2803x C/C++ Header Files and Peripheral Examples Quick Start

1 Device Support:

This software package supports 2803x devices. This includes the following: TMS320F28035, TMS320F28034, TMS320F28033, TMS320F28032, TMS320F28031, and TMS320F28030.

Throughout this document, TMS320F28035, TMS320F28034, TMS320F28033, TMS320F28032, TMS320F28031, and TMS320F28030 are abbreviated as F28035, F28034, F28033, F28032, F28031, and F28030 respectively.

2 Introduction:

The 2803x C/C++ peripheral header files and example projects facilitate writing in C/C++ Code for the Texas Instruments TMS320x2803x devices. 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 '2803x development platform.

  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 2803x C/C++ peripheral header files.
- 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 280x header files to the 2803x header files.

† Code Composer Studio is a trademark of Texas Instruments (www.ti.com).

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 2803x 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

Version 1.21

- This version includes minor fixes to a couple of the files. A detailed revision history can be found in Section 9.

Version 1.20

- This version includes minor corrections and comment fixes to the header files and examples. A detailed revision history can be found in Section 9.

Version 1.10

- This version includes minor corrections and comment fixes to the header files and examples, and also adds a separate example folder, DSP2803x_examples_ccsv4, with examples supported by the Eclipse-based Code Composer Studio v4. A detailed revision history can be found in Section 9.

Version 1.01

- This version includes minor corrections to comments in the common files, and adds additional LIN and ADC temperature sensor examples. A detailed revision history can be found in Section 9.

Version 1.00

- This version is the first release of the 2803x header files and examples. It is an internal release used for customer trainings and tools releases.

2.2 Where Files are Located (Directory Structure)

As installed, the *2803x C/C++ Header Files and Peripheral Examples* is partitioned into a well-defined directory structure.

Table 1 describes the contents of the main directories used by DSP2803x header files and peripheral examples:

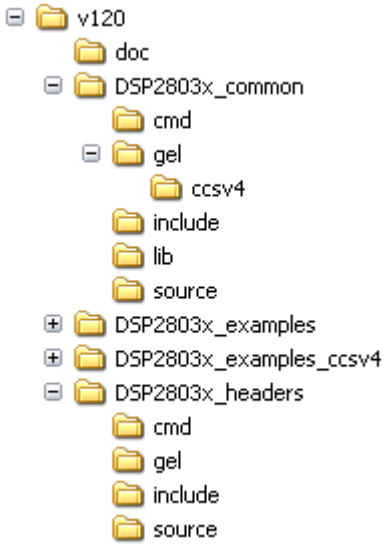


Table 1. DSP2803x Main Directory Structure

Directory	Description
<base>	Base install directory
<base>\doc	Documentation including the revision history from the previous release.
<base>\DSP2803x_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>\DSP2803x_examples	Example Code Composer Studio projects compiled with floating point unit *enabled*. These example projects illustrate how to configure many of the on-chip peripherals. An overview of the examples is given in Section 4.
<base>\DSP2803x_examples_ccsv4	Example Code Composer Studio v4 projects compiled with floating point unit *enabled*. These examples are identical to those in the \DSP2803x_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>DSP2803x_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 8.

Under the *DSP2803x_headers* and *DSP2803x_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. DSP2803x Sub-Directory Structure

Sub-Directory	Description
DSP2803x_headers\cmd	Linker command files that allocate the bit-field structures described in Section 3.
DSP2803x_headers\source	Source files required to incorporate the header files into a new or existing project.
DSP2803x_headers\include	Header files for each of the on-chip peripherals.
DSP2803x_common\cmd	Example memory command files that allocate memory on the devices.
DSP2803x_common\include	Common .h files that are used by the peripheral examples.
DSP2803x_common\source	Common .c files that are used by the peripheral examples.
DSP2803x_common\lib	Common library (.lib) files that are used by the peripheral examples.
DSP2803x_common\gel	Code Composer Studio v3.x GEL files for each device. These are optional.
DSP2803x_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 TMS320x28xx and 28xxx Peripherals in C/C++ **(**SPRAA85)

4 Peripheral Example Projects

This section describes how to get started with and configure the peripheral examples included in the 2803x Header Files and Peripheral Examples software package.

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 connected to a host with Code Composer Studio installed.**

   NOTE: As supplied, the '2803x example projects are built for the '28035 device. If you are using another 2803x 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 '2803x 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: *DSP2803x_examples\cpu_timer*

   c. Select *Example_2803xCpuTimer.gel* and click on *open*.

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

      DSP2803x CpuTimerExample-> Load_and_Build_Project

      This will load the project and build compile the project.

3. Edit DSP28_Device.h

Edit the DSP2803x_Device.h file and make sure the appropriate device is selected. By default the 28035 is selected.

```
/*-----------------------------------------------------------------------
* DSP2803x_headers\include\DSP2803x_Device.h
-----------------------------------------------------------------------*/

#define   TARGET   1

//---------------------------------------------------------------------------
// User To Select Target Device:

#define   DSP28_28030PAG   0
#define   DSP28_28030PN    0

#define   DSP28_28031PAG   0
#define   DSP28_28031PN    0

#define   DSP28_28032PAG   0
#define   DSP28_28032PN    0

#define   DSP28_28033PAG   0
#define   DSP28_28033PN    0

#define   DSP28_28034PAG   0
#define   DSP28_28034PN    0

#define   DSP28_28035PAG   0
#define   DSP28_28035PN    TARGET
```

4. Edit DSP2803x_Examples.h

Edit DSP2803x_Examples.h and specify the clock rate, the PLL control register value (PLLCR and DIVSEL). These values will be used by the examples to initialize the PLLCR register and DIVSEL bits.

The default values will result in a 60Mhz SYSCLKOUT frequency.

```
/*************************************************************
* DSP2803x_common\include\DSP2803x_Examples.h
*************************************************************/
/*-------------------------------------------------------------------------
      Specify the PLL control register (PLLCR) and divide select (DIVSEL) value.
-------------------------------------------------------------------------*/
//#define DSP28_DIVSEL   0   // Enable /4 for SYSCLKOUT(default at reset)
//#define DSP28_DIVSEL   1   // Disable /4 for SYSCKOUT
#define DSP28_DIVSEL     2   // Enable /2 for SYSCLKOUT
//#define DSP28_DIVSEL   3   // Enable /1 for SYSCLKOUT

  #define DSP28_PLLCR   12
//#define DSP28_PLLCR   11
//#define DSP28_PLLCR   10
//#define DSP28_PLLCR    9
//#define DSP28_PLLCR    8
//#define DSP28_PLLCR    7
//#define DSP28_PLLCR    6
//#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 DSP2803x_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 60 Mhz SYSCLKOUT.

```
/*************************************************************
* DSP2803x_common\include\DSP2803x_Examples.h
*************************************************************/
……
#define CPU_RATE     16.667L   // for a 60MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   20.000L   // for a 50MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   25.000L   // for a 40MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   33.333L   // for a 30MHz CPU clock speed  (SYSCLKOUT)
……
```

5. Review the comments at the top of the main source file: Example_2803xCpuTimer.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. Table 4 and Table 5 list the EMU boot modes (when emulator is connected) and the Get Mode boot mode options (mode is programmed into OTP) respectively. Refer to the documentation for your hardware platform for information on configuring the boot mode pins. For more information on the '2803x boot modes refer to the device specific *Boot ROM Reference Guide*.

Table 3. 2803x Boot Mode Settings

GPIO37 TDO	GPIO34 CMP2OUT	TRSTn	Mode
X	X	1	EMU Mode
0	0	0	Parallel I/O
0	1	0	SCI
1	0	0	Wait
1	1	0	"Get Mode"

Table 4. 2803x EMU Boot Modes (Emulator Connected)

EMU_KEY 0x0D00	EMU_BMODE 0x0D01	Boot Mode Selected
!= 0x55AA	x	Wait
0x55AA	0x0000	Parallel I/O
	0x0001	SCI
	0x0002	Wait
	0x0003	Get Mode
	0x0004	SPI
	0x0005	I2C
	0x0006	OTP
	0x0007	eCAN
	0x0008	Wait
	0x000A	Boot to RAM
	0x000B	Boot to FLASH
	Other	Wait

Table 5. 2803x GET Boot Modes (Emulator Disconnected)

OTP_KEY 0x3D7BFE	OTP_BMODE 0x3D7BFF	Boot Mode Selected
!= 0x55AA	x	Get Mode - Flash
0x55AA	0x0001	Get Mode - SCI
	0x0003	Get Mode – Flash
	0x0004	Get Mode - SPI
	0x0005	Get Mode - I2C
	0x0006	Get Mode - OTP
	0x0007	Get Mode - eCAN
	Other	Get Mode - Flash

When the emulator is connected for debugging:

TRSTn = 1, and therefore the device is in EMU boot mode. In this situation, the user must write the key value of 0x55AA to EMU_KEY at address 0x0D00 and the desired EMU boot mode value to EMU_BMODE at 0x0D01 via the debugger window according to Table 4. The 2803x gel files in the DSP2803x_common/gel/ directory have a GEL function – EMU Boot Mode Select -> EMU_BOOT_SARAM() which performs the debugger write to boot to "SARAM" mode when called.

When the emulator is not connected for debugging:

SCI or Parallel I/O boot mode can be selected directly via the GPIO pins, or OTP_KEY at address 0x3D7BFE and OTP_BMODE at address 0x3D7BFF can be programmed for the desired boot mode per Table 5.

7. Load the code

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

DSP2803x CpuTimerExample-> Load_Code (for '2803x devices)

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 *DSP2803x_headers* directory and shared source in the *DSP2803x_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 '2803x example projects are built for the '28035 device. If you are using another 2803x 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 '2803x 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: *DSP2803x_examples_ccsv4\cpu_timer*. Select the *Finish* button.

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

3. Edit DSP28_Device.h

Edit the DSP2803x_Device.h file and make sure the appropriate device is selected. By default the 28035 is selected.

```
/*-----------------------------------------------------------------------
* DSP2803x_headers\include\DSP2803x_Device.h
-----------------------------------------------------------------------*/

#define   TARGET   1

//---------------------------------------------------------------------------
// User To Select Target Device:

#define   DSP28_28030PAG   0
#define   DSP28_28030PN    0

#define   DSP28_28031PAG   0
#define   DSP28_28031PN    0

#define   DSP28_28032PAG   0
#define   DSP28_28032PN    0

#define   DSP28_28033PAG   0
#define   DSP28_28033PN    0

#define   DSP28_28034PAG   0
#define   DSP28_28034PN    0

#define   DSP28_28035PAG   0
#define   DSP28_28035PN    TARGET
```

4. Edit DSP2803x_Examples.h

Edit DSP2803x_Examples.h and specify the clock rate, the PLL control register value (PLLCR and DIVSEL). These values will be used by the examples to initialize the PLLCR register and DIVSEL bits.

The default values will result in a 60Mhz SYSCLKOUT frequency.

```
/*************************************************************
* DSP2803x_common\include\DSP2803x_Examples.h
*************************************************************/
/*-------------------------------------------------------------------------
      Specify the PLL control register (PLLCR) and divide select (DIVSEL) value.
-------------------------------------------------------------------------*/
//#define DSP28_DIVSEL   0   // Enable /4 for SYSCLKOUT(default at reset)
//#define DSP28_DIVSEL   1   // Disable /4 for SYSCKOUT
#define DSP28_DIVSEL     2   // Enable /2 for SYSCLKOUT
//#define DSP28_DIVSEL   3   // Enable /1 for SYSCLKOUT

  #define DSP28_PLLCR   12
//#define DSP28_PLLCR   11
//#define DSP28_PLLCR   10
//#define DSP28_PLLCR    9
//#define DSP28_PLLCR    8
//#define DSP28_PLLCR    7
//#define DSP28_PLLCR    6
//#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 DSP2803x_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 60 Mhz SYSCLKOUT.

```
/*************************************************************
* DSP2803x_common\include\DSP2803x_Examples.h
*************************************************************/
……
#define CPU_RATE     16.667L   // for a 60MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE   20.000L   // for a 50MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   25.000L   // for a 40MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   33.333L   // for a 30MHz CPU clock speed  (SYSCLKOUT)
……
```

5. Review the comments at the top of the main source file: Example_2803xCpuTimer.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. Table 4 and Table 5 list the EMU boot modes (when emulator is connected) and the Get Mode boot mode options (mode is programmed into OTP) respectively. Refer to the documentation for your hardware platform for information on configuring the boot mode pins. For more information on the '2803x boot modes refer to the device specific *Boot ROM Reference Guide*.

Table 6. 2803x Boot Mode Settings

GPIO37 TDO	GPIO34 CMP2OUT	TRSTn	Mode
X	X	1	EMU Mode
0	0	0	Parallel I/O
0	1	0	SCI
1	0	0	Wait
1	1	0	"Get Mode"

Table 7. 2803x EMU Boot Modes (Emulator Connected)

EMU_KEY 0x0D00	EMU_BMODE 0x0D01	Boot Mode Selected
!= 0x55AA	x	Wait
0x55AA	0x0000	Parallel I/O
	0x0001	SCI
	0x0002	Wait
	0x0003	Get Mode
	0x0004	SPI
	0x0005	I2C
	0x0006	OTP
	0x0007	eCAN
	0x0008	Wait
	0x000A	Boot to RAM
	0x000B	Boot to FLASH
	Other	Wait

Table 8. 2803x GET Boot Modes (Emulator Disconnected)

OTP_KEY 0x3D7BFE	OTP_BMODE 0x3D7BFF	Boot Mode Selected
!= 0x55AA	x	Get Mode - Flash
0x55AA	0x0001	Get Mode - SCI
	0x0003	Get Mode – Flash
	0x0004	Get Mode - SPI
	0x0005	Get Mode - I2C
	0x0006	Get Mode - OTP
	0x0007	Get Mode - eCAN
	Other	Get Mode - Flash

When the emulator is connected for debugging:

TRSTn = 1, and therefore the device is in EMU boot mode. In this situation, the user must write the key value of 0x55AA to EMU_KEY at address 0x0D00 and the desired EMU boot mode value to EMU_BMODE at 0x0D01 via the debugger window according to Table 4. The 2803x gel files in the DSP2803x_common/gel/ directory have a GEL function – EMU Boot Mode Select -> EMU_BOOT_SARAM() which performs the debugger write to boot to "SARAM" mode when called.

When the emulator is not connected for debugging:

SCI or Parallel I/O boot mode can be selected directly via the GPIO pins, or OTP_KEY at address 0x3D7BFE and OTP_BMODE at address 0x3D7BFF can be programmed for the desired boot mode per Table 5.

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.

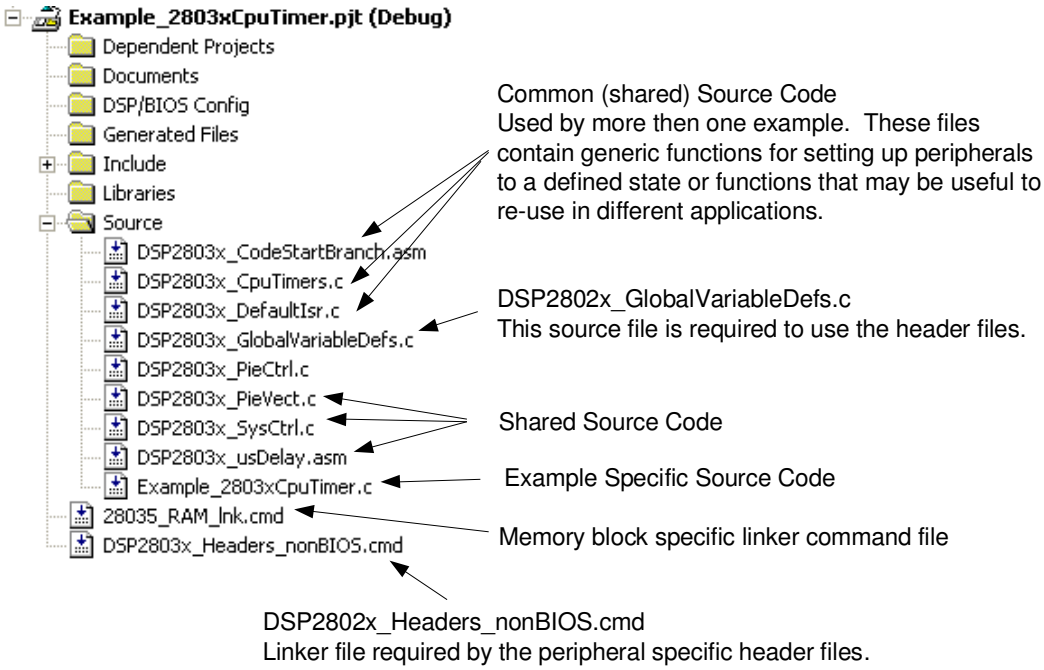
10. When done, delete the project from the Code Composer Studio v4 workspace.

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 *DSP2803x_headers* directory and shared source in the *DSP2803x_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



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.

```
/*****************************************************************
* DSP2803x_examples\cpu_timer\Example_2803xCpuTimer.c
*****************************************************************/

#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
```

- **DSP28x_Project.h**

  This header file includes DSP2803x_Device.h and DSP2803x_Examples.h. Because the name is device-generic, example/custom projects can be easily ported between different device header files. With this file included in the example source files, only the example/custom project (.pjt) file and DSP28x_Project.h file would need to be modified when porting source code between different devices. This file is found in the *<base>\DSP2803x_common\include* directory.

- **DSP2803x_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>\DSP2803x_headers\include* directory.