory Window_0x00002000

Last, note that upon attempting to access other addresses in the FLASH_EX region, the debugger will prompt an error in the console window as shown in Figure 41. This message is expected and indicates that the Flash is read protected for those addresses.

Figure 41. Debugger Console_2

9 Conclusion

Using a simple "hello" example from StellarisWare, it has been demonstrated how execute, write, erase-only Flash protection can be used on Stellaris microcontrollers to enable developers to protect their code and IP in the end application while providing the flexibility to upgrade the firmware. Flash protection has been implemented by creating a read-only region in the Flash memory to prevent code from being read but executed (the memory block may be written, erased, or executed but not read) using TI's Code Composer Studio v4.2.3 and Code Generation Tools v4.9 that offers the advantage of separating the literal dump and executable code.

> **NOTE:** Once committed (saved), the Flash protection settings can not be changed or undone by power-cycling or resetting the MCU. Performing the Debug Port Unlock (JTAG toggle mass erase sequence) using LM Flash Programmer will erase the entire Flash memory and reset the Flash protection settings to factory default.

10   References

- *Stellaris LM3S9B96 Microcontroller Data Sheet* (SPMS182)
- Code Composer Studio v 4.2.3 with Code Generation Tools v4.9 can be downloaded from the following URL: http://www.ti.com/ccs
- StellarisWare Package can be downloaded from the following URL: http://www.ti.com/stellarisware
- FTDI-based ICDI driver can be downloaded from the following URL: http://www.ti.com/tool/lm_ftdi_driver

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46C and to discontinue any product or service per JESD48B. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information 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.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have *not* been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components which meet ISO/TS16949 requirements, mainly for automotive use. Components which have not been so designated are neither designed nor intended for automotive use; and TI will not be responsible for any failure of such components to meet such requirements.

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 Mobile Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2012, Texas Instruments Incorporated