Converting UCD3138 Firmware Project from Code Composer Studio Version 3.3 to 5.2

SLUA679A
May 2013

Table of Contents

List of Figures

1 Introduction

This document details a simple step by step process for converting projects made in CCSv3.3 to a project that will work in CCSv5.2. Moreover, following this method will allow for the same project to be compiled and linked in either CCSv3.3 or CCSv5.2.

By using the provided CCSv5.2 Template project, project settings have already been so a user can follow a few simple steps to allow project compilation in both compilers. From a high level, one simply needs to place the template's project files into their project directory, change a few settings in CCSv5.2, rename the project, edit a few files, and the project will work in both CCSv3.3 and CCSv5.2 using the same source files and in the same folder.

This document is specifically designed for use with the projects provided with the TI EVMs for the UCD3138. Projects in a similar format should also work. However, if the project organization is significantly different, these instructions may not work.

What follows is a detailed description of this simple step by step method for upgrading a CCSv3.3 based project to work with CCSv5.2.

2 Equipment needed

It is assumed that the user has at their disposal an installation of Code Composer Studio version 3.3 and Code Composer Studio version 5.2.

The screenshots in this document use Windows 7 as an operating system, but it can be assumed that this method will work for any operating system that has the ability to run CCSv3.3 and CCSv5.2.

2.1 CCSv5.2 Conversion Template

The CCSv5.2 Conversion template is a project that has been created in CCSv5.2 that has the necessary project settings to allow for conversion of an existing CCSv3.3 project. To obtain a copy of this project template, please visit: http://www.ti.com/litv/zip/sluc456

The template contains 8 files, as tabulated below:

File name	Function
.cproject	Functions as .pjt file used in CCSv3.3
.csproject	
.project	
UCD3xxx.ccxml	Defines device for debugging
software_interrupt_wrapper.c	Makes software interrupt usable for both CCS versions
software_interrupts.h	
software_interrupts.c	Moves the software_interrupt into a separate file for changing optimization parameters.
interrupts.c	
clear_program_flash.c	Split up clear_program_flash.c and zero_out_integrity_word.c for more reliable checksum and program flash clearing.
zero_out_integrity_word.c	

2.2 Appropriate Header and Linker Files directories

Make sure that you have the appropriate "Header Files" and "Linker Files" directories (each containing appropriate project specific files) in the same location as the project directory as shown in Figures 1, 2, and 3 below:

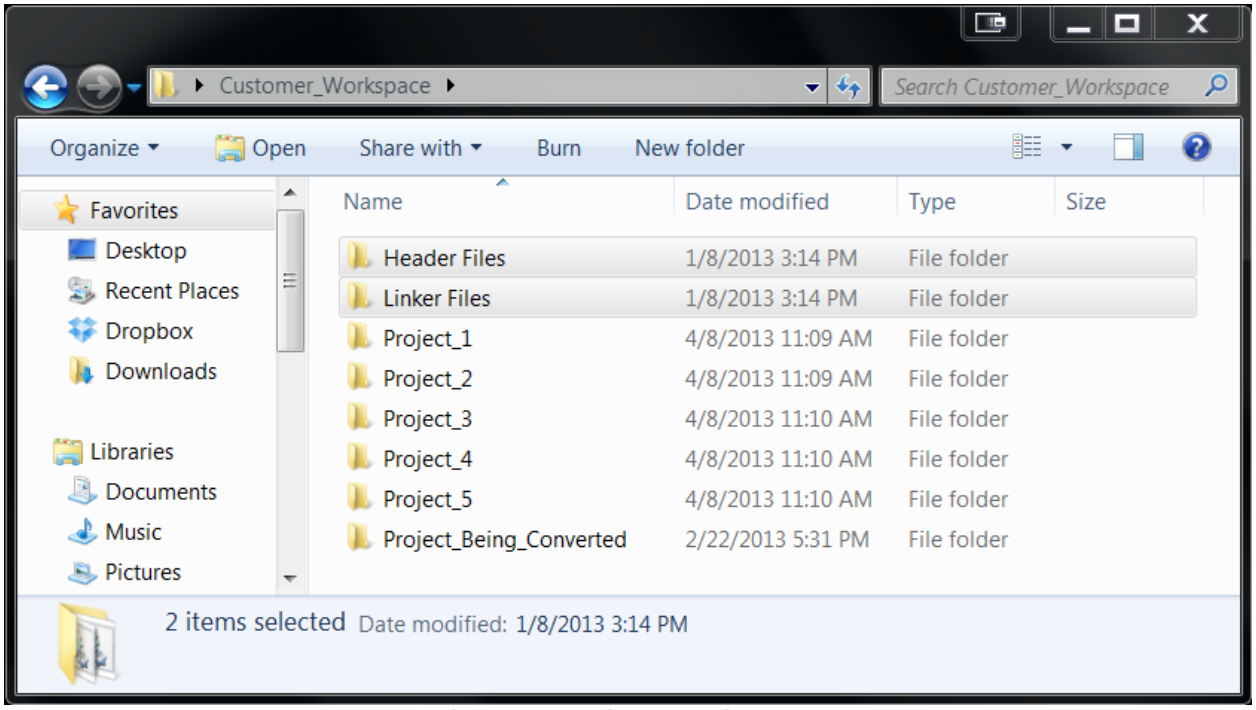

Figure 1: Workspace Directory

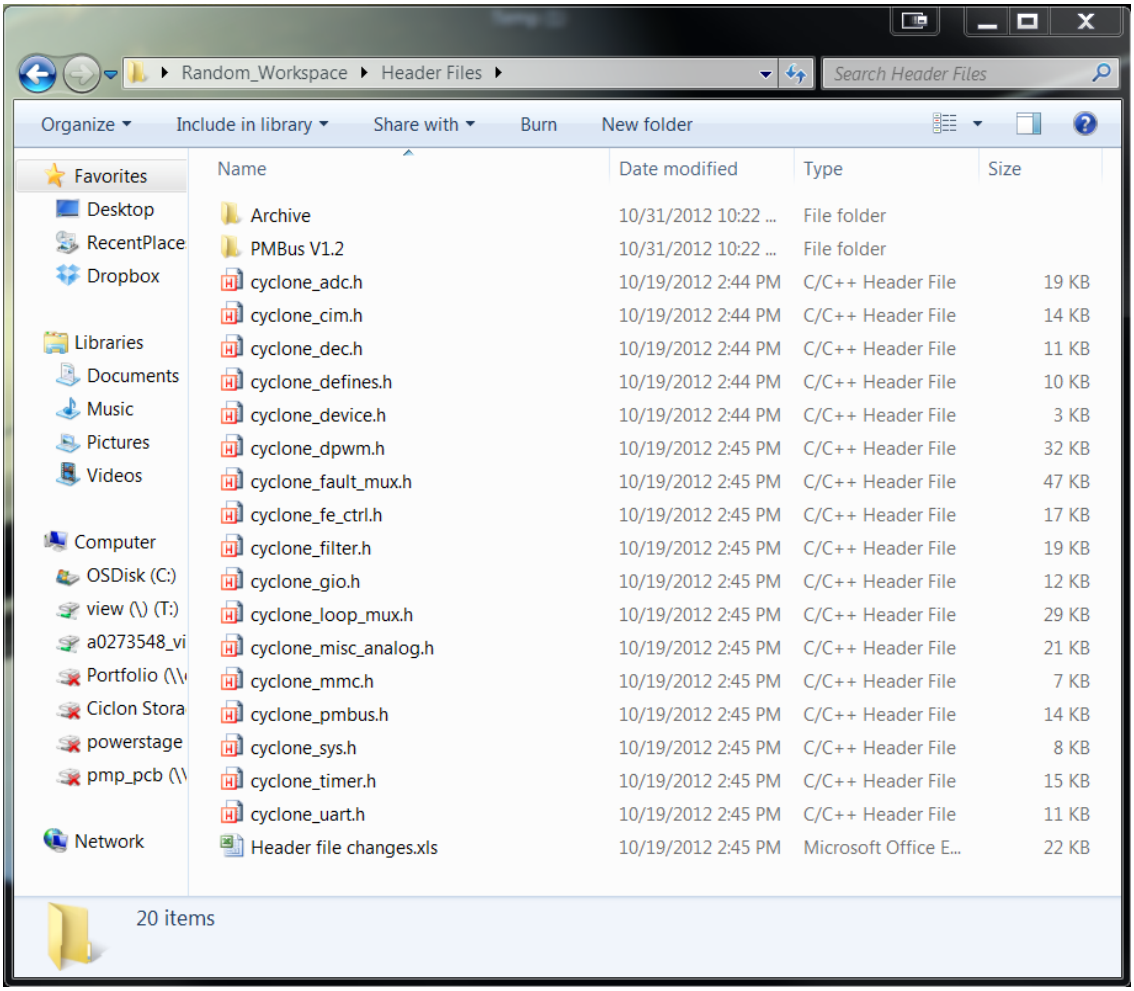

Figure 2: Header Files Directory

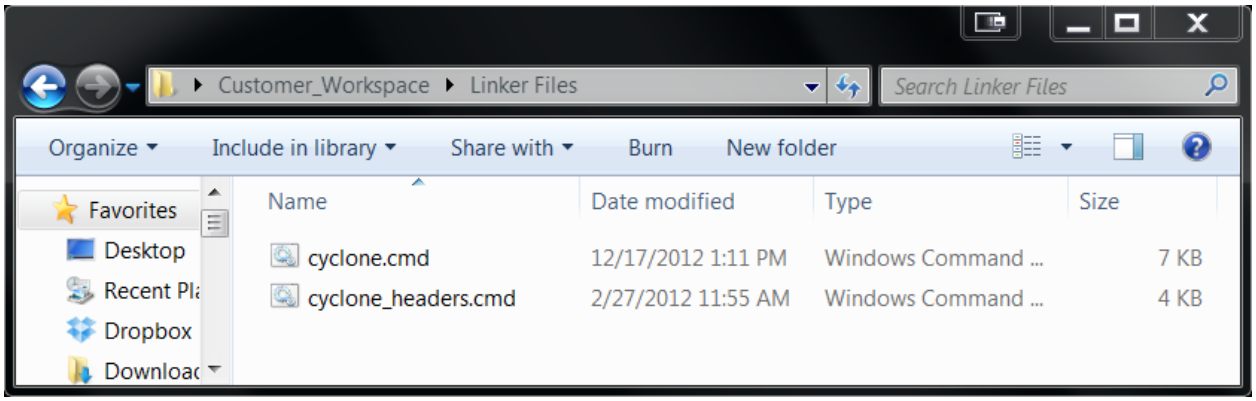

Figure 3: Linker Files Directory

3 Before Getting Started

Make sure to read this section carefully before going through the procedure in Section 4 in order to avoid compiler/linker errors and other undesirable behavior during project conversion.

3.1 The CCSv5.2 Template

The template can be obtained at http://www.ti.com/litv/zip/sluc456. The template provided will not compile on its own as a stand-alone project. The template is merely a container for introducing project files.

3.2 Using Version Control Software

If using version control software, make sure that you check out the .**project** and **.cproject** files. The **.project** file will be updated when renaming the project during step 8 of the import procedure, and the **.cproject** file will be updated when making any changes to the project settings.

3.3 New ARM library, and a warning for CYGWIN Users

CCSv5.2 uses a new library for the ARM architecture when compiling. When compiling this project for the first time, the new library is built and stored with the other ARM libraries. This process may fail if an installation of CYGWIN is present on the user's system. It is suggested to temporarily rename the directory of the CYGWIN compiler (C:\CYGWIN) to something else such as "C:\CYGWIN-temp" during this one-time compilation. If CYGWIN is present, CYGWIN error messages will be seen during the creation of the library.

Once the library is built, the CYGWIN directory name can be changed back, and CYGWIN can be used again.

4 Conversion Procedure

4.1 Gather All Necessary Project and File Information in CCSv3.3

Specific information will need to be known for this conversion. This section describes the different items that are needed in order to complete the procedure. Open a new instance of CCSv3.3. Open the project to be converted, and note the following information.

4.1.1 Know the Source File Code State Mode.

1) Most of the **.c** files are compiled with a **code state mode** set to **16-bit**. However, some files are compiled with a **code state mode** set to **32-bit**. This code state mode is specific to the individual files and could be changed by the user. Therefore it is necessary to find out the code state of the individual source files and preserve the change in CCSv5.2.
2) In general, the source file icons in the project directory that are **32-bit mode** will be highlighted in **yellow** (as shown in the figure below). Note the code state of the **.c** files before continuing. Alternatively, the user can **[Right-Click]** on the file in question, and select **File Specific Options**. In the **Compiler** tab, under the **Basic** category, the current Code State Mode will be shown.

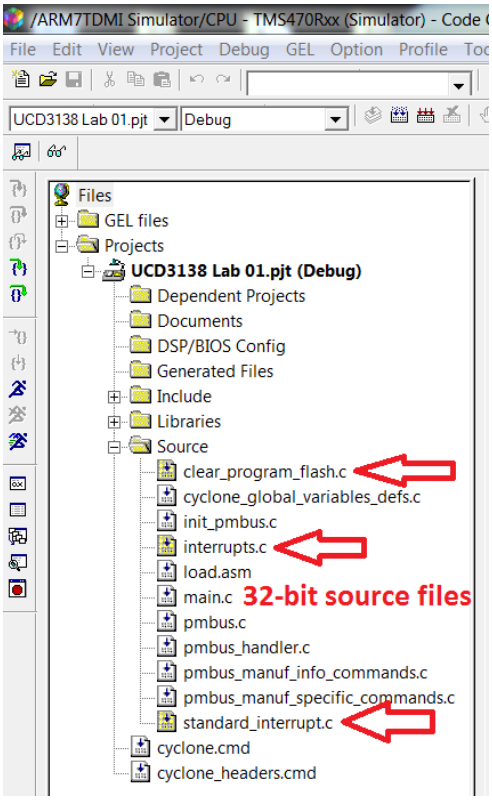

Figure 4: Yellow = 32-bit Code State

4.1.2 Know the Location of the Command Files

Depending on the UCD3138 project to be converted, the command files can either be located in the **Linker Files directory**, or in the **project directory** itself. For reasons that will be mentioned later, the location of the command files will need to be known.

1) **[Right-Click]** on a command file and choose **Properties.** Make note of the file location. Is it in the **Linker Files directory** or the **project directory**? Remember this for later. In the figure below, both command files are located in the Linker Files directory.

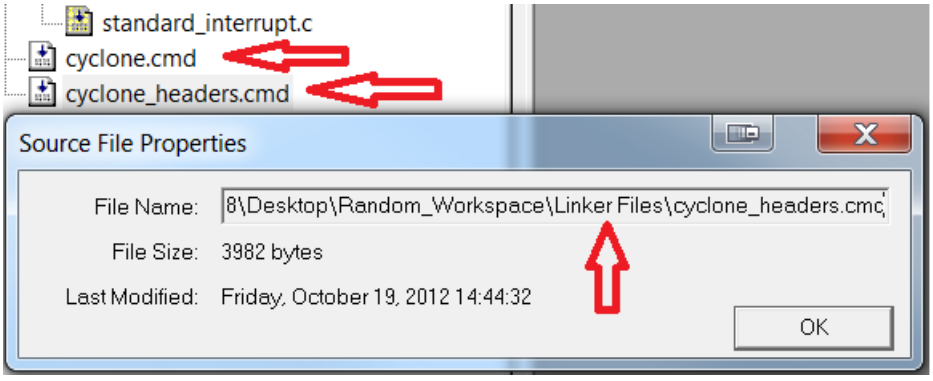

Figure 5: Command File Location

4.2 Import and Rename the CCSv5.2 Template

1) **Copy** and **Paste** the three project files located in the Template Project directory and place them directly inside the project directory of the project to be converted.
2) Open up a new instance of CCSv5.2. When prompted, you can choose any location you want for a workspace, we will not be using the CCS workspace in this document.
3) In the program menu, **[Left-Click]** on **Project**, and **[Left-Click]** on **Import Existing CCS Eclipse Project**. Even though we are converting a CCSv3.3 project, we do not want to **Import an existing CCSv3.3 project**. The template has been created in CCSv5.2. As a result we will need to choose the former. This will bring up a menu as shown in Figure 6. The **Import an existing CCSv3.3 project** function of CCSv5.2 does not provide optimal results when used with UCD3138 programs.
4) **[Left-Click]** on the **Browse** button next to the **Select search-directory** option and browse for the directory of the project to be converted..
5) **Do not select Copy projects into workspace. By not selecting this checkbox, all file changes occur in the existing project folder. This is important for version control, and if CCSv3.3 is to be used for the same files in the future.**

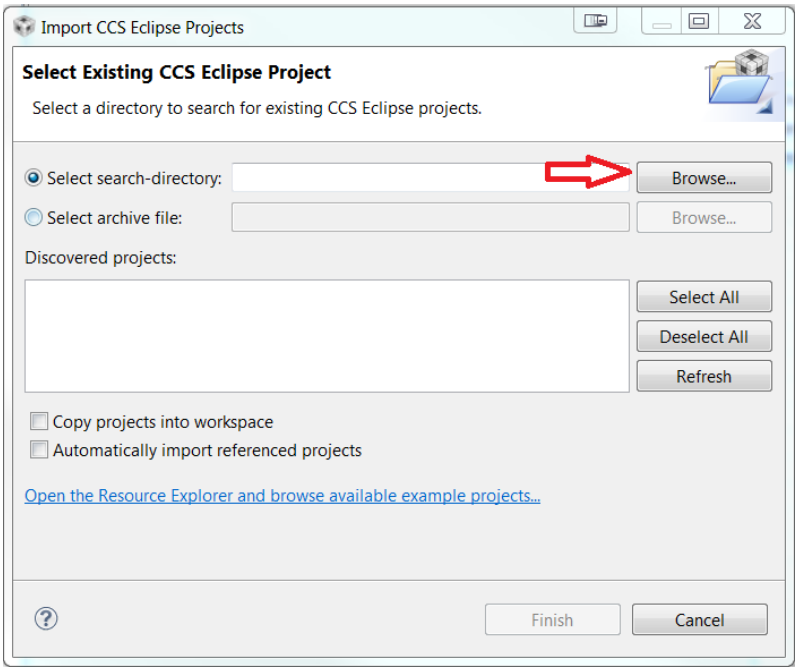

Figure 6: Importing the Template

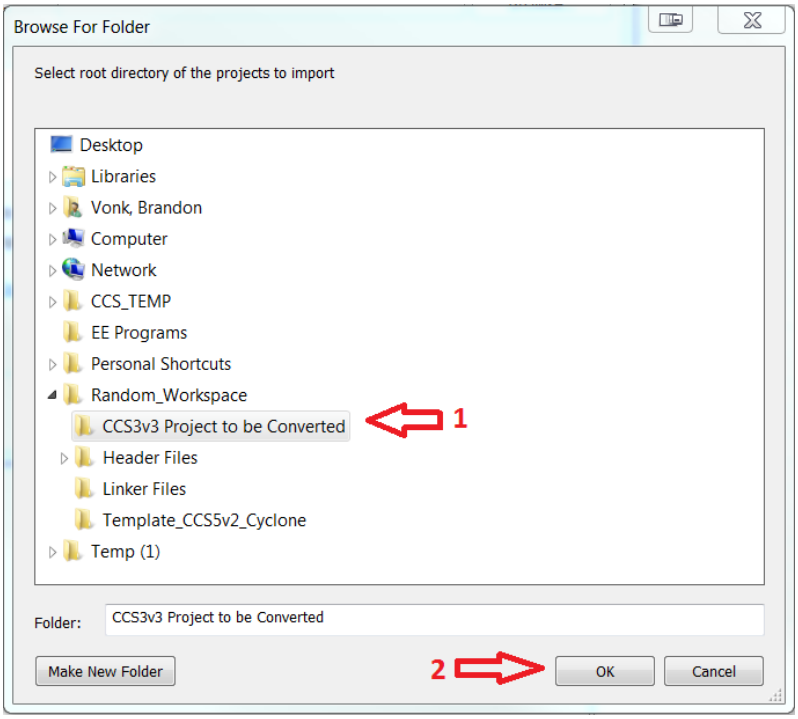

Figure 7: Browse for the Project Directory

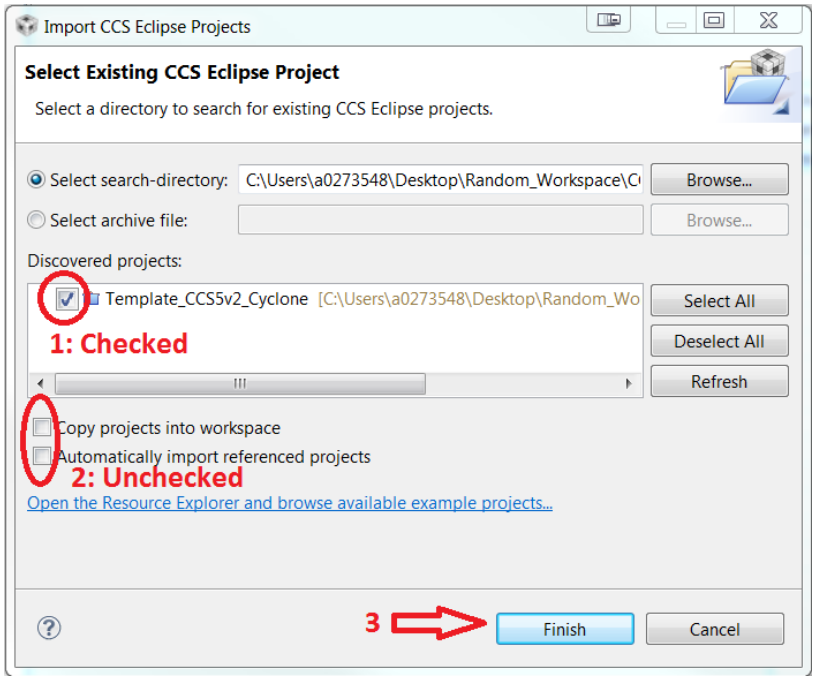

Figure 8: Choose Discovered Project

6) After browsing for the project directory, the **Discovered projects** section of the main Import window should now be populated with the name of the template project. Make sure this is **[checked]**.
7) Make sure that the **Copy projects into workspace** option is **[unchecked]**. **[Left-Click]** on **Finish**.
8) This should import the project into the **project explorer window**. It should have the template project name, but should include all **.c** files, **command** files, and **load.asm**. If there are any other

files that are not listed in the project explorer file list, they must be added explicitly by **[Right-Clicking]** on the project in the project explorer, and selecting the **Add files** option. Conversely, if there are files that are not used in the project, yet are in the listed in the project explorer, they should be “excluded” from the build. To do this, refer to Section 4.3.4.

The last thing to do to finish the import is to rename the project as desired. It is highly recommended to use the same name as the CCSv3.3 version of the project in order to avoid any confusion with the output files. For example, if the names are not the same, two **.map** files will be created. If the names of the projects must be different, the user must manually delete the old .map file before using the memory debugger of the **Fusion Gui**.

9) **[Right-Click]** on the project in the project explorer and choose **Rename**. Rename the project as desired and **[Left-Click]** on **OK**.

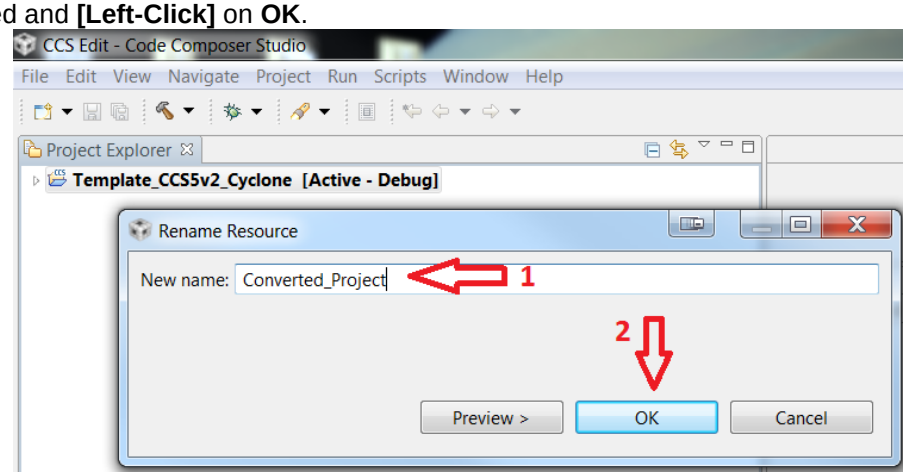

Figure 9: Rename the Project

4.3 Editing the converted project for compilation

By now the project should be imported into CCSv5.2. However, there are a few changes that will need to be made to the project and individual files so that the project will compile without errors. Follow the steps below to get the project compiling properly.

4.3.1 Setting the Code State Mode

Referring to Section 4.1.1, we need to make the **CCSv5.2 compiler** use the same code state mode that was used in the **CCSv3.3 compiler** for the old project. To do this, in Section 4.1.1, the code state mode was noted. Since the 16-bit state mode is the template default, For the 32-bit state mode files, follow the procedure below to tell the CCSv5.2 compiler to use a 32-bit mode.

1) **[Right-Click]** on one of these files and choose properties. Under the **ARM Compiler**’s **Processor Options**, choose **32** in the drop down menu for the **Designate code state, 16-bit (thumb) or 32-bit (--code_state)** option.

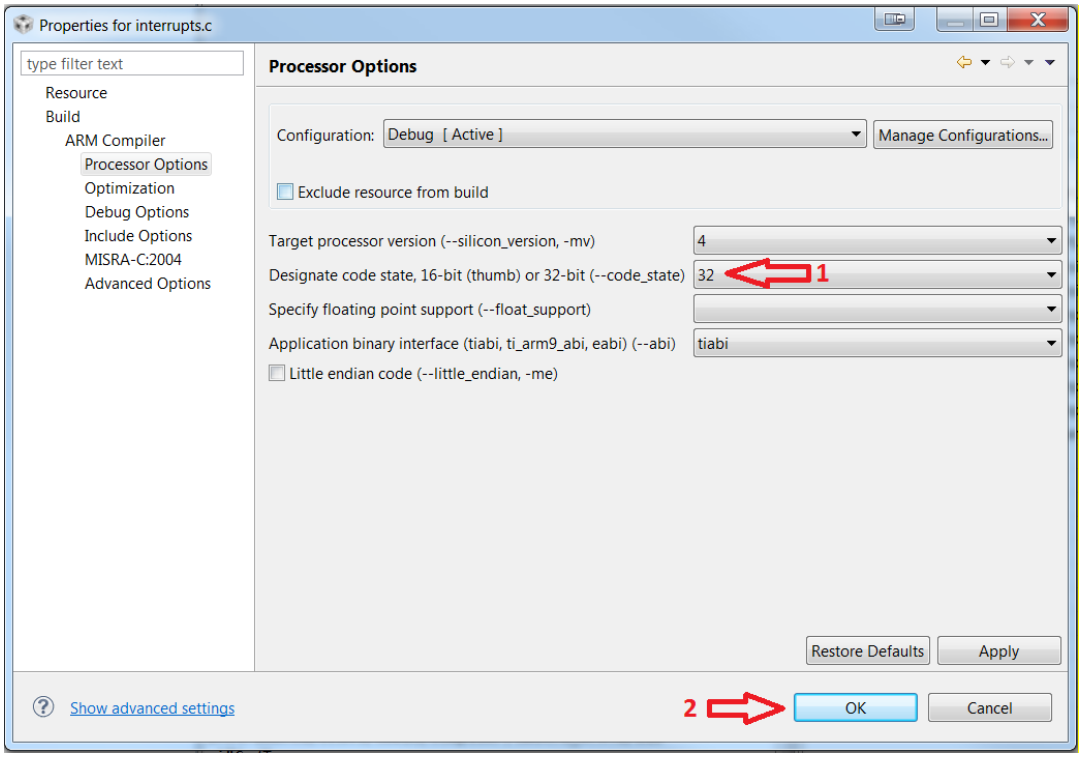

Figure 10: Designate a 32-bit Code State

2) Make sure to do this for all 32-bit code state mode **.c** files that were noted in Section 4.1.1.

4.3.2 Optimization

The optimization levels in CCSv5.2 do not match those in CCSv3.3. The template is set to a reasonable level where further optimization doesn't seem to decrease code size significantly. If there are demanding system requirements, experimentation with optimization may be required. There is a specific change regarding optimization that needs to be made to **interrupts.c**. See **Section 4.6.2** for details.

4.3.3 Setting the Command (.cmd) File Location

The **CCSv5.2 compiler** needs to know what command files to use. If no location is specified for the project setting, the compiler will assume that the command files are located in the **project directory**. However, sometimes the command files will be located in the **Linker Files directory**. The default setting in the CCSv5.2 template is that **cyclone_headers.cmd**, and **cyclone.cmd** files are located in the Linker Files directory. All others are located in the project directory.

After noting the location of the command files for the CCSv3.3 project in 4.1.2, continue to the procedure below:

1) Go to the project properties by **[Right-Clicking]** on the project in the project explorer, and selecting **properties**. In the Properties window that appears, **[Left-Click]** on the arrow next to ARM Linker, and **[Left-Click]** on **File Search Path**.
2) Use one of the buttons to the side to Add, Edit, or Delete the command file as directed in steps 3 through 5, and depicted in Figure 11 below.
3) If the file is located in the Linker Files directory, make sure that the path to the command file is included in the **Include library file or command file as input (--library, -l)** section.
4) If the file is located in the project directory, make sure it is not listed here. To delete a file path, choose the **Delete** button on the upper right (looks like a small page with an "X" on it).

5) Make sure that any files located in the project directory are NOT listed here. Conversely, also make sure that any files located in the **Linker Files directory** that are not in the project directory ARE listed here.

Make sure that there are no duplicates, or conflicting files. Only one mention of each command file should be included in the project. The user must choose to use command files either in the **Project directory**, or the **Linker Files directory.**

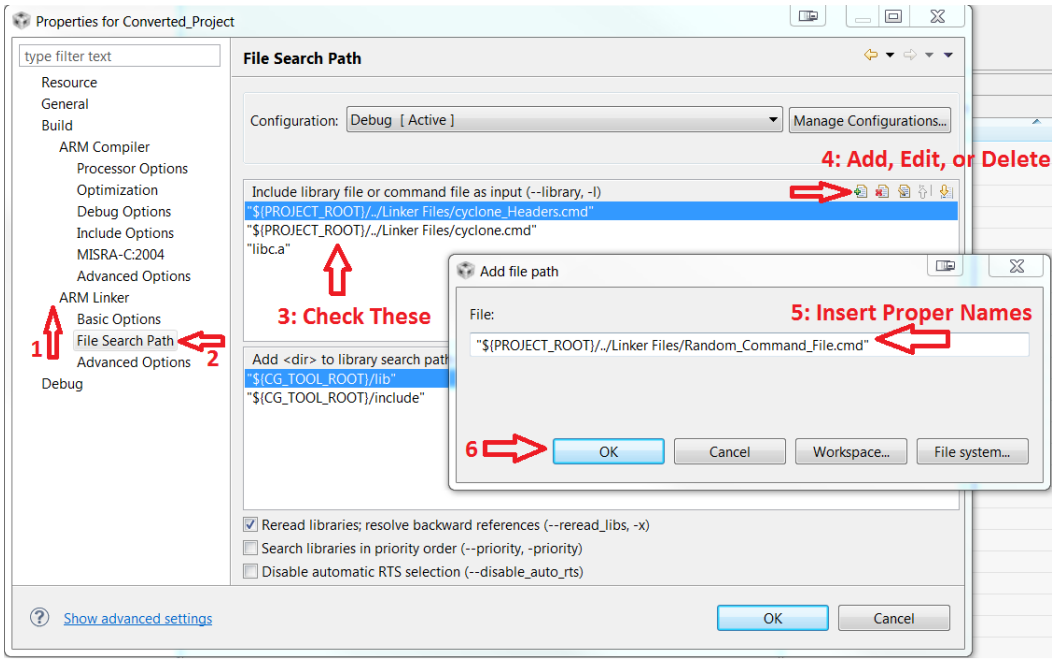

Figure 11: Project Properties

It may be necessary that a command file remain in the project directory, while also being located in the Linker Files directory. In this case, make sure that only one file is included in the build. To remove a file from the project directory without deleting it from the file-system, see 4.3.4.

4.3.4 Exclude From Build

The CCSv5.2 template is set to use all .c files in the project directory. CCSv3.3 files must be included explicitly, so sometimes there are unused .c files in the project directory. The simplest thing is to delete these unused files. It is also possible to use the **Exclude from build** option as shown below.

1) Go to the file's properties by **[Right-Clicking]** the file in the project explorer window, and choosing **Properties**. In the left menu, **[Left-Click]** on **Build**, then make sure that the **Exclude resource from build** option is **checked**. **Apply** the changes, and finally click on **OK**. As you will see, the file should now be relocated to the bottom of the project explorer list, and should appear grayed-out as it will no longer be compiled with the rest of the project.

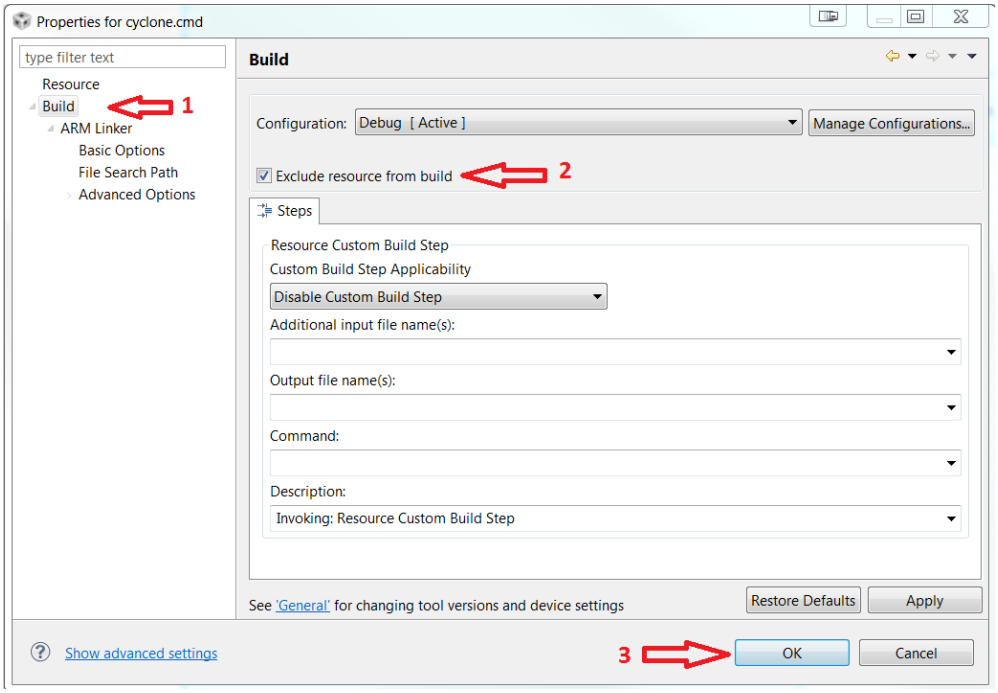

Figure 12: Properties for specific command file

4.4 Specific File Changes

Some files in any UCD3138 projects will need to be changed. This section document contains information for the changes to the following files:

- **software_interrupt_wrapper.c**
- **software_interrupts.h**
- **interrupts.c**
- **load.asm**
- **clear_program_flash.c**
- **zero_out_integrity_word.c**

There are two audience potentials for this section of the Document:

1) The user has not edited any of the following files and/or just needs a step by step method for getting the project working. In this case, follow the simple step by step process in Section 4.5.
2) The more advanced user projects may have some of these files edited for their own project. In this case, Section 4.6 details the changes that have been made to the aforementioned files.

At this point, you can go ahead and compile the project. However, even though the project will compile without errors, there will be several warnings and the .x0 file generated will not run properly until a few changes are made to the specific project files. Follow the guidelines below to eliminate the warnings and allow for the output files to run properly on the target board.

4.5 Specific File Changes for Non-User Edited files.

This step by step process deals with changing the project settings and individual files only if the file in question has not been edited by the user. In this case, the process is fairly simple:

1) **software_interrupt_wrapper.c, software_interrupts.h**, **interrupts.c,** and **load.asm** have all been updated. If these files have not been edited by the user, simply replace them with the updated versions in the template. To do this follow any one of the solutions below:

- Open up the current versions of the file in the project directory and comment out, or delete everything. Then open the new versions of the file located in the template directory, and copy and paste the contents of the file into the current version. This will make sure that both files are usable by both CCSv3.3 and CCSv5.2.
- Rename the current versions of the file (eg. **interrupts.c→OLDinterrupts.c**). Then copy the updated file from the template directory over to the project directory. This should accomplish the same as the previous option. However, you must investigate to see if the old renamed files are included in the project for both CCSv3.3 and CCSv5.2. If so, exclude them from the build.

NOTE: If one or more of these files have been edited, see Section 4.6 for an explanation of the changes that were added to the file in question, and repeat those changes on the version being used.

2) **clear_program_flash.c**, **zero_out_integrity_word.c**, and **software_interrupt.c** are new files that are included with the template. These files muse be added to both CCS versions of the project.

a. **CCSv5.2:**

i. Make sure that the **clear_program_flash.c**, **zero_out_integrity_word.c**, and **software_interrupt.c** files provided with the template have been copied into the project directory.

ii. In CCSv5.2, **[Right-Click]** on the project name in the project explorer and select **Add Files**. Navigate to the project directory and select the **clear_program_flash.c** and **zero_out_integrity_word.c** files from the navigation window (hold **[Ctrl]** in order to select multiple files).

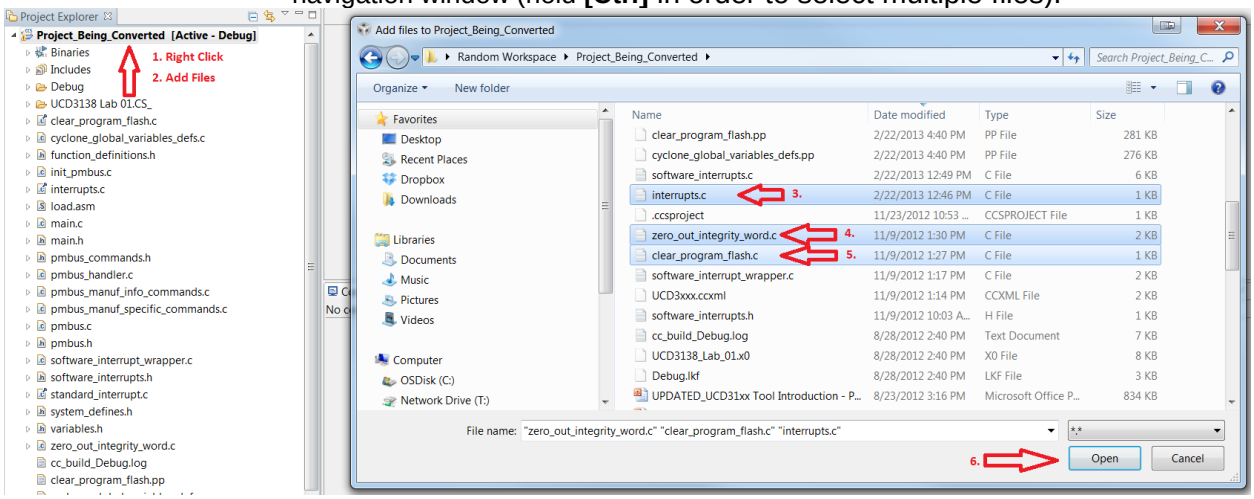

Figure 13: Insert New Files to Project

iii. You should now be able to compile the project. This will create the .x0 file as expected. This .x0 file can be downloaded to and run on the target board using the memory debugger.

4.6 Specific File Changes for User Edited files

If any of the files mentioned at the beginning of Section 4 have been edited, this section describes the changes that have been made to each file so that they can be re-done on the user version of the file.

4.6.1 Adding the software interrupt wrapper for CCSv3.3 and CCSv5.2 compatibility

Affected Files: **software_interrupt_wrapper.c**
software_interrupts.h

interrupts.c

The software interrupt functions have been moved into a wrapper. In CCSv3.3, an assembly instruction was inserted into the C code:

asm(" LDRB R3,[R14,#-1]"); //get swi number into R3 as fourth operand

This instruction loads the SWI number from the SWI instruction into R3, which makes it appear as swi_number in the software interrupt function parameter list:

void software_interrupt(Uint32 arg1, Uint32 arg2, Uint32 arg3, Uint8 swi_number)

This method does not work in CCSv5.2, because R14 and the function parameters are handled differently. So instead of multiple functions in **software_interrupts.h** which had each function defined for a different software interrupt number, there is now only a single software interrupt call that adds the software interrupt (swi) number in the parameters being passed to it as the fourth operand.

```
#pragma SWI_ALIAS (swi_single_entry, 0)
void swi_single_entry(Uint32 arg1, Uint32 arg2, Uint32 arg3, Uint8 swi_number);
```

The function calls that were handled by software_interrupt.h are now handled by a new function, with a new file:

software_interrupt_wrapper.c.

For instance, what used to be

```
#pragma SWI_ALIAS (write_data_flash_word, 3)
void write_data_flash_word(unsigned long * address,unsigned long data);
```

is now:

```
void write_data_flash_word(Uint32 address,unsigned long data)
{
        swi_single_entry(address,data,0,3); //code is 3;
}
```

Note that the 2 parameters from write_data_flash_word – address and data – are passed to the software interrupt. In addition, the swi_number is now passed as a parameter.

This function is defined in **software_interrupt_wrapper.c**.

A set of common SWI functions has been included in software_interrupt_wrapper.c.

To maintain compatibility, the prototypes for these new functions have been added to software_interrupts.h. This way, including software_interrupts.h will still be sufficient to provide function definitions.

If any software interrupt functions have been added and defined by the user, they must now follow this format. Mainly, the swi number, and any other parameters, must be passed as a parameter to **software_interrupt_wrapper.c** via the **swi_single_entry** function call.

As this method is now being used, the instruction to store the swi number is now removed in **interrupts.c** as shown below:

```
#pragma INTERRUPT(software_interrupt,SWI)
void software_interrupt(Uint32 arg1, Uint32 arg2, Uint32 arg3, Uint8 swi_number)
//void software_interrupt(Uint32 *address, Uint32 data, Uint32 more_data, Uint8 swi_number)
{
      //make sure interrupts are disabled
      asm(" MRS     r4, cpsr ");         // get psr
      asm(" ORR     r4, r4, #0xc0 "); // set interrupt disables
      asm(" MSR     cpsr, r4");          // restore psr

      asm("   LDRB   R3,[R14,#-1]"); //get swi number into R3 as
fourth operand
```

Also note that the registers used for disabling the interrupt have been changed from r3 to r4. This is because the software interrupt number is now coming in r3, rather than being placed after the interrupts are disabled.

4.6.2 Splitting **interrupts.c** into two files for proper optimization

As it turns out, CCS compiler versions above CCS version 3.3.38.2 have different optimization parameters, especially regarding assembly statements in C code (i.e. the **asm("…")** command). When using a CCS version above 3.3.38.2, care must be taken to change the optimization parameters for any file containing assembly instructions.

As **interrupts.c** has such commands, this template has split off the software interrupt routing (containing the assembly instructions) into a separate file called **software_interrupts.c**. This file's optimization parameters must be changed.

To do this, add the new file **software_interrupts.c** to the project as shown in back in Figure 13. This file contains the general software_interrupt routine code. If this code has been edited, then port over the changes, or replace the contents of the software_interrupt routine with the one in **interrupts.c**. Make sure to delete this routine in interrupts.c after having moved it over to the new file.

After having split up the **interrupts.c** file into **interrupts.c** and **software_interrupts.c** as directed in the last paragraph, changes to the optimization can now be made. **[Right-Click]** on the **software_interrupts.c** file in the project explorer window and **Properties**.Under **Build**, **Arm Compiler**, and **Optimization**, Select **1** from the dropdown menu next to **Optimization level (--opt_level, -O)**. This is shown in Figure 14 below:

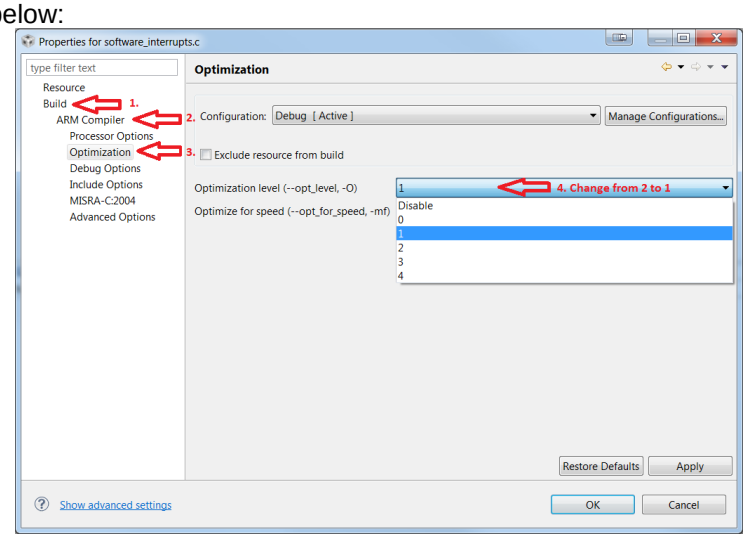

Figure 14: Changing the Compiler Optimization Level for software_interrupt.c

NOTE: If interrupts.c was compiled in a 32-bit code state mode, make certain that the new software_interrupts.c file is also compiled in a 32-bit code state mode as directed in Section 4.3.1.

4.6.3 Fixing any implicit function definition warnings

Affected Files: **Any files that use functions defined/prototyped in other files**

CCSv5.2 has more rigorous checking for some compilation warnings. Many CCSv3.3 projects may return errors similar to the following:

```
"../main.c", line 139: warning #225-D: function declared implicitly
```

To avoid this warning, each function must be defined before it is called. This means that either the function itself, or a function prototype must be placed in the file before the function is called. If this is not done, the warning above will occur.

Most of the TI standard programs have a file called **function_definitons.c**, which contains function prototypes. This needs to be included in any file which calls functions defined in some other file. If the warning occurs do the following:

1) Make sure the function prototype for the function called is in **function_definitions.c**.
2) Make sure there is a "`#include functions_definitions.c`" statement in the file where the function is called.

Adding **function_definitions.c** to a file may also require other #include statements. Very frequently, it is necessary to put a "`#include pmbus.h`" statement before the "`#include functions_definitions.c`" because the function prototypes use structures defined in **pmbus.h**.

A function prototype is just the name and parameters of the function with a semi-colon at the end. For example, here is one for **send_string**:

```
void send_string(const Uint8 string_to_send[], Uint8 num_bytes);
```

4.6.4 Avoiding the "Deprecated use of PSR; flag bits not specified, "cf" assumed" warning

Affected Files: **interrupts.c**
load.asm

The ARM7 Move to Status Register (MSR) assembly instruction has an optional field code which selects which parts of the ARM general purpose register are moved to the specified Processor Status Register (psr). The CCSv3.3 assembler always defaults to setting the right bits for the TI program. The CCSv5.2 also defaults to setting the correct bits, but it also issues a warning. This warning occurs because of the MSR instruction in **interrupts.c**. To eliminate this warning, open up **interrupts.c** and add "_cf" to the end of the spsr or cpsr operand as shown below:

```
case 5: //disable fiq
        asm(" MRS     r0, spsr "); //get saved psr
        asm(" ORR     r0, r0, #0x40 "); // set fiq disable
        asm(" MSR     spsr, r0"); //restore saved psr
        asm(" MSR     spsr_cf, r0"); //restore saved psr
```

load.asm also will need some of these changes:

```
;*-------------------------------------------------------
;* SET TO IRQ MODE, init IRQ stack
```

```
;*-------------------------------------------------------
        MRS     r0, cpsr
        BIC     r0, r0, #0x1F   ; CLEAR MODES
        ORR     r0, r0, #0x12   ; SET IRQ MODE
        MSR     cpsr, r0
        MSR     cpsr_cf, r0

               LDR            R13, c_irq_stack_top ; initialize stack pointer
```

NOTE: There are 3 instances of this change in **load.asm**, and 5 instances of this change in **interrupts.c**. These changes only apply to the MSR instruction, not the MRS instruction. See both examples above and notice that the MSR instruction has been changed, but the MRS instruction is left unchanged.

4.6.5 Forcing a reset of the checksum in interrupts.c

Affected Files: **interrupts.c**

interrupts.c now has a forced reset by setting the function pointer to an illegal location in order to make sure that the checksum is reset, avoiding a chip lock-up.

```
{
        register FUNC_PTR func_ptr;
        func_ptr=(FUNC_PTR)0x69000;     //Set function to 0x19000
        func_ptr(); //execute erase checksum
        func_ptr=(FUNC_PTR)0x70000;     //Set function to illegal location
        func_ptr();  //force reset
}
return;
```

4.6.6 Splitting up zero integrity word and clear_program_flash.c

Affected Files: **clear_program_flash.c**
zero_out_integrity_word.c

NOTE: Since these are both new files, they must be added to the project in both CCSv3.3 and CCSv5.2

The zero_out_integrity_word function must be pulled out into its own file named **zero_out_integrity_word.c** in this file, place the function that used to be in **clear_program_flash.c**, and place them in this new file (making sure to add the proper #include statements):

```
#define program_flash_integrity_word (*((volatile unsigned long *) 0x7ffc))
//last word in flash, when executing from Flash.  used to store integrity code

void zero_out_integrity_word(void)
{
        DecRegs.FLASHILOCK.all = 0x42DC157E;// Write key to Program Flash Interlock Register
        DecRegs.MFBALR1.all = MFBALRX_BYTE0_BLOCK_SIZE_32K; //enable program flash write
        program_flash_integrity_word = 0;
        DecRegs.MFBALR1.all = MFBALRX_BYTE0_BLOCK_SIZE_32K + //expand program flash out to 4x
real size
                                                    MFBALRX_BYTE0_RONLY;

        while(DecRegs.PFLASHCTRL1.bit.BUSY != 0)
        {
                ; //do nothing while it programs
        }
                //now reset processor.
//      TimerRegs.WDCTRL.bit.CPU_RESET_EN = 1;  // Make sure the watchdog is enabled.
//      TimerRegs.WDCTRL.bit.WD_PERIOD = 1;          // Set WD period to timeout faster.

        // This will never test true, but it prevents a compiler warning about unreachable code.
//      while(1);                                           // Wait for the watchdog.
        return;
}
```

4.7 Testing the project

At this point, the project should compile without errors and with minimal warnings.

Note: There are three warnings that are acceptable during compilation. They are as follows:

1) "This project is currently in 'manual' Parser Preprocessing mode - no dependency graph is automatically generated. This mode should only be temporarily used to generate various pre-processor listing files. Please switch the Parser Preprocessing mode back to 'automatic' for you regular builds."
2) #1311-D nonstandard conversion between pointer to function and pointer to data"
3) Section .text:retain already has .RETAIN specified - ignoring .CLINK directive

Warning 1 is caused by the settings which make the .pp files appear for use by the memory debugger
Warning 2 is caused by the operations in the software interrupt which copy the program flash modification functions into RAM

Note: When switching back and forth between CCSv3.3 and CCSv5.2 versions, it is necessary to perform a **clean** (if in CCSv5.2) or a **rebuild all** (if in CCSv3.3). CCSv3.3 cannot use object files that were made in CCSv5.2. Likewise, CCSv5.2 cannot use object files that were made in CCSv3.3. New object files must be created during every Code Composer Studio version switch. If this is not done, mysterious error messages are likely to appear.

Try loading the .x0 file onto the target board and ensure that it runs without issue. If there are problems, refer to Section 5 on Troubleshooting, and make sure no mistakes were made while following the procedure in section 4.

5 Troubleshooting

1) **Can't rename project:**
   - **a.** Check the project directory and be sure that .project, and .cproject are not "Read Only" files.
   - **b.** For those using version control software, make sure you have write permissions to the project files mentioned above.
   - **c.** See Section 3.2 for more information.
2) **Can't Import Project: (After browsing, the project found is greyed-out and cannot be checked)**
   - **a.** This is usually due to another project already in CCSv5.2 that has the same name. Rename any other projects that might have the same name as the one being imported, and try again.
3) **There is no .x0 file being generated after a successful compilation:**
   - **a.** Make sure the project name does not have any spaces. For example, if the project is named "Converted Project", rename it to "Converted_Project".
4) **"Implicit Function Declaration" compilation warning**
   - **a.** This can be caused by not including the .h files that have declared the functions used in the file. Make sure they are included as shown in Section 4.6.3.
5) **Memory space overlaps or has already been declared**
   - **a.** This is most likely due to command files being included in the project more than once. Make sure that the command files listed in the project directory are not included in the linker directory, or visa-versa. See Section 4.1.2 and Section 4.3.3 for the solution.
6) **CYGWIN Warning during linking process**
   - **a.** If this error is received during the linking process, the output file will not be generated. See Section 3.3 for the solution.

```
cygwin warning:
  MS-DOS style path detected: Y:\dp_firmware\Cyclone 64\Ian's Test area\llc for both ccs versions\Debug
  Preferred POSIX equivalent is: /cygdrive/y/dp_firmware/Cyclone 64/Ian's Test area/llc for both ccs versions/Debug
  CYGWIN environment variable option "nodosfilewarning" turns off this warning.
  Consult the user's guide for more details about POSIX paths:
    http://cygwin.com/cygwin-ug-net/using.html#using-pathnames
```

Figure 15: CYGWIN Error

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed. TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards. TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions. Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements. TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications. TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use. TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements. Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

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
Wireless Connectivity www.ti.com/wirelessconnectivity	

TI E2E Community Home Page e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

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 JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. 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 as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265