应用报告
ZHCA504A–August 2012–Revised August 2012

在使用 Code Composer Studio 的 Stellaris 微控制器上使用只执行、只写入和只擦除闪存保护

Ashish Ahuja

摘要

微控制器闪存存储器中的保护代码和 IP 一直是系统设计人员的一个重要考虑因素。 Stellaris® 微控制器特有一个代码保护机制，此机制使得开发人员能够在最终应用中保护他们的代码和 IP，同时又使用一个引导加载程序来提供固件升级的灵活性。 这份应用报告描述了使用闪存保护来防止代码被读取而又使其能够执行的方法（例如，存储器块可被写入、擦除或执行但不能被读取），此方法使用 Code Composer Studio™ v4.2.3 实现。

内容

图片列表

图表列表

1 简介

Stellaris 微控制器提供如表 1中所示的不同的闪存存储器保护模式。 本文档专注于只执行、只写入和只擦除保护模式，并解释了如何使用在 StellarisWare®中提供的示例中的一个来执行这些模式。

注: 要获得与只读闪存保护相关的更多细节，请见*Stellaris LM3S9B96 微控制器数据表*(SPMS182)中的*闪存存储器保护*。

当闪存存储器保护程序使能 (FMPPEn) 和闪存存储器保护读取使能 (FMPREn) 寄存器内的相应位被分别设定为 1 和 0 时，您能够擦除和写入闪存，并且 Cortex 内核可从闪存执行代码。 总的来说，大多数编译程序将转储文字（常量和数据）放置在可执行代码中。 当这样一个代码位于系统内存的只可执行（并受到读取保护）块中时，不能读取转储文字。 因此，应用将不能正常执行。 但使用只执行、只写入和只擦除闪存保护时，将代码编译很重要，这样转储文字就不会驻留在代码的可执行部分中，这也许需要在建立过程中使用特别的编译器。 与德州仪器 Code Composer Studio 的代码生成工具（至少为 v4.9 版本）一起提供的编译器具有此功能。

表 1. 闪存保护模式

保护模式	FMPPEn	FMPREn	执行	读取	写入和擦除
只执行	0	0	√	x	x
只进行执行、写入和擦除	1	0	√	x	√

表 **1.** 闪存保护模式 **(continued)**

保护模式	FMPPEn	FMPREn	执行	读取	写入和擦除
只执行和只读	0	1	√	√	x
无保护	1	1	x	x	x

这份应用报告使用一个针对 DK-LM3S9B96 套件的 'hello' 示例来演示只执行、只写入和只擦除闪存保护。

2 要求

您将需要：

- 一台安装了以下软件并运行这些软件的计算机：
  - Windows XP 或 7 操作系统
  - 可从以下 URL 内下载的具有代码生成工具 v4.9 的 Code Composer Studio v4.2.3：http://www.ti.com/ccs
  - 可从以下 URL 下载的 StellarisWare 软件：http://www.ti.com/stellarisware
  - 可从以下 URL 下载的用于 Stellaris 虚拟 COM 端口，Stellaris 评估板 A 和 B 的 Stellaris USB 驱动程序：http://www.ti.com/tool/lm_ftdi_driver
- DK-LM3S9B96 Stellaris 开发套件
- 在一端具有标准 A 类插头，而在另一端具有迷你 B 插头（5 引脚）的 USB 线缆

3 过程

图 1图示了本应用报告所包含的关键步骤。 每个步骤将随后在以下部分中进行解释。

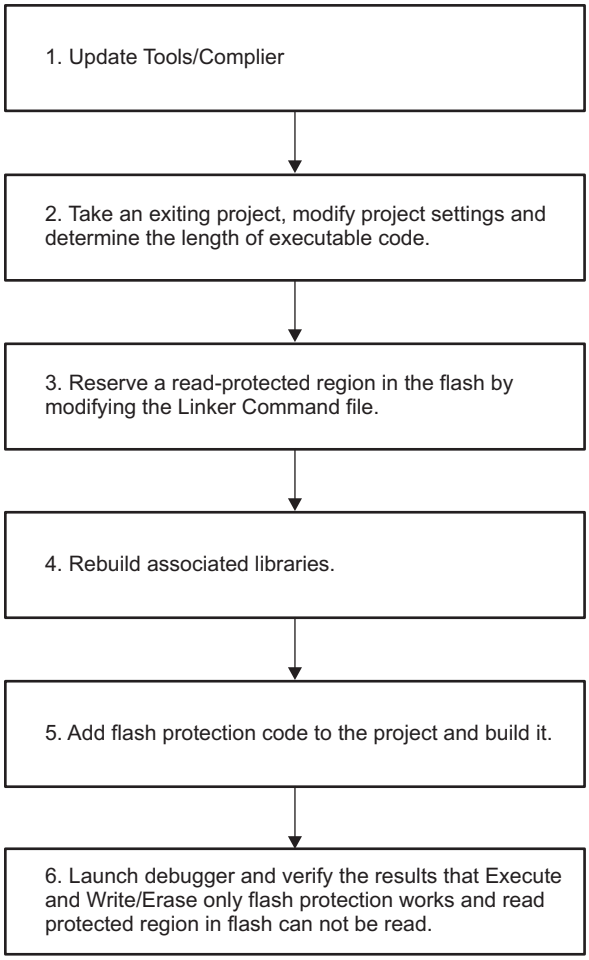


图 **1.** 关键步骤的流程图显示序列

3.1 更新工具和编译器

3.1.1 检查当前安装的代码生成工具版本

代码生成工具 v4.9 并不是当前可用的 Code Composer Studio v4.2.3 安装包的标准组件

要确定在您机器上安装的代码生成工具的当前版本：

1. 启动 Code Composer Studio。
2. 前往 Help 菜单。
3. 单击 Software Updates（软件升级）。
4. 前往 Manage Configuration（管理配置）选项。 将出现以下窗口。

编译器版本已经在图 2中标出。

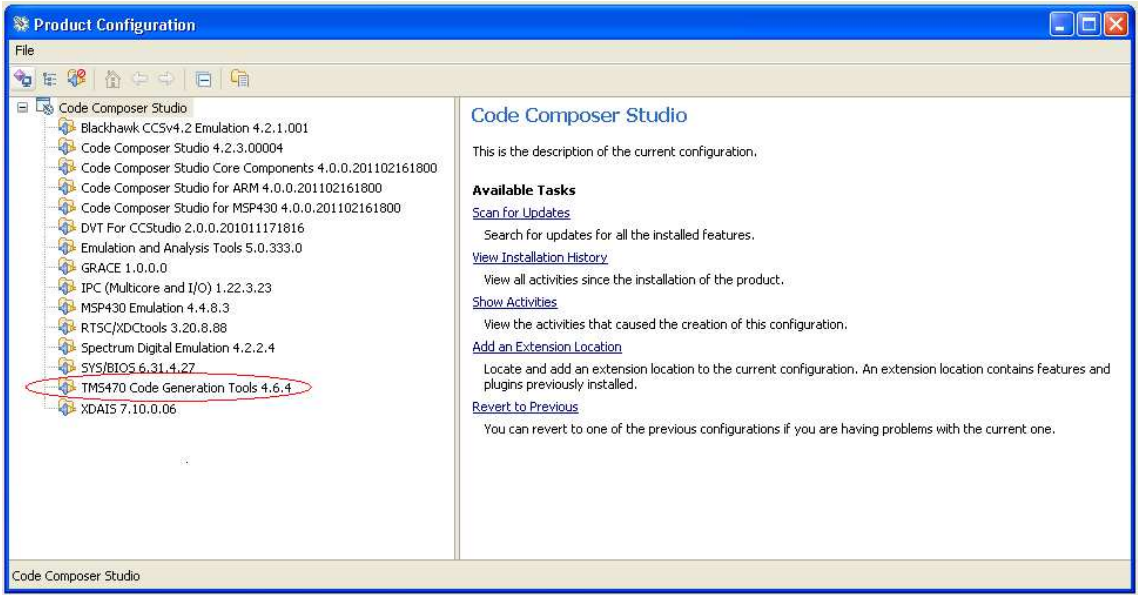


图 **2.** 确定已安装组件的版本

如果您的机器上未安装代码生成工具 v4.9，从3.2 节to update your current compiler to v4.9 or else proceed to（将您当前的编译器升级至 v4.9 或进入）3.3 节开始按照指令进行操作。

3.2 安装并更新代码生成工具至版本 v4.9

在 Code Composer Studio 中：

1. 前往 Help 菜单。
2. 单击 Software Updates。
3. 单击 Find（查找）和 Install（安装）。 将出现一个 Install 和 Update 向导。
4. 选择 Search for new features（搜索新特性）来进行安装。
5. 单击 Next（下一步）按钮。
6. 在下一个出现的屏幕上选择 Code Generation Tools Updates（如图 3中所示）。
7. 单击 Finish（完成）按钮。

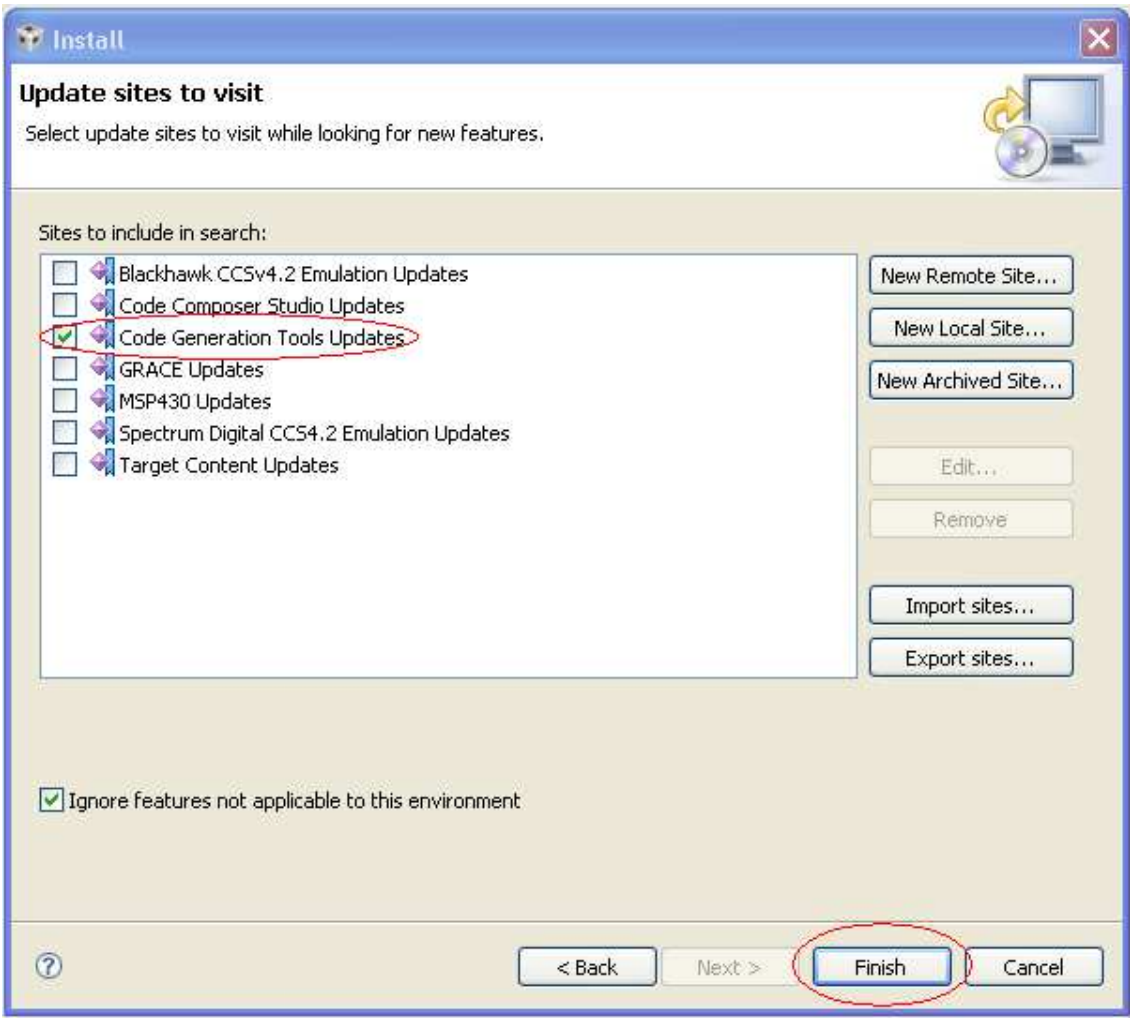


图 **3.** 选择 **Update** 选项。

Code Composer Studio 搜索更新并如图 4中所示打开一个可用更新的列表。

1. 选择 TMS470 Code Generation Tools 4.9.0 → Other → Code Generation Tools Update。
2. 单击 Next 并进入 Feature License（特性许可）屏幕。

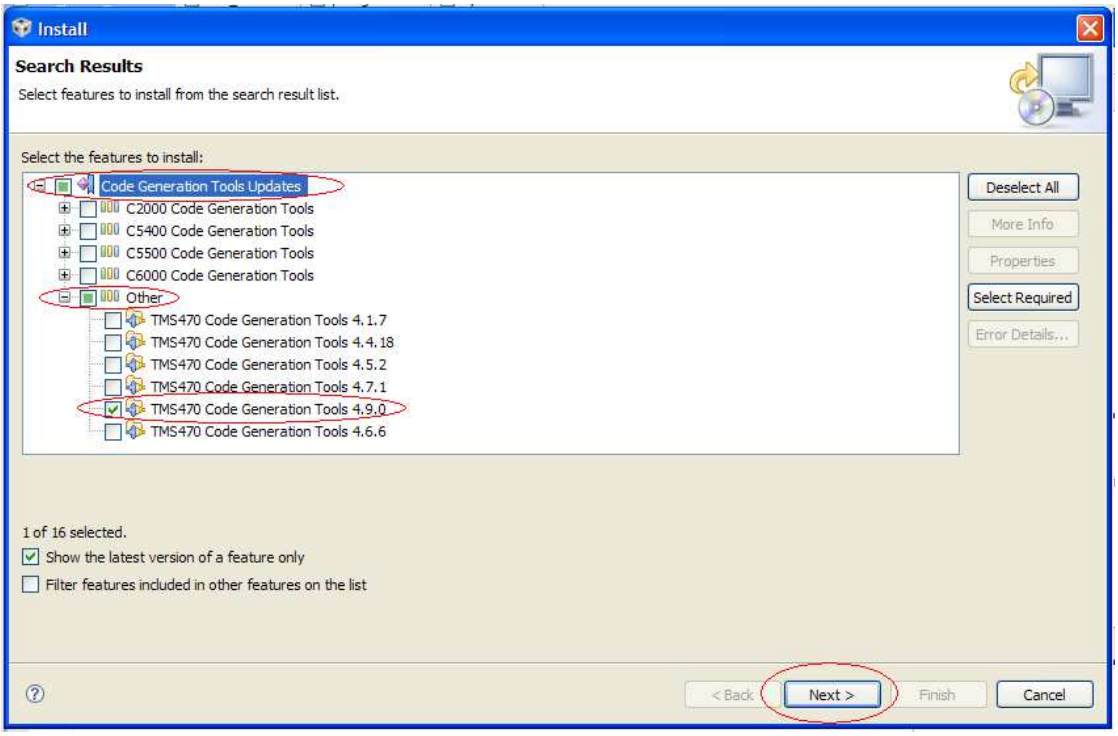


图 4. 选择 Update Components（升级组件）

3. 如图 5所示，阅读并接受许可协议并单击 Next 按钮。

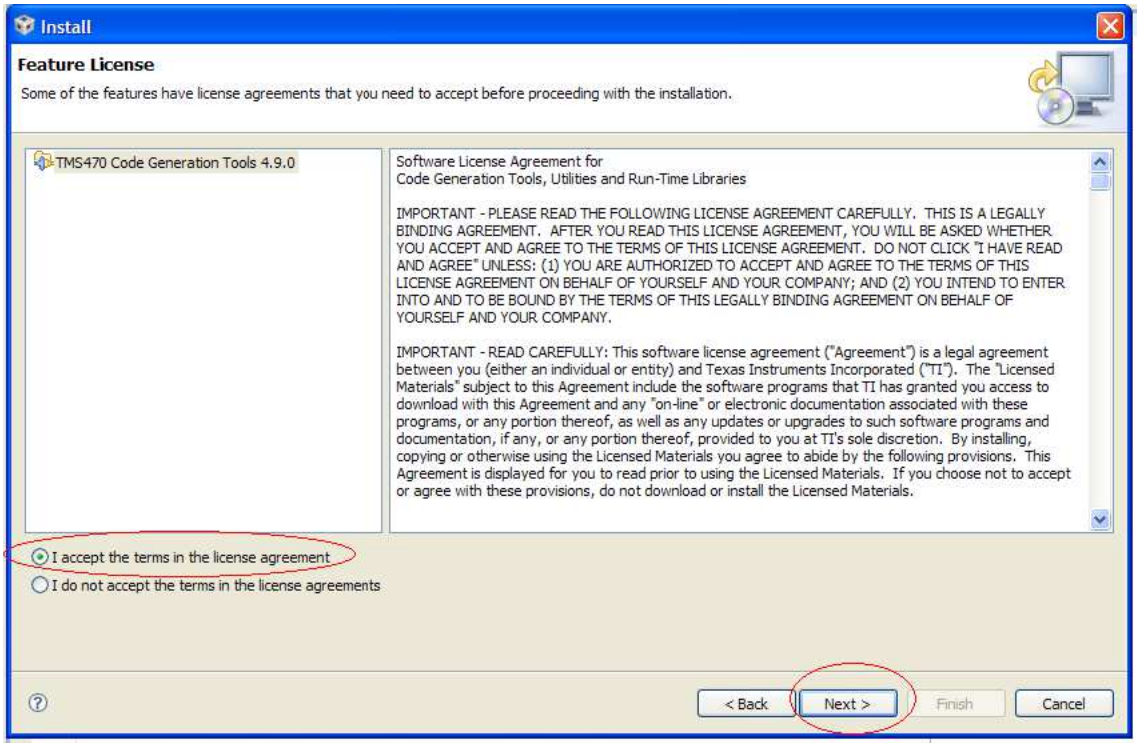


图 **5.** 安装 **Window_Accept** 术语

4. 如图 6所示，单击 Finish。

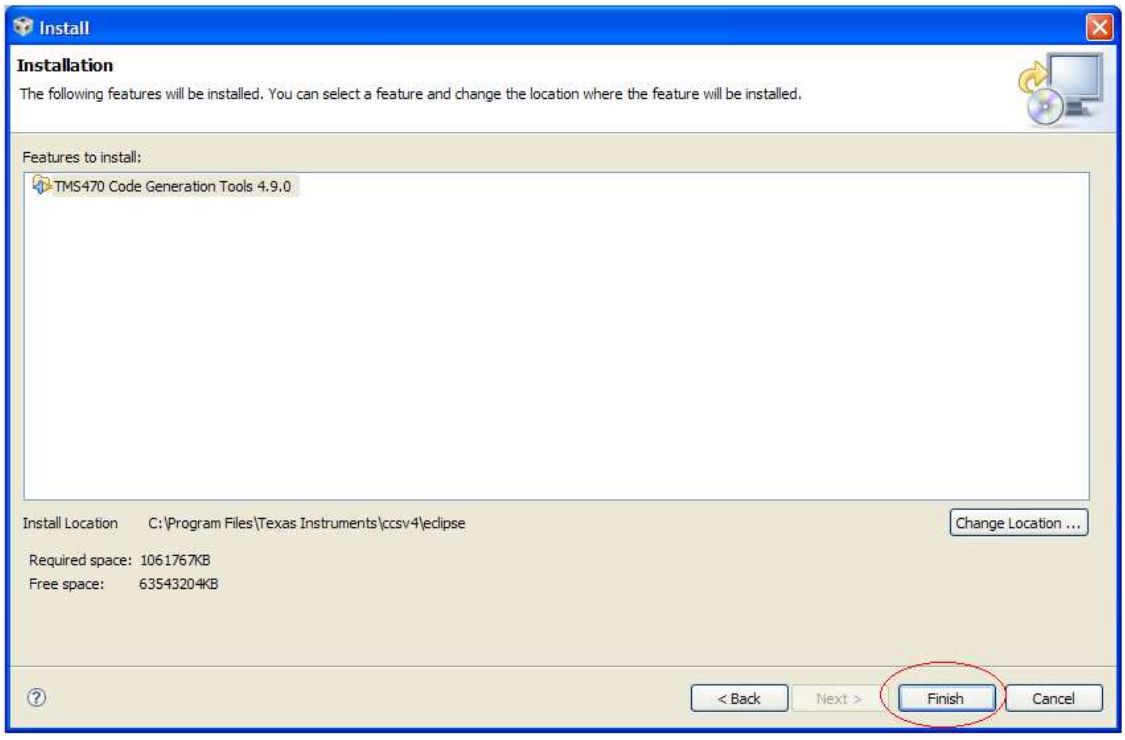


图 **6.** 安装 **Window_Finish**

5. 如图 7所示，单击 Install 按钮。 下一步出现验证屏幕。

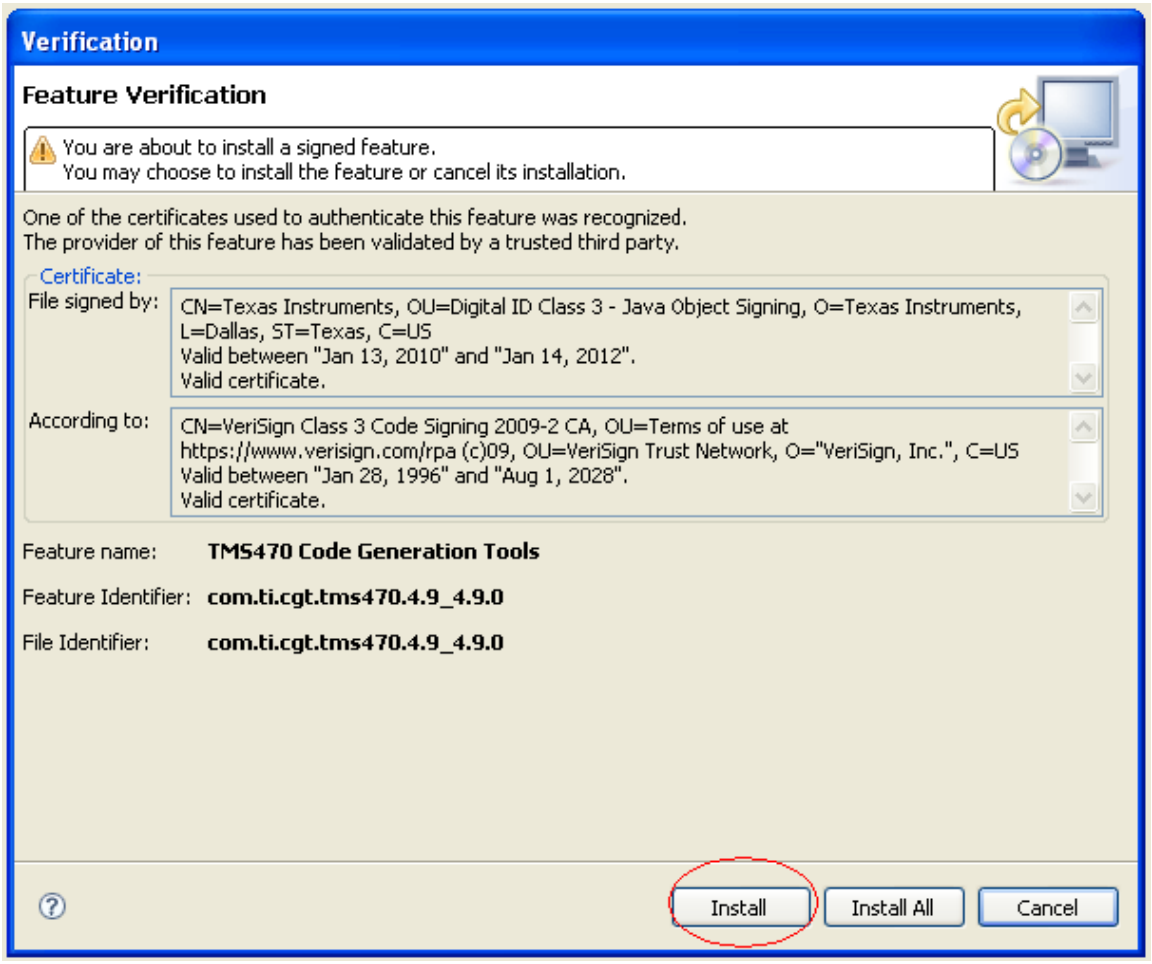


图 **7.** 验证窗口

6. 在更新成功安装后，下面的弹出窗口显示（请见图 8）。

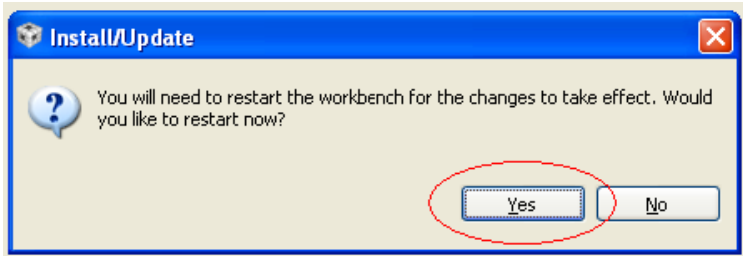


图 **8.** 安装和更新对话框

7. 单击 Yes 按钮并重新启动 Code Composer Studio。

*3.3 请确认当前安装的代码生成工具的版本为 **v4.9**。*

在启动 Code Composer Studio 时：

1. 前往 Help 菜单。
2. 单击 Software Updates。
3. Manage Confirmation 选项。 将出现以下窗口。

4. 请确定代码生成工具 v4.9 已经被成功安装，如图 9中的红圈所示。

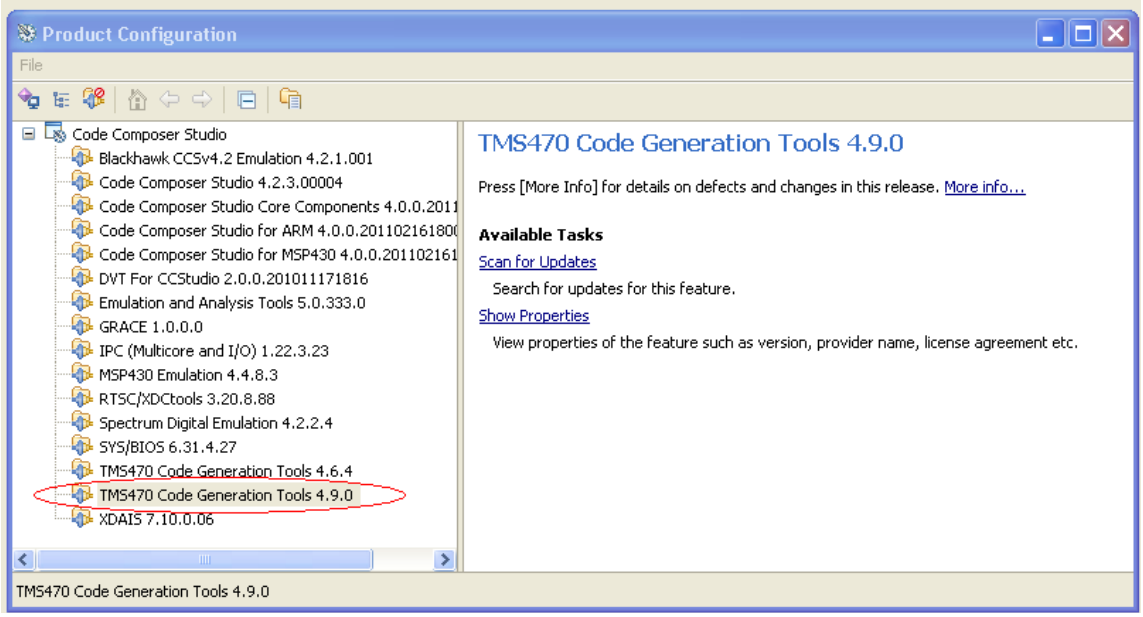


图 9. 检查 **Update Options**

4 修改现有的项目设置并确定可执行代码的长度

在以下的部分中，使用一个来自 StellarisWare 的现有示例。 使用一个方法来修改和编译项目设置，使得转储文字不会驻留在代码的可执行部分。 这个协议使您能够使用只执行、只写入和只擦除闪存保护，这为您提供了写入、擦除和执行闪存的功能，但是您不能读取闪存。

4.1 从 StellarisWare 中在 Code Composer Studio 中建立一个现有项目 (hello.c)

1. 将 DK-LM3S9B96 板连接到计算机。
2. 启动 Code Composer Studio
3. 前往 Project 菜单。
4. 选择 Import Existing CCS/Eclipse Project（导入现有的 CCS / 之前的项目）。
5. 单击 Browse 按钮，浏览*C:\StellarisWare\boards\dk-lm3s9b96\hello*位置，或者浏览 'hello' 项目在您计算机中位置。
6. 将项目文件导入一个工作区。
7. 使用缺省设置建立、调试和下载项目。 此项目应该被成功建立而不会出现任何错误，应用的执行也应该如预期的那样正常。

4.2 修改项目设置

在 Code Composer Studio 中：

1. 前往 Project 菜单
2. 选择 Properties 来打开如图 10中所示的项目属性窗口。
3. 选择 Code Composer Studio Build
4. 在 General 标签页中，在 Code Generation Tools 域中选择 TI v4.9 编译器。
5. 为 Runtime Support Library（运行时间支持库）选择 <automatic>。
6. 在选择了这些选项后，单击 Apply（应用）按钮。

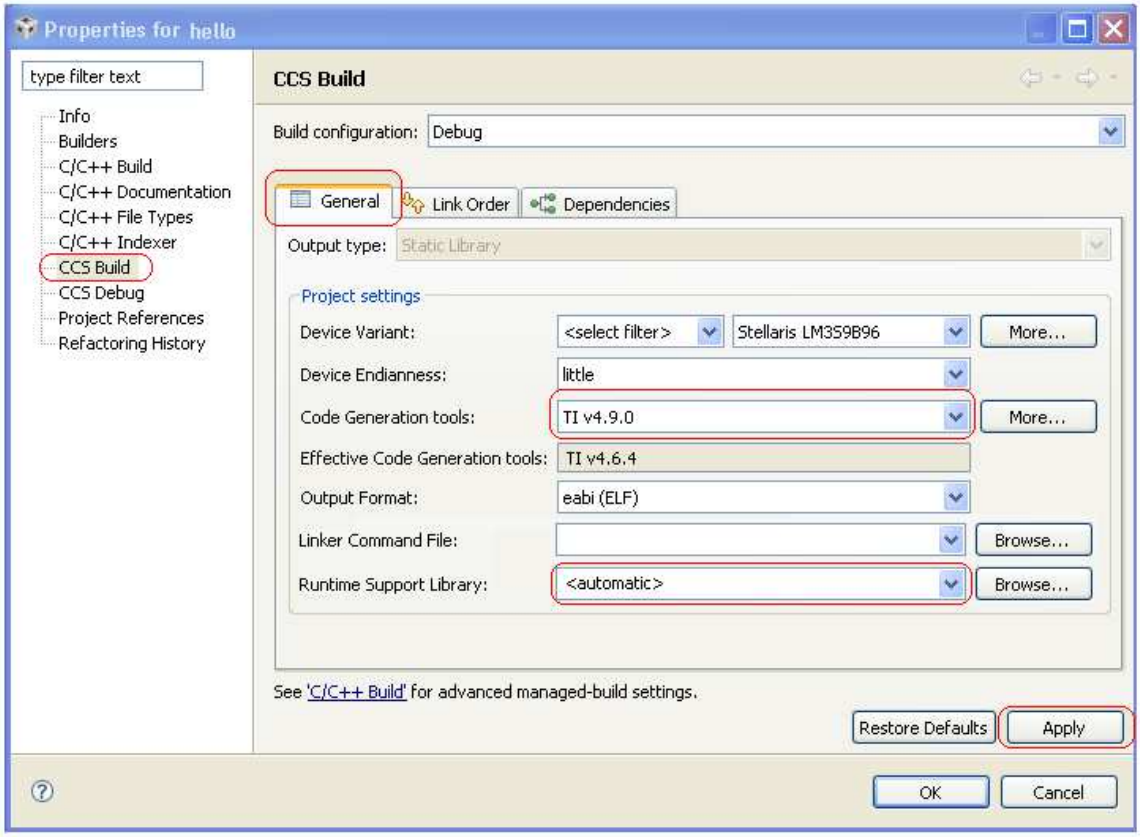


图 **10. 'hello'**的属性

7. 将出现 Build Confirmation Settings（建立配置设置）窗口。 如图 11所示，选择选项并单击 OK 按钮。

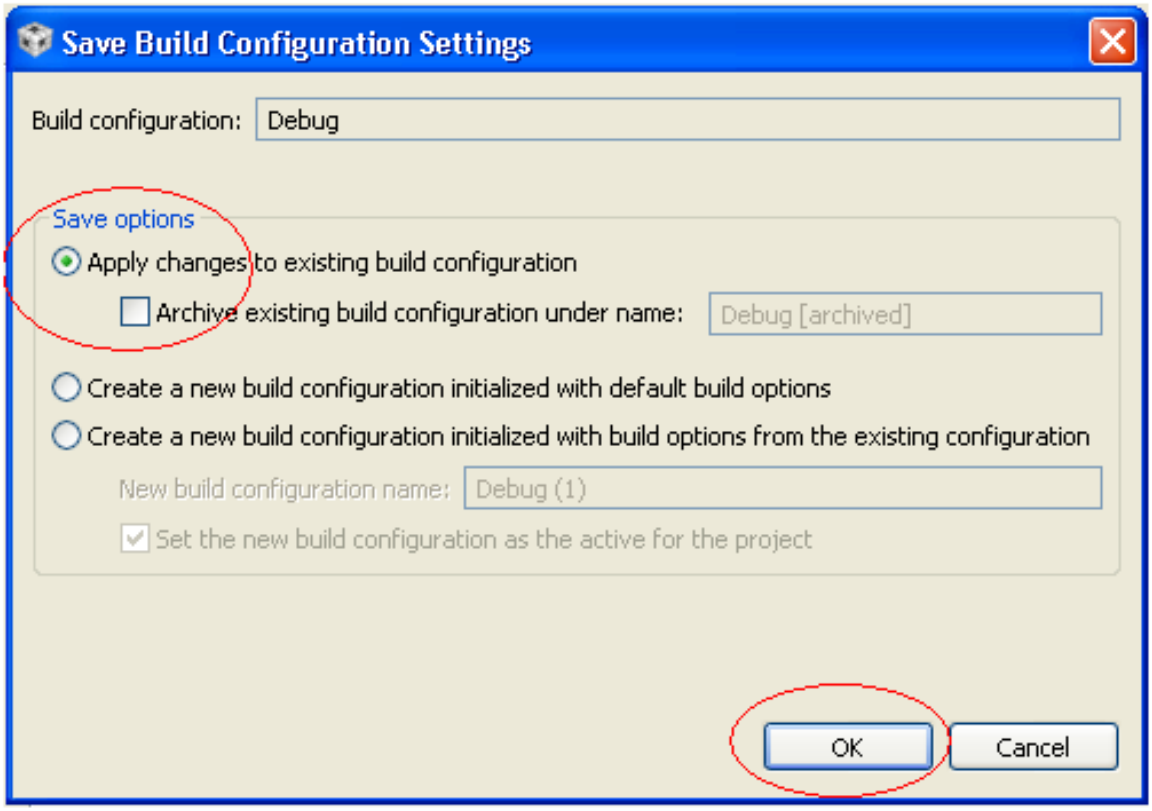


图 **11.** 保存 **Build Configuration Settings**

8. 当如图 12中所示出现对话框时，单击 OK 按钮。

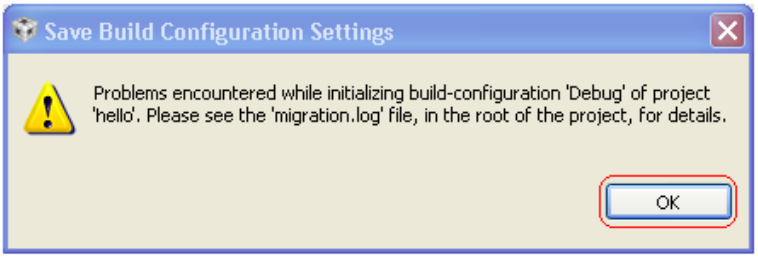


图 **12.** 保存 **Build Configuration Settings**

9. 在下一个出现的窗口中单击 OK 按钮（如图 13所示）并为这个项目返回 Code Composer Studio 项目属性。

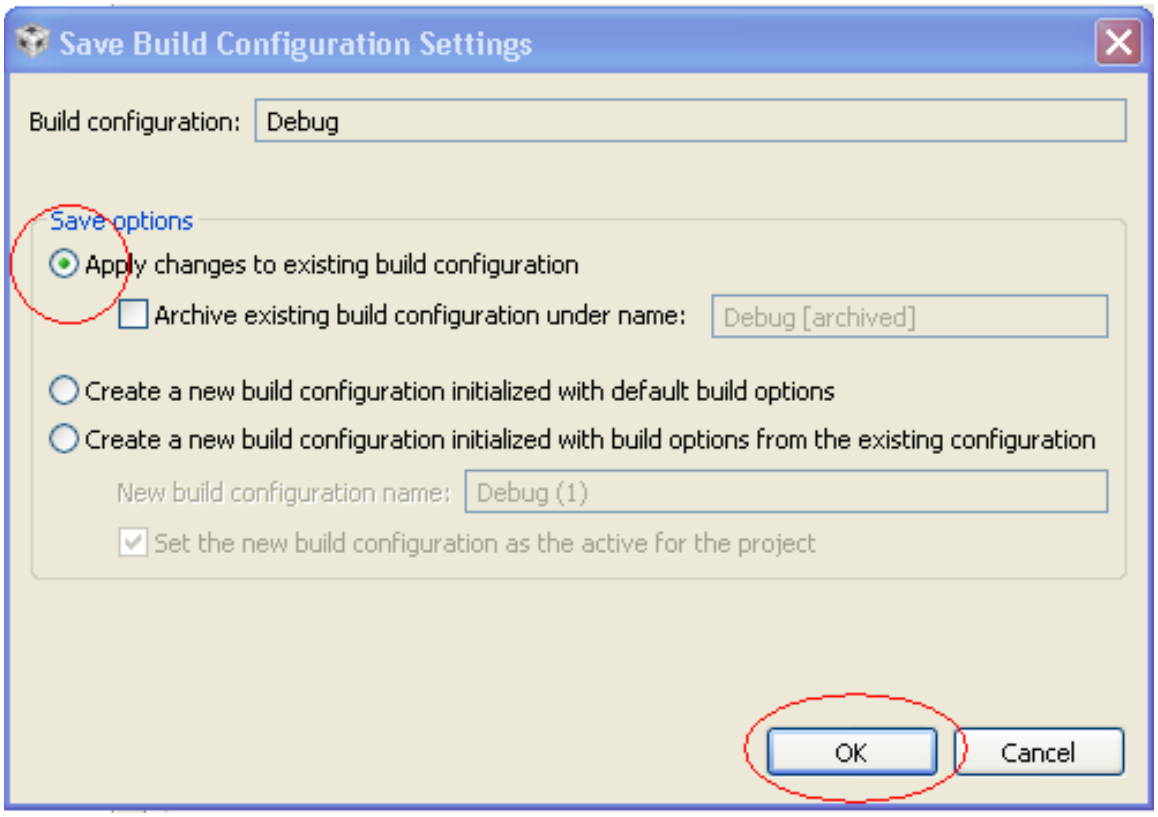


图 **13.** 针对 **'hello'** 的属性 **- Code Composer Studio** 建立

10. 在 Tool Setting（工具设置）标签页下选择 C/C++ Build
11. 在 TMS470 编译器域内选择 Runtime Model Options

12. 如图 14中的红圈所示选择选项（开和关），→单击 Apply 按钮。

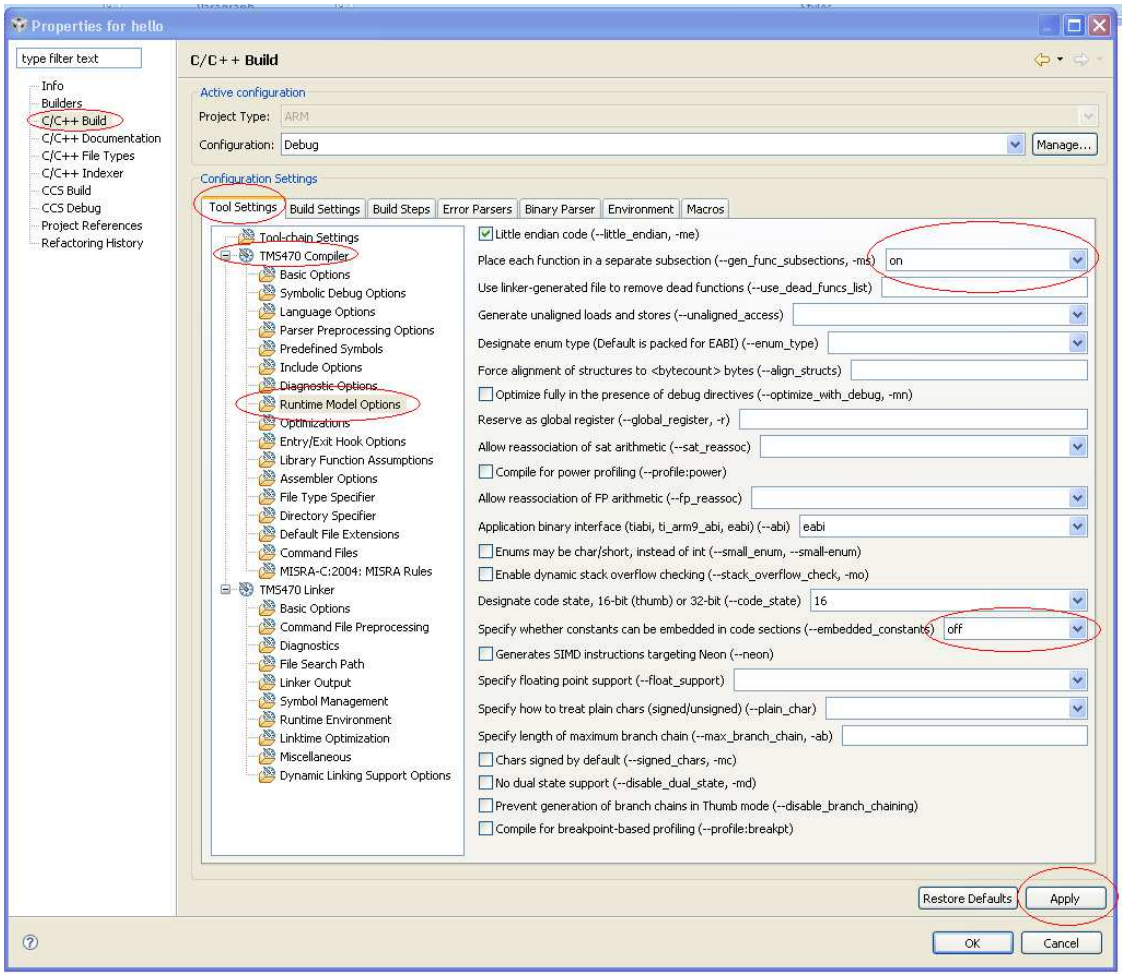


图 **14.** 针对 **'hello'** 的属性 **- C/C++ Build**

如图 15中所示，选择 Predefined Symbol（预先定义的符号）并通过单击 按钮来将附加的定义添加到 Predefined NAME 部分。 如图 15中的红圈所示，添加 Predefined NAMEs（预先定义的名称）。 请注意字母的大小写；并包括以下三个符号：

- TARGET_IS_TEMPEST_RB1
- ccs
- PART_LM3S9B96

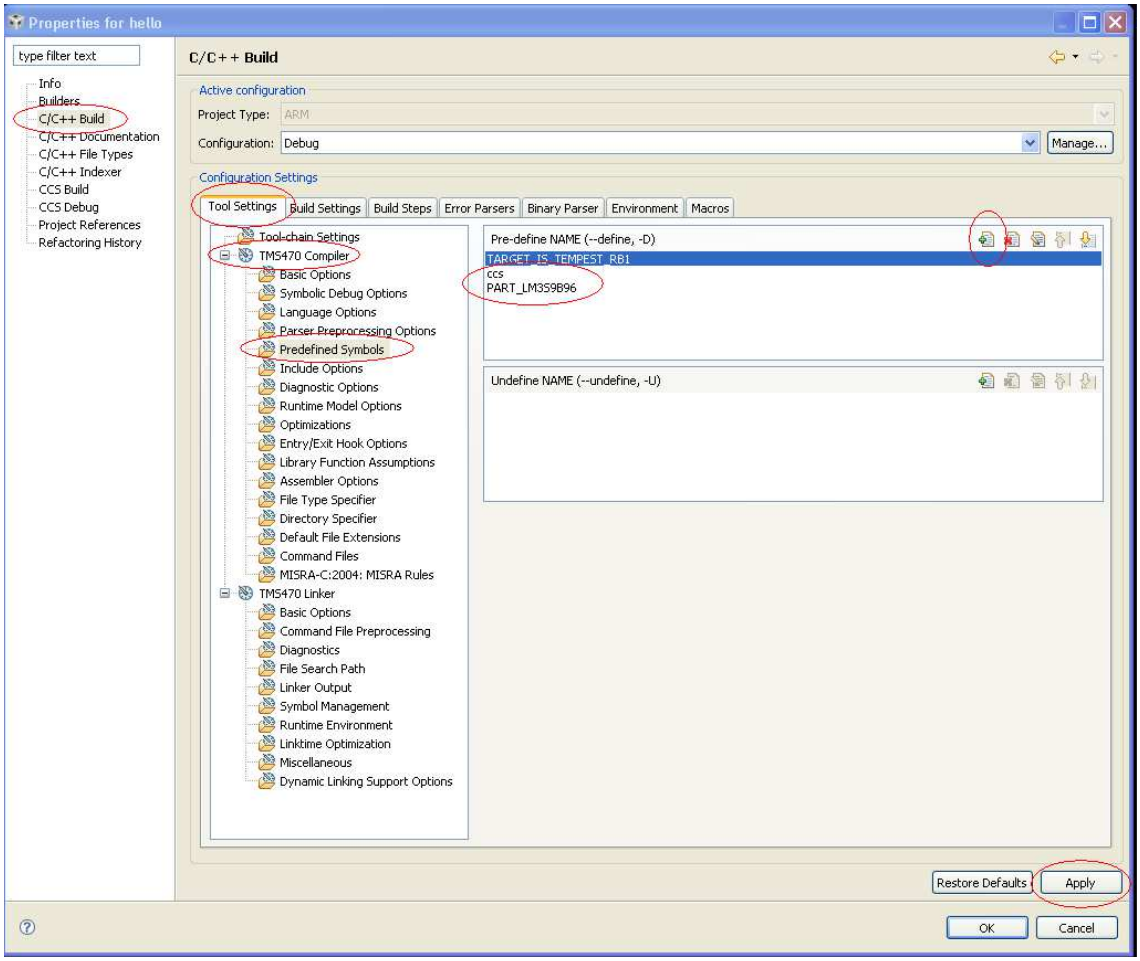


图 15. 针对 **'hello'** 的属性 **- Predefined Name**

13. 单击 Apply → 返回至 Code Composer Studio 主窗口/文件编辑器窗口。

4.3 重建项目

通过在 Project 菜单中选择 Rebuild All 选项来重建代码。 项目应该被成功建立而不会发生任何错误。

4.4 通过检查映射文件的内容来确定需要保护多少闪存

在这个示例中，通过使其成为只执行、只写入和只擦除来保护闪存内代码的可执行部分。 此代码的剩余部分，诸如矢量表、常量和初始化值无需保护。

为了保护可执行代码，首先检查映射文件以确定可执行代码的长度。 Code Composer Studio 内的映射文件的扩展名为 .map 并且 (help.map) 可在 Code Composer Studio 项目文件夹的调试目录中找到，此文件夹位于*C:\StellarisWare\boards\dk-lm3s9b96\hello\ccs\Debug*内，前提是 StellarisWare 已经被安装在*C:\directory*内。 可使用文本编辑器来修改映射文件。

如果此项目在之前由一个较早版本的代码生成工具（更早的编译器）进行编辑，项目文件夹也将包含一个与那个版本编译器相关的映射文件 (hello_ccs.map)。 请确保所指的映射文件 (hello.map) 由代码生成工具 v4.9 生成。如图 16所示，这个信息在映射文件的开头提供。 或者，您可以使用 Windows 时间戳来识别刚刚被生成的映射文件。

```
******************************************************************************
                  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
```

图 16. hello.map 的屏幕截图

如图 16所示，代码的可执行部分为 0x16C6 字节长（5830 字节长）。 正如*Stellaris LM3S9B96 微控制器数据表*(SPMS182)中解释的那样，受保护的闪存的大小只能为 2KB 的倍数。 考虑到本项目的用途，必须至少保护 6KB（3 个扇区）的闪存以适应 5830 字节的可执行代码。

下一步，必须确定代码的可对部分的长度，为以下部分的总和：

- 矢量表 (.intvecs)：0x11C
- 常量 (.const)：0x1e9c
- 初始化数据 (.cinit)：0x40

这三个部分的总和为 8186 个字节（例如，7KB），转换为闪存中的 8KB（4 个扇区。）

注： 如果 'hello' 示例被修改或重建，上面提到的部分的长度会被修改。 诸如编译器优化的选项也会影响不同代码部分的长度。 因此，始终建议保留额外的余量，但这并不是必须的。

5 通过修改连接器命令文件来在闪存中保留一个读取保护区域

连接器命令文件（也被称为连接器文件，或在某些集成开发环境 (IDE) 中被称为散列文件）被用来定义存储器中部分的地址。 Code Composer Studio 中的连接器文件的扩展名为 .cmd 并且此文件 (hello_ccs.cmd) 可在*C:\StellarisWare\boards\dk-lm3s9b96\hello*目录中找到，前提是 StellarisWare 已经被安装在*C:*\directory 目录中。 可使用一个文本编辑器来修改连接器命令文件。图 17显示了为了这个示例创建的最初缺省连接命令文件的内容。 正如红圈标出的那样，存储器的两个部分被定义为 FLASH 和 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;
```

图 17. 系统内存映射

闪存存储器映射可在图 18中进行图形化表示并包括 0x0000 0000 至 0x0040 000 的地址。 换句话说，256KB 闪存存储器是可执行且可读的。

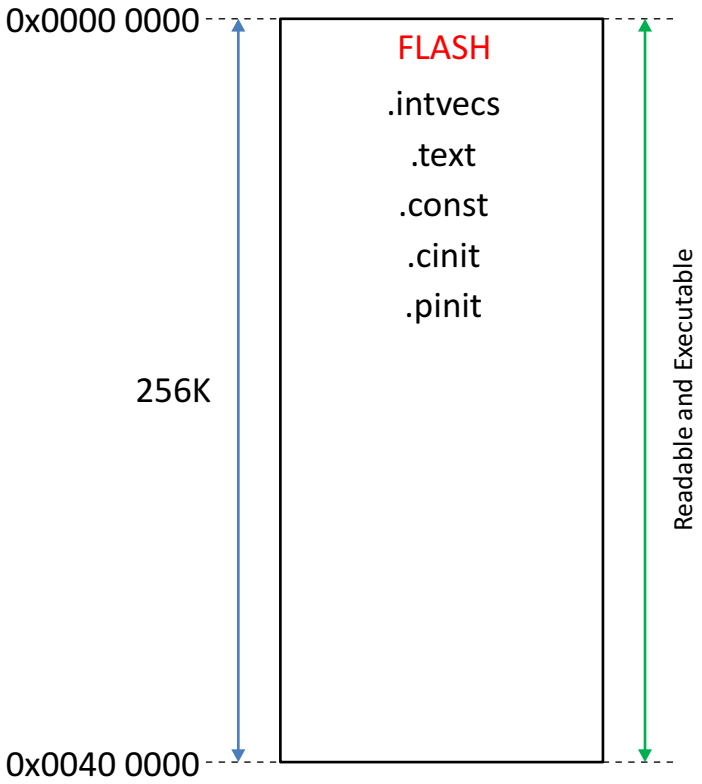


图 **18.** 内存映射_闪存

为了保护可执行代码不被读取，必须在闪存内保留一个独立的读取保护部分。 在这样一个部分被保留前，必须在内存映射中对其进行定义。 考虑到这个项目的用途，闪存内的 2KB 被保留用于矢量表；被命名为 FLASH_1 并被配置为可读和可执行（缺省配置）。

根据4.4 节，闪存的 6KB 被保留用于可执行代码；命名为 FLASH_EX 并被配置为可执行但不可读。 剩余的闪存可被命名为 FLASH_2。 针对包含常量和初始化数据的剩余代码，它被配置成可读和可执行（缺省配置）。 这表示在图 19中。

地址空间	0x0000 0000 至 0x0000 0800	可读且可执行
地址空间	0x0000 0800 至 0x0000 2000	可执行但不可读
地址空间	0x0000 2000 至 0x0040 0000	可读且可执行

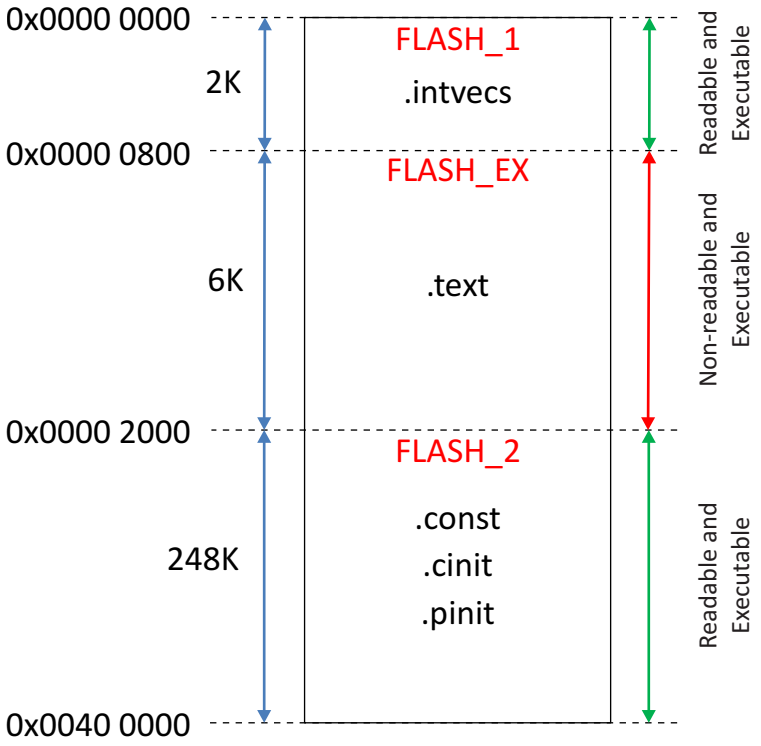


图 **19.** 内存映射（**FLASH_1**，**FLASH_EX** 和 **FLASH_2**）

为了执行刚刚描述的更改，如图 20所示那样修改连接器命令文件。

```
#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;
```

图 20. 连接器命令文件

保存并关闭连接器命令文件并重建此项目。

现在打开映射文件并确保它表示了已经在内存中创建的新的部分，如图 21所示。 将它与之前引用的映射文件相比较。 请注意已经在闪存内创建额外的读取保护区域，"FLASH_EX"，以及其它区域，"FLASH_2"，此区域可读且可写入。

```
******************************************************************************
                  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
```

图 **21. TMS470** 连接器 **PC v4.9.0** 映射文件

6 重建关联库（使用代码生成工具 **v4.9** 建立驱动程序库和图形库）

这个示例 ("hello") 使用来自 StellarisWare 驱动程序库和图形库的 API 函数调用。 因此，也必须使用代码生成工具 v4.9 来编译这些库，而且代码生成工具 v4.9 所使用的编译器设置要与项目 ("hello") 被建立时的编译器设置一致。 编译 DriverLib 和 GraphicsLib 的指令在以下部分中进行了说明。

6.1 编译驱动程序库

将驱动程序库 (driverlib) 项目添加到一个已经存在 "hello" 项目的现有工作区内。

1. 指向 Code Composer Studio 内的 Project 菜单 → 选择 Import Existing CCS/Eclipse Project → 单击 Browse 按钮并浏览*C:\StellarisWare\driverlib*位置。 在当前的工作区内为 DriverLib 导入项目文件。
2. driverlib 项目将被设定为激活项目。 如果不是这样，那么通过右键单击 driverlib 项目并选择如图 22中所示的 Set as Active Project（设定为激活项目）选项来将其手工设定为激活项目。

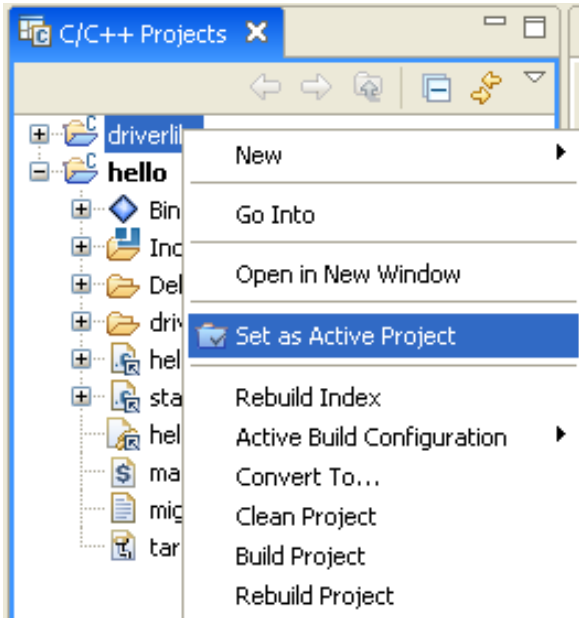


图 **22.** 将一个项目设定为激活项目

3. 指向 Project 菜单 → 选择 Properties 来打开如图 23中所示的项目属性窗口。
4. 在 General 标签页中选择 Code Composer Studio Build → 在 Code Generation Tools 域中选择 TI v4.9.0 compiler。 在选择这些选项后 → 单击 Apply 按钮。

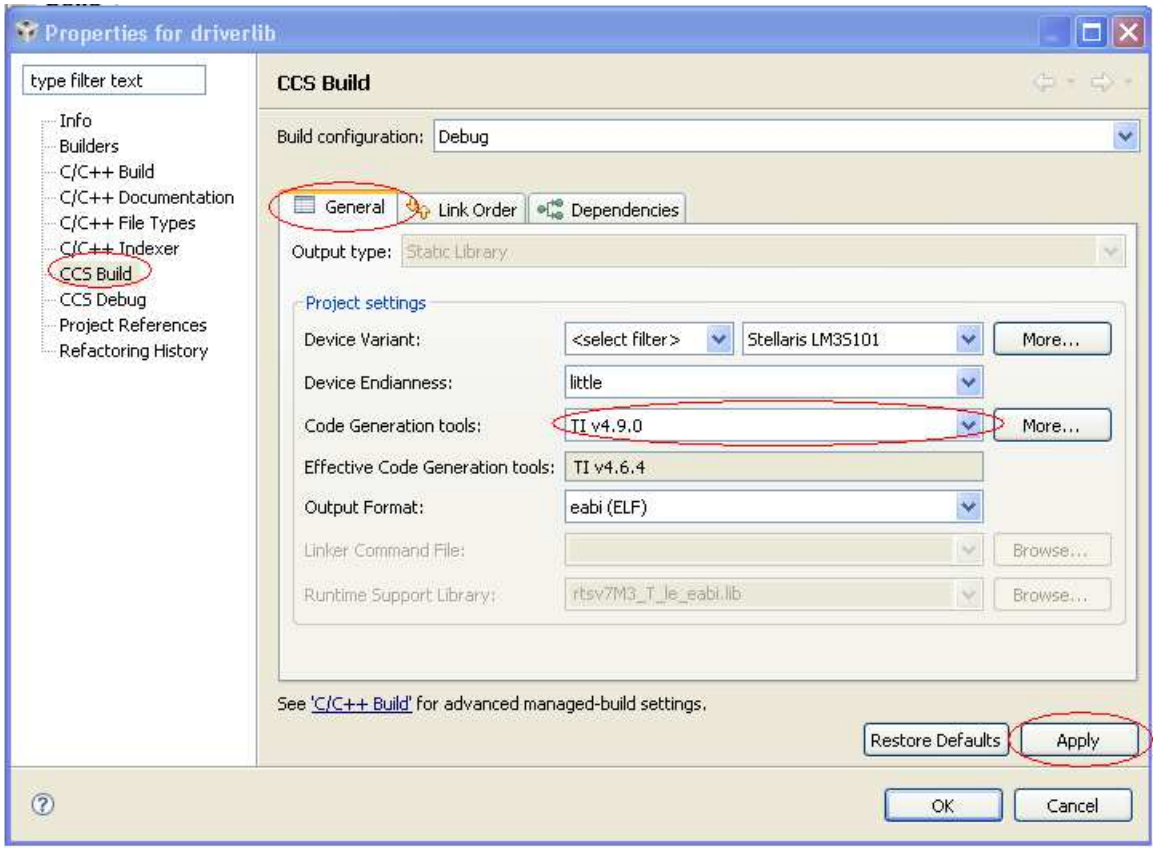


图 **23.** 针对驱动程序库的属性

5. Build Configuration Settings 窗口将出现 → 如图 24中所示选择选项 → 单击 OK 按钮。

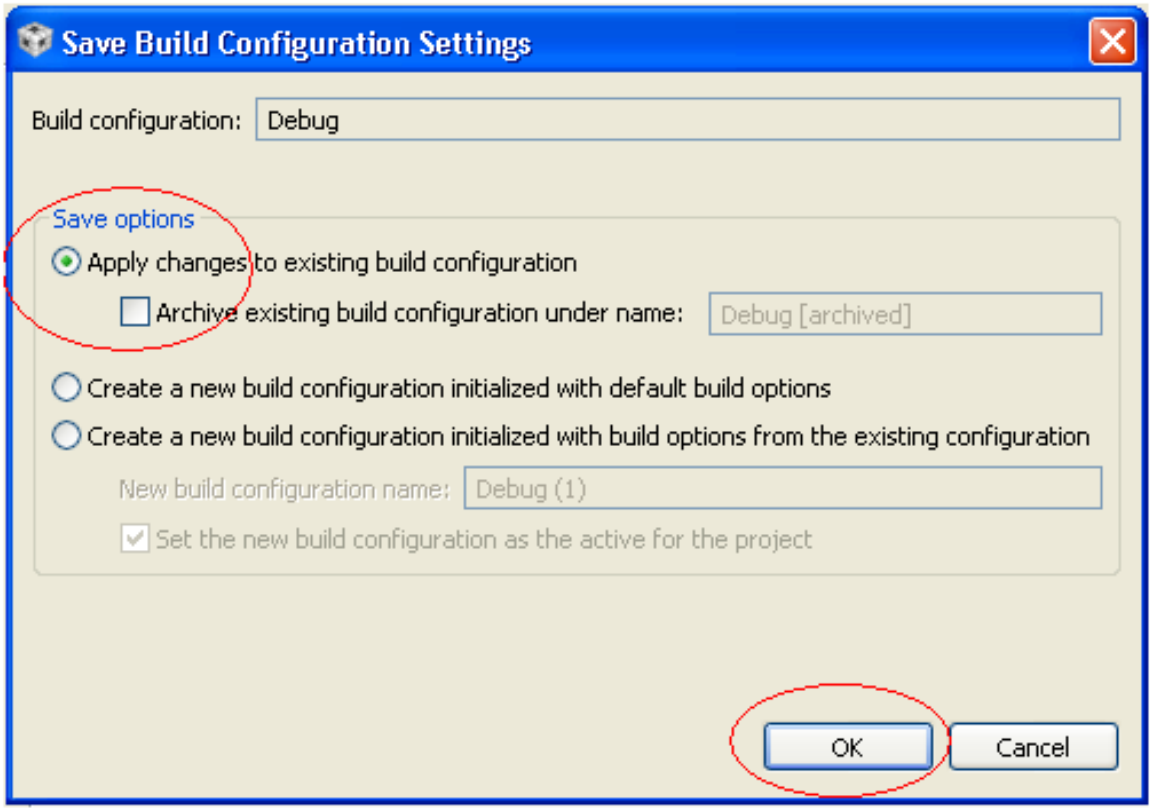


图 **24.** 保存 **Build Configuration Settings**

6. 当以下对话框出现时，单击 OK 按钮。

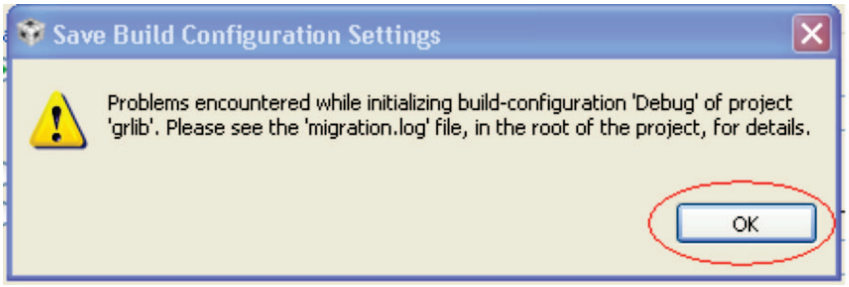


图 **25.** 保存 **Build Configuration Settings**（问题）

7. 在之后出现的窗口上单击 OK（如图 26中所示）→ 返回用于此项目的 Code Composer Studio 项目属性。

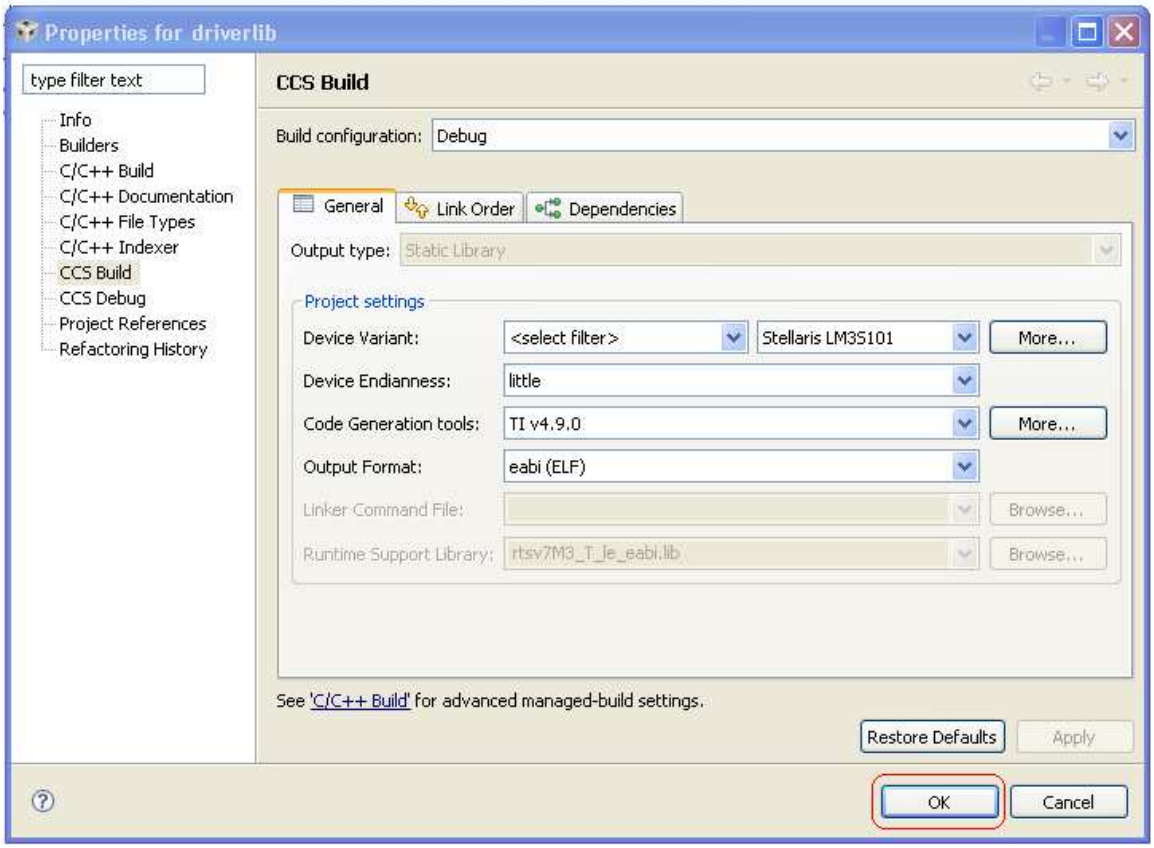


图 **26.** 针对 **driverlib** 的属性（**Code Composer Studio** 建立）

8. 选择 Tool Settings 标签页下的 C/C++ Build → 在 TMS470 Compiler 域内选择 Runtime Model Options 如图 27中红圈所示的那样选择选项（打开和关闭）并单击 Apply 按钮。

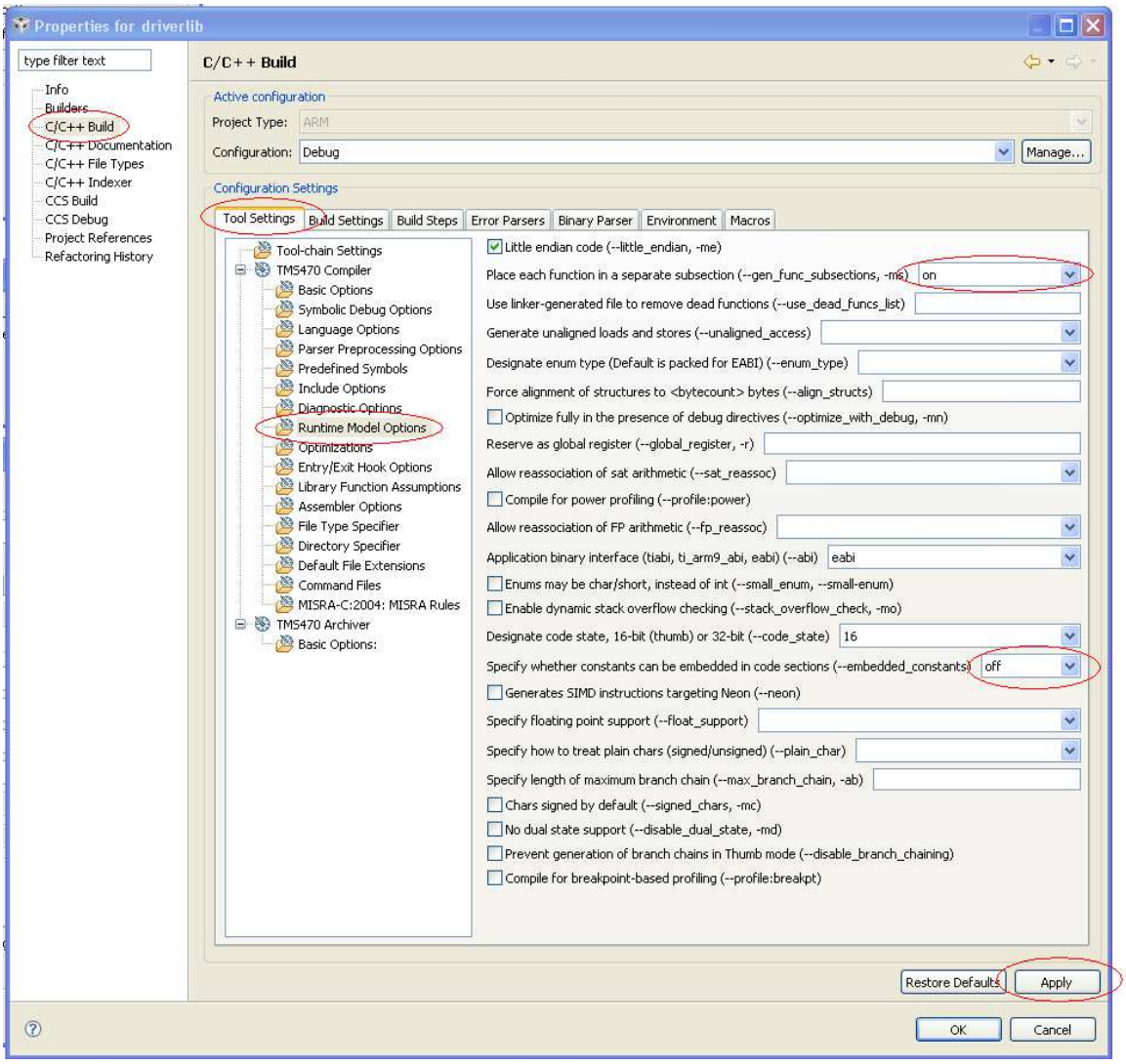


图 **27.** 针对 **driverlib** 的属性（**C**，**C++** 建立）

9. 下一步，如图 28中所示，选择 Predefined Symbols 并通过单击 按钮来把附加定义添加到 Pre-define NAME 部分中。 如图 28中的红圈所示，添加预先定义的名称。 请注意大小写并包含以下两个符号：
   - ccs
   - PART_LM3S1101

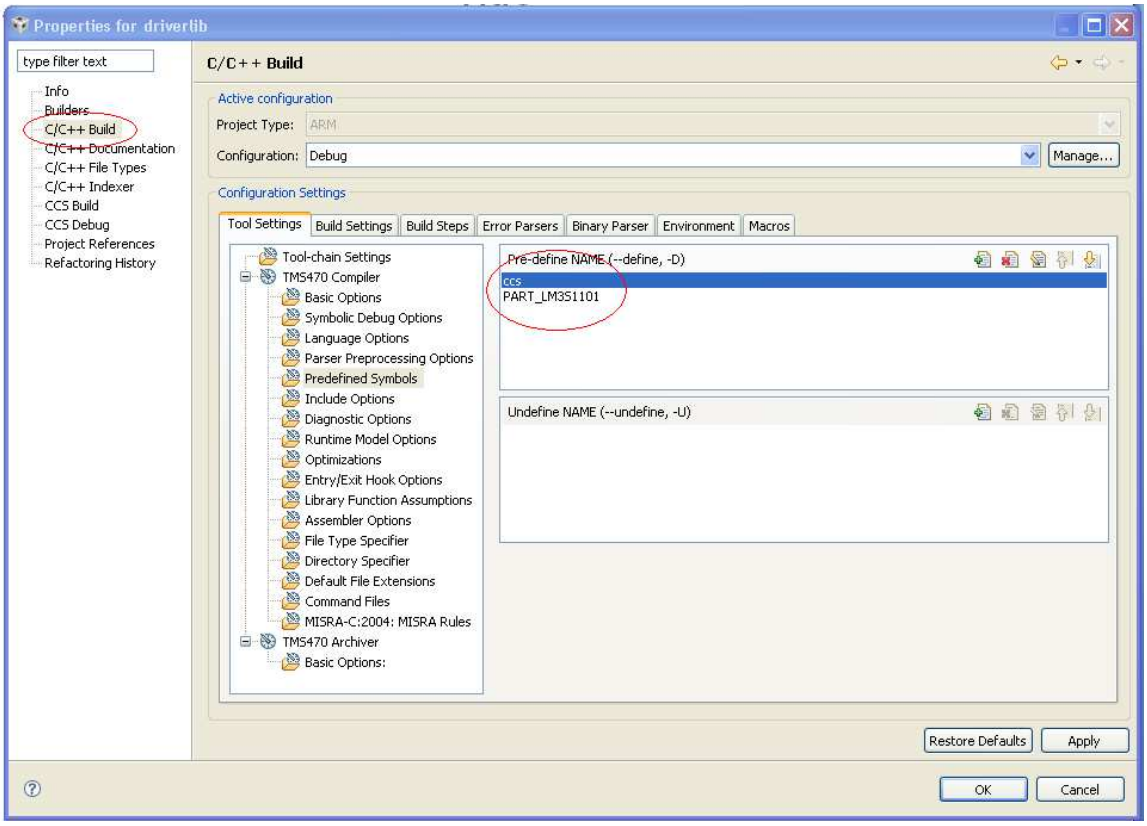


图 **28.** 针对 **driverlib** 的属性（预先定义的名称）

10. 单击 Apply 按钮 → 返回 Code Composer Studio 主窗口/文件编辑器窗口并重建库。

6.2 编译 Graphics-Lib

将 GraphicsLib 项目添加到已存在 "hello" 项目的现有工作区中。

1. 指向 Code Composer Studio 内的 Project 菜单 → 选择 Import Existing CCS/Eclipse Project → 单击 Browse 按钮 → 浏览*C:\StellarisWare\grlib*位置。 将针对图形库 (grlib) 的项目文件导入至当前的工作区。

   grlib 项目将被设定为激活项目。 如果不是这样，那么通过右键单击 grlib 项目并选择 Set as Active Project 选项来将其手工设定为激活项目。
2. 指向 Project 菜单 → 选择 Properties 来打开如图 29中所示的项目属性窗口。
3. 在 General 标签页中选择 CCS Build。 在 Code Generation Tools 域中选择 TI v4.9.0 compiler。 在选择了这些选项后，单击 Apply 按钮。

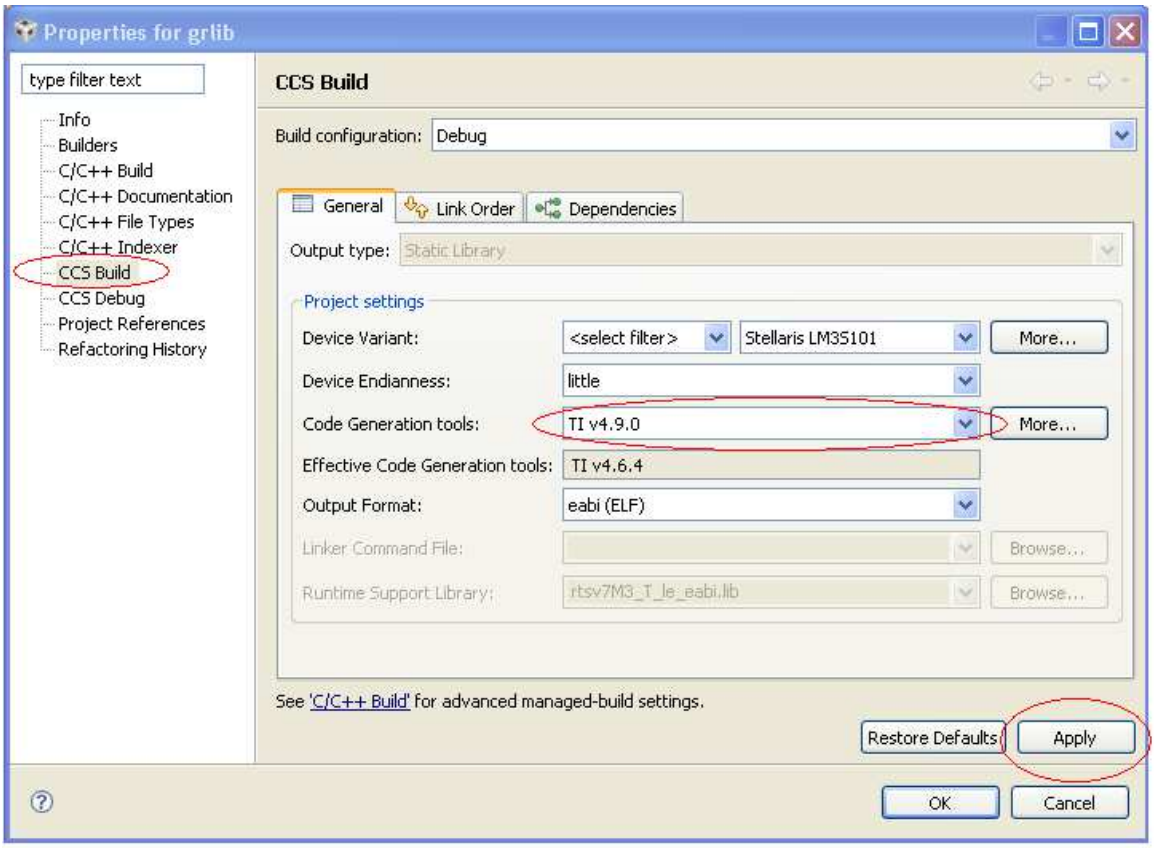


图 **29.** 针对 **grlib** 的属性

4. Build Configuration Settings 窗口将出现 → 如图 30中所示选择选项 → 单击 OK 按钮。

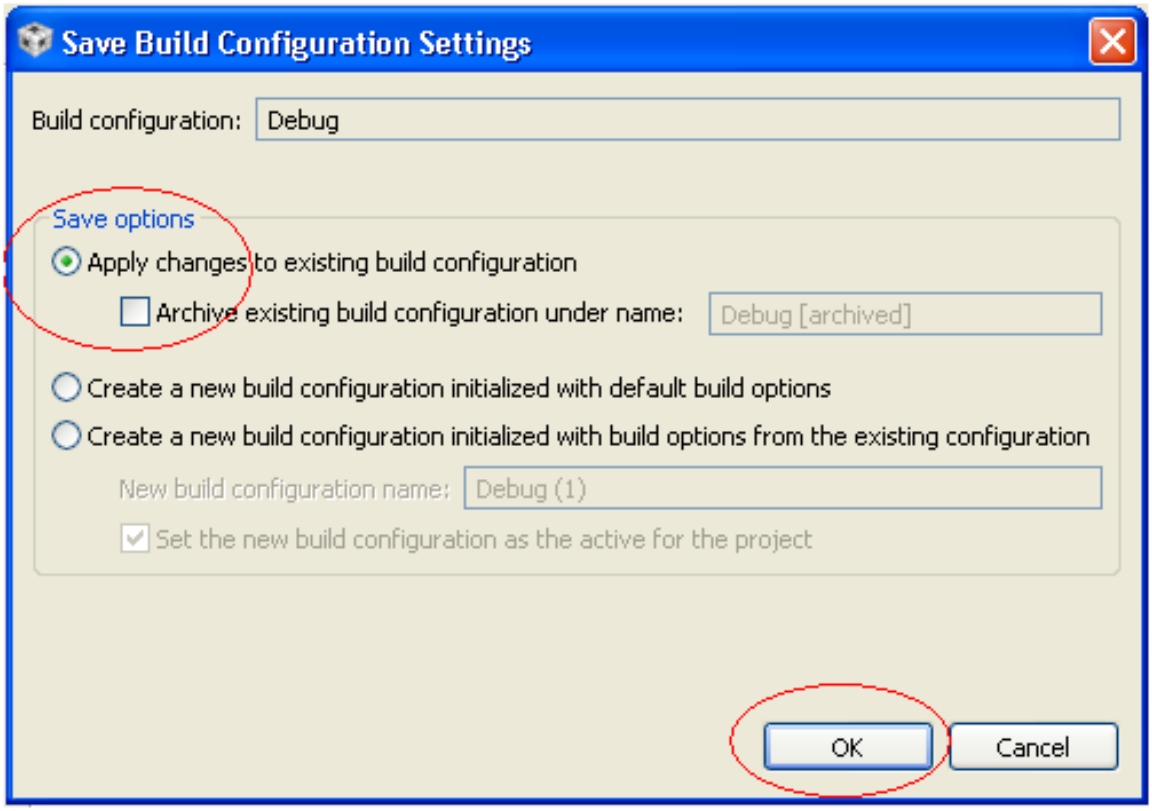


图 **30.** 保存 **Build Configuration Settings**（调试）

5. 当以下对话框出现时，单击 OK 按钮。

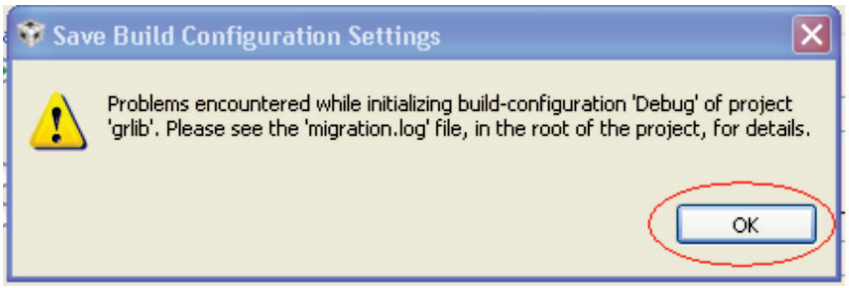


图 **31.** 保存 **Build Configuration Settings**（问题）

6. 在下一个出现的窗口中单击 OK 按钮（如图 32所示）并为这个项目返回 Code Composer Studio 属性。

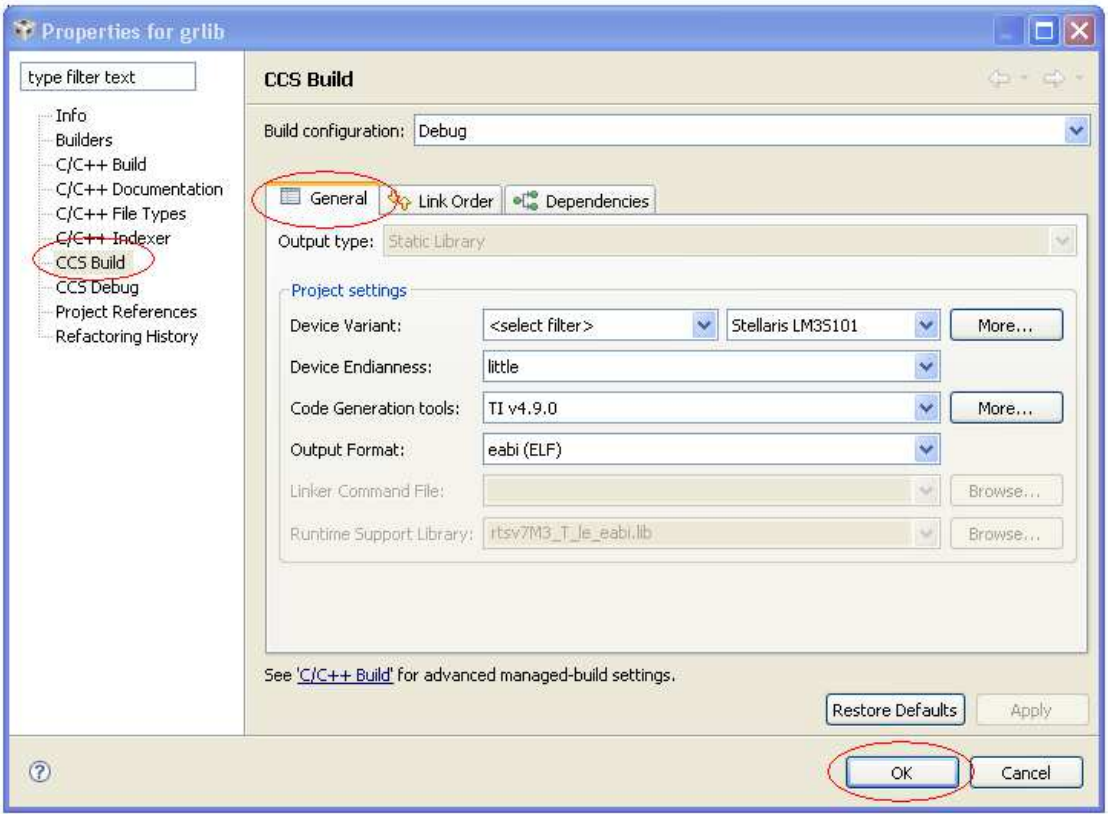


图 **32.** 针对 **grlib** 的属性（**Code Composer Studio** 建立）

7. 选择 Tool Settings 标签页下的 C/C++ Build → 在 TMS470 Compiler 域内选择 Runtime Model Options 如图 33中的红圈所示选择选项（开和关）→单击 Apply 按钮。

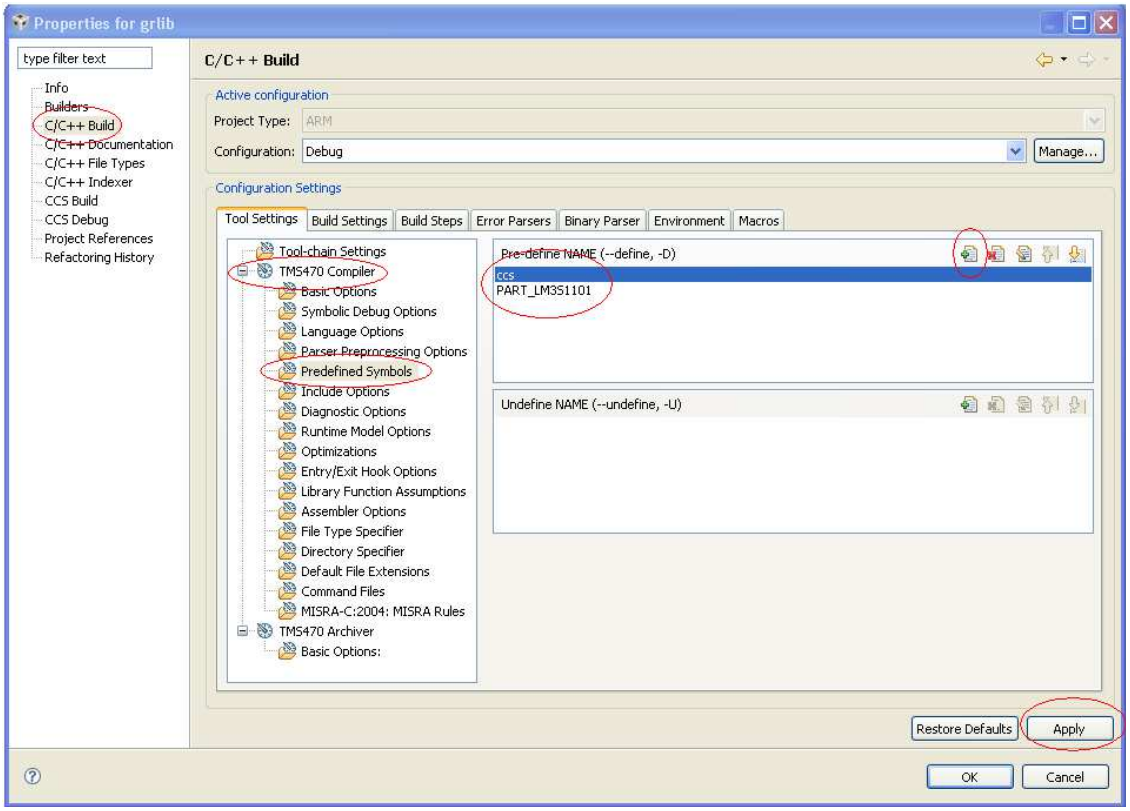


图 **33.** 针对 **grlib** 的属性（**C**，**C++** 建立）

8. 下一步，如图 34中所示 → 选择 Predefined Symbols 并通过单击 按钮来把附加定义添加到 Pre-define NAME 部分中。 如图 34中的红圈所示，添加预先定义的名称。 请注意大小写并包含以下两个符号：
   - ccs
   - PART_LM3S1101

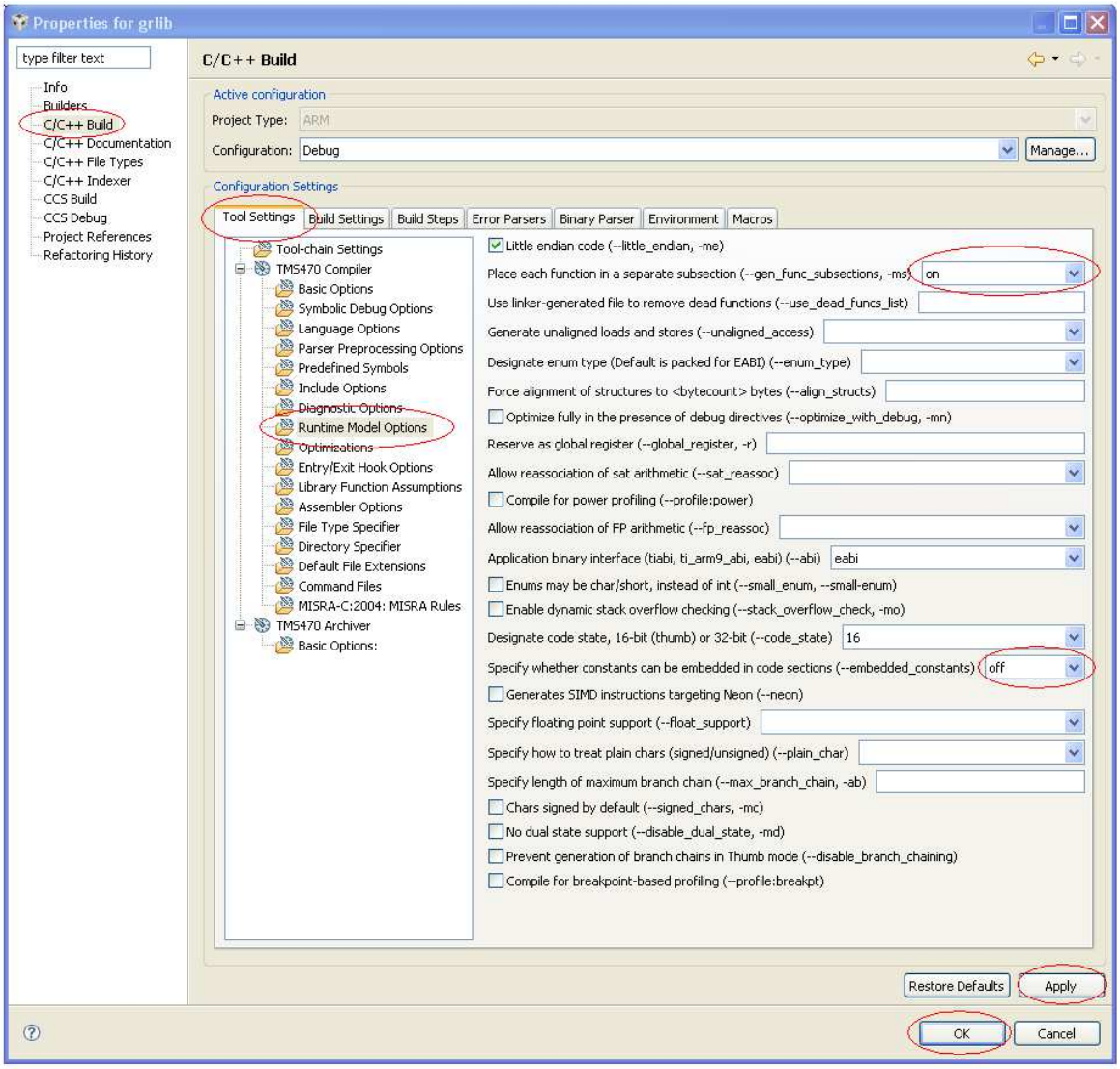


图 **34.** 针对 **grlib** 的属性（调试）

9. 单击 Apply 按钮 → 返回 Code Composer Studio 主文件编辑器窗口并重建库。

7 将闪存保护代码添加到项目中并建立它

7.1 使 "hello" 项目再次激活

通过右键单击 "hello" 项目并选择 Set as Active Project，在项目浏览器窗口框内将 "hello" 设定为激活项目。

7.2 将合适的头文件包含在 II 中

在*flash.h*头文件中声明闪存 API。 为了使用这些 API，如图 35中所示，将*flash.h*头文件包含在内。

```
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
```

图 **35.** 包含头文件

7.3 将闪存保护代码添加到 "hello" 项目中

为了将闪存存储器配置为只执行（读取保护），应该将适当的自变量传递到*FlashProtectSet()*API 中。 可使用*FlashProtectSave()*API 来作出这些改变。 一旦提交，保护设置是永久的，不能通过执行芯片复位或功率循环来取消操作。 建议只在您确认您的代码运行方式与设计运行方式一致时才使用*FlashProtectSave()*。 图 36中显示了此函数调用。

请注意，通过使用 LM 闪存编辑器来执行调试端口解锁序列可将闪存保护设计复位为它们的厂家缺省配置。

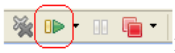
```
63 int
64 main(void)
65 {
66     tContext sContext;
67     tRectangle sRect;
68
69     //
70     // Set the clocking to run directly from the crystal.
71     //
72     ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_XTAL_16MHZ |
73                        SYSCTL_OSC_MAIN);
74
75     //
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();
```

图 **36.** 添加闪存保护代码

现在重建此项目。

8 启动调试程序

下一步，您能够证明只执行、只写入和只擦除闪存保护的运行结果与预期的运行结果一样，闪存内受读取保护的区域不能被读取。

在代码已经被建立后，将应用下载到闪存存储器中。 通过单击 Run 按钮  来开始代码执行。

单击 Pause/Suspend（暂停/挂起）按钮  来暂停代码执行。

启动调试程序来检查内存位置的内容。 如图 37所示，可通过从 Code Composer Studio 菜单条的 View 菜单中选择 Memory 选项来启动 Memory 窗口。

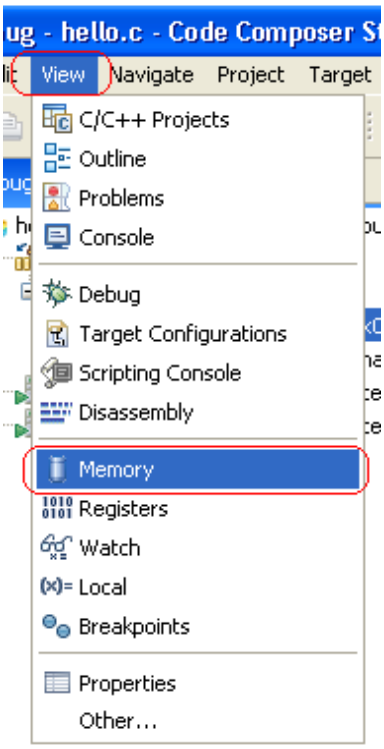


图 **37.** 启动内存窗口

现在，检查不同区域 - 即 FLASH_1，FLASH_EX，FLASH_2 地址内的闪存存储器的内容。

首先，读取 FLASH_EX 区域内的内容。 您将会注意到 FLASH_EX 区域内的闪存内容，比如地址 0x0000 0800 内的内容不可读。 对于其无法读取的内存单元，调试程序显示 "????????"。 此外，会像 GrStringDraw 那样显示某些符号。 图 38显示了这一过程。

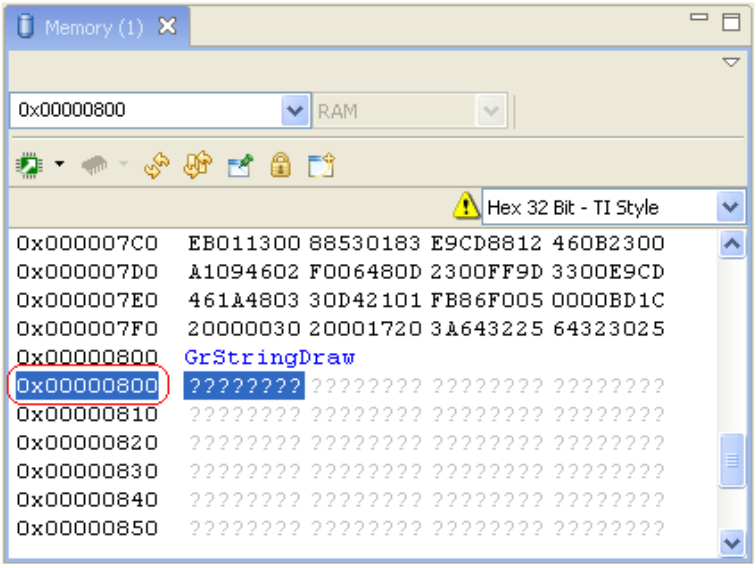


图 **38.** 内存窗口_**0x00000800**

在试图访问 FLASH_EX 区域的其它地址时，调试程序将如图 39所示在控制台窗口中提示一个错误。 这是预料之中的，并且表示对于这些地址，闪存是读取保护的。

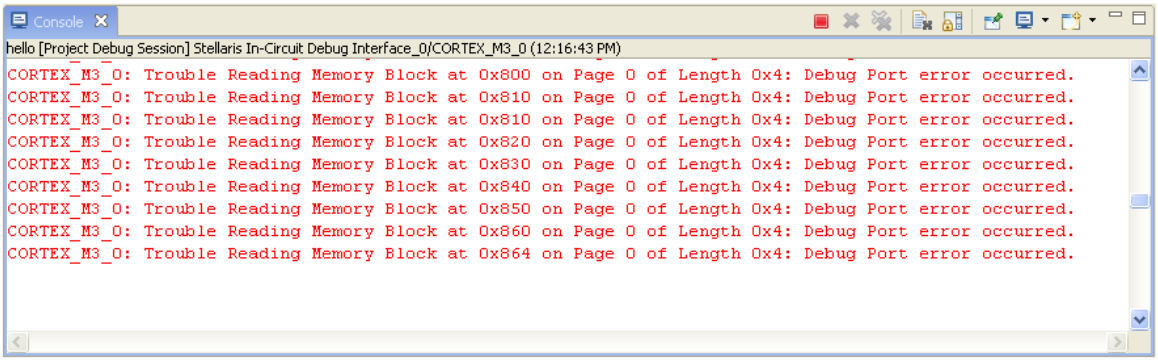


图 **39.** 调试程序控制台

其次，读取 FLASH_2（或 FLASH_1）区域中的内容。请注意，FLASH_2 区域内的闪存内容，比如地址 0x0000 2000，如图 40所示是可读的。

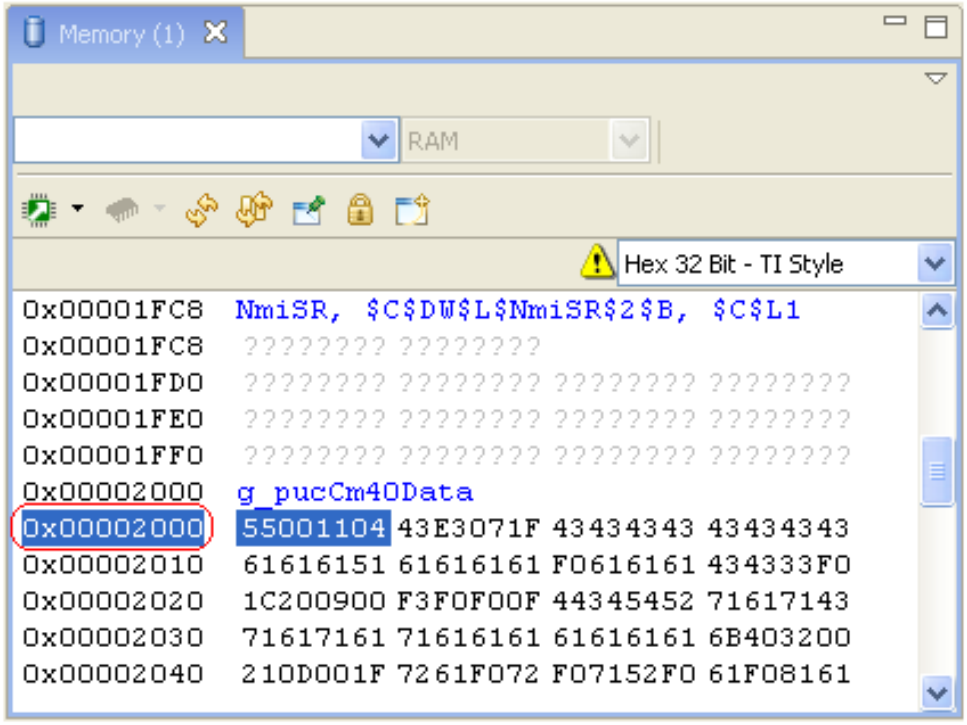


图 **40.** 内存窗口_**0x00002000**

最后，请注意，在试图访问 FLASH_EX 区域的其它地址时，调试程序将在控制台窗口中提示一个错误，如图 41所示。 这条消息是预料之中的并表明针对这些地址，闪存是写保护的。

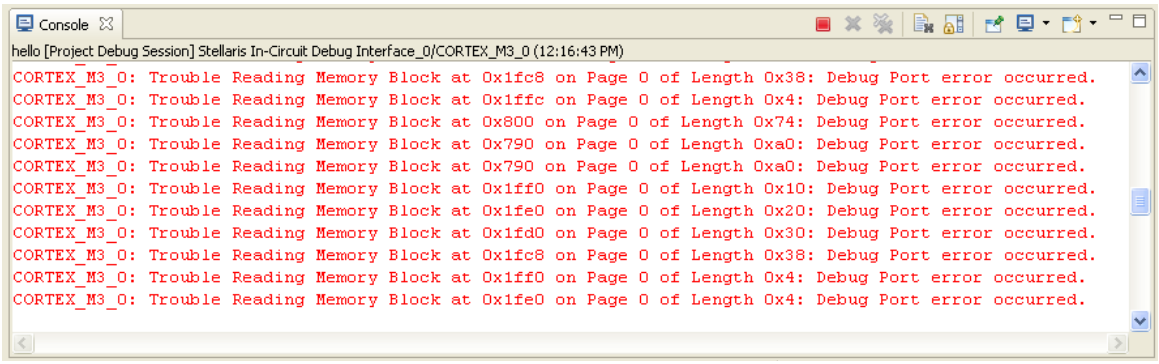


图 **41.** 调试程序控制台_**2**

9 结论

通过使用一个来自 StellarisWare "hello" 示例，已经演示了如何在 Stellaris 微控制器上使用只执行、只写入和只擦除闪存保护，这使得设计人员能够在最终应用中保护他们的代码和 IP，同时又为固件升级提供了灵活性。 使用 TI 的 Code Composer Studio v4.2.3 和代码生成工具 v4.9，闪存保护已经通过在闪存存储器中创建一个只读区域被执行以防止代码被读取但可被执行（内存块可被写入、擦除或执行但不能被读取），这提供了将转储文字与可执行代码分离的优势。

注: 一旦提交（保存），闪存保护设置不能由功率循环或将 MCU 复位来改变或取消。 使用 LM 闪存编程工具执行调试端口解锁（JTAG 切换批量擦除序列）将擦除整个闪存存储器并将闪存保护复位为厂家缺省值。

10 参考书目

- 《*Stellaris LM3S9B96 微处理器数据表*》(SPMS182)
- 可从以下 URL 中下载带有代码生成工具 v4.9 的 Code Composer Studio v 4.2.3：http://www.ti.com/ccs
- StellarisWare 软件包可从以下 URL 中下载：http://www.ti.com/stellarisware
- 基于 FTDI 的 ICDI 驱动程序可从以下 URL 中下载：http://www.ti.com/tool/lm_ftdi_driver

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122