C280x/C2801x C/C++ Header Files and Peripheral Examples Quick Start

1 Device Support: 2
2 Introduction: 2
2.1 Revision History 3
2.2 Where Files are Located (Directory Structure) 4
3 Understanding The Peripheral Bit-Field Structure Approach 5
4 Peripheral Example Projects 6
4.1 Getting Started 6
4.1.1 Getting Started in Code Composer Studio v3.x 6
4.1.2 Getting Started in Code Composer Studio v4 10
4.2 Example Program Structure 16
4.2.1 Include Files 16
4.2.2 Source Code 17
4.2.3 Linker Command Files 17
4.3 Example Program Flow 19
4.4 Included Examples: 20
4.5 Executing the Examples From Flash 22
5 Steps for Incorporating the Header Files and Sample Code 25
5.1 Before you begin 25
5.2 Including the DSP280x Peripheral Header Files 25
5.3 Including Common Example Code 29
6 Troubleshooting Tips & Frequently Asked Questions 33
6.1 Effects of read-modify-write instructions. 35
6.1.1 Registers with multiple flag bits in which writing a 1 clears that flag 35
6.1.2 Registers with Volatile Bits. 36
7 Migration Tips for moving from the TMS320x281x header files to the TMS320x280x header files 37
8 Packet Contents: 39
8.1 Header File Support – DSP280x_headers 39
8.1.1 DSP280x Header Files – Main Files 39
8.1.2 DSP280x Header Files – Peripheral Bit-Field and Register Structure Definition Files 40
8.1.3 Code Composer .gel Files 40
8.1.4 Variable Names and Data Sections 41
8.2 Common Example Code – DSP280x_common 43
8.2.1 Peripheral Interrupt Expansion (PIE) Block Support 43
8.2.2 Peripheral Specific Files 44
8.2.3 Utility Function Source Files 44
8.2.4 Example Linker .cmd files 45
8.2.5 Example Library .lib Files 46
9 Detailed Revision History: 47
10 Errata 55

1 Device Support:

This software package supports 280x and 2801x devices. This includes the following: TMS320F2809, TMS320F2808, TMS320F2806, TMS320F2802, TMS320F2801, TMS320F28015, TMS320F28016, TMS320C2802, and TMS320C2801. The UCD9501 is equivalent to the TMS320F2801 and specifically targets power management control applications.

Throughout this document, TMS320F2809, TMS320F2808, TMS320F2806, TMS320F2802, TMS320F2801/UCD9501, TMS320C2802, TMS320C2801, TMS320F28015, and TMS32028016 are abbreviated as F2809, F2808, F2806, F2802, F2801/9501, F28015, F28016, C2802, and C2801, respectively. TMS320F28015 and TMS320F28016 are abbreviated as F2801x. TMS320x280x device reference guides, flash tools, and other collateral are applicable to the UCD9501 (equivalent replacement is the TMS320F2801) device as well.

2 Introduction:

The C280x/C2801x C/C++ peripheral header files and example projects facilitate writing in C/C++ Code for the Texas Instruments TMS320x280x DSPs. The code can be used as a learning tool or as the basis for a development platform depending on the current needs of the user.

- Learning Tool:

  This download includes several example Code Composer Studio™[†] projects for a ‘280x/2801x development platform. One such platform is the eZdsp™[††] F2808 USB from Spectrum Digital Inc. (www.spectrumdigital.com).

  These examples demonstrate the steps required to initialize the device and utilize the on-chip peripherals. The provided examples can be copied and modified giving the user a platform to quickly experiment with different peripheral configurations.

  These projects can also be migrated to other devices by simply changing the memory allocation in the linker command file.

- Development Platform:

  The peripheral header files can easily be incorporated into a new or existing project to provide a platform for accessing the on-chip peripherals using C or C++ code. In addition, the user can pick and choose functions from the provided code samples as needed and discard the rest.

To get started this document provides the following information:

- Overview of the bit-field structure approach used in the DSP280x C/C++ peripheral header files.

[†] Code Composer Studio is a trademark of Texas Instruments (www.ti.com).
[††] eZdsp is a trademark of Spectrum Digital Inc (www.spectrumdigital.com).
Trademarks are the property of their respective owners.

- Overview of the included peripheral example projects.
- Steps for integrating the peripheral header files into a new or existing project.
- Troubleshooting tips and frequently asked questions.
- Migration tips for users moving from the DSP281x header files to the DSP280x header files.

Finally, this document does not provide a tutorial on writing C code, using Code Composer Studio, or the C28x Compiler and Assembler. It is assumed that the reader already has a 2808 hardware platform setup and connected to a host with Code Composer Studio installed. The user should have a basic understanding of how to use Code Composer Studio to download code through JTAG and perform basic debug operations.

2.1 Revision History

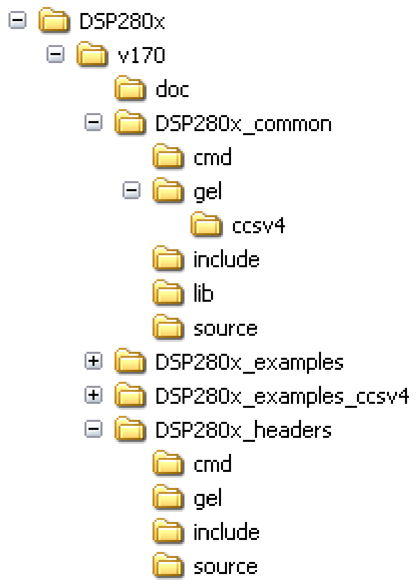
V1.70 makes some minor corrections and comment fixes to the header files and examples, and also adds a separate example folder, DSP280x_examples_ccsv4, with examples supported by the Eclipse-based Code Composer Studio v4. V1.60 makes some minor corrections. V1.51 adds support for SFO_TI_Build_V5B.lib and fixes some typos. V1.50 is a minor release to fix multiple typos and to add support for 60 MHz devices in various files. V1.41 is a minor release to fix a stack size allocation issue for some examples. V1.40 was a minor update to incorporate the TMS320F2809, TMS320F28015 and F28016 devices and to make minor corrections.

A detailed revision history can be found in Section 9.

2.2 Where Files are Located (Directory Structure)

As installed, the *C280x/C2801x C/C++ Header Files and Peripheral Examples* is partitioned into a well-defined directory structure. By default, the source code is installed into the c:\tidcs\c28\DSP280x\<version> directory.

Table 1 describes the contents of the main directories used by DSP280x/2801x header files and peripheral examples:



```
DSP280x
  v170
    doc
    DSP280x_common
      cmd
      gel
        ccsv4
      include
      lib
      source
    DSP280x_examples
    DSP280x_examples_ccsv4
    DSP280x_headers
      cmd
      gel
      include
      source
```

Table 1. DSP280x/2801x Main Directory Structure

Directory	Description
<base>	Base install directory. By default this is c:\tidcs\c28\DSP280x\v150. For the rest of this document <base> will be omitted from the directory names.
<base>\doc	Documentation including the revision history from the previous release.
<base>\DSP280x_headers	Files required to incorporate the peripheral header files into a project . The header files use the bit-field structure approach described in Section 3. Integrating the header files into a new or existing project is described in Section 5.
<base>\DSP280x_examples	Example Code Composer Studio projects. These example projects illustrate how to configure many of the on-chip peripherals. An overview of the examples is given in Section 4.
<base>\DSP280x_examples_ccsv4	Example Code Composer Studio v4 projects compiled with floating point unit *enabled*. These examples are identical to those in the \DSP280x_examples directory, but are generated for CCSv4 and cannot be run in CCSv3.x. An overview of the examples is given in Section 4.
<base>DSP280x_common	Common source files shared across example projects to illustrate how to perform tasks using header file approach. Use of these files is optional, but may be useful in new projects. A list of these files is in Section 7.

Under the *DSP280x_headers* and *DSP280x_common* directories the source files are further broken down into sub-directories each indicating the type of file. Table 2 lists the sub-directories and describes the types of files found within each:

Table 2. DSP280x/2801x Sub-Directory Structure

Sub-Directory	Description
DSP280x_headers\cmd	Linker command files that allocate the bit-field structures described in Section 3.
DSP280x_headers\source	Source files required to incorporate the header files into a new or existing project.
DSP280x_headers\include	Header files for each of the on-chip peripherals.
DSP280x_common\cmd	Example memory command files that allocate memory on the devices.
DSP280x_common\include	Common .h files that are used by the peripheral examples.
DSP280x_common\source	Common .c files that are used by the peripheral examples.
DSP280x_common\lib	Common library (.lib) files that are used by the peripheral examples.
DSP280x_common\gel	Code Composer Studio v3.3 GEL files for each device. These are optional.
DSP280x_common\gel\ccsv4	Code Composer Studio v4.x GEL files for each device. These are optional.

3 Understanding The Peripheral Bit-Field Structure Approach

The following application note includes useful information regarding the bit-field peripheral structure approach used by the header files and examples.

This method is compared to traditional #define macros and topics of code efficiency and special case registers are also addressed. The information in this application note is important to understand the impact using bit fields can have on your application code.

Programming TMS32028xx and 28xxx Peripherals in C/C++ (SPRAA85)

4 Peripheral Example Projects

4.1 Getting Started

4.1.1 Getting Started in Code Composer Studio v3.x

To get started, follow these steps to load the 32-bit CPU-Timer example. Other examples are set-up in a similar manner.

1. **Have a hardware platform, such as the eZdsp F2808 USB, connected to a host with Code Composer Studio installed.**

   NOTE: As supplied, the example projects are built for the '2808 device. If you are using another 280x/2801x device the memory definition in the linker command file (.cmd) will need to be changed and the project rebuilt.

2. **Load the example's GEL file (.gel) or Project file (.pjt).**

   Each example includes a Code Composer Studio GEL file to help automate loading of the project, compiling of the code and populating of the watch window. Alternatively, the project file itself (.pjt) can be loaded instead of using the included GEL file.

   To load the CPU-Timer example's GEL file follow these steps:

   a. In Code Composer Studio: *File->Load GEL*

   b. Browse to the CPU Timer example directory: *DSP280x_examples\cpu_timer*

   c. Select *Example_280xCpuTimer.gel* and click on *open*.

   d. From the Code Composer GEL pull-down menu select

      DSP280x CpuTimerExample-> Load_and_Build_Project

      This will load the project and build compile the project.

3. **Edit DSP28_Device.h**

   Edit the DSP280x_Device.h file and make sure the appropriate device is selected. By default the 2808 is selected.

```
/********************************************************************
* DSP280x_headers\include\DSP280x_Device.h
********************************************************************/

#define   TARGET   1
//----------------------------------------------------------------------------
// User To Select Target Device:

#define   DSP28_28015  0
#define   DSP28_28016  0
#define   DSP28_2809   0
#define   DSP28_2808   TARGET
#define   DSP28_2806   0
#define   DSP28_2802   0
#define   DSP28_2801   0
```

4. Edit DSP280x_Examples.h

Edit DSP280x_Examples.h and specify the clock rate, the PLL control register value (PLLCR and CLKINDIV). These values will be used by the examples to initialize the PLLCR register and CLKINDIV bit.

The default values will result in a 100Mhz SYSCLKOUT frequency. If you have a 60Mhz device you will need to adjust these settings accordingly.

```
/********************************************************************
* DSP280x_common\include\DSP280x_Examples.h
********************************************************************/
/*-----------------------------------------------------------------------------
      Specify the PLLCR and CLKINDIV value.

      if CLKINDIV = 0: SYSCLKOUT = (OSCCLK * PLLCR)/2
      if CLKINDIV = 1: SYSCLKOUT = (OSCCLK * PLLCR)
-----------------------------------------------------------------------------*/
#define DSP28_CLKINDIV   0     // Enable /2 for SYSCLKOUT
//#define DSP28_CLKINDIV   1   // Disable /2 for SYSCLKOUT

#define DSP28_PLLCR   10
//#define DSP28_PLLCR    9
//#define DSP28_PLLCR    8
//#define DSP28_PLLCR    7
//#define DSP28_PLLCR    6 // Uncomment for 60 MHz devices [60 MHz = (20MHz * 6)/2]
//#define DSP28_PLLCR    5
//#define DSP28_PLLCR    4
//#define DSP28_PLLCR    3
//#define DSP28_PLLCR    2
//#define DSP28_PLLCR    1
//#define DSP28_PLLCR    0  // (Default at reset) PLL is bypassed in this mode
//----------------------------------------------------------------------------
```

In DSP280x_Examples.h, also specify the SYSCLKOUT rate. This value is used to scale a delay loop used by the examples. The default value is for a 100Mhz SYSCLKOUT. If you have a 60 MHz device you will need to adjust these settings accordingly.

```
/********************************************************************
* DSP280x_common\include\DSP280x_Examples.h
********************************************************************/
……
#define CPU_RATE   10.000L   // for a 100MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   13.330L   // for a 75MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   16.667L   // for a 60MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   20.000L   // for a 50MHz CPU clock speed  (SYSCLKOUT)
```

In DSP280x_Examples.h specify the maximum SYSCLKOUT frequency (100MHz or 60MHz) by setting it to 1 and the other to 0. This value is used by those examples with timing dependent code (i.e. baud rates or other timing parameters) to determine whether 100 MHz code or 60 MHz code should be run.

The default value is for 100Mhz SYSCLKOUT. If you have a 60 MHz device you will need to adjust these settings accordingly. If you intend to run examples which use these definitions at a different frequency, then the timing parameters in those examples must be directly modified accordingly regardless of the setting here.

```
/********************************************************************
* DSP280x_common\include\DSP280x_Examples.h
********************************************************************/
……
#define CPU_FRQ_100MHZ   1     // 100 Mhz CPU Freq - default, 1 for 100 MHz devices
#define CPU_FRQ_60MHZ    0     // 60 MHz CPU Freq - 1 for 60 MHz devices
//---------------------------------------------------------------------------
……
```

For 60 MHz operation, certain files (listed below) must be modified for the examples to work properly. In the list below, files with * indicate that code modifications are required by the user in the DSP280x_Examples.h file to operate at 60 MHz. If the user wants to run these example files at frequencies other than 100MHz or 60MHz, the user must modify these files directly with the appropriate timing values.

Common Files:

- ECan.c* in the DSP280x_common\source\ directory
- usDelay.asm in the DSP280x_common\source\ directory
- Examples.h* in the DSP280x_common\include\ directory

Example Files:

- All HRPWM example source files

- Example_280xI2C_eeprom.c*
- All ADC examples
- Example_280xECap_apwm.c*
- Example_280xSci_Echoback.c8
- Example_280xEqep_pos_speed.c* and related files (Example_posspeed.c*, Example_posspeed.h*, Example_EPwmSetup.c*, Example_posspeed.xls)
- Example_280xEqep_freqcal.c* and related files (Example_freqcal.c*, Example_freqcal.h*, Example_EPwmSetup.c*, Example_freqcal.xls)

5. **Review the comments at the top of the main source file: Example_280xCpuTimer.c.**

   A brief description of the example and any assumptions that are made and any external hardware requirements are listed in the comments at the top of the main source file of each example. In some cases you may be required to make external connections for the example to work properly.

6. **Perform any hardware setup required by the example.**

   Perform any hardware setup indicated by the comments in the main source. The CPU-Timer example only requires that the hardware be setup for "Boot to SARAM" mode. Other examples may require additional hardware configuration such as connecting pins together or pulling a pin high or low.

   Table 3 shows a listing of the boot mode pin settings for your reference. Refer to the documentation for your hardware platform for information on configuring the boot mode pins. For more information on the '280x boot modes refer to the device specific *Boot ROM Reference Guide*.

Table 3. 280x/2801x Boot Mode Settings

GPIO18	GPIO29	GPIO34	Mode
1	1	1	Boot to flash 0x3F7FF6
1	1	0	Call SCI-A boot loader
1	0	1	Call SPI-A boot loader
1	0	0	Call I2C boot loader
0	1	1	Call eCAN-A boot loader [1]
0	1	0	Boot to M0 SARAM 0x000000
0	0	1	Boot to OTP 0x3D7800
0	0	0	Call parallel boot loader

Note 1: The eCAN boot mode is reserved on devices that do not include the eCAN-A module. Should it be selected, the boot ROM code will loop as if waiting for a CAN message to arrive.

7. **Load the code**

   Once any hardware configuration has been completed, from the Code Composer GEL pull-down menu select

DSP280x CpuTimerExample-> Load_Code

This will load the .out file into the 28x device, populate the watch window with variables of interest, reset the part and execute code to the start of the main function. The GEL file is setup to reload the code every time the device is reset so if this behavior is not desired, the GEL file can be removed at this time. To remove the GEL file, right click on its name and select *remove*.

8. **Run the example, add variables to the watch window or examine the memory contents.**

9. **Experiment, modify, re-build the example.**

   If you wish to modify the examples it is suggested that you make a copy of the entire header file packet to modify or at least create a backup of the original files first. New examples provided by TI will assume that the base files are as supplied.

   Sections 4.2 and 4.3 describe the structure and flow of the examples in more detail.

10. **When done, remove the example's GEL file and project from Code Composer Studio.**

    To remove the GEL file, right click on its name and select *remove*.The examples use the header files in the *DSP280x_headers* directory and shared source in the *DSP280x_common* directory. Only example files specific to a particular example are located within in the example directory.

    Note: Most of the example code included uses the .bit field structures to access registers. This is done to help the user learn how to use the peripheral and device. Using the bit fields has the advantage of yielding code that is easier to read and modify. This method will result in a slight code overhead when compared to using the .all method. In addition, the example projects have the compiler optimizer turned off. The user can change the compiler settings to turn on the optimizer if desired.

4.1.2 Getting Started in Code Composer Studio v4

To get started, follow these steps to load the 32-bit CPU-Timer example. Other examples are set-up in a similar manner.

1. **Have a hardware platform connected to a host with Code Composer Studio installed.**

   NOTE: As supplied, the '280x example projects are built for the '2808 device. If you are using another 280x device, the memory definition in the linker command file (.cmd) will need to be changed and the project rebuilt.

2. **Open the example project.**

   Each example has its own project directory which is "imported"/opened in Code Composer Studio v4.

   To open the '280x CPU-Timer example project directory, follow the following steps:

   a. In Code Composer Studio v 4.x: Project->Import Existing CCS/CCE Eclipse Project.

   b. Next to "Select Root Directory", browse to the CPU Timer example directory: *DSP280x_examples_ccsv4\cpu_timer*. Select the *Finish* button.

   This will import/open the project in the CCStudio v4 C/C++ Perspective project.

3. Edit DSP28_Device.h

Edit the DSP280x_Device.h file and make sure the appropriate device is selected. By default the 2808 is selected.

```
/*********************************************************************
* DSP280x_headers\include\DSP280x_Device.h
*********************************************************************/

#define   TARGET   1
//---------------------------------------------------------------------------
// User To Select Target Device:

#define   DSP28_28015  0
#define   DSP28_28016  0
#define   DSP28_2809   0
#define   DSP28_2808   TARGET
#define   DSP28_2806   0
#define   DSP28_2802   0
#define   DSP28_2801   0
```

4. Edit DSP280x_Examples.h

Edit DSP280x_Examples.h and specify the clock rate, the PLL control register value (PLLCR and CLKINDIV). These values will be used by the examples to initialize the PLLCR register and CLKINDIV bit.

The default values will result in a 100Mhz SYSCLKOUT frequency. If you have a 60Mhz device you will need to adjust these settings accordingly.

```
/********************************************************************
* DSP280x_common\include\DSP280x_Examples.h
********************************************************************/
/*-----------------------------------------------------------------------------
      Specify the PLLCR and CLKINDIV value.

      if CLKINDIV = 0: SYSCLKOUT = (OSCCLK * PLLCR)/2
      if CLKINDIV = 1: SYSCLKOUT = (OSCCLK * PLLCR)
-----------------------------------------------------------------------------*/
#define DSP28_CLKINDIV   0     // Enable /2 for SYSCLKOUT
//#define DSP28_CLKINDIV   1   // Disable /2 for SYSCLKOUT

#define DSP28_PLLCR   10
//#define DSP28_PLLCR    9
//#define DSP28_PLLCR    8
//#define DSP28_PLLCR    7
//#define DSP28_PLLCR    6 // Uncomment for 60 MHz devices [60 MHz = (20MHz * 6)/2]
//#define DSP28_PLLCR    5
//#define DSP28_PLLCR    4
//#define DSP28_PLLCR    3
//#define DSP28_PLLCR    2
//#define DSP28_PLLCR    1
//#define DSP28_PLLCR    0  // (Default at reset) PLL is bypassed in this mode
//---------------------------------------------------------------------------
```

In DSP280x_Examples.h, also specify the SYSCLKOUT rate. This value is used to scale a delay loop used by the examples. The default value is for a 100Mhz SYSCLKOUT. If you have a 60 MHz device you will need to adjust these settings accordingly.

```
/********************************************************************
* DSP280x_common\include\DSP280x_Examples.h
********************************************************************/
……
#define CPU_RATE   10.000L   // for a 100MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   13.330L   // for a 75MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   16.667L   // for a 60MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   20.000L   // for a 50MHz CPU clock speed  (SYSCLKOUT)
```

In DSP280x_Examples.h specify the maximum SYSCLKOUT frequency (100MHz or 60MHz) by setting it to 1 and the other to 0. This value is used by those examples with timing dependent code (i.e. baud rates or other timing parameters) to determine whether 100 MHz code or 60 MHz code should be run.

The default value is for 100Mhz SYSCLKOUT. If you have a 60 MHz device you will need to adjust these settings accordingly. If you intend to run examples which use these definitions at a different frequency, then the timing parameters in those examples must be directly modified accordingly regardless of the setting here.

```
/*******************************************************************
* DSP280x_common\include\DSP280x_Examples.h
******************************************************************/
……
#define CPU_FRQ_100MHZ   1     // 100 Mhz CPU Freq - default, 1 for 100 MHz devices
#define CPU_FRQ_60MHZ    0     // 60 MHz CPU Freq - 1 for 60 MHz devices
//-----------------------------------------------------------------------------
……
```

For 60 MHz operation, certain files (listed below) must be modified for the examples to work properly. In the list below, files with * indicate that code modifications are required by the user in the DSP280x_Examples.h file to operate at 60 MHz. If the user wants to run these example files at frequencies other than 100MHz or 60MHz, the user must modify these files directly with the appropriate timing values.

Common Files:

- ECan.c* in the DSP280x_common\source\ directory
- usDelay.asm in the DSP280x_common\source\ directory
- Examples.h* in the DSP280x_common\include\ directory

Example Files:

- All HRPWM example source files
- Example_280xI2C_eeprom.c*
- All ADC examples
- Example_280xECap_apwm.c*
- Example_280xSci_Echoback.c8
- Example_280xEqep_pos_speed.c* and related files (Example_posspeed.c*, Example_posspeed.h*, Example_EPwmSetup.c*, Example_posspeed.xls)
- Example_280xEqep_freqcal.c* and related files (Example_freqcal.c*, Example_freqcal.h*, Example_EPwmSetup.c*, Example_freqcal.xls)

5. **Review the comments at the top of the main source file: Example_280xCpuTimer.c.**

A brief description of the example and any assumptions that are made and any external hardware requirements are listed in the comments at the top of the main source file of each example. In some cases you may be required to make external connections for the example to work properly.

6. Perform any hardware setup required by the example.

Perform any hardware setup indicated by the comments in the main source. The CPU-Timer example only requires that the hardware be setup for "Boot to SARAM" mode. Other examples may require additional hardware configuration such as connecting pins together or pulling a pin high or low.

Table 3 shows a listing of the boot mode pin settings for your reference. Refer to the documentation for your hardware platform for information on configuring the boot mode pins. For more information on the '280x boot modes refer to the device specific *Boot ROM Reference Guide*.

Table 4. 280x/2801x Boot Mode Settings

GPIO18	GPIO29	GPIO34	Mode
1	1	1	Boot to flash 0x3F7FF6
1	1	0	Call SCI-A boot loader
1	0	1	Call SPI-A boot loader
1	0	0	Call I2C boot loader
0	1	1	Call eCAN-A boot loader [1]
0	1	0	Boot to M0 SARAM 0x000000
0	0	1	Boot to OTP 0x3D7800
0	0	0	Call parallel boot loader

Note 1: The eCAN boot mode is reserved on devices that do not include the eCAN-A module. Should it be selected, the boot ROM code will loop as if waiting for a CAN message to arrive.

7. Build and Load the code

Once any hardware configuration has been completed, in Code Composer Studio v4, go to *Target->Debug Active Project.*

This will open the "Debug Perspective" in CCSv4, build the project, load the .out file into the 28x device, reset the part, and execute code to the start of the main function. By default, in Code Composer Studio v4, every time *Debug Active Project* is selected, the code is automatically built and the .out file loaded into the 28x device.

8. Run the example, add variables to the watch window or examine the memory contents.

At the top of the code in the comments section, there should be a list of "Watch variables". To add these to the watch window, highlight them and right-click. Then select *Add Watch expression*. Now variables of interest are added to the watch window.

9. Experiment, modify, re-build the example.

If you wish to modify the examples it is suggested that you make a copy of the entire header file packet to modify or at least create a backup of the original files first. New examples provided by TI will assume that the base files are as supplied.

Sections 4.2 and 4.3 describe the structure and flow of the examples in more detail.

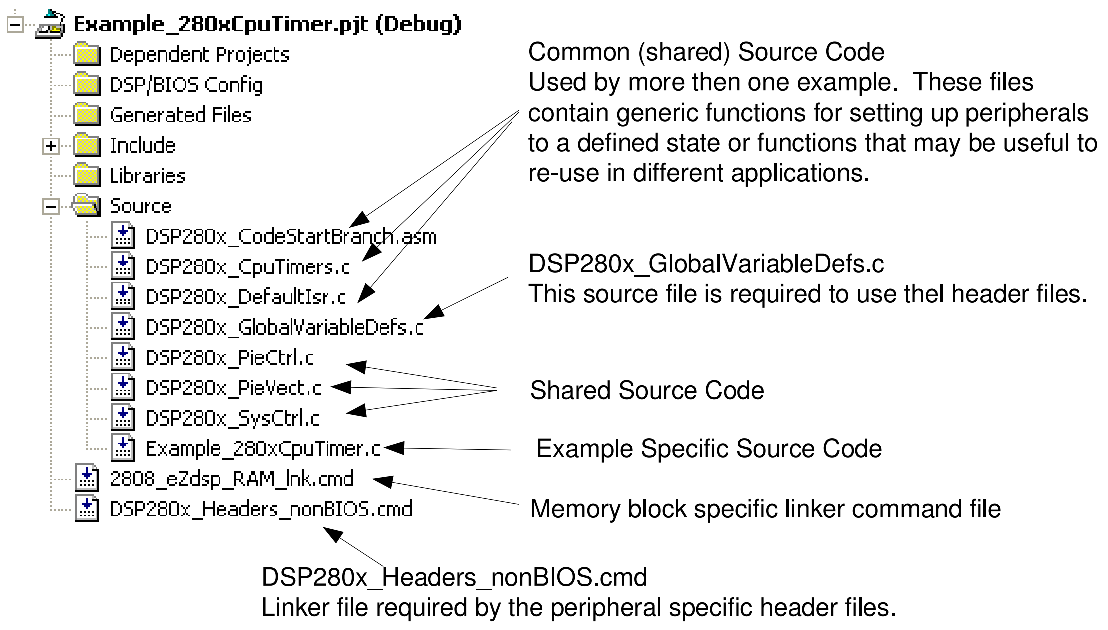
10. When done, delete the project from the Code Composer Studio v4 workspace.

Go *to View->C/C++ Projects* to open up your project view. To remove/delete the project from the workspace, right click on the project's name and select *delete*. Make sure the *Do not delete* contents button is selected, then select *Yes.* This does not delete the project itself. It merely removes the project from the workspace until you wish to open/import it again.

The examples use the header files in the *DSP2802x_headers* directory and shared source in the *DSP2802x_common* directory. Only example files specific to a particular example are located within in the example directory.

Note: Most of the example code included uses the .bit field structures to access registers. This is done to help the user learn how to use the peripheral and device. Using the bit fields has the advantage of yielding code that is easier to read and modify. This method will result in a slight code overhead when compared to using the .all method. In addition, the example projects have the compiler optimizer turned off. The user can change the compiler settings to turn on the optimizer if desired.

4.2 Example Program Structure


Example_280xCpuTimer.pjt (Debug)
Dependent Projects
DSP/BIOS Config
Generated Files
Include
Libraries
Source
DSP280x_CodeStartBranch.asm
DSP280x_CpuTimers.c
DSP280x_DefaultIsr.c
DSP280x_GlobalVariableDefs.c
DSP280x_PieCtrl.c
DSP280x_PieVect.c
DSP280x_SysCtrl.c
Example_280xCpuTimer.c
2808_eZdsp_RAM_lnk.cmd
DSP280x_Headers_nonBIOS.cmd
Common (shared) Source Code
Used by more then one example. These files contain generic functions for setting up peripherals to a defined state or functions that may be useful to re-use in different applications.
DSP280x_GlobalVariableDefs.c
This source file is required to use thel header files.
Shared Source Code
Example Specific Source Code
Memory block specific linker command file
DSP280x_Headers_nonBIOS.cmd
Linker file required by the peripheral specific header files.

Each of the example programs has a very similar structure. This structure includes unique source code, shared source code, header files and linker command files.

4.2.1 Include Files

All of the example source code #include two header files as shown below:

```
/*********************************************************************
* DSP280x_examples\cpu_timer\Example_280xCpuTimer.c
*********************************************************************/

#include "DSP280x_Device.h"     // DSP280x Headerfile Include File
#include "DSP280x_Examples.h"   // DSP280x Examples Include File
```

- **DSP280x_Device.h**

  This header file is required to use the header files. This file includes all of the required peripheral specific header files and includes device specific macros and typedef statements. This file is found in the *<base>\DSP280x_headers\include* directory.

- **DSP280x_Examples.h**

  This header file defines parameters that are used by the example code. This file is not required to use just the DSP280x peripheral header files but is required by some of the common source files. This file is found in the *<base>\DSP280x_common\include* directory.

4.2.2 Source Code

Each of the example projects consists of source code that is unique to the example as well as source code that is common or shared across examples.

- **DSP280x_GlobalVariableDefs.c**

  Any project that uses the DSP280x peripheral header files must include this source file. In this file are the declarations for the peripheral register structure variables and data section assignments. This file is found in the *<base>\DSP280x_headers\source* directory.

- **Example specific source code:**

  Files that are specific to a particular example have the prefix Example_280x on their filename. For example Example_280xCpuTimer.c is specific to the CPU Timer example and not used for any other example. Example specific files are located in the *<base>\DSP280x_examples\<example>* directory.

- **Common source code:**

  The remaining source files are shared across the examples. These files contain common functions for peripherals or useful utility functions that may be re-used. Shared source files are located in the DSP280x_common\source directory. Users may choose to incorporate none, some, or the entire shared source into their own new or existing projects.

4.2.3 Linker Command Files

Each example uses two linker command files. These files specify the memory where the linker will place code and data sections. One linker file is used for assigning compiler generated sections to the memory blocks on the device while the other is used to assign the data sections of the peripheral register structures used by the DSP280x peripheral header files.

- **Memory block linker allocation:**

The linker files shown in Table 5 are used to assign sections to memory blocks on the device. These linker files are located in the *<base>\DSP280x_common\cmd* directory. Each example will use one of the following files depending on the memory used by the example.

Table 5. Included Memory Linker Command Files

Memory Linker Command File Examples	Location	Description
2808_eZdsp_RAM_lnk.cmd	DSP280x_common\cmd	eZdsp F2808 USB memory map that only allocates SARAM locations. No Flash, OTP, or CSM password protected locations (L0/L1) are used. This linker command file is used for most of the examples.
2809_RAM_lnk.cmd	DSP280x_common\cmd	2809 memory linker command file. Includes all of the internal SARAM blocks on a 2809 device. "RAM" linker files do not include flash or OTP blocks.
2808_RAM_lnk.cmd	DSP280x_common\cmd	2808 SARAM memory linker command file.
2806_RAM_lnk.cmd	DSP280x_common\cmd	2806 SARAM memory linker command file.
2801_RAM_lnk.cmd	DSP280x_common\cmd	2801 SARAM memory linker command file.
28015_RAM_lnk.cmd	DSP280x_common\cmd	28015 SARAM memory linker command file.
28016_RAM_lnk.cmd	DSP280x_common\cmd	28016 SARAM memory linker command file.
F2809.cmd	DSP280x_common\cmd	F2809 memory linker command file. Includes all Flash, OTP and CSM password protected memory locations.
F2808.cmd	DSP280x_common\cmd	F2808 memory linker command file
F2806.cmd	DSP280x_common\cmd	F2806 memory linker command file.
F2802.cmd	DSP280x_common\cmd	F2802 memory linker command file.
F2801.cmd	DSP280x_common\cmd	F2801 memory linker command file.
F28015.cmd	DSP280x_common\cmd	F28015 memory linker command file.
F28016.cmd	DSP280x_common\cmd	F28016 memory linker command file.

- **Header file structure data section allocation:**

  Any project that uses the header file peripheral structures must include a linker command file that assigns the peripheral register structure data sections to the proper memory location. These files are described in Table 6.

Table 6. DSP280x/2801x Peripheral Header Linker Command File

Header File Linker Command File	Location	Description
DSP280x_Headers_BIOS.cmd	DSP280x_headers\cmd	Linker .cmd file to assign the header file variables in a BIOS project. This file must be included in any BIOS project that uses the header files. Refer to section 5.2.
DSP280x_Headers_nonBIOS.cmd	DSP280x_headers\cmd	Linker .cmd file to assign the header file variables in a non-BIOS project. This file must be included in any non-BIOS project that uses the header files. Refer to section 5.2.

4.3 Example Program Flow

All of the example programs follow a similar recommended flow for setting up a 280x/2801x device. Figure 1 outlines this basic flow:

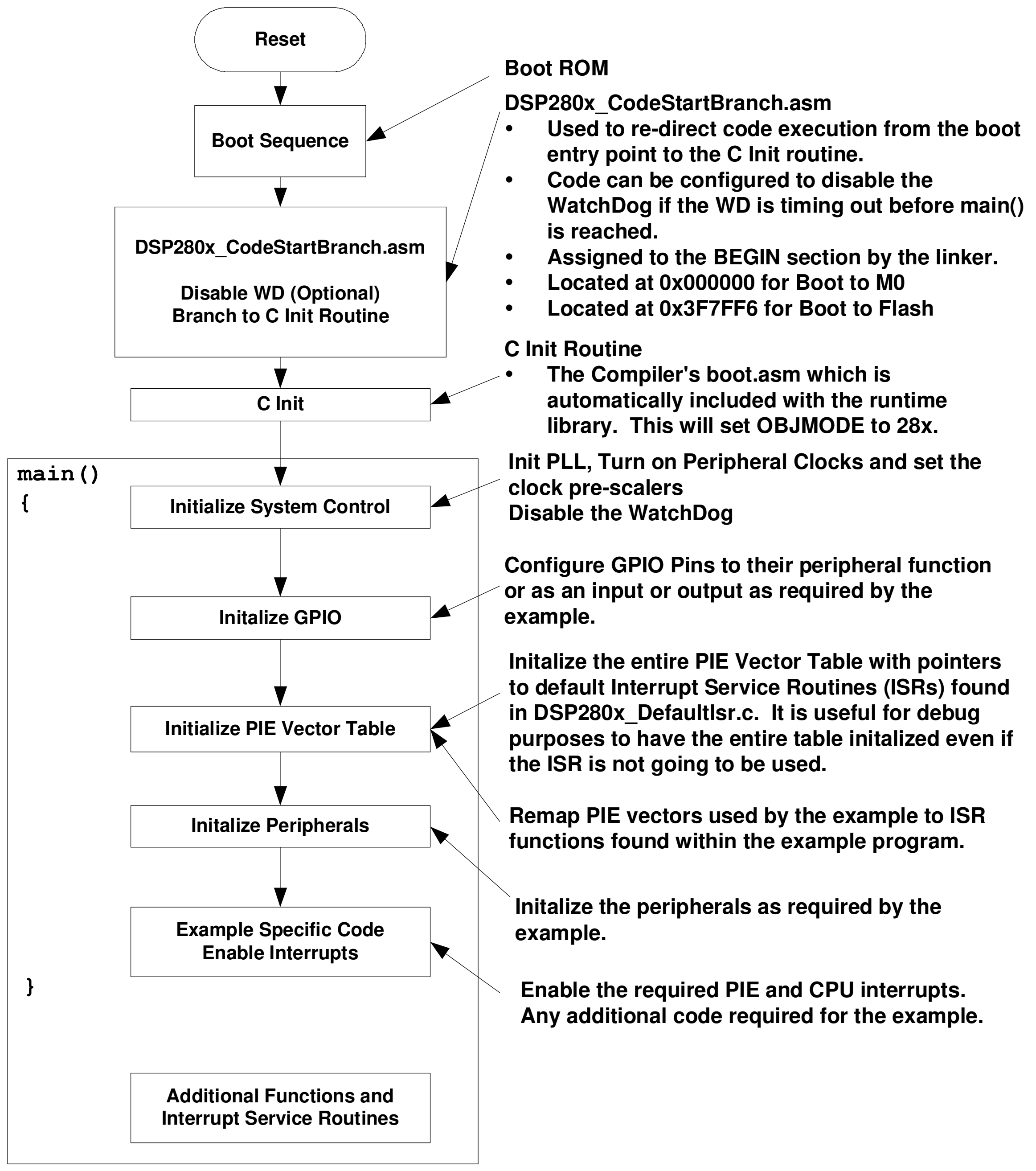

Reset
Boot Sequence
DSP280x_CodeStartBranch.asm
Disable WD (Optional)
Branch to C Init Routine
C Init
main()
{
Initialize System Control
Initalize GPIO
Initialize PIE Vector Table
Initalize Peripherals
Example Specific Code
Enable Interrupts
}
Additional Functions and Interrupt Service Routines
Boot ROM
DSP280x_CodeStartBranch.asm
• Used to re-direct code execution from the boot entry point to the C Init routine.
• Code can be configured to disable the WatchDog if the WD is timing out before main() is reached.
• Assigned to the BEGIN section by the linker.
• Located at 0x000000 for Boot to M0
• Located at 0x3F7FF6 for Boot to Flash
C Init Routine
• The Compiler's boot.asm which is automatically included with the runtime library. This will set OBJMODE to 28x.
Init PLL, Turn on Peripheral Clocks and set the clock pre-scalers
Disable the WatchDog
Configure GPIO Pins to their peripheral function or as an input or output as required by the example.
Initalize the entire PIE Vector Table with pointers to default Interrupt Service Routines (ISRs) found in DSP280x_DefaultIsr.c. It is useful for debug purposes to have the entire table initalized even if the ISR is not going to be used.
Remap PIE vectors used by the example to ISR functions found within the example program.
Initalize the peripherals as required by the example.
Enable the required PIE and CPU interrupts.
Any additional code required for the example.

Figure 1. Flow for Example Programs

4.4 Included Examples:

Table 7. Included Examples

Example	Description
adc_seq_ovd_tests	ADC test using the sequencer override feature.
adc_seqmode_test	ADC Seq Mode Test. Channel A0 is converted forever and logged in a buffer
adc_soc	ADC example to convert two channels: ADCINA3 and ADCINA2. Interrupts are enabled and PWM1 is configured to generate a periodic ADC SOC on SEQ1.
cpu_timer	Configures CPU Timer0 and increments a count each time the ISR is serviced.
ecan_a_to_b_xmit	Transmit from eCANa to eCANb
ecan_back2back	eCAN self-test mode example. Transmits eCAN data back-to-back at high speed without stopping.
ecap_apwm	This example sets up the alternate eCAP pins in the APWM mode
ecap_capture_pwm	Captures the edges of a ePWM signal.
epwm_deadband	Example deadband generation via ePWM3
epwm_timer_interrupts	Starts ePWM1-ePWM6 timers. Every period an interrupt is taken for each ePWM.
epwm_trip_zone	Uses the trip zone signals to set the ePWM signals to a particular state.
epwm_up_aq	Generate a PWM waveform using an up count time base ePWM1-ePWM3 are used.
epwm_updown_aq	Generate a PWM waveform using an up/down time base. ePWM1 – ePWM3 are used.
eqep_freqcal	Frequency cal using eQEP1
eqep_pos_speed	Pos/speed calculation using eQEP1
external_interrupt	Configures GPIO0 as XINT1 and GPIO1 as XINT2. The interrupts are fired by toggling GPIO30 and GPIO31 which are connected to XINT1 (GPIO0) and XINT2 (GPIO1) externally by the user.
flash	ePWM timer interrupt project moved from SARAM to Flash. Includes steps that were used to convert the project from SARAM to Flash. Some interrupt service routines are copied from FLASH to SARAM for faster execution.
gpio_setup	Three examples of different pinout configurations.
gpio_toggle	Toggles all of the I/O pins using different methods – DATA, SET/CLEAR and TOGGLE registers. The pins can be observed using an oscilloscope.
hrpwm	Sets up ePWM1-ePWM4 and controls the edge of output A using the HRPWM extension. Both rising edge and falling edge are controlled.
hrpwm_sfo	Use TI's MEP Scale Factor Optimizer (SFO) library to change the HRPWM. This version of the SFO library supports HRPWM on ePWM channels 1-4 only.
hrpwm_sfo_v5	Use TI’s MEP Scale Factor Optimizer (SFO) library version 5 to change the HRPWM. This version of the SFO library supports HRPWM on up to 16 ePWM channels (if available)
hrpwm_slider	This is the same as the hrpwm example except the control of CMPAHR is now controlled by the user via a slider bar. The included .gel file sets up the slider.
i2c_eeprom	Communicate with the EEPROM on the eZdsp F2808 USB platform via I2C
lpm_halt	Puts device into low power halt mode. GPIO0 is configured to wake the device from halt when an external high-low-high pulse is applied to it.
lpm_idle	Puts device into low power idle mode. GPIOE0 is configured as XINT1 pin. When an XINT1 interrupt occurs due to a falling edge on GPIOE0, the device is woken from idle.
lpm_standby	Puts device into low power standby mode. The watchdog interrupt is used to wake the device from standby mode.

Included Examples Continued…

sci_autobaud	Externally connect SCI-A to SCI-B and send data between the two peripherals. Baud lock is performed using the autobaud feature of the SCI. This test is repeated for different baud rates.
sci_echoback	SCI-A example that can be used to echoback to a terminal program such as hyperterminal. A transceiver and a connection to a PC is required.
scia_loopback	SCI-A example that uses the peripheral's loop-back test mode to send data.
scia_loopback_interrupts	SCI-A example that uses the peripheral's loop-back test mode to send data. Both interrupts and FIFOs are used in this example.
spi_loopback	SPI-A example that uses the peripherals loop-back test mode to send data.
spi_loopback_interrupts	SPI-A example that uses the peripherals loop-back test mode to send data. Both interrupts and FIFOs are used in this example.
sw_prioritized_interrupts	The standard hardware prioritization of interrupts can be used for most applications. This example shows a method for software to re-prioritize interrupts if required.
Watchdog	Illustrates feeding the dog and re-directing the watchdog to an interrupt.

4.5 Executing the Examples From Flash

Most of the DSP280x examples execute from SARAM in “boot to SARAM” mode. One example, *DSP280x_examples\Flash*, executes from flash memory in “boot to flash” mode. This example is the PWM timer interrupt example with the following changes made to execute out of flash:

1. **Change the linker command file to link the code to flash.**

   Remove 2808_eZdsp_RAM_lnk.cmdfrom the project and add one of the flash based linker files (ex: F2809.cmd, F2808.cmd, F2806.cmd, F2802.cmd, F2801.cmd F28015.cmd, or F28016). These files are located in the *<base>DSP280x_common\cmd\ directory*.

2. **Add the *DSP280x_common\source\DSP280x_CSMPasswords.asm* to the project.**

   This file contains the passwords that will be programmed into the Code Security Module (CSM) password locations. Leaving the passwords set to 0xFFFF during development is recommended as the device can easily be unlocked. For more information on the CSM refer to the appropriate *System Control and Interrupts Reference Guide*.

3. **Modify the source code to copy all functions that must be executed out of SARAM from their load address in flash to their run address in SARAM.**

   In particular, the flash wait state initialization routine must be executed out of SARAM. In the DSP280x examples, functions that are to be executed from SARAM have been assigned to the ramfuncs section by compiler CODE_SECTION #pragma statements as shown in the example below.

```
/********************************************************************
* DSP280x_common\source\DSP280x_SysCtrl.c
********************************************************************/

#pragma CODE_SECTION(InitFlash, "ramfuncs");
```

The ramfuncs section is then assigned to a load address in flash and a run address in SARAM by the memory linker command file as shown below:

```
/********************************************************************
* DSP280x_common\include\F2808.cmd
********************************************************************/
SECTIONS
{
   ramfuncs    : LOAD = FLASHD,
                 RUN = RAML0,
                 LOAD_START(_RamfuncsLoadStart),
                 LOAD_END(_RamfuncsLoadEnd),
                 RUN_START(_RamfuncsRunStart),
                 PAGE = 0
}
```

The linker will assign symbols as specified above to specific addresses as follows:

Address	Symbol
Load start address	RamfuncsLoadStart
Load end address	RamfuncsLoadEnd
Run start address	RamfuncsRunStart

These symbols can then be used to copy the functions from the Flash to SARAM using the included example MemCopy routine or the C library standard memcopy() function.

To perform this copy from flash to SARAM using the included example MemCopy function:

a. Add the file *DSP280x_common\source\DSP280x_MemCopy.c* to the project.

b. Add the following function prototype to the example source code. This is done for you in the *DSP280x_Examples.h* file.

```
/********************************************************************
* DSP280x_common\include\DSP280x_Examples.h
********************************************************************/

MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
```

c. Add the following variable declaration to your source code to tell the compiler that these variables exist. The linker command file will assign the address of each of these variables as specified in the linker command file as shown in step 3. For the DSP280x example code this has is already done in *DSP280x_Examples.h*.

```
/********************************************************************
* DSP280x_common\include\DSP280x_GlobalPrototypes.h
********************************************************************/

extern Uint16 RamfuncsLoadStart;
extern Uint16 RamfuncsLoadEnd;
extern Uint16 RamfuncsRunStart;
```

d. Modify the code to call the example MemCopy function for each section that needs to be copied from flash to SARAM.

```
/********************************************************************
* DSP280x_examples\Flash source file
********************************************************************/

MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
```

4. **Modify the code to call the flash initialization routine:**

   This function will initialize the wait states for the flash and enable the Flash Pipeline mode.

```
/********************************************************************
* DSP280x peripheral example .c file
********************************************************************/

InitFlash();
```

5. **Set the required jumpers for "boot to Flash" mode.**

   The required jumper settings for each boot mode are shown in

Table 8. 280x Boot Mode Settings

GPIO18	GPIO29	GPIO34	Mode
1	1	1	Boot to flash 0x3F7FF6
1	1	0	Call SCI-A boot loader
1	0	1	Call SPI-A boot loader
1	0	0	Call I2C boot loader
0	1	1	Call eCAN-A boot loader
0	1	0	Boot to M0 SARAM 0x000000
0	0	1	Boot to OTP 0x3D7800
0	0	0	Call parallel boot loader

Refer to the documentation for your hardware platform for information on configuring the boot mode selection pins.

For more information on the '280x boot modes refer to the appropriate *Boot ROM Reference Guide*.

6. **Program the device with the built code.**

   In Code Composer Studio v4, when code is loaded into the device during debug, it automatically programs to flash memory.

   This can also be done using SDFlash available from Spectrum Digital's website (www.spectrumdigital.com). In addition the C2000 on-chip Flash programmer plug-in for Code Composer Studio v3.x.

   These tools will be updated to support new devices as they become available. Please check for updates.

7. **In Code Composer Studio v3, to debug, load the project in CCS, select *File->Load Symbols->Load Symbols Only*.**

   It is useful to load only symbol information when working in a debugging environment where the debugger cannot or need not load the object code, such as when the code is in ROM or flash. This operation loads the symbol information from the specified file.

5 Steps for Incorporating the Header Files and Sample Code

Follow these steps to incorporate the peripheral header files and sample code into your own projects. If you already have a project that uses the DSP281x header files then also refer to Section 7 for migration tips.

5.1 Before you begin

Before you include the header files and any sample code into your own project, it is recommended that you perform the following:

1. **Load and step through an example project.**

   Load and step through an example project to get familiar with the header files and sample code. This is described in Section 4.

2. **Create a copy of the source files you want to use.**

   DSP280x_headers: code required to incorporate the header files into your project
   DSP280x_common: shared source code much of which is used in the example projects.
   DSP280x_examples: example projects that use the header files and shared code.

5.2 Including the DSP280x Peripheral Header Files

Including the DSP280x header files in your project will allow you to use the bit-field structure approach in your code to access the peripherals on the DSP. To incorporate the header files in a new or existing project, perform the following steps:

1. **#include “DSP280x_Device.h” in your source files.**

   This include file will in-turn include all of the peripheral specific header files and required definitions to use the bit-field structure approach to access the peripherals.

```
/********************************************************************
* User’s source file
********************************************************************/

#include “DSP280x_Device.h”
```

2. **Edit DSP280x_Device.h and select the target you are building for:**

   In the below example, the file is configured to build for the ‘2808 device.

```
/********************************************************************
* DSP280x_headers\include\DSP280x_Device.h
********************************************************************/
#define   TARGET       1
#define   DSP28_28015  0
#define   DSP28_28016  0
#define   DSP28_2809   0
#define   DSP28_2808   TARGET
… etc
```

By default, the ‘2808 device is selected.

3. Add the source file *DSP280x_GlobalVariableDefs.c* to the project.

This file is found in the *DSP280x_headers\source\* directory and includes:

– Declarations for the variables that are used to access the peripheral registers.

– Data section #pragma assignments that are used by the linker to place the variables in the proper locations in memory.

4. Add the appropriate DSP280x header linker command file to the project.

As described in Section 3, when using the DSP280x header file approach, the data sections of the peripheral register structures are assigned to the memory locations of the peripheral registers by the linker.

To perform this memory allocation in your project, one of the following linker command files located in *DSP280x_headers\cmd\* must be included in your project:

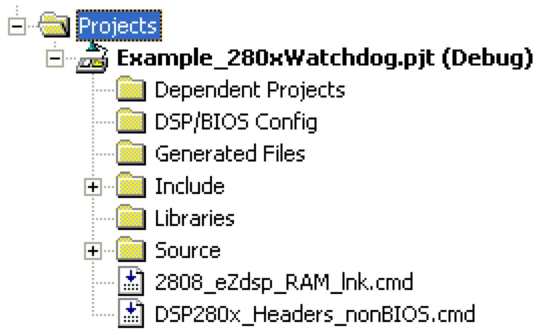
– For non-DSP/BIOS[†] projects:	*DSP280x_Headers_nonBIOS.cmd*
– For DSP/BIOS projects:	*DSP280x_Headers_BIOS.cmd*

The method for adding the header linker file to the project depends on the version of Code Composer Studio being used.

Code Composer Studio V2.2 and later:

As of CCS 2.2, more then one linker command file can be included in a project.

Add the appropriate header linker command file (BIOS or nonBIOS) directly to the project.



Code Composer Studio prior to V2.2

Prior to CCS 2.2, each project contained only one main linker command file. This file can, however, call additional .cmd files as needed. To include the required memory allocations for the DSP280x header files, perform the following two steps:

1) Update the project's main linker command (.cmd) file to call one of the supplied DSP280x peripheral structure linker command files using the -l option.

```
/********************************************************************
* User's linker .cmd file
********************************************************************/

/* Use this include file only for non-BIOS applications */
-l DSP280x_Headers_nonBIOS.cmd
/* Use this include file only for BIOS applications */
/* -l DSP280x_Headers_BIOS.cmd */
```

† DSP/BIOS is a trademark of Texas Instruments

2) Add the directory path to the DSP280x peripheral linker .cmd file to your project.

Code Composer Studio 3.x

a. Open the menu: *Project->Build Options*

b. Select the *Linker tab* and then Select *Basic.*

c. In the *Library Search Path*, add the directory path to the location of the *DSP280x_headers\cmd* directory on your system.

Code Composer Studio 4.x:

Method #1:

a. Right-click on the project in the project window of the C/C++ Projects perspective.

b. Select *Link Files to Project…*

c. Navigate to the *DSP280x_headers\cmd* directory on your system and select the desired .cmd file.

Note: The limitation with Method #1 is that the path to <install directory>\DSP280x_headers\cmd\<cmd file>.cmd is fixed on your PC. If you move the installation directory to another location on your PC, the project will "break" because it still expects the .cmd file to be in the original location. Use Method #2 if you are using "linked variables" in your project to ensure your project/installation directory is portable across computers and different locations on the same PC. (For more information, see: http://tiexpressdsp.com/index.php/Portable_Projects_in_CCSv4_for_C2000)

Method #2:

a. Right-click on the project in the project window of the C/C++ Projects perspective.

b. Select *New->File*.

c. Click on the *Advanced>>* button to expand the window.

d. Check the *Link to file in the file system* checkbox.

e. Select the *Variables…* button. From the list, pick the linked variable (macro defined in your *macros.ini* file) associated with your installation directory. (For the 280x header files, this is INSTALLROOT_280X_V<version #>). For more information on linked variables and the macros.ini file, see: http://tiexpressdsp.com/index.php/Portable_Projects_in_CCSv4_for_C2000#Method_.232_for_Linking_Files_to_Project:

f. Click on the *Extend…"* button. Navigate to the desired .cmd file and select *OK.*

5. Add the directory path to the DSP280x header files to your project.

Code Composer Studio 3.x:

To specify the directory where the header files are located:

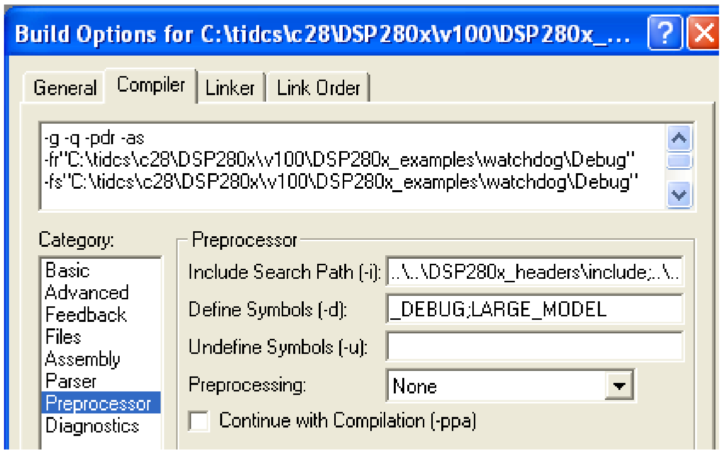
a. Open the menu:

   Project->Build Options

b. Select the *Compiler tab*

c. Select *pre-processor*.

d. In the *Include Search Path*, add the directory path to the location of *DSP280x_headers\include* on your system.



```
-g -q -pdr -as
-fr"C:\tidcs\c28\DSP280x\v100\DSP280x_examples\watchdog\Debug"
-fs"C:\tidcs\c28\DSP280x\v100\DSP280x_examples\watchdog\Debug"
```

Include Search Path (-i): ..\..\DSP280x_headers\include;..\..

Define Symbols (-d): _DEBUG;LARGE_MODEL

Undefine Symbols (-u):

Preprocessing: None

Continue with Compilation (-ppa)

Code Composer Studio 4.x:

To specify the directory where the header files are located:

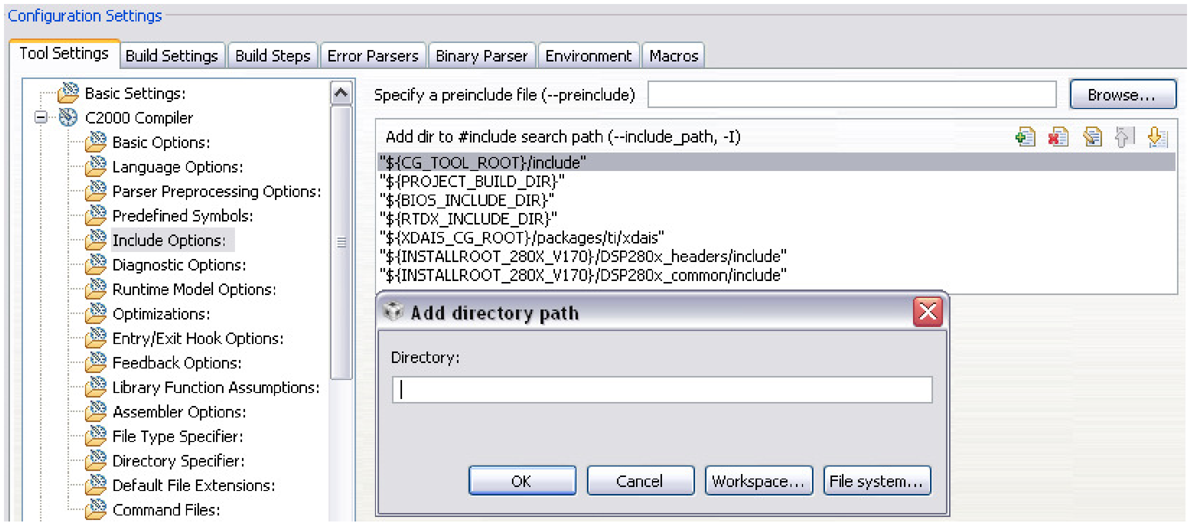
a. Open the menu: *Project->Properties*.

b. In the menu on the left, select “C/C++ Build”.

c. In the *“Tool Settings”* tab, Select *“C2000 Compiler -> Include Options:”*

d. In the “Add dir to #include search path (--include_path, -I” window, select the “Add” icon in the top right corner.

e. Select the “*File system…”* button and navigate to the directory path of *DSP280x_headers\include* on your system.



Specify a preinclude file (--preinclude)

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

```
"${CG_TOOL_ROOT}/include"
"${PROJECT_BUILD_DIR}"
"${BIOS_INCLUDE_DIR}"
"${RTDX_INCLUDE_DIR}"
"${XDAIS_CG_ROOT}/packages/ti/xdais"
"${INSTALLROOT_280X_V170}/DSP280x_headers/include"
"${INSTALLROOT_280X_V170}/DSP280x_common/include"
```

Directory:

6. **Additional suggested build options:**

   The following are additional compiler and linker options. The options can all be set via the *Project->Build Options* menu.

   – ***Compiler Tab:***

   - ❑ **-ml Select *Advanced* and check –ml**

     Build for large memory model. This setting allows data sections to reside anywhere within the 4M-memory reach of the 28x devices.

   - ❑ **-pdr Select *Diagnostics* and check –pdr**

     Issue non-serious warnings. The compiler uses a warning to indicate code that is valid but questionable. In many cases, these warnings issued by enabling -pdr can alert you to code that may cause problems later on.

   – ***Linker Tab:***

   - ❑ **-w Select *Advanced* and check –w**

     Warn about output sections. This option will alert you if any unassigned memory sections exist in your code. By default the linker will attempt to place any unassigned code or data section to an available memory location without alerting the user. This can cause problems, however, when the section is placed in an unexpected location.

   - ❑ **-e Select *Basic* and enter Code Entry Point –e**

     Defines a global symbol that specifies the primary entry point for the output module. For the DSP280x examples, this is the symbol “code_start”. This symbol is defined in the DSP280x_common\source\DSP280x_CodeStartBranch.asm file. When you load the code in Code Composer Studio, the debugger will set the PC to the address of this symbol. If you do not define a entry point using the –e option, then the linker will use _c_int00 by default.

5.3 Including Common Example Code

Including the common source code in your project will allow you to leverage code that is already written for the device. To incorporate the shared source code into a new or existing project, perform the following steps:

1. **#include “DSP280x_Examples.h” in your source files.**

   This include file will include common definitions and declarations used by the example code.

```
/********************************************************************
* User’s source file
********************************************************************/

#include “DSP280x_Examples.h”
```

2. **Add the directory path to the example include files to your project.**

Code Composer Studio 3.x

To specify the directory where the header files are located:

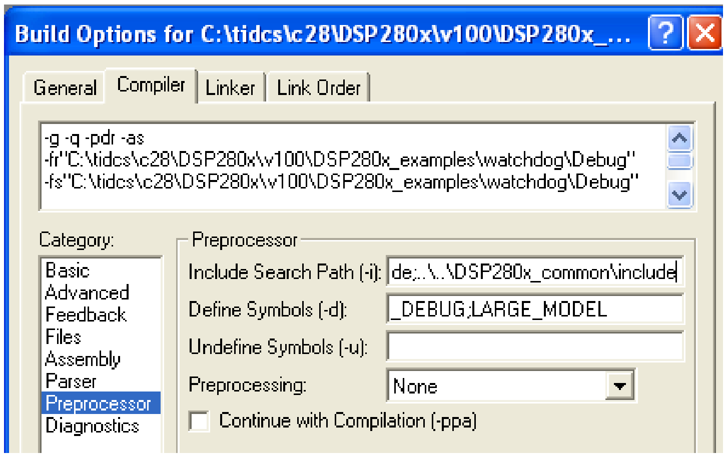
a. Open the menu:

Project->Build Options

b. Select the *Compiler tab*

c. Select *pre-processor*.

d. In the *Include Search Path*, add the directory path to the location of DSP280x_common/include on your system.
Use a semicolon between directories.



Build Options for C:\tidcs\c28\DSP280x\v100\DSP280x_...

```
-g -q -pdr -as
-fr"C:\tidcs\c28\DSP280x\v100\DSP280x_examples\watchdog\Debug"
-fs"C:\tidcs\c28\DSP280x\v100\DSP280x_examples\watchdog\Debug"
```

Include Search Path (-i): de;..\..\DSP280x_common\include
Define Symbols (-d): _DEBUG;LARGE_MODEL
Undefine Symbols (-u):
Preprocessing: None
Continue with Compilation (-ppa)

For example the directory path for the included projects is:
..\..\DSP280x_headers\include;..\..\DSP280x_common\include

Code Composer Studio 4.x:

To specify the directory where the header files are located:

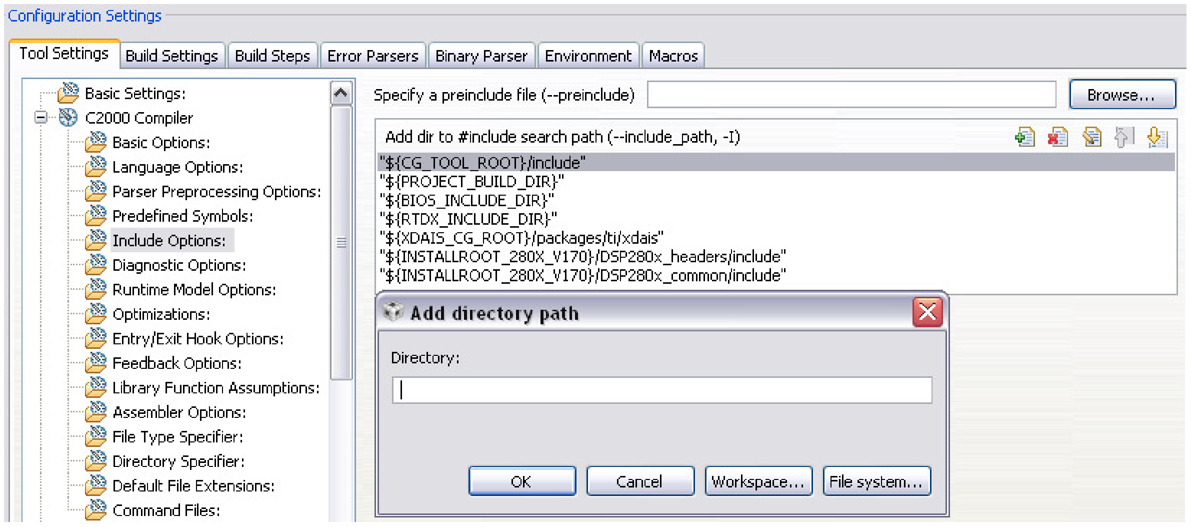
a. Open the menu: *Project->Properties*.

b. In the menu on the left, select “C/C++ Build”.

c. In the *“Tool Settings”* tab, Select *“C2000 Compiler -> Include Options:”*

d. In the “Add dir to #include search path (--include_path, -I” window, select the “Add” icon in the top right corner.

e. Select the “*File system…”* button and navigate to the directory path of *DSP280x_headers\include* on your system.



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

```
"${CG_TOOL_ROOT}/include"
"${PROJECT_BUILD_DIR}"
"${BIOS_INCLUDE_DIR}"
"${RTDX_INCLUDE_DIR}"
"${XDAIS_CG_ROOT}/packages/ti/xdais"
"${INSTALLROOT_280X_V170}/DSP280x_headers/include"
"${INSTALLROOT_280X_V170}/DSP280x_common/include"
```

3. Add a linker command file to your project.

The following memory linker .cmd files are provided as examples in the *DSP280x_common\cmd* directory. For getting started the basic *2808_eZdsp_RAM_lnk.cmd* file is suggested and used by most of the examples.

Table 9. Included Main Linker Command Files

Memory Linker Command File Examples	Location	Description
2808_eZdsp_RAM_lnk.cmd	DSP280x_common\cmd	eZdsp F2808 USB memory map that only allocates SARAM locations. No Flash, OTP, or CSM password protected locations (L0/L1) are used. This linker command file is used for most of the examples.
2809_RAM_lnk.cmd	DSP280x_common\cmd	2809 memory linker command file. Includes all of the internal SARAM blocks on a 2809 device. "RAM" linker files do not include flash or OTP blocks.
2808_RAM_lnk.cmd	DSP280x_common\cmd	2808 SARAM memory linker command file.
2806_RAM_lnk.cmd	DSP280x_common\cmd	2806 SARAM memory linker command file.
2802_RAM_lnk.cmd	DSP280x_common\cmd	2802 SARAM memory linker command file.
2801_RAM_lnk.cmd	DSP280x_common\cmd	2801 SARAM memory linker command file.
28015_RAM_lnk.cmd	DSP280x_common\cmd	28015 SARAM memory linker command file.
28016_RAM_lnk.cmd	DSP280x_common\cmd	28016 SARAM memory linker command file.
F2809.cmd	DSP280x_common\cmd	F2809 memory linker command file. Includes all Flash, OTP and CSM password protected memory locations.
F2808.cmd	DSP280x_common\cmd	F2808 memory linker command file
F2806.cmd	DSP280x_common\cmd	F2806 memory linker command file.
F2802.cmd	DSP280x_common\cmd	F2802 memory linker command file.
F2801.cmd	DSP280x_common\cmd	F2801 memory linker command file.
F28015.cmd	DSP280x_common\cmd	F28015 memory linker command file.
F28016.cmd	DSP280x_common\cmd	F28016 memory linker command file.

4. Set the CPU Frequency

In the *DSP280x_common\include\DSP280x_Examples.h* file specify the proper CPU frequency. Some examples are included in the file.

```
/*********************************************************************
* DSP280x_common\include\DSP280x_Examples.h
*********************************************************************/

#define CPU_RATE   10.000L   // for a 100MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   13.330L   // for a 75MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   20.000L   // for a 50MHz CPU clock speed  (SYSCLKOUT)
```

For 60 MHz operation, certain files (listed) must be modified for the examples to work properly. In the list below, files with * indicate that code modifications are required by the user in the DSP280x_Examples.h file to operate at 60 MHz. If the user wants to run these example files at frequencies other than 100MHz or 60MHz, the user must modify these files directly with the appropriate timing values.

Common Files:

- ECan.c* in the DSP280x_common\source\ directory
- usDelay.asm in the DSP280x_common\source\ directory
- Examples.h* in the DSP280x_common\include\ directory

Example Files:

- All HRPWM example source files
- Example_280xI2C_eeprom.c*
- All ADC examples
- Example_280xECap_apwm.c*
- Example_280xSci_Echoback.c8
- Example_280xEqep_pos_speed.c* and related files (Example_posspeed.c*, Example_posspeed.h*, Example_EPwmSetup.c*, Example_posspeed.xls)
- Example_280xEqep_freqcal.c* and related files (Example_freqcal.c*, Example_freqcal.h*, Example_EPwmSetup.c*, Example_freqcal.xls)

5. **Add desired common source files to the project.**

   The common source files are found in the *DSP280x_common\source\* directory.

6. **Include .c files for the PIE.**

   Since all catalog ‘280x applications make use of the PIE interrupt block, you will want to include the PIE support .c files to help with initializing the PIE. The shell ISR functions can be used directly or you can re-map your own function into the PIE vector table provided. A list of these files can be found in section 8.2.1.

6 Troubleshooting Tips & Frequently Asked Questions

- **In the examples, what do "EALLOW;" and "EDIS;" do?**

  EALLOW; is a macro defined in DSP280x_Device.h for the assembly instruction EALLOW and likewise EDIS is a macro for the EDIS instruction. That is EALLOW; is the same as embedding the assembly instruction asm(" EALLOW");

  Several control registers on the 28x devices are protected from spurious CPU writes by the EALLOW protection mechanism. The EALLOW bit in status register 1 indicates if the protection is enabled or disabled. While protected, all CPU writes to the register are ignored and only CPU reads, JTAG reads and JTAG writes are allowed. If this bit has been set by execution of the EALLOW instruction, then the CPU is allowed to freely write to the protected registers. After modifying the registers, they can once again be protected by executing the EDIS assembly instruction to clear the EALLOW bit.

  For a complete list of protected registers, refer to *TMS320x280x Control and Interrupts Reference Guide* (SPRU712).

- **Peripheral registers read back 0x0000 and/or cannot be written to.**

  There are a few things to check:

  - Peripheral registers cannot be modified or unless the clock to the specific peripheral is enabled. The function InitPeripheralClocks() in the DSP280x_common\source directory shows an example of enabling the peripheral clocks.
  - Some peripherals are not present on all 280x family derivatives. Refer to the device datasheet for information on which peripherals are available.
  - The EALLOW bit protects some registers from spurious writes by the CPU. If your program seems unable to write to a register, then check to see if it is EALLOW protected. If it is, then enable access using the EALLOW assembly instruction. *TMS320x280x Control and Interrupts Reference Guide* (SPRU712) for a complete list of EALLOW protected registers.

- **Memory block L0, L1 read back all 0x0000.**

  In this case most likely the code security module is locked and thus the protected memory locations are reading back all 0x0000. Refer to the for information on the code security module.

- **Code cannot write to L0 or L1 memory blocks.**

  In this case most likely the code security module is locked and thus the protected memory locations are reading back all 0x0000. Code that is executing from outside of the protected cannot read or write to protected memory while the CSM is locked. Refer to the *TMS320x280x Control and Interrupts Reference Guide* (SPRU712) for information on the code security module

- **A peripheral register reads back ok, but cannot be written to.**

  The EALLOW bit protects some registers from spurious writes by the CPU. If your program seems unable to write to a register, then check to see if it is EALLOW protected. If it is, then enable access using the EALLOW assembly instruction. *TMS320x280x Control and Interrupts Reference Guide* (SPRU712) for a complete list of EALLOW protected registers.

- **I re-built one of the projects to run from Flash and now it doesn't work. What could be wrong?**

  Make sure all initialized sections have been moved to flash such as .econst and .switch.

  If you are using SDFlash, make sure that all initialized sections, including .econst, are allocated to page 0 in the linker command file (.cmd). SDFlash will only program sections in the .out file that are allocated to page 0.

- **Why do the examples populate the PIE vector table and then re-assign some of the function pointers to other ISRs?**

  The examples share a common default ISR file. This file is used to populate the PIE vector table with pointers to default interrupt service routines. Any ISR used within the example is then remapped to a function within the same source file. This is done for the following reasons:

  – The entire PIE vector table is enabled, even if the ISR is not used within the example. This can be very useful for debug purposes.

  – The default ISR file is left un-modified for use with other examples or your own project as you see fit.

  – It illustrates how the PIE table can be updated at a later time.

- **When I build the examples, the linker outputs the following: warning: entry point other than _c_int00 specified. What does this mean?**

  This warning is given when a symbol other then _c_int00 is defined as the code entry point of the project. For these examples, the symbol code_start is the first code that is executed after exiting the boot ROM code and thus is defined as the entry point via the –e linker option. This symbol is defined in the DSP280x_CodeStartBranch.asm file. The entry point symbol is used by the debugger and by the hex utility. When you load the code, CCS will set the PC to the entry point symbol. By default, this is the _c_int00 symbol which marks the start of the C initialization routine. For the DSP280x examples, the code_start symbol is used instead. Refer to the source code for more information.

- **When I build many of the examples, the compiler outputs the following: remark: controlling expression is constant. What does this mean?**

  Some of the examples run forever until the user stops execution by using a while(1) {} loop The remark refers to the while loop using a constant and thus the loop will never be exited.

- **When I build some of the examples, the compiler outputs the following: warning: statement is unreachable. What does this mean?**

  Some of the examples run forever until the user stops execution by using a while(1) {} loop. If there is code after this while(1) loop then it will never be reached.

- **I changed the build configuration of one of the projects from “Debug” to “Release” and now the project will not build. What could be wrong?**

  When you switch to a new build configuration (*Project->Configurations*) the compiler and linker options changed for the project. The user must enter other options such as include search path and the library search path. Open the build options menu *(Project->Build Options)* and enter the following information:

  – Compiler Tab, Preprocessor: Include search path

  – Linker Tab, Basic: Library search path

  – Linker Tab, Basic: Include libraries (ie rts2800_ml.lib)

  Refer to section 4.5 for more details.

- **In the flash example I loaded the symbols and ran to main. I then set a breakpoint but the breakpoint is never hit. What could be wrong?**

  In the Flash example, the InitFlash function and several of the ISR functions are copied out of flash into SARAM. When you set a breakpoint in one of these functions, Code Composer will insert an ESTOP0 instruction into the SARAM location. When the ESTOP0 instruction is hit, program execution is halted. CCS will then remove the ESTOP0 and replace it with the original opcode. In the case of the flash program, when one of these functions is copied from Flash into SARAM, the ESTOP0 instruction is overwritten code. This is why the breakpoint is never hit. To avoid this, set the breakpoint after the SARAM functions have been copied to SARAM.

- **The eCAN control registers require 32-bit write accesses.**

  The compiler will instead make a 16-bit write accesses if it can in order to improve codesize and/or performance. This can result in unpredictable results.

  One method to avoid this is to create a duplicate copy of the eCAN control registers in RAM. Use this copy as a shadow register. First copy the contents of the eCAN register you want to modify into the shadow register. Make the changes to the shadow register and then write the data back as a 32-bit value. This method is shown in the DSP280x_examples\ ecan_back2back example project.

6.1 Effects of read-modify-write instructions.

When writing any code, whether it be C or assembly, keep in mind the effects of read-modify-write instructions.

The ‘28x DSP will write to registers or memory locations 16 or 32-bits at a time. Any instruction that seems to write to a single bit is actually reading the register, modifying the single bit, and then writing back the results. This is referred to as a read-modify-write instruction. For most registers this operation does not pose a problem. A notable exception is:

6.1.1 Registers with multiple flag bits in which writing a 1 clears that flag.

For example, consider the PIEACK register. Bits within this register are cleared when writing a 1 to that bit. If more then one bit is set, performing a read-modify-write on the register may clear more bits then intended.

The below solution is incorrect. It will write a 1 to any bit set and thus clear all of them:

```
/*********************************************************************
* User’s source file
*********************************************************************/

     PieCtrl.PIEACK.bit.Ack1 = 1;    // INCORRECT! May clear more bits.
```

The correct solution is to write a mask value to the register in which only the intended bit will have a 1 written to it:

```
/*********************************************************************
* User’s source file
*********************************************************************/

     #define PIEACK_GROUP1  0x0001
             ……
      PieCtrl.PIEACK.all = PIEACK_GROUP1;   // CORRECT!
```

6.1.2 Registers with Volatile Bits.

Some registers have volatile bits that can be set by external hardware.

Consider the PIEIFRx registers. An atomic read-modify-write instruction will read the 16-bit register, modify the value and then write it back. During the modify portion of the operation a bit in the PIEIFRx register could change due to an external hardware event and thus the value may get corrupted during the write.

The rule for registers of this nature is to never modify them during runtime. Let the CPU take the interrupt and clear the IFR flag.

7 Migration Tips for moving from the TMS320x281x header files to the TMS320x280x header files

This section includes suggestions for moving a project from the 281x header files to the 280x header files.

1. **Create a copy of your project to work with or back-up your current project.**

2. **Open the project file(s) in a text editor**

   In Code Composer Studio v3.x:

   Open the .pjt project file for your project. Replace all instances of 281x with 280x so that the appropriate source files and build options are used. Check the path names to make sure they point to the appropriate header file and source code directories.

   In Code Composer Studio v4.x:

   Open the *.project*, *.cdtbuild*, and *macros.ini* files in your example folder. Replace all instances of 281x with 280x so that the appropriate source files and build options are used. Check the path names to make sure they point to the appropriate header file and source code directories. Also replace the header file version number for the paths and macro names as well where appropriate. For instance, if a macro name was INSTALLROOT_**281**X_V**120** for your 281x project using 281x header files V1.20, change this to INSTALLROOT_**280**X_V**170** to migrate to the 280x header files V1.70. If not using the default macro name for your header file version, be sure to change your macros according to your chosen macro name in the *.project, .cdtbuild*, and *macros.ini* files.

3. **Load the project into Code Composer Studio**

   Use the edit-> find in files dialog to find instances of DSP281x_Device.h and DSP281x_Example.h. Replace these with DSP280x_Device.h and DSP280x_Example.h respectively.

4. **Make sure you are using the correct linker command files (.cmd) appropriate for your device and for the DSP280x header files.**

   You will have one file for the memory definitions and one file for the header file structure definitions. Using a 281x memory file can cause issues since the H0 memory block has moved to a new location on the 280x devices.

5. **Build the project.**

   The compiler will highlight areas that have changed. Most of these changes will fall into one of the following categories:

   - Bit-name or register name corrections to align with the peripheral user guides. See Table 9 for a listing of these changes.
   - Code that was written for the 281x event manager (EV) will need to be re-written for the 280x ePWM, eCAP and eQEP peripherals.
   - Code for the 281x McBSP and XINTF will need to be removed as these peripherals are not available on the 280x devices.

Table 10. Summary of Register and Bit-Name Changes from DSP281x V1.00 to DSP280x V1.20

Peripheral	Register	Bit Name Old	Bit Name New	Comment
AdcRegs				
	ADCTRL2	EVB_SOC_ SEQ2	EPWM_SOCB_ SEQ2	SOC is now performed by ePWM
		EVA_SOC_ SEQ1	EPWM_SOCA_ SEQ1	SOC is now performed by ePWM
		EVB_SOC_ SEQ	EPWM_SOCB_ SEQ	SOC is now performed by ePWM
DevEmuRegs				
	DEVICEID		PARTID REVID	Split into two registers, PARTID and REVID
EcanaRegs				
	CANMDL	BYTE1	BYTE3	Order of bytes was incorrect
		BYTE3	BYTE1	
		BYTE4	BYTE0	
	CANMDH	BYTE5	BYTE7	Order of bytes was incorrect
		BYTE7	BYTE5	
		BYTE8	BYTE4	
GpioMuxRegs				
				The GPIO peripheral has been redesigned from the 281x. All of the registers have moved from 16-bit to 32-bits. The GpioMuxRegs are now the GpioCtrlRegs and the bit definitions have all changed. Please refer to *TMS320x280x Control and Interrupts Reference Guide* (SPRU712) for more information on the GPIO peripheral.
PieCtrlRegs				
	PIECTRL	PIECRTL	PIECTRL	Typo
SciaRegs, ScibRegs				
	SCIFFTX	TXFFILIL	TXFFIL	Typo
		TXINTCLR	TXFFINTCLR	Alignment with user’s guide.
	SCIFFRX	RXFIFST	RXFFST	Typo – Also corrected in user’s guide

8 Packet Contents:

This section lists all of the files included in the release.

8.1 Header File Support – DSP280x_headers

The DSP280x header files are located in the *<base>\DSP280x_headers\* directory.

8.1.1 DSP280x Header Files – Main Files

The following files must be added to any project that uses the DSP280x header files. Refer to section 5.2 for information on incorporating the header files into a new or existing project.

Table 11. DSP280x Header Files – Main Files

File	Location	Description
DSP280x_Device.h	DSP280x_headers\include	Main include file. Include this one file in any of your .c source files. This file in-turn includes all of the peripheral specific .h files listed below. In addition the file includes typedef statements and commonly used mask values. Refer to section 5.2.
DSP280x_GlobalVariableDefs.c	DSP280x_headers\source	Defines the variables that are used to access the peripheral structures and data section #pragma assignment statements. This file must be included in any project that uses the header files. Refer to section 5.2.
DSP280x_Headers_BIOS.cmd	DSP280x_headers\cmd	Linker .cmd file to assign the header file variables in a BIOS project. This file must be included in any BIOS project that uses the header files. Refer to section 5.2.
DSP280x_Headers_nonBIOS.cmd	DSP280x_headers\cmd	Linker .cmd file to assign the header file variables in a non-BIOS project. This file must be included in any non-BIOS project that uses the header files. Refer to section 5.2.

8.1.2 DSP280x Header Files – Peripheral Bit-Field and Register Structure Definition Files

The following files define the bit-fields and register structures for each of the peripherals on the 280x devices. These files are automatically included in the project by including *DSP280x_Device.h.* Refer to section 4.2 for more information on incorporating the header files into a new or existing project.

Table 12. DSP280x Header File Bit-Field & Register Structure Definition Files

File	Location	Description
DSP280x_Adc.h	DSP280x_headers\include	ADC register structure and bit-field definitions.
DSP280x_CpuTimers.h	DSP280x_headers\include	CPU-Timer register structure and bit-field definitions.
DSP280x_DevEmu.h	DSP280x_headers\include	Emulation register definitions
DSP280x_ECan.h	DSP280x_headers\include	eCAN register structures and bit-field definitions.
DSP280x_ECap.h	DSP280x_headers\include	eCAP register structures and bit-field definitions.
DSP280x_EPwm.h	DSP280x_headers\include	ePWM register structures and bit-field definitions.
DSP280x_EQep.h	DSP280x_headers\include	eQEP register structures and bit-field definitions.
DSP280x_Gpio.h	DSP280x_headers\include	General Purpose I/O (GPIO) register structures and bit-field definitions.
DSP280x_I2c.h	DSP280x_headers\include	I2C register structure and bit-field definitions.
DSP280x_PieCtrl.h	DSP280x_headers\include	PIE control register structure and bit-field definitions.
DSP280x_PieVect.h	DSP280x_headers\include	Structure definition for the entire PIE vector table.
DSP280x_Sci.h	DSP280x_headers\include	SCI register structure and bit-field definitions.
DSP280x_Spi.h	DSP280x_headers\include	SPI register structure and bit-field definitions.
DSP280x_SysCtrl.h	DSP280x_headers\include	System register definitions. Includes Watchdog, PLL, CSM, Flash/OTP, Clock registers.
DSP280x_XIntrupt.h	DSP280x_headers\include	External interrupt register structure and bit-field definitions.

8.1.3 Code Composer .gel Files

The following Code Composer Studio .gel files are included for use with the DSP280x Header File peripheral register structures.

Table 13. DSP280x Included GEL Files

File	Location	Description
DSP280x_Peripheral.gel	DSP280x_headers\gel	This is relevant for CCSv3.x only. Provides GEL pull-down menus to load the DSP280x data structures into the watch window. You may want to have CCS load this file automatically by adding a GEL_LoadGel("<base>DSP280x_headers\/gel\DSP280x_peripheral.gel") function to the standard F2808.gel that was included with CCS.

8.1.4 Variable Names and Data Sections

This section is a summary of the variable names and data sections allocated by the DSP280x_headers\source\DSP280x_GlobalVariableDefs.c file. Note that all peripherals may not be available on a particular 280x device. Refer to the device datasheet for the peripheral mix available on each 280x family derivative.

Table 14. DSP280x Variable Names and Data Sections

Peripheral	Starting Address	Structure Variable Name
ADC	0x007100	AdcRegs
ADC Mirrored Result Registers	0x000B00	AdcMirror
Code Security Module	0x000AE0	CsmRegs
Code Security Module Password Locations	0x3F7FF8- 0x3F7FFF	CsmPwl
CPU Timer 0	0x000C00	CpuTimer0Regs
Device and Emulation Registers	0x000880	DevEmuRegs
eCAN-A	0x006000	ECanaRegs
eCAN-A Mail Boxes	0x006100	ECanaMboxes
eCAN-A Local Acceptance Masks	0x006040	ECanaLAMRegs
eCAN-A Message Object Time Stamps	0x006080	ECanaMOTSRegs
eCAN-A Message Object Time-Out	0x0060C0	ECanaMOTORegs
eCAN-B	0x006200	ECanbRegs
eCAN-B Mail Boxes	0x006300	ECanbMboxes
eCAN-B Local Acceptance Masks	0x006240	ECanbLAMRegs
eCAN-B Message Object Time Stamps	0x006280	ECanbMOTSRegs
eCAN-B Message Object Time-Out	0x0062C0	ECanbMOTORegs
ePWM1	0x006800	EPwm1Regs
ePWM2	0x006840	EPwm2Regs
ePWM3	0x006880	EPwm3Regs
ePWM4	0x0068C0	EPwm4Regs
ePWM5	0x006900	EPwm5Regs
ePWM6	0x006940	EPwm6Regs
eCAP1	0x006A00	ECap1Regs
eCAP2	0x006A20	ECap2Regs
eCAP3	0x006A40	ECap3Regs
eCAP4	0x006A60	ECap4Regs
eQEP1	0x006B00	EQep1Regs
eQEP2	0x006B40	EQep2Regs
External Interrupt Registers	0x007070,	XIntruptRegs
Flash & OTP Configuration Registers	0x000A80	FlashRegs
General Purpose I/O Data Registers	0x006fC0	GpioDataRegs
General Purpose Control Registers	0x006F80	GpioCtrlRegs
General Purpose Interrupt Registers	0x006fE0	GpioIntRegs
I2C	0x007900	I2caRegs

Peripheral	Starting Address	Structure Variable Name
PIE Control	0x000CE0	PieCtrlRegs
SCI-A	0x007050	SciaRegs
SCI-B	0x007750	ScibRegs
SPI-A	0x007040	SpiaRegs
SPI-B	0x007740	SpibRegs
SPI-C	0x007760	SpicRegs
SPI-D	0x007780	SpidRegs

8.2 Common Example Code – DSP280x_common

8.2.1 Peripheral Interrupt Expansion (PIE) Block Support

In addition to the register definitions defined in DSP280x_PieCtrl.h, this packet provides the basic ISR structure for the PIE block. These files are:

Table 15. Basic PIE Block Specific Support Files

File	Location	Description
DSP280x_DefaultIsr.c	DSP280x_common\source	Shell interrupt service routines (ISRs) for the entire PIE vector table. You can choose to populate one of functions or re-map your own ISR to the PIE vector table. **Note: This file is not used for DSP/BIOS projects.**
DSP280x_DefaultIsr.h	DSP280x_common\include	Function prototype statements for the ISRs in DSP280x_DefaultIsr.c. **Note: This file is not used for DSP/BIOS projects.**
DSP280x_PieVect.c	DSP280x_common\source	Creates an instance of the PIE vector table structure initialized with pointers to the ISR functions in DSP280x_DefaultIsr.c. This instance can be copied to the PIE vector table in order to initialize it with the default ISR locations.

In addition, the following files are included for software prioritization of interrupts. These files are used in place of those above when additional software prioritization of the interrupts is required. Refer to the example in *DSP280x_examples\sw_prioritized_interrupts* and the *Example_280xISRPriorities.doc* documentation in *<base>\doc* for more information.

Table 16. Software Prioritized Interrupt PIE Block Specific Support Files

File	Location	Description
DSP280x_SWPrioritizedDefaultIsr.c	DSP280x_common\source	Default shell interrupt service routines (ISRs). These are shell ISRs for all of the PIE interrupts. You can choose to populate one of functions or re-map your own interrupt service routine to the PIE vector table. **Note: This file is not used for DSP/BIOS projects.**
DSP280x_SWPrioritizedIsrLevels.h	DSP280x_common\include	Function prototype statements for the ISRs in DSP280x_DefaultIsr.c. **Note: This file is not used for DSP/BIOS projects.**
DSP280x_SWPrioritizedPieVect.c	DSP280x_common\source	Creates an instance of the PIE vector table structure initialized with pointers to the default ISR functions that are included in DSP280x_DefaultIsr.c. This instance can be copied to the PIE vector table in order to initialize it with the default ISR locations.

8.2.2 Peripheral Specific Files

Several peripheral specific initialization routines and support functions are included in the peripheral .c source files in the *DSP280x_common\src\* directory. These files include:

Table 17. Included Peripheral Specific Files

File	Description
DSP280x_GlobalPrototypes.h	Function prototypes for the peripheral specific functions included in these files.
DSP280x_Adc.c	ADC specific functions and macros.
DSP280x_CpuTimers.c	CPU-Timer specific functions and macros.
DSP280x_ECan.c	Enhanced CAN specific functions and macros.
DSP280x_ECap.c	eCAP module specific functions and macros.
DSP280x_EPwm.c	ePWM module specific functions and macros.
DSP280x_EPwm_defines.h	#define macros that are used for the ePWM examples
DSP280x_EQep.c	eQEP module specific functions and macros.
DSP280x_Gpio.c	General-purpose IO (GPIO) specific functions and macros.
DSP280x_I2C.c	I2C specific functions and macros.
DSP280x_I2c_defines.h	#define macros that are used for the I2C examples
DSP280x_PieCtrl.c	PIE control specific functions and macros.
DSP280x_Sci.c	SCI specific functions and macros.
DSP280x_Spi.c	SPI specific functions and macros.
DSP280x_SysCtrl.c	System control (watchdog, clock, PLL etc) specific functions and macros.

Note: The specific routines are under development and may not all be available as of this release. They will be added and distributed as more examples are developed.

8.2.3 Utility Function Source Files

Table 18. Included Utility Function Source Files

File	Description
DSP280x_CodeStartBranch.asm	Branch to the start of code execution. This is used to re-direct code execution when booting to Flash, OTP or M0 SARAM memory. An option to disable the watchdog before the C init routine is included.
DSP280x_DBGIER.asm	Assembly function to manipulate the DEBIER register from C.
DSP280x_DisInt.asm	Disable interrupt and restore interrupt functions. These functions allow you to disable INTM and DBGM and then later restore their state.
DSP280x_usDelay.asm	Assembly function to insert a delay time in microseconds. This function is cycle dependant and must be executed from zero wait-stated RAM to be accurate. Refer to *DSP280x_examples\adc* for an example of its use.
DSP280x_CSMPasswords.asm	Include in a project to program the code security module passwords and reserved locations.

8.2.4 Example Linker .cmd files

Example memory linker command files are located in the *DSP280x_common\cmd* directory. For getting started the basic 2808_eZdsp_RAM_lnk.cmd file is suggested and used by many of the included examples.

The SARAM blocks L1, L2 and H0 are mirrored on these devices. For simplicity these memory maps only include one instance of these memory blocks.

Table 19. Included Main Linker Command Files

Memory Linker Command File Examples	Location	Description
2808_eZdsp_RAM_lnk.cmd	DSP280x_common\cmd	eZdsp F2808 USB memory map that only allocates SARAM locations. No Flash, OTP, or CSM password protected locations (L0/L1) are used. This linker command file is used for most of the examples.
2809_RAM_lnk.cmd	DSP280x_common\cmd	2809 memory linker command file. Includes all of the internal SARAM blocks on a 2809 device. "RAM" linker files do not include flash or OTP
2808_RAM_lnk.cmd	DSP280x_common\cmd	2808 SARAM memory linker command file.
2806_RAM_lnk.cmd	DSP280x_common\cmd	2806 SARAM memory linker command file.
2802_RAM_lnk.cmd	DSP280x_common\cmd	2802 SARAM memory linker command file.
2801_RAM_lnk.cmd	DSP280x_common\cmd	2801 SARAM memory linker command file.
28015_RAM_lnk.cmd	DSP280x_common\cmd	28015 SARAM memory linker command file.
28016_RAM_lnk.cmd	DSP280x_common\cmd	28016 SARAM memory linker command file.
F2809.cmd	DSP280x_common\cmd	F2809 memory linker command file. Includes all Flash, OTP and CSM password protected memory locations.
F2808.cmd	DSP280x_common\cmd	F2808 memory linker command file
F2806.cmd	DSP280x_common\cmd	F2806 memory linker command file.
F2802.cmd	DSP280x_common\cmd	F2802 memory linker command file.
F2801.cmd	DSP280x_common\cmd	F2801 memory linker command file.
F28015.cmd	DSP280x_common\cmd	F28015 memory linker command file.
F28016.cmd	DSP280x_common\cmd	F28016 memory linker command file.

8.2.5 Example Library .lib Files

Example library files are located in the *DSP280x_common\lib* directory. For this release the IQMath library is included for use in the example projects. Please refer to the *C28x IQMath Library - A Virtual Floating Point Engine (*SPRC087) for more information on IQMath and the most recent IQMath library. The SFO libraries are also included for use in the example projects. Please refer to *TMS320x28xx, 28xxx HRPWM Reference Guide* (SPRU924) for more information on SFO library usage and the HRPWM module.

Table 20. Included Library Files

Main Liner Command File Examples	**Description**
IQmath.lib	Please refer to the *C28x IQMath Library - A Virtual Floating Point Engine (*SPRC087) for more information on IQMath.
IQmathLib.h	IQMath header file.
SFO_TI_Build.lib	Please refer to the *TMS320x28xx, 28xxx HRPWM Reference Guide* (SPRU924) for more information on the SFO library
SFO.h	SFO header file
SFO_TI_Build_V5.lib/ SFO_TI_Build_V5B.lib	Please refer to the *TMS320x28xx,28xxx HRPWM Reference Guide* (SPRU924) for more information on the SFO V5 library. Updated versions will be marked with alphabetical characters after “V5” (i.e. SFO_TI_Build_V5B.lib)
SFO_V5.h	SFO V5 header file

9 Detailed Revision History:

Changes from V1.60 to V1.70

Changes to Header Files:

a) **DSP280x_CpuTimers.h –** Uncommented CpuTimer1 and CpuTimer2 code.

b) **DSP280x_Device.h-** Added int64 and Uint64 typedefs.

c) **DSP280x_Spi.h-** Changed SPIPRI register bit 6 to reserved bit.

d) **DSP280x_Gpio.h-** In GPIO_DATA_REGS struct changed GPBPUD_REG for GPBDAT register to GPBDAT_REG.

Changes to Common Files:

e) **DSP280x_CpuTimers.c –** Updated comments to indicate only CpuTimer2 is reserved for DSP/BIOS use. User must comment out CpuTimer2 code when using DSP/BIOS.

f) **DSP280x_I2c_defines.h-** Fixed typo for DSP280x_I2CDEINFES_H and replaced with DSP280x_I2CDEFINES_H.

g) **DSP280x_ECan.c-** Bit configuration parameter for 60 MHz changed to "BT of 15". Also added disclaimer to file.

h) **CCSv4 gel files –** Added ccsv4 directory in /gel directory for CCSv4-specific device gel files (GEL_WatchAdd() functions removed).

Changes to Example Files:

i) **All PJT Files-** Removed the line: Tool="DspBiosBuilder" from all example PJT files for easy migration path to CCSv4 Microcontroller-only (code-size limited) version users.

j) **Example_280xHRPWM.c and Example_280xHRPWM_slider.c–** Changed initialization code to set TBPRD register to period-1 instead of period to achieve correct period and duty cycle.

k) **Example_280xLPMHaltWake.c** – Updated description comments for wakeup.

l) **Example_280xHRPWM_SFO_V5.c-** Added line of code before calling MepDis() function to enable HRPWM logic for the channel first.

m) **Added DSP280x_examples_ccsv4 directories** - Added directories for CCSv4.x projects. The example projects in these directories are identical to those found in the normal CCSv3.x DSP280x_examples directory with the exception that the examples now support the Code Composer Studio v4.x project folder format instead of the Code Composer Studio v3.x PJT files. The example gel files have also been removed for the CCSv4 example projects because the gel file functions used in the example gels are no longer supported.

Changes from V1.50 to V1.60

Changes to Header Files:

n) **DSP280x_DevEmu.h –** Removed MONPRIV, EMU0SEL, and EMU1SEL bits in the DEVICECNF register.

Changes to Common Files:

o) **DSP280x_SWPrioritizedDefaultIsr.c –** Fixed some PIEIER number typos.

p) **SFO_TI_Build_V5B.lib** and **SFO_TI_Build_V5Bfpu.lib –** Because the SFO_MepEn() function in the original version of the SFO library was restricted to MEP control on falling edge only with HRLOAD on CTR=ZRO, a new version of the V5 library, V5B, was added, which includes a SFO_MepEn() function that supports *all* available HRPWM configurations – falling and rising edge as well as HRLOAD on CTR=ZRO and CTR=PRD.

q) **DSP280x_SysCtrl.h –** Added EALLOW access to code which sets CLKINDIV bit to 0. Also removed "!" from code which sets CLKINDIV to divider value.

Changes to Example Files:

r) **Example_280xECanBack2Back.c–** Removed initialization code and replaced with InitECana() function from DSP2833x_ECan.c file.

<u>Changes from V1.41 to V1.50</u>

Changes to Header Files:

a) Updated device .gel files with correct syntax for configuring addressing modes.

b) In .gel files, corrected previously incorrect QEP address locations.

c) Deleted all UDC9501 references – deleted UDC9501.gel and sim9501.gel. Users can use '2801 files as replacement.

d) **DSP280x_SysCtrl.h** – Corrected FSTDBYWAIT and FACTIVEWAIT register bit field lengths so that STDBYWAIT and ACTIVEWAIT are 9 bits long and rsvd fields are 7 bits long instead of 8 bits long each.

e) **DSP280x_Headers_BIOS.cmd and DSP280x_Headers_nonBIOS.cmd** – Peripheral Frame 1 and Peripheral Frame 2 comment headings are now above the appropriate peripheral regfiles.

Changes to examples:

a) The root of the default path in all example project files was changed from C:\tidcs\c28\DSP280x\v141\ to C:\tidcs\c28\DSP280x\v150\ to reflect the version change.

b) **Example_280xEcanA_to_B_Xmit.c** – Changed CANTA register read to a shadow read as shown below:

```
do
{
        ECanaShadow.CANTA.all = ECanaRegs.CANTA.all;
        } while(ECanaShadow.CANTA.bit.TA25 == 0 );
```

c) **Example_280xSpi_FFDLB_int.c** – Changed comment regarding RXFIFO level set at 31 levels to 8 levels.

d) **Example_280xSci_FFDLB_int.c** – fixed baud rate calculation defines at the top of file by adding: `#define LSPCLK_FREQ  CPU_FREQ/4` and fixing the SCI baud rate period formula such that: `#define SCI_PRD  LSPCLK_FREQ/((SCI_FREQ*8)-1)`.

e) **DSP280x_ECan.c**
Added 60 MHz code (inside #define (CPU_FRQ_60MHZ) … #endif directives) for 60 MHz devices. Also fixed incorrect comments such that GPIO30 refers to CANRXA and GPIO31 to CANTXA, and not vice versa. Also, made accesses to CANTIOC and CANRIOC registers shadow accesses for both CAN-A and CAN-B.

f) **DSP280x_Examples.h**– Added the following lines so example and common code can be run at 60 MHz if configured for 60 MHz:

```
#define CPU_FRQ_100MHZ   1     // 100 Mhz CPU Freq
#define CPU_FRQ_60MHZ    0     // 60 MHz CPU Freq
```

g) Updated the following files for 60 MHz devices – either by adding 60 MHz comments or by adding #if(CPU_FRQ_60MHZ)..#endif directives around code concerning 60 MHz operation.

Files with * indicate that code modifications are required by the user in the DSP280x_Examples.h file to operate at 60 MHz. If the user wants to run these example files at frequencies other than 100MHz or 60MHz, the user must modify these files directly with the appropriate timing values.

- ECan.c*
- usDelay.asm
- Examples.h*
- All HRPWM example source files
- Example_280xI2C_eeprom.c*
- All ADC examples
- Example_280xECap_apwm.c*
- Example_280xSci_Echoback.c*
- Example_280xEqep_pos_speed.c* and related files (Example_posspeed.c*, Example_posspeed.h*, Example_EPwmSetup.c*, Example_posspeed.xls)

- Example_280xEqep_freqcal.c* and related files (Example_freqcal.c*, Example_freqcal.h*, Example_EPwmSetup.c*, Example_freqcal.xls)

h) **Example_280xSci_Autobaud.c** – Added comments to clarify autobaud detection and locking flow in example program. Also changed code in SCI-A Receive ISR to check for ABD bit set instead of CDC bit set to indicate that autobaud detection occurred.

i) All HRPWM examples–

   a. .pjt files - Fixed pathnames for Debug and Release folders so that they are generated within the hrpwm example folders instead of the global DSP280x_examples folder.

   b. All examples – added "DSP280x_Examples.h" include file, and deleted declarations to unnecessary function prototypes (i.e. SysCtrl.h, PieVectorTable.h, etc.)

j) Added 3 low power mode examples – Example_280xHaltWake.c in lpm_haltwake, Example_280xIdleWake.c in lpm_idlewake, and Example_280xStandbyWake.c in lpm_standbywake.

k) **DSP280x_SysCtrl.c** – Added CsmUnlock() function which allows user to unlock the CSM in code, if desired.

l) **DSP280x_GlobalPrototypes.h** – Added extern function prototype for CsmUnlock().

m) **Eqep_pos_speed example** – Changed the example so that an external motor with a QEP encoder is no longer required to run the program. QEPA, QEPB, and the index signal are now simulated by EPWM1A, EPWM1B, and a GPIO pin respectively. Also added detailed comments about calculations performed in the example. Fixed and added appropriate watch variables as well.

n) **Eqep_freqcal example** – Fixed example so that EPWM1A is now appropriately configured to generate a QA signal for frequency measurement. Also added detailed comments about calculations performed in the example. Fixed and added appropriate watch variables as well.

o) **SFO library V5 (SFO_TI_Build_V5.lib + SFO_V5.h)** added to support up to maximum number of HRPWM channels. Note – V5 runs faster and takes less memory than the original version did, but when using SFO_MepEn_V5(n), Mep_En must be called repetitively until it is finished on the current channel before it is called on a different channel. Therefore, it now returns a "1" when it has finished running on a channel and a "0" otherwise. Due to this change, SFO_TI_Build.lib is still included in the event that it is necessary to run MepEn concurrently on up to 4 HRPWM channels. Also updated Readme in /lib folder.

p) **Example_280xHRPWM_SFO_V5** (hrpwm_sfo_v5) example added to demonstrate SFO library V5's optimizations and limitations.

q) **DSP280x_ECan Back2Back.c** updated to use DSP280x_ECan.c initialization function (which includes hooks for 60 MHz operation). Comments were also updated.

r) **Example_280xEcanA_to_B_Xmit.c** updated to use DSP280x_ECan.c initialization function (which includes hooks for 60 MHz operation). Comments were also updated.

Changes from V1.40 to V1.41

V1.41 is a minor update to fix stack allocation issues for some examples. No changes were made to the header files themselves.

Changes to examples:

a) Changed the stack size allocation from 0x400 to 0x380.

b) The root of the default path in all example project files was changed from C:\tidcs\c28\DSP280x\v140\ to C:\tidcs\c28\DSP280x\v141\.

Changes from V1.30 to V1.40

V1.40 is a minor update to incorporate the TMS320F2809, TMS320F28015 and F28016 devices and to make minor corrections. The following changes were made:

Changes to Header Files:

a) **DSP280x_Device.h**
Added #define for 2809 devices (DSP28_2809)
Added #define for 28015 devices (DSP28_28015)
Added #define for 28016 devices (DSP28_28016)

b) **DSP280x_Gpio.h**
Added GPIO35 to the pull-up disable (GPBPUD) and data (GPBDAT) registers. This is an internal pin that is not pinned out on the device. It is recommended that the pull-up for this internal pin be enabled to avoid current consumption in the HALT low power mode.

Changes to examples:

c) The root of the default path in all example project files was changed from C:\tidcs\c28\DSP280x\v130\ to C:\tidcs\c28\DSP280x\v140\ to reflect the version change.

d) Added a linker command files for the TMS320F2809, TMS320F28015 and TMS320F28016 devices.

e) All linker command files: added a reserved section for the boot ROM stack.

f) Added Code Composer gel files for all of the devices. These can be found in the DSP28_Common\gel directory.

g) **DSP280x_Examples.h**
Added predefined values for the PLL control register (PLLCR) and clock input divider (CLKINDIV). This allows these parameters to be easily configured for 60Mhz devices.

Updated the PLL init function prototype to include to inputs (PLLCR and CLKINDIV).

h) **DSP280x_SysCtrl.c**
Updated the InitPeripheralClocks(void) function to support the new devices. Updated the PLL initialize function to take two input values (PLLCR and CLKINDIV).

i) Flash: Added a version of this example for the TMS320F2809, TMS320F28015 and TMS320F28016 devices. Note: This release is prior to silicon availability of the 28015 and 28016. These examples will be tested only once silicon is available.

j) adc_seqmode_test
Removed 12 NOP's after the loop to poll the ADCST[INT_SEQ1] bit. These NOPs were required on 281x devices, but are not required for 280x or 2801x devices.

Changes from V1.20 to V1.30

V1.30 is a minor update to incorporate the TMS320F2802, TMS320C2802 and TMS320C2801 devices. The following changes were made:

Changes to Header Files:

f) **DSP280x_Device.h**
Added #define for 2802 devices (DSP28_2802)

Changes to examples:

s) The root of the default path in all example project files was changed from C:\tidcs\c28\DSP280x\v120\ to C:\tidcs\c28\DSP280x\v130\ to reflect the version change.

t) Added linker command files for the following devices: TMS320F2802, TMS320C2801 and TMS320C2802

u) Flash
Added a version of this example for the TMS320F2802 device.

v) **DSP280x_Examples.h**
Added #define for the PART_NO of a 2802 device (#define PARTNO_2802 0x24)

w) **DSP280x_ECan.c**
In some cases a shadow register was not being used. This has been corrected.

Changes from V1.10 to V1.20

Changes to Header Files:

x) **DSP280x_SysCtrl.h:**
Corrected the length of the ACTIVEWAIT and STDBYWAIT bit fields.
Added the CLKINDIV bit field to the PLLSTS register.

Also edited XCLKOUT (XCLK) register bit descriptions to read "reserved for TI internal use only" to align with System Control User Guide (with the exception of XCLKOUTDIV bits).

y) **DSP280x_DevEmu.h**:
Split the PARTID register into two bit fields: PARTTYPE and PARTNO.

Changes to examples:

a) The root of the default path in all example project files was changed from C:\tidcs\c28\DSP280x\v110\ to C:\tidcs\c28\DSP280x\v120\ to reflect the version change.

b) Added an example for the high resolution extension of the ePWM module (HRPWM)

c) Changed all instances of the term “HiRes” to HRPWM.

d) InitFlash() now uses the flash and OTP wait states required for SYSCLKOUT = 100MHz.

e) The watchdog reset function has been given the more humane name: ServiceDog(). For source-code compatibility, the old name can still be used as the following has been added to the DSP280x_GlobalPrototypes.h file: #define KickDog ServiceDog.

f) InitPll() now checks the state of the CLKINDIV bit before changing the PLLCR register.

g) InitPeripheralClocks() now only sets bits that are valid for the particular device.

h) Corrected a number of cut-and-paste errors in the example software prioritized default ISR functions (Example_280xSWPrioritizedDefaultIsr.c).

i) Added required EALLOW and EDIS instructions to the GPIO setup example.

j) Updated the comments at the end of the DSP280x_ECan.c file.

k) Removed references to XF. The XF functionality is not available on these devices.

l) Corrected the CMPA register access in the adc_soc example.

m) Corrected the size of the OTP block in the linker command files (F2808.cmd, F2806.cmd and F2801.cmd)

n) Added a linker command file for each device that defines all of the internal SARAM on the particular device regardless of whether it is CSM protected or not. (2808_RAM_lnk.cmd, 2806_RAM_lnk.cmd and 2801_RAM_lnk.cmd)

o) Updated all example .gel files to avoid conflicts with functions within the standard .gel files used by CCS 3.1.

Changes from V1.00 to V1.10

Changes to Header Files:

a) **DSP280x_EPwm.h:**
Added the following Hi-Resolution ePWM (HRPWM) registers:

Register Name	Address offset	Description
TBPHSHR	0x0002	HRPWM extension of phase TBPHS register
CMPAHR	0x0008	HRPWM extension of compare A CMPA register
HRCNFG	0x0020	HRPWM Configuration register

The header file definition of the CMPA and TBPHS registers have been changed to a union with the HRPWM extension registers to provide for .half (16-bit) and .all (32-bit) accesses. This was done to allow 16-bit access to CMPA and TBPHS as well as 32-bit access to the extended registers CMPA:CMPAHR and TBPHS:TBPHSHR.

Accessing the registers is done as follows:

```
EPwm1Regs.CMPA.half.CMPA = 0x1234;       // Access 16-bit CMPA register
EPwm1Regs.CMPA.half.CMPAHR = 0x5600;     // Access only HRPWM extension
EPwm1Regs.CMPA.all = 0x12345600;         // 32-bit write CMPA:CMPAHR

EPwm1Regs.TBPHS.half.TBPHS = 0x1234;     // Access 16-bit TBPHS register
EPwm1Regs.TBPHS.half.TBPHSHR = 0x5600;   // Access only HRPWM extension
EPwm1Regs.TBPHS.all = 0x12345600;        // 32-bit write TBPHS:TBPHSHR
```

Note, accesses to COMPB remain as is and do not require .all:

```
EPwm1Regs.CMPB = 0x5000;
```

This change requires users migrating from the DSP280x 1.00 header files to make modifications to their ePWM code. The changes required are as follows:

	DSP280x V1.00	DSP280x V1.10
Access CMPA	EPwm1Regs.CMPA=VALUE;	EPwm1Regs.CMPA**.half.CMPA**=VALUE;
Access TBPHS	EPwm1Regs.TBPHS=VALUE;	Epwm1Regs.TBPHS**.half.TBPHS**=VALUE;

Note:

The HRPWM extension is not available on all ePWM modules. The register file definition used, however, is identical for all ePWM modules. Thus, HRPWM register definitions will appear even if the ePWM module does not include the HRPWM extension.

b) **DSP280x_EPwm.h**
Made the following changes to the DBCTL register (Dead Band Control)

- ❑ Changed the MODE bit-field name to OUT_MODE
- ❑ Changed reserved bits 5:4 to the IN_MODE bit field

This corresponds to a silicon change made on Flash devices as of Rev A silicon.

c) **DSP280x_ECap.h**
The STOPVALUE bit-field in the ECCTL2 register was changed to STOP_WRAP. This corresponds to a silicon change made on Flash devices as of Rev A silicon. This register was previously used as a stop value for one-shot capture mode. It is now also used to specify a wrap value when using continuous capture mode.

d) **DSP280x_EQep.h**
Added UPEVNT (bit 7) to the QEPSTS register. This reflects changes made as of F280x Rev A devices.

e) **DSP280x_Spi.h**
Added definitions for SPI-B, SPI-C, SPI-D.

f) **DSP280x_Headers_nonBIOS.cmd and DSP280x_Headers_BIOS.cmd**
Updated the memory space allocated for ePWM1 – ePWM6 registers to include the HRPWM configuration register (HRCNFG).

g) **DSP280x_Peripheral.gel:**
The hotmenu item for `EPwm2Regs` was repeated twice. Removed the duplicate instance.

h) **DSP280x_EPwm_defines.h:**
Added useful #defines for the HRPWM.

Changes to examples:

p) The root of the default path in all example project files was changed from C:\tidcs\c28\DSP280x\v100\ to C:\tidcs\c28\DSP280x\v110\ to reflect the version change.

q) ecap_capture_pwm example:
Updated the function that initializes the eCAP peripheral.

r) Updated the CMPA and TBPHS register accesses in all ePWM and eQEP examples to use the .half of the union introduced for HRPWM register extension.

s) Added two ePWM with HRPWM extension example.

t) Updates to examples as required for changes described above to the header files.

V1.00

- This version was the first customer release of the DSP280x header files and examples.

10 Errata

This section lists known typos in the header files which have not been updated to prevent incompatibilities with code developed using earlier versions of the header files.

a) **DSP280x_I2C.h:**

Details— When the C-header files are included in an assembly project, the assembler views the AL (Arbitration Lost) bit in both the I2CIER and the I2CSTR structures as reserved words and issues an error.

Workaround— When including the C-header files in an assembly project, rename the AL bits to ARBL in DSP280x_I2C.h as follows to prevent conflicts with the assembler:

```
//----------------------------------------------------
// I2C interrupt mask register bit definitions */
struct I2CIER_BITS {           // bits   description
   Uint16 ARBL:1;              // 0      Arbitration lost interrupt
   Uint16 NACK:1;              // 1      No ack interrupt
   Uint16 ARDY:1;              // 2      Register access ready interrupt
   Uint16 RRDY:1;             // 3      Recieve data ready interrupt
   Uint16 XRDY:1;             // 4      Transmit data ready interrupt
   Uint16 SCD:1;              // 5      Stop condition detection
   Uint16 AAS:1;              // 6      Address as slave
   Uint16 rsvd:9;             // 15:7   reserved
};

//----------------------------------------------------
// I2C status register bit definitions */
struct I2CSTR_BITS {          // bits   description
   Uint16 ARBL:1;             // 0      Arbitration lost interrupt
   Uint16 NACK:1;             // 1      No ack interrupt
   Uint16 ARDY:1;             // 2      Register access ready interrupt
   Uint16 RRDY:1;             // 3      Recieve data ready interrupt
   Uint16 XRDY:1;             // 4      Transmit data ready interrupt
   Uint16 SCD:1;              // 5      Stop condition detection
   Uint16 rsvd1:2;            // 7:6    reserved
   Uint16 AD0:1;              // 8      Address Zero
   Uint16 AAS:1;              // 9      Address as slave
   Uint16 XSMT:1;             // 10     XMIT shift empty
   Uint16 RSFULL:1;           // 11     Recieve shift full
   Uint16 BB:1;               // 12     Bus busy
   Uint16 NACKSNT:1;          // 13     A no ack sent
   Uint16 SDIR:1;             // 14     Slave direction
   Uint16 rsvd2:1;            // 15     reserved
};
```